Mesh Generation

Mesh Generation

Application to Finite Elements

Second Edition

Pascal Jean Frey
Paul-Louis George

ISTE

WILEY

First edition published in 2000 by Hermes Science Ltd
Second edition published in Great Britain and the United States in 2008 by ISTE Ltd and John Wiley & Sons, Inc.

ISTE Ltd	John Wiley & Sons, Inc.
6 Fitzroy Square	111 River Street
London W1T 5DX	Hoboken, NJ 07030
UK	USA
www.iste.co.uk	www.wiley.com

Library of Congress Cataloging-in-Publication Data

Frey, Pascal Jean.
[Maillages. English]
Mesh generation application to finite elements / Pascal Jean Frey, Paul-Louis George. -- 2nd ed.
p. cm.
Includes index.
ISBN 978-1-84821-029-5
1. Finite element method. 2. Numerical grid generation (Numerical analysis) 3. Triangulation. I. George, Paul-Louis. II. Title.
TA347.F5F6913 2008
620.001'51825--dc22

2007046234

British Library Cataloguing-in-Publication Data
A CIP record for this book is available from the British Library
ISBN: 978-1-84821-029-5

Printed and bound in Great Britain by Antony Rowe Ltd, Chippenham, Wiltshire.

Contents

Introduction

Mesh generation techniques are widely employed in various engineering fields including those related to physical models described by partial differential equations (PDE). Numerical simulations of such models are intensively used for design, dimensioning and validation purposes. One of the most frequently used methods, among many others, is the *finite element* method (FEM). In this method, a continuous problem (the initial PDE model) is replaced by a discrete problem that can actually be computed thanks to the power of currently available computers. The solution to this discrete problem is an approximate solution to the initial problem whose accuracy is based on the various choices that were made in the numerical process.

The first step (in terms of actual computation) of such a simulation involves constructing a mesh of the computational domain (i.e., the domain where the physical phenomenon under interest occurs and evolves) so as to replace the continuous region by means of a finite union of (geometrically simple and bounded) elements such as triangles, quadrilaterals, tetrahedra, pentahedra, prisms, hexahedra, etc., based on the spatial dimension of the domain. For this reason, mesh construction is an essential pre-requisite for any numerical simulation of a PDE problem. Moreover, mesh construction could be seen as a bottleneck for a numerical process in the sense that a failure in this mesh construction step jeopardizes any subsequent numerical simulation.

★
★ ★

Mesh construction in general and more precisely for numerical simulation purposes involves several different fields and domains. These include (classical) geometry, so-called computational geometry and numerical simulation (engineering) topics coupled with advanced knowledge about what is globally termed computer science. The above classification in terms of disciplines which can interact in mesh construction for numerical simulation clearly shows why this topic is not so straightforward. Indeed, people with a geometrical, a computational geometry or a numerical background may not have the same perception of what a mesh (and, *a fortiori*, a computational mesh) should be, and subsequently do not share a common idea of what a mesh construction method could be.

★
★ ★

To give a rough idea of this problem, we mention, without in any way claiming to be exhaustive, some commonly accepted ideas about meshes based on the background of those considering the issue.

From a purely geometrical point of view, meshes are mostly of interest for the properties enjoyed by such or such geometrical item, a triangle for instance. In this respect, various issues have been investigated regarding the properties of such an element including aspect ratios, angle measures, orthogonality properties, affine properties and various related constructions (centroids, circumcenters, circumcircles, incircles, particular (characteristic) points, projections, intersections, etc.).

A computational geometry point of view mainly focuses on theoretical properties about triangulation methods including a precise analysis of the corresponding complexity. In this respect, Delaunay triangulation and its dual, the Voronoï diagram, have received much attention since nice theoretical foundations exist and lead to interesting theoretical results. However, triangulation methods are not necessarily suitable for general meshing purposes and must, to some extent, be adapted or modified.

Mesh construction from a purely numerical point of view (where, indeed, meshes are usually referred to as triangulations or grids) tends to reduce the mesh to a finite union of (simply shaped) elements whose size tends towards 0:
" Let $\mathcal{T}_h$ be a triangulation where h tends to 0, then ..., "
where $\mathcal{T}_h$ is provided in some way or other (with no further details given on this point). The construction of $\mathcal{T}_h$ is no longer a relevant problem if a theoretical study is envisaged (such as a convergence issue for a given numerical scheme).

In contrast to all the previous aspects, people actually involved in mesh construction methods face a different problem. Provided with some data, the problem is to develop methods capable of constructing a mesh (using a computer) that conforms to the needs of "numerical" and more generally "engineering" people. With regard to this, the above subscript h does not vanish, the domain geometry that must be handled could be of arbitrary complexity and a series of requirements may be demanded based on the subsequent use of the mesh once it has been constructed. On the one hand, theoretical results about triangulation algorithms (mainly obtained from computational geometry) may not be so realistic when viewed in terms of actual computer implementation. On the other hand, engineering requirements may differ slightly from what the theory states or needs to assume.

★
★ ★

As a brief conclusion, people involved in "meshing" must make use of knowledge from various disciplines, mainly geometry and computation, then combine this knowledge with numerical requirements (and computational limitations) to decide whether or not an *a priori* attractive aspect (for a particular discipline) is relevant in a meshing process. In other words, good candidates for mesh construction activities must have a sound knowledge in various disciplines in order to be able to select from these what they really require for a given goal.

Fortunately, we should point out that meshing things are becoming increasingly recognized as a subject of interest in its own right, not only in engineering but also at universities as well. In practice the subject is being addressed in several places all over the world, and a numerous people are spending a great deal of time on it. A few specialized conferences and workshops do exist and papers on meshing technologies can be found in various journals. Currently a few books[1] entirely (or substantially) devoted to meshing technologies are available.

Purpose and scope

The scope of this book is multiple and so are the potential categories of intended readers. As a first remark, we like to think that the theoretical background that is strictly necessary to understand the book is anything but specialized. We are confident that a reasonable knowledge of basic geometry, a touch of computational geometry and a good guess of what a numerical simulation is (for instance, some basic notions about the finite element method) provide a sufficient background for the reader to profit from this material. With regard to this, one of our objectives has been to make most of the presentations self-contained.

One issue underlying some of the discussions developed in the book was what material the reader might expect to find in such a book. A tentative answer to this point has led us to incorporate some material that could be judged trivial by readers who are already familiar with some meshing methods, yet we believe that its inclusion may well prove useful to less experienced readers.

We have introduced some recent developments in meshing activities, even if they have not necessarily been well validated (at least to the industrial standard), so as to allow advanced readers to initiate new progress based on this material.

It might be said that constructing a mesh for a given purpose (academic or

[1]Probably the very first significant reference about mesh generation is the book by Thompson, Warsi and Mastin, [Thompson *et al.* 1985], authored in 1985, which mainly discussed structured meshes. A few years after, in 1991, a book by George, [George-1991], was written which aimed to cover both structured and unstructured mesh construction methods. More recently, a book authored in 1993 by Knupp and Steinberg, [Knupp, Steinberg-1993] together with a book by Liseikin, [Liseikin-2000], provided an updated view of structured meshes. In 1998, a book fully devoted to Delaunay meshing techniques, [George, Borouchaki-1997], appeared. Among books that contain significant parts about meshing issues, one can find the book authored by Carey in 1997, [Carey-1997].

Thus, it is now possible to find some references about mesh technology topics. In this respect, one needs to see the publication of the Handbook of Grid Generation, edited by Thompson, Soni and Weatherill, [Thompson *et al.* 1999], which, in about 37 chapters by at least the same number of contributors, provides an impressive source of information. To conclude, notice the publication of another collective work, "Maillage et Adaptation", [George-2001], in the MIM (Mécanique et Ingénierie des Matériaux) series published by Hermès, Paris, together with a concise vulgarization book, "le maillage facile",[Frey, George-2003]. More recently, the Encyclopedia of Computational Mechanics, edited by Stein, de Borst and Hughes, [Stein *et al.* 2004], offered a chapter on mesh generation.

industrial) does not strictly require knowing what the meshing technologies are. Numerous engineers confronted daily with meshing problems, as well as graduate students facing the same problem, have been able to complete what they need without necessarily having a precise knowledge of what the software package they are familiar with actually does. Obviously, this point of view can be refuted and clearly a minimum knowledge of the available meshing technologies is a key to making this mesh construction task more efficient. Finally, following the above observations, the book is intended for both academic (educational) and industrial purposes.

Synopsis

Although we could have begun by a general purpose introduction and led on to a presentation of classical methods, followed by a discussion of advanced methods, specialized topics, etc., we chose to structure the book in such a way that it may be read sequentially. Relevant ideas are introduced when they are strictly necessary to the discussion, which means that the discussion about simple notions is made easy while when more advanced discussions are made, the more advanced ideas are given at the same time. Also, some almost identical discussions can be found in several sections, in an attempt to make each section as self-contained as possible.

★
★ ★

The book contains 24 chapters. The first three chapters introduce some general purpose definitions (Chapter 1) and basic data structures and algorithms (Chapter 2), then classical mesh generation methods are briefly listed prior to more advanced techniques (Chapter 3). The following chapters provide a description of the various mesh generation methods that are in common use. Each chapter corresponds to one type of method. We include discussions about algebraic, PDE-based or multi-block methods (Chapter 4), quadtree-octree based methods (Chapter 5), advancing-front technique (Chapter 6), Delaunay-type methods (Chapter 7), mesh generation methods for implicitly defined domains (Chapter 16) and other mesh generation techniques (Chapter 8) not covered by the previous cases. Chapter 9 deals with Delaunay-admissible curve or surface meshes and then discusses medial axis construction along with the various applications that can be envisaged based on this entity. Prior to a series of five chapters on lines, curves and surfaces, a short chapter concerns the metric aspects that are encountered in mesh generation activities (Chapter 10). As previously mentioned, Chapters 12 to 16 discuss curves and surfaces while Chapter 11 recalls the basic notions regarding differential geometry for curves and surfaces. One chapter presents various aspects about mesh modification tools (Chapter 17), then, two chapters focus on optimization issues (Chapter 18 for planar or volumic meshes and Chapter 19 for surface meshes). Basic notions about the finite element method are recalled in Chapter 20 before looking at a more advanced mesh generation problems, namely how to construct adapted, mobile or deformable meshes (Chapters 21, 22 and 23). Parallel aspects are discussed in Chapter 24. To conclude, an index is provided to the readers.

Acknowledgements

This work was carried out at the Gamma project, INRIA-Rocquencourt and was contributed to by many people in many ways. The authors would like to thank some of these people here. Friends and colleagues at INRIA, Frédéric Hecht, Patrick Laug, Éric Saltel and Maryse Desnous (who helped us for most of the figures that illustrate the discussion) were a great source of support and provided much of the technical material and illustrations for this work. Their comments and suggestions contributed greatly to its technical content.

The authors are especially indebted to Frédéric Noël (Université Joseph Fourier and Laboratoire des Sols Solides et Structures, Grenoble) who took the responsibility for the chapters devoted to curve and surface definitions and meshing. Bruce Simpson (Waterloo University, Ontario, Canada) must be thanked for his generous and significant contribution to this book in checking the consistency of the discussion. Mark Loriot (Distène, France) kindly accepted to help us with the part devoted to partitioning methods. Frédéric Eichelbrenner (ibid) helped us with surface meshing methods and Philippe Pébay (INSA, Lyon), through his PhD work, helped us for the chapter about Delaunay admissibility and medial axis construction. To finish, we would like to thank Frédéric Cazals (INRIA, Sophia Antipolis) for his help on the fundamentals of data structures and basic algorithms.

In addition to these people, several contributors were always willing to help us on various topics: Laurent Francez (Distène), Loïc Marechal (CNAM-INRIA), Éric Séveno (previously at INRIA), Rachid Ouatchaoui (EDF-INRIA) as well as Houman Borouchaki (UTT).

The first edition of this book is a translation of "Maillages. Applications aux éléments finis", published by Hermès Science Publications, Paris, 1999. The translation was carried out by the two authors with the valuable help of Richard James whom we would like to thank here.

This second edition includes various corrections, improvements, a fully updated index together with a new chapter about mobile and deformable meshes due to Pascal Frey, currently Full Professor at the Université Pierre et Marie Curie.

Finally, let us mention, *at this time*, two websites devoted to meshing technologies and a website offering thousands of surface meshes for downloading:

```
http://www-users.informatik.rwth-aachen.de/~roberts/meshgeneration.html
http://www.andrew.cmu.edu/user/sowen/mesh.html
http://www-c.inria.fr/gamma/download/
```

However, to find more information or to find interesting sites, simply use your favorite browser.

Symbols and Notations

Notations

d	refers to the spatial dimension
$\mathbb{N}, \mathbb{R}$	set of integers, set of reals
Ω	refers to a closed geometric domain of $\mathbb{R}^d$
$\partial\Omega$	refers to the (discretized) boundary of Ω
$\Gamma(\Omega)$	refers to the boundary of Ω
Γ, Σ	refers to a curve, a surface
γ, σ	refers to the parametrization of a curve, a surface
$\mathcal{T}, \mathcal{T}_h, \mathcal{T}_r$	refers to a triangulation or a mesh
$\mathcal{V}$ or $\mathcal{S}$	refers to a set of vertices
$Const$	refers to a constraint (a set of entities)
$Conv(\mathcal{V})$	refers to the convex hull of $\mathcal{V}$
(Δ, H)	refers to a control space
K	refers to a mesh element
S_K, V_K	refers to the surface area, the volume of element K
$\mathcal{Q}_K$	shape quality of mesh element K
d_{AB}, $d(A,B)$	(Euclidean) distance between A and B
$\lVert \overrightarrow{PQ} \rVert$	Euclidean length of segment PQ
l_{AB}	(normalized) length of edge AB

Symbols

∇	gradient operator
$\mathcal{H}$	Hessian tensor
$\lvert a \rvert$	absolute value
$\lfloor . \rfloor$	integer part or restriction
$\lVert . \rVert$	Euclidean length of a vector
$[a, b]$	a closed interval
$\langle u, v \rangle$	dot product of two vectors
$(. \wedge .)$	cross product of two vectors
${}^t u$	u transposed (also u^t)

Abbreviations

ALE	Arbitrary Lagrangian Eulerian
BRep, F-Rep	Boundary Representation, Function Representation
CAD	Computer Aided Design
CSG	Constructive Solid Geometry
MAT	Medial Axis Transform
FEM	Finite Element Method
PDE	Partial Derivative Equation
NURBS	Non Uniform Rational B-Splines
LIFO	Last In First Out
FIFO	First In First Out
BST	Binary Search Tree
AVL	Adelson, Velskii and Landis tree

Chapter 1

General Definitions

Before going further, it seems important to clarify the terminology and to provide some basic definitions together with some notions of general interest. First, we define the *covering-up* of a bounded domain, then we present the notion of a *triangulation* before introducing a particular triangulation, namely the well-known *Delaunay triangulation.*

A domain covering-up simply corresponds to the naive meaning of this word and the term may be taken at face value. On the other hand, a triangulation is a specific covering-up that has certain specific properties. Triangulation problems concern the construction, of a covering-up of the convex hull of a given set of points. In general, a triangulation is a set of simplices, triangles in two dimensions, tetrahedra in three dimensions, with certain properties. If, in addition to a set of vertices, the boundary of a domain (more precisely a discretization of this boundary whose vertices are in the above set) is specified or, simply if any set of required edges (faces) is provided, we encounter a problem of *constrained triangulation.* In this case, the expected triangulation of the convex hull must contain these required items.

In contrast, the notion of a *mesh* may now be specified. Given a domain, namely defined by a discretization of its boundary, the problem comes down to constructing a "triangulation" that accurately matches this specific domain. In a way, we are dealing with a constrained triangulation but, now, we no longer face a convex hull problem and, moreover, the mesh elements are not necessarily simplices.

After having established triangulation and mesh definitions, some other aspects are discussed, including a suitable element definition (as an element is the basic component of both a triangulation and a mesh), finite element definition as well as mesh data structure definition which are the fundamental ingredients of any further processing (such as using a finite element method). In addition, we introduce some definitions related to certain data structures which are widely used in mesh construction and mesh optimization processes. To conclude, we propose measures of mesh quality and of mesh optimality.

Obviously this chapter cannot claim to be exhaustive. In fact, more specific ideas will be introduced and discussed as required throughout the book.

1.1 Covering-up and triangulation

If $\mathcal{S}$ is a finite set of points in $\mathbb{R}^d$ ($d = 2$ or $d = 3$), the convex hull of $\mathcal{S}$, denoted as $Conv(\mathcal{S})$, defines a domain Ω in $\mathbb{R}^d$. Let K be a simplex[1] (triangle or tetrahedron according to d, always considered as a connected and closed set). Then a covering-up $\mathcal{T}_r$ of Ω by means of simplices corresponds to the following definition:

Definition 1.1 *$\mathcal{T}_r$ is a simplicial covering-up of Ω if the following conditions hold*

- $(H0)$ *The set of element vertices in $\mathcal{T}_r$ is exactly S.*
- $(H1)$ $\Omega = \overline{\bigcup_{K \in \mathcal{T}_r} K}^{\circ}$, *where K is a simplex.*
- $(H2)$ *The interior of every element K in $\mathcal{T}_r$ is non empty.*
- $(H3)$ *The intersection of the interior of two elements is an empty set.*

Here is a "natural" definition. With respect to condition $(H1)$ (where while not strictly necessary, we restrict ourselves to simplicial elements), one can see that Ω is the open set corresponding to the domain that means, in particular, that $\overline{\Omega} = \bigcup_{K \in \mathcal{T}_r} K$. Condition $(H2)$ is not strictly necessary to define a covering-up, but it is nevertheless practical with respect to the context and, thus, will be assumed. Condition $(H3)$ means that element overlapping is proscribed.

Similarly, we will consider conforming coverings-up, referred to as triangulations.

Definition 1.2 *$\mathcal{T}_r$ is a conforming triangulation or simply a triangulation of Ω if $\mathcal{T}_r$ is a covering-up following Definition (1.1) and if, in addition, the following condition holds:*

- $(H4)$ *the intersection of two elements in $\mathcal{T}_r$ is either reduced to*
  - *the empty set or to*

[1]Let us briefly recall the definition of a d-simplex: we consider $d+1$ points $a_j = (a_{ij})_{i=1}^{d} \in \mathbb{R}^d, 1 \leq j \leq d+1$, not all in the same hyper-plane, i.e., such that the matrix of order $d+1$:

$$\mathcal{A} = \begin{pmatrix} a_{11} & \cdots & a_{1,d+1} \\ \cdots & \cdots & \cdots \\ a_{d1} & \cdots & a_{d,d+1} \\ 1 & 1 & 1 \end{pmatrix},$$

is invertible. D-simplex K whose vertices are the a_j is the convex hull of these points a_j. Every point x in $\mathbb{R}^d$, with Cartesian coordinates x_i is fully specified by the data of $d+1$ scalar values $\lambda_j = \lambda_j(x)$ that are solutions of the linear system:

$$\left\{ \sum_{j=1}^{d+1} a_{ij}\lambda_j = x_i \text{ with } \sum_{j=1}^{d+1} \lambda_j = 1 \right.,$$

whose matrix is $\mathcal{A}$. The $\lambda_j(x)$ are the *barycentric coordinates* of point x with respect to the points a_j.

– *a vertex, an edge or a face (for $d = 3$).*

More generally, in d dimensions, such an intersection must be a k-face[2], for $k = -1, ..., d-1$, d being the spatial dimension.

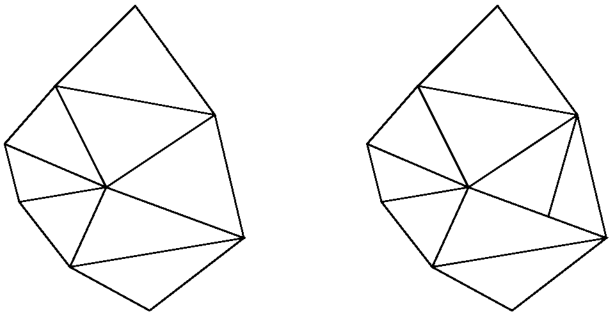


Figure 1.1: *Conformal triangles (left-hand side) and non-conformal triangles (right-hand side). Note the vertex located on one edge in this case.*

Remark 1.1 *For the moment, we are not concerned with the existence and possibly uniqueness of such a triangulation for a given set of points. Nevertheless, a theorem of existence will be provided below and, based on some specific assumptions, the particular case of a Delaunay triangulation will be described.*

Euler characteristics. The Euler formula, and its extensions, the Dehn-Sommerville relationships, relate the number of k-faces ($k = 0, ..., d-1$) in a triangulation of Ω. Such formula can be used to check the topological validity of a given mesh or also for other purposes, such as the determination of the genus of a surface.

Definition 1.3 *The Euler characteristics of a triangulation $\mathcal{T}_r$, is the alterned summation:*

$$\chi = \sum_{k=0}^{d} (-1)^k n_k \,, \tag{1.1}$$

where $n_k, k = 0, .., d$ denotes the number of the k-faces in the triangulation.

When the triangulation is homotopic to the topological ball, its characteristic is 1. If the triangulation is homeomorphic to the topological sphere, its Euler characteristic is $1 + (-1)^d$. In two dimensions, the following relation holds:

$$nv - ne + nt = 2 - c \,,$$

where nv, ne and nt are respectively the number of vertices, edges and triangles in the triangulation, c corresponds to the number of connected components of

[2] A (-1)-face is the empty set, a 0-face is a vertex, a 1-face is an edge, a k-face is in fact a k-simplex with $k < d$, d being the spatial dimension.

the boundary of Ω. More precisely, if the triangulation includes no hole, then $nv - ne + nt = 1$. In three dimensions, the above formula becomes:

$$nv - ne + nf - nt = 2 - 2g\,,$$

where nf is the number of faces, nt the number of tets and g stands for the *genus* of the surface (i.e., the number of holes) of the triangulation. Thus, a triangulation of a closed surface is such that $nv - ne + nf = 2$.

Delaunay triangulation. Among the different possible types of triangulations, the Delaunay triangulation is of great interest. Let us recall that S is a set (a cloud) of points (sites) and that Ω is $Conv(S)$, the convex hull of S .

Definition 1.4 *$\mathcal{T}_r$ is the* Delaunay triangulation *of Ω if the open discs (balls) circumscribed to any of its elements does not contain any vertex of S.*

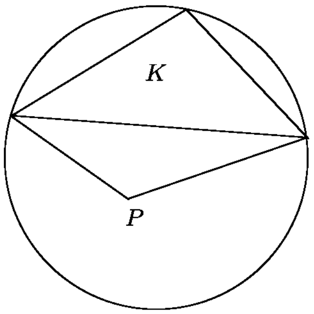


Figure 1.2: *The empty sphere criterion is violated, the disc of K encloses the point P. Similarly, the circumdisc of the triangle with vertex P includes the vertex of triangle K opposite the common edge (the criterion is symmetric for any pair of adjacent elements).*

This criterion, the so-called *empty sphere criterion* or *Delaunay criterion*, means that all open balls associated with all elements do not contain any vertex, a closed ball containing the vertices of the element under consideration only. This is the main characterization of the Delaunay triangulation. The Delaunay criterion leads to several other characteristics of any Delaunay triangulation. Figure 1.2 shows an example of an element K which does not meet the Delaunay criterion.

A basic theoretical issue follows.

Theorem 1.1 *There exists a unique Delaunay triangulation of a set of points.*

The proof is evident by involving the duality with the Voronoï diagram associated with the set of points (cf. Chapter 7). The existence is then immediate and the uniqueness is achieved as the points are assumed in general position[3] if one wishes to have a simplicial triangulation. Otherwise, the following remark holds.

[3] A set of points is said to be in general position if there is no configuration of more than three points that are co-circular (more than four co-spherical points) such that the corresponding open disk (ball) is empty.

Remark 1.2 *In the case of more than three co-circular (resp. four co-spherical) points, a circle (resp. sphere) exists enclosing these points. If the related disk (resp. ball) is empty, the Delaunay triangulation exists but contains non-simplicial elements such as polygons (resp. polyhedra).*

Hence, the uniqueness holds if non-simplicial elements are allowed while if the latter are subdivided by means of simplices, several solutions can be found. Nevertheless, while it may be excessive, we will continue to speak of the *Delaunay triangulation by observing that all any partitions of a non-simplicial element are equivalent after swapping*[4] *a k-face.*

A brief digression. The notion of a Voronoï diagram (though it had yet to be called as such!) first appeared in the work of the French philosopher R. Descartes (1596-1650) who introduced this notion in 1644 in his *Principia Philosophiae*, which aimed to give a mathematical description of the arrangement of matter in the solar system. In 1850, G. Dirichlet (1805-1859) studied this idea in two and three dimensions and this diagram came to be called the *Dirichlet tessellation* [Dirichlet-1850]. However, its definitive name came after M.G. Voronoï (1868-1908), who generalized these results in d dimensions [Voronoï-1908].

Nature provides numerous examples of arrangements and quasi-regular paving which bear a strange resemblance to Voronoï diagrams. Figure 1.3 illustrates some of these typical arrangements[5].

Constrained triangulation. Provided a set of points and, in addition, a set of edges (resp. edges and faces in three dimensions), an important problem is to ensure the existence of these edges (resp. these edges and faces) in a triangulation. In the following, *Const* denotes a set of such entities.

Definition 1.5 *$\mathcal{T}_r$ is a constrained triangulation of Ω for Const if all and any element of Const is an entity of $\mathcal{T}_r$.*

In particular, a constrained triangulation[6] can satisfy the Delaunay criterion locally, except in some neighborhood of the constraints.

Remark 1.3 *As above, provided a set of points and a constraint, we are not concerned here with the existence of a solution triangulation.*

[4]A 2-face swap (flip) consists of replacing the diagonal of the convex quadrilateral made up of two adjacent triangles by the alternate configuration, see Chapter 18 for the precise definition.

[5]Given a set of geometric objects, an arrangement is a covering-up of the space by means of the regions (cells) formed by the given objects and their (potential) intersections.

[6]Whereas a *constrained Delaunay triangulation* in two dimensions is a triangulation which satisfies the empty sphere criterion, where a open ball can contain a vertex in the case where the latter is not seen, due to a constrained edge, by all the vertices of the considered element. In other words, a constrained entity exists which separates the above vertices and the others.

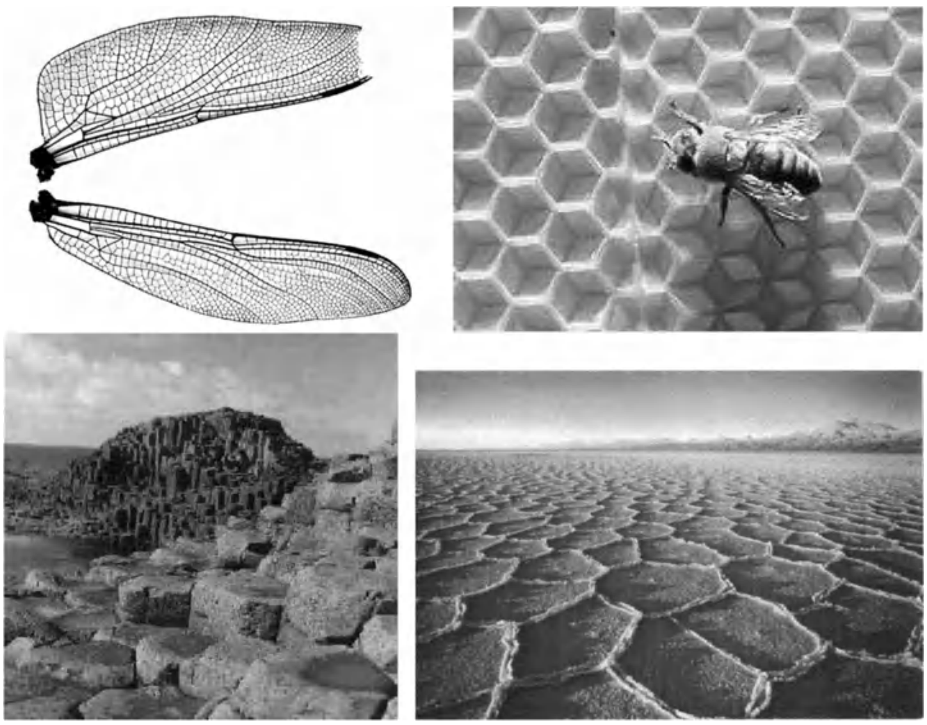


Figure 1.3: *Top, the wings of a dragonfly (doc. A. LeBéon) show an alveolar structure apparently close to a Voronoï diagram (left-hand side) and one of the more representative examples of regular paving (consisting of hexagonal cells) is that of a bee's nest (right-hand side). Bottom, two examples of natural arrangements. Left-hand side: the basaltic rock site of the Giant's Causeway, Co Antrim, Northern Ireland (photo credit: John Hinde Ltd.). Right-hand side, desert region of Atacama (Chile), the drying earth forms patterns close to Voronoï cells.*

1.2 Mesh, mesh element, finite element mesh

Now we turn to a different problem. Let Ω be a closed bounded domain in $\mathbb{R}^2$ or $\mathbb{R}^3$. The question is how to construct a conforming triangulation of this domain. Such a triangulation will be referred to as a *mesh* of Ω and will be denoted by $\mathcal{T}_r$ or $\mathcal{T}_h$ for reasons that will be made clear in the following. Thus,

Definition 1.6 *$\mathcal{T}_h$ is a mesh of Ω if*

- $(H1)$ $\Omega = \overline{\overset{\circ}{\bigcup_{K \in \mathcal{T}_h}} K}\,.$
- $(H2)$ *The interior of every element K in $\mathcal{T}_h$ is non-empty.*
- $(H3)$ *The intersection of the interior of two elements is empty.*

Condition ($H2$) is clearly not verified for a beam element for instance. Condition ($H3$) avoids element overlapping. In contrast to the definition of a triangulation, Condition ($H0$) is no longer assumed, which means that the vertices are not, in general, given *a priori* (see hereafter) and, in ($H1$), the K's are not necessarily simplices.

Most computational schemes using a mesh as a spatial support assume that this mesh is conforming (although, this property is not strictly necessary for some solution methods).

Definition 1.7 *$\mathcal{T}_h$ is a conformal mesh of Ω if Definition (1.6) holds and*

- ($H4$) *the intersection of two elements in $\mathcal{T}_h$ is either the empty set, a vertex, an edge or a face ($d = 3$).*

Clearly, the set of definitions related to a triangulation is again met. There is a fundamental difference between a triangulation and a mesh. A triangulation is a covering-up of the convex hull of a given set of points which, in general, is composed of simplicial elements. A mesh is a covering-up of a given domain defined, in most of the applications, via a given discretization of its boundary, this covering-up being composed of possibly non simplicial elements. On the other hand, at least two new problems occur, namely:

- the respect or *enforcement*, in some sense, of the boundary of the domain so that the triangulation is a constrained triangulation,
- the necessity of *constructing* the set of points which will define the vertices of the mesh. Usually the boundary points of the given boundary discretization are given as sole input and field points must be explicitly created.

Remark 1.4 *For a boundary discretization defining a domain, the existence of a mesh conforming to this discretization holds in two dimensions but is still, at least from a computer point of view, a delicate question in three dimensions.*

Remark 1.5 *In the finite element method, the meshes*[7] *are generally denoted by $\mathcal{T}_h$, where the index h of the notation refers to the diameters of the elements in the mesh, these quantities being used in error bound theorems.*

As previously mentioned, a mesh can be composed of elements of different geometric natures. A mesh consists of a finite number of segments in one dimension, segments, triangles and quadrilaterals (quads for short) in two dimensions and the above elements, tetrahedra (tets), pentahedra and hexahedra (hexes) in three dimensions. The mesh elements must generally satisfy some specific properties depending on the application involved.

Meshes can be classified into three main classes according to their *connectivity*.

[7]It should be noted that people with a finite element background use the term triangulation and use the term mesh synonymously.

Definition 1.8 *The connectivity of a mesh is the definition of the connection between its vertices.*

Then, following this definition

Definition 1.9 *A mesh is called structured (resp. unstructured) if its connectivity is of the finite difference type (resp. any other type).*

A structured mesh can be termed as a *grid*[8]. In two dimensions, a grid element is a quadrilateral while, in three dimensions, a grid consists of hexahedra. The connectivity between nodes is of the type (i, j, k), i.e., assuming the indices of a given node, the node with indices (i, j, k) has the node with indices $((i-1), j, k)$ as its "left" neighbor and that with indices $((i+1), j, k)$ as its "right" neighbor; this kind of mesh is convenient for geometries for which such properties are suitable, i.e., for generalized quadrilateral or hexahedral configurations.

Remark 1.6 *Peculiar meshes other than quad or hex meshes could have a structured connectivity. For instance, one can consider a classical grid of quads where each of them are subdivided into two triangles using the same subdivision pattern.*

Such a mesh is usually composed of triangles (tetrahedra) but can also be a set of quadrilaterals (hexahedra) or, more generally, a combination of elements of a different geometric nature. Note that quad or hex unstructured meshes are such that the internal vertices may be shared by more than 4 (8) elements (unlike the case of structured meshes).

For completeness, we introduce two more definitions.

Definition 1.10 *A mesh is said to be* mixed *if it includes some elements of a different geometric nature.*

Definition 1.11 *A mesh is said to be* hybrid *if it includes some elements with a different spatial dimension.*

A mixed mesh, in two dimensions, is composed of triangles and quads. A hybrid mesh, again in two dimensions, is clearly a mixed mesh but, for instance, includes some triangles together with some segments.

To complete this classification, a mesh may be *manifold* or not. This point concerns only surface meshes.

Definition 1.12 *A (conformal) surface mesh is called manifold if its internal edges are shared by exactly two elements or only one element in the case of a boundary edge for an open surface.*

Otherwise, the surface mesh is said to be *non-manifold*. This is the case of surface meshes which include stiffeners or which have two or more connected components.

[8]Note that some authors use the term "grid" to refer to any kind of mesh whatever its connectivity.

Mesh element

The elements are the basic components of a mesh. An element is defined by its geometric nature (triangle, quadrilateral, etc.) and a list of vertices. This list, enriched with some conventions (see hereafter), allows the complete definition of an element, including the definition of its edges and faces (in three dimensions).

Definition 1.13 *The connectivity of a mesh element is the definition of the connections between the vertices at the element level.*

This connectivity, the local equivalent of the mesh connectivity, makes the description of the *topology* of the element possible.

Definition 1.14 *The topology of a mesh element is a definition of the relationships between its faces, edges and vertices.*

Triangle connectivity and topology. For convenient purposes, the (local) numbering of vertices and edges is pre-defined in such a way that some properties are implicitly induced[9]. This definition is only a convention leading to implicit properties. In particular, a ordered numbering of the vertices enables us to compute the surface area of a triangle with a positive, or directional, sense. It also allows us to evaluate directional normals for each edge.

In the case of a triangle with connectivity $[1,2,3]$, the first vertex (1) having been chosen, the numbering of the others is deduced counterclockwise. Then the topology can be well defined by means of the edge definition:

- edge [1] runs from vertex (1) to vertex (2),
- edge [2] : $(2) \rightarrow (3)$,
- edge [3] : $(3) \rightarrow (1)$,

or alternatively,

- edge [1] is opposite vertex (1), it runs from vertex (2) to vertex (3),
- edge [2] : $(3) \rightarrow (1)$,
- edge [3] : $(1) \rightarrow (2)$.

Once a topology has been chosen, all mesh elements must conform to this rule. Such an implicit definition will be a source of simplicity hereafter, avoiding explicit definitions at the element level during the computational step, as mentioned earlier.

Usual element connectivities and topologies. Elements other than triangles are now defined in terms of the two above definitions.

- The segment: $[1,2]$, $(1) \rightarrow (2)$.
- The quadrilateral: $[1,2,3,4]$ with a numbering as for the triangle,

edge [1] : $(1) \rightarrow (2)$ edge [2] : $(2) \rightarrow (3)$
edge [3] : $(3) \rightarrow (4)$ edge [4] : $(4) \rightarrow (1)$

[9]Given a vertex numbering (index) based on an implicit definition results in implicit definitions for both the edges and the faces, thus avoiding an explicit definition of these entities at the element level, which would be not unique and memory consuming.

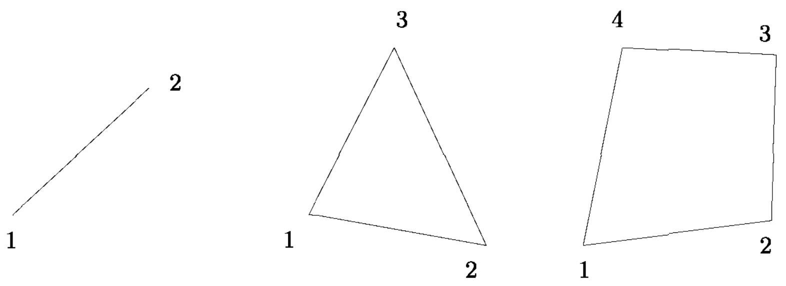


Figure 1.4: *Local vertex numbering of segment, triangle and quadrilateral, given the first vertex index.*

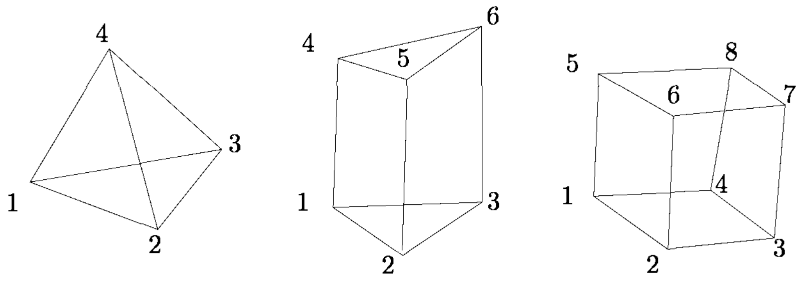


Figure 1.5: *Tetrahedron, pentahedron and hexahedron.*

- The tetrahedron[10]: $[1,2,3,4]$ with $(\vec{12},\vec{13},\vec{14})$ assumed to be positive with, for the edges:

edge [1] : (1) $\rightarrow$ (2)	edge [2] : (2) $\rightarrow$ (3)	edge [3] : (3) $\rightarrow$ (1)
edge [4] : (1) $\rightarrow$ (4)	edge [5] : (2) $\rightarrow$ (4)	edge [6] : (3) $\rightarrow$ (4)

and, for the faces:

face [1] : (1) (3) (2)	face [2] : (1) (4) (3)
face [3] : (1) (2) (4)	face [4] : (2) (3) (4)

- The pentahedron: $[1,2,3,4,5,6]$ with $(\vec{12},\vec{13},\vec{14})$ assumed to be positive, with, for the edges:

edge [1] : (1) $\rightarrow$ (2)	edge [2] : (2) $\rightarrow$ (3)	edge [3] : (3) $\rightarrow$ (1)
edge [4] : (1) $\rightarrow$ (4)	edge [5] : (2) $\rightarrow$ (5)	edge [6] : (3) $\rightarrow$ (6)
edge [7] : (4) $\rightarrow$ (5)	edge [8] : (5) $\rightarrow$ (6)	edge [9] : (6) $\rightarrow$ (4)

and, for the faces:

face [1] : (1) (3) (2)	face [2] : (1) (4) (6) (3)
face [3] : (1) (2) (5) (4)	face [4] : (4) (5) (6)
face [5] : (2) (3) (5) (6)	

[10]Similarly to the triangle, an alternative definition also suits well where face [i] is opposite vertex (i). Actually, the latter convention leads to greater simplicity.

- The hexahedron: $[1,2,3,4,5,6,7,8]$, $(\vec{12},\vec{14},\vec{15})$ assumed to be positive, with

edge [1] : (1) → (2)	edge [2] : (2) → (3)	edge [3] : (3) → (4)
edge [4] : (4) → (1)	edge [5] : (1) → (5)	edge [6] : (2) → (6)
edge [7] : (3) → (7)	edge [8] : (4) → (8)	edge [9] : (5) → (6)
edge [10] : (6) → (7)	edge [11] : (7) → (8)	edge [12] : (8) → (5)

and, for the faces:

face [1] : (1) (4) (3) (2)	face [2] : (1) (5) (8) (4)
face [3] : (1) (2) (6) (5)	face [4] : (5) (6) (7) (8)
face [5] : (2) (3) (7) (6)	face [6] : (3) (4) (8) (7)

Other types such as pyramid may be defined. Actually, this type of element allows for some flexibility in mixed meshes. This is the case when structured hex meshes must be combined with unstructured tet meshes.

Finite element mesh

So far, we have considered meshes as geometric entities. Now we turn to the notion of *finite element meshes* since we are mainly interested in finite element computation as the mesh has a great deal of importance for this application. As will be discussed in Chapter 20, finite elements will be constructed based on the mesh element. To this end, it will be necessary to properly define the nodes, degrees of freedom, interpolation schemes, etc. so as to define the required structures (stiffness matrix, right-hand side, etc.) in order to compute the solution to the problem at hand.

Let us briefly recall for those not so familiar with a finite element style computation, that the classical scheme of such calculus includes the following steps (for the simple case of a linear system to solve):

- a definition of the computational domain,
- a mesh construction step whose purpose is to complete the list of the (geometric) elements of this mesh,
- an interpolation step which constructs the finite elements from the mesh elements,
- a matrix and right-hand side construction step to complete the system corresponding to the discretization of the initial equations, based on the element connectivity,
- a solution step which computes the solution of the above system.

This being clarified, we can proceed by giving some definitions related to the finite element meshes.

Node definition. The finite elements will be associated with the mesh elements at the computational step. For the moment, a finite element is a geometric element supplied with a list of *nodes.*

Definition 1.15 *A node is a point supporting one or several unknowns or degrees of freedom (dof).*

The nodes are defined according to the interpolation used in the computation. For a given geometric element, several finite elements may be exhibited as a function of the interpolation step. The "simplest" finite element is the Lagrange P^1 finite element whose nodes are the element vertices. A Lagrange P^2 finite element includes as nodes the element's vertices and a point on each of its edges (in the usual case, these nodes are the edge midpoints). Other finite elements may involve several nodes for each edge, nodes located on faces or inside the element while the element vertices may be nodes or not (see Chapter 20).

Once the node location has been established, it is a case of defining a local numbering for the nodes of the finite element. This task is trivial when the only element vertices are the nodes as the node numbering follows the vertex numbering. If the nodes are defined elsewhere, the local numbering must be well defined. It can be either implicitly defined as for the vertex or explicitly defined in some cases where an implicit definition is not possible, for instance, when the number of nodes varies from one edge to the other as it does for some finite elements. If we consider a Lagrangian P^2 triangle, it is common to define the three first nodes as the three vertices and then to define as fourth node the node located along the first edge of the triangle (and so on for the other nodes). See Chapter 20 and Chapter 17 for node (re)numbering issues.

Physical attributes. At the solution step, the finite elements are the support of various computations or specific treatments. The mesh, through its elements must contain information making it possible the selection of a set of elements, a set of faces, set of edges or a set of nodes, in other words, making possible any processing concerning these entities, in particular to take into account the loads, flows, pressures, boundary conditions, graphic requirements at the time the solutions must be displayed, etc., related to the problem considered. This will allow to carry out the adequate assignment (i.e., associate the relevant value with such or such an item) or the proper computation of the useful integrals over particular entities.

It is then convenient to associate a *physical attribute* with all the mesh entities (elements, faces, edges, nodes). This task can be carried on in many manners and, at the computer level, can be implemented in various different ways.

Geometric attributes. For similar reasons, a geometric attribute (provided in some way) proves to be useful for some operations such as the proper definition of the nodes in the case of a finite element other than P^1 or the definition of a subdomain. Thus, it is also convenient to associate a geometric attribute with all the element entities (faces, edges, nodes).

1.3 Mesh data structures

A data structure is simply, but not only, a way of organizing a set of given information (values). A mesh data structure is a way to store all the values that will be useful for a further processing. Following this idea, two categories of structures can be distinguished. One is typically used when constructing the mesh. In other words, such a structure, referred to as the *internal mesh data structure*, is the organization of the values describing the mesh which are required inside the mesh generation method chosen for this task. On the other hand, the structure referred to as the *output mesh data structure* is the mesh organization used at the computational step, i.e., outside the mesher. Thus, the values stored in it as well as the way in which they are organized can slightly differ from those in the previous structure. The possible differences between the two mesh structures depend both on the type of the mesh generation method and the solution method that are used. However, for the output structure, it could be desirable to have a "universal" structure or, at least, a (presumably) generic structure, more suitable for data exchanges.

The internal mesh data structure

The key idea is to define as simple a structure as possible which is well suited to the problem. In some sense, it is the minimal amount of information needed for the application. Various reasons can justify this policy. Among them, a given mesh generation method, due to the algorithm used and the nature of the meshes it is capable of processing, may require a given data organization that is different from another method. Hence, a "universal" structure is not, *a priori*, a good solution, as it would be unnecessarily complex in certain cases. Thus, for a given method, one has to find what is needed specifically and what is not strictly required. In this respect, efficiency as well as memory occupation reasons can be invoqued to justify such or such a choice.

The internal mesh structure is only used within the algorithm and, when the mesh is created, the information stored within this structure is transformed to complete the output data structure which is the natural and unique link with the other computational steps.

Some values and data items are specific to one structure or the other. Some others must be included in both. The point is to make this requirement precise and to define the structure accordingly. In the next chapter, we will give some indication about what such an internal structure could be.

The output mesh data structure

Defining a suitable, general purpose and reasonably simple output data structure for mesh storage is not a trivial task. A number of issues must be addressed in order to fulfill various requirements. These include:

- the definition of the nodes, when defining the finite elements at the so-called interpolation step,

- the definition of boundary conditions and loads, the computation and the assembly of the relevant matrices and right-hand sides,
- the solution of the resulting system(s),
- the visualization of the results,
- and many others types of processing related to the nature of the problem to solve.

In this respect, it is clear that a classic computational scheme involving a mesh generation step, an interpolation step, the computation of the system to be solved and the solution step is less demanding than an adaptive scheme requiring a loop of such operations where, for instance, the mesh (and the geometry itself, in some cases) is modified at each iteration step of the loop. Thus, a mesh data structure must be defined in such a way as to provide easy access to:

- the vertex coordinates,
- the element vertices,
- the physical attributes of the mesh entities,
- the geometrical attributes of these entities,

in order to make the previous computational requirements possible.

These basic principles being stated, it is beyond the scope of this book to discuss further what the "ideal" data structure could be. Nevertheless, it must be observed that, at this time, existing *norms* do not give a satisfactory answer to the question of knowing what a mesh data structure should be. On the other hand, in Chapters 2 and 20, one can gain some indications about data structures when seen from a more abstract point of view or from a purely practical point of view.

The ".mesh" data structure

Flexibility and versatility have been the major concerns when designing the following mesh data structure, proposed in [George, Borouchaki-1997], Chapter 10. With no surprise, this mesh data structure is named with the suffix .mesh.

Notations. The terms in `policy` font are file items. The blanks, <<new lines>> and tabs are item separators. The comments start with the character "#" and end at the end of the line, unless if they are in a string. The comments are placed between the fields.

The notation $\left(\ldots , i=1,n \right)$ stands for an implicit DO loop.

The syntactic entities are field names, integer values `(I)` , (double) floating values `(R)` , strings `(C*)` (up to 1024 characters) being placed between `""`. The blanks and <<new lines>> are significant when used between quotes and to use a quote " in a string, one has to type it twice " ".

Booleans `(B)` : 0 for false and any other value for true (1 in general). Numbers, for instance, a vertex number, is denoted by `@Vertex`.

The entities of number type (assuming that the numbering starts from 1) are the vertex numbers **@Vertex**, the edge numbers **@Edge**, the triangle numbers **@Tria**, the quadrilateral numbers **@Quad**, the tetrahedron numbers **@Tetra**, the pentahedron numbers **@Penta**, the hexahedron numbers **@Hexa**, the numbers of a vertex in the appropriate support (described later), **@Vertex**^{supp}, the numbers of a support edge **@Edge**^{supp}, the numbers of a support triangle **@Tria**^{supp}, the numbers of a support quadrilateral **@Quad**^{supp}, the numbers of a support tetrahedron **@Tetra**^{supp}, the numbers of a support pentahedron **@Penta**^{supp}, the numbers of a support hexahedron **@Hexa**^{supp}.

In addition, $Ref\phi_i$ denotes a number based on a physical attribute.

Description *in extenso.* The data structure first includes a string identifying the release and then the various fields that can be used.

- `MeshVersionFormatted 1`
- `Dimension (I)` dim
- `Vertices (I)` NbOfVertices
  $\left(\left(\texttt{(R)}\ x_i^j\ ,\text{j=1,dim} \right)\ ,\ \texttt{(I)}\ Ref\phi_i^v\ ,\text{i=1 , NbOfVertices} \right)$
- `Edges (I)` NbOfEdges
  $\left(\texttt{@Vertex}_i^1\ ,\texttt{@Vertex}_i^2\ ,\ \texttt{(I)}\ Ref\phi_i^e\ ,\text{i=1 , NbOfEdges} \right)$
- `Triangles (I)` NbOfTriangles
  $\left(\left(\texttt{@Vertex}_i^j\ ,\text{j=1,3} \right),\ \texttt{(I)}\ Ref\phi_i^t\ ,\ \text{i=1 , NbOfTriangles} \right)$
- `Quadrilaterals (I)` NbOfQuadrilaterals
  $\left(\left(\texttt{@Vertex}_i^j\ ,\text{j=1,4} \right),\ \texttt{(I)}\ Ref\phi_i^t\ ,\ \text{i=1 , NbOfQuadrilaterals} \right)$
- `Tetrahedra (I)` NbOfTetrahedra
  $\left(\left(\texttt{@Vertex}_i^j\ ,\text{j=1,4} \right),\ \texttt{(I)}\ Ref\phi_i^t\ ,\ \text{i=1 , NbOfTetrahedra} \right)$
- `Pentahedra (I)` NbOfPentahedra
  $\left(\left(\texttt{@Vertex}_i^j\ ,\text{j=1,6} \right),\ \texttt{(I)}\ Ref\phi_i^t\ ,\ \text{i=1 , NbOfPentahedra} \right)$
- `Hexahedra (I)` NbOfHexahedra
  $\left(\left(\texttt{@Vertex}_i^j\ ,\text{j=1,8} \right),\ \texttt{(I)}\ Ref\phi_i^t\ ,\ \text{i=1 , NbOfHexahedra} \right)$
- `SubDomain (I)` NbOfSubDomain

$$\left(\texttt{(I)}\ type_i, \text{if} \left\{ \begin{array}{ll} type_i == 2: & \texttt{@Edge}_i \\ type_i == 3: & \texttt{@Tria}_i \\ type_i == 4: & \texttt{@Quad}_i \end{array} \right\}\ ,\ \texttt{(I)}\ \text{Orientation}_i\ ,\ \texttt{(I)}\ Ref\phi_i^s\ , \text{i=1 , NbOfSubDomain} \right)$$

- `Corners (I)` NbOfCorners
  $\left(\texttt{@Vertex}_i\ ,\text{i=1 , NbOfCorners} \right)$
- `Ridges (I)` NbOfRidges
  $\left(\texttt{@Edge}_i\ ,\text{i=1 , NbOfRidges} \right)$
- `RequiredVertices (I)` NbOfRequiredVertices
  $\left(\texttt{@Vertex}_i\ ,\text{i=1 , NbOfRequiredVertices} \right)$
- `RequiredEdges (I)` NbOfRequiredEdges
  $\left(\texttt{@Edge}_i\ ,\text{i=1 , NbOfRequiredEdges} \right)$

- **RequiredTriangles** (I) NbOfRequiredTriangles
  $\Big($ @Tria$_i$, i=1 , NbOfRequiredTriangles $\Big)$
- **RequiredQuadrilaterals** (I) NbOfRequiredQuadrilaterals
  $\Big($ @Quad$_i$, i=1 , NbOfRequiredQuadrilaterals $\Big)$
- **TangentAtEdges** (I) NbOfTangentAtEdges
  $\Big($ @Edge$_i$, (I) VertexInEdge, $\Big($ (R) x_i^j , j=1,dim $\Big)$,
  i=1 , NbOfTangentAtEdges $\Big)$
- **NormalAtVertices** (I) NbOfNormalAtVertices
  $\Big($ @Vertex$_i$, $\Big($ (R) x_i^j , j=1,dim $\Big)$,
  i=1 , NbOfNormalAtVertices $\Big)$
- **NormalAtTriangleVertices** (I) NbOfNormalAtTriangleVertices
  $\Big($ @Tria$_i$, (I) VertexInTrian., $\Big($ (R) x_i^j , j=1,dim $\Big)$,
  i=1 , NbOfNormalAtTriangleVertices $\Big)$
- **NormalAtQuadrilateralVertices** (I) NbOfNormalAtQuadrilateralVertices
  $\Big($ @Quad$_i$, (I) VertexInQuad., $\Big($ (R) x_i^j , j=1,dim $\Big)$,
  i=1 , NbOfNormalAtQuadrilateralVertices $\Big)$
- **AngleOfCornerBound** (R) θ
- **Geometry**
  (C*) FileNameOfGeometricSupport

  - **VertexOnGeometricVertex**
    (I) NbOfVertexOnGeometricVertex
    $\Big($ @Vertex$_i$, @Vertex$_i^{geo}$, i=1 , NbOfVertexOnGeometricVertex $\Big)$
  - **EdgeOnGeometricEdge**
    (I) NbOfEdgeOnGeometricEdge
    $\Big($ @Edge$_i$, @Edge$_i^{geo}$, i=1 , NbOfEdgeOnGeometricEdge $\Big)$
  - **TriangleOnGeometricTriangle**
    (I) NbOfTriangleOnGeometricTriangle
    $\Big($ @Tria$_i$, @Tria$_i^{geo}$, i=1 , NbOfTriangleOnGeometricTriangle $\Big)$
  - **TriangleOnGeometricQuadrilateral**
    (I) NbOfTriangleOnGeometricQuadrilateral
    $\Big($ @Tria$_i$, @Quad$_i^{geo}$, i=1 , NbOfTriangleOnGeometricQuadrilateral $\Big)$
  - **QuadrilateralOnGeometricTriangle**
    (I) NbOfQuadrilateralOnGeometricTriangle
    $\Big($ @Quad$_i$, @Tria$_i^{geo}$, i=1 , NbOfQuadrilateralOnGeometricTriangle $\Big)$
  - **QuadrilateralOnGeometricQuadrilateral**
    (I) NbOfQuadrilateralOnGeometricQuadrilateral
    $\Big($ @Quad$_i$, @Quad$_i^{geo}$, i=1 , NbOfQuadrilateralOnGeometricQuadrilateral $\Big)$
- **MeshSupportOfVertices**
  (C*) FileNameOfMeshSupport

- **VertexOnSupportVertex**
  (I) NbOfVertexOnSupportVertex
  $\Big($ @Vertex$_i$, @Vertex$_i^{supp}$, i=1 , NbOfVertexOnSupportVertex $\Big)$
- **VertexOnSupportEdge**
  (I) NbOfVertexOnSupportEdge
  $\Big($ @Vertex$_i$, @Edge$_i^{supp}$, (R) u_i^{supp} , i=1 , NbOfVertexOnSupportEdge $\Big)$
- **VertexOnSupportTriangle**
  (I) NbOfVertexOnSupportTriangle
  $\Big($ @Vertex$_i$, @Tria$_i^{supp}$, (R) u_i^{supp} , (R) v_i^{supp} ,
  i=1 , NbOfVertexOnSupportTriangle $\Big)$
- **VertexOnSupportQuadrilateral**
  (I) NbOfVertexOnSupportQuadrilateral
  $\Big($ @Vertex$_i$, @Quad$_i^{supp}$, (R) u_i^{supp} , (R) v_i^{supp} ,
  i=1 , NbOfVertexOnSupportQuadrilateral $\Big)$
- **VertexOnSupportTetrahedron**
  (I) NbOfVertexOnSupportTetrahedron
  $\Big($ @Vertex$_i$, @Tetra$_i^{supp}$, (R) u_i^{supp} , (R) v_i^{supp} , (R) w_i^{supp} ,
  i=1 , NbOfVertexOnSupportTetrahedron $\Big)$
- **VertexOnSupportPentahedron**
  (I) NbOfVertexOnSupportPentahedron
  $\Big($ @Vertex$_i$, @Penta$_i^{supp}$, (R) u_i^{supp} , (R) v_i^{supp} , (R) w_i^{supp} ,
  i=1 , NbOfVertexOnSupportPentahedron $\Big)$
- **VertexOnSupportHexahedron**
  (I) NbOfVertexOnSupportHexahedron
  $\Big($ @Vertex$_i$, @Hexa$_i^{supp}$, (R) u_i^{supp} , (R) v_i^{supp} , (R) w_i^{supp} ,
  i=1 , NbOfVertexOnSupportHexahedron $\Big)$

- **CrackedEdges** (I) NbOfCrackedEdges
  $\Big($ @Edge$_i^1$, @Edge$_i^2$, i=1 , NbOfCrackedEdges $\Big)$
- **CrackedTriangles** (I) NbOfCrackedTriangles
  $\Big($ @Tria$_i^1$, @Tria$_i^2$, i=1 , NbOfCrackedTriangles $\Big)$
- **CrackedQuadrilaterals** (I) NbOfCrackedQuadrilaterals
  $\Big($ @Quad$_i^1$, @Quad$_i^2$, i=1 , NbOfCrackedQuadrilaterals $\Big)$
- **EquivalentEdges** (I) NbOfEquivalentEdges
  $\Big($ @Edge$_i^1$, @Edge$_i^2$, i=1 , NbOfEquivalentEdges $\Big)$
- **EquivalentTriangles** (I) NbOfEquivalentTriangles
  $\Big($ @Tria$_i^1$, @Tria$_i^2$, i=1 , NbOfEquivalentTriangles $\Big)$
- **EquivalentQuadrilaterals** (I) NbOfEquivalentQuadrilaterals
  $\Big($ @Quad$_i^1$, @Quad$_i^2$, i=1 , NbOfEquivalentQuadrilaterals $\Big)$
- **PhysicsReference** (I) NbOfPhysicsReference
  $\Big($ (I) $Ref\phi_i$, (C*) CommentOnThePhysic
  , i=1 , NbOfPhysicsReference $\Big)$

- `IncludeFile (C*)` filename
- `BoundingBox` $\Big($ `(R)` Min_i `(R)` Max_i , i=1 , dim $\Big)$
- `End`

Remark 1.7 *In the following, we give some comments concerning the different fields. At first, it may be seen that some fields are mandatory while some others are optional*[11].

The comments and remarks about the fields are given according to their introduction order in the above description. Info contained in the data structure are supposed to be sufficient to make any processing possible. if the mesh tool processing the data has not the relevant capability, the corresponding info is simply ignored.

The string `MeshVersionFormatted` indicates the release identificator and the type of the file. `MeshVersionUnformatted` is an alternative case for this field.

The edge table, `Edges`, includes only, *a priori*, the edges with a significant reference number $Ref\phi$.

The elements are given with respect to their geometric nature (triangle, quadrilateral, etc.). In this way, when several types of elements coexist in the mesh, it is not required to manage a table of element types.

The sub-domains are defined using one edge in two dimensions or one face in three dimensions combined with the orientation information, (Orientation$_i$), indicating on which side of this entity the sub-domain lies. The sub-domain number is $Ref\phi^s$.

A corner point, `Corners` (for a support type structure), is a point where there is only a C^0 continuity between the edges sharing the point. Thus, a corner is necessarily an existing mesh vertex, listed in the `Vertices` list.

A ridge is an edge where there is a C^0 continuity between the faces sharing it. Thus, a ridge is necessarily a mesh edge, listed in the `Edges` list.

The required vertices, `RequiredVertices`, are the vertices of the support that must be present in the mesh as element vertices. Similarly, some edges or (triangular or quadrilateral) faces can be tagged as required entities.

The tangent vector to an edge, `TangentAtEdges`, gives the tangent vector (with respect to the surface) for this edge at the indicated endpoint. Giving the tangent vector of an edge by means of the tangent vector at a point enables us to deal with the case where several edges (boundary lines) emanate from a point.

The normal at a vertex, `NormalAtVertices`, gives the (unit) vector normal the surface at this vertex.

[11]In this way, it will be possible to add some fields that are not yet defined at the time such or such capability must be made available.

The normal at a vertex of a triangle (a face), `NormalAtTrianglesVertices`, gives the normal vector at the vertex of the specified triangle. The difference between the normal at vertices and the normal at a triangle vertex allows for local discontinuities between neighbouring triangles.

The corner threshold, `AngleOfCornerBound`, is a value which enables us to determine the continuity type between two edges or two faces that was not clearly defined or not explicitely specified.

The mesh vertices are related to the type of support in which they exist. There are two categories of supports, a geometric support and a current mesh.

When the support is of a *geometric nature*, `Geometry`, and is defined by a file, it gives the relationships between the vertices, boundary edges and boundary faces of the current mesh with the geometric entities. Thus, a mesh vertex can be identical to a geometric vertex, a mesh edge can have a geometric edge as support and, in three dimensions, a face can have a geometric face as support. These relationships allow us to classify the entities of the current mesh with respect to an entity defining the domain geometry, this information will be particularly useful when constructing finite elements of higher order.

When the support is a *(usual) mesh* by itself, `MeshSupportOfVertices`, and is defined by a file, it gives the relationships between the current mesh and the above mesh. A vertex of the current mesh belongs to an entity[12] of the support mesh. This information may be relevant when interpolating or transferring a solution from one mesh to another, in an adaptive iterative process for instance.

Hence, in an iterative computational process, the support for the mesh at a given iteration step is the mesh of the previous step. In this way, we indicate that a vertex, i, of the current mesh

- is identical with a vertex of the support,
- lies on an edge of the support at abcissa u,
- falls within a triangle of the support, u, v being the coordinates in the reference element,
- falls within a quadrilateral of the support, u, v (idem),
- falls within a tetrahedron of the support, u, v, w (idem),
- falls within a pentahedron of the support, u, v, w (idem),
- falls within an hexahedron of the support, u, v, w (idem).

A vertex not in this "table" is considered to be a free vertex. The relationships defined in this way enable us to know the location of a vertex using the reference element related to the support entity which includes this vertex. Using the reference element to come to the current element, requires using one of the following relations according to the geometric type of the element (see Figures 1.4 and 1.5 for the numbering convention):

[12]For a boundary element, a projection will be needed to obtain the desired location.

- for an edge with endpoints k_1 and k_2

$$x_i^j = (1-u)x_{k_1}^j + ux_{k_2}^j$$

- for a triangle with vertices $k_l, l = 1,3$

$$x_i^j = (1-u-v)x_{k_1}^j + ux_{k_2}^j + vx_{k_3}^j$$

- for a quad with vertices $k_l, l = 1,4$

$$x_i^j = (1-u)(1-v)x_{k_1}^j + u(1-v)x_{k_2}^j + uvx_{k_3}^j + (1-u)vx_{k_4}^j$$

- for a tetrahedron with vertices $k_l, l = 1,4$

$$x_i^j = (1-u-v-w)x_{k_1}^j + ux_{k_2}^j + vx_{k_3}^j + wx_{k_4}^j$$

- for a pentahedron with vertices $k_l, l = 1,6$

$$\begin{aligned} x_i^j = {} & (1-u-v)(1-w)x_{k_1}^j + u(1-w)x_{k_2}^j + v(1-w)x_{k_3}^j \\ & +(1-u-v)wx_{k_4}^j + uwx_{k_5}^j + vwx_{k_6}^j \end{aligned}$$

- for an hex with vertices $k_l, l = 1,8$

$$\begin{aligned} x_i^j = {} & (1-u)(1-v)(1-w)x_{k_1}^j + u(1-v)(1-w)x_{k_2}^j + uv(1-w)x_{k_3}^j \\ & +(1-u)v(1-w)x_{k_4}^j + (1-u)(1-v)wx_{k_5}^j + u(1-v)wx_{k_6}^j \\ & +uvwx_{k_7}^j + (1-u)vwx_{k_8}^j . \end{aligned}$$

Remark 1.8 *In principle, this information is naturally known by the mesh generation algorithm and is relatively easy to obtain. Moreover, when simplicial elements are used, the barycentric coordinates are trivial to obtain and thus do not strictly need to be stored.*

Crack definition is the purpose of three fields, `CrackedEdges`, `CrackedTriangles` and `CrackedQuadrilaterals`; we specify then that an edge (a face, respectively) is identical in terms of geometry to another edge (face).

The three fields, `EquivalentEdges`, `EquivalentTriangles` and `EquivalentQuadrilaterals` indicate that two edges (resp. faces) must be meshed the same way (for instance, in periodic meshes).

A comment about the meaning of the physical reference numbers is provided in the field `PhysicsReference`.

It is possible to include a file in the data structure, `IncludeFile`. This inclusion will be made without ensuring any compatibility.

For some applications, it is useful to know the size of the domain, i.e., the extrema of its point coordinates. This will be stored in the field `BoundingBox`.

The string `End` indicates the end of the data struture wherever it is encountered.

To conclude with this description, one must consider the data structure together with a suite of vizualisation and manipulation (reading, writing, converting) tools to make the things coherent and consistent altogether.

1.4 Control space and neighborhood space

Control space

It is useful to introduce the notion of a *control space* for various purposes, as will be seen in the chapters devoted to adapted meshes. Indeed, this space serves to determine the current background.

For the sake of simplicity, an *ideal* control space is simply the specification of a function $H(x,y)$ defined at any point $P(x,y)$ of $\mathbb{R}^2$ in a two-dimensional problem. In other words, function H is defined analytically and is used to specify the size and the directional features that must be conformed to by the mesh elements anywhere in the space. However, from a practical point of view, the control spaces will not be ideal in the sense that above function H is actually provided only in a discrete manner. Indeed, formally speaking, a control space can be defined as follows, [George-1991]:

Definition 1.16 *(Δ, H) is a* **control space** *for the mesh T of a given domain Ω if:*

- *$\Omega \subseteq \Delta$ where Δ covers the domain Ω,*
- *a function $H(P,\vec{d})$ is associated with every point $P \in \Delta$, where $\vec{d}$ is the direction of the disc S^1 (or the sphere S^2 in three dimensions):*

$$H(P,\vec{d}) : \Delta \times S^1 \rightarrow \mathbb{R}.$$

Thus a control space includes two related ingredients. First a covering triangulation Δ, is defined. Next, a function H is defined, whose support is a covering triangulation Δ. This pair allows for the specification of some properties or criteria to which the elements of the mesh should conform.

In terms of geometry, Δ is an arbitrary covering triangulation. For example, it can be one of the following types:

type 1: a regular partition, such as a finite difference type,

type 2: a quadtree or octree-based partition (see Chapters 2 and 5),

type 3: an arbitrary user-constructed mesh or the current mesh, for instance, in an iterative process, the last computed mesh which then serves as a background mesh.

In addition to this partitioning aspect, (Δ, H) contains, by means of the function H, the global information related to different aspects. One could be the geometry of the domain, the other could be the physics of the problem. These values allow us to determine whether the mesh T, under construction, conforms to the function everywhere.

To construct H, one can consider one of the following approaches:

- compute, from the data, the local stepsizes h (the value h being the desired distance between two points) related to the given points. A generalized interpolation then enables us to obtain H. This process is purely geometric in the sense that it relies on geometric data properties: boundary edge lengths, etc.;

- manually define the value of H for every element of Δ. A desired size is then given everywhere in the space for *isotropic* control, or the desired sizes according to specific directions are given for *anisotropic* control;

- manually specify H by giving its value for each element of the covering triangulation constructed for this purpose (we return here to a *type 3* space as introduced above);

- use the cell sizes (in the above *type 2* case), where this size is used to encode the value of H. This then leads to the construction of the (Δ, H) space so as to satisfy this requirement;

- define H from an *a posteriori* error estimate. We are then in an iterative adaptive process. A mesh T is constructed, the corresponding solution is computed and the error estimate analyzes this solution so as to complete H. Then Δ is set to T and the pair (T, H) forms the control space used to govern the new mesh construction (cf. Chapter 21).

For each of the different types, this definition results in one or the other control space types. In what follows, we will show how to create the internal points in accordance to the specifications contained in this space (for a mesh construction method) or to optimize a mesh (for a mesh optimization method).

Definition 1.17 *When the geometric locus of points $P + H(P, \vec{d})$ is a circle (resp. a sphere in three dimensions), with P in Δ and $\vec{d}$ varying, the control space is isotropic. When this locus is an ellipse (resp. an ellipsoid), the control space is anisotropic.*

Only these two cases will be discussed hereafter, leading to the definition of the metric maps which are used to govern the mesh construction process.

Remark 1.9 *The popular notion of a* background mesh, *extensively used for instance, in adaptive mesh construction processes, is nothing more than a control space. In this case the covering triangulation is the mesh of the previous iteration step while the function results from the analysis of the current solution by means of an* a posteriori *error estimate.*

Remark 1.10 *The above function H is strictly related to a metric tensor field as will be seen later. In this context the discrete meaning of $H(P, \vec{d})$ will be seen as a $d \times d$ symmetric definite matrix denoted, at point P, by $\mathcal{M}(P)$.*

Note that some authors, while using different approaches, return in fact to this notion of a control space. For instance, source points can be introduced and this information is used to complete an equivalent of the above H function, [Löhner-1989].

Neighborhood space

Close to the previous idea of control space is the notion of a *neighborhood space.* Such a space acts to help any neighboring information or queries. For instance, this is a way to facilitate the search for all the vertices located in some regions as such a query could be of great interest for various purposes.

The neighborhood space structure is similar to that of a control space. It includes two components, a spatial support together with some information related to this support. The spatial support may be defined as above with, in this case, a special emphasis on simple structures (grid, tree structure such as a binary tree or a quadtree in two dimensions) in which localization problems are evident. Along with this support, the space includes some information of a topological nature. For instance, for a grid, one encodes in its cells the fact that a point, an edge, etc., exists or not. Thus, when such a cell is queried, we know whether a point, an edge, etc., exists within a certain neighborhood. This is a simple example of what could be encoded in this kind of space.

1.5 Mesh quality and mesh optimality

The purpose of constructing a mesh is not simply a question of creating a mesh (a covering-up) of the domain of interest but to obtain a good quality mesh. Therefore, the immediate question is: " what is a good quality mesh?" or, similarly: " how is it possible to define an optimality criterion?".

In response to these questions, the literature contains a number of tentative definitions which are more or less naive (and, in some cases, fanciful to the point of eccentricity !). In this respect, let us mention some widely held, although not necessarily true, ideas. For instance, a nice looking mesh is a good mesh. This raises the problem of what "nice looking" actually means. A good mesh is a mesh whose elements have the same size and conform to a nice aspect ratio. In addition, some peremptory assertions can be found such as "a Delaunay mesh is optimal". The list goes on, but our aim here is to find some reasonable criteria that are well suited to qualifying a mesh.

A preliminary and obvious remark helps us in this discussion. The mesh is constructed for a specific purpose: to solve a given problem. Therefore, the true quality or optimality problem is related to the solution that can be computed with the mesh as a support. In this respect, it makes sense to claim that the mesh quality is good if the resulting solution quality is good. As a consequence and following the same approach, optimality is obtained if the mesh size is minimal (in some sense, for instance, if the number of nodes and vertices is minimal) resulting in a minimal cost when computing the problem solution. This characterization is then related to the nature of the problem and, moreover, implicitly assumes that a suitable way to qualify the solution quality is given.

In a numerical simulation by means of finite elements, an error estimate is the sole judge. A nice mesh is one which leads to the best possible numerical accuracy, i.e., to a minimal bound for the approximation error. For other cases, for instance, in a surface visualization problem based on a mesh of the surface of interest, the quality must be measured with regard to the approximation quality or the graphic aspect of the surface mesh with respect to the real surface. For other problems, the quality aspect may be related to different objectives.

At first, this computational process is based on an approximation of domain Ω where the problem is formulated. Thus, a first condition regarding the mesh quality is to make this approximation precise. In fact, we want to solve the given problem in domain Ω and not in an approximate domain different from the real domain. This being established (that results in conditions about the boundary elements of the mesh leading to a good boundary approximation), the mesh quality is related to the solution and thus to the nature of the problem under investigation. Using an error estimate enables us to see the quality of the mesh and, more precisely, to know if its elements conform to a desirable size, a nice directional aspect.

When such an appreciation is not available, i.e., when no error estimate is used, one can only guess in advance the mesh quality notion and it becomes then important to translate this evaluation in terms of a quality function about the mesh elements. For a problem of an isotropic nature, considering the equilateral triangle as an optimal element is a natural choice, while for a quad the square is *a priori* optimal. When no sizing information (about the element size or the desired edge length) is provided, the only reasonable behavior is to take advantage of the boundary (assumed to be appropriately sized) and to complete regular elements whose size follows at best the size of the boundary items. A finer discretization in a boundary region results in smaller elements, while a coarser discretization may lead to larger elements. Then, these sizing features will be used to find a reasonable size value for the elements located at some distance from the domain boundaries. In particular, a progressive gradation of the sizes is often a desirable feature when considering the elements in a given neighborhood.

When a size map is provided via a control space, we will see that a unit length in the metric associated with this control space is the targeted value. This leads us to the notion of a unit mesh as the good mesh we like to create.

Definition 1.18 *A* unit *mesh is a mesh with unit edge lengths.*

Comparing the edge length with the metric specification as defined in the control space provides the expected actual values. For the moment, it is enough to say that, in an isotropic context, a unit length in a metric map specifying a value h actually gives a length h in the usual sense and that a similar notion is also valid in an anisotropic context.

Remark 1.11 *A triangle with unit edge lengths is necessarily a good quality triangle. This is not the case of a tetrahedron that may have an inconsistent volume (quite small) while its edges are (about) unit edges.*

After this remark about simplicial elements and due to the other various geometries that can be envisaged, edge length control must be coupled with other criteria to make sure that a good quality is obtained.

Notice that the unit value is related to the underlying metric map and that the latter could be the combination of a map of a geometric nature (due to the domain geometry) with one or several other maps of a physical nature depending on the behavior of the problem under investigation.

★
★ ★

To conclude this brief overview of the main concepts of mesh generation, let us indicate that numerous detailled discussions will be provided in the following chapters to clarify all the notions introduced in this chapter as well as practical advices to deal with them in computational applications.

Chapter 2

Basic Structures and Algorithms

The aim of this chapter is to introduce a variety of data structures and to show how they can be used profitably in a mesh generation context. To this end, some basic as well as more sophisticated data structures are recalled together with some algorithms of greater or lesser complexity. The discussion is then developed by means of various application examples related to situations extensively used in a meshing context.

While people having a computer science background may be familiar with these basic notions, we would nevertheless like to address this topic here in order to allow people less directly concerned to gain some knowledge of this (vast) topic, notably applied mathematicians or numericians. Moreover, in the mesh generation context, specific applications and uses of the classical data structures lead to specific situations and merit some comments.

The literature about data structures and algorithms is quite abundant. Among the usual references, the textbooks would include [Aho *et al.* 1983], [Wirth-1986], [Sedgewick-1988], [Cormen *et al.* 1990], [Samet-1990] and [Gonnet *et al.* 1991] as well as the unchallengeable [Knuth-1998a], not to mention many others that can also be consulted.

The complexity of an algorithm, both in terms of the number of operations and of the memory resource allocated, is analyzed from a theoretical point of view. However, we will show that specific theoretical results obtained in *ad-hoc* academic situations must be slightly nuanced. Indeed, numerous assumptions like "the points must be in general position" in a triangulation problem or all operations involved in a given numerical process have the "same cost" or again are "exactly computed" are unlikely to be what we meet in "real life". Nevertheless, despite these remarks, theoretical results allow for a good understanding of some difficulties and will help us to find appropriate solutions.

★
★ ★

Therefore, after having described the theoretical point of view in the first sections of this chapter, we give some indications and remarks about the most commonly encountered difficulties in realistic applications. After that, we turn to some application examples to illustrate how to benefit from the theoretical material.

The first section introduces the general problem from an academic point of view. The second section presents the most commonly used elementary data structures (array, list, stack, etc.). The third section deals with complexity problems for a given algorithm. Section four analyzes the sorting and searching techniques and introduces three main paradigms used in many methods. Data structures in one and two dimensions are discussed in sections five and six, while topological data structures are mentioned in section seven. Sections eight and nine deal respectively with the notions of robustness and optimality of an implementation. The last section proposes several practical application examples.

2.1 Why use data structures?

As an introduction, we look at a "naive" algorithm that can be used to construct a triangulation. Let consider a set of points $\mathcal{S}$, each of which contained (to simplify even more) in a single initial triangle. The algorithm consists of finding, for each and any point P, the triangle K enclosing it and then to subdivide K into three new triangles by connecting P to the three vertices of K:

- For all $P \in \mathcal{S}$
  - Find triangle K containing point P,
  - Subdivide this triangle into three.
- End

While very simple, this algorithm raises several questions. Among these, we simply mention the need to define the concept of a triangulation and how to represent it, for instance, by using adjacency relationships between the triangles. Another question is related to the quick identification of the triangle containing the point P to be inserted. Should we examine all triangles of the current triangulation or take into account the fact that any triangle is obtained by subdividing its parent?

This simple example gives some indications on how to proceed and what to know to implement such an algorithm (simple as it may be). This is not indeed restricted to defining the operations required to code this algorithm, but also to finding the data structure(s) adapted to the problem, in such a way as to define a *Program*. According to [Wirth-1986], we have the following paradigm:

$$\textit{Algorithm} \quad + \quad \textit{Data Structures} \quad = \quad \textit{Program}\,.$$

There is obviously a close link between an algorithm and the data structures it uses. Usually, the more complex a data structure, the simpler the algorithm will be, although the simplicity of the algorithm is generally altered during the data structure update. For triangulation (meshing) algorithms in particular, a rich data structure allows useful data to be stored and retrieved thus simplifying the task of the algorithm, but on the other hand, any modification of the mesh induces a set of modifications and updates of the data structure. On the other hand, a simple data structure is efficient to update but forces the algorithm to perform explicitly a set of operations to recreate some data each time needed. As an example, a data structure that keeps only the adjacency relationships between the triangles of a mesh provides instantaneously the neighbors of a given triangle but requires an algorithm to identify all the triangles sharing a common vertex. However, if this information is stored, it can be retrieved immediately, provided it has been updated as the mesh evolves.

Notice that a (mesh) data structure contains integer values (numbers, indices, etc.) as well as real values (triplets of coordinates of the vertices in three dimensions, for instance). In Section 2.2, we will describe data structures allowing this kind of information to be stored.

A data structure being fixed, we discuss the behavior of the algorithm. Therefore, we recall, in Section 2.3, several fundamental notions about the complexity, allowing to analyze the efficiency of standard algorithms and basic data structures. In Section 2.4, we describe a series of algorithms, based on one of the computer science paradigms, namely, *Divide and Conquer.* We give some examples of methods for searching and sorting, some of which we describe, such as the insertion sorting technique, the quicksort and bucket sorting as well as binary searching (dichotomy) or interpolation methods. Section 2.5 discusses the manipulation of entities of dimension one (integers). To this end, we look at:

- general data structures allowing to store, retrieve or analyze sets of objects,

- structures allowing a selective access to some entities already stored. The access can be performed according to several criteria of selection. We find here, for instance, the approaches where the smallest item (in some sense), the first or the last recorded, the neighbor(s) of a given item, etc. is sought. Here we will find the data structures like stack (LIFO), queue (FIFO), priority queues, array with sorting and binary searching trees.

- data structures like dictionaries that can provide answers to questions like "does this item exist?" and allow items to be inserted or suppressed. We will find here BST and hash coding techniques.

In Section 2.6, we discuss how to use data structures in two and three dimensions for fast storing and retrieving of items such as points, segments (edges) or polygons. Section 2.7 is devoted to the computer implementation of topological data. After this overview of basic data structures and algorithms, we discuss robustness problems inherent to any implementation of a mathematical expression in a computer. The degree of the problems and the notion of predicate are then analyzed as well as the cost in terms of the number of operations and of memory requirements (Sections 2.8 and 2.9). To conclude, we mention some applications where the previously described material can be used, in the specific context of the development of mesh generation and modification algorithms (Section 2.10).

2.2 Elementary structures

In this section, we describe tables (arrays), pointers, lists, stacks and queues. These structures are briefly introduced below on deliberately simple examples.

Table or array

The table or the array is most certainly the simplest and the most efficient data structure for numerous applications. An array can be simply defined as a fixed set (connected or contiguous) of memory where items of the same nature (more precisely, items having the same storage requirement) are sequentially stored and are accessible by one or several indices. The important point is that an array allows *direct access* to each and any of its elements. Indeed, if the array begins at the address a and if each item requires b words of memory to be stored, then the item of index i starts at the address $a + (i - 1)\,b$. This simple property means that the array is a convenient data structure, easy to use and hence, is used as a basic component in more sophisticated structures (trees, hash tables, grids, etc.).

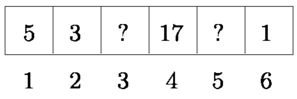


Figure 2.1: *An array of length 6.*

Figure 2.1 shows an example of an array of length 6 containing integer values. Items 3 and 5 are not yet affected and thus, the corresponding values are undefined (symbolized by the ? sign).

The intrinsic drawback of the array structure (besides the need to detect a possible overflow) is related to the static memory allocation it requires, before being used. In other words, if more space is needed at some point, a new array must be allocated and the old one should be copied into this new one.

An array allows to store vectors (a one-index array, in the usual sense), matrices (array of two indices), etc. and, as mentioned, the arrays are used as basic components in more elaborate data structures.

List

The list is another data structure in which the items are stored in a linear way (sequentially). Unlike an array in which the items follow each other in a portion of the memory, the nodes of a list can be accessed using an address in memory or, more precisely, a pointer. The notion of a pointer is naturally available in most programming languages and, if not, can be emulated as will be shown in Section 2.5.

To clarify, we now describe the case of a double linked list. In this case, each node contains three fields: the *data value* which represents the expected information and two *pointers* that allow access to the neighboring nodes. To handle a list correctly, we need two additional pieces of information: the *head* and the *tail* of the list (Figure 2.2). The head and the tail give access to the first and the last node of the list. To check whether a data value is contained in a list, it is sufficient to cover the list starting from the head, following the pointers until the data value is found or the tail is reached (then the data value does not exist in the list, except if it is in the last node).

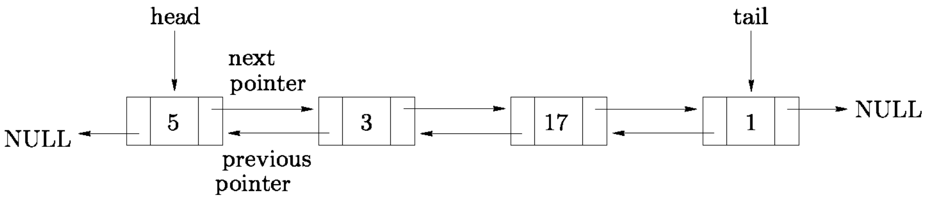


Figure 2.2: *A double linked list containing the same data values as in Figure 2.1).*

Adding or deleting an item in a list is equivalent to adding a node and updating the relevant pointers or breaking existing pointers, while managing head and tail pointers, if necessary. The following schemes illustrate the operations of searching for and inserting an item in a list, that is (with obvious notations):

Algorithm 2.1 *Searching for an item x in a list.*

```
Procedure ListContains(List,x)
$node \leftarrow head(List)$
WHILE $node \neq NULL$ AND $node.value \neq x$
    $node \leftarrow node.next$
END WHILE
RETURN $node$ (if $node \neq NULL$, then $x$ is present in the list)
```

Algorithm 2.2 *Insertion of an item x at the beginning of a list (newnode being the new element).*

```
Procedure ListInsert(List,x)
$node \leftarrow newnode$
$node.value \leftarrow x$
$node.prev \leftarrow NULL$ and $node.next \leftarrow head(List)$
$head(List).prev \leftarrow node$
$head(List) \leftarrow node$
```

Algorithm 2.3 *Insertion of an item x in a list after a given position current.*

```
Procedure ListAppend(List,current,x)
$node \leftarrow newnode$
$node.value \leftarrow x$
$node.next \leftarrow current.next$ and $node.prev \leftarrow current$
$current.next.prev \leftarrow node$ and, finally, $current.next \leftarrow node$
```

Several types of lists exist (other than the doubled linked list). Indeed, one can find the simple linked lists (one of the two pointers is omitted), the circular lists (items are accessible via a circular order), the sorted lists, etc. Irrespective of the case, a list is a simple and flexible way of managing dynamic sets of entities (the number of entities varies throughout the process), although in some cases (Section 2.3) the searching operations can be expensive. Hence, notice that lists are well adapted to cases where the considered values do not follow any specific order. Notice also that sometimes the size of the list must be known in advance (if it is implemented as an array), otherwise dynamic allocations are to be expected as it evolves.

Remark 2.1 *A list can contain data that are not single values (some languages allow for this).*

Exercise 2.1 *Explain how to implement a circular list. How many pointer(s) is (are) required?*

Exercise 2.2 *Examine a data structure based on a linked list but in which the entities point to an array. What advantages can be expected from such an organization?*

Stack

As for a plate stack, where only the plate on top of the stack can be accessed, the stack is a data structure allowing access only to the last item inserted. For this reason, it is referred to as a LIFO list (Last In First Out). The usual operations associated with a stack are twofold: *Push* add an item to the top of the stack and *Pop* remove the item on top of the stack (if one exists, i.e., if the stack is not empty).

A stack is very easy to implement using simultaneously an array (the stack itself) and an integer, the *stack pointer*, that indicates the index of the last item stored in the stack, Figure 2.3.

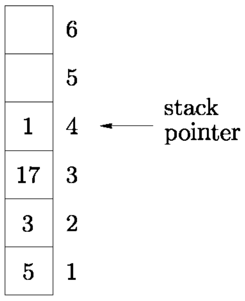


Figure 2.3: *A stack (containing the same values as the array of Figure 2.1 or the list of Figure 2.2).*

If the stack pointer is null (void), the stack is empty. If this pointer exceeds the size of the stack, the latter overflows. In this case, a solution consists of allocating an array that is bigger and copying the old stack in this new structure. Another solution consists of using a linked list (see next exercise).

Exercise 2.3 *Can we use a simple linked list or should we use a double linked list to implement a stack?*

Queue

A queue is a data structure whose behavior is very close to that of a queue of persons (as can be seen in a post office[1]). The main operation with a queue is to access the next entity, which is equivalent to removing this item from the queue. Moreover, each new item is appended at the end of the queue. According to this logic, we have a *FIFO* structure, First In First Out. If the data structure used to implement a queue (an array for instance) is of bounded size, an *overflow* is encountered whenever an item is added to an already full queue. On the other hand, attempting to remove an item from an empty queue leads to an *underflow.*

A simple linked list offers all the required facilities to implement a queue. However, if the maximum number of items to be stored is known in advance, an array is preferable as it avoids the memory allocation or deallocation problem and the possible overflow. For more details, we refer the reader to [Cormen *et al.* 1990] and to the following exercise.

Exercise 2.4 *Find a data structure to implement a queue using an array of fixed size, and address the problems of overflow and underflow (Hint: if the head (the tail) of the queue is reached and if free space is available in the array, "move" (pack down) the items of the queue).*

Objects and pointers

In Section 2.2, we assumed that it is possible to associate to each entity one or two additional fields (the pointer(s)) indexing nodes of the same type. On the one hand, the notion of pointer may not exist in some languages and, on the other hand, if the number of items is known in advance, the use of pointers serves no purpose. To demonstrate this, let us look at memory allocation/deallocation problems in this context. A simple way of handling this situation is to manage a list of free memory entries, a *free list.* When more space is needed, we apply a *First-Fit* method. The list is explored and the first free available block of memory (of appropriate size) is used. If this area is bigger than necessary, it is split and the remaining block is appended to the free list. In a similar way, the addition of a free block is performed by referencing it in the free list (possibly by merging it with the neighboring memory blocks if such blocks are free).

This point is emphasized in Figure 2.4 where H represents the pointer to the head of the free list. Notice that two integers are associated with each information

[1]Without wishing to make any unfair assumptions about this type of institution in any particular country.

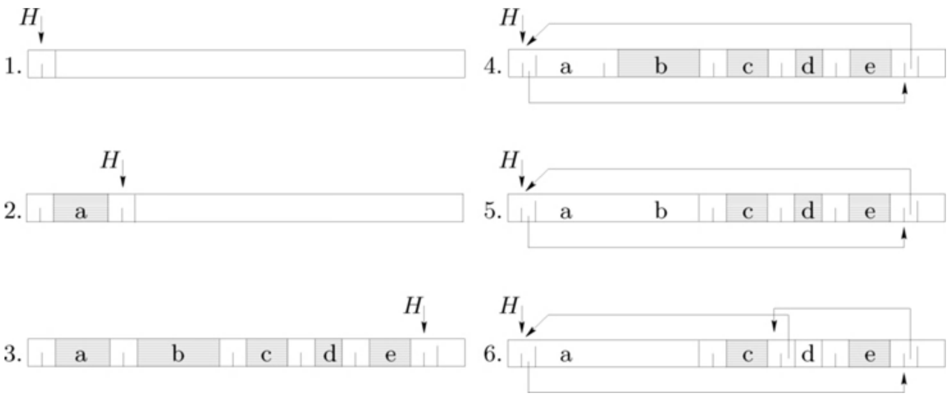


Figure 2.4: *Memory management with the First-Fit strategy.*

unit, a pointer to another block linking the free list) and an integer giving the actual block size (used in the First-Fit operation). In the example, we have allocated a, b, c, d and e in line. If a becomes available and if b also becomes available, then the area $a \cup b$ is defined as free (available). Then, d becomes available. Hence, the free list is maintained.

Fortunately, more sophisticated yet less time consuming memory management mechanisms exist. Nevertheless, if the size of the memory required is not known in advance or if the notion of pointer does not exist, attention must be paid to the memory management procedure. We emphasize this point in the case where a double linked list stored in an array is used. Each record contains the given information and two integers defining the next and previous records in the list. The example of Figure 2.5 illustrates this implementation for the list given in Figure 2.2. The same representation can be obtained using three arrays (one for the value, one for the pointer to the next and one for the pointer to the previous item) instead of only one.

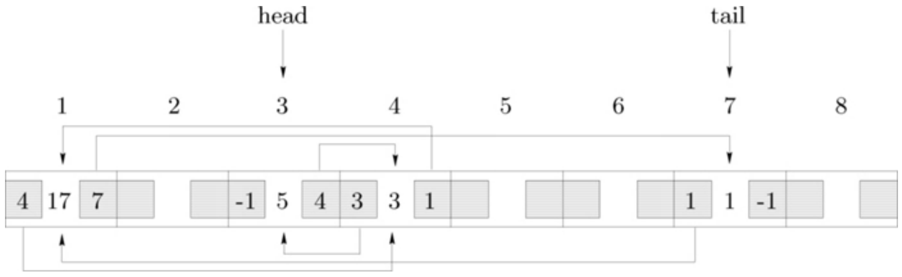


Figure 2.5: *Linked list of Figure 2.2 implemented with an array. Middle, the list, top, the pointer to the following node and bottom, the pointer to the previous node.*

2.3 Basic notions about complexity

We discuss here the problems related to the complexity of an algorithm. As will be seen, this notion concerns various aspects and affects the perception of the algorithmic "quality".

Behavior of a function

As seen in Section 2.1, the construction of an algorithm and/or a data structure requires evaluating the number of operations involved and, in addition, looking closely at memory problems.

Addressing these points involves analyzing the *complexity in time* as well as the *complexity in size* of the algorithm and/or the data structure. A simple and generic way to analyze these notions is to introduce a mathematical function f related to the size n of the inputs of the problem, where for instance, n is the number of points to be inserted in the triangulation. Finding the exact value of $f(n)$ can be difficult or even impossible. However, we are usually not interested in this value, but rather in a rough estimate of it. Several ways of quantifying this point exist. The usual notations are $f(n) \equiv \Theta(g(n))$ or $\Omega(g(n))$ or $\mathcal{O}(g(n))$ or $o(g(n))$ which indicate respectively that, for a sufficiently large n and for a "known" $g(n)$, we have:

- Θ: $c_1\, g(n) < f(n) < c_2\, g(n)$ where c_1 and c_2 are two constants. Hence, f and g have the same behavior when n grows.
- $\mathcal{O}$: $f(n) < c\, g(n)$, c being a constant, and we have an upper bound,
- Ω: $f(n) > c\, g(n)$ where c is a constant. We have then a lower bound,
- o: $f(n) < c\, g(n)$ for any positive constant c.

The quantifications in $\mathcal{O}$ and o can be seen as two ways of comparing two functions from slightly different points of view when looking at the bounds. One way of qualifying these two measures is indeed to write, for $\mathcal{O}$:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} < c\,, \quad \text{while for } o\,, \text{ we have:} \quad \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0\,.$$

As an example, notice that $2\,n^2 = \mathcal{O}(n^2)$ but $2\,n^2 \neq o(n^2)$, while we have both $n\, log\, n = \mathcal{O}(n^2)$ and $n\, log\, n = o(n^2)$. These relations indicate the asymptotic behavior of the functions, see Figure 2.6.

Notice however that the previous expressions involve some constants that can be very large. For instance, an algorithm of complexity n^2 is *a priori* faster than an algorithm of complexity $100\,n$, if n is smaller than 100 while it is slower if n is larger than 100. Another remark is that the complexity measures the behavior for large input sizes rather than for small size problems. This will be discussed in Section 2.4 when dealing with sorting algorithms.

Exercise 2.5 *For which values is an algorithm in n^3 more efficient than an algorithm in $1000\, n^2\, log\, n$?*

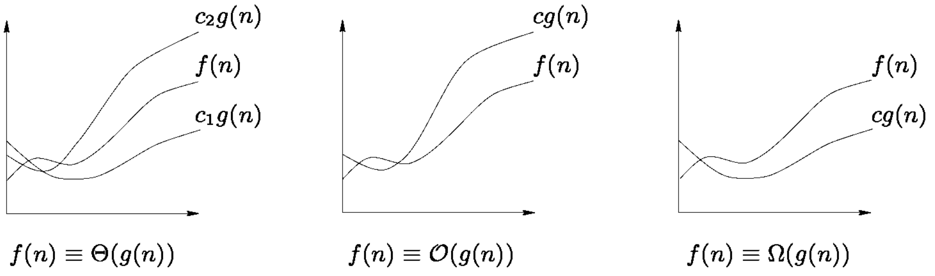


Figure 2.6: *Illustrations of the notations* Θ, $\mathcal{O}$ *and* Ω.

Exercise 2.6 *Let f and g be two functions asymptotically positive. Among the following assertions, indicate which are true and which are false:*

- *i)* $f(n) = \mathcal{O}(g(n))$ *with* $f(n) = \sqrt{n}$ *and* $g(n) = n^{\sin n}$,
- *ii)* $f(n) + g(n) = \Theta(min(f(n), g(n)))$,
- *iii)* $f(n) = \mathcal{O}(g(n))$ *means* $2^{f(n)} = \mathcal{O}(2^{g(n)})$,
- *iv)* $f(n) = \mathcal{O}(g(n))$ *means* $(1 + \frac{1}{logn})f(n) = \mathcal{O}(g(n))$.

Complexity, worst case, average case, optimal case

Looking back to the study of the complexity of an algorithm, one has to notice that several complexities can be defined. More specifically, related to:

- the complexity in the most favorable case and that in the worst case, i.e., the minimum and the maximum number of operations strictly required,
- the average complexity, i.e., the complexity obtained by averaging the complexity on a series of cases.

The worst and optimal complexities are usually easy to determine as it is sufficient to look at the extreme configurations. The average complexity, however, requires the introduction of probabilities. These notions are now illustrated:

Algorithm 2.4 *Searching for a value x in the array Tab.*

```
Procedure IsContainedInArray(Tab,x,found)
$i \leftarrow 1$
$found \leftarrow .FALSE.$
WHILE ($found = .FALSE.$) AND ($i \leq n$)
   IF $Tab(i) = x$ THEN $found \leftarrow .TRUE.$
   ELSE $i \leftarrow i + 1$
END WHILE
IF $i = n + 1$ THEN $found = .FALSE.$ (lost, $x$ has not been found),
ElSE $found = .TRUE.$ (found)
END IF
```

where the goal is to see whether a given value x belongs to the array Tab at an index i, the length of Tab being n. The problem is to determine how many comparisons are performed in this algorithm in the optimal case, the worst case and on average.

If the first entry of the array is x,the algorithm stops after only one test and its optimal complexity is 1. If the array does not contain x, n comparisons are necessary to conclude. To find the average, in the case where x exists in the array, we find a number of comparisons equal to:

$$E(x \in Tab) = \sum_{i=1}^{n} i \, Pr(x,i)$$

where $Pr(x,i)$ represents the probability of finding x at the index i of the array. As $Pr(x,i) = \frac{1}{n}$, we find that $E(.) = \frac{n+1}{2}$, which can be written as $E(.) = \Theta(n)$. In other words, if the array is not sorted, a number of tests of the order of n is expected on average.

These three measures of complexity show different information. More precisely, the worst complexity is a good measure whenever the time to run an algorithm has been fixed. We find here the concrete situations where we expect to have a given complexity, for instance, linear, depending on the sizes of the input of the problem. Actually, in this case, the worst case gives the desired information.

Exercise 2.7 *The previous example, in the line* WHILE, *requires two tests of equality and a comparison, hence three tests. Show that the number of tests can be reduced to only one if x, the sought value, is inserted in the array at the index $n+1$ (by increasing its size by 1, this new node being called* sentinel*).*

Amortized complexity

Another notion of complexity, known as the amortized complexity, measures the average performance of each operation in the worst case. More precisely, some algorithms or structures are such that the more costly operations are very rarely executed. In some other cases, a costly operation means that the required operations will be inexpensive afterwards. An example of such a behavior is given in the case of a stack.

Let consider a stack and let us assume that the operator *Multi-Pop(Pile,k)* which consists of applying the operator *Pop* to k items (for k smaller or equal to the size of the stack). Let us look at a sequence of n *Push, Pop* and *Multi-Pop*. A *Push* and a *Pop* are in $\mathcal{O}(1)$ while a *Multi-Pop* is in the worst case in $\mathcal{O}(n)$, hence, the worst case for a sequence of n operations is in $\mathcal{O}(n^2)$. Is this the true complexity of the algorithm?

As each item cannot be popped more than once, the number of *Pop* (*Pop* and *Multi-Pop*) is at most equal to the number of *Push*, that is about $\mathcal{O}(n)$. Then, whatever the value of n, a sequence of *Push, Pop* and of *Multi-Pop* takes a time in $\mathcal{O}(n)$. Indeed, the amortized complexity of the operation is in $\frac{\mathcal{O}(n)}{n} = \mathcal{O}(1)$. To summarize, this quantity measures the average efficiency of each operation in the worst case during the execution of a set of operations.

2.4 Sorting and searching

Numerous methods may be used to sort an array containing items on which an ordering is defined.

Sorting by comparison algorithms

Sorting by insertion. Let us consider an array of size n and let us assume, at the stage i of the processing, for $i = 1, ..., n-1$, that the sub-array i (between the indices 1 and i) has already been sorted. The algorithm of *sorting by insertion* consists of inserting $v = Tab(i+1)$ in the sub-array i in the right place, by moving the items greater than v towards the right. This can be written as follows:

Algorithm 2.5 *Sorting by insertion from the smallest to the largest.*

```
Procedure InsertionSort(Tab)
FOR $i = 2, n$
    $value \leftarrow Tab(i)$
    $j \leftarrow i - 1$
    WHILE $j > 0$ AND $TAB(j) > key$
        $TAB(j+1) \leftarrow TAB(j)$
        $j \leftarrow j - 1$
    END WHILE
    $TAB(j+1) \leftarrow value$
END FOR
```

to obtain a sorting algorithm from the smallest to the largest. In trems of complexity, the relevant quantity in the analysis of this sorting algorithm is clearly the number of items moved towards the right side. If the input is seen as a permutation of n different numbers, this quantity can be easily seen as the number of inversions in this permutation (i.e., the sum for all elements of the number of larger items located on the left hand side).

If the array is originally sorted in the reverse order, the worst complexity is obtained. It corresponds to $\sum_{i=1}^{n} i$, that is $\mathcal{O}(n^2)$. Finding the average complexity is more difficult and requires constructing a random model. Here, we simply indicate that the required number of permutations is equally probable. Under this assumption, the average number of permutations is still in $\mathcal{O}(n^2)$.

While not really efficient in the worst case and on average, a sorting algorithm by insertion is still very useful for small examples or in cases where the data are already almost ordered. In the last case, a simple comparison is sufficient to decide whether or not each item is in the right place. Hence, this sorting algorithm is almost linear in time.

Exercise 2.8 *Indicate how to use this sorting algorithm to sort a linked list (its values). Is the overall complexity of the process affected?*

Quicksort. The quicksort algorithm sorts in place, that is by permutating the data. This method is widely used because it has proven to be robust, efficient and easy to implement. Its efficiency is close to the optimum. Its robustness is mainly related to the fact that it is insensitive to the properties of the values to be sorted. The ease of implementation is due to the simplicity of the underlying concept of permutation.

Indeed, the main idea consists of taking one item of the array, the *pivot*, and splitting the array into two pieces around the pivot. The elements which are larger (resp. smaller) are placed on the right-hand side (resp. left-hand side) and the process is iterated on each of the sub-arrays. The splitting procedure is also simple and consists of scanning the array and exchanging the elements larger than or smaller than the pivot (once it has been fixed). The sorting algorithm can be written as follows:

`Quicksort(Tab,left,right)`

where Tab is the array to be sorted and $left$ and $right$ are the left and right indices of this array (for instance, $left = 1$ and $right$ is the number of values in Tab). This procedure is written recursively as follows:

Algorithm 2.6 *Quicksort from the smallest to the largest.*

```
Procedure Quicksort(Tab,left,right)
IF $left < right$
    $m \leftarrow Partition(Tab, left, right)$
    $Quicksort(Tab, left, m-1)$
    $Quicksort(Tab, m+1, right)$
END IF
```

The procedure *Partition* corresponds to the following algorithm:

Algorithm 2.7 *Procedure Partition for the Quicksort algorithm.*

```
Procedure Partition(Tab,left,right)
$ipivot \leftarrow \frac{left+right}{2}$
$pivot \leftarrow Tab(ipivot)$
$j \leftarrow left$
Exchange $Tab(left+1)$ and $Tab(ipivot)$
FOR $i = left+1$ TO $right$ DO
    IF $TAB(i) < pivot$,
        $j \leftarrow j+1$ and exchange $Tab(i)$ and $Tab(j)$,
    END IF
END FOR
exchange $Tab(left)$ and $Tab(j)$
RETURN $j$
```

In [Knuth-1998b], it is proved that the average complexity of the *quicksort* is in the order of $1.38\, n \log_2 n$, where the logarithm is taken in the base of 2, thus leading to $\mathcal{O}(n \log n)$. This is an interesting result because the sole operation used in the sorting algorithm is the two-two comparison of the elements. It is then possible to prove that, on average, each sorting algorithm requires a minimum number of comparisons is on the order of $\Omega(n\, log\, n)$, in the worst case.

Notice however, that the worst complexity is in $\mathcal{O}(n^2)$. This is obtained when the array is already sorted in such a way that, at each step of the recursion, the smallest (resp. largest) element is picked as the pivot. To avoid this unbalanced case, a solution consists of choosing the pivot in a different way. One strategy can be to pick an element randomly in the array. But, using a random number generator in this case can be seen as superfluous. The easiest way to improve the efficiency is to use the technique known as the *average of three* in which the pivot is chosen as the average value of the element on the left, middle and right of the array.

The version of this algorithm described above can be improved in two different ways, thus leading to almost 30 % speedup, depending on the implementation. Firstly, the recursion can be avoided by using a loop. This requires storing the bounds of the sub-arrays for a further processing. Then, to avoid the time devoted to pushing and poping the data because of the recursion, a simple sorting algorithm by insertion can be performed in place of a recursive call when the number of entities to be sorted is very small. In a similar way, the small sub-arrays can be kept (not processed) during the recursion and sorted by insertion in the whole set of data. The critical size below which a recursion is not efficient depends on the implementation and is found to range between 2 and 25.

Exercise 2.9 *Analyze the case of a quicksort algorithm that does not account for the sub-arrays of size smaller than k, these being processed using a sorting algorithm by insertion on the whole set. Show that the expected complexity is in* $\mathcal{O}(n\, k + n\, log\frac{n}{k})$.

Exercise 2.10 *Replace, in the algorithm* `Quicksort(Tab,left,right)` *above, the two recursive calls by a loop using a stack for the two sub-arrays.*

Bucket sorting

We have just seen that the quicksort algorithm is based on two-two comparisons of the elements. Hence, the number of operations is only related to the order of the data and not to the specific value to be sorted.

A dramatically different approach for sorting is based on the use of the value of the entities to be sorted in such a way as to separate them. More specifically, the domain containing the entities to sort is divided into equally sized pieces of size δ and each entity is associated with the block containing it. This association is obtained using a function (integer part), denoted $\lfloor\ \rfloor$, and then, all items belonging to the same block are chained together in a linked list.

The general scheme of a (recursive) *bucket sort* is the following:

Algorithm 2.8 *Recursive bucket sort.*

```
Procedure Bucketsort($x_1, x_2, \dots, x_n$)
$x_{min} \leftarrow min(x_1, x_2, \dots, x_n)$
$x_{max} \leftarrow max(x_1, x_2, \dots, x_n)$
$\delta \leftarrow \lfloor \frac{1}{n_b}(x_{max} - x_{min}) \rfloor$
FOR $i = 1, n$
    $idx \leftarrow \lfloor \frac{1}{\delta}(x_i - x_{min}) \rfloor$
    add $x_i$ in $List(idx)$
END FOR
FOR $i = 1, n_b$
    IF the size of $List(i)$ is at least 1, $Bucketsort(List(i))$
END FOR
```

Remark 2.2 *The number n_b of blocks can be taken to be equal to the number n of data, so as to achieve an balanced partition. However, when n becomes very large, it is desirable to choose $n_b < n$ (if only for memory allocation problems).*

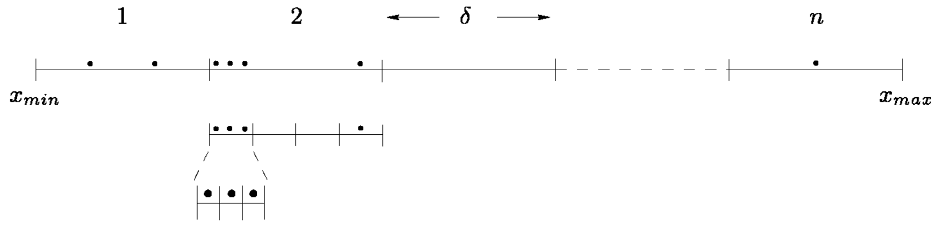


Figure 2.7: *Example of a block defined during the construction of the partition related to the bucket sorting.*

If each block contains $\mathcal{O}(1)$ points after the block construction, then the partition is said to be balanced. In this favorable case, the sorting algorithm is of linear complexity. This is not certain if some blocks contain a lot of points and, in this case, a solution consists of sorting these points recursively. Moreover, if at each stage, all points but one belong to the same block, for instance, if $x_i = i!$, for $i = 1, ..., n$, the recursion requires $n - 1$ levels and the time is $\sum_{i=1}^{n} i$ that is $\mathcal{O}(n^2)$.

The efficiency of bucket sorting is thus related to the number of points to be sorted in an interval of size δ, which is related to the distribution of the x_i's. More precisely, in [Devroye-1986], it is shown that points for which the distribution function is of compact support and of square integrable[2] are sorted in linear time by recursive bucket sort[3]. Intuitively, this just means that although some regions seem to be highly populated in the original sample, a good separation of the points is achieved after a small number of recursions with interval lengths δ.

[2]If f is this density function, f is said to be square integrable if the integral of its square converges, i.e., $\int f^2 < \infty$.

[3]Sufficient conditions are usually well-known for a probability density to be sorted in linear time. But coming up with necessary conditions remains an open issue.

Remark 2.3 *We have described here the bucket sort for one-dimensional data (integer or real numbers for instance). This technique can also be applied to data in $\mathbb{R}^2$ or $\mathbb{R}^3$, for example points that need to be sorted according to various criteria.*

Remark 2.4 *Notice also, as for the previous examples, that numerical problems are not taken into account in the discussion. Indeed, what happens if a point is located on the right-hand side of a point (being considered as larger, within the roundoff errors) when it should be on the left-hand side (mathematically speaking)?*

Leaving these remarks to one side, does this mean Bucket sort should be preferred to Quicksort when the dataset to be sorted is known to have certain properties? The answer is not that clear: Quicksort sorts an array in place whereas Bucket sort requires more memory (several points may fall within the same bucket). So, depending upon the implementation, the $\mathcal{O}(n)$ time algorithm may outrun another one in $\mathcal{O}(n \log n)$ for some sample sizes.

Searching algorithms and dichotomies

In Section 2.3, we have analyzed the performances of a sequential search in an array and we have shown that a linear time complexity can be achieved for a fruitful search as well as for a failure. We now turn to a more powerful strategy, the *dichotomy* that leads to a boolean result (positive or negative) in a $\mathcal{O}(\log n)$ time.

The intrinsic weakness of a sequential search is that, at each step, the comparison performed between the sought value x and the part of the array analyzed does not provide any global information about the array. However, for a sorted array, this drawback can be avoided. Indeed, by comparing x with the element in the middle of the array, one can decide which part of the array should contain x. A simple comparison thus allows the number of potential candidates to be divided by two. As the size of the problem decreases by a factor 2 at each step, the size of the searching domain is reduced to 1 after $\log n$ comparisons. Such a searching algorithm based on dichotomy follows the general scheme:

Algorithm 2.9 *Dichotomy search of an item x in an array Tab.*

```
Procedure IsContainedInSortedArray(Tab,x,found)
$l \leftarrow 1$, $r \leftarrow n$,
WHILE $l \neq r$
   $idx \leftarrow \lfloor \frac{l+r}{2} \rfloor$
   IF $x \leq Tab(idx)$, THEN $r \leftarrow idx$
   ELSE $l \leftarrow idx + 1$
   END IF
END WHILE
IF $x = Tab(l)$, $found = .TRUE.$,
ElSE $found = .FALSE.$
END IF
```

Despite its performances, we must notice that a dichotomy search is not always the best method. For instance, if we want to find a name beginning by B in a phone book, it is usually recommended to start from the beginning of the book rather than from the end! This simple remark suggests an improvement in the binary search. When the value x is searched for in a sub-array indexed from l to r, we start by estimating the place where x is most likely to be. The natural way of doing this is to interpolate x in the sub-array $Tab(l, ..., r)$, which is equivalent to replacing the index idx in the above scheme by:

$$idx = l + \left\lfloor (r-l)\frac{x - Tab(l)}{Tab(r) - Tab(l)} \right\rfloor .$$

For this reason, the searching algorithm, known as interpolation-search algorithm or interpolating search, is very similar (in its concept) to a bucket sort. Its efficiency is strongly related to the properties of the dataset. It can be proven that for a large variety of datasets (in terms of density), a successfull search or a failure can be achieved in $\mathcal{O}(log\, log\, n)$.

Another search strategy of the same kind consists of using a bucket tree to store the elements. Similarly, the complexity is related to the partition density of the elements, see [Devroye-1986].

Exercise 2.11 *The* `WHILE` *statement in the algorithm is skipped if $Tab(idx) = x$. Does this affect the complexity of a successful search (x is found) or a failure (x is not in Tab)?*

Three main paradigms

Before going further, we introduce three paradigms that will be used later and which can be considered fundamental in algorithmics.

The first paradigm is known as **Divide-and-conquer**. Briefly speaking, it consists of solving a problem $\mathcal{P}$ by

i) dividing $\mathcal{P}$ into several sub-problems, for instance into two sub-problems $\mathcal{P}_1$ and $\mathcal{P}_2$ (of smaller sizes),

ii) solving the sub-problems, here $\mathcal{P}_1$ and $\mathcal{P}_2$,

iii) merging the partial solutions together.

The recursive division stops in practice when the sub-problem becomes sufficiently small and its solution is easy to obtain. The number of sub-problems created and the way of merging the solutions are related to the nature of the problem at hand. For instance, the Quicksort divides a problem in two and the merging operation simply consists of putting the solutions end to end.

The second paradigm concerns the **Computational model**. This consists of defining the type of operations that may be used when devising an algorithm. For instance, in quicksort or in bucket sort, we noted the difference between the former approach, which uses only pairwise comparisons, and the second approach which requires the floor function $\lfloor\, \rfloor$. We noticed that this resulted in complexities that are sensitive to various properties of the dataset (random permutations versus probability densities in our examples).

The last one is the **Randomization** paradigm which we encountered with the pivot selection in the quicksort algorithm. Broadly speaking, making choices at random is an elegant and efficient way to avoid worst-case complexities with high-probability. Intuitively speaking, if an event of bad complexity occurs with some probability, having it occur over a long sequence is very unlikely. Thus, this technique will be widely used in the following.

These three techniques are of course independent. In particular, quicksort-like strategies as compared with bucketsort-like ones can be viewed as two independent implementations of the Divide-and-Conquer paradigm. The former splits the task into a constant number of sub-problems while the latter attempts to make decisions faster using a higher branching factor.

Exercise 2.12 *Let $List_1$ and $List_2$ be two ordered linked lists of sizes n and m respectively. Show that they can be merged in time $n + m - \nu(n, m)$ by changing pointers only, with $\nu(n, m)$ the number of items of $List_1$ bigger than the largest element of $List_2$ (or vice versa).*

Let List be a list containing n items. Show that List can be recursively sorted by first splitting it into two equally sized sub-lists, sorting these sub-lists and merging them. What is the time complexity of this method? (This sorting algorithm is known as the Merge sort*).*

Exercise 2.13 *Let $L_0 = (x_1, x_2, ..., x_n)$ be a set of n real numbers. Suppose also we have a coin and define L_{i+1} from L_i as follows: for each x in L_{i+1}, toss the coin and add x to L_{i+1} if the output is heads. Now, call e the first integer such that L_e is the empty-set. Show that $E(e) = \mathcal{O}(log\, n)$ and $Pr(e \geq \alpha\, log n) \leq \frac{1}{n^{\alpha-1}}$.*

2.5 One-dimensional data structures

One-dimensional data structures for handling (one-dimensional) objects are considered amongst the most fundamental since they are the building blocks on which more involved algorithms and data structures are based. In Section 2.2, we saw how to store unordered objects. In this section, we shall see how to define *dictionaries* and *priority queues*. As for the data structures already described, these can easily carry out operations such as "is this element contained in", "insert" or "delete" an element and, moreover, these are more geared to handling requests such as "find the min", "find the max".

Binary tree

In Section 2.4, we used a Divide-and-Conquer paradigm to search a sorted array. We noticed the running time improvement over the naive algorithm of Section 2.3.

In this section, we do the same in a dynamic context based on a *Binary Search Tree* (referred to hereafter as *BST*) that improves the complexity of any searching operation on linked lists. We start with a definition.

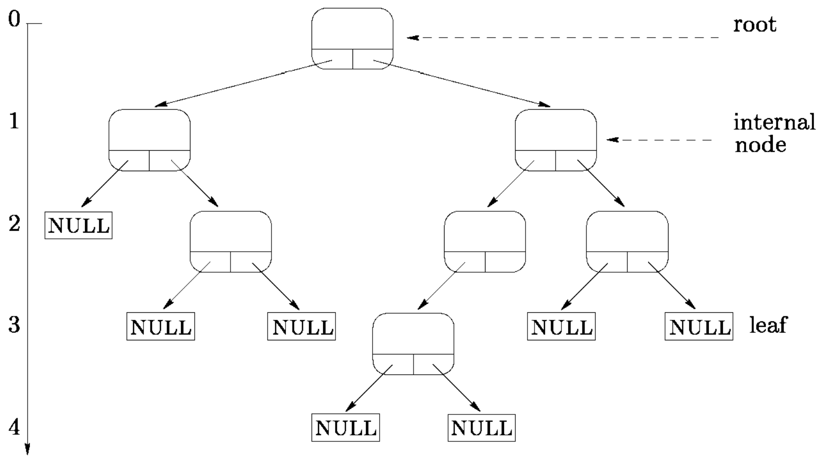


Figure 2.8: *Pointer representation of a binary tree.*

Definition 2.1 *A binary tree is a data structure whose node contains, in addition to the information field (the value), two pointers; the left and the right children. If the information field obeys some ordering relationship, such a tree is called a* Binary Search Tree *(BST).*

The topmost node is called the *root* of the tree. A distinction is made between a node that has children, called *internal*, and a node without children, called *terminal* or a *leaf*. The *depth* (height) of a node is the number of edges (branches) crossed from the root to that node (Figure 2.8).

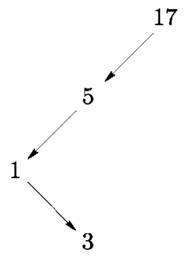


Figure 2.9: *Binary tree constructed from the sequence of values* 17, 5, 1 *and* 3. *Notice that in this tree, according to the sequence of the values, the nodes have only one child. A different ordering could lead to nodes having two children.*

An example of binary search tree growth is illustrated in Figure 2.9 where the insertion of the values 17, 5, 1 and 3 is depicted. First, 17 is inserted and stored at the root since the tree is empty. Then 5 is inserted and put in the left subtree as it is smaller than 17. Similarly, 1 goes to the left of 5 and 3 to the right of 1. The resulting tree has one internal node at depths 0, 1, 2 and a leaf 3, it has an edge at depths 1, 2 and 3.

To see which parameters influence the operations on BST, let us consider the searching and insertion procedures whose general schemes are given below:

Algorithm 2.10 *Searching in a BST.*

```
Procedure Contains(node,x)
IF $node = NULL$ THEN $return = .FALSE.$, END
IF $node.value = x$ THEN $return = .TRUE.$, END
ELSE
   IF $x \leq node.value$ THEN $Contains(node.left, x)$
   ELSE $Contains(node.right, x)$
   END IF
END IF
```

Algorithm 2.11 *Inserting a value in a BST.*

```
Procedure Insert(node,nodeFather,x)
IF $node = NULL$
   allocate a new node, $newnode$
   $newnode.value \leftarrow x$
   $newnode.left \leftarrow NULL$ and $newnode.right \leftarrow NULL$
   IF $x \leq nodeFather.value$ THEN $nodeFather.left \leftarrow newnode$
   ELSE $nodeFather.right \leftarrow newnode$
   END IF
END IF
IF $x \leq node.value$ THEN $Insert(node.left, node, x)$
ELSE $Insert(node.right, node, x)$
END IF
```

Basically, the strategy consists of tracing a path down the tree and making the right decision at each node encountered. If a value already present in the tree is asked for, the searching process stops in an internal node or in a leaf (terminal node). If the value is not in the tree, the process ends up in a node and adds a child to it. The average cost of a search is related to the sum of the depths of the node in the tree. As for the worst-cases, these quantities are bounded by the depth of the tree, h_n which is such that $\lfloor log_2 n \rfloor \leq h_n \leq n-1$. On average, see [Mahmoud-1992], the expected depth of a random tree containing n values is $E(h_n) \approx 2.98\, log\, n$.

Randomly building BST trees is therefore interesting and is easy to handle if one can possibly afford bad performances. However, should this not be the case or should deletions be allowed (very little is known for a random tree after a sequence of deletions and insertions), a different strategy must be applied.

Bad performances clearly arise from "skinny" and "elongated" trees. This is, for example, the case in Figure 2.9. One would prefer the configuration given in Figure 2.10, right-hand side.

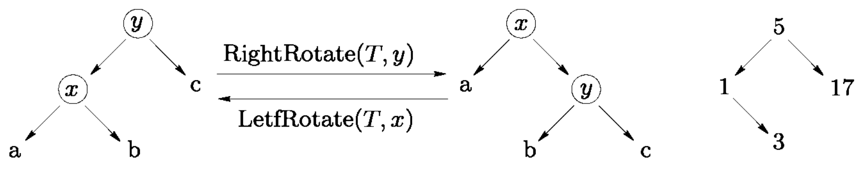


Figure 2.10: *Left hand side: balancing by rotation around a node. Right hand side: balancing the tree of Figure 2.9.*

To avoid this type of situation, the strategy consists of *balancing* the tree to have its subtrees containing roughly the same number of items. The relevant elementary operation is the rotation around a node depicted in Figure 2.10.

In practice, there are several methods to balance a tree. For instance, *AVL*[4] trees, [Wirth-1986], are such that for any node the depths of the two subtrees differ by at most one. For the red-black trees [Cormen *et al.* 1990], the balancing is achieved by some constraints satisfied by the color of the nodes. The reader is referred to the cited references for more details about tree balancing, an operation that reveals to be a bit tricky.

The performances of the red-black trees are summarized in the following theorem:

Theorem 2.1 *The depth of a red-black tree containing n values is bounded by* $\frac{log\, n}{2} < h_n < 2\, log\, n$. *The operations "insert", "delete", "exists", "find the min", "find the max" are in* $\mathcal{O}(log\, n)$.

We now give some schemes for traveling through (traversing) a binary tree.

Algorithm 2.12 *Tracing a path in a binary tree.*

```
Procedure inOrderProcessing(node)
IF $node \neq NULL$
   $inOrderProcessing(node.left)$
   process $node$
   $inOrderProcessing(node.right)$
END IF
```

Algorithm 2.13 *Tracing a path in a binary tree (pre-order processing).*

```
Procedure preOrderProcessing(node)
IF $node \neq NULL$
   process $node$
   $preOrderProcessing(node.left)$
   $preOrderProcessing(node.right)$
END IF
```

[4]Acronym for Adelson, Velskii et Landis, inventors of this type of tree.

Algorithm 2.14 *Tracing a path in a binary tree (post-order processing).*

```
Procedure postOrderProcessing(node)
IF $node \neq NULL$
   $postOrderProcessing(node.left)$
   $postOrderProcessing(node.right)$
   process $node$
END IF
```

Exercise 2.14 *A m-ary search tree is defined as a search tree whose nodes have exactly m children (a node contains the value and m pointers). How do m-ary search trees compare against BST in terms of searching time and memory requirements?*

Exercise 2.15 *Show how a red-black tree can be used for sorting in $\Theta(n \log n)$.*

Hashing

Like the bucket sort, the hash functions consist in splitting the dataset processed into bins or buckets. The hashing process is viewed here as a one-dimensional structure, in particular to emphasize the fundamental difference between the hashing technique and the bucket sort.

However, it seems obvious that this structure is adequate for multi-dimensional data and, practically, is commonly used in such situations. If h denotes the hash function and if x is the element considered, then $h(x)$ is the hash value associated with x, as will be seen hereafter. In three dimensions, we will find, with obvious notations, $h(x, y, z)$ the hash value associated to the element (x, y, z).

There is however a fundamental difference between the hash function used in bucket sort, namely $h(x) = \lfloor \frac{1}{\delta}(x - x_{min}) \rfloor$, and general hash functions. While the former is monotonic, i.e., if $x \geq y$ then $h(x) \geq h(y)$, this property is not strictly required in the latter case. Moreover, general hash functions are usually implemented using *modulo*, see Figure 2.11, left-hand side, for instance. It should

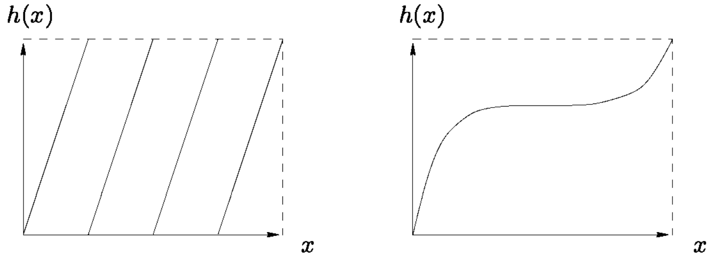


Figure 2.11: *Non monotonic hashing (left) and monotonic hashing (right).*

be emphasized that this enables the building and handling of dictionaries[5], but

[5]A set on which insertion, deletion and searching operations are defined.

does not allow the processing of proximity queries such as "given $h(x)$, report the neighbors of x from the values of h in a neighborhood of $h(x)$". We shall return to this issue in Section 2.6.

To define precisely a few standard hash functions, we start with the following definitions.

Definition 2.2 *Let $\mathcal{U}$ be a universe, i.e., a set of possible keys (0, 1, ...), let $\mathcal{S}$ be a subset of $\mathcal{U}$ of size n, let $\mathcal{T}$ be an array of buckets indexed by a set of integers $\mathcal{I}$. Suppose, in addition, that each bucket is endowed with an auxiliary data structure (linked list, array, BST, ...) that may contain up to b items. Then, a hash function is an application h from $\mathcal{U}$ to $\mathcal{I}$. Two keys x and y are said to* collide *if for $x \neq y$ we have $h(x) = h(y)$. A bucket is said to* overflow *if more than b keys have been hashed into it.*

As an example, consider Figure 2.12. Here, the set of all possible keys is the integer (in this example) range 1, ..., 50. The values to be hashed are five numbers $\mathcal{S} = (2, 5, 10, 15, 37)$. The hash table is an array of 10 linked lists (thus $b = \infty$). The hash function satisfies $h(10) = h(37) = 1$, $h(2) = 3$, $h(15) = 6$ and $h(5) = 8$. Typically, if one wants to know whether x is stored in the hash table, the algorithm

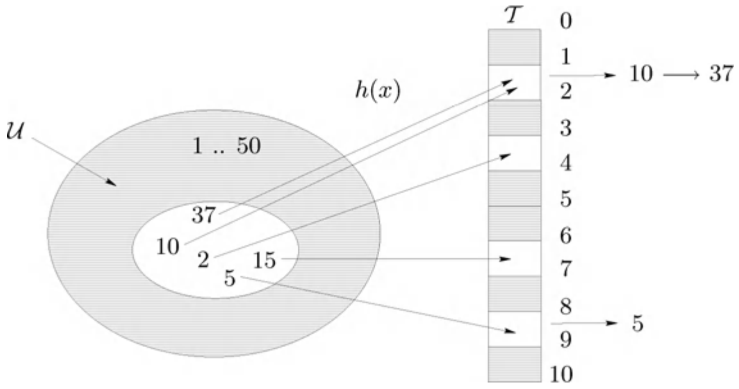


Figure 2.12: *Example of a hash table.*

consists of checking whether x is present in the data structure associated with the bucket of $\mathcal{T}$ indexed by $h(x)$. The important parameters are therefore the relative size of $\mathcal{S}$ as compared with that of $\mathcal{T}$ and the number of items referenced within each bucket. Several types of hash function can be envisaged:

- universal hashing, for which h is chosen randomly,
- perfect hashing, where h is injective,
- minimal hashing, where $card(\mathcal{S}) = card(\mathcal{T})$,
- dynamic hashing, where $card(\mathcal{S})$ is not known beforehand,
- monotonic hashing, where h keeps the ordering on the keys.

Two main problems are to be considered when implementing a hash technique. The first is related to the choice of the hash function (see above). The second, related to this choice, deals with collisions handling.

The collisions handling is indeed very important. One can of course rehash the whole table if some buckets are overflowing, but that does not tell us how to choose the right hash function.

An initial strategy consists of setting $b = 1$, in which case a single key at most is stored per bucket. This peculiar hashing is called "open addressing". The hash function maps $\mathcal{U} \times \mathcal{I}$ to $\mathcal{I}$ and the sequence of buckets attached to the key x is $seq(x) = (h(x, 0), h(x, 1), ..., h(x, m-1))$. The function h should be chosen such that for any key x, $seq(x)$ is one of the $m!$ permutations of $0, 1, ..., m-1$ with equal probability. Several such functions are described in [Cormen *et al.* 1990]. Their design is mainly concerned with avoiding overly long common sub-sequences between $seq(x)$ and $seq(y)$.

The second strategy aims at avoiding collisions as much as possible. A fundamental notion in this context is that of universality:

Definition 2.3 *Given two integer ranges $\mathcal{U} = 0, ..., n-1$ and $\mathcal{I} = 0, ..., m-1$, with $n \geq m$, a family of hash functions $\mathcal{H}$ is called 2-universal if for any x_1 and x_2 of $\mathcal{U}$ such that $x_1 \neq x_2$ and h chosen at random in $\mathcal{H}$, the following holds:*

$$Pr(h(x_1) = h(x_2)) \leq \frac{1}{m}\,.$$

Interestingly, the condition:

$$x_1 \neq x_2,\ y_1 \neq y_2, \quad \text{AND} \quad Pr(h(x_1) = y_1, h(x_2) = y_2) = \frac{1}{m^2}$$

implies the previous condition and corresponds to a 2-universal hashing called *strong*. The idea is to have the images of two points behave as independent random variables.

Priority queues

Many situations require the records to be processed in order, but not necessarily in full order and/or not necessarily all at once. For instance, an algorithm may require the highest value to be processed, then more values to be collected, etc. The data structures supporting this kind of operation are called *priority queues* and can be viewed as generalizations of stacks or queues. More precisely, a priority queue is a data structure containing records with numerical keys (numerical values), the priorities, and supporting the following operations:

- the construction of a priority queue for a set of items,
- the insertion of a new value,
- the search for the maximum,
- the modification of the *priority* of an item,

- the deletion of an arbitrary specified item,
- the merging of two priority queues.

Interestingly enough, the red-black tree data structure turns out to be a simple yet efficient implementation for priority queues. From the discussion of Section 2.5 and according to Theorem 2.1, we know that the insertion, deletion and search operations have a $\mathcal{O}(log\, n)$ complexity. Moreover, the construction, the modification of the priority and the merge requests simply require a sequence of insertions and deletions.

More sophisticated implementations of priority queues provide better complexities for the construction, modification of priority and merge operations. In particular, a fashionable data structure for that purpose is the *heap* data structure, which is a complete binary tree with the property that the key in each node should be larger than (or equal to) the keys in its children (if any). The reader is referred to [Sedgewick-1988] and [Cormen *et al.* 1990] for further reading.

Exercise 2.16 *Show how to implement a priority queue using an ordered linked list. What are the complexities of the "merge" and "change priority" operations?*

Exercise 2.17 *Suppose a red-black tree is endowed with an additional pointer to the maximum (or minimum) entry stored. Give the complexity of the "find the Max" ("find the min"), "insert" and "delete" operations.*

2.6 Two and three-dimensional data structures

After one-dimensional data structures, we now turn to multi-dimensional data structures. Here, we find grids and trees (quadtrees and octrees). Described in this section as data structures, these structures will be viewed again (Chapters 5 and 8) as they also help meshing algorithms.

In this section, we show how several ideas developed for one-dimensional data structures can be reused in two and three dimensions to handle more complex objects such as points, polygons, etc.

Grid-based data structures

Grid-based data structures are the two and three-dimensional equivalent of the one-dimensional bucket-like data structure. Notably, their performances vary greatly depending on the kind of items processed. If points are stored in two- or three-dimensional grids, very precise theoretical results have been known for a while, see [Devroye-1986] for example, that give indications about the behavior of the operations involved with such grids. However, when more complex objects are involved, such as polygons, many theoretical results remain to be established and justified experimentally. The aim of this section is precisely to present guidelines for the efficient use of grid-like partitions for applications such as mesh generation. A good starting point for further reading is [Cazals, Puech-1997].

Let $\Delta = \Delta_x \times \Delta_y \times \Delta_z = [x_{min}, x_{max}] \times [y_{min}, y_{max}] \times [z_{min}, z_{max}]$ be some three-dimensional domain containing a set $\mathcal{E}$ of n objects, which may be polygons, etc. Suppose we also want to answer proximity requests over the items of $\mathcal{E}$. For example, we want to find the closest polygon to a given point or find the pairs of intersecting polygons.

Similar to the bucket sort for the one-dimensional case, a clever way to address this class of problems consists of subdividing the domain of interest Δ into $n_x n_y n_z$ axis aligned boxes (voxels) of size $\delta_x \delta_y \delta_z$ with $\delta_x = \frac{\Delta_x}{n_x}$ (and similar values for the other dimensions) and having each voxel reference the items intersecting it. Once this pre-processing step has been done, the voxel containing a point $P(x, y, z)$ is identified by the triple ind_x, ind_y and ind_z with, for instance, $ind_x = \frac{x - x_{min}}{\delta_x}$. From this voxel, the items of $\mathcal{E}$ close to P are easily retrieved. In order for this construction to be efficient, two types of constraints must be taken into account. First, each voxel should reference a small number of items of $\mathcal{E}$. Then, the memory requirements should be affordable, i.e., $n_x n_y n_z = \mathcal{O}(n)$ with a small constant. We discuss here three grid-based data structures which achieve these goals for different input datasets, see Figure 2.13.

A grid is in fact a structure that can be uniform, recursive or related to a hierarchy of uniform grids.

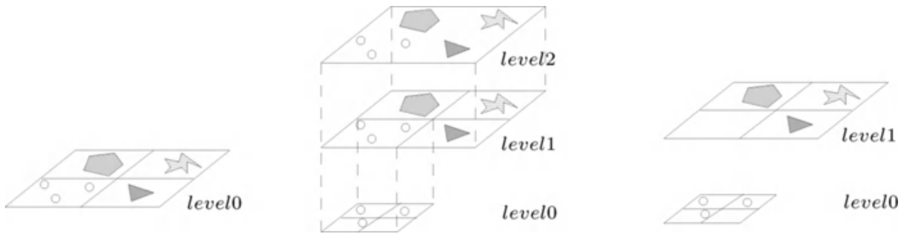


Figure 2.13: *The three types of grid-based structures. Left, a uniform grid, middle, a recursive grid and right, a hierarchy of uniform grids.*

Uniform grid. A uniform grid for the set $\mathcal{E}$ is a partition of its bounding box into $n_x n_y n_z$ subdivisions of equal lengths along the x, y and z axes. To get a memory requirement that is linear in the number of objects, the n_i are usually taken so that $n_i = \alpha_i \sqrt[3]{n}$. The α_i are positive constants and the simplest choice is $\alpha_x = \alpha_y = \alpha_z = 1$. Other choices may be preferred, for instance, related to heterogeneous values based on the ratios of the dimensions of the box, the amount of memory available, etc.

Uniform grids provide a simple yet efficient way of handling uniform distributions. But their performances drop catastrophically for more structured datasets, so other solutions have to be found.

Recursive grid. Once a uniform grid has been built for a set $\mathcal{E}$, one may find that some voxels are too heavily populated (they record too many items). If *maxi*, standing for the maximum number of items, is some positive integer (e.g. 50), a *recursive grid* for $\mathcal{E}$ is a hierarchical structure based on uniform grids such that whenever a voxel contains N items, $N > maxi$ (the initial voxel being the bounding box) it is recursively split into a uniform grid of approximatively N voxels.

Especially if no memory limitation is set, recursive grids provide the simplest and fastest implementation for handling many proximity problems. For unevenly distributed inputs, the performance gain over uniform grids can be up to several orders of magnitude.

Hierarchy of uniform grids. A weakness of the previous construction is that the recursion may waste a lot of empty voxels (Figure 2.13, middle). To solve this problem, the blind recursion can be replaced by a process that figures out more cleverly which are the dense areas that should be allocated resources and which are the empty areas. The strategy proposed in [Cazals, Puech-1997] successively separates the objects according to their size (filtering step), finds subsets of neighbors called clusters within these classes (clustering step), stores each cluster in a uniform grid and finally builds a hierarchy of uniform grids (Figure 2.13, right). For a description of the filtering and clustering steps, the reader is referred to the previous reference.

Nevertheless, the hierarchy of uniform grids offers a flexible and efficient data structure for input datasets with strong coherence properties. This is especially true since its construction overheads are almost the same as those of a recursive grid.

Quadtrees and octrees

A quadtree is a two-dimensional spatial data structure whose counterpart is the octree in three dimensions. The term quadtree refers to a class of hierarchical data structures with the common property of recursively decomposing a spatial region. Very much like grid-based subdivisions, quadtrees can be used to store a variety of inputs (points, line-segments, polygons, etc.) in any dimension (to simplify, the term quadtree will be employed whatever the dimension). And also similarly to grids, very little is known about the theoretical properties of this kind of structure, other than for points.

The basic idea consists of splitting the region processed into two sub-regions along each axis. In dimension d, a region is therefore split into 2^d children. The way the splitting hyper-plane (a line in two dimensions, a plane in three dimensions) is chosen, together with the recursion termination condition determines the quadtree type. We present below the two basic schemes when the input is a set of n points, see [Samet-1990] for more details.

Point quadtrees. The point quadtree can be seen as a generalization of a binary search tree. In two dimensions for instance, (Figure 2.14), each point is stored in a node and serves as the pivot in the subdivision of the associated region into four quadrants. The quadrants are numbered from left to right and from top to bottom (see the figure). As will be shown, locating a point in a quadtree is an easy matter that requires comparing its coordinates to those of the nodes currently traversed in order to decide in which quadrant it belongs.

The optimal strategy for building a point quadtree depends on whether or not the input dataset is known *a priori*. If it is, choosing at each step the median point along one axis (which luckily may also be the median of the other axis) results in a tree of depth between $log_4 n$ and $log_2 n$. If no *a priori* information is known, the points have to be sequentially inserted and the weakness already mentioned for BST trees may arise, i.e., the tree may be very elongated. Finally, similarly to BST trees, point quadtrees can be made dynamic, that is to support deletions. Nevertheless, this operation is rather tricky, see [Samet-1990].

Overall, point quadtrees are an interesting and versatile data structure which however suffers from several drawbacks. First, the higher the dimension, the higher the number of empty null pointers created . Then, its depth is usually larger than that of grid-based structures. Its use is therefore recommended when recursive grids or a hierarchy of uniform grids are too demanding resource-wise and/or when a dynamic feature for the process is desirable.

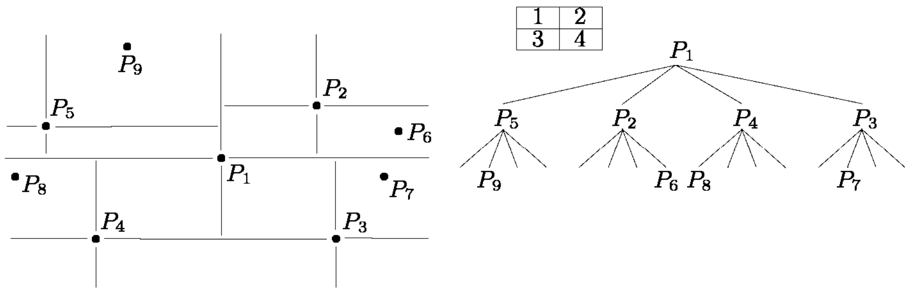


Figure 2.14: *Point quadtree based on a set of points in* $\mathbb{R}^2$.

Point-Region quadtrees. If one requires the four regions (the quadrants in two dimensions) attached to a node to have the same size, the data structure obtained is called a Point-Region quadtree (*PR-quadtree*). The recursion stops whenever a quadrant contains at most one data point, so that every point may be stored in a leaf node. As an illustration, consider Figure 2.15 which represents the PR-quadtree for the set of points of Figure 2.14. Before going further, notice that this type of quadtree is the one that will be commonly used in this context (see Chapter 5).

Locating a point in a PR-quadtree requires finding the quadrant containing it, which is similar, although certainly simpler here, to the previous quadtree case.

Inserting a point starts with a quadrant location. If the quadrant found is empty, simply insert the point and it is finished. Otherwise, it is refined into four regions (children) and if the other point does not belong to the same quadrant, insert the two points and it is finished. Otherwise both points are recursively inserted into the quadrants. It should be clear that many splits may be necessary to separate points located very close to one to another. More precisely, the depth of the tree may be as much as $log_2(\frac{L}{l})$, where L and l are the distances between the closest and farthest pair, respectively. In other words, the depth of the *PR*-quadtree depends upon the values manipulated (see also Section 2.4). In order to avoid wasteful memory allocation, a conservative approach consists of splitting a node only if its depth in the tree is not more than a certain threshold. Alternatively, the leaves can be allowed to accommodate up to *maxi* points, with *maxi* a small integer. Deletion of a node is considerably simpler than for point quadtrees because there is no need to rearrange the tree as all values are stored in the leaf nodes. However, the deletion of a node having exactly one brother should be followed by a step to collapse the four leaves.

In conclusion, *PR*-quadtrees offer an interesting alternative to usual point quadtrees especially because it is easier to obtain a dynamical aspect. However, one has to be careful about the height of the tree which may become very great if two points happen to be very close to each other. Hence, grid-based data structures are usually (much) more efficient because of their greatly reduced depth.

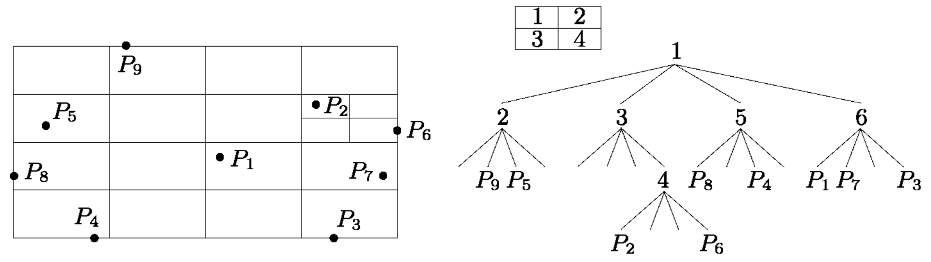


Figure 2.15: *The equivalent Point-Region quadtree data structure.*

Filters and side-effects

Numerous operations on meshes and triangulated surfaces require *a priori* running more or less tricky and expensive algorithms. For example, in order to compute the intersection between a line and a discretized surface (a triangulation), we have to compute pairwise intersections between the line and each triangle. Similarly, to render and plot such a (meshed) surface on a graphical device using the ray tracing method[6], we have to intersect lines (in $\mathbb{R}^3$) with the surface. Because this could be very expensive, as with any algorithm involving two geometric entities, a conservative approach consists of first making sure some necessary conditions are satisfied before running the computation. For example, for a ray intersecting a polygon, the intersection point P is necessarily contained in the bounding box of the polygon. If this is so, it must be further checked whether P lies within the

[6]Very briefly, the ray-tracing method consists of figuring out, for a given scene and viewpoint, the color of each pixel in the image representing this scene is rendered into has to be painted with. For a given pixel, this is done by casting a ray onto the scene from the viewpoint that intersects the pixel. The pixel color is deduced from the objects hit by the ray together with the contributions of the reflected and refracted rays.

polygon and this requires a linear time in the number of vertices of the polygon, otherwise, any further check is unnecessary, see [O'Rourke-1994].

Suppose now that the polygons are accessed through a grid-like data structure. In order to lessen the number of entities referenced by each voxel and thus the number of polygons tested (for intersection for a given ray), one is tempted to reduce the voxel sizes (thus increasing the total size of the grid in memory). By doing so, one also increases the probability of the ray belonging to the rectangle bounding box. Put differently, reducing the number of intersections results in more expensive computations because the coarse filter given by the bounding box becomes less efficient. This kind of side effect should be borne in mind when performing fine tuning of an algorithm.

2.7 Topological data structures

For the sake of simplicity, we restrict ourselves here to the two-dimensional case and we only consider triangulations. Not surprisingly, the central building block to describe such meshes is the triangle, together with a couple of data structures encoding the adjacency relationships. For surface meshes, a more general structure is necessary as a triangle could share an edge with more than one other triangle.

Triangulations (meshes) can be represented in many different ways. First, we present a representation based essentially on the triangles themselves. Then, we briefly discuss another representation based upon the triangle edges.

A triangle-based representation

The triangles are indexed from 1 on. A triangle is an (oriented) triple of vertices (cf. Chapter 1). With each triangle are associated its (at most) three edge neighbors (two dimensional case). The neighborhood relationships are encoded such that

$$k = Neigh(j, i)$$

which means that the triangle of index k is adjacent to the triangle of index i and that the edge j of triangle i is the common edge ($k = 0$ means that the edge j of triangle i is a boundary edge). Suppose also that vertex j of triangle i is opposite to edge j of this triangle (see also Chapter 1).

The pair of triples *vertices-neighbors* is one possible way of representing a triangulation (and probably the most concise one) also called the *adjacency graph* of the triangulation.

A richer representation is based upon the triples of vertices and the connection matrices. Each triangle K is endowed with a 3×3 matrix defined as follows (Figure 2.16):

- the diagonal coefficient i, c_{ii} indicates the local index of the triangle vertex opposite to triangle K by the edge i,
- another coefficient c_{ij} gives the index of the jth vertex of K in its ith neighbor.

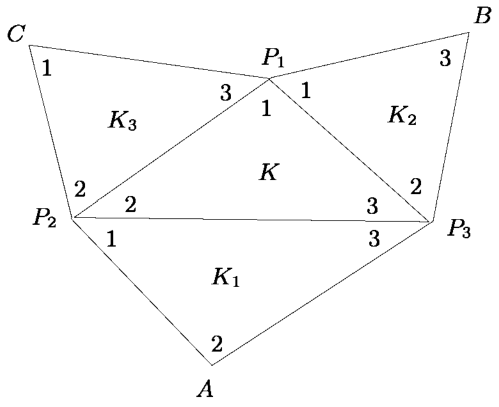


Figure 2.16: *Definition of a triangle K and of its three neighbors K_i. One can see the global indices of the vertices (P_i,A, ...) and the local indices (in any triangle) of these points (*1, 2 *and* 3*).*

According to its definition, the adjacency matrix of triangle K in Figure 2.16 is:

$$C^K = \begin{pmatrix} 2 & 1 & 3 \\ 1 & 3 & 2 \\ 3 & 2 & 1 \end{pmatrix}.$$

Hence, for the first line, point A of K_1 sees K and $c_{11} = 2$ because A is the second vertex of K_1. Similarly, $c_{12} = 1$ because the index of P_2 in K_1 is 1 and finally, $c_{13} = 3$. Notice that, unlike the depicted example, the matrix has no specific property (symmetry, for instance).

This representation is richer than the previous one and makes access to neighborhood items easier, as vertex information is added to element information.

Remark 2.5 *One can notice that the diagonal of this adjacency matrix allows the whole set of coefficients to be reconstructed.*

A question arises at this point: "Which data structure should table $Neigh$ be implemented with?" As pointed out in Section 2.7, if the number of vertices of the triangulation is known, so is the number of triangles (an upper bound). In this case, an array can be allocated beforehand, although the triangulation algorithm creates intermediate (transient) triangles. Another solution consists of using a dynamic hash table.

Remark 2.6 *To decide which is the best structure, we face a recurrent problem. Shall we use a "rich" and "complex" structure, which is more memory consuming but provides more information at once or a "poor" structure which is less expensive but gives less directly accessible information. The answer is strongly related to the memory available, to the cost of retrieving stored information and also to the cost of updating the structure.*

Winged-edge data structure

Another way to represent a mesh consists of viewing it from the point of its edges. This solution, as opposed to the previous one (for which a constant number of neighbors per face is assumed), allows edges common to more than two faces to be dealt with as well as with faces having an arbitrary number of vertices. We can find here two alternatives, the winged-edge (see [Baumgart-1974], [Baumgart-1975], [Weiler-1985]) and the half-edge (see [Mäntylä-1988], [Kettner-1998]) data structures which are edge-based structures. This kind of structure, also described in [Knuth-1975], essentially allows the following operations:

- walk around the edges of a given face,
- access a face from the adjacent one when given a common edge,
- visit all edges adjacent to a vertex.

Hierarchical representation

A more general description, useful for triangulations as well as for arbitrary meshes (manifold or non-manifold, conforming or not) is based upon the exhaustive enumeration of the relationships between the mesh entities.

Schematically, this description indicates the hierarchy between the entities according to their dimensions (points, edges, faces, elements). Hence, we have a direct link such as:

$$Points \longrightarrow Edges \longrightarrow Faces \longrightarrow Elements\,,$$

and the reverse link

$$Elements \longrightarrow Faces \longrightarrow Edges \longrightarrow Points\,.$$

This type of storage, [Beall, Shephard-1997], offers numerous advantages, although it is rather memory consuming and expensive when updating the structure. It provides a direct access to the entities of a higher (resp. smaller) dimension.

For a dynamic type of situation (such as graphical visualization, for instance), this kind of structure is especially appealing.

Other representations

Some peculiar applications can benefit from a very specific organization of the data. For instance, for a two-dimensional triangulation, if each vertex is endowed with the oriented list of the neighboring vertices (sharing an edge), this structure allows the related triangles to be easily retrieved, [Rivara-1986].

Other representations can also be found, for instance, the *STL* format which consists of enumerating all faces (elements) using the vertex coordinates (necessarily duplicating this information).

To conclude, one can notice that interchange formats (*STEP, IGES, SET, VDI, CGM, etc.*), although not directly aimed at meshing structures, can nevertheless give some information on how to design data structures for meshes. The data structure described in Chapter 1 is the one we advocate for meshing structure. It has been already adopted by several groups and seems well suited for many types of applications.

2.8 Robustness

Non-robustness refers to two notions. First, the result may not be correct (for instance, the convex hull of a set of points is not convex). Then, the program stops during the execution with an error or in a more catastrophic way (the program fails) with an overflow, an underflow, a division by zero, an infinite loop, etc. If the algorithm is reputed to be error-free (from a mathematical point of view), this means that its implementation leads to an erroneous behavior. Anyone who has implemented a geometric algorithm is likely to have faced this type of problem at some point.

This section has several aims. First, we give a very brief overview of the potential reasons why numeric problems arise; we recall how real numbers are encoded on most computers, and explain why the issues are even more difficult in a geometric context. We then provide some guidelines to reduce these risks, and finally we give an overview of the state-of-the-art techniques used to make floating-point operations robust.

Robustness issues

Numerical issues in scientific computing have been known since the early days of computers. The core of the problem lies in the limited resources used to encode numbers (the bits) and the drawbacks are twofold. First, the biggest and smallest numbers that can be represented are upper and lower bounded, so that some calculations cannot be carried out if the intermediate value exceeds these bounds. Second, real numbers have to be represented approximatively since one cannot squeeze infinitely many of them into a finite number of bits. These difficulties can yield to erroneous results and also generate undefined operations such as $\sqrt{x}$ with $x < 0$ or $x \times y$ with $x = 0$ and $y = \infty$. The unpredictability of floating point operations across different platforms led in the eighties to the adoption of the IEEE-754 standard [Goldberg-1991], [Kahan-1996]. In addition to the previously mentioned exceptions, this standard also defines several floating-point storage formats and templates for the $+, -, \times, \div$ and $\sqrt{.}$ algorithms, that is, any implementation of an operation should produce the same result as the operation provided by the standard. Note that this makes calculations consistent across different architectures, but does not eradicate exceptions at all.

More practically, the way floating-point numbers are represented on most modern processors is in the form $mantissa \times 2^{exponent}$ where the mantissa is represented by a p bits binary number. For example, $p = 24$ (resp. $p = 53$) for simple (resp. double) precision in the IEEE-754 standard. Rephrasing the issues raised above, how should one proceed when calculations produce numbers that cannot be represented using that many bits? If one does not care so much about exactness, the standard format specifies how such results have to be rounded to fit back into the finite representation. If one does care about exactness, one needs to switch to another representation.

One solution is the *multiple-digit* format based on a sequence of digits and a single exponent. Another solution is the *multiple-term* format where a number is expressed as a sum of ordinary floating point words. This latter approach has the advantage that the result of an addition such as $2^{40} + 2^{-40}$ is encoded in two words of memory while the multiple-digit solution requires 81 bits and incurs a corresponding speed penalty when further processed.

Computational geometry

From the computational point of view, geometric computations are even more difficult to handle than pure numerical calculations. A simple remark makes this point clear. Consider for instance, the computation of the solution of a general matrix system by any suitable method. A solution close to the exact one, within a given precision, is usually obtained, except in some pathological cases. Let us now consider two examples of geometric calculations.

The first example concerns the problem of computing the convex hull of a set of points $\mathcal{S} = (P_1, ..., P_n)$ in the plane.

Clearly, two points P_i and P_j contribute to the convex hull if all the remaining points lie on the same side of the line passing through P_i and P_j. So that the only *predicate* one needs for the computation is the so-called orientation test that, for three points A, B and C, indicates whether C lies on one side or the other of the line passing through A and B (Figure 2.17).

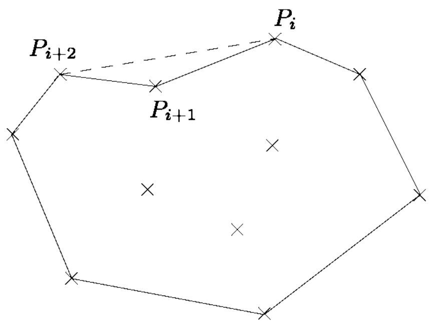


Figure 2.17: *A convex hull that is non convex !*

Using this predicate, the output of any convex hull algorithm consists of a combinatorial structure over $\mathcal{S}$, namely the list of vertices defining the convex hull, together with a consistency condition stating that the corresponding polygon is convex. But should a predicate calculation yield an erroneous result, the result computed may not be convex (as on the figure where P_{i+1} was reported to lie on the right side of P_iP_{i+2}). The result may be 100 % erroneous with respect to the goal aimed at, but we could possibly consider that this result is close enough to the theoretical result, if the error is small, as in the previous example, a correct answer about a few percent.

The second example corresponds to a more critical situation. To construct a Delaunay triangulation using an incremental method, one has to determine whether a given point belongs to the circumscribed disk (in two dimensions) of an element. The predicate used in this case is the *inCircle* predicate described above. A slight error, however minor, in the answer can lead to a triangulation that may be correct although it does not satisfy the Delaunay criterion or to overlapping elements. With respect to the goal expected, one can get an approximate answer within a few percent (the result is valid but the triangulation is not Delaunay) or a totally wrong answer (no correct triangulation at all).

Here, we have emphasized two different cases, one where a result is obtained and the other one for which the errors are so great that no result at all can be obtained. Notice also that such errors can lead to a "fatal" error, a program failure.

The algebraic degree of a problem and predicates

In spite of this apparent difficulty, geometric computations have the nice property of requiring numeric calculations which mostly consist of evaluating algebraic expressions. Examples are distance computations, the orientation predicate already mentioned (*Orientation*), as well as the *inCircle* predicate which indicates whether a point D belongs to the disk whose boundary is the circle passing through three points A, B and C or is external to it.

The common feature between these calculations can be formalized as follows: define an elementary predicate as the sign $(-, 0, +)$ of some homogenous polynomial over the input variables and its degree as the maximum degree of the irreducible factors. Also define the degree of an algorithm as the maximum degree of its predicates. Similarly, the *algebraic degree* of a problem is defined as the minimum degree of any algorithm solving it. For example, a convex hull algorithm using only the orientation predicate has degree 2. But a Delaunay triangulation using only the *inCircle* predicate has degree 4.

We now introduce the *Orientation* and *inCircle* predicates.

$$Orientation(A,B,C) = \begin{vmatrix} A_x & A_y & 1 \\ B_x & B_y & 1 \\ C_x & C_y & 1 \end{vmatrix}, \tag{2.1}$$

where A_x denotes the x coordinate of point A. Here we analyze C with respect to the line passing through A and B.

$$inCircle(A,B,C,D) = \begin{vmatrix} A_x & A_y & A_x^2 + A_y^2 & 1 \\ B_x & B_y & B_x^2 + B_y^2 & 1 \\ C_x & C_y & C_x^2 + C_y^2 & 1 \\ D_x & D_y & D_x^2 + D_y^2 & 1 \end{vmatrix}. \tag{2.2}$$

We analyze the position of point D with respect to the circle (disk) passing through A, B and C.

Exercise 2.18 *What is the algebraic degree of bucket sort and quicksort algorithms described in Section 2.4?*

Robust and efficient floating-point geometric predicates

We are interested in evaluating the sign of the predicates. We will assume in the discussion that the sole operations required are the addition and the multiplication. As mentioned above, the main issue of floating-point calculations is related to rounding and we noticed that the arbitrary precision computations solved this difficulty. Unfortunately, the overhead makes these solutions impractical. In this context and especially since we are interested in the sign of the expressions rather than their value, it was observed that floating-point calculations were actually very often reliable, so all that was needed was a way to take care of the confusing situations. To that end, the following paradigms have been proposed.

- Interval analysis: the evaluation of an expression is replaced by that of guaranteed upper and lower bounds on its value. When the sign is needed, if the interval does not contain 0, one can conclude. Otherwise, the evaluation must be performed again with higher precision (or by any other method).
- Arithmetic filters: the evaluation of an expression is accompanied by that of the maximum absolute error.

Details can be found in [Shewchuk-1997b] and also in [Devillers-Preparata 1998], for instance. Several methods are available (or have been studied) and it seems likely that these methods will be further refined and, possibly, work their way into real-world applications.

Remark 2.7 *In mesh generation, the main idea is to make sure the algorithm will provide a valid result, not necessarily strictly conforming to the theory but usable (a non-convex convex hull is theoretically awkward although its use depends on the application envisaged).*

2.9 Optimality of an implementation

The design and the implementation of a computer program performing some task usually requires taking care of various aspects. What is needed is to design the algorithm, to implement and test it and, possibly, to profile it (speed and memory requirements). Optimizing a computer program consists of optimizing its running time and/or its memory requirements, or fine tuning its time-space trade-off. In this section, we briefly review some features any programmer should be familiar with when pursuing these goals (dealing with efficiency and robustness).

Memory handling

Broadly speaking, the memory handling of a computer by the operating system is usually divided into two categories, the *static* and the *dynamic* memories. Basically, the static memory is a chunk used in a stack-like manner to store the local variables and the parameters used in the user-defined procedures and functions. The dynamic memory is a pool the user can request slots from in a dynamic fashion.

In Section 2.2, we saw that allocating and de-allocating dynamic memory could be very costly. A good strategy sometimes consists of writing a special case dedicated memory handler using particular features of the requests processed. For example, if one knows beforehand how much space is going to be needed, a linked list or a stack may be better implemented by a static array.

Another problem is fragmentation. If too many small slots are requested and freed too often, the memory map may end up like a piece of gruyere cheese. In this case, although a significant amount of memory may be available overall, any request for a big chunk may fail since no such continuous block is available. Garbage collecting must then be used to rearrange the memory.

Finally, there is another problem worth mentioning. It is desirable to group into memory the data manipulated in a program so as to avoid, as far as possible, *cache* defaults which are extremely costly. This kind of problem is rarely mentioned. Cache memory is random access memory RAM that a computer microprocessor can access more quickly than it can access regular RAM. As the microprocessor processes data it looks first in the cache memory and if it finds the data there (from a previous reading of data), it does not have to do the more time consuming reading of data from larger memory. Therefore, a judicious data organization can save a lot of time in memory reading because of cache defaults.

Running time profiling

When trying to reduce the time required by a calculation, two questions have to be addressed:

- which are the most time consuming functions (procedures) of the program?
- can one significantly reduce the amount of time spent therein?

One way of knowing how much time is spent in the different steps of a computer program is to use a *profiler.* Most modern computers come with tools geared to this goal and the functionalities offered are twofold. First, the number of times a given block is called (typically a function) is reported. Second an estimate of the time spent within the block is given. This value is obtained either via a compiler directive or by sampling the program counter regularly. In the latter case, the desired value is obtained by calling (and loading) very often (usually several times per period) the internal clock, thus running the risk of distorting the measurement. Indeed, this estimate may not be very sound for functions whose unit call cost is much less than the sampling grain. Getting much better information can be done by running a system call when stepping in and out of a particular function to retrieve the system time, thus slowing down less dramatically the execution time and not altering the measurement as much.

Several strategies may be employed to optimize a piece of code. In practical terms, a set of simple rules can be followed. Before reviewing this briefly, we provide Table 2.1 that shows, with respect to the number of cycles, the total cost of the classical operations (at the turn of the century, for some machines).

-	M_1	M_2	M_3	M_4	M_5
$+, -, \times$	1	1	1	1	1
/	6	18	5	42	7
$\sqrt{\cdot}$	11	39	17	207	27
exp	58	163	97	685	14
arctan	66	120	52	242	27
log	69	158	152	361	22
sin, cos	89	521 - 532	74 - 77	293-307	33
arcsin,arccos	98	173 - 184	109 - 119	538-558	33 - 43
a^x	157	522	304	1,741	59
tan	168	563	174	298	27

Table 2.1: Number of elementary cycles for some classical operations on a range of computer architectures.

In this table, the computer architectures M_i are the following:

M_1 : *HP PA 7100*	M_2 : *Sun HyperSparc*
M_3 : *IBM power pc G5*	M_4 : *Apollo 68040*
M_5 : *Intel Core 2 Duo*	

The simple analysis[7] of Table 2.1 shows that some operations must be avoided as far as possible. To this end, one has to find another way of implementing the desired functionality while asking the question about the pertinence of such an operation (say, for instance, a distance calculation d, if d^2 makes it possible to decide unambiguously, thus allowing to avoid the extra $\sqrt{\cdot}$ call. Similarly, comparing triangle angles can be achieved by comparing their cosine values).

Following these remarks, we propose here some ideas to optimize a program. As will be seen later, this approach concerns high level as well as low level functionality, some of these operations being simply common sense:

- analyze the predicate likely to give the desired information. If several predicates can be used, pick the best one (in terms of its degree),
- if a predicate has a high degree, look for another formulation of the problem in which this predicate is no longer involved,
- examine the operations used and keep track of costly operations,

[7]The goal is not to compare such or such an architecture but to point out that significant differences exist between numerical operations.

- minimize the number of parameters of a function,
- avoid indirections as far as possible (pointers or, even worse, pointers to pointers),
- use arrays if possible (take care of multi indices arrays or matrices with more than 3 indices, for example),
- avoid small loops[8] (typically a loop $i = 1, 2$ is not legitimate) and in the case of nested loops, use the most judicious implementation,
- etc.

To conclude and without pursuing this discussion further, notice that optimizing a program can lead to a less elegant or less formal implementation, for example, when a recursive call is replaced by a loop.

2.10 Examples of generic algorithms

For the sake of simplicity, in this section we consider triangular meshes only, although most of the constructions described can be extended (more or less easily) to other kinds of meshes. The following examples are given to emphasize how to benefit from algorithms and data structures described in the previous sections when dealing with applications related to mesh generation.

Therefore, numerous examples are linked to frequently encountered operations in various tasks in mesh generation or mesh modification algorithms. The order in which examples are given is not strictly significant. Some of these examples are purely academic, others deal with more real-world applications.

Remark 2.8 *The following examples can be seen as a set of exercises. Starting from data assumed to be known beforehand and depending on the goal envisaged, the reader is welcomed, on the one hand, to examine the proposed solution and, on the other hand, to look for alternate solutions to the same problem.*

Enumerating the ball of a vertex (1)

Given a mesh and a vertex of this mesh, the *ball* of this vertex is the set of elements sharing the vertex. Here, we propose a method which, for any vertex of a mesh, provides the list of the elements in its ball. Our interest is motivated by the fact that vertex balls are commonly used in numerous parts of mesh generation or mesh optimization algorithms (see Chapter 18, for example). The proposed method works without the knowledge of the adjacency relationships between the elements.

Let ne be the number of triangles in the mesh and let $Tria(1:3, 1:ne)$ be the array that stores the vertex indices of the mesh elements. Let np be the number

[8] Theoretically, compilers should be able to perform this task in most cases.

of mesh vertices[9]. The array $Tab(1 : np)$ is initialized to the value -1. Now, in view of a further usage explained below, we fill the arrays Tab and $List$ (of length $3 \times ne$) as follows:

Algorithm 2.15 *Construction of the ball of the mesh vertices.*

```
Procedure PrepareBall(Tria)
$ij \leftarrow 0$
FOR $i = 1, ne$
   FOR $j = 1, 3$
      $s \leftarrow Tria(j, i)$
      $List(ij) \leftarrow Tab(s)$
      $Tab(s) \leftarrow ij$
      $ij \leftarrow ij + 1$
   END FOR $j$
END FOR $i$
```

It is now easy to obtain the indices of all the elements sharing a given vertex. Let P be the index of the considered vertex, then its ball is obtained as follows:

Algorithm 2.16 *Enumerating the ball of a vertex.*

```
Procedure BallPoint1(P)
$ij \leftarrow Tab(P)$
IF $ij \neq -1$ THEN
   $i = \frac{ij}{3} + 1$,
   (vertex $P$ is the vertex of element $i$),
   $j = ij - 3(i-1) + 1$,
   (vertex $P$ is the vertex of index $j$ in element $i$),
   $ij \leftarrow List(ij)$ and back to IF.
ElSE END.
```

On completion of this procedure, the different indices i obtained in the algorithm are the indices of the elements[10] in the ball of the point P used as entry point while for each triangle of index i, the index j gives the position of point P.

Note that the above method consists of two algorithms. The first one is a preparation step which constructs the relevant tables. Once that has been done, the second one can be used repeatedly to access the ball of any vertex in the mesh.

Enumerating the ball of a vertex (2)

Here, we consider a similar problem but now only one ball is of interest (i.e., we consider only one vertex P) and, in addition, we assume that the neighboring relationships are available (see below how to compute this information).

[9]The points are assumed to be sequentially numbered from 1 to np ,thus, in a connected way, if this last property is not satisfied, np must be the largest number (index) of a point.

[10]In what follows, depending on the context, we will not differentiate between the index of an entity and this entity itself. For instance, point P and point of index P must be considered as two possible expressions of the same notion. Similarly, element k is the element of index k.

Given a triangle, its three neighbors are given via a table $Neigh(1:3, 1:ne)$ (where ne is the number of triangles). Indeed, $k = Neigh(j,i)$ means that element k is adjacent to element i and edge j of element i is the shared edge (while $k = 0$ if edge j of element i is a boundary edge). Also we assume that vertex j of triangle i is opposite edge j of this triangle (see Chapter 1). Now, let k_0 be a triangle having a vertex P, the following algorithm computes the indices of the elements in the ball of P (we assume that P is not the index of a boundary vertex):

Algorithm 2.17 *Enumerating the ball of a vertex.*

```
Procedure BallPoint2(P)
$k \leftarrow k_0$, $ltab \leftarrow 0$,
REPEAT
    $ltab \leftarrow ltab + 1$,
    $tab(ltab) \leftarrow k$
    take $j$ the index of $P$ in triangle $k$,
    take $j_{next}$ the index following index $j$,
    $k \leftarrow Neigh(j_{next}, k)$,
WHILE $k \neq k_0$.
```

On completion, $ltab$ is the number of triangles in the ball of vertex P and the indices of the desired triangles are the k's in the array tab.

Exercise 2.19 *Examine the case where the vertex P in question is a boundary vertex. Modify the above scheme accordingly. (Hint: take care of the case where $Neigh(j,i) = 0$).*

Notice that the proposed scheme does not extend to solving the same problem when a tetrahedral mesh is considered, where a more subtle algorithm must be defined as it is not easy to turn around a vertex.

Exercise 2.20 *Construct the ball of a vertex using the adjacency matrices described in Section 2.7.*

Searching operations

The problem is to find the item (the box, the cell or again the element) of a structure (a grid, a quadtree or an arbitrary mesh) within which a given point falls. Such problems are so-called searching problems or localization problems or again point location problems and are fundamental for various mesh generation methods. Let x, y be the coordinates of the given point.

Searching in a grid. Using a grid (Section 2.6) is a source of simplification by many respects. First, the indices of a box containing a point can be computed trivially. Second, it is easy to have access to the neighborhood of a given box and to that of a given point.

Let Δ_x (resp. Δ_y) be the size of the grid box in direction x (resp. y), the grid being constructed (see above) with the point x_0, y_0 as left bottom corner. Then

$$ind_x = \left\lfloor \frac{x - x_0}{\Delta_x} \right\rfloor \quad \text{and} \quad ind_y = \left\lfloor \frac{y - y_0}{\Delta_y} \right\rfloor$$

are the two indices of the box containing the point. Actually, ind_x as well as ind_y are integer values while the point coordinates could be floating-point values. Depending on the information stored in the grid, these indices can be used for various purposes (for instance, to find a point close to the point considered, any point in the box being a candidate, or, in the case of an empty box, any point in a non-empty box found in a certain neighbourhood of the initial box).

Searching in a quadtree. The easiest way to locate the quadtree cell (a *PR*-quadtree here according to Section 2.6) containing a given point is to start from the root of the tree and to use the values of the coordinates of the center of this cell to determine which one of the four children contains the given point. The center of a cell is easily obtained based on the box indices, we have:

$$x_c = ind_x\,\Delta_x + \frac{\Delta_x}{2} \quad \text{and} \quad y_c = ind_y\,\Delta_y + \frac{\Delta_y}{2}\,,$$

where x_c, y_c are the coordinates of this centre. The process is then recursively performed until a leaf (a terminal node) is reached.

An alternative approach is based on the underlying binary encoding of the quadtree by which a cell can be defined by an index consisting of a series of 0 and 1. The root is the 0 cell while the four first children can be identified by the following indices (Chapter 5):

$$(00, 01, 11, 10)$$

where 00 is the bottom left cell, 01 is the cell on the right of the previous one, 11 is the top right cell and 10 is the cell on the left of the previous (it is also the cell on top of the 00 cell). Actually, adding 01 to an index enables us to go to the cell on the right while adding 10 at the current index leads to the cell top of the initial cell (at the lower level, this effect will be obtained when adding 0001 and 0010 respectively, and so on).

Thus, binary operations can be used to locate a given point when a suitable system of coordinates has been defined.

Searching in a mesh. In this case, we assume that we are given a triangular mesh $\mathcal{T}$ covering a convex (planar) domain (for the sake of simplicity we consider this simple case only) and we want to find which element in $\mathcal{T}$ contains point P.

Let K be a triangle in $\mathcal{T}$ and let V_1, V_2, V_3 be its three vertices whose coordinates are denoted by x_i and y_i, then the signed surface area of K is:

$$S_K = \frac{1}{2} \begin{vmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{vmatrix}\,. \tag{2.3}$$

Actually, we can define S_K as twice the above value so as to avoid a division (notice that, due to the numbering convention of Chapter 1, S_K is strictly positive if K is a valid element).

Let us define the virtual triangle K^j as the triangle K where the vertex V_j of K is replaced by the point P considered. Then, we can compute S_{K^j}, $(j = 1, 3)$ whose sign enables us to determine[11] where the point P is located with respect to the half-planes bounded by the lines supporting the three edges of K (note that 7 regions are defined in this way). According to the sign of S_{K^j}, we pass through the corresponding neighbor of K and we repeat this process until the three S_{K^j}'s are positive (assuming that P is distinct from all the mesh vertices) meaning that the visited triangle contains P.

Based on these observations, a searching algorithm is easy to design and implement. One has to select a triangle K_0 and then follow the above scheme.

Rapid searching procedure in an arbitrary mesh. The previous algorithm can be very time consuming if a large number of elements needs to be visited between triangle K_0, the initial guess, and the solution triangle. This could lead in fact to a large number of area computations. Therefore, this algorithm could be combined with a grid (or a tree-like structure). A grid, or a tree-like structure enclosing the mesh is constructed and, for each cell one mesh vertex contained in it, if any, is recorded. In our previous examples, we showed that it is easy to find the cell of a grid or tree containing the given point. Also, a mesh element is associated with every point recorded in the cells. Hence, we can associate the given point with a close mesh point in the same cell. Any element, K_0, having this mesh point as a vertex can be used as an initial guess for the above searching procedure. In this way, the number of visited triangles is reduced and the number of necessary computations is reduced as well.

Remark 2.9 *Note that the grid (the tree structure) could be defined in various ways depending on the nature of the dataset. In this respect, for a grid, the number of boxes (indeed the values Δ_x and Δ_y as introduced above) and thus the occupation of the boxes are parameters that strongly affect the efficiency of the whole process.*

Intersection of a line segment with the elements of a mesh. Intersection problems are important components for various meshing techniques. One such problem is the following: given a mesh and a line segment between any two mesh vertices, construct the list of elements that are intersected by this line segment. A modification of the above searching technique allows for this task.

Enumerating the set of edges in a mesh

In this section, we describe several examples of methods for creating the list of the edges in a mesh. Let na be the number of edges, which we will label from 1 to na in the process of building the lists. In general, na is not known beforehand,

[11] We meet again here the barycentric coordinates.

so that in practice, an upper bound *namax* for *na* is needed to allocate memory resources for the arrays used.

An elementary method. Let *ne* be the number of elements in the mesh and let $Tab(1:2, 1:namax)$ be the table used to store all the edges. The following procedure enables us to fill the table *Tab*:

Algorithm 2.18 *Enumerating the edges in a mesh.*

```
Procedure TableEdge1()
$na \leftarrow 0$,
FOR $i = 1, ne$
   FOR $j = 1, 3$
      let $e_1, e_2$ be the indices of the endpoints of edge $j$ of element $i$,
      $k \leftarrow 1$ and IF $found$ is a boolean, set $found = .FALSE.$,
      WHILE ( $found = .FALSE.$ AND $k < na + 1$)
         IF [ ($e_1 = Tab(1,k)$ AND $e_2 = Tab(2,k)$) OR
         ($e_2 = Tab(1,k)$ AND $e_1 = Tab(2,k)$) ], THEN $found = .TRUE.$
         ELSE $k \leftarrow k + 1$
         END IF
      END WHILE
      IF $found = .FALSE.$, the edge considered is a new one, THEN
         $na \leftarrow na + 1$ AND $TAB(1, na) \leftarrow e_1$, $TAB(2, na) \leftarrow e_2$,
      END IF
   END FOR $j$
END FOR $i$
```

On completion, *na* is the number of edges and *Tab* contains the mesh edges (in fact, the indices of the edge endpoints). Notice that the number of times the comparisons are performed in the inner loop is proportional to $ne \times na$. This method is very time consuming in terms of complexity; however, it can handle a non-manifold surface mesh in $\mathbb{R}^3$ (a mesh is said to be manifold if all of its edges are shared by exactly two triangles or belong to the boundary, see Chapter 1) without any changes.

Using an edge coloring scheme. In this example, we build the same table using a technique of edge coloring to ensure that every mesh edge is recorded only once. We assume that the mesh is manifold (if a surface mesh is considered) and that we can access the neighboring elements across each internal edge, i.e., we have constructed a list $Neigh(j, i)$ which is the index of the neighbor of element i on side j. (See the next section.) Let $ColorTab(1:3, 1:ne)$ be a table which stores a color value (0 or 1) for edge j of element i in $ColorTab(j, i)$. Then we can build $Tab(1:2, 1:na)$ as in the first example by:

Algorithm 2.19 *Edge coloring scheme.*

```
Procedure TableEdge2()
$na \leftarrow 0$,
FOR $i = 1, ne$
   FOR $j = 1, 3$
      let $e_1, e_2$ be the indices of the endpoints of edge $j$ in element $i$,
      IF $ColorTab(j,i) = 0$ THEN, $na \leftarrow na + 1$,
      set $Tab(1,na) = e_1$, $Tab(2,na) = e_2$ and $ColorTab(j,i) = 1$,
      $k = Neigh(j,i)$, let $j_k$ be the index of this edge in element $k$,
      set $ColorTab(j_k, k) = 1$, IF $k \neq 0$.
      ELSE, edge $j$ of element $i$ has already been visited.
      END IF
   END FOR $j$
END FOR $i$
```

On completion, na is the number of edges and Tab contains the edges. The outer pair of loops of this method and the first one are the same. But the inner loop of this method is only executed $3 \times ne$ times. We reduced the amount of computation by using the large temporary table $ColorTab$. Note that this algorithm, serving as an example of *static* coloring, requires the input of the neighborhood relationships between the elements so as to know, for a given element, its (one, two or) three neighbors.

Notice also that this algorithm does not extend to three dimensions as the coloring a vertex does not identify an edge (while a similar property holds for a face).

Using hashing. In the two previous examples, we constructed arrays for the edges of a graph which in fact consists of labeling each edge with a number. The arrays give a direct access to the endpoints of the edge from the edge number. But to determine the number of an edge from its endpoints, you have to search that table. In this example, we build a set of lists that provide the opposite access to edge data, i.e., we construct a list that allows a direct access to the number of an edge from the edge endpoints. This construction is an example of *hashing* (Section 2.5). It works even in the non-manifold case where an edge is shared by more than two elements.

Let ne be the number of elements, and e_1 and e_2 be the endpoints of an edge a. We assume that we have a table $Sum(1 : 2 * np)$ where np is the number of vertices in the mesh, a table $Link(1 : namax)$ with $namax > na$ the number of edges in the mesh and a table $Min(1 : namax)$.

We first give the construction, then we add some comments. The construction consists of (after initializing all the arrays to 0):

Algorithm 2.20 *Construction of the edges of a mesh.*

```
Procedure TableEdge3()
$na \leftarrow 0$
FOR $i = 1, ne$
   FOR $j = 1, 3$
      compute $s = e_1 + e_2$,
      IF $Sum(s) = 0$, $na \leftarrow na + 1$, $Sum(s) \leftarrow na$, $Min(na) \leftarrow \min(e_1, e_2)$,
      ELSE IF $Min(l) \neq min(e_1, e_2)$ with $l = Sum(s)$, THEN
         (A) IF $Link(l) = 0$ THEN
         $na \leftarrow na + 1$, $Link(l) \leftarrow na$ and $Min(na) \leftarrow min(e_1, e_2)$,
         ELSE, consider $m = Link(l)$,
         IF $Min(m) \neq min(e_1, e_2)$, THEN set $l = m$ and back to (A).
      END IF
   END FOR $j$
END FOR $i$
```

As a result, *na* is the number of edges in the triangulation.

More precisely, for each element edge, we compute an index s as the sum of its two endpoint numbers giving an entry point in the table *Sum*. A zero value for $Sum(s)$ means that the current edge must be considered as a new edge (thus, it could be stored or processed as desired). Otherwise, one or several edge(s) with the same sum index have already been encountered. Hence, we just have to check if any of these edges matches the current edge (thanks to *Link*). This list traversal is done until the current edge is found (thanks to *Min*). Actually, if it is found, we proceed to the next edge, if not, it is inserted at the next available entry in *Link*.

Based on this construction, the edges can be retrieved using the following procedure:

Algorithm 2.21 *Retrieving the mesh edges.*

```
Procedure RetrieveEdge()
$na \leftarrow 0$
FOR $s = 1, 2 \times np$
   IF $k = Sum(s) \neq 0$, we find an edge such that $e_1 + e_2 = s$ and
   $min(e_1, e_2) = Min(k)$ and, while scanning the array $Link$ we find
   all the edges having the same sum of indices $s$.
   Practically, these edges can be obtained as follows
      $na \leftarrow na + 1$, the pair $s - Min(k), Min(k)$ is the edge $na$,
      WHILE $l = Link(k) \neq 0$
      $na \leftarrow na + 1$,
      $k \leftarrow l$ and the pair $s - Min(l), Min(l)$ is the edge $na$.
      END WHILE
   END IF
END FOR s
```

Many variations of this example can be obtained by modifying the keys of the hashing (replacing the *Sum* and the *Min* by different encoding schemes) or by modifying the purpose of the algorithm. For instance, it is possible to obtain the list of the boundary edges. Note that different choices of hashing function lead to different numbers of collisions (the number of edges with the same key) which could dramatically affect the efficiency of the method.

Exercise 2.21 *Analyze how this technique could be used to improve the efficiency of the elementary method in the first example.*

About set membership

The question is here to decide (quickly) whether an edge is a member of a set of edges stored in an array.

Depending on how the edge table is constructed (see the examples discussing how to construct this table in the previous section), finding if a given edge is a member of this table can be efficiently solved provided a suitable data structure is used (conversely, a less suitable data structure leads to a time consuming method).

In practical terms, if a hashing technique has been used to establish the edge table (see the above procedure), checking whether an edge is a member of this table is easy. Let e_1, e_2 be the two indices of the edge, then:

Algorithm 2.22 *Check the existence of an edge* e_1, e_2 *in a mesh.*

```
Procedure ExistEdge($e_1,e_2$)
compute $s = e_1 + e_2$,
IF $k = Sum(s) \neq 0$, one or more edges
such that $e_1 + e_2 = s$ exist.
IF $Min(k) = min(e_1,e_2)$, then edge $e_1,e_2$ belongs to the table.
ElSE, scan the table $Link$ to find
   all edges having the same sum of indices $s$:
   WHILE $l = Link(k) \neq 0$, $k \leftarrow l$ and analyze $Min(k)$.
ELSE, the edge is not stored.
END IF
```

provides the correct answer. Notice that the efficiency to answer the question depends on the way the edge table has been constructed.

Constructing the neighborhood relationships

We consider again a triangular mesh and we would like to construct the neighborhood relationships from element to element. Let $Neigh(1:3, 1:ne)$ be this table, then $k = Neigh(j,i)$ means that element k is adjacent to element i and edge j of element i is the shared edge (while $k = 0$ if edge j of element i is a boundary edge).

To construct this table, a variation of the algorithm previously employed to construct the edge table can be used where additional information is associated with the edges at the time they are visited. In this respect, it is necessary, for a given edge, to know the element of which it is a member and to know its index in its element.

The first time an edge is visited, for element i and index j, these two values are stored. When, say for element I at index J, the edge is met again then, the two following relationships are completed:

$$Neigh(j,i) = I \quad \text{and} \quad Neigh(J,I) = i\,.$$

Exercise 2.22 *Discuss the non-manifold case (for a surface) where more than two triangles share a given edge.*

Static and dynamic coloring

In a previous procedure, we showed one application of static coloring applied to some items of a mesh with the purpose of deciding quickly whether such or such a situation occurs.

In this case, such a tool can be seen as a boolean operator where the status of an item is defined as $.TRUE.$ or $.FALSE.$ (or 0 or 1) depending on the situation. In some cases, a two-flag operator is inefficient and *dynamic* coloring can be used more effectively.

Let us give a very simple example. We would like to construct balls about each vertex using a technique like the second enumeration method for balls discussed earlier in this chapter. For this purpose, let $List_i$ be a list of the indices of elements in the ball around the vertex of index i, and let $ColorTab(1:ne)$ be a color table for the elements of the mesh. An initial solution (np being the number of internal vertices) can be as follows:

Algorithm 2.23 *Construction of the ball of points.*

```
Procedure BallPoint3()
FOR $j = 1, ne$
   $ColorTab(j) \leftarrow 0$
END FOR $j$
FOR $i = 1, np$
   define $List_i$ as the empty list
   find an element of index $k$ which belongs to the ball of vertex $i$
   store $k$ into $List_i$ and set $ColorTab(k) = 1$
   WHILE element $k$ has a neighbor of index $j$ that has the vertex $i$
   AND that $ColorTab(j) = 0$
      $k \leftarrow j$; $ColorTab(k) = 1$;
   END WHILE
   FOR $j = 1, ne$
      $ColorTab(j) \leftarrow 0$
   END FOR $j$
END FOR $i$
```

This method uses two colors which must be maintained, which in turn requires additional processing of $ColorTab$ to reset its entries input to zero. One way to avoid this would be to maintain a list of the element indices that were encountered for the current i value and use it to reinitialize the relevant $ColorTab$ entries to 0. Another way is to use *dynamic* coloring method using colouring values 1 through np as follows:

Algorithm 2.24 *Construction of the ball of vertices.*

```
Procedure BallPoint4()
FOR $j = 1, ne$
   $ColorTab(j) \leftarrow 0$
END FOR $j$
FOR $i = 1, np$
   define $List_i$ as the empty list
   find an element of index $k$ that belongs to the ball of vertex $i$
   store $k$ in $List_i$ and set $ColorTab(k) = i$
   WHILE element $k$ has a neighbor of index $j$ having the vertex $i$
   AND that $ColorTab(j) < i$
      $k \leftarrow j$; $ColorTab(k) \leftarrow i$;
   END WHILE
END FOR $i$
```

Here the vertex label i is used as a dynamic color code to simplify the management of the element status which is automatically updated without any explicit processing.

About the construction of a dichotomy

The dichotomy approach, illustrated by a small example here, is a useful approach to a variety of meshing problems. This example involves a partition of an interval $a < x < b$ which is a set of points x_i for $i = 1...N$ such that $a = x_1$, $b = x_N$ and $x_i < x_{i+1}$. Suppose that we are given a continuous function $f(x)$ defined on the interval $a < x < b$ and a tolerance value τ. We wish to construct a partition of this interval such that $|f(x_{i+1}) - f(x_i)| \leq \tau$ for each sub-interval.

The following method uses the dichotomy approach to build tables *Tab* and *Next* which store the desired partition.

Algorithm 2.25 *Construction of a dichotomy.*

```
Procedure IntervalDichotomy()
$Tab(1) \leftarrow a$ , $Tab(2) \leftarrow b$
$i \leftarrow 1$, $ltab \leftarrow 2$, $Next(1) \leftarrow 2$
REPEAT
   $x = Tab(i)$ ; $y = Tab(Next(i))$
   IF $|f(y) - f(x)| > \varepsilon$,
   THEN $ltab \leftarrow ltab + 1$, $Tab(ltab) \leftarrow \frac{x+y}{2}$
   $Next(ltab) \leftarrow Next(i)$, $Next(i) \leftarrow ltab$
   ELSE $i \leftarrow Next(i)$
UNTIL $i = 2$.
```

Note that this kind of algorithm has various applications and, in some sense, can emulate a recursive process, whereas this capability is not necessarily included in some programming languages.

★
★ ★

We have discussed a few examples of algorithms to illustrate how to use such or such basic structures or basic algorithms. Obviously, numerous other application examples can be found and, actually, meshing algorithms as well as mesh optimization algorithms or, in general, mesh manipulation algorithms or even mesh visualization algorithms can take advantage of using such or such basic ingredients in order to easily find or process the mesh entities that are involved in the whole procedure.

Chapter 3

A Comprehensive Survey of Mesh Generation Methods

Mesh generation has evolved rapidly over the last decades and meshing techniques seem to have reached a level of maturity that allows them to calculate complete solutions to complex three-dimensional problems. Typically, unstructured meshes for complex three-dimensional domains of arbitrary shape can be completed on current workstations in reasonable time. Further improvements may still be expected, for instance, regarding the robustness, reliability and optimality of the meshing techniques.

Early mesh generation methods employed meshes consisting of quadrilaterals in two dimensions or hexahedra in three dimensions. Each vertex of such meshes can be readily defined as an array of indices and these types of meshes are commonly referred to as *structured* meshes. By extension, any mesh having a high degree of ordering (for example, a Cartesian grid) is said to be structured. More recent developments have tried to cope with the complex geometries (for instance, in CAD models involving multiple bounding surfaces) that were difficult to handle (i.e., to mesh) with fully structured meshes. Nowadays unstructured meshes are commonly associated with finite element methods to provide an efficient alternative to structured meshes.

★
★ ★

The purpose of this chapter is to provide a comprehensive overview of the current techniques for both structured and unstructured mesh generation and to discuss their intrinsic advantages or weaknesses. These techniques will be further discussed in more detail in the relevant chapters of this book. First, a preliminary classification of existing meshing techniques is proposed. One section is dedicated to surface meshing as surfaces play an important role in unstructured mesh generation techniques. Finally, a brief outline of mesh adaptation approaches is given.

3.1 Classes of methods

Despite many conceptual differences (since mesh generation methods have been developed in different contexts and for different applications), the classification of these techniques into seven classes has been proposed, for instance, in [George-1991]. Although this classification reflects the main approaches published, it appears that several techniques can be gathered together due to their intrinsic properties, thus leading to a classification into only five categories.

Class 1. *Manual* or *semi-automatic* methods.

These are mainly applicable to geometrically simple domains. Enumerative methods (mesh entities are explicitly user-supplied) and explicit methods (which take advantage of the geometric features of the domain) are representative of this class.

Class 2. *Parameterization (mapping)* methods.

The final mesh is the result of the inverse transformation, or *mapping*, of a regular lattice of points in a parametric space to the physical space. Two main approaches belong to this class, depending on whether the mapping function is implicitly or explicitly defined:

- *algebraic interpolation* methods. The mesh is obtained using, in one popular example, a transfinite interpolation from boundary curves (surfaces) or other related techniques that are explicitly defined,
- *solution-based* methods. The mesh is generated based on the numerical solution of a partial differential system of equations (elliptic, hyperbolic or parabolic), thus relying on an analytically defined function.

Class 3. *Domain decomposition* methods.

The mesh is the result of a top-down analysis that consists of splitting the domain to be meshed into smaller domains that are geometrically close to a domain of reference (in terms of shape). Two main approaches have been proposed, the difference being the structured or unstructured nature of the mesh used to cover the small domains:

- *block decomposition* methods: the domain is decomposed into several simpler sub-domains (blocks), each of which is then covered with a structured mesh (obtained for instance, using a mapping technique, as seen above),
- *spatial decomposition* methods: the domain is approximated with a union of disjoint cells that are subdivided to cover a spatial region object, each cell then being further decomposed into mesh elements. Quadtree and octree-based techniques are representative of this class.

Class 4. *Point-insertion/element creation* methods.

These methods generally start from a discretization of the boundary of the domain (although this feature is not strictly required) and mainly consist of creating and inserting internal nodes (elements) in the domain. Advancing-front (element creation) and Delaunay-based (point insertion) approaches are two methods belonging to this class.

Class 5. *Constructive* methods.

The final mesh of the domain is the result of merging several meshes using topological or geometric transformations, each of these meshes being created by any of the previous methods.

Remark 3.1 *Needless to say, this classification is necessarily arbitrary. However, while not unique, it does account for the different approaches published. Other methods not included in this classification exist, which are designed to handle specific situations.*

A difficult task consists of clearly identifying the method capable of providing an adequate mesh, related to the field of application. Basically, the geometry of the domain and the physical problem direct the user towards one method or the other.

On the other hand, the emphasis can be put on the type of meshes created by any of the proposed methods. From this point of view, meshing techniques can be classified into two types, according to whether they lead to structured or unstructured meshes.

3.2 Structured mesh generators

In this section, we briefly describe the main approaches generally used to create structured meshes. While not claiming exhaustivity, the techniques mentioned here are representative of the current developments in this field.

The basic idea common to all structured mesh generation methods consists of meshing a canonical domain (i.e., a simple geometry) and mapping this mesh to a physical domain defined by its boundary discretization. Numerous types of such transformations exist and have been successfully applied to computational domains, for instance parametric space for surfaces (Bézier patches, B-splines), Lagrange or transfinite interpolation formula, quasi-conformal transformations, etc.

The first problem to solve is *where* to place the mesh vertices in such a way as to achieve a *natural* ordering appropriate to the problem considered. A trivial observation shows that simple domains such as squares and discs, in two dimensions, have an intrinsic curvilinear coordinate system. In this sense, the mapping techniques described below provide a basis for mesh generation.

Curvilinear coordinates. The physical domain discretization requires some level of organization to efficiently compute the solution of the PDEs. This organization is usually provided using a Cartesian or cylindrical coordinate system. More precisely, the grid points are defined using coordinate line intersections, which allow all numerical computations to be performed in a fixed (square or rectangular) grid. Hence, the Cartesian coordinates used to represent the PDEs have been replaced by the curvilinear coordinates.[1] A constant value of one curvilinear coordinate (and a monotonic variation of the other) in the physical space corresponds to vertical or horizontal lines in the *logical* space.[2]

Theoretically, two procedures can be used to generate a system of curvilinear coordinates, algebraic interpolation techniques [Gordon, Hall-1973] and solution-based techniques [Thompson-1982a]. From the computational point of view, the classical algebraic methods are usually faster than the differential equation methods.

3.2.1 Algebraic interpolation methods

A simple, though efficient, way to achieve a structured mesh is to use a sequence of mappings to reduce the possibly complex domain to simple generic shapes (e.g. a triangle, a quadrilateral, a hexahedron, etc.). After a structured mesh has been defined in the logical space, the mapping function is used to generate a mesh conforming to all domain boundaries. This technique has proved useful for two and three-dimensional domains as well.

The mapping function(s) and the mesh point distribution in the logical space can be chosen arbitrarily. However, it may be of some interest and it is sometimes more efficient to enforce the boundary discretization in the logical space to match the given domain boundary discretization.[3] The control of the mesh point distribution in the parametric space makes it possible to control the density of mesh vertices in the real domain, for instance, to obtain a finer mesh in regions of high curvature.

Remark 3.2 *In general, the domain discretization must be a convex*[4] *polygon (polyhedron, in three dimensions) to guarantee the validity and the conformity (Definition 1.7) of the resulting mesh.*

The problem of finding a proper mapping function is equivalent to finding a specific function of the curvilinear coordinates. This function contains coefficients that enable the function to match specific values of the Cartesian coordinates on the boundary (and possibly elsewhere). Algebraic grid generation is thoroughly discussed in [Shmit-1982] and [Eriksson-1982]. Figure 3.1 shows an example of an algebraic mesh generated by the method described in [Baker-1991b].

To emphasize the algebraic method feature, we simply mention one particular mapping function, the transfinite interpolation scheme. This approach was first investigated by [Gordon, Hall-1973] and, then, by [Eriksson-1983], among others. Its most significant feature is its ability to control the mesh point distribution and particularly the slope of the mesh lines meeting the boundary surfaces [Baker-1989a]. In this chapter, we do not pursue the notion of transfinite elements and refer the reader to Chapter 4. However, we describe its application to the mapping of a unit square, in two dimensions.

[1]The mapping of the physical space onto the *logical* space must be one-to-one.
[2]Also called transformed or parametric space.
[3]This feature makes it possible to conform exactly to the given domain boundaries.
[4]or at least close to a convex shape.

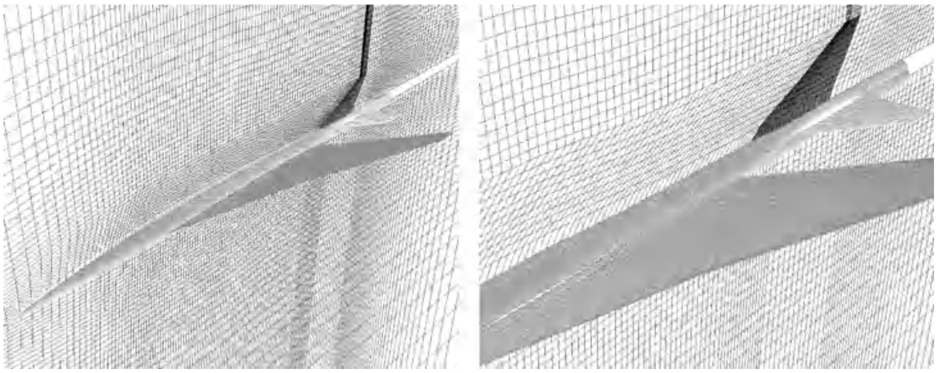


Figure 3.1: *Single block algebraic grid for a fuselage plus two lifting surfaces (data courtesy of T.J. Baker, Princeton University, NJ, USA).*

Unit square mapping by transfinite interpolation. Here, we are concerned with a continuous transformation which maps the unit square $(\xi, \eta) \in [0,1] \times [0,1]$ one-to-one onto a simply connected, bounded two-dimensional domain. The mapping can be seen as a topological distortion of the square into the domain. The problem is to construct the mapping function that matches the boundary of the domain and, more precisely, the boundary discretization of this domain.

Let $\phi_i(\xi, \eta)\,, i = 1,4$ be the parameterization of side i of the real domain, for which four such sides have been identified, and let a_i be the corresponding edge endpoints (corners). For the sake of simplicity, we have assumed that the discretizations of any two opposite edges of the domain have the same number of vertices[5]. A discretization of the unit square is constructed, which is analogous to that of the real domain, i.e., each side of the square conforms to the discretization of the corresponding real side, in terms of the relative distances between successive vertices. A quadrilateral mesh is then formed in the logical space by joining the matching points on opposite edges, the internal nodes thus being the line intersections.

The mapping function then takes the lattice of vertices in the parametric space (unit square) and maps it to the physical space (real domain) using a transfinite interpolation based on the Lagrange interpolation formula as follows:

$$\begin{aligned} F(\xi,\eta) &= (1-\eta)\phi_1(\xi) + \xi\phi_2(\eta) + \eta\phi_3(\xi) + (1-\xi)\phi_4(\eta) \\ &- ((1-\xi)(1-\eta)a_1 + \xi(1-\eta)a_2 + \xi\eta a_3 + (1-\xi)\eta a_4). \end{aligned} \tag{3.1}$$

Figure 3.2 shows a mesh of a domain mapped by applying a transfinite interpolation formula.

The same technique can be applied to map a right triangle onto a triangular-shaped domain and can be extended to three dimensions as well (cf. Chapter 4).

[5]Such a situation can always been obtain by adding more nodes along a boundary edge, if needed.

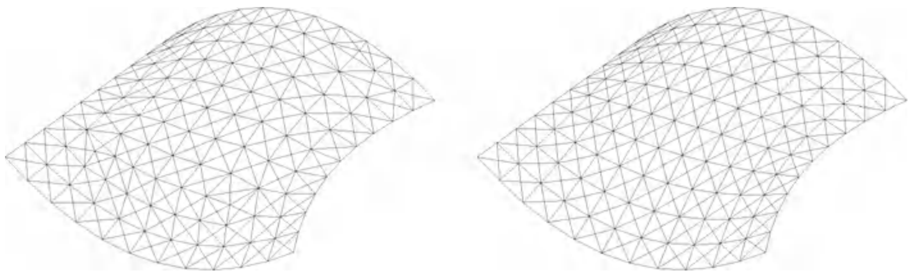


Figure 3.2: *Surface mesh obtained using a transfinite mapping of the unit square (right: with element shape control).*

Remark 3.3 (Sequential mappings) *Another algebraic method which is similar to the coordinate transformation method introduced above is based on the combination of a sequence of mappings. It becomes possible to reduce a complex domain to a simple generic shape by introducing several simple conformal mappings successively.*

Remark 3.4 (Blending approach) *The association of several meshes, each one being generated separately as a simple domain, to form a global smooth mesh using a weighted combination of functions is the attractive key feature of the blending grid technique. Although very promising, this approach is by no means easy to implement for arbitrary complex configurations. To some extent, this technique prefigures the multiblock approach.*

3.2.2 PDE-based methods

Since the problems to solve are usually systems of partial differential equations, it seems obvious to link the system of coordinates to the solution of a system of PDEs. If the coordinates vertices are specified on the boundary of the region, the equations must be elliptic, as they would be parabolic or hyperbolic if the specification concerned only part of the domain boundary. Hence, the important step of finding a mapping between a Cartesian and a boundary-fitted curvilinear coordinate system is to clearly identify the equations. The elliptical equation method is the most popular of this kind of technique (cf. [Eiseman, Erlebacher-1987] for a general survey of PDE-based methods).

Elliptic method. The main advantage of the elliptic equation method is that it preserves grid orthogonality in the vicinity of the boundary. Another property is the inherent smoothness over the entire domain that prevails in the solution of elliptic problems. Moreover, boundary discontinuities do not propagate far inside the domain. A drawback is that the coordinate system is the solution of a system of PDEs, thus resulting in more computing time than other methods of generation. This technique has provided some interesting results, especially in computational fluid dynamics (CFD) simulations for transonic flow over airfoils [Thompson-1982b], [Thompson-1987]. One of the most simple elliptic systems is the Laplace system defined as

$$\nabla^2 \xi^i = 0 \qquad i = 1,3 \tag{3.2}$$

which can be obtained from the Euler equations for the minimization of the integral

$$I = \iiint \sum_{i=1}^{3} |\nabla \xi^i|^2 dV$$

where the quantity $|\nabla \xi^i|$ represents the grid point density in a certain way along the coordinate line for a variation of ξ^i ($\xi^1 = \xi$, $\xi^2 = \eta$, ...). The smoothing effect of the Laplacian tends to closely or equally space the lines according to the boundary curvature.

Remark 3.5 *The strong smoothing effect of the Laplace transform may lead to an undesirable node point distribution. To overcome this problem, control functions can be introduced in Equation (3.2).*

Parabolic and hyperbolic methods. The mesh generation procedure can also be based on parabolic or hyperbolic PDEs. Equations of the parabolic method can be derived from the elliptical method by modifying the proper terms. The parabolic technique is useful to generate a mesh between two boundaries of a multi-connected domain, with two boundary specifications. The hyperbolic approach tolerates only one boundary specification and is mainly interesting for use in calculations over unbounded domains or for generating orthogonal meshes.

For instance, the solutions of the following set of equations

$$x_\xi x_\eta + y_\xi y_\eta = 0\,, \qquad x_\xi y_\eta - x_\eta y_\xi = V(\xi, \eta)\,,$$

where $x_\xi = \frac{\partial x}{\partial \xi}$, defines a hyperbolic system [Steger, Sorenson-1980], where the first equation corresponds to the orthogonality condition and the second equation defines the local cell area based on a specified distribution $V(\xi, \eta)$. The main known drawback of the hyperbolic-type approach is its inability to create meshes for internal flow problems.

3.2.3 Multiblock method

Wrapping a mesh around a complex domain boundary may be a tough and even intractable problem to solve. One way of getting around this difficulty is to consider a multiblock scheme. In this approach, the whole domain is decomposed into several simpler sub-domains (i.e., *blocks*), each of which is then covered automatically with a structured mesh, resulting, for instance, from an algebraic technique or a PDE methods. This feature makes the multiblock approach particularly interesting for parallel computations.

Remark 3.6 *In general, the resulting mesh is structured at the block level but no longer at the global domain level.*

Several possibilities of implementing the multiblock technique arise depending on which constraints are required between the blocks (for instance, what degree of continuity or conformity is desired).

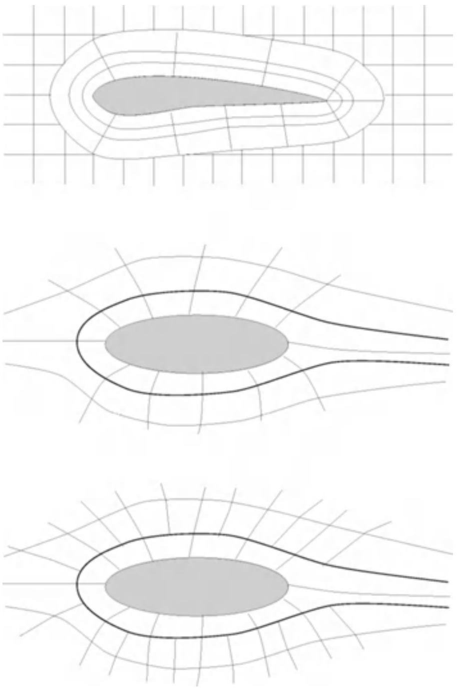


Figure 3.3: *Several implementations of the multiblock approach: overlapping (top), composite (center, mesh lines are continuous across the boundary) and patched (bottom, conforming surfaces of the boundaries, however discontinuous mesh lines).*

Overlapping. If no special attention is paid to the block interfaces, each block can be meshed separately for each component of the domain. The resulting mesh is a system of overlapping sub-meshes. Although the meshes are easy to generate, the main drawbacks of the technique are the transfer of information between neighboring meshes and the accuracy of the interpolation which can prevent the stability of the method.

Patched. An additional constraint to the overlapping multiblock technique requires the separate meshes to conform to the surfaces of their common boundaries, even if mesh lines are not continuous. In this patched approach, the interpolation procedure is easier than that required by the previous approach and mesh refinement can vary in specific regions without propagating elsewhere.

Composite. If mesh lines are required to be continuous across the boundaries, thus propagating mesh refinement throughout the entire domain, we obtain the composite multiblock method. This technique requires a global vertex numbering. Its main advantage is the improved accuracy that results.

General scheme. The composite multiblock decomposition method can be summarized as follows.

Step 1. Decompose the computational domain into simple blocks.

- Use a global vertex numbering.
- Define the block interfaces to ensure conformity.

Step 2. Discretize the block interfaces. The requirements are to:

- obtain good geometric approximation,
- ensure the mesh lines are continuous across the boundaries,
- ensure the adequacy of each block with respect to the local mesh generation process (for instance, with an algebraic method, the number of points on opposite edges must be identical).

Step 3. Mesh each block separately (create internal points).

Step 4. Construct the final mesh by merging the sub-meshes.

3.2.4 Product method (topology-based method)

Semi-automated procedures sometimes give additional meshing capabilities to the user. Actually, the mesh of a complex domain of cylindrical topology in d-dimensions can be easily obtained from a d-1-dimensional mesh of a section, the *source* (for instance, a cylinder can be defined using a circle and a direction in three dimensions). More precisely, a point leads to a series of segments and a segment results in a set of quadrilaterals. The efficiency of the method is related to the ability of the user to define the reference mesh.

In three dimensions, the purpose of the two-dimensional reference mesh is to provide a pattern from which to extrude the final mesh. The number of layers (slices) and their positions (i.e., the node locations along the reference line) can be supplied implicitly (the discretization of the line is given) or explicitly (using a stretching function for instance). The reference mesh is then extruded along the direction to create the desired number of three dimensional element layers between an upper and lower bound. Based on the type of two-dimensional element (triangles or quadrilaterals), the resulting quasi-regular mesh is defined from either wedge or hexahedral shaped elements apart for some peculiar configurations leading to degenerate elements (cf. Figure 3.4).

Remark 3.7 *In general, the final mesh is not structured as the reference mesh is not necessarily structured. However, the connectivity of the resulting mesh is derived from that of the reference mesh.*

The main drawback of this approach is the possibility of degeneracies (for instance, when part of the domain boundary is coincident with the axis).

General scheme. Schematically, the product technique reads:

Step 1. Identify and mesh the domain of reference (section).

Step 2. Extrude the reference mesh based on:

- the specified direction,
- the desired number of layers.

Step 3. Optimize the mesh.

Figure 3.4 illustrates the principle of a product method by depicting a source mesh composed of quads and the resulting hex mesh. For the sake of clarity, only one layer of elements, between two sections, is displayed.

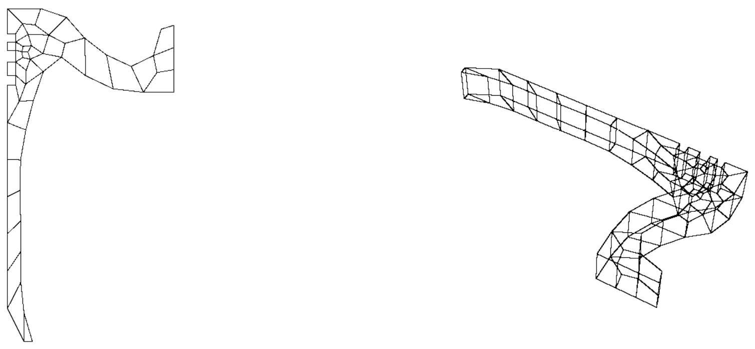


Figure 3.4: *A two-dimensional source mesh (section) and one layer offset in the resulting three-dimensional mesh.*

3.3 Unstructured mesh generators

In general, structured meshes for arbitrary complex geometries are difficult to obtain in a fully automatic manner. An alternative to a structured mesh consists of using simplices (triangles or tetrahedra). This feature gives the mesh maximum flexibility to address complex geometries as well as to control mesh point distribution. As previously mentioned, unstructured meshes are mostly composed of simplicial elements and the major automatic mesh generation methods produce such elements. However, there also exist some methods resulting in unstructured meshes consisting of quads or hex. Nevertheless, such methods are more tedious both to design and to implement.

In this short survey, we will focus mainly on simplicial methods while other methods will be discussed after.

Unstructured mesh generation is a task that may appear both easy and difficult at the same time. The first point is related to the fact that theoretical issues can be used to help the algorithm design for some approaches. The second point, *a contrario*, is that we don't have such results for some other methods. In addition, whatever the case, the mesh generation technique must be robust and reliable since complex geometries and delicate situations must be envisaged.

After these remarks, we now introduce the main approaches that enable us to create two and three dimensional unstructured meshes. To this end, we give the main characteristics of the various methods while the following chapters will give a detailed discussion.

The generation of unstructured meshes involves the creation of points and the relevant connectivities. This is usually achieved through different stages that can be summarized as follows.

Step 1. Definition of the domain boundaries.

Step 2. Specification of an element size distribution function.

Step 3. Generation of a mesh respecting the domain boundaries[6].

Step 4. Optimization of the element shapes (optional).

The boundary discretization (which represents a polygonal (polyhedral) approximation of the boundary of the real domain) can be achieved as a separate procedure or simultaneously with the creation of the mesh (in which case, the boundary integrity must be guaranteed by the mesh generation technique).

The element size distribution function can be defined in two ways, implicitly or explicitly. In the first case, either the size of an interior element is deduced from the boundary discretization by interpolation, or, if a control space is supplied, for which the element size is defined at each vertex, the value at any point can be computed by interpolation between vertices (cf. Chapter 1). On the other hand, a function $f(x, y, z)$, defined over the entire (three-dimensional) domain, can be either constructed analytically to define explicitly the element size distribution or user-supplied (e.g., academic test case).

In general, good-quality meshes cannot be obtained directly from the meshing techniques. Therefore, a post-processing step is required to optimize the mesh with respect to the element shapes. Regardless of the mesh generation method used, topological and geometrical mesh modification techniques are required to obtain a high-quality mesh suitable for finite element computations (cf. Chapters 17 and 18).

Three approaches can be identified in the context of automatic methods including the spatial decomposition based methods, the advancing-front and the Delaunay type methods. After a survey of these methods, we will turn to some other approaches.

[6]Or the boundary discretization.

3.3.1 Spatial decomposition methods

Spatial decomposition methods were applied to mesh generation purposes about three decades ago [Yerry, Shephard-1983]. In such approaches, the resulting hierarchical tree structure (quadtree and octree in two and three dimensions respectively) serves as a neighborhood space as well as a control space (cf. Chapter 1) used to prescribe the desired element sizes which are related to the cell diameter.

General principle. At first, the domain is enclosed in a bounding box (one cell, the root). The domain is then approximated with a union of disjoint, variably sized cells, representing a partition of the domain. The cells are recursively subdivided until each terminal cell is no larger than the desired element size (local value of the size distribution function). A covering up of a a spatial region enclosing the object (the bounding box) is then obtained. Each terminal cell is then further decomposed into simplices (triangles or tetrahedra), thus leading to a suitable finite element mesh of the domain. The stopping criterion can be based on the curvature of the model entity or supplied by an adaptive analysis error estimate.

This type of method is usually capable of proceeding either directly from a given discretization of the domain boundary or, more generally, by generating the boundary representation of the domain using simple queries to a geometric modeling system, a CAD system.

General scheme. Schematically, a classical quadtree/octree-based technique involves the following steps.

Step 1. Initializations.

- boundary discretization (or analytical description of the boundaries),
- definition of the size distribution function (if available).

Step 2. Tree decomposition.

- Initialization: the tree representation is derived from a box enclosing the domain,
- Recursive subdivision of the box up to a satisfactory criterion.

Step 3. Tree balancing: to limit the difference between neighboring cells to only one level (the so-called 2 : 1 rule).

Step 4. Cell meshing using predefined patterns (internal cells) and local connections (boundary cells).

Step 5. Optimization: topological and geometrical modifications.

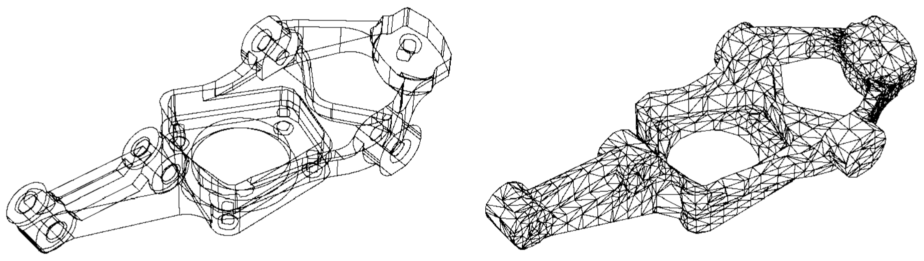


Figure 3.5: *Original CAD model (Patran geometric modeler) and octree-based mesh before optimization (data courtesy of MacNeal-Schwendler Corp.)*

Main features. The tree decomposition technique produces a set of cells that must have a size that is compatible with the element-size distribution function. As the size of the cells in the tree is directly related to the expected local size, the element size resulting from the method will have a size close to the targeted value. In contrast to other mesh generation methods, there is no particular difficulty at the time the field points are created. As the field points are chosen at the quadrant (octant) corners, this stage is simple and does not require some specific checks (such as a filtering stage used to detect points that are too close together, etc.). However, this strategy of point location induces some rigidity. In other words, the point distribution may conform to the size function but the exact location of these points is not necessarily optimal. Hence, the extent of optimization required after the mesh completion may be relatively great.

The simplicity of the method implies in most cases its robustness. The only problem regarding convergence is related to the cases where it is not so easy to distinguish two entities which are rather close (two very close points which are not directly connected). This is why the case of the corners (points where the incident entities may form a rather acute angle) must be carefully considered.

Boundary discretization. The boundary mesh of the domain boundaries could be an input data of the problem or not.

In the case where this mesh is supplied, it is necessary to identify the points and the characteristics of this mesh (corners, ridges, discontinuities, etc.). On the other hand, it is not necessary to provide this surface mesh with an orientation, which is not the case for an advancing-front technique, for instance. As the spatial decomposition introduces some points (including some in the boundaries), it is necessary to check that the input mesh is a fine enough geometric approximation of the boundaries to avoid difficulties when creating a point on these boundaries.

In the case where the boundary discretization is not supplied, such a mesh will be automatically created based on the tree structure. In this case, we assume that a geometric modeler is available and is used by means of a series of queries. Indeed, it is necessary to know the point on the boundary closest to a given point, to find the intersection between the boundary and a cell, etc.

Curvature-based refinement. This approach is carried out to improve the accuracy of the geometric approximation of the domain boundary. Hence, finer meshes are achieved in highly-curved regions and coarser meshes in regions of low curvature. The geometric approximation error[7] can be related to a fraction of the desired mesh size as represented by the cell size [Shephard *et al.* 1996]. Hence, the length of a mesh edge is related to the level of the cell in the tree structure.

Tree decomposition. Starting from a box enclosing the whole domain, a tree is developed by recursive partition of its cells. The initial tree includes four cells in two dimensions and eight such cells in three dimensions. Each cell is analyzed so as to determine whether it conforms to a stopping criterion. If not, it is subdivided into four (resp. eight) cells of identical size. At completion, a covering-up of the enclosing box is obtained.

The stopping criterion used in the method is related to various facts based on the application at hand. The most widely used criteria state that:

- all cells include at most one boundary point,
- all cells with no point inside include at most one edge (face) of the boundary discretization,
- the size of all cells conforms to the size map.

This strategy implies that the decomposition tree enables us to *separate* two close boundary entities. Therefore, in two dimensions, for instance, two edges belonging to two opposite but close domain sides will belong to two different cells. This means that at least one point will be created inside the domain in each region. Hence, the tree acts as a neighborhood space and as a separator.

Tree balancing. Using this tree construction approach results in adjacent cells which can differ greatly in terms of size. Therefore, as the element sizes are related to the cell sizes, a smooth enough size gradation from element to element will be obtained if the size variation from cell to cell is bounded by a factor of two; the well known $2:1$ rule. Prescribing such a rule also results in another positive property. In fact, it allows us to know in advance the possible combinations of the cells and then those of the elements resulting from such combinations, at least for the internal cells[8]. It is then easy to define *a priori* all the patterns (*templates*) that will be subsequently employed to mesh the internal cells at a very low cost.

For efficiency reasons, tree balancing can be carried out when building the tree. When a cell subdivision is performed, the tree balancing is verified and the adjacent cells are subdivided or not depending on the case, then, in turn, the neighboring cells are considered and such a process is recursively applied.

[7]The maximal gap between the mesh edge or face and the curve or the surface.

[8]A cell is said to be internal (with respect to the bounding box) if it is not intersected by the domain boundary.

Filtering step. Due to the way the tree is constructed, say following an arbitrary order (with no special attention paid to the geometry of the domain), it is possible to have two points related to the intersection of the boundary with a cell side quite close to a cell corner or even to another intersection point related to another cell side. To prevent the creation of necessarily bad quality elements using these points, a point filtering step is applied. Provided the domain topology is preserved, some points can be merged together. Actually, this task may be tedious, particularly in three dimensions.

Element creation. The quadtree-octree method idea consists of using the tree structure for internal point creation as well as mesh element creation. As previously seen, the tree balancing rule reduces the number of transition patterns from cell to cell. Predefined patterns can then be used to quickly fill up the internal cells of the tree. A more general triangulation method (see [Shephard, Georges-1991]) is necessary to fill up the boundary cells (i.e., those that are intersected by the object boundaries). In this case, it is necessary to conform to the domain topology as well as to the domain geometry. For instance, in two dimensions, one must be sure that an edge related to a given geometric entity actually links two intersection points.

Following this approach, it should be noted that the cell corners and sides will be members of the final mesh where they will be element vertices and edges (faces).

External element removal. The mesh resulting from the previous stage is a mesh of the enclosing box. To obtain a domain mesh, all elements outside of this domain must be removed. To this end, a coloring algorithm is required (Chapter 2).

Optimization. Meshes as completed by this method are globally good quality meshes. Nevertheless, since the internal points correspond to the cell corners (apart from the boundary points), a certain rigidity may be present. Moreover, the tree construction does not explicitly pay attention to the intersections of the boundaries with the cells. In other words, the boundary may intersect a cell close to one side of it. As a consequence, ill-shaped elements can be constructed, for instance, rather flat elements. Thus, the classical optimization procedures (see Chapter 18) by means of topological and geometrical local operators can be used.

Numerical issues. Quadtree-octree type methods are relatively easy to implement. However this apparent simplicity may hide some numerical difficulties, for instance, point localization problems (in a cell), intersection problems (from the model and the tree cells), and tree traversal procedures. However, the global complexity, in terms of memory occupation and CPU time, is of order $\mathcal{O}(ne)$, where ne stands for the number of elements.

3.3.2 Advancing-front method

First suggested by [A.George-1971], this type of method has undergone significant improvements proposed by [Lo-1985], [Löhner, Parikh-1988] and more recently by

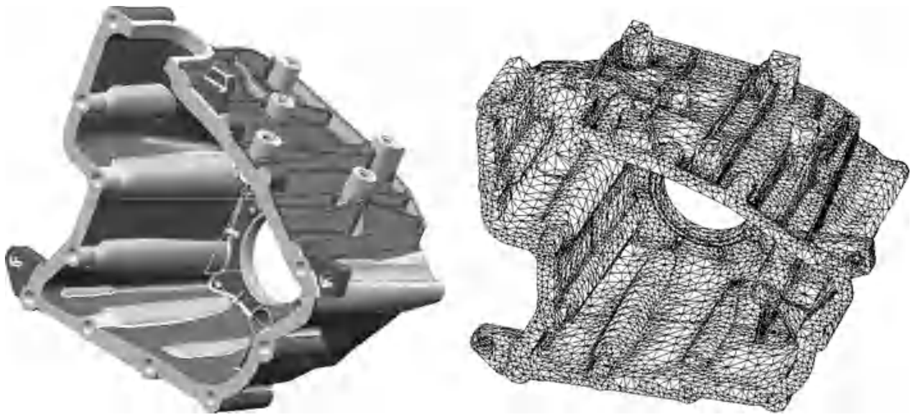


Figure 3.6: *Initial CAD model and octree type mesh (CATIA, courtesy of Dassault Systèmes)).*

[George, Seveno-1994], [Rassineux-1995] and various references like [Löhner-1996b] or [Peraire, Morgan-1997]. In this approach, the main idea is to construct the mesh element by element, starting from an initial *front* (i.e., a domain boundary discretization supplied as a list of edges in two dimensions and a list of faces in three dimensions). The technique proceeds by creating new points, or using a set of *a priori* created points, and connecting them with some points of the current front so as to construct the mesh elements. Thus, the yet unmeshed space is then gradually *nibbled* since the *front* moves across the domain. The front can be simply defined as the set of mesh entities (edges or faces, thus entities of d-1 dimensions) separating the part of the domain already meshed from the region not yet filled. The technique is iterative, an entity of the front (edge or face) is selected and a mesh element is formed either by connecting this entity to an existing point or to a newly created point so as to form a new good quality simplex. At each element creation, the front is updated and then dynamically evolves. This iterative process terminates when the list is empty; the domain is then entirely meshed.

General scheme. A classical advancing-front technique reads as follows.

1. Initialize the front with the domain boundary entities which can be sorted based on a given criterion.

2. Define the element size distribution function which could be provided as input data or constructed as the best from the available input informations.

3. Select the next entity from the front (based on a specific criterion). This leads to the following:

   - create an optimal point P based on the entity,
   - determine whether a mesh vertex V exists that should be used instead of P. If such a point exists, set V to P,

- check for element intersection, element size, etc., to validate the above choice,
- once a correct point has been identified, add the corresponding new element, update the mesh data structure and update the front.

4. Then as long as the front is not empty, return to 3.
5. Optimize the mesh (if needed).

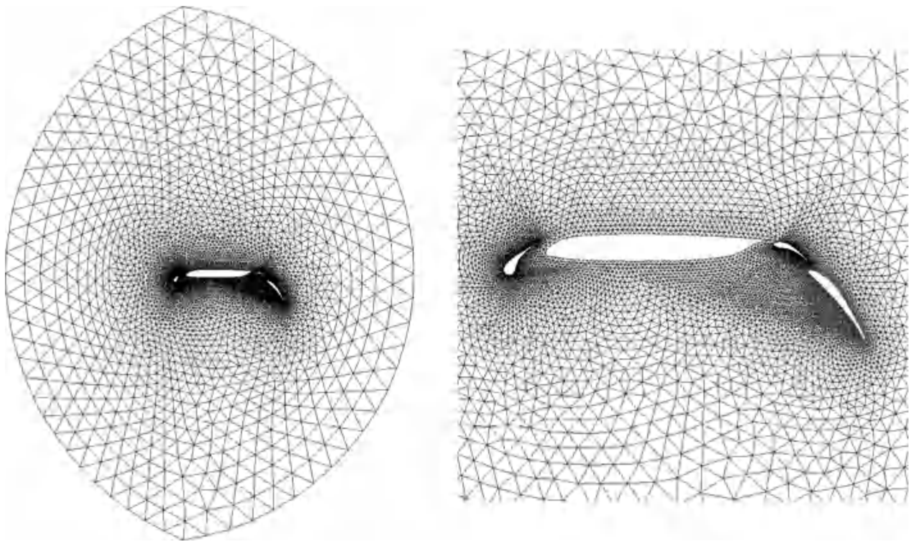


Figure 3.7: *Two-dimensional advancing-front mesh of a multi-airfoil without mesh optimization. Left-hand side: entire computational domain. Right-hand side: partial enlargement around the wing body.*

Critical features. Recurrent problems of any advancing-front method include the way to select a front entity, the (optimal) points identification and the element validation checks once a candidate point has been analyzed. All these operations must be made using robust and efficient algorithms since the convergence of the full process is strongly related[9].

Boundary discretization. The mesh of the domain surface (input data of the problem) forms the initial front. Each connected component of the boundary is orientated in a consistent way that allows the domain to be precisely defined with respect to its position around the boundary. In two dimensions, this leads to defining in which way the polygonal segments of the boundary are considered. In three

[9]In two dimensions, a theoretical issue about simple polygon triangulation (cf. Chapter 6) insures the convergence of an advancing-front method. However, this useful result is not so clear in three dimensions.

dimensions, the face orientations are such that, for a given connected component, all the face normals have the same orientation. The case of a *non-manifold* surface, in three dimensions, is more tedious and requires specific attention (for instance, the *non-manifold* face of the surface must be repeated).

Remark 3.8 *Since the boundary discretization is the initial front, boundary integrity is part of any advancing-front method and thus is preserved.*

Front analysis. The mesh elements are created based on the front entities. The selection of a front entity can be related to various criteria, based on the targeted solution. In general, this operation is not a purely local process. Indeed, it is necessary to anticipate the front evolution to prevent bottlenecks at a later stage. In two dimensions, one strategy resulting in a nice mesh gradation consists in selecting the smallest front edge (using, for instance, a *heap* structure). In three dimensions, it could be worthwhile selecting the faces at some neighborhood of the last created elements so as to minimize the required intersection checks.

Internal point creation. In general, mesh quality is a function of the internal point distribution. Provided that the desired element size is supplied everywhere (via a control space, cf. Chapter 1), several strategies can be used to find the location of a point from a given front entity. An *optimal* point is defined at the place where the element composed of the front entity and this point is regular (equilateral). In three dimensions, such a position belongs to the normal passing through the face centroid G at a distance from G dependent on the size function that is desired for the element. This location is then, if necessary, iteratively adjusted using an optimization procedure [Seveno-1997].

Candidate vertices. The above optimal point P is not necessarily inserted in the mesh. In fact, another vertex of the current mesh can be seen as a potential candidate due to its proximity to the point P previously computed. This proximity notion depends on the distance from point to point, according to the size specification $h(P)$ at point P when this information is provided (or has been constructed). In this respect, any point in a sphere centered at P, with radius $h(P)$ is *ipso facto* part of the candidate points. The set of such points is ordered based on the distance to P after which these points are analyzed so as to determine which one will be used to construct an element regarded as optimal.

Validation. The question here is to select among the admissible points the best candidate to construct an "optimal" element, in terms of aspect ratio (shape) and/or size. In fact, it is necessary to check that a guest element (if retained as a mesh element) has no intersection with the front and does not include any other mesh vertices. These checks must be rigorously implemented and the number of intersection tests must be reduced so as to minimize the cost of the full meshing process. To this end, specific data structures (binary trees or *quadtrees-octrees*) can be used to reduce the number of tests necessary. A point leading to an invalid element or leading to any intersection is removed from the list of admissible points.

Then, among the candidate points now identified (for which all the tests have been successful), the one that will result in the best quality possible (in shape or size) must be chosen. Moreover, it is of the utmost importance to analyze precisely the configuration obtained after any valid element creation to decide whether or not it may lead to a delicate or blocked configuration.

Convergence issue. Since the advancing-front method is based on local operations, convergence problems may be encountered, especially in three dimensions. A wide variation in size for the elements between two neighboring fronts may lead to intersection (overlapping) problems and the algorithm may have difficulty when meshing such configurations. As no theoretical results can guarantee that the method will complete a mesh in three dimensions, it is sometime useful to cancel some iterations and thereby removing some elements and points in the mesh. For efficiency reasons, such operations that enable us to overcome some local bottlenecks must be reduced as far as possible.

Front updating. Once an optimal point has been inserted in the mesh, one or several elements are created. The external faces of such elements (those separating the already meshed domain and the not already meshed regions) are stacked into the front list while the edge (face) of the former front used in the construction is removed from this list.

Mesh optimization. Meshes completed by an advancing-front method are in general good quality meshes. Nevertheless, due to the local aspect of the algorithms, it may be useful to optimize the resulting meshes. In such a case, the classical optimization procedures are used, see Chapter 18.

Remark 3.9 *The theoretical complexity of an advancing-front method is established to be in $\mathcal{O}(ne\log(ne))$, where ne is the number of simplices in the final mesh. In practice, efficient data structures are necessary to achieve this result, as pointed out in [George, Seveno-1994] and [Löhner-1996b].*

Surface meshing. A two-dimensional implementation of the advancing-front technique can be adapted to create surface meshes, provided that the surfaces are mapped onto a parametric space (logical space) or by using a direct approach. The aim is to obtain a nice approximation of the real surface by means of a piecewise surface (a mesh) in such a way as to obtain sufficiently good regularity (for instance, G^1 continuity).

In the first approach, the method uses the fundamental forms of the surface and completes an anisotropic mesh in a planar domain, the parameter space. This mesh is then mapped onto the physical space in such a way as to bound the gap between the triangle edges and the surface. This is done thanks to the so-called tangent plane metric or the metric of the maximal radii of curvature (cf. Chapters 13 and 15) that make it possible to compute the lengths of the segments in the parameter space using the geometry of the real surface, see [George, Borouchaki-1997], for instance.

Remark 3.10 (Variant) *The same technique can be applied to the generation of quadrilateral or hexahedral elements. In two dimensions, it is usually known as the* paving *technique [Blacker, Stephenson-1991], and in three dimensions as the* plastering *technique [Blacker, Meyers-1993].*

3.3.3 Delaunay technique

The computational-geometry properties of the Delaunay triangulations have been investigated for many years [Delaunay-1934]. Even before this date, in 1850, Dirichlet proposed a method to decompose a domain into a set of convex polyhedra [Dirichlet-1850]. However, the application of these techniques to mesh generation has only more recently been explored [Hermeline-1980], [Cendes *et al.* 1985], [Cavendish *et al.* 1985], [Baker-1986], [Weatherill-1985], [Mavriplis-1990]. The earliest strategies used predetermined sets of points as the Delaunay construction provides a suitable technique to connect these points, although it does not provide a mechanism to generate points. Moreover, the Delaunay triangulation of a domain may not preserve boundary integrity[10] which is a key requirement for mesh generation procedures and, thus, this point must receive special attention. Most of the current procedures for point insertion are based either on Bowyer-Watson's algorithm [Bowyer-1981], [Watson-1981] or Green-Sibson's algorithm [Green, Sibson-1978].

For a given set of points (or sites) $\mathcal{S} = \{P_k\}$, $k = 1, n$, a set of regions $\{V_k\}$[11] assigned to each of these points can be defined, such that any location within V_i is closer to P_i than any other of the points:

$$V_i = \{P : |P - P_i| \leq |P - P_j| \, , \forall j \neq i\} \, .$$

The regions are convex polyhedra, the Voronoï regions or cells. Joining all the pairs P_iP_j sharing a common segment of a Voronoï region boundary results in a triangulation of the convex hull of $\mathcal{S}$, the so-called *Delaunay triangulation.* The set of triangles (resp. tetrahedra) defining the Delaunay triangulation satisfies the property that the open circumdisk (resp. circumball) associated with the nodes of the element is empty (i.e., does not contain any other point of $\mathcal{S}$). This condition is referred to as the in-circle (resp. in-sphere) criterion and is valid in any dimension.

In this approach, an initial mesh is constructed, for instance, from a bounding box[12] enclosing the boundary discretization (list of edges and/or faces) of the domain. All boundary points are inserted iteratively into the initial triangulation of the bounding box, thus leading to a Delaunay triangulation with no internal vertex. The boundary connectivity constraint is not taken into account in this construction scheme. Hence, it is necessary to ensure that the entities of the boundary discretization are present in the resulting Delaunay triangulation and, if

[10] I.e., does not conform to a given boundary discretization.

[11] Known as the *Dirichlet tessellation* or the Voronoï cells.

[12] Introducing a bounding box is not strictly required, but is a source of simplification. Without this trick, the discussion becomes more subtle and makes it necessary to include several situations (rather than just one).

needed, to retrieve the missing entities by modifying the triangulation. Local mesh modifications are applied to remedy the situation and to finally obtain a mesh of the bounding box of the domain conforming the given discretization. Then, the external elements are removed (using a coloring scheme) and additional internal vertices can be created and inserted in the current mesh. Finally, the resulting mesh can be optimized to improve its quality.

General scheme. Provided with a size distribution function and a boundary discretization of the domain under interest, the global procedure for the mesh generation using a constrained Delaunay method can be outlined as follows:

Step 1. Initializations:

- input the boundary entities,
- construct an initial triangulation $\mathcal{T}_B$ of the bounding box of the domain.

Step 2. Insert all boundary vertices into $\mathcal{T}_B$.

Step 3. Construct an empty mesh $\mathcal{T}_e$ (no interior vertices) starting from $\mathcal{T}_B$:

- recover the missing boundary entities (boundary integrity),
- identify the connected components of the domain.

Step 4. Field point creation and insertion (i.e., enrich $\mathcal{T}_e$).

Step 5. Mesh optimization.

In this scheme, some steps deserve a special attention as they relate to the robustness of the method and influence the resulting mesh quality.

Delaunay kernel. Let $\mathcal{T}_i$ be the Delaunay triangulation of the convex hull of the set of points $\mathcal{S} = \{P_k\}$, $k = 1, i$, where $i = 1, n$. The insertion of $P = P_{i+1}$ in $\mathcal{T}_i$ results in the triangulation $\mathcal{T}_{i+1}$. This can formally be written as

$$\mathcal{T}_{i+1} = \mathcal{T}_i - \mathcal{C}_P + \mathcal{B}_P \,,$$

where $\mathcal{B}_P$ denotes the set of elements formed by joining P with the external edges (resp. faces) of the set of elements $\mathcal{C}_P$, whose circumdisk (resp. circumball) contains P. This insertion procedure is known as the *Delaunay kernel.*

Boundary integrity. Boundary recovery is commonly performed before the insertion of internal points, right after the insertion of boundary points. An alternative approach recovers the boundary integrity in the final stage of the mesh generation process, although it may result in a poor quality mesh, thus requiring an additional optimization stage [Mavriplis-1995].

One technique of recovering boundary integrity consists of inserting a number of additional boundary points until the triangulation conforms to the boundary, although the initial boundary discretization is obviously not stricly preserved. An alternative (and more elegant) approach consists of modifying the Delaunay triangulation using local mesh modifications operators to conform the boundary (cf. Chapter 18). This procedure matches exactly the specified boundary discretization. In two dimensions, if a boundary edge is missing, but its two endpoints belong to two adjacent triangles, an edge swap is used to recover the missing edge. In three dimensions however, the implementation of the procedure is more complex and more tedious. Moreover, additional points (the so-called *Steiner* points) are often required to enable the boundary recovery procedure to be performed [George *et al.* 1991a].

Field point creation. Various approaches have been investigated to create internal points. One strategy consists of inserting the new mesh points at the circumcenters of the elements [Holmes, Snyder-1988], [Weatherill, Hassan-1994]. This technique results in meshes for which the (dihedral) angles are bounded. However, the resulting meshes can be irregular and the mesh gradation is not well handled. An alternative approach consists of driving the point creation by the boundary point distribution.

It is assumed that the point distribution on the surface matches the geometric (curvature) as well as finite element requirements. This surface distribution is then extended into the domain using an interpolation scheme. One way is to create the points along the internal mesh edges, first in the empty mesh and then in the current mesh, so as to conform to the desired element-size distribution function [George *et al.* 1991b]. In this approach, the point creation is controlled by a background mesh (actually the empty mesh).

A third technique uses point sources considered as control functions with elliptic partial differential equations [Weatherill, Hassan-1994]. Another technique consists of using an advancing-front point-placement strategy to create the internal nodes. The front is then defined as the transition region between well-shaped and badly-shaped elements [Rebay-1993], [Frey *et al.* 1998].

Once created, the internal points are then inserted randomly[13] in the current mesh using the Delaunay kernel procedure.

Robustness issues. In two dimensions, thie Delaunay technique is very robust and reliable[14]. In three dimensions, however, the delicate task for the boundary recovery, which can be translated in terms of computer procedures, greatly diminishes the robustness of the method, which relies explicitly on heuristic techniques.

The Delaunay method results in a very efficient mesh generation technique. Most of the operations can be achieved in constant time (provided adequate data structures; cf. Chapter 2). The overall complexity (space and time requirements) is $\mathcal{O}(n \log(n))$, where n is the number of vertices (or the number of elements). Nowadays, this method appears to be one of the most efficient meshing techniques available, as speeds in excess of 500,000 or even more elements/minute[15] on current computers have been reported by several authors.

[13]In case a number of points must be inserted and not one at a time.

[14]It is easily proven that the boundary integrity can always be recovered.

[15]Compared to the same amount but in the matter of a second for a triangulation problem!

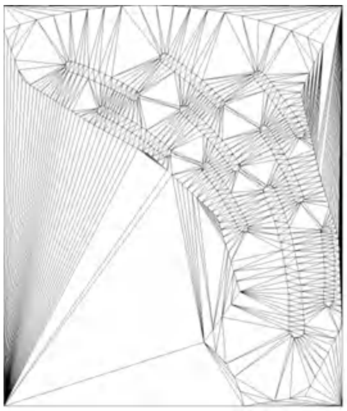
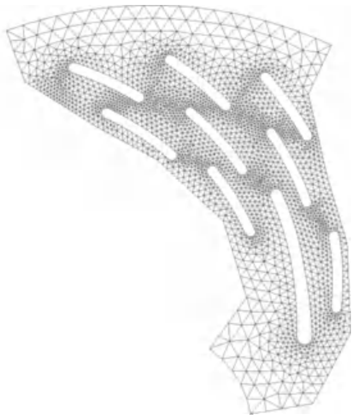



Figure 3.8: *Delaunay mesh of a mechanical device. Boundary mesh with no internal vertex (left-hand side) and resulting mesh after optimization (right-hand side).*

3.3.4 Tentative comparison of the three classical methods

Without seeking to provide a classification, it could be of interest to compare the meshes all methods previously discussed (*quadtree*, advancing-front or Delaunay) are likely to complete using the same description of a domain. This is quite easy to do since all these methods assume a boundary mesh as input data, and complete the same type of simplicial meshes.

Figure 3.9 displays different meshes of the same computational domain resulting from the quadtree, advancing-front and Delaunay type methods. Table 3.1 adds some statistics about the number of vertices np and the number of elements ne, the shape quality of the mesh $\mathcal{Q}_M$ and the quality value of the worst element $\mathcal{Q}_{worst}$ in these meshes (Chapters 1 and 18).

method	np	ne	$\mathcal{Q}_M$	$\mathcal{Q}_{worst}$
quadtree	$1,246$	$2,171$	1.25	1.88
advancing-front	$2,557$	$4,795$	1.1	1.61
Delaunay	$2,782$	$5,528$	1.16	1.82

Table 3.1: Statistics relative to the different meshes depicted in Figure 3.9.

Observing these examples indicates a non-negligible variation in size for the meshes (the number of elements is in a ratio of 2 or more) while the mesh qualities are of the same order (close to 1 !). The CPU costs are all less than one second. Thus, at least in two dimensions, all the methods produce rather similar meshes (in terms of quality). Notice that the element sizes inside the domain are only related to the boundary discretization provided as input data. The quadtree type method,

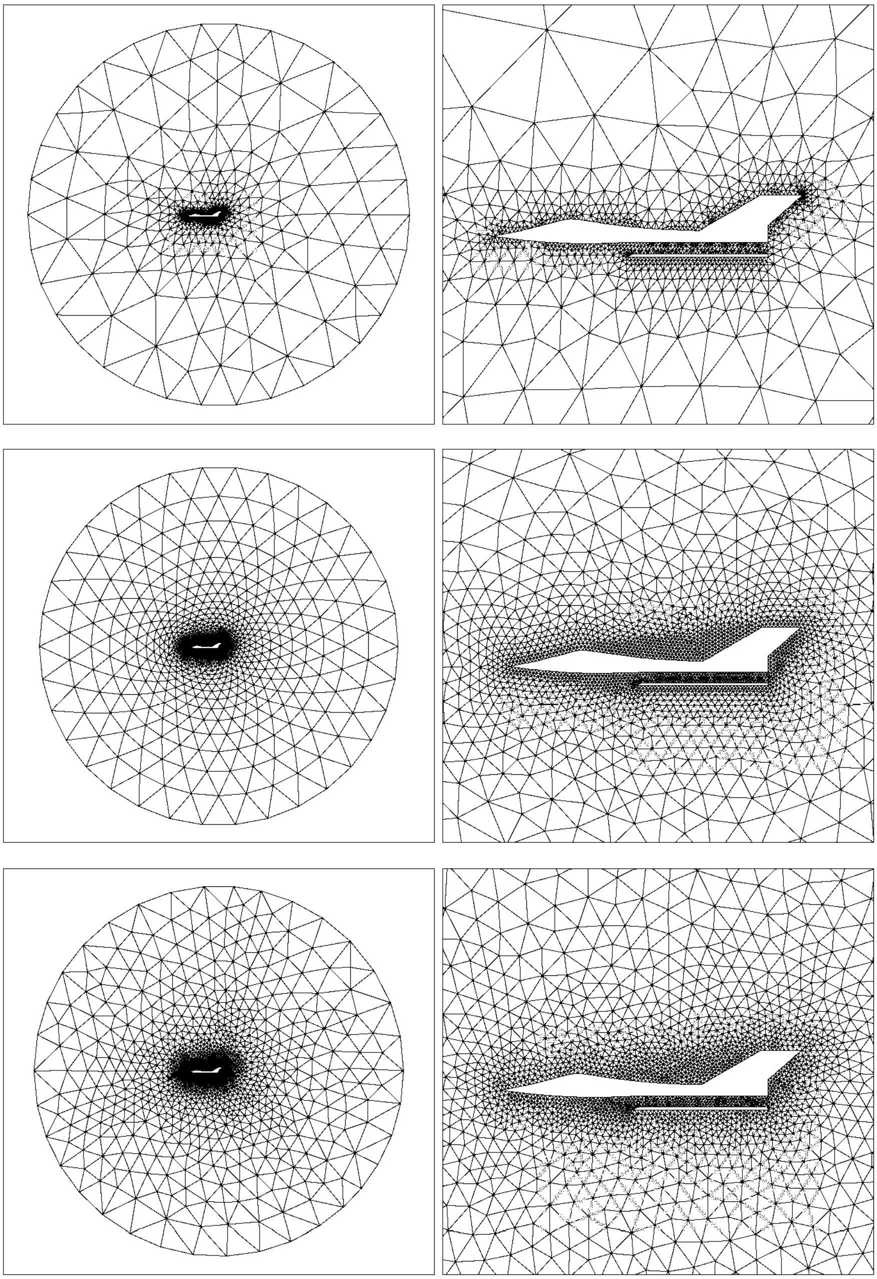


Figure 3.9: *Overview of the different mesh generation methods when applied in a domain (used, for instance, for a CFD problem). Quadtree type mesh (top), advancing-front type mesh (middle) and Delaunay type mesh (bottom) including a close-up view around the fuselage.*

based on a 2 : 1 gradation rule, results in larger elements inside the domain (say, far away from the boundary) and thus produces fewer elements than the two other methods that could be tuned to produce such behavior too. Advancing-front and Delaunay meshes are quite similar in terms of size.

A more subjective view could produce some differences regarding the *aesthetics* of the created meshes.

3.3.5 Other methods

Various other techniques have been developed for unstructured mesh generation, although none of them seems to be widely used today. However, two classes of techniques offer interesting features in specific fields of applications: hybrid methods, which are useful for CFD (Computational Fluid Dynamics) computations, and partitioning methods, which can be used in parallel applications.

Hybrid methods. These approaches combine features of structured meshes (in general in the vicinity of the domain boundaries) and unstructured meshes (for instance, [Weatherill-1988], [Kallinderis *et al.* 1995], [Khawaja *et al.* 1995], and [McMorris, Kallinderis-1997]). In hybrid prismatic/tetrahedral meshes, the initial surface triangulation is the outer prismatic surface. In general, layers of prisms are used to resolve boundary layers and wakes, while tetrahedral elements cover the remaining part of the computational domain. A hybrid type mesh combining elements of different orientations seems more flexible to accommodate the different flow features. The most common technique employed for generating prismatic elements is a *marching* method that starts from a surface and propagates towards an outer boundary. The marching direction vectors are based on the normal at the surface vertices and the marching distances along these vectors (i.e. the stretching of the nodes along the direction) are dependent on the physics of the problem (for instance related to the Reynolds number) [Garimella, Shephard-1998]. The grid is built one layer at a time in an iterative process. The unstructured mesh is generated using any classical unstructured mesh generation method (Delaunay, advancing-front, octree, etc.).

The tedious part in developing this type of mesh generator lies in the management of the interfaces between the structured and unstructured meshes. For instance, if a Delaunay algorithm is used, the boundary integrity constraint may be relaxed or even omitted. A promising technique consists of using a *buffer* layer for the transition from the prismatic to the tetrahedral elements; for instance, pyramidal elements with quadrilateral bases can be introduced. An alternative method consists of allowing the outer layer of prismatic elements to be broken down during the boundary recovery stage.

Partitioning methods So far, this chapter has only dealt with unstructured mesh generation methods that generate simplicial or hybrid meshes. Quadrilateral meshes may be desirable in some applications (structural mechanics for instance) where they lead to increased computational performance and numerical accuracy.

In this context, partitioning methods offer a way to design automated algorithms for producing well-structured quadrilateral or hexahedral meshes.

Most of the first partitioning algorithms subdivided the domain recursively until simple elements (i.e., patterns) or very simple transition meshes remained [Cavendish-1974], [Schoofs *et al.* 1979]. This class of techniques is known as recursive partitioning. Another approach consists of separating the mesh generation in two phases: an automatic subdivision of the domain into subregions and the meshing of the resulting subregions. The first stage is based on the identification of suitable subdivisions and uses the medial axis (surface) of the domain or the Voronoï diagram of its edges (faces) [Tam, Armstrong-1991], [Armstrong *et al.* 1995]. In the second stage, an algebraic method (or a similar method) can be used to mesh the various regions resulting from the partition.

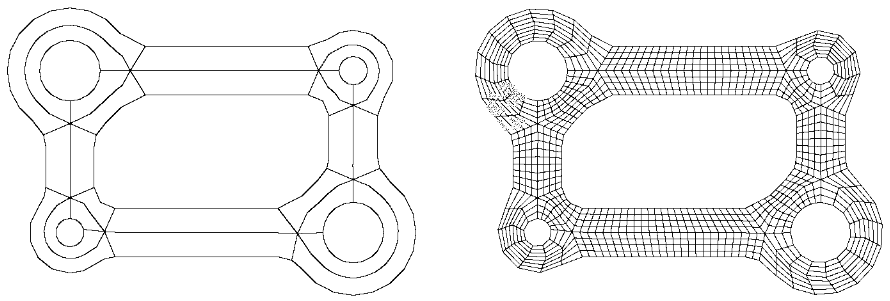


Figure 3.10: *Partitioning method based on medial axis subdivision: skeleton of the domain (left-hand side) and quadrilateral mesh of the domain (right-hand side).*

Quadrilateral meshing. Direct or indirect approaches may be adopted to generate quadrilateral meshes for domains of arbitrary shape.

- Direct methods.

Among the direct methods, essentially two approaches have been investigated: a domain decomposition technique followed by quadrilateral sub-domain filling by means of an algebraic method [Armstrong *et al.* 1995], [Talbert, Parkinson-1991] and the quadrilateral paving techniques [Blacker, Stephenson-1991]. The first approach is domain decomposition sensitive and relies on the quasi-convex nature of the resulting sub-domains. The domain decomposition algorithms usually require local or global knowledge of the domain. In this last case, in particular, the skeleton fully defines and allows an accurate decomposition of the domain. The second method consists of paving the domain from the boundary to the interior and of managing the front collisions. By its very nature, the performances of this method are closely related to the boundary discretization.

When a constant isotropic metric field is specified, these two classes of methods are likely to lead to the same results. In fact, in an advancing-front method, the front shape tends toward the skeleton. On the other hand, if a generalized metric map is specified, the second method is more likely to respect the field.

• Indirect methods.

Given a triangular mesh of the domain, the indirect approaches aim at combining triangles to form quadrilaterals [Lo-1989], [Johnston *at al.* 1991], [Lee, Lo-1994], [Zhu *et al.* 1991], [Rank *et al.* 1993], [Lewis *et al.* 1995] and lead to two related merging processes. The triangle merging procedure is either driven by the quadrilateral quality [Borouchaki, Frey-1998] and may lead to mixed triangular-quadrilateral meshes, or starts from the boundary and moves to the interior of the domain, ensuring an even number of vertices when two fronts collide and results in pure quadrilateral meshes (if the boundary discretization has an even number of vertices). This second method requires a topological classification of the front collisions.

The resulting quad meshes can be enhanced using a specific optimization procedure capable of optimizing the shape or the size quality of the elements[16] (cf. Chapter 18).

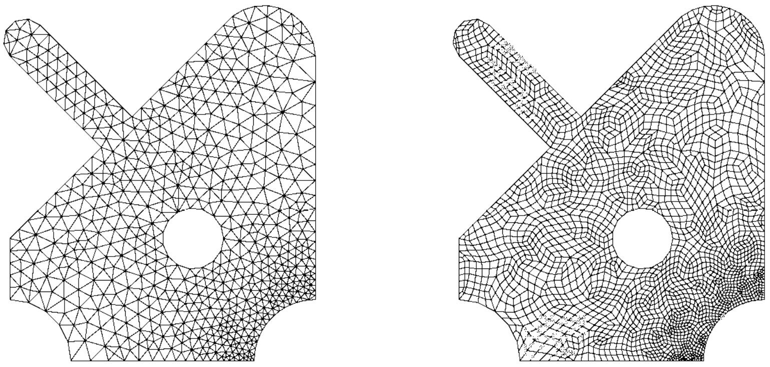


Figure 3.11: *Triangular to quadrilateral conversion, indirect approach. Original triangular mesh (left-hand side) and optimized quadrilateral mesh (right-hand side).*

Mesh generation by local optimizations. In the context of moving (evolving) mesh methods (such as those encountered in forming calculations for instance), it is often desirable to allow the mesh topology to change progressively rather than to build a new mesh. The local remeshing is required to avoid element distortions due to large deformations and to adapt the mesh dynamically to the new conditions [Coupez-1995].

An initial mesh of the domain can be optimized for any kind of criterion, for instance, one related to the volume of the elements. Starting from a very crude mesh (obtained, for instance, by connecting a node to each face of the boundary of a non-convex domain, thus leading to overlapping elements), the optimization

[16]While bearing in mind that merging two good quality triangles does not necessarily lead to a good quality quad or even a valid quad.

stage will attempt to minimize the sum of the absolute value of the element volume. The improvement process tends to optimize an element shape quality function and internal nodes can be introduced to remove locked configurations and to optimize the mesh, see Chapter 18.

3.4 Surface meshing

Surface meshes play an important role in numerical simulations using finite (volume) element methods and the quality of the geometric approximation may affect the accuracy of the numerical solutions. In this context, a surface mesh is usually intended to be the boundary description of a domain used in a three-dimensional finite element analysis. Therefore, the surface mesh must conform to specific properties, related for instance, to the geometry of the surface it represents and to the behavior of the physical phenomenon. The aim is to create an optimal piecewise planar approximation of the original surface in which the maximal distance between the original and the approximating surface does not exceed a given tolerance.

Depending on the surface definition, three techniques have been investigated: mesh generation via a parametric space (if a CAD modeling system has been defined, for instance), mesh generation for implicitly defined surfaces (e.g. isosurfaces or levelsets) and, if the sole data is a given surface discretization (i.e., a surface triangulation), surface mesh optimization which proves especially useful in the study of large deformations in structural mechanics.

3.4.1 Mesh generation via a parametric space

A regular surface parameterized by u, v can be defined using a function σ as:

$$\sigma : \Omega \subset \mathbb{R}^2 \longrightarrow \Sigma \subset \mathbb{R}^3, (u, v) \longmapsto \sigma(u, v), \tag{3.3}$$

where Ω is a domain of $\mathbb{R}^2$ and σ is a smooth enough function. The goal is to achieve the final surface mesh via a triangulation in the parametric (logical) space. The two-dimensional mesh generation is governed by a set of metrics related to the intrinsic properties of the underlying surface. These metrics correspond to sizing and/or directional specifications. The control induced in this way is then explicitly written as a criterion about the lengths of the mesh edges; see [Dolenc, Makela-1990], [Samareh-Abolhassani, Stewart-1994], among many others.

General scheme. Let Ω be the domain in $\mathbb{R}^2$ corresponding to the parameterization of a surface Σ. The parameter space is supplied with a Riemaniann structure, used to govern the meshing process, which is constructed according to the nature of the expected mesh (isotropic, anisotropic, specified sizes, constant sizes, etc.). Actually, the metric of the two fundamental forms of the surface is involved. The issue here is to mesh Ω with normalized unit length mesh edges

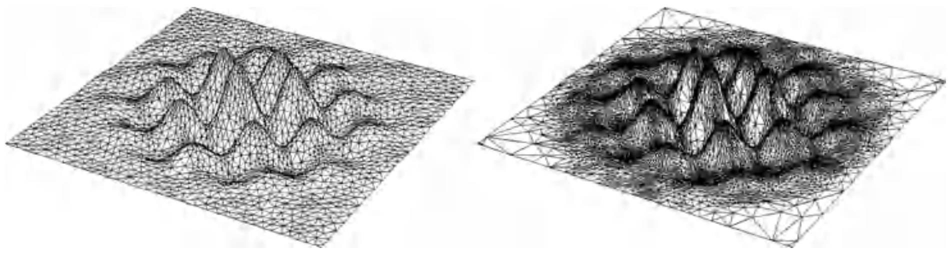


Figure 3.12: *Parametric surface meshing, analytical example. The surface is defined as:* $z(x, y) = 2.5e^{-0.1(x^2+y^2)} \sin(2x) \cos(2y)$ *in the domain* $[-6, 6] \times [-6, 6]$ *of the Oxy plane. Left-hand side: constant-size surface mesh, right-hand side: geometric surface mesh.*

(with regard to the metric) and in such a way that the resulting elements are of good quality.

The domain meshing process consists of three stages. The two first concern the parametric space and consist of meshing the boundary of Ω and then meshing Ω using this boundary mesh as input data. The final stage consists of mapping this mesh onto the surface.

Meshing a surface boundary. The discretization of the curves defining the surface boundary enables us to construct a geometric support using a well-suited mathematical representation. This support is approximated by a polygonal segment whose constitutive segments are unit length segments. This polygonal segment is the sought mesh (Chapter 14) unless a map that is not necessarily of compatible size must be adopted. This mesh is constructed using a mesh of the boundaries of Ω and, at completion, the boundary discretization is achieved.

Meshing the domain. To complete the surface mesh, we construct, using a suitable method (Delaunay or advancing- front, in general), a mesh whose vertices are the points of the boundary discretization together with a series of internal points such that unit edges and good quality elements are obtained (with respect to an appropriate metric). This construction is made in Ω using the discretization of the sides of this region as previously created.

Mapping onto the surface. Mapping the mesh in Ω onto surface Σ is rather easy; it is merely necessary to apply function σ. The vertex positions are the image by σ of the vertices in the parametric space. The connections are those of this planar mesh.

Remark 3.11 *This technique can be applied to surfaces defined using several patches, each of which corresponds to a parametric space. In this case, the meshing procedure starts by meshing the interfaces between the patches so as to insure the overall mesh conformity. In this way, a patch-dependent mesh is obtained. This constraint can be overpassed by using local remeshing procedures (Chapter 15).*

3.4.2 Implicit surface triangulation

Recent years have witnessed increasing interest in the use of implicit functions for defining geometric objects, for instance, in the field of Computer-Aided Geometric Design [Requicha-1980], [Ricci-1972], [Wyvill *et al.* 1986], [Pasko *et al.* 1995] or in applications where the domains involved are obtained using tridimensional scanning and sensing devices (e.g. biomedical imaging systems). They are called *implicit* because they represent subsets of $\mathbb{R}^3$ that are not specified explicitly by their boundaries or parameterizations.

An implicit algebraic surface can be defined as the set of points (x, y, z) in $\mathbb{R}^3$ that conform to an equation such that:

$$f(x, y, z) = 0\,. \tag{3.4}$$

A geometric object is considered as a closed subset of $\mathbb{R}^3$ with the definition $f(x, y, z) \geq 0$. The boundary of such an object is a so-called implicit surface and is the two-dimensional manifold in $\mathbb{R}^3$ such that $f(x, y, z) = 0$. The *defining function* f may be defined or approached in different ways, depending on the field of application.

Scheme of the method. There are relatively few papers[17] on implicitly defined surfaces (cf. [Ning, Bloomenthal-1993] for an overview). The classical scheme proposed by [Allgower, Schmidt-1985] is now well recognized in most of the approaches is based on two operations:

- a sample of the function values at the vertices of a covering-up set of the domain,
- the connection of these vertices in order to obtain a mesh.

The sampling step aims at creating a set of points, all belonging to the implicit surface. This task is delicate as, in general, it is necessary to solve non-linear equations. The aim of the connection step is to construct a topology similar to that of the surface and, in addition, such that a well-suited surface approximation is obtained, from a geometric point of view.

Various approaches. A popular technique consists of sampling the basis function in the space and uses a numerical method to find the zeros of this function. The mesh vertices will then correspond to the roots of the equation. In practice, a spatial decomposition of the domain is developed (for instance, using an *octree*) and the function is approximated locally (in each cell) by a piecewise linear surface. The global conformity of the mesh is insured by the cell subdivision rule with the same idea as in an *octree* type method. The cell size may depend on the local curvature or on some other explicitly defined parameters. There are two classes of algorithms depending on the nature of the data. In the case of discrete data, the implicit function is not exactly accessed since the sole values available

[17]In comparison with the literature on parametric surfaces.

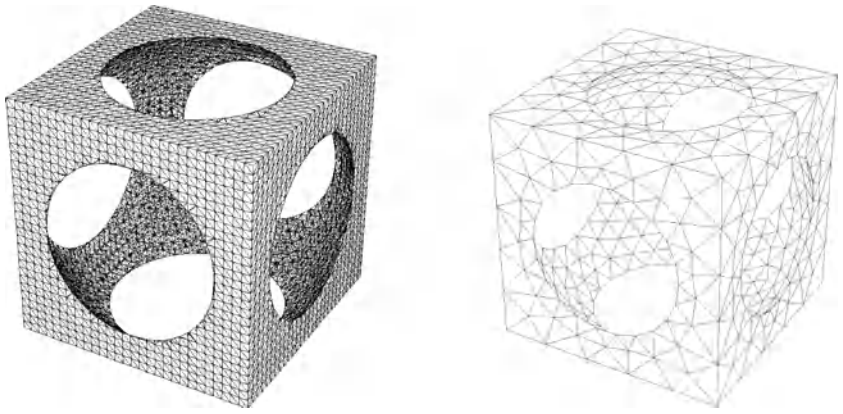


Figure 3.13: *Example of implicit surface meshing, the domain is defined as the extrusion of a sphere from a cube using CSG primitives. Left: original triangulation based on a regular grid partitioning; right: optimized geometric surface mesh.*

are those at the vertices of the covering-up. A linear interpolation can then be used to compute the values of the function everywhere in the domain and thus find the intersections between the function and the edges of the covering-up.

A simple numerical technique for constructing such a covering-up uses an octree [Bloomenthal-1988]. Lorensen and Cline introduced an algorithm[18] which is now commonly used for constructing a polygonal representation of a constant density surface using voxels, and numerous algorithms to guarantee topological correctness of the polygonization of isosurface have been proposed since then, see [Lorensen, Cline-1987]. Also, a Delaunay mesh of the convex hull of the points in the sample can be used to construct this covering-up [Frey, Borouchaki-1996]. The meshes obtained by any of these techniques are then optimized (for instance, according to size specifications) using classical mesh modifications operations (cf. Chapter 19). This is due to the fact that no special attention is paid to the quality of the triangles at the time they are constructed since the only concern is to track the surface.

3.4.3 Direct surface meshing

This approach consists of applying a classical meshing technique directly to the body of the surface, without using any kind of mapping [Shephard, Georges-1991], [Nakahashi, Sharov-1995], [Chan, Anatasiou-1997], [McMorris, Kallinderis-1997]. In such approaches, the mesh element sizes and shapes are controlled by analyzing the local surface variations. At first, the curves representing the surface boundary are discretized, then the surface mesh is created using any unstructured meshing technique. The difference between the various approaches proposed lies in the algorithm used to find an optimal point location on the surface. An optimization

[18]The so-called *Marching Cubes.*

stage is usually required to improve the element shapes after the generation of the initial surface mesh.

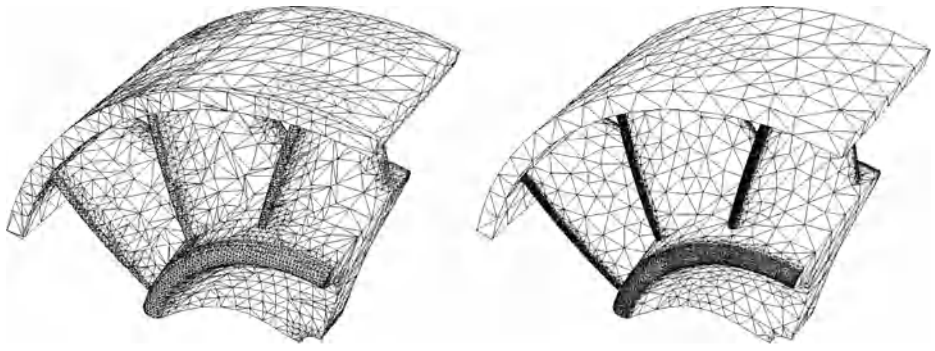


Figure 3.14: *Example of direct surface meshing (octree-based method with curvature-based refinement). Left-hand side: original triangulation (data courtesy of MacNeal-Schwendler Corp.); right-hand side: optimized surface mesh.*

3.4.4 Surface remeshing

Although the surface meshing approaches described in the previous sections seem appropriate, in many cases the domains are not defined in terms of analytical functions but rather by means of a surface triangulation. Such applications include: numerical simulations where the surface results from measurements, biomedical engineering where the domain is provided by a sensing or scanning device, numerical simulations that involve remeshing (e.g. forging problems, fluid-structure interaction problems, etc.). In this context, we consider the problem of generating geometric finite element meshes given an arbitrary surface triangulation representing the surface, possibly supplied with geometric specifications (ridges, singular points, etc.) [Löhner-1996a], [Frey-2000].

Problem statement. We are concerned with a case where the surface is defined through an initial triangulation enjoying some geometric properties[19]. From this triangulation, a continuous mathematical support is constructed, which will be used to collect the required information by a system of queries. The problem then involves constructing a new mesh, conforming to the given specifications, by means of successive modifications applied to the initial triangulation. The required information are as follows:

- the placement of a point on the surface,
- the surface property collection at a local level (discontinuities, minimum radius of curvature, main curvature radii, normals and tangents, etc.).

[19]If a CAD model is available, the surface is known using a series of queries to the modele.

The remeshing procedure uses local modification operators that can be of a topological nature (to control the geometric approximation) or of a metric nature (subdivision of edges that are too long, vertex removal, vertex relocation, etc.).

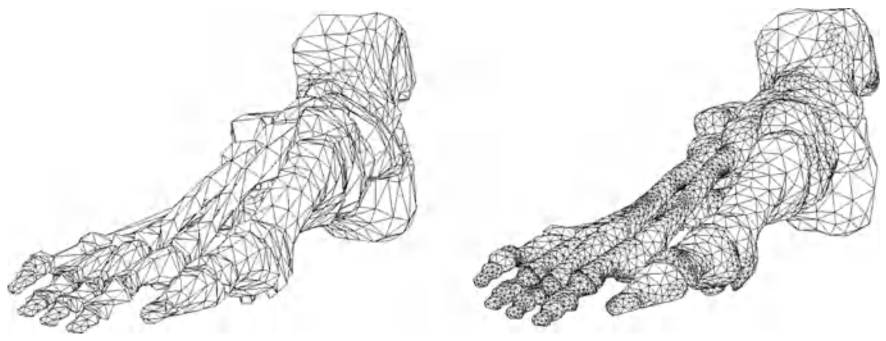


Figure 3.15: *Surface mesh optimization. Polyhedral biomedical iso-surface reconstruction from volumetric data (public domain data, Naval Air Warfare Center Weapons Division). Initial triangulation (left-hand side) and optimized surface mesh for a tangent plane deviation bounded by 37 degrees (right-hand side).*

Control of the geometric approximation. The initial surface triangulation is optimized in accordance with the geometry to obtain a so-called *geometric* mesh (regarding the geometric approximation of the surface) and also to the element quality, in other words, a mesh which best fits the surface geometry. In such a mesh, the maximal gap between an edge of the discretization and the real surface is bounded by a tolerance threshold value. Additional constraints regarding the element shape and size can be enforced.

Remark 3.12 *The problem is to start from an initial surface triangulation which contains a reasonably small number of elements and still represents an accurate polyhedral approximation of the surface. Usually, the given initial surface triangulation needs to be geometrically simplified (with respect to a geometric tolerance), prior to building the geometric support.*

Governed surface remeshing. The given triangulation initializes the current mesh. In this mesh, we identify the singularities (corners, ridges, C^0-discontinuities). A size map is constructed by evaluating in a discrete manner the intrinsic properties of the surface (Chapter 19). The edges in the current mesh are then analyzed one at a time so as to obtain unit mesh edges. This leads to:

- subdividing any edge with a length greater than "one" into sub-edges of unit length (or a value close to one),
- removing any short edge, provided the topological consistency is preserved.

Point enrichment or removal are combined with edge swap (cf. Chapter 18) in order to enhance element quality. This process is repeated until all edges conform to th e given specifications. At completion, the surface mesh conforms to the intrinsic size map (i.e., the geometric map) or to any other given map.

3.5 Mesh adaptation

Solution-adaptive meshing is a very promising technique that improves the numerical accuracy of the solution at a lower computational cost. It relies on a more efficient (optimal) point distribution (in terms of their number and their location), and also on the shape of the element in the mesh (isotropic or anisotropic shape, for instance). A new mesh is then constructed (using one of the approaches discussed in this chapter or by using a remeshing procedure), then the process is repeated. The new mesh is assumed to better capture the physics of the problem in hand. The successive iterations aim at optimizing this distribution based on*a priori* or *a posteriori* error estimate. Starting from an initial mesh, an initial solution is computed. It is then analyzed by an error estimate and this analysis is converted in terms of size specifications used in turn to govern the mesh adaptation. The solution accuracy is also strongly related to the interpolation step from the computational mesh (the previous iterate) and the current mesh.

Mesh adaptation scheme

Despite several differences between the possible approaches suitable for adaptive meshing, the following steps are usually representative of an adaptive meshing strategy:

- Construction of an initial mesh $\mathcal{T}_i$.
- Computation of the initial solution u_i on $\mathcal{T}_i$.
- (A) Estimation of the local error in u_i.
- (Re-)Construction of a mesh $\mathcal{T}_{i+1}$ according to the estimated error values:
  - construction of the control space $\mathcal{CS}_i$ associated with u_i,
  - construction of the governed mesh $\mathcal{T}_{i+1}$ with respect to $\mathcal{CS}_i$.
- Transfer of solution u_i on $\mathcal{T}_{i+1}$.
- Resumption of the solution procedure: return to (A) with $i = i + 1$.

Mesh adaptation techniques

Adaptation methods can be broadly classified into three categories. The first category consists of a local or global modification of an initial mesh so as to adapt it to the computational requirements. The second category includes the

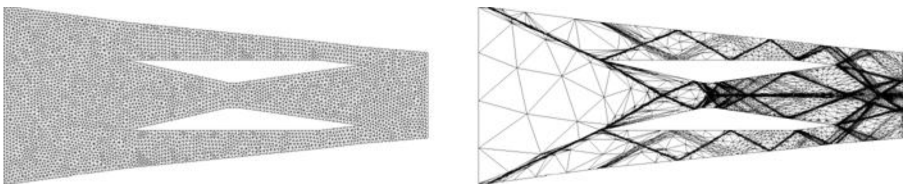


Figure 3.16: *Adaptively generated mesh for the computation of a supersonic flow (Mach 3) over a Scramjet. Left-hand side: initial mesh (*4,000 *vertices); right-hand side: final mesh (*90,000 *mesh points).*

global methods that reconstruct the whole mesh at each iteration step. Finally, some other methods attempt to combine these two approaches. The adaptation is first made at a local level during a few iterations, then the mesh is entirely reconstructed prior to being locally updated again. In this classification, we can distinguish between r-methods, h-methods, p-methods and the coupling of these last two resulting in hp-methods (cf. Chapters 21 and 22).

Adaptive remeshing. The mesh on which a solution has been computed becomes the *background mesh* for the next iteration step. A discrete element-size specification function is defined at the vertices of the background mesh, for instance, from a Hessian-based criterion or any type of error analysis method. A new unstructured mesh is then generated by a classical mesh technique governed by this new size map [Peraire *et al.* 1987], [Shephard *et al.* 1988], [Löhner-1989], [Mavriplis-1990].

Mesh modifications. Local remeshing (refinement, coarsening) is an alternative and sometimes a less computationally expensive approach to mesh adaptation, based on mesh optimization techniques. The current mesh is modified in the regions where the discrepancies between the current and the specified element size are too large [Rivara-1984b], [Cendes, Shenton-1985a], [Rivara-1991]. Subdivision refinement (*h-refinement*) can be used to produce nested meshes (i.e., containing a subset of the initial mesh vertices), thus allowing an accurate transfer of variables from one mesh to another (in multigrid approaches, see [Désidéri-1998]).

3.6 Parallel unstructured meshing

The requirement to develop fast and reliable unstructured mesh generation algorithms is common to several computational fields of application. Moreover, large meshes (in excess of several million elements, for instance, in some CFD problems or wave propagation) are now frequently managed in these disciplines. In order to benefit from parallel architectures, the whole simulation process (including mesh generation, numerical solving, adaptive remeshing and visualization) need to be efficiently parallelized. Parallel computing is then a way to solve very large size problems (irrespective of the cost).

The unstructured mesh generation techniques commonly used are intrinsically scalar as they create one entity (point or element) at a time. Parallelism can be achieved if the points to be inserted are sufficiently far apart (the neighborhoods associated with these points are distinct). As pointed out by [Shostko, Löhner-1995], *distance* is the enabling factor for parallelism. Moreover, parallel mesh generation is a tedious task as it requires the ability to decompose the computational domain into sub-regions that can be meshed separately on different processors. This is referred to as the *partitioning* stage.

Parallelism and meshing processes

The strategy of parallel mesh generation can be divided into two categories:

- the mesh generation method includes parallelism,
- the mesh generation process is parallel while based on a serial mesh generation method.

The first approach introduces parallelism at the mesh generation method level, while the second consists of using a given meshing method in parallel. This approach leads to meshing each sub-domain separately after the definition of the various domain interfaces and after a mesh of these interfaces have been completed.

The second approach avoids the specific tests about the boundaries of the different meshes (in terms of conformity). On the other hand, meshing the interfaces is relatively tedious [Shostko, Löhner-1995].

The mesh generation methods resulting in unstructured meshes as seen in this chapter are basically scalar methods. Indeed, in general, they allow the creation of one point or one element at a time. It is possible to include some degree of parallelism if the points they try to insert are sufficiently far. In other words, apart when considering a point, there exists a certain neighborhood with no intersection with another one. Distance is then the key factor in meshing parallelism. On the other hand, a solution method based on a domain decomposition technique requires the construction of several sub-meshes whose union provides a covering-up of the whole domain. Then, each sub-domain is dealt with in one processor. One of the difficulties is therefore to transfer the data values associated with one sub-domain (and then belonging to one processor) to another sub-domain (another processor). The efficiency of the method then relies on the load balancing between the different processors (Chapter 24).

Domain decomposition

The aim of domain partitioning is to minimize the amount of inter-processor communication as well as to balance the computational load per processor. The partitioning process can be subdivided into three classes depending on how the domain is split [deCougny-1997]:

- partitioning of an initial (coarse) mesh [Wu, Houstis-1996],
- partitioning of the domain (and not a boundary mesh) using a spatial decomposition method [Saxena, Perucchio-1992],
- direct partitioning (also called pre-partitioning) of the mesh of the domain boundaries (a surface mesh in three dimensions) [Galtier, George-1996].

Once the partitioning has been completed, sub-domain meshing is performed in parallel with or without inter-processor communication.

***A posteriori* partitioning.** Provided with a mesh of the domain under interest, an *a posteriori* partitioning method consists of splitting this mesh into several sub-meshes so as to have the sub-domains defined. Various techniques have been proposed for this purpose (cf. [Simon-1991], [Farhat, Lesoinne-1993] for instance).

The main drawback of such a method is related to its memory requirement. In fact, it is necessary to store the initial mesh and, at least, one of the sub-meshes. Moreover, all the problems related to any partitioning methods must be addressed (load balancing, interface smoothness, etc.).

***A priori* partitioning.** The purpose of this approach is to construct *a priori* a partition of the domain, from the data of a coarse mesh of it or directly from a discretization of the domain boundaries. Once this partition is available, the resulting sub-domains are meshed in parallel thus taking advantage of the parallel capabilities of the computers right from the meshing stage.

The main difficulties in this approach are related to the load balancing aspect (that must be deduced from the coarse mesh or the domain boundary) and to the proper management of the domain interfaces. The coarse mesh may be an empty mesh (without internal vertices), for instance, resulting from a Delaunay method. The interface between two sub-domains is constructed either from the data of the coarse mesh or from the data of the domain boundary discretization.

Chapter 4

Algebraic, PDE and Multiblock Methods

This chapter describes some algebraic methods, some methods based on the solution of PDEs and multiblock-type methods. An algebraic method is designed to carry out the mesh construction of domains having an analogy with a simple shaped logical domain (such as a square or a quadrangle, a triangle, etc.). A PDE-type method is designed to handle domains that can be mapped onto a square (a cuboid in three dimensions) using different kinds of analogies. These methods are therefore limited regarding the shape of the domains they can successfully deal with. A multiblock type method is one possible solution to carry out arbitrarily shaped domains. First, the domains are decomposed into simply shaped regions where the previous methods can be used. Then, the mesh of the entire domain is obtained as the union of the local meshes corresponding to the above regions.

★
★ ★

The first section discusses various algebraic methods based on a mapping function which is defined *a priori*. The second section briefly considers PDE style methods where the mesh is obtained by solving an adequate system of differential equations. The third section shows how to define a multiblock method using one of the above methods as a local meshing process.

4.1 Algebraic methods

Any algebraic mesh generation method consists of constructing a mesh on a (real) domain using a given function that is explicitly defined. Main references about algebraic methods, mostly for quad or hex geometries, include [Gordon, Hall-1973], [Cook-1974] and, for transfinite interpolation style methods suitable for simple shapes, a synthesis by [Perronnet-1998]. The given function is used to map an easily defined mesh in a logical domain $\hat{\Omega}$ geometrically simple (unit square, unit triangle, etc.) onto the real domain Ω (see Figure 4.1). The logical mesh thus created is in essence a structured mesh. In general, the mapping function consists of polynomials defined in such a way as to ensure certain properties.

In this section, we assume that a suitable domain boundary discretization is supplied as input data and we show how to construct a mesh covering the domain thus defined. In two dimensions, the boundary discretization consists of a series of segments (a polygonal line) enjoying some specific properties. In three dimensions, this discretization is a surface mesh which also conforms to a peculiar type.

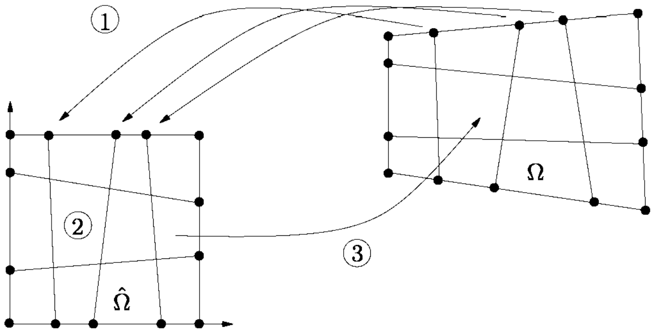


Figure 4.1: *General principle of any algebraic method in the case where the domain Ω is considered as a quad. Three steps are involved, the data mapping on the boundary, the mesh construction into the reference square and the mapping of this mesh onto the actual domain (steps denoted by 1, 2 and 3 in the figure).*

4.1.1 Trivial mapping functions

It seems natural to use as the mapping function the shape function corresponding to the shape of the domain. The mapping function is then the shape function of the finite element with the corresponding geometry. Thus, in the case where the domain "looks like" a triangle, the mapping function can be defined by:

$$F(\xi,\eta) = (1-\xi-\eta)\,a_1 + \xi\,a_2 + \eta\,a_3\,, \tag{4.1}$$

where a_i is the *corner* with index i (see hereafter) of the real domain. Obviously such a function only matches the corners of the domain. In other words, given a mesh on the logical triangle, the resulting mesh does not match the given boundary discretization unless the latter is composed of straight lines. As indicated, the above function is the shape function of the classical P^1 triangle (Chapter 20).

To improve the accuracy of the approximation of the domain, a more complicated shape function can be used, i.e., by reference to a more sophisticated finite element. For instance, the mapping function:

$$\begin{aligned} F(\xi,\eta) = \;& (1-\xi)(1-\eta)(1-2\xi-2\eta)\,a_1 + \xi(1-\eta)(2\xi-2\eta-1)\,a_2 \\ & -\xi\eta(3-2\xi-2\eta)\,a_3 + (1-\xi)\eta(-2\xi+2\eta-1)\,a_4 \\ & +4\xi(1-\xi)(1-\eta)\,a_5 + 4\xi\eta(1-\eta)\,a_6 + 4\xi(1-\eta)\eta\,a_7 \\ & +4(1-\xi)\eta(1-\eta)\,a_8\,, \end{aligned} \tag{4.2}$$

insures the respect of a boundary composed by arcs of parabola for a domain (Figure 4.2) similar to a quad (a_i is a the corners while a_{i+4} is an edge midpoints, for $i = 1,4$).

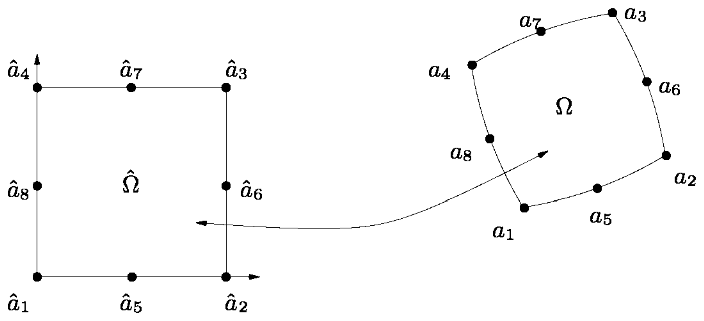


Figure 4.2: *Quadrilateral domain whose boundary is approximated by several arcs of parabola.*

Since the above functions, as well as the similar functions that can be easily found for the other shape analogies, do not guarantee sufficient properties, other types of functions must be developed.

4.1.2 Quadrilateral or triangular analogy

Quadrilateral analogy

In this case, the given discretization of the real domain can be considered as a set of four logical sides, each of which consists of a series of segments. Thus, an analogy with a quadrilateral is exhibited. The endpoints of the sides are the so-called *corners*, a_i, $i = 1, 4$, defined counterclockwise. Similarly, the domain is assumed to be on the "left-hand side" of the boundary which is also defined counterclockwise. The first side, defined from a_1 to a_2, consists of a series of n_1 segments. The second side, from a_2 to a_3 includes n_2 segments. For the sake of simplicity, the third side (a_4, a_3) is formed by n_1 segments while the fourth side (a_1, a_4) has n_2 segments. This means that the number of segments of the side discretization is the same for two logically opposite sides. In other words, two opposite sides must enjoy some similarity (for instance, in terms of length) so that only two *subdivision parameters*, n_1 and n_2, can be used.

Let $\phi_i(.)$ be the discretization of side i. In fact, $\phi_i(.)$ is defined as a series of straight segments joining two consecutive points of the given boundary discretization.

Logical mesh on the unit square boundary. Let a_j^i be the j^{th} point serving to define the discretization of the real side i with a_0^i the first corner of side i and a_n^i the other endpoint this side (where n stands for n_1 or n_2), then a discretization of side i of the unit square can be constructed in such a way as to conform to the discretization of the real corresponding side, in terms of relative distances from point to point. Let (omitting index i associated with the side under treatment)

$l_j = l_{a_j,a_{j+1}}$ be the distance between a_j and a_{j+1} for $j = 0, 1, ..., n-1$; then

$$l_{side} = \sum_{j=0}^{n-1} l_j \,,$$

with this notation, the j^{th} point on the logical side, say $\hat{a}_j$, is constructed on the relevant side by means of a formula like

$$\hat{a}_j = \frac{\sum\limits_{k=0}^{j-1} l_k}{l_{side}} \,.$$

This process is then repeated for the four sides, resulting in the desired points $\hat{a}^i_j\,, i = 1, 4$.

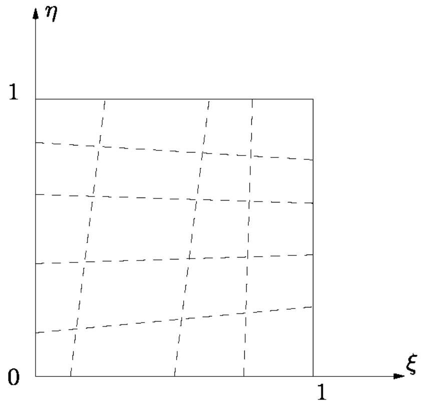


Figure 4.3: *Logical mesh on the unit square. The dotted lines show the two families of lines serving as a support for the construction.*

Logical mesh on the unit square. The above $\hat{a}^i_j$ are now used to define a simple mesh on the unit square. We define the lines joining two opposite points. Then considering a line joining side 1 and side 3 together with a line joining line 2 and side 4, an intersection point can be found. Actually, the intersection points result from the intersection of two families of lines. Applying this process for all the lines results in a valid mesh in the unit square. Both the connectivities (i.e., the element vertex connectivities and numbering) and the vertex positions of this logical mesh are trivially obtained.

Mapping onto the real domain. The aim is now to map the above logical mesh onto the real domain. To this end, a suitable mapping function, F, is needed. The question is how to construct such a function. Indeed, several choices are possible leading to different mesh generation methods. To help the choice, it is necessary to define what properties must be ensured. In this respect, we introduce three categories of properties which concern some suitable invariances and some degree of regularity.

- Pr_1: the image of a logical corner is a real corner. This is the well-known *corner identity.* For instance, $F(0,0) = a_1$.
- Pr_2: the image of a logical side matches the corresponding real side. Thus, $F(\xi, 0) \equiv \phi_1(\xi, 0) = \phi_1(\xi)$ (and similar expressions for the other sides).
- Pr_3: the image of a regular logical mesh is a regular real mesh. In fact, this means that a regular mesh is obtained for a real domain whose boundary discretization is uniform (for the two possible pairs of opposite sides). This leads to an *invariant property* of F when applied to the logical mesh itself. Then, $F(\xi, \eta) = (\xi, \eta)$, point of coordinates ξ and η is mapped onto itself.

We initially restrict ourselves to simple polynomials (in fact, forms of polynomials of first order for each of the variables). Then one possible method corresponds to:

$$F(\xi,\eta) = \frac{\sum_{i=1}^{4} \alpha_i\, \phi_i(\xi,\eta)}{\sum_{i=1}^{4} \alpha_i}\,, \tag{4.3}$$

where ξ and η live in $[0,1]$. The α_i's are functions of ξ and η defined as:

$$\alpha_1 = (1-\eta)\frac{1-\xi}{(1-\xi+\eta)}\frac{\xi}{(\xi+\eta)}\,, \qquad \alpha_2 = \xi\frac{\eta}{(1-\xi+\eta)}\frac{1-\eta}{(2-\xi-\eta)}\,,$$

$$\alpha_3 = \eta\frac{1-\xi}{(2-\xi-\eta)}\frac{\xi}{(1+\xi-\eta)}\,, \qquad \alpha_4 = (1-\xi)\frac{\eta}{(\xi+\eta)}\frac{1-\eta}{(1+\xi-\eta)}\,.$$

Remark 4.1 *Note that the sum*[1] *of the α_i's is 1. Thus, in the previous formula, the denominator can be removed.*

For this function, Pr_1 and Pr_2 are satisfied while Pr_3 does not hold. Thus, another type of function may be sought. For instance, the function

$$F(\xi,\eta) = \frac{\sum_{i=1}^{4} \beta_i\, \phi_i(\xi,\eta)}{\sum_{i=1}^{4} \alpha_i}\,, \tag{4.4}$$

where now:

$$\beta_1 = (\alpha_1+\alpha_3)(1-\eta)\,, \qquad \beta_2 = (\alpha_2+\alpha_4)\,\xi\,,$$

$$\beta_3 = (\alpha_1+\alpha_3)\,\eta\,, \qquad \beta_4 = (\alpha_2+\alpha_4)(1-\xi)\,.$$

Remark 4.2 *Obviously, the sum of the β_is is also equal to one.*

With this method, Pr_1, Pr_2 along with Pr_3 hold.

[1] A system like Mapple can be used to verify this property.

Proof. First, it is obvious to see that Pr_2 holds. For instance, for $\eta = 0$, $\alpha_2 = \alpha_4 = 0$, then $\beta_2 = \beta_4 = 0$ and $\beta_3 = 0$. Thus, $F(\xi, 0) = \beta_1\, \phi_1(\xi, \eta)$. As $\beta_1 = \alpha_1 + \alpha_3$, then $\beta_1 = \sum \alpha_i - \alpha_2 - \alpha_4 = 1$, which implies that $F(\xi, 0) = \phi_1(\xi, 0) = \phi_1(\xi)$.

Now, obviously Pr_1 holds. For instance, if $\xi = 0$, the above relation leads to $F(0,0) = \phi_1(0) = a_1$.

Property Pr_3 leads to checking whether the image of point (ξ, η) is this point when the real domain is the logical domain. So, we have to compute $\sum\limits_{i=1}^{4} \beta_i\, \phi_i(\xi, \eta)$.

In this particular case we have $\phi_1(\xi) = \begin{pmatrix} \xi \\ 0 \end{pmatrix}$, $\phi_2(\eta) = \begin{pmatrix} 1 \\ \eta \end{pmatrix}$, $\phi_3(\xi) = \begin{pmatrix} \xi \\ 1 \end{pmatrix}$ and $\phi_4(\eta) = \begin{pmatrix} 0 \\ \eta \end{pmatrix}$, then:

$$\sum_{i=1}^{4} \beta_i\, \phi_i(\xi, \eta) = \beta_1 \begin{pmatrix} \xi \\ 0 \end{pmatrix} + \beta_2 \begin{pmatrix} 1 \\ \eta \end{pmatrix} + \beta_3 \begin{pmatrix} \xi \\ 1 \end{pmatrix} + \beta_4 \begin{pmatrix} 0 \\ \eta \end{pmatrix}.$$

The first component of this expression is $\beta_1\, \xi + \beta_2 + \beta_3\, \xi$ whose value is:

$$\xi\,(1-\eta)\,(\alpha_1+\alpha_3) + \xi\,(\alpha_2+\alpha_4) + \xi\,(\alpha_1+\alpha_3)\,\eta \;=\; \xi\,(\alpha_1+\alpha_3) + \xi\,(\alpha_2+\alpha_4) \;=\; \sum_{i=1}^{4} \alpha_i\, \xi\,,$$

that is ξ. Similarly, the second component is η, thus:

$$\sum_{i=1}^{4} \beta_i\, \phi_i(\xi, \eta) = \begin{pmatrix} \xi \\ \eta \end{pmatrix}.$$

and the proof is completed. □

Another popular method is the *transfinite interpolation* which is defined in a slightly different manner. In fact, the corresponding function is not only a combination of the ϕ_i's but involves such a combination coupled with a correction term based on the corners. Note that a tensor product is used to define the main part of the expression and the correction term is then added to meet the desired properties:

$$\begin{aligned} F(\xi, \eta) \;=\; & (1-\eta)\,\phi_1(\xi) + \xi\,\phi_2(\eta) + \eta\,\phi_3(\xi) + (1-\xi)\,\phi_4(\eta) \\ & - ((1-\xi)(1-\eta)\,a_1 + \xi(1-\eta)\,a_2 + \xi\eta\,a_3 + (1-\xi)\,\eta a_4)\,. \end{aligned} \tag{4.5}$$

Exercise 4.1 *Show that function F satisfies the three above properties (hint: follow the scheme of the above proof).*

The mesh of the real domain is then easily obtained. Its connectivity is that of the logical mesh. The coordinates of its vertices are known by applying F to the above $\hat{a}^i_j$s.

The question is to decide which function F is the best among (4.3), (4.4) or (4.5). Clearly the function of Relation 4.3 is probably not so good. To select one of the two others, we have to look at the limitations of the method.

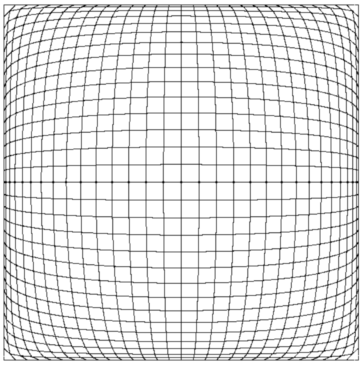
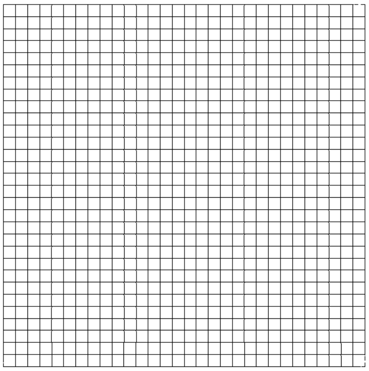



Figure 4.4: *Image of a uniform unit square when using the function of Relation (4.3), left-hand side, and that of Relation (4.4) or (4.5), right-hand side.*

Limitations. Unfortunately, the above method is unlikely to be suitable when handling complex four sided geometries. Actually, if the real domain is convex, the resulting mesh is valid, while for some non-convex domains, two complications may arise. First, the image of a point inside the logical mesh may fall outside the real domain and, on the other hand, the image of a valid quadrilateral in the logical mesh may be a quadrilateral with a zero or negative surface area, thus resulting in overlapping elements and an invalid mesh.

In other words, such a method is suitable only under some restrictive conditions about the shape of the real domain (in addition to its quadrilateral analogy).

Nevertheless, we would like to give some examples. In Figure 4.4, we consider how the three above functions act with regard to Pr_3. Then, as previously mentioned, Relation (4.3) is unlikely to be suitable. In Figure 4.5, we show the mesh obtained using methods (4.4) and (4.5) for a non-convex domain. Both meshes are wrong but it seems that method (4.4) is less sensitive to the non-convex geometry. Nevertheless, other non-convex geometries seem to indicate that the method of Relation (4.5) is more robust in most cases.

It can also be observed that it is not possible to enforce some orthogonality properties in the element edges (in contrast to the PDE methods presented below).

A final comment about this mesh generation method concerns its lack of flexibility. Indeed, as for all structured meshes, the type of connectivities from point to point results in a certain level of rigidity which means that it is not easy to deal with a problem where some flexibility (in terms of sizes, variations from region to region, etc.) is needed.

Variations. The two parameters n_1 and n_2 allow for some flexibility in the method. The restriction leading to only two parameters being considered can be, to some extent, over-passed. Indeed, it is possible to define four parameters, one for each side, and to define a more sophisticated method based on the same approach. Such a construction results in a mesh consisting of quadrilateral elements as well

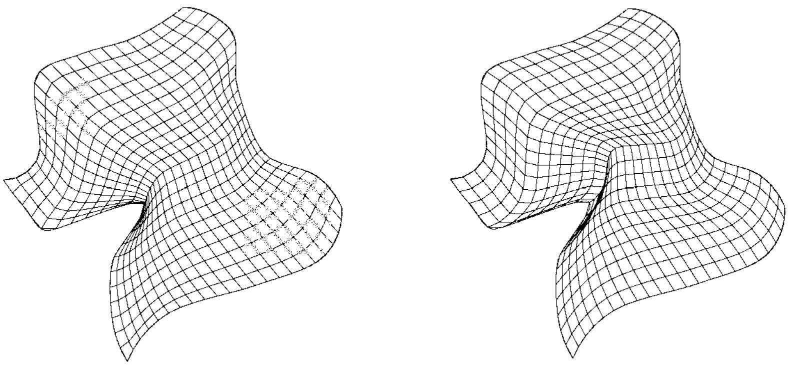


Figure 4.5: *Non-convex domain meshed using the function of Relation (4.4), left-hand side, and that of Relation (4.5), right-hand side.*

as some triangles to ensure a transition between the layers of elements.

Triangular meshing for a quadrilateral domain. In essence, this algebraic method completes a mesh whose elements are quadrilaterals except in the case where four parameters are defined. To meet a triangular mesh, the quads must be split into two triangles to maintain the same number of vertices. Such an operation is trivial at the time a criterion is selected to decide which diagonal is used to split a given element. In this respect, a given direction for the diagonals, the diagonal lengths, the element qualities, etc., can be chosen to govern the process. It should be noted that, in general, it is advisable to pay special attention to the elements that have one of the corners as a vertex. In this case, considering the diagonal that includes such a corner as endpoint is often a nice solution (see Chapter 18 about the notion of an over-constrained mesh).

Computational aspects. As for the computer implementation of the method (whatever the function F may be), several remarks can be made.

- First, one should note that the memory resources that are required can be easily known (or estimated) using the two (four) parameters defining the boundary discretization.
- The key idea to complete the logical mesh is the intersection of two families of lines, thus no difficulties are expected as such intersections are well defined (in contrast to the case of a triangular analogy, as shown below).
- Applying Relation (4.i) ($i = 3, 5$) is easy at the time the terms $\phi(.)$'s are computed. Actually $\phi(.)$ is known in a discrete manner. For instance, $\phi_1(\eta, 0)$ is the polygonal line written as $(a^1_1, a^1_2)\,(a^1_2, a^1_3) \dots (a^1_{n_1-1}, a^1_{n_1})$. This implies the use of an interpolation scheme. Thus, the computational process can be as follows:

  - Given $\hat{M}$ a vertex in the logical mesh with coordinates ξ and η, we pick the interval $(\hat{a}^1_j, \hat{a}^1_{j+1})$ within which falls ξ as well as the interval $(\hat{a}^3_k, \hat{a}^3_{k+1})$ (in terms of the third logical side). Similarly we find the two intervals (sides 2 and 4) corresponding to η.
  - We find the relationships between ξ and $\hat{a}^1_j$ and $\hat{a}^1_{j+1}$ and we map the same ratio between a^1_j and a^1_{j+1} (say the discrete form of ϕ_1). Similarly we find the three other relationships for the three other sides.
  - Then, the desired function is used by replacing the terms $\phi_j(.,.)$ by the above interpolations.

- Zero or negative surface area elements must be checked explicitly. To this end, associating four triangles with a given quad (i.e., by using the two diagonals, each of them making it possible to define two triangles) proves to be an efficient way to detect the cases where the method fails (see also Chapter 18).

As a consequence, implementing such a method, in its range of application, results in a rather inexpensive mesh generation method and, actually, is a quite easy task.

Triangular analogy

An analogy with a triangular shape is exhibited in the case where the given discretization of the real domain can be considered as a set of three logical sides (consisting of a series of segments). In addition, we assume that each side includes n segments, thus only one parameter is defined. Then the algebraic mesh generation method follows the same principle as for the previous case.

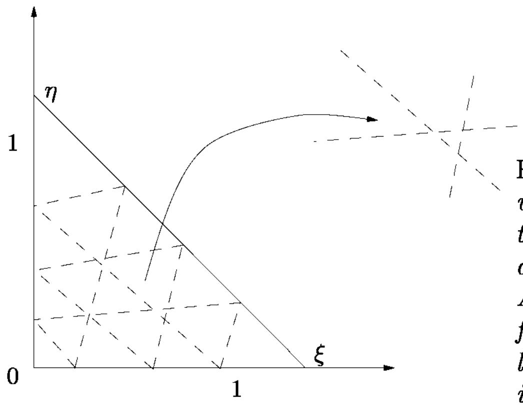


Figure 4.6: *Logical mesh on the unit triangle. In dotted lines are the three families of lines serving as a support for the construction. A close-up shows the region defined by the intersection of three lines that serves to find a unique intersection point.*

The boundary discretization is mapped onto the unit logical triangle. The logical corners $\hat{a}_1$, $\hat{a}_2$ and $\hat{a}_3$ are the points $(0,0)$, $(1,0)$ and $(0,1)$ and the three logical sides are defined as $\hat{a}_1 \longrightarrow \hat{a}_2$ for the first, $\hat{a}_2 \longrightarrow \hat{a}_3$ for the second and $\hat{a}_1 \longrightarrow \hat{a}_3$ for the third. Three families of lines are constructed. Points $\hat{a}^i_j, i = 1, 3$ are obtained by intersecting these lines. It should be noted that now the desired intersections are not well defined in the sense that the three relevant lines do not define a point as a solution but a small region. Nevertheless, a point, for instance, the centroid[2] of such a region, can be defined which allows us to reach what we are seeking. In this way the logical mesh is obtained.

To map the above mesh onto the real domain, we use the same idea as for the above quad method. In fact, several categories of functions can be used. For instance, a function may be used like:

$$F(\xi,\eta) = \frac{\sum\limits_{i=1}^{3} \alpha_i \, \phi_i(\xi,\eta)}{\sum\limits_{i=1}^{3} \alpha_i}, \tag{4.6}$$

where the α_is are the functions of ξ and η, defined as:

$$\alpha_1 = (1+\eta)\frac{1-\xi-\eta}{(1-\xi)}\frac{\xi}{(\xi+\eta)}, \quad \alpha_2 = (2-\xi-\eta)\frac{\xi}{(1-\eta)}\frac{\eta}{(1-\xi)},$$

$$\alpha_3 = (1+\xi)\frac{\eta}{(\xi+\eta)}\frac{1-\xi-\eta}{(1-\eta)}.$$

For this function, Pr_1 and Pr_2 are satisfied while Pr_3 does not hold.

Remark 4.3 *Note that the sum of the α_is is 1.*

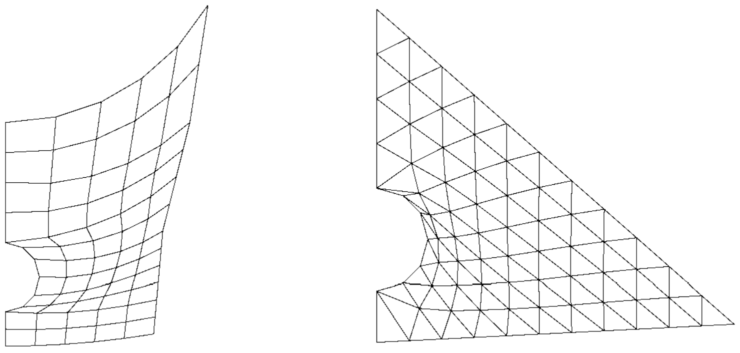


Figure 4.7: *Transfinite interpolation for a "quadrilateral" domain, left-hand side and same type of interpolation for a "triangular" domain, right-hand side.*

[2]In principle, the point minimizing the distances to the three lines is the best possible solution.

A form similar to Relation (4.5) is the following:

$$\begin{aligned}F(\xi,\eta) &= (1-\xi-\eta)\,(\phi_1(\xi)+\phi_3(\eta)) + \xi\,(\phi_1(\xi+\eta)+\phi_2(\eta)) \\ &+\eta\,(\phi_3(\xi+\eta)+\phi_2(1-\xi)) \;\; - ((1-\xi-\eta)\,\phi_1(0)+\xi\,\phi_2(0)+\eta\,\phi_3(0))\,,\end{aligned} \tag{4.7}$$

while a form involving rational polynomials is:

$$\begin{aligned}F(\xi,\eta) &= \tfrac{1-\xi-\eta}{1-\xi}\,\phi_1(\xi) + \tfrac{\xi}{1-\eta}\,\phi_2(\eta) + \tfrac{\eta}{\xi+\eta}\,\phi_3(\xi+\eta) \\ &- (\tfrac{\eta}{\xi+\eta}(1-\xi-\eta)\,a_1 + \tfrac{1-\xi-\eta}{1-\xi}\xi\,a_2 + \tfrac{\xi}{1-\eta}\eta\,a_3)\,.\end{aligned} \tag{4.8}$$

The previous functions satisfy the three desired properties. Limits and variations of the present method are of the same nature as for the above case. Computational issues do not lead to major difficulties.

Application examples

In this section, two simple application examples are provided where the domains in question are not strictly convex. Nevertheless, Figure 4.7, the resulting meshes are valid. The last example, Figure 4.8, concerns a case where the (transfinite interpolation) method fails. Indeed, due to the geometry of the domain, some overlapping elements are constructed (i.e., negative elements exist). To some extent, it is possible to untangle such an invalid mesh so as to obtain a suitable solution. For instance, in this test example, successive iterations of point relocations allow the mesh to be corrected.

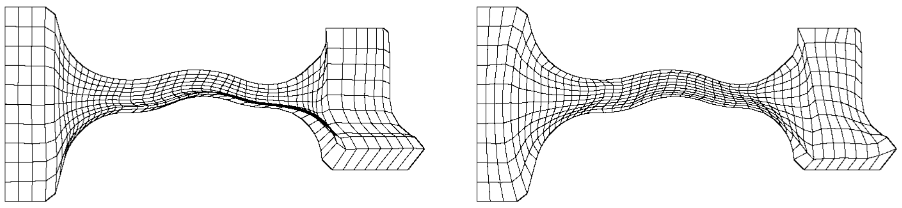


Figure 4.8: *Domain inducing degeneracies (overlapping elements are present), left-hand side, and mesh resulting from local corrections, right-hand side.*

4.1.3 Surface meshing

The above two types of methods also provide a way to mesh a surface following the desired analogy which is defined by a discretization of its four (three) boundary edges. It should be noted that the resulting surface is only controlled by its boundary discretization. Thus, while rather easy to implement, such a method may result in a poor approximation of the real surface or even may result in undesirable twists or folds (see Chapters 13 and 15 where the transfinite interpolation is used for surface definition).

4.1.4 Hexahedral, pentahedral or tetrahedral analogy

Methods similar to those previously described can be defined so as to handle three-dimensional domains that can be considered as analogous to a simple logical domain.

Hexahedral analogy

In this case, the real domain is assumed to be similar to a cuboid. Actually, eight corners can be identified as well as six faces which are similar to quadrilateral faces. To follow a mesh generation method similar to those in two dimensions, the discretization of the faces is assumed to be of the structured type. For a quad face, a typical discretization is a grid defined by two subdivision parameters. For a triangular face (see hereafter the pentahedral or the tet analogies), only one subdivision parameter is supposed. Thus, provided $\phi_i(.,.,.)$, $(i = 1,6)$, the adequate discretization of these faces, the logical mesh of the unit cube can be mapped on the real domain using the following function:

$$F(\xi,\eta,\zeta) = \frac{\sum_{i=1}^{6} \alpha_i\, \phi_i(\xi,\eta,\zeta)}{\sum_{i=1}^{6} \alpha_i}, \tag{4.9}$$

where the α_is are the following functions of ξ,η,ζ:

$$\alpha_i = \overline{x}_{i-1}\frac{x_i}{(x_i + x_{i-1})}\frac{\overline{x}_i}{(\overline{x}_i + x_{i-1})}\frac{x_{i+1}}{(x_{i+1} + x_{i-1})}\frac{\overline{x}_{i+1}}{(\overline{x}_{i+1} + x_{i-1})},$$

and

$$\alpha_{i+3} = x_{i-1}\frac{x_i}{(x_i + \overline{x}_{i-1})}\frac{\overline{x}_i}{(\overline{x}_i + \overline{x}_{i-1})}\frac{x_{i+1}}{(x_{i+1} + \overline{x}_{i-1})}\frac{\overline{x}_{i+1}}{(\overline{x}_{i+1} + \overline{x}_{i-1})},$$

where $i = 1,3$ with $x_{j+1} = \xi, \eta$ or ζ, $j = 0,2$ ($i + j$ *modulo* 3) and $\overline{x}_j = 1 - x_j$. As before, this function satisfies the "corner identities" and matches the boundary.

Proof. First, the following properties can be verified (held for $i = 1,3$ and $j = 1,6$). First, the α_i's are such that:

- $x_{i-1} = 0 \longrightarrow \alpha_i = 1$ and $\alpha_j = 0$ for $j \neq i$,
- $x_{i-1} = 1 \longrightarrow \alpha_{i+3} = 1$ and $\alpha_j = 0$ for $j \neq i$.

Thus, we have:

- $F(0,0,0) = a_1$ where a_1 is the first corner and we have similar relations for the other corners (i.e., property Pr_1 holds).
- $F(\xi,\eta,0) = \phi_1(\xi,\eta,0)$ where ϕ_1 stands for the discretization of the real face identified to the first face of the logical cube (i.e., the face $\zeta = 0$), etc. Then, property Pr_2 holds.

- $\alpha_i(\overline{x}_{i-1}) = \alpha_{i+3}(x_{i-1})$, $\alpha_i(\overline{x}_i) = \alpha_i(x_i)$ and $\alpha_i(\overline{x}_{i+1}) = \alpha_i(x_{i+1})$, then Relation (4.9) is symmetric in some sense, faces ϕ_i and ϕ_{i+3} act in a symmetric way when $\overline{x}_{i-1}$ replaces x_{i-1}. Indeed, we return to the discussion about Relation (4.4). While some symmetric features exist, property Pr_3 is not satisfied. □

As for the quad analogy, a method can be easily derived so as to ensure Pr_3. The idea is the same, we take as a function the following:

$$F(\xi,\eta,\zeta) = \frac{\sum\limits_{i=1}^{6} \beta_i\,\phi_i(\xi,\eta,\zeta)}{\sum\limits_{i=1}^{6} \alpha_i}\,, \tag{4.10}$$

with (for $i = 1, 3$):

$$\beta_i = (\alpha_i + \alpha_{i+3})\,\overline{x}_{i-1} \quad \beta_{i+3} = (\alpha_i + \alpha_{i+3})\,x_{i-1}\,.$$

The transfinite interpolation for the hex analogy is as follows:

$$\begin{aligned} F(\xi,\eta,\zeta) \ = \frac{1}{2} \ & \big[(1-\zeta)\,\phi_1(\xi,\eta) + (1-\eta)\,\phi_2(\xi,\zeta) + (1-\xi)\,\phi_3(\eta,\zeta) \\ & +\zeta\,\phi_4(\xi,\eta) + \eta\,\phi_5(\xi,\zeta) + \xi\,\phi_6(\eta,\zeta) \\ & -\big((1-\xi)(1-\eta)(1-\zeta)\,a_1 + \xi(1-\eta)(1-\zeta)\,a_2 \\ & +\xi\eta(1-\zeta)\,a_3 + (1-\xi)\eta\zeta\,a_4 + (1-\xi)(1-\eta)\zeta\,a_5 \\ & +\xi(1-\eta)\zeta\,a_6 + \xi\eta\zeta\,a_7 + (1-\xi)\eta\zeta\,a_8\,\big)\big]\,. \end{aligned} \tag{4.11}$$

Then, Pr_1, Pr_2 as well as Pr_3 are satisfied.

Proof. First, it is obvious that Pr_1 holds. Regarding Pr_2, the proof needs to have the ϕ_is defined by a transfinite interpolation. To show Pr_3, we use the same method as before. I.e., we prove that the image of point (ξ,η,ζ) is invariant due to the peculiar form of the ϕ_is. □

Pentahedral analogy

In this case, the same method can be followed with a mapping function like:

$$F(\xi,\eta,\zeta) = \frac{\sum\limits_{i=1}^{5} \alpha_i\,\phi_i(\xi,\eta,\zeta)}{\sum\limits_{i=1}^{5} \alpha_i}\,, \tag{4.12}$$

where the α_i are defined by:

$$\alpha_1 = (1-\zeta)\frac{1-\xi-\eta}{(1-\xi-\eta+\zeta)}\frac{\xi}{(\xi+\zeta)}\frac{\eta}{(\eta+\zeta)}\,,$$

$$\alpha_2 = (1-\xi)\frac{\zeta}{(\xi+\zeta)}\frac{1-\zeta}{(1-\zeta+\xi)}\frac{1-\xi-\eta}{(1-\eta)}\frac{\eta}{(\xi+\eta)},$$
$$\alpha_3 = (1-\eta)\frac{\zeta}{(\eta+\zeta)}\frac{1-\zeta}{(1+\eta-\zeta)}\frac{\xi}{(\xi+\eta)}\frac{1-\xi-\eta}{(1-\xi)},$$
$$\alpha_4 = \zeta\frac{\xi}{(1+\xi-\zeta)}\frac{\eta}{(1+\eta-\zeta)}\frac{1-\xi-\eta}{(2-\xi-\eta-\zeta)}$$
$$\alpha_5 = (\xi+\eta)\frac{\zeta}{(\zeta+1-\xi-\eta)}\frac{1-\zeta}{(2-\xi-\eta-\zeta)}\frac{\xi}{(1-\eta)}\frac{\eta}{(1-\xi)}.$$

It should be noted that $\alpha_1(1-\zeta) = \alpha_4(\zeta)$, which means that faces one and four play a special role (in fact, they correspond to the logical faces $\zeta = 0$ and $\zeta = 1$). With this definition, Pr_1 and Pr_2 hold while Pr_3 is not achieved. Note that defining a function ensuring Pr_3 is a tedious task.

Tetrahedral analogy

For a tetrahedral analogy, the following function could be retained:

$$F(\xi,\eta,\zeta) = \frac{\sum_{i=1}^{4} \alpha_i\,\phi_i(\xi,\eta,\zeta)}{\sum_{i=1}^{4} \alpha_i} \tag{4.13}$$

where the α_i are defined by:

$$\alpha_i = (1-x_{i-1})\frac{x_i}{(x_i+x_{i-1})}\frac{x_{i+1}}{(x_{i+1}+x_{i-1})}\frac{x_{i+2}}{(x_{i+2}+x_{i-1})}$$

for $i = 1,4$ with $x_{j+i} = \xi,\eta,\zeta$ or $1-\xi-\eta-\zeta$ for $j = 0,3$ ($i+j$ *modulo* 4). In fact, we encounter the barycentric coordinates (which can be used in this case, as we are considering a simplex). In other words, we have:

$$\alpha_1 = (1-\zeta)\frac{\xi}{(\xi+\zeta)}\frac{\eta}{(\eta+\zeta)}\frac{1-\xi-\eta-\zeta}{(1-\xi-\eta)},$$
$$\alpha_2 = (1-\xi)\frac{\eta}{(\eta+\xi)}\frac{1-\xi-\eta-\zeta}{(1-\eta-\zeta)}\frac{\zeta}{(\zeta+\xi)},$$
$$\alpha_3 = (1-\eta)\frac{1-\xi-\eta-\zeta}{(1-\xi+\zeta)}\frac{\zeta}{(\zeta+\eta)}\frac{\xi}{(\xi+\eta)},$$
$$\alpha_4 = (\xi+\eta+\zeta)\frac{\zeta}{(1-\xi-\eta)}\frac{\xi}{(1-\eta-\zeta)}\frac{\eta}{(1-\xi-\zeta)}.$$

Thus, Pr_1 and Pr_2 hold while Pr_3 is not satisfied. Note that it is not possible to partition a tetrahedron into uniform sub-tetrahedra (unlike the same problem in two dimensions, for a triangle). Nevertheless, it could be of interest to have the corresponding uniform distribution of points. However, exhibiting a function ensuring Pr_3 is, *a priori*, not obvious.

4.1.5 Other algebraic methods and alternative methods

Other algebraic methods

The above algebraic methods only infer point coordinates, in this sense, they are of a Lagrangian type. Thus, no way is provided to control directional features such as orthogonality (for instance, at the boundary level). Different algebraic methods can be investigated to enable such a control. For instance, a Hermitian type method can be defined involving not only point coordinates but also some derivatives. We will see in Chapter 13 the Coons patches which are based on a method that can also be used in this mesh generation context.

An alternative to algebraic methods

We consider in detail the case of a three-dimensional method. Apart for the hexahedral analogy, the two other shape analogies (tet and pentahedral) can be dealt with by the previously described algebraic method. Nevertheless, while simple to describe, these methods are not so easy to implement and most likely do not guarantee property Pr_3. Thus, a different mesh generation principle can be advocated. To start the meshing process, we follow the first three steps of a classical algebraic method, i.e.,

- construct a discretization on the boundary of the logical domain in accordance with the given discretization of the boundary of the real domain,
- complete the mesh of the logical domain following the above boundary definition,
- map the logical mesh onto the real domain by means of the classical P^1 interpolation scheme corresponding to the shape analogy.

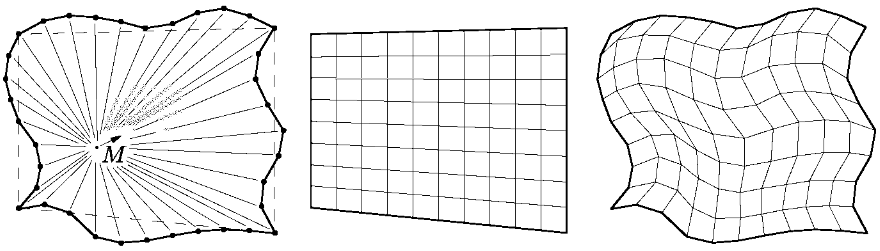


Figure 4.9: *Deformation governed by the boundary points, (left-hand side). Straight mesh (middle) and resulting mesh (right-hand side).*

The resulting mesh is then modified using a deformation technique. If M is a point of the current mesh, we compute a new location for M as follows

$$M = M + \frac{1}{\alpha} \sum_{a_k \in \Gamma} \omega_k(M)\, \alpha_k\, def(a_k) \tag{4.14}$$

-	np	ne	t	v_p	v_e
cas 1	2,024	9,261	1.10	1,840	8,419
case 2	5,984	29,791	6.56	912	4,541
case 3	2,601	4,096	1.24	2,097	3,303
case 4	5,566	9,261	4.51	1,234	2,053
case 5	16,896	29,791	31.27	540	952
case 6	2,744	2,197	1.68	1,633	1,307
case 7	17,576	15,625	44.08	398	354

Table 4.1: Statistics related to some selected examples (the first two lines concern a tet analogy while the following three concern a pentahedral analogy and the last two concern a hexahedral analogy).

where the a_ks are the members of Γ the boundary of the domain (i.e., the a_ks are the boundary vertices of the real domain) and the quantities involved in the formula are:

- $def(a_k)$, the distance between the (real) point a_k and the image of the corresponding $\hat{a}_k$,
- α_k is a weight associated with a_k. Actually, the average length of the edges sharing a_k acn be computed,
- α, a normalization factor defined as

$$\alpha = \sum_{a_k \in \Gamma} \omega_k(M)\, \alpha_k\,, \tag{4.15}$$

- $\omega_k(M)$, a coefficient associated with a_k, is defined as $d_k^{-\beta}(M)$ where β is a value of attraction (for instance, 4) and $d_k(M)$ is the distance between the image of $\hat{a}_k$ and M.

Now, note that $def(a_k)$ vanishes at the corners[3].

Computational issues. While almost quadratic, the above algorithm has proved to be robust enough to carry out some non-trivial geometries. Nevertheless, a "too non-convex" domain will be quite difficult to handle with success.

The theoretical analysis of this method is quite easy. Actually, each point creation needs the analysis of the members of Γ. Thus, for a hexahedral domain whose subdivision parameters are equal ($n_1 = n_2 = n_3 = n$), the cardinality of Γ is of the order of $(n+2)^2$, i.e., n^2, while the number of points is something like n^3. Then, the number of operations is in n^5 resulting in a complexity in $n^{\frac{5}{3}}$, say $\mathcal{O}(n^{\frac{5}{3}})$, n now being the number of vertices. As a consequence, the method is in essence almost quadratic.

[3]Due to the use of the interpolation function which preserves the corner identity.

In Table 4.1, np is the number of vertices, ne is the number of elements, t is the CPU time in seconds while v_p indicates the number of points dealt with within one second and v_e the number of elements created in the same time. The ratio $\left(\frac{np_2}{np_1}\right)^{\frac{5}{3}}$ between t_2 and t_1 could be observed where t_i $(i = 1, 2)$ is the time required for creating np_i points. While the behavior is the same for the three types, the global efficiency is related to the number of members in Γ compared with the number of points (thus, the constant from case to case is rather different). Indeed, this constant is related to the number of face vertices of the logical element as compared with the number of vertices of the real mesh.

Extensions. It should be noted that the above meshing method by means of deformations is also a way to update a given mesh whose boundary discretization moves from one step to another of a given iterative process.

Remark 4.4 *The same method applies in two dimensions. In this case, it is very easy to define. Moreover, its complexity remains reasonable as it is something like* $n^{\frac{3}{2}}$, *say* $\mathcal{O}(n^{\frac{3}{2}})$.

4.2 PDE-based methods

PDE mesh generation methods represent an elegant alternative to algebraic methods and may be used when the domain (Ω with boundary denoted by Γ hereafter) can be identified by a quad (in two dimensions), or a cuboïd (in three dimensions). The major reference for PDE type methods is [Thompson *et al.* 1985]. Contrary to any algebraic methods, a transformation from the domain to this quad (cuboid), the logical domain, is sought. A *generation system* is associated with such a transformation, which allows us to compute the required mesh.

4.2.1 Basic ideas

In what follows, variables x, y, (resp. x, y, z) describe the domain (Figure 4.10) while the logical region is described using variables ξ, η, (resp. ξ, η, ζ). The problem now becomes one of finding the functions

$$x = x(\xi, \eta) \quad \text{and} \quad y = y(\xi, \eta) ,$$

or

$$x = x(\xi, \eta, \zeta) , \quad y = y(\xi, \eta, \zeta) , \quad \text{and} \quad z = z(\xi, \eta, \zeta) ,$$

according to the spatial dimension, assuming that the transformation maps the logical region one-to-one onto the domain and that the boundaries are preserved.

The one-to-one property is ensured by requiring that the Jacobian of the transformation is non-zero. The transformation (for example, in two dimensions) is defined by the matrix:

$$[\mathcal{M}] = \begin{pmatrix} x_\xi & x_\eta \\ y_\xi & y_\eta \end{pmatrix}$$

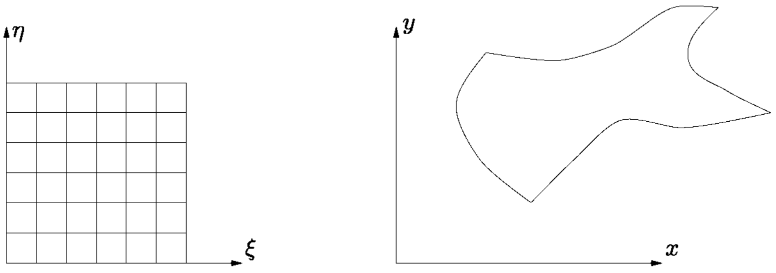


Figure 4.10: *Logical domain (a unit square), left-hand side, and real domain, right-hand side.*

where x_ξ stands for $\frac{\partial x}{\partial \xi}$, x_η stands for $\frac{\partial x}{\partial \eta}$, and so on. The Jacobian J is $x_\xi y_\eta - x_\eta y_\xi$. As it is assumed to be non-zero, the inverse of the transformation exists and variables ξ, η can be expressed in terms of x, y as follows:

$$\xi = \xi(x,y) \quad \text{and} \quad \eta = \eta(x,y)$$

The two ways of expressing the variables are mathematically equivalent and lead to two possibilities for solving the problem. If variables ξ, η are expressed in terms of x, y, the logical mesh can be transformed into a mesh on the domain, and the physical problem is solved in the domain as in the classical way. On the other hand, if variables x, y are expressed in terms of ξ, η, either the physical problem can be written in terms of these variables and then solved in the logical region, or we return to the above classical solution.

As a simple example of PDE methods, we briefly consider the following generation system based on the regularizing properties of the Laplacian operator Δ. We consider the two following systems:

$$\left\{ \begin{array}{ccc} \xi_{xx} + \xi_{yy} & = & 0 \text{ in } \Omega, \\ \text{Boundary conditions} & \text{on} & \partial\Omega \end{array} \right. \tag{4.16}$$

and

$$\left\{ \begin{array}{ccc} \eta_{xx} + \eta_{yy} & = & 0 \text{ in } \Omega, \\ \text{Boundary conditions} & \text{on} & \partial\Omega \end{array} \right. \tag{4.17}$$

which are then inverted in order to find $x(\xi, \eta)$ and $y(\xi, \eta)$, thus we obtain as the generation system:

$$\left\{ \begin{array}{lcl} g_{11} x_{\xi\xi} + g_{22} x_{\eta\eta} + 2 g_{12} x_{\xi\eta} & = & 0. \\ g_{11} y_{\xi\xi} + g_{22} y_{\eta\eta} + 2 g_{12} y_{\xi\eta} & = & 0. \end{array} \right. \tag{4.18}$$

with

$$g_{ij} = \sum_{m=1}^{2} A_{mi} A_{mj}$$

where $A_{mi} = (-1)^{i+m}(\text{Cofactor}_{m,i} \text{ of } [\mathcal{M}])$ and $[\mathcal{M}]$ is the above matrix.

This results in a system expressed in the logical space (where a mesh exists). It is a non-linear coupled system which can be solved using relaxation techniques or, more generally, iterative methods after an initialization by a solution in which the real boundary conditions are prescribed.

Variants. Variants of the previous generation systems can be experimented with to obtain special properties. For example, adding a non-zero right-hand side and considering:

$$\left\{\begin{array}{ccc} \xi_{xx} + \xi_{yy} & = & P \\ \text{Boundary conditions} & & \end{array}\right. \tag{4.19}$$

along with

$$\left\{\begin{array}{ccc} \eta_{xx} + \eta_{yy} & = & Q \\ \text{Boundary conditions} & & \end{array}\right. \tag{4.20}$$

enables us to control the distribution of points inside the domain. In this situation, the inverse system is:

$$\left\{\begin{array}{lcl} g_{11}x_{\xi\xi} + g_{22}x_{\eta\eta} + 2g_{12}x_{\xi\eta} + J^2(Px_\xi + Qx_\eta) & = & 0, \\ g_{11}y_{\xi\xi} + g_{22}y_{\eta\eta} + 2g_{12}y_{\xi\eta} + J^2(Py_\xi + Qy_\eta) & = & 0. \end{array}\right. \tag{4.21}$$

using the same notation and J, the Jacobian, being defined by $J = det([\mathcal{M}])$.

The right-hand sides P and Q interact as follows:

For $P > 0$, the points are attracted to the "right", $P < 0$ induces the inverse effect.

For $Q > 0$ the points are attracted to the "top", $Q < 0$ leads to the inverse effect.

Close to the boundary, P and Q induces an inclination of lines $\xi = constant$ (or $\eta = constant$).

P (Q) can also be used to concentrate lines $\xi = constant$ or $\eta = constant$ towards a given line, or to attract them towards a given point. To achieve this, the right-hand side can be defined as follows:

$$P(\xi,\eta) = -\sum_{i=1}^{n} a_i sign(\xi - \xi_i)e^{-c_i|\xi-\xi_i|} - \sum_{i=1}^{m} b_i sign(\xi - \xi_i)e^{-d_i\left[(\xi-\xi_i)^2+(\eta-\eta_i)^2\right]^{\frac{1}{2}}}$$

where n and m denote the number of lines in ξ and η of the grid. Such a control function induces the following:

for $a_i > 0$, $i = 1, n$, lines ξ are attracted to line ξ_i,

for $b_i > 0$, $i = 1, m$, lines ξ are attracted to point (ξ_i, η_i).

These effects are modulated by the amplitude of a_i (b_i) and by the distance from the attraction line (attraction point), modulated by coefficients c_i and d_i. For $a_i < 0$ $(b_i < 0)$, the attraction is transformed into a repulsion. When $a_i = 0$ $(b_i = 0)$, no particular action is connected to line ξ_i or point (ξ_i, η_i).

A right-hand side Q of the same form produces analogous effects with respect to η by interchanging the roles of ξ and η.

The major difficulty for automating this class of mesh generation systems is how to choose the control functions (P, Q, etc.) and the parameters they involve. However, these methods can be extended to three dimensions and, for a complete discussion, the reader is referred to [Thompson-1982a], where other forms of right-hand sides P and Q producing other properties are discussed (for example, the concentration of lines ξ or η towards an arbitrary line and not only towards a particular one ($\xi = constant$ or $\eta = constant$) or towards a given point to increase the mesh density near this point). In the above mentioned reference, and some others (for instance, [Knupp, Steinberg-1993]), other types of generation systems, including parabolic and hyperbolic operators, are discussed and numerous examples are provided.

How to define the quadrilateral (cuboid) analogy. When using a generation method of the present class, it is convenient to find the best analogy from the domain to a logical shape (quadrilateral or cuboïd). Such an analogy is often obtained either by partitioning the domain into several simpler domains, or by identifying the domain with the required shape, using several methods.

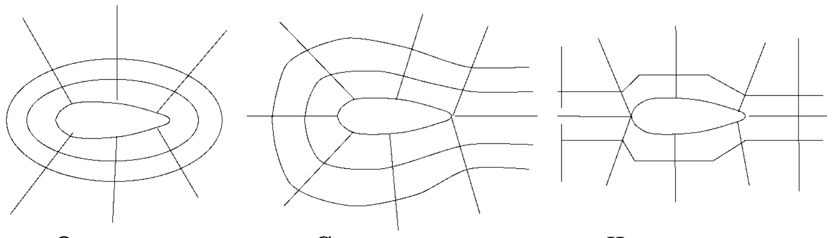


Figure 4.11: *O-type, C-type and H-type decompositions.*

For example, in two dimensions, there are several major classes of decompositions of the domain under consideration from which different kinds of grids will result in order to capture the physics of the problem as well as possible. In this respect, a domain can be discretized following an O-type, C-type or H-type analysis (Figure 4.11). To obtain such an analogy, artificial cuts must be introduced. Such analogies extend to a greater or lesser degree to three dimensions.

4.2.2 Surface meshing and complex shapes

PDE-based methods can also be employed to generate surface meshes or domains having more complex geometries as local mesh generation processes (as will be discussed below in the multiblock method).

4.3 Multiblock method

As algebraic methods and PDE methods are unlikely to be suitable for complex geometries, other methods must be used to deal with such geometries. Multiblock type methods are an initial answer to this problem. The underlying idea of such methods is to take advantage of a local meshing process such as an algebraic or a PDE meshing algorithm and to overpass its limitations by applying it in its successful range of applications.

4.3.1 Basic ideas

Provided with a local meshing process (of the algebraic or PDE type) the aim is to split the geometry in terms of regions where the local meshing process applies. Thus, in two dimensions, a domain is decomposed in terms of convex (or not too deformed) triangles or quadrilaterals (when an algebraic method is used) or in terms of quadrilaterals only (when a PDE method is involved). In three dimensions, the partitions that can be handled successfully are made of tetrahedral, pentahedral or hexahedral regions (when algebraic methods are considered) or hexahedral regions only (when PDE methods are used).

Thus the key point is to obtain such a suitable partition. Two kinds of partitions may be considered. The first type considers a partition to be *conformal* in itself while the second does not require such a property (Figure 4.12), see Chapter 3 where three categories of multiblock methods are introduced. In what follows, we consider a composite method which is more demanding, in terms of continuity, at the block interfaces. Note that patch or overlapping type methods are less demanding in this respect but, for some aspects, can be based on what is described below.

Conformal partitions lead to a very simple method as the union of the meshes of two distinct members of the partition automatically results in a conformal mesh. Otherwise some care must be taken to partition the domain, unless if a non conformal mesh is desired, as it is for some solution methods. For instance, in the example in Figure 4.12 (right), the continuity between the lower line of the upper region with the upper lines of the two lower blocks must ne enforced.

Obtaining a partition is a tedious task and generally it is the user who must undertake the task, which means that automating such a process is not so easy (see Chapter 9).

Some constraints must be considered to construct the partition of the whole domain, especially when a conformal partition is expected. A member of the partition is called a block[4] or a super-element, and some consistency is needed

[4]And as several blocks are created, the method is called a multiblock method.

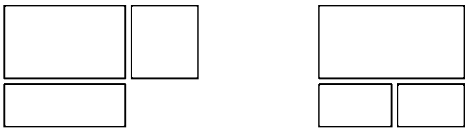


Figure 4.12: *Conformal and non-conformal decomposition of a two-dimensional domain. Three quad regions are defined which form a conformal partition (left-hand side) and a non-conformal partition (right-hand side).*

from block to block (or super-element to super-element). Thus, the problem is one of finding an adequate partition into blocks or super-elements such that the interfaces between the blocks are consistently defined.

4.3.2 Partitioning the domain

This task is done by the user and, in this sense a multiblock method can be regarded as a *semi-automatic method.* The aim of this task is to define the different blocks necessary to define the domain in such a way that each block is *a priori* suitable for the local meshing process which will be applied to it. At the same time, some consistency must be ensured from block to block. Moreover, while following these constraints, an accurate approximation of the geometry must be obtained together with a control of the nature of the expected mesh (in terms of the number of elements, element sizes, etc.). To this end, both the way in which the partition is defined and the choice of the different subdivision parameters must be properly carried out.

To illustrate this multiple aspect, we consider a problem in two dimensions where two kinds of local meshing processes are available, one capable of carrying out triangular regions, the other suitable for quad regions (see Figure 4.13 where one can see how the subdivision parameters of the various blocks are related to one another). There remain two subdivision parameters, one for the triangle analogy and another one for the quad regions. The problem is now to choose these parameters appropriately in order to define a suitable partition.

Geometric aspect. This step is motivated by two goals. First, the geometry of the domain must be well approached by the blocks and, on the other hand, the shape of the blocks must be as close as possible to a convex region (to insure, in addition, a successful application of the local meshing process). With regard to these two aspects, a certain number of blocks must be constructed. A block is described by its corners, its edges (and its faces, in three dimensions). While several blocks may be useful, care must be taken of the interfaces from block to block. In two dimensions, this leads to defining the block edges in such a way as to insure a nice continuity. In three dimensions, faces from block to block must be carefully defined.

A pertinent choice of the number of blocks, together with that of the possible subdivision parameters, enables us to obtain a good approximation of the geome-

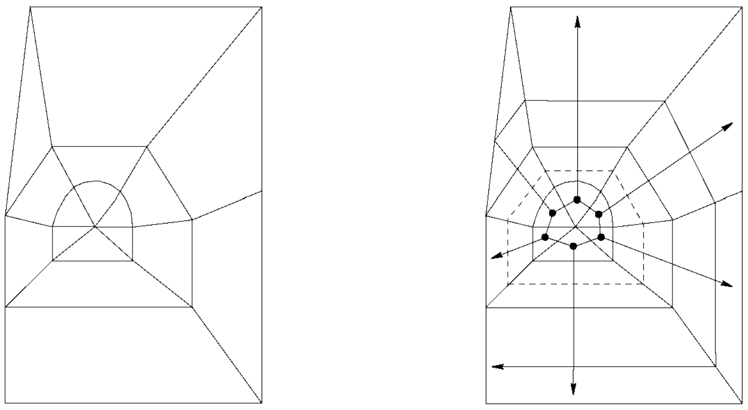


Figure 4.13: *Definition of the blocks defining the partition (left-hand side). Relationships between the subdivision parameters so as to enforce the continuity (right-hand side).*

try. In the regions with high curvature, several blocks or a fine discretization (i.e., a large enough subdivision parameter) is one solution to suit the geometry.

On completion to this step, several blocks are available in which corners, edges and faces are known.

Conformal and consistency requirements. The previous coarse partition can, in some cases, be refined to insure both the conformal aspect of the partition and the consistency between the items which are logically connected. Indeed, a subdivision parameter is associated with the block edges, but some edges are connected (in the same block and from block to block), so the number of possible subdivision parameters can be reduced. See again Figure 4.13 where we have only two possible different subdivision parameters.

4.3.3 Computational issues and application examples

Given the previous analysis, the blocks are now well defined. This means that we have defined the necessary corners, edges and faces which are the constitutive entities of the blocks. Now, we have to consider all these entities in a global way. Thus, a possible synthetic scheme of a multiblock method is as follows:

Step 1: corner definition. This stage involves a global numbering of the corners of the different blocks so as to avoid a local numbering (when only a given block is considered) but a global corner numbering that could be used subsequently.

Step 2: edge definition and meshing. The edges are defined by their two endpoints (which are corners as previously introduced) and a subdivision parameter (n). Then, according to the meshing capabilities, each edge is split into $n+1$ segments (meaning that n intermediate points are created along it). In terms of point locations, we face a curve meshing problem (see Chapter 14). In terms of point numbering, we have, as above, to define a global numbering of the thus created vertices.

Step 3: face definition and meshing. The faces are defined by their edges. Following the type of the face (triangle or quad) and according to the different subdivision parameters, the face is meshed using, for instance, an algebraic method or a similar surface mesh generation method which completes a structured mesh. The resulting mesh is then of a nature that will enable us to continue the process (i.e., the surface meshes are of the structured type). As above, a global numbering of the created vertices must be done.

Step 4: block definition and meshing. The blocks are defined by their faces and then, the local meshing process is applied. The numbering of the internal vertices can be then made sequentially starting from the first available index, i.e., the last number of the last face vertices plus 1.

Step 5: global mesh construction. Actually, this step is automatically completed since a global numbering of the vertices has been developed in the previous steps.

Thus the idea is to process all the corners, all the edges and all the faces before processing the mesh of the different blocks. Relationships between corners, edges, faces and parent blocks are to be properly defined. In peculiar, a given edge (face) shared by several blocks must be defined in a *global* and *unique* way but, from a block point of view, can appear in various ways. Thus, some flags are needed to insure consistency between the global definition (which is unique) and the different local definitions (which can vary based on the block considered).

Global definition of the corners. The corners are introduced to match the previous requirements (geometry, consistency, etc.). An index (for instance, starting from index 1 for the first corner of the first block) is associated with each corner.

Global definition of the edges and faces. Once the corners are given (let np_c be the number of such corners), the edges must be defined in a global way. One solution is, when considering an edge, say ab where a and b are the two endpoint indices, to define the edge as ab if $a < b$ and as ba otherwise.

Then, when visiting an element like abc, we encounter edge ab whereas when looking at element dba, we encounter edge ba. In terms of edge definition, edge ab as well as edge ba must be uniquely defined. Note that the previous convention allows this to be done.

The global definition of the faces follows a similar rule. Let us consider a quad face whose endpoints are a, b, c and d. These indices are sorted from the smallest to the largest and this new series of indices is the global definition of the face. In fact, if a is the smallest index among the four face indices, we consider the index of point b (which is next to a) and that of point d (which precedes a); then

- if $d < b$, use ad as a basis for global numbering,
- otherwise, use ab (see below).

Definition of blocks in terms of corners, edges and faces. The point is to identify the corners, edges and faces of the various blocks so as to return to the global definition of these entities.

Let us consider a hexahedral block involving three subdivision parameters, n_i $(i = 1, 3)$. The block definition involves 8 corners, 12 edges and 6 quad faces. The issue is to find the proper correspondence between these entities and the block and to know the indices of the points that are involved (the corners, the edge vertices and the face vertices) as well as the indices of the vertices which will be created inside the block. A simple way to access all these vertex indices is to associate a *numbering matrix* with the block. To this end, we introduce a matrix with three indices which conforms to the following:

$$\mathcal{M}(0 : n_1 + 1, 0 : n_2 + 1, 0 : n_3 + 1)\,,$$

where, for instance, $\mathcal{M}(i, j, 0)$, for $i = 0, n_1$ and $j = 0, n_2$ corresponds to the "bottom" face of the block. More precisely, $\mathcal{M}(0, 0, 0)$, $\mathcal{M}(n_1, 0, 0)$, $\mathcal{M}(0, n_2, 0)$ and $\mathcal{M}(n_1, n_2, 0)$ are the four corners of the face and $\mathcal{M}(i, 0, 0)$, $\mathcal{M}(n_1, j, 0)$, $\mathcal{M}(i, n_2, 0)$ and $\mathcal{M}(0, j, 0)$ where i and j vary corresponding to the four edges of the face. Thus, the proper definition of the block involves filling its numbering matrix for the entities already known while the part of the matrix not yet known will be defined when the block is meshed.

Global numbering of the edge vertices. The given edges are first dealt with. Let $free = np_c + 1$. Then, for the first edge we apply an algorithm as below. Next, the $free$ value being completed, we turn to the next edge until all the edges have been visited:

Algorithm 4.1 *Global numbering of the edge vertices.*

```
Procedure GlobalNumbering
DO FOR $i = 1, n$ ($n$ being the number of desired points
   along the current edge, after the subdivision parameter
   $free = free + 1$,
   $v_i = free$, i.e., vertex $i$ of the edge is labeled
      with index $free$,
END DO FOR $i = 1, n$
```

Then, for edge ab, if $a < b$, the vertex indices are:

$$a, free, free + 1, ..., free + n, b\,,$$

while if $a > b$, these indices are:

$$a, free + n, free + n - 1, ..., free, b\,.$$

The adequate sequence of indices is put on the various numbering matrices which correspond to the various blocks sharing this edge. Depending on the block, i.e., depending on the location of a and b in the block under examination, the sequence $free, free+1, ..., free+n$ or $free+n, free+n-1, ..., free$ is merged in the matrix at the relevant place. Actually, the matrix indices of interest are those of the line (of indices) "joining a and b".

Global numbering of the face vertices. Then we consider the face vertices. First, the vertices located along the face edges are already labeled (see above). It remains to find a global label for the vertices interior to the face prior to filling the corresponding numbering matrices. The idea is to define (for the sake of simplicity, we consider a quad analogy) two directions of numbering. For instance, if $free$ is the first available label, the first line of the matrix could be:

$$free, free+1, free+2, ..., free+n$$

and, the second line could be:

$$free+n+1, free+n+2, ..., free+n+n\,.$$

Thus, according to the global definition of a face, several systems of numbering can be found. Let a_i be the four corners of the face. We pick the smallest index, say a_1, and we examine the indices of the corners before and after a_1 (see above), then

- if $a_2 < a_4$, the base of numbering is a_1a_2 meaning that a (sequential) variation from a_1 to a_2 is used,
- if $a_4 < a_2$, the base of numbering is a_1a_4 meaning that a (sequential) variation from a_1 to a_4 is used,

thus resulting, a_1 being identified, in two possible situations. Then, depending on the case, eight possible numbering systems can be found.

a_4	-	-	-	a_3
-	7	8	9	-
-	4	5	6	-
-	1	2	3	-
a_1	-	-	-	a_2

a_4	-	-	-	a_3
-	3	6	9	-
-	2	5	8	-
-	1	4	7	-
a_1	-	-	-	a_2

Table 4.2: The two "global" index systems when a_1 is the smallest index among the four a_i's and, left, when $a_2 < a_4$ or, right, $a_4 < a_2$ (in this table, for the sake of simplicity, we assume $free = 1$ (which is obviously not possible, as in practice a shift must be made) and $n = 2$ for both pairs of edges).

Now, the global indices of the face vertices are stored onto the numbering matrices at the proper places depending on what the a_i's are.

Global numbering of the internal vertices. Once again, let *free* be the last index used when labeling the face vertices, then the internal vertices are sequentially numbered. Three directions are used (in terms of index variation), an i-direction, a j-direction and a k-direction. Given the corners of the block, the i-direction follows edge a_1, a_2, the j-direction follows edge a_1, a_4 and the third follows edge a_1, a_8 (for a hex block).

Element vertex enumeration. At this stage, a global numbering system is available for all the vertices. In fact, a vertex is also known through its logical position in a given block. This is done using the matrix $\mathcal{M}$ associated with the block (we assume the same hex example as above). Then, the enumeration of the vertices of the different elements of the resulting mesh is easy to obtain. For instance, given i, j and k, the vertices of the corresponding (final) element are the values contained in:

$$\mathcal{M}(i,j,k), \mathcal{M}(i+1,j,k), \mathcal{M}(i+1,j+1,k), \mathcal{M}(i,j+1,k) \quad \text{and}$$

$$\mathcal{M}(i,j,k+1), \mathcal{M}(i+1,j,k+1), \mathcal{M}(i+1,j+1,k+1), \mathcal{M}(i,j+1,k+1)$$

for the bottom (resp. top) face of the element.

Note that while this enumeration is trivial for a hex or a pentahedral analogy, it requires some care for a tet analogy. This is due to the fact that, on the one hand, the final mesh is not formed of similar layers of elements and, on the other hand, that a region bounded by two triangles leads to the construction of one, two or three tet (in contrast to the other cases where two faces on two consecutive layers define only one element).

Thus the tet case is more tedious. Actually, given a face (say the bottom face of the block), we can classify the triangles covering this face into four categories.

- those resulting in three tets when considering a layer (say, in terms of index k, when going from k to $k+1$). Here, we must fill a small pentahedron,

- those also resulting in three tets (although they are inverted as compared with the previous ones),

- those resulting in two tets (those sharing a point with the "blended" block face), where the region to be meshed reduces to a prism (a pentahedron from which a tet is removed),

- those resulting in one tet (those sharing an edge with the "blended" block face or those located at a corner at level $n-1$). In this case, only a small tet must be meshed.

Exercise 4.2 *Find the four patterns encountered when dealing with a tet. Find the vertex indices (prior to applying the numbering matrix) of the final mesh based on these cases.*

Limitations. In principle, a multiblock method has the same range of applications than the local meshing processes that are used. Nevertheless, as the partition of the domain is made so as to overpass these limits, a multiblock method can successfully handle any arbitrarily shaped domain. In fact, the user is largely responsible for the success of the method by constructing a partition in suitable blocks.

Remark 4.5 *Applied in the surface case, for example if the blocks consist of triangles and quads in $\mathbb{R}^3$, the multiblock method provides a way to mesh a surface. Note that, in this case, the geometry of the surface is only related to the edges of the partition (see also Chapters 13 and 15).*

Application examples

The example in Figure 4.14 is a two-dimensional application of a multiblock method. The coarse partition consists of 10 quad regions and 8 triangular regions. There are 18 corners and 33 edges. Actually, only one subdivision parameter is used (for consistency reasons), leading to rather different element density. It should be noted that different block partitions can be used in this case resulting in different global meshes.

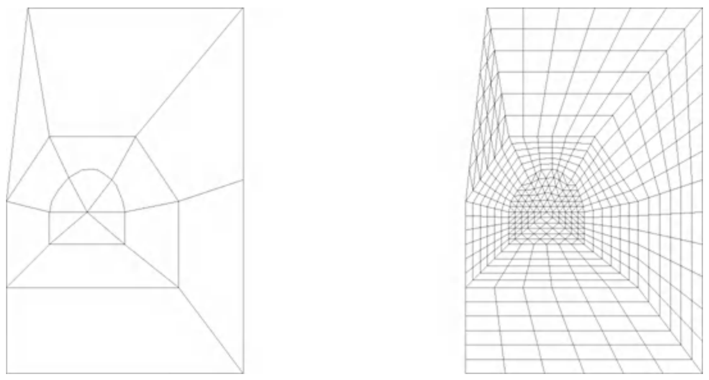


Figure 4.14: *Input data for the multiblock method (left-hand side), resulting mesh (right-hand side) (two-dimensional example).*

Figure 4.15 depicts a simple three dimensional application example. The figure on the left shows the part of the domain effectively considered. It consists of seven blocks. The figure on the right displays the mesh of the whole domain obtained from the previous mesh after several symmetries and (sub)mesh merging operations (see Chapter 17 for such mesh manipulation operators).

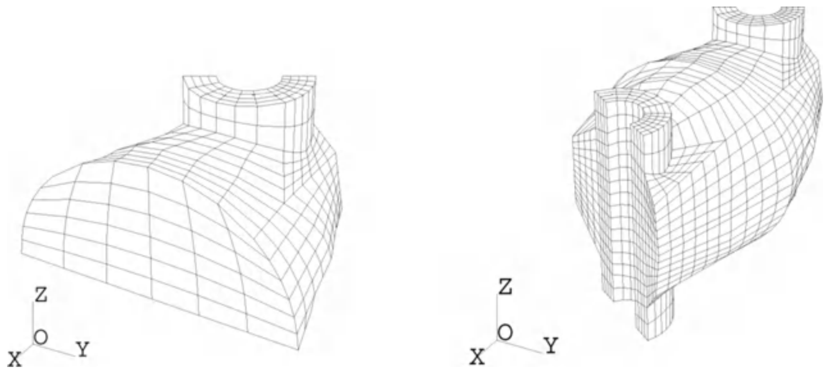


Figure 4.15: *Detail of the part effectively dealt with and resulting mesh after post-processing (three-dimensional example).*

★
★ ★

Algebraic, PDE-based and multiblock methods are usable in some specific cases while general situations require using more flexible methods as those decribed in the next chapters.

Chapter 5

Quadtree-octree Based Methods

Spatial decomposition methods were originally proposed as a way to represent approximations of geometric objects [Knuth-1975], [Samet-1984]. Quadtree- and octree-based mesh generation methods have been a topic of research for about three decades (see especially the surveys of the literature on spatial decomposition algorithms for mesh generation by [Thacker-1980] and [Shephard-1988]). In this context, decomposition approaches[1] have been designed to meet the needs of fully automatic mesh generation of arbitrary complex non-manifold objects and to reduce the extensive amount of time and effort required to generate meshes with semi-automatic methods [Yerry, Shephard-1983]. These approaches have proved to be robust and reliable and are commonly used in a wide range of engineering applications, see for instance, [Kela *et al.* 1986], [Perucchio *et al.* 1989], and [Shephard, Georges-1991].

In this type of approach applied to mesh generation, the object to be meshed is first approximated with a union of disjoint and variably sized *cells* representing a partition of the domain. These cells are obtained from a recursive refinement of a root cell enclosing the domain (i.e., a bounding box). Therefore, we obtain a covering up of a spatial region enclosing the object rather than of the object itself. In a second stage, each terminal cell is further decomposed into a set of elements whose union constitutes the final mesh of the domain (cf. Figure 5.1). The basic principle behind the method is that as the subdivision becomes finer, the geometry of the portion of the region in each cell becomes simpler, which simplifies the task of the second stage. One of the main features of the approach is its ability to proceed either directly from a given discretization of the domain boundary or, more generally, to interact with a geometric modeling system (a CAD system) and generate the boundary representation of the domain as a part of the whole meshing process.

★
★ ★

[1]So-called because they combine quadtree/octree decomposition techniques with quadrant-octant level meshing procedures.

This chapter is divided into five sections. The first section is devoted to the main concepts underlying spatial decomposition approaches and reviews the principal algorithms for constructing and searching with hierarchical structures. The second section discusses the construction of the tree representation and describes the algorithm used to subdivide the boxes into finite elements. The third section extends the tree decomposition to the generation of meshes governed by a size distribution function. In the fourth section, techniques that combine tree decomposition and other meshing techniques are introduced. In the fifth section, extensions to the generation of surface meshes and adapted meshes are briefly mentioned.

5.1 Overview of spatial decomposition methods

In this section, we recall some terminology and the basic definitions of the spatial decomposition structures in two and three dimensions, within the framework of mesh generation. PR-quadtrees will be used here as meshing structures (Chapter 2).

In order to fix the terminology, let us consider a meshing problem for which the domain to be meshed is an arbitrary non-convex domain Ω in $\mathbb{R}^2$ (resp. $\mathbb{R}^3$) represented by a boundary representation. The latter is in fact a polygonal (resp. polyhedral) contour $\Gamma(\Omega)$ composed of a set of vertices $\mathcal{V}$, a list of edges $\mathcal{E}$ and, in three dimensions, a list of edges and faces $\mathcal{F}$.

Terminology and definitions

The basic concept of any spatial decomposition consists first of enclosing an arbitrarily shaped domain Ω in an axis-aligned bounding box, denoted $\mathcal{B}(\Omega)$, (a square or a rectangle in two dimensions and a cube or a parallelepiped in three dimensions). This box is then subdivided into four (resp. eight) equally-sized *cells*, each of which may possibly be recursively refined several times. The *size* of a cell c is the length of the longest side of c. The *stopping criterion* used to subdivide a cell can be based on the local geometric properties of the domain (e.g., the local curvature of the boundary) or user-defined (the maximum level of refinement, for instance).

Remark 5.1 *The* stopping criterion *used for finite element mesh generation is usually different from the standard geometric criterion used in classical space partitioning procedures.*

The four (resp. eight) vertices at the corners of a cell are called the *corners*. The edges (resp. faces) connecting two (resp, four) consecutive corners are the *sides* of the cell. The edges (faces) of the decomposition that belong to the boundary of the cell are called the *edges* (faces) of the cell. Hence, each side of a cell contains at least one edge (resp. face). By definition, two cells are said to be *adjacent* if they share an edge (cf. Figure 5.2, right-hand side). The set of cells composes the *tree* (the tree structure) associated with the spatial decomposition. The subdivision

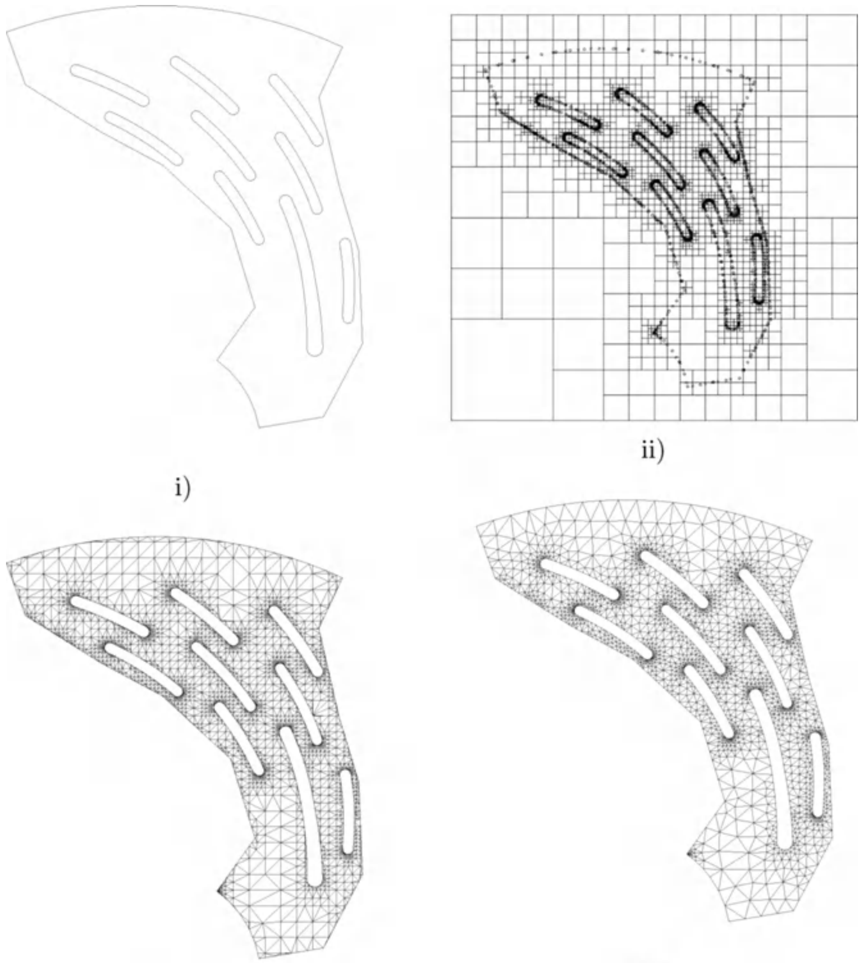


Figure 5.1: *A two-dimensional domain* Ω *i); spatial decomposition of* $\mathcal{B}(\Omega)$, *the bounding box of* Ω *ii); the resulting mesh iii); and the final mesh of* Ω *iv).*

of a cell consists of adding four cells to the node of the tree for that cell. The bounding box $\mathcal{B}(\Omega)$ is the *root* of the spatial decomposition tree.

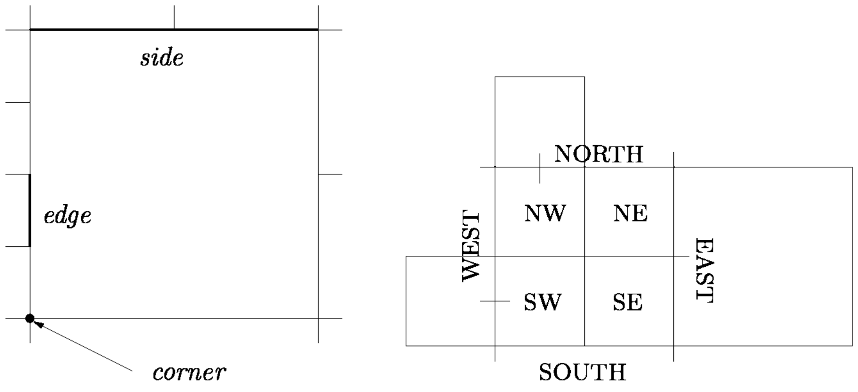


Figure 5.2: *Terminology: side, edge and corner of a cell (left-hand side). Adjacent cells sharing a common edge, canonical notations (right-hand side).*

The *level* of a cell corresponds to its depth in the related tree (i.e., the number of subdivisions required to reach a cell of this size). The bounding box is at level 0. The *depth* of the tree corresponds to the maximum level of subdivision. In Figure 5.3, the depth of the tree is 4. A cell that is not subdivided further is called a *terminal* cell or a *leaf*. Each non-terminal cell is called an *internal* cell.

In the classical spatial decomposition scheme, the current cell is subdivided into four (resp. eight) cells, the quadrants (resp. octants) and the set of points $\mathcal{V}$ is partitioned accordingly into several subsets. The insertion of a point into the tree consists of identifying the cell containing it. If this cell is empty, the point is inserted. Otherwise, the cell is refined and the process is iterated on each of the sub-cells. The recursive subdivision stops when the set of points associated with a cell is reduced to a single point or is the empty set. Initially, the choice of the bounding box is arbitrary and is usually a square (resp. cube) containing all of $\mathcal{V}$. This box is determined by computing the extrema of the coordinates of the points in x, y (and z) directions (note: the corresponding algorithm is linear in time: $O(np)$, where $np = Card(\mathcal{V})$).

Remark 5.2 *At each stage, a cell can be subdivided into sub-cells. This does not automatically imply that the set of points $\mathcal{V}$ is split accordingly: all points can belong to the same sub-cell. In this case, the resulting decomposition tree will be quite unbalanced (the number of subdivision levels not being constant in each cell).*

From the previous remark it can be seen that the size (i.e., the number of cells) and the depth of the tree are not determined only by the number of points in $\mathcal{V}$. The depth of the tree is related to the smallest distance d between two points of $\mathcal{V}$ as well as to the length of a side of the bounding box, as stated in the following lemma, in two dimensions:

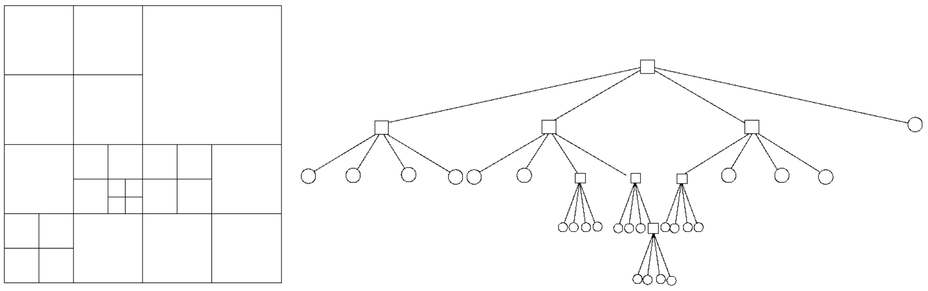


Figure 5.3: *Quadtree decomposition and corresponding data structure (here the tree depth is $p = 4$).*

Lemma 5.1 *The depth p of the quadtree obtained by inserting all points of $\mathcal{V}$ is bounded by $p < \log(b/d) + 1/2$, with d the smallest distance between two points of $\mathcal{V}$ and b the length of a side of $\mathcal{B}(\Omega)$.*

Proof. In a quadrant (a square for the sake of simplicity), the largest distance between two corners (i.e., the diagonal) is $l\sqrt{2})$, with l the length of a cell side. At each level of refinement, the size l of a cell is divided by two. At level i, the size l is then equal to $b/2^i$, with b the size of the bounding box and thus the length of the diagonal is equal to $\sqrt{2}b/2^i$. The smallest distance between two points of $\mathcal{V}$ being d, we must have $\sqrt{2}/2^i \geq d$, which relates the level of a non-terminal cell (i.e., containing at least two points) to the distance d. The maximum level sought (the depth of the tree) is then: $p \leq 1 + log(\sqrt{2}b/d)$. □

Exercise 5.1 *Find the corresponding upper bound in three dimensions.*

The previous remark about tree balancing leads to the following rule, which can be justified by the need to somehow control the mesh gradation in finite elements methods (i.e., the size variation between neighboring elements).

Definition 5.1 [2:1] rule. *A tree subdivision is* balanced *if every side of a terminal cell contains at most one corner (cf. Figure 5.4).*

This definition is equivalent to writing that the sizes of any two adjacent cells differ by at most a factor of two.

Moreover, a key aspect of spatial decomposition methods is their ability to classify the cells with respect to the domain Ω. The minimum classification requires a decision about whether a cell is fully inside the domain, fully outside the domain or contains portions of the domain boundary $\Gamma(\Omega)$ (e.g., the cell contains a point of $\mathcal{V}$ or is intersected by an edge of $\mathcal{E}$ or a face of $\mathcal{F}$, the list of faces). These cells are respectively denoted as $\mathcal{I}$ (inside), $\mathcal{O}$ (outside) and $\mathcal{B}$ (boundary).

Operations on trees

Two kinds of operations are commonly performed on trees:

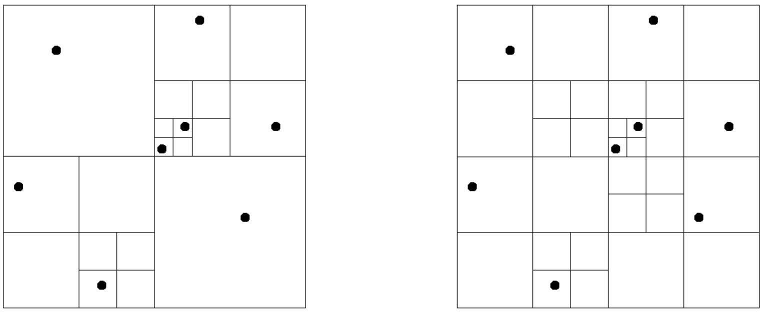


Figure 5.4: *Balanced tree according to [2:1] rule. Initial and balanced quadtrees.*

- topological operations: creating four (resp. eight) cells from a given cell, finding the cells adjacent to the current cell in a given direction,
- geometric operations: finding the cell containing a given point, calculating the intersections between an edge of $\mathcal{E}$ and the sides of the current cell.

These operations do not all have the same frequency or the same computational cost. The *neighbor finding* operation concerns seeking the cells adjacent to a given cell: let c be a cell and consider a direction, the problem is to find a cell c' adjacent to c in the given direction. Usually, c is a leaf and the problem is one of finding the adjacent cell that is of level less than or equal to that of c (i.e., the size of c' must be greater or equal to the size of c). Basically, the algorithm checks the current cell and compares the direction with the relative position of the cell in its *parent cell* c' (the non-terminal cell containing c). Notice that in two cases (i.e., directions), the adjacent cell belongs to the same parent of the current cell. In the other cases, the algorithm analyzes the parent of c to find a neighbor c' in the given direction. If c' is a leaf, c' is the neighbor of c, otherwise the neighbor of c is any of the *children* q_i corresponding to the subdivision of c'.

In two dimensions, the neighbors are identified by the four cardinal directions, $\{NORTH, SOUTH, EAST, WEST\}$ and the quadrants obtained during the refinement of a cell are identified by their relative position within the parent cell: $\{NW, NE, SW, SE\}$ (cf. Figure 5.2). These quadrants are so-called *siblings* to indicate that they result from the subdivision of the same cell. The notation $SE - child(parent(c))$ denotes the $SOUTH - EAST$ quadrant (not necessarily a leaf) of a cell c. Algorithm 5.1 searches for the $EAST$-neighbor (not necessarily a leaf) of a cell c in a quadtree $\mathcal{Q}$. The following lemma specifies the complexity of this algorithm.

Lemma 5.2 *Let $\mathcal{Q}$ be a quadtree of depth p. The searching algorithm used to identify the neighbor of a cell $c \in \mathcal{Q}$ in a given direction is of complexity $\mathcal{O}(p+1)$.*

Proof. First, notice that the tree is not assumed to be well balanced (the size difference between adjacent cells is not bounded). For each recursive call, the local complexity is in $\mathcal{O}(1)$ and the depth of the sub-tree traversed is decreased by 1. Thus, the complexity of the algorithm is linear in the depth of the tree. □

Algorithm 5.1 *Search for the EAST neighbor of depth not greater than that of a given cell c in the tree $\mathcal{Q}$.*

```
Procedure EastNeighbor($\mathcal{Q}$,$c$)
IF $c = \mathcal{B}(\Omega)$ (look for the root of $\mathcal{Q}$)
   RETURN $\mathcal{B}(\Omega)$
IF $c$=NW-child(parent($c$)) THEN ($c$ has a sibling on this side)
   RETURN NE-child(parent($c$))
IF $c$=SW-child(parent($c$)) THEN
   RETURN SE-child(parent($c$))
$c'$ = EastNeighbor($\mathcal{Q}$,parent($c$)) (recursive call)
IF IsLeaf($c'$) THEN
   RETURN $c'$
ELSE
   IF $c$=NE-child(parent($c$)) THEN
      RETURN NW-child(parent($c'$))
   ELSE
      RETURN SW-child(parent($c'$))
   END IF
END IF.
```

Exercise 5.2 *Update Algorithm (5.1) to always return a terminal cell. Write a more general algorithm to return a neighbor in any direction.*

This algorithm is only one of the components of the *a posteriori* tree balancing algorithm. The quadtree is balanced in a post-processing step that immediately follows the construction stage. Each and any leaf must conform to the [2:1] rule introduced previously. Hence, the computational cost of the balancing stage is related to the number m of leaves in the tree. According to [de Berg *et al.* 1997], the balancing operation is of complexity $O(m(p+1))$.

Exercise 5.3 *It is possible to balance a quadtree during the construction stage, without first constructing the unbalanced version. Describe such an algorithm and analyze its complexity.*

From the geometric point of view, the searching operation to find a cell enclosing a given point of $\mathcal{V}$ is an operation based on a comparison of the coordinates of the corners of a cell at a given level. Depending on the result of the comparison, a cell containing this point is identified and becomes the new current cell. The operation is then repeated recursively until a leaf is found that contains the point. Similarly, the intersections of an edge of $\mathcal{E}$ with a tree $\mathcal{Q}$ can be found.

Data structures

From the algorithmic point of view, D. Knuth identified in his book three possible approaches to representing trees [Knuth-1975]:

- A tree structure using pointers. In this *natural* approach, each non-terminal cell requires four (resp. eight) pointers, one for each of the subtrees and some information to indicate the cell classification. In addition, extra pointers can be added to improve the efficiency, such as links to the parent cell, links to neighboring cells, etc.

- A list of the nodes encountered by a traversal of the structure. Within this implementation, intersection algorithms can be efficiently performed, although other algorithms may be less efficient. For instance, visiting the second subtree of a cell requires visiting each node of the initial tree to locate the root of the subtree.

- A system of locational codes, for instance, with a *linear quadtree* (see below).

For example, Figure 5.3 (right-hand side) shows a tree representation associated with the quadtree decomposition depicted in the same figure (left-hand side). In practice, the context in which the method is applied dictates which of the three possible representations should be used. For instance, the need to quickly identify neighbors leads to favoring a linear structure or a pointer-based structure containing pointers to the parent cell and siblings.

An example: the linear quadtree.

We show here the linear quadtree implementation in two dimensions. The quadtree decomposition approach can be implemented as a hierarchical grid (Chapter 2), the number of boxes in any direction then being a power of two. A linear quadtree avoids the allocation of explicit pointers to maintain the spatial order of quadrants, a single array index being used to access the data structure. The idea behind the representation is rather simple. Each quadrant is represented by a number using a systematic scheme. For instance, label 1 corresponds to the bounding box, labels 2-5 represents the four quadrants of the bounding box. The labels are used as array indices to access the cell information. Each possible subdivision of a quadrant leads to four consecutive indices referring the four sub-quadrants (cf. Figure 5.5).

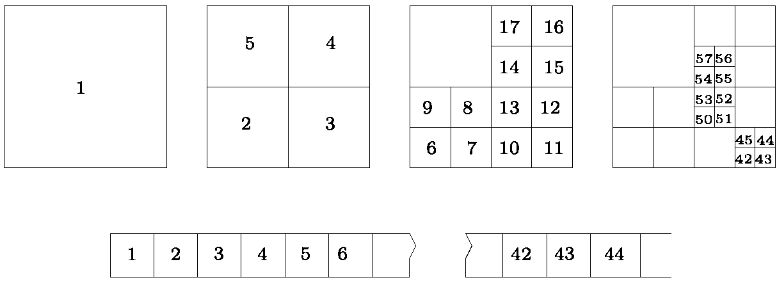


Figure 5.5: *Linear quadtree encoding: spatial addressing structure and numbering.*

Hence, it is pretty easy to calculate the index of any cell in the tree from its size and location. For instance, the four children of a cell c are given by $4c - 2 + j$,

$j = 0,3$ and its parent cell is trivially obtained as: $E((c+2)/4)$ where $E(i)$ denotes the integer part of i. Conversely, from this index, the size and position of any quadrant are easy to retrieve.

Exercise 5.4 *Indicate how to determine the neighbor indices of a given cell c. Write the algorithm to find the index of a cell enclosing a given point $P \in \mathcal{V}$.*

If the lowest level grid requires n^2 cells, the maximum depth p of the linear quadtree is given by: $p = \log_2 n$. Although the address of a box can be easily computed, this representation carries a storage penalty. Indeed, the total amount of memory m required to store the whole tree (in the worst case scenario) corresponds to the sum of a geometric series of ratio 2^2, i.e.,

$$m = \frac{2^{2p} - 1}{2^2 - 1} .$$

As compared with n^2 boxes required for its lowest level (corresponding to a regular grid of size n), $1/3$ more memory in two dimensions is required for a hierarchical linear tree.

Exercise 5.5 *Establish the previous formula and justify the extra storage penalty carried out by the structure as compared with a classical tree structure using pointers. How much more storage memory is required in three dimensions to store a hierarchical linear tree as compared with a regular grid of same resolution? (Hint: see [Gargantini-1982] or [Ibaroudene, Acharya-1991] for a possible solution.)*

Remark 5.3 *An alternative implementation approach consists of using a binary encoding: the locational index of each quadrant is composed of a sequence of digits c_i in base 4, each c_i being obtained by concatenation of the x_is and y_is of the binary representation of the box coordinates x, y.*

5.2 Classical tree-based mesh generation

The use of a quadtree decomposition for meshing purposes was pioneered about 25 years ago, in particular by [Yerry, Shephard-1983]. Several variants have been proposed (see for instance, [Kela *et al.* 1986] or [Perucchio *et al.* 1989]). Currently, such a meshing technique is usually based on three successive steps:

Stage 1: the parameterization of the mesh (specification of the element size distribution function, discretization of the boundary, etc.).

Stage 2: construction of a spatial covering up from a bounding box of the domain.

Stage 3: internal point and element creation.

This general scheme is somewhat different from that used in advancing-front or Delaunay type methods (see Chapters 6 and 7), in which the boundary discretization is a necessary input of the problem. In fact, in this approach, the construction of the covering up can be either based on a given boundary discretization[2] or performed using geometric queries to a geometric modeling system. The covering up is such that its cells

- have a size distribution compatible with the desired mesh gradation,
- maintain an efficient hierarchical data structure and
- store all the information required by the meshing step to generate a valid mesh of the geometric model (Stage 3 of the previous scheme).

In a spatial decomposition method, the mesh vertices are created by the element creation procedure. However, unlike other mesh generation techniques (advancing-front or Delaunay type, for example), the internal vertices are usually the vertices of the decomposition (i.e., the corners of the cells). The boundary vertices are the initial points of $\mathcal{V}$ and some additional points, created during the tree construction and corresponding to the intersection of the boundary with the tree. Therefore, the initial boundary discretization is not usually preserved in this approach.

General scheme

Let us consider here a boundary discretization represented by a polygonal (polyhedral) contour described as a list of points, a list of edges and possibly a list of faces (in three dimensions). For the sake of simplicity, the boundaries of the domain are considered as straight segments (for curved boundaries, see Section 5.4).

Formally speaking, the spatial decomposition technique is an iterative procedure that builds the covering tree of the domain from its description before meshing each terminal cell. At each stage of the tree construction, each leaf is analyzed and refined into 2^d (with d the space dimension) equally sized cells, based on a specific criterion. Schematically, a spatial decomposition method reads:

Stage 1: preliminary definitions:

- construction of a size distribution function defined by interpolation (governed mesh generation) or implicitly[3],
- boundary discretization (general case),
- creation of a bounding box of the domain.

Stage 2: initialization of the tree by the bounding box.

Stage 3: tree construction (insertion of each and any entities of the boundary discretization):

- (a) selection of a boundary entity, in ascending order (points, edges and faces),
- (b) identification of the cell in the current tree that contains this entity,

[2]In which case, the points and the elements created at Stage 3 are internal to the domain.
[3]See Section 5.3.

- (c) analysis of the cell, if it already contains an entity of the same dimension then refine the cell, otherwise back to (b),
- (d) insertion of the entity in the cell and back to (a)

Stage 4: tree balancing, the level difference between any pair of adjacent cells is limited to one.

Stage 5: point filtering, in each terminal cell intersected by the boundary.

Stage 6: creation of the internal points and the mesh elements:

- predefined patterns (the *templates*) are used to mesh internal cells,
- boundary leaves are meshed using an automatic technique (decomposition into simpler domains, etc.),
- removal of the external element, using a coloring algorithm,
- intersection points are then adjusted on the true geometry.

Stage 7: mesh optimization.

Despite conceptual differences among the various approaches published, several aspects of any tree-based approach used for meshing purposes are consistent:

- the data structure is used for localization and searching purposes,
- the mesh generation has two stages, first the tree is generated, then the mesh is created based on cell classification within the tree,
- the cell classification drives the mesh generation stage (although boundary cells may need special care),
- the cell corners are generally used as mesh vertices,
- the mesh gradation is controlled by the level of refinement of the cells (for instance, according to the [2:1] rule described above).

In the following sections, we will describe the classical tree-based meshing approach as described in the previous scheme, and indicate the main difficulties that may arise.

Main difficulties. The general scheme above hides several potential difficulties. Most of these problems only occur in three dimensions, especially regarding the robustness of the method. In fact, the construction of the tree requires enforcing a specific criterion, the latter possibly being incompatible with the geometric requirements. For instance, we could specify a maximum level of refinement, thus leading to problems when trying to discriminate closely spaced entities (when the distance is much smaller than the specified cell size). Expected difficulties can easily be identified and are usually related to:

- the knowledge of the local neighborhood of a (boundary) entity during its insertion in the current tree,

- the intersection computations during the insertion of the entity,
- the filtering of closely spaced points, which requires a robust algorithm in order to preserve the topology and the geometry of the domain.

Notice that in this approach, the points are not optimal (in the sense that they do not necessarily lead to the creation of optimal elements, Chapter 1), but result from intersection calculations (boundary/cells) or correspond to the corners of the cells. Therefore, the resulting meshes must be optimized (Chapter 18).

Remark 5.4 *These difficulties are essentially twofold, numerical (intersection calculations) and algorithmical (neighborhood, ...).*

Preliminary requirements

Unlike the advancing-front method (Chapter 6), no specific assumption is made on the nature of the input data for boundary discretization. Obviously, the data structures are important in this approach, the decomposition tree acting as a neighborhood space as well as a control space (Chapter 1).

Boundary mesh. Similar to Delaunay-type methods, the boundary mesh is not necessarily oriented. However, for efficiency purposes during the insertion of the boundary entities, it may be useful to enforce an orientation of the discretization, so as to insert sequentially neighboring entities during the tree construction stage, thus reducing tree traversals as much as possible (in searching operations).

Data structures. As searching and insertion operations are usually local features, it is important to use suitable data structures in the process. So, the tree structure (*quadtree* in two dimensions and *octree* in three dimensions) should contain in each terminal cell all the information required for a further analysis of the cell. Therefore, the boundary items are stored in the terminal cells of the tree. Two adjacent cells can share some common information (an intersection point, for instance), so it is desirable to refer to this information in the cell rather than duplicating it (to avoid numerical errors). Further updates of the tree (for example, a cell refinement operation) will then propagate the information to the terminal cells. The tree construction requires only refinement operations; no cell deletion is involved.

Tree construction

The tree representation of an arbitrary domain Ω is defined from the bounding box $\mathcal{B}(\Omega)$ that fully encloses the set of points of $\mathcal{V}$ of the boundary discretization $\Gamma(\Omega)$. The tree is initialized by $\mathcal{B}(\Omega)$. The items of $\Gamma(\Omega)$ are iteratively inserted into the cells of the current tree, so that each terminal cell contains at most one item of $\Gamma(\Omega)$. Any cell that contains more than one such item is refined into 2^d identical sub-cells and the insertion procedure is resumed for each cell of the sub-tree.

The insertion of the boundary items is a tedious process that can be implemented in many ways. The common method consists of inserting the items according to the ascending order of their dimensions (points, edges and faces). Moreover, if the boundary has been oriented, it may be interesting to insert adjacent items, so as to benefit from the local aspect of the operations and thus to limit somehow the tree traversals. Generally speaking, it is desirable to avoid:

- the loss of previously computed or known information (for example, numerical values related to the intersection points),
- the degradation of the computational cost of tree operations (cache default, Chapter 2).

Point insertion. The localization of the cell containing a boundary point or intersected by a boundary item requires special care (especially in three dimensions). Actually, a point can be located inside a cell, along a cell side (and then it belongs to two adjacent cells) or even on a corner (it is then associated with all the cells sharing this common corner). It can be easily imagined that numerical problems can arise when trying to correctly identify the different configurations.

Remark 5.5 *A commonly used technique consists of slightly modifying the position of the point so that it belongs to only one cell. Its initial position will be restored after the mesh generation.*

When the cell containing a point has been identified, two situations can arise. First, if this cell is empty (i.e., does not include any boundary item), the process consists of associating the related information with the cell. Otherwise, the cell is refined into 2^d identical cells and the analysis is propagated to each of the sub-cells.

In practice, the processing is even more complex if the item is an edge or a face as several sub-cells can be intersected.

Peculiar points. From a computational point of view, two kinds of points require specific attention, the *corners* and the *non-manifold* points. Corners are boundary points at which incident items form a small (dihedral) angle (cf. Figure 5.6, left-hand side). In two dimensions, non-manifold points are boundary points with more than two incident edges (cf. Figure 5.6, right-hand side). In three dimensions, this concept is more tedious to handle as non-manifold edges (shared by more than two faces) have to be defined first, their endpoints then being non-manifold points.

The test making is possible to decide whether a terminal cell must be further refined is called the *stopping criterion.* Several such criteria have been suggested in the context of mesh generation.

Stopping criteria. Usually, the stopping criterion must take into account the different types of items to be inserted. The most commonly used criterion is such that any leaf of the tree contains at most one connected component of $\Gamma(\Omega)$. If a

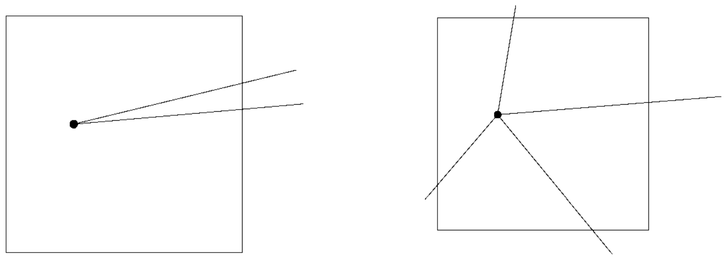


Figure 5.6: *Identification of corners (left-hand side) and non-manifold points (right-hand side) in two dimensions.*

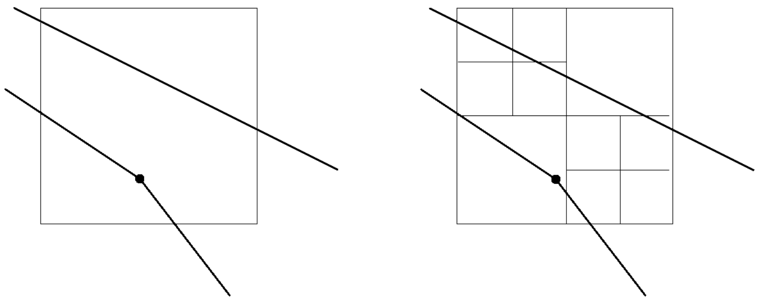


Figure 5.7: *Tree refinements based on two different stopping criteria in two dimensions: any leaf contains at most one vertex of* $\Gamma(\Omega)$ *(left-hand side), each leaf contains at most one point (shared by two edges) or a small part of an edge of the discretization (right-hand side).*

leaf contains no point of $\mathcal{V}$, then it should contain at most one edge or portion of an edge (resp. face) of $\Gamma(\Omega)$ (Figure 5.7, right-hand side).

For arbitrary domains, the stopping criterion is of importance and affects the depth of the resulting tree and thus the complexity of the whole algorithm. Figure 5.7 illustrates the impact of the stopping criterion on the decomposition. On the left, the quadtree is intersected by three segments and contains one point of $\Gamma(\Omega)$. The corresponding criterion consists of stopping the decomposition when each cell contains no more than one point. On the right side, the stopping criterion is such that each cell must be intersected by no more than one edge (or part of an edge) if it does not include a vertex of $\mathcal{V}$.

Remark 5.6 *In two dimensions, the most strict stopping criterion consists of forcing each cell side to include no more than one intersection point, unless the cell contains a corner or a non-manifold point [Frey, Maréchal-1998].*

Remark 5.7 *In three dimensions, the classical criteria assign control parameters to the boundary items (for instance, related to the local curvature of the geometric features; see Section 5.5).*

The construction stage attempts to separate two opposite sides of the domain based on a distance criterion. In other words, if the domain is such that locally two sides are very close to each other, with really different discretization sizes, the tree decomposition will automatically reduce this shock between the sizes, by inserting additional points and refining the boundary items. This is in fact equivalent to refining the tree until the cell sizes become compatible with the local distance between the opposite sides of the domain. The resulting mesh gradation is thus controlled in an implicit fashion. Moreover, this control is also increased by the balancing stage (see above).

Remark 5.8 *As the tree construction stage may introduce additional points, the initial boundary discretization is no longer preserved (unlike the advancing-front or Delaunay-based mesh generation methods, for which the boundary discretization is a constraint that has to be strictly preserved).*

Remark 5.9 *The specification of additional (non-boundary) points in the input dataset does not lead to substantial modification of the classical algorithm. These points will be inserted in the tree just after the insertion of the vertices of the boundary discretization.*

Figure 5.8 depicts the different steps of the tree construction of the decomposition of a planar domain in two dimensions. The domain is illustrated on the left-hand side of the figure. The tree (Figure 5.8, middle) corresponds to the decomposition obtained after the insertion of all boundary points. At this stage of the process, no additional point has been created. Finally, the insertion of boundary edges leads to the domain decomposition shown in Figure 5.8 (right-hand side). Several intersection points have been created and the tree has been locally refined to accommodate the stopping criterion.

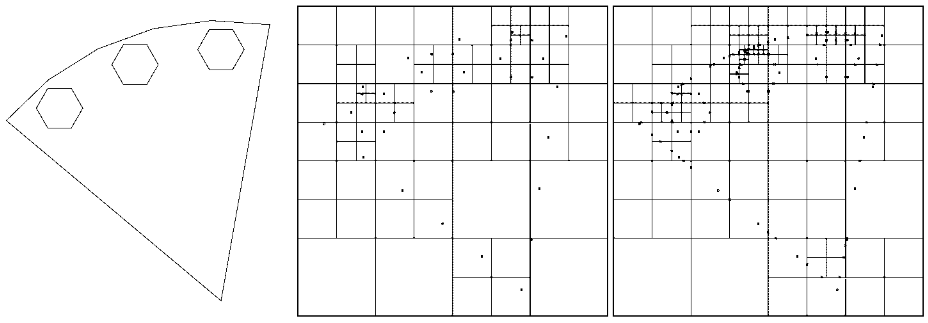


Figure 5.8: *Quadtree decomposition of a two-dimensional domain. Initial discretization (left-hand side), resulting quadtree after the insertion of the set of points $\mathcal{V}$ (middle) and after the insertion of the edges of $\mathcal{E}$ (right-hand side).*

Exercise 5.6 *[de Berg* et al. *1997] Let us consider a two-dimensional domain Ω such that the points of $\mathcal{V}$ have integer coordinates and that each boundary edge makes an angle of $0°, 45°, 90°$ or $135°$ with the x-axis. The stopping criterion* *could be such that a quadrant is not subdivided if the intersection between an edge of $\mathcal{E}$ and its sides is empty or if the minimal size is reached. Show that the interior of the quadrants can be intersected by the polygonal contour only in a single way. Write the resulting construction algorithm for this specific stopping criterion.*

Notice that a variant of the construction procedure described above has been proposed by [Yerry, Shephard-1983]. In this approach, a limited number of intersections (i.e., of segments coming from boundary edges intersections) is tolerated in boundary quadrants. To preserve the basic structure, intersection points are chosen as quarter, half or endpoints of the quadrant's sides. This technique does not improve the geometric accuracy of the representation, although it improves the numerical approximation and the consistency of the algorithm. It allows the use of integer coordinates to represent the domain boundary discretization.

The result of the tree decomposition is a covering up of the bounding box of the domain in which adjacent cells may have dramatically different sizes. In other words, the quadtree may be quite unbalanced. This feature can become a major drawback during the element mesh generation stage (as the mesh gradation is not explicitly controlled). To overcome this problem, a balancing stage is applied.

Tree balancing

As mentioned previously, the tree construction rules concerned the boundary items of the given discretization. Hence, the tree resulting from the general scheme can be rather unbalanced (according to Definition 5.1). This feature, related to the difference between the levels of each pair of adjacent cells, can be inconvenient in many ways. First, the size variation between neighboring cells tends to increase the cost of tree traversals (in searching operations), the number of levels explored varying from one cell to the next one. Then, as spatial covering up is used to generate a mesh, the mesh edges will have a length close to the cell size that is used to generate them. In fact, when the size distribution function is implicitly defined (no user-specified function is explicitly supplied), the cell size reflects the average size of the elements generated in this cell. Finally, as a cell can be surrounded by cells of arbitrary dimensions, the number of possible combinations is not limited, thus preventing the use of an automatic procedure (based on predefined patterns) to mesh the cells.

Therefore, in order to control mesh gradation and also to simplify the mesh generation stage, another rule is introduced so as to limit the difference between the levels of two adjacent cells to a factor 1 at most. This rule is the so-called *[2:1] rule*.

This procedure affects the spatial decomposition of the domain obtained using the former construction rules. In particular, it leads to refining several cells and thus propagates the refinement recursively throughout the tree, to the neighboring cells. The final decomposition usually contains (experimental result) up to 10% more leaves than the initial one.

Implementation of the [2:1] rule. From a practical point of view, the [2:1] rule can be added to the construction rules. This is equivalent to saying that it is not necessary to initially construct an unbalanced tree, on which the balancing rule is applied *a posteriori*. In fact, if the current tree is already balanced, it is sufficient to check whether the balancing rule is enforced in each leaf that is subdivided during the construction stage, otherwise, the levels of the adjacent cells are adjusted (refined) until the balance has been achieved. In practice, it can be observed that the propagation of the refinement is usually contained in two cells adjacent to the current cell. This feature results in a local balancing procedure, which is easy to implement and computationally inexpensive as only limited tree traversals and updates of the tree structure are involved.

Remark 5.10 *Notice that, similarly to the construction stage, the corners and non-manifold points are not concerned with the balancing stage (i.e., the cell containing such a point is not refined). Figure 5.9 illustrates such a situation where the tree is locally unbalanced because of a non-manifold point.*

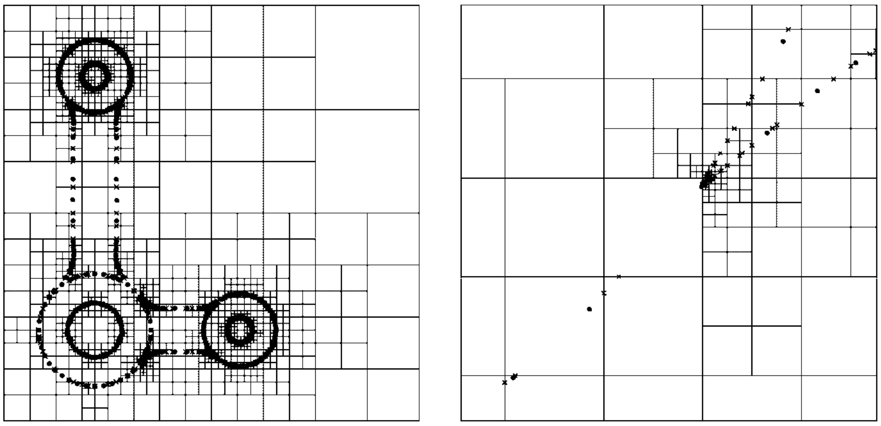


Figure 5.9: *Example of an unbalanced tree in the vicinity of a non-manifold point, in two dimensions. Spatial decomposition of the whole domain (left-hand side) and close up in the vicinity of the cell containing the non-manifold point corresponding to the intersection of three boundary edges (right-hand side).*

Remark 5.11 *It is however possible to balance a given tree* a posteriori. *To do so, a procedure based on a* breadth-first *tree traversal is applied (starting from the root of the tree).*

The tree construction procedure can potentially introduce some points that are not part of the initial dataset. For instance, the points resulting from the intersection between the cell and the boundary discretization. These intersection points can be very close to the cell sides, thus leading to the creation of poorly-shaped elements. To overcome this problem, a filtering procedure is carried out on the boundary points.

Filtering of the intersection points

Figure 5.10 (left-hand side) illustrates one of most common cases of intersection points creation during the insertion of a boundary edge into the current tree, in two dimensions. The two intersection points that belong to the upper-left quadrant are spaced closely to each other as well as close to the quadrant corner. This usually leads to the generation of a triangle which has these three points as vertices and whose size does not match the local size specified by the distribution function.

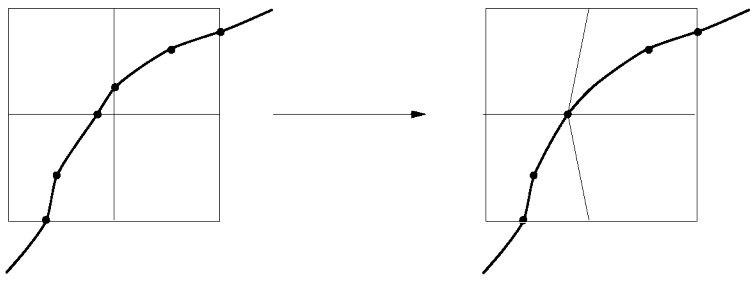


Figure 5.10: *Insertion of the boundary edges in the tree. Left, two intersection points are close to the quadrant corner. Right, the points have been merged together into a single one, the quadrant corner then being moved toward the resulting point.*

As the construction rules do not explicitly take into account inter-point distances (the discretization of the domain boundary is the input data), a post-processing stage applied on the resulting tree appears to be the only solution to this problem.

The intersection points filtering procedure consists of removing some vertices in the tree based on a distance criterion. This tedious operation is not easy to implement as it affects the geometry of the spatial decomposition. The local geometric modifications carried out mainly consist of:

- associating a cell corner with the closest (boundary or intersection) point belonging to the cell side,
- merging two intersection points into one,
- associating an intersection point and the closest boundary point within the same cell.

All these checks can be carried out provided the topology of the geometric model and that of its decomposition are not altered. Thus, two intersection points can be merged together if and only if they are classified onto the same geometric boundary item. Moreover, the geometry of the initial boundary discretization should not be modified. For instance, the deletion of a boundary point (a vertex of $\mathcal{V}$) is not allowed.

At completion of the procedure, there should be no point closer to another one than a given tolerance value[4]. Figure 5.10 (right-hand side) depicts the merging

[4]This tolerance can vary according to the type of entity to be snapped.

between two intersection points followed by the relocation of the quadrant corner onto the remaining intersection point.

Remark 5.12 *As the filtering procedure may change the coordinates of a quadrant corner, this cell may lose its axis-alignment property. This can result in additional difficulties for the searching procedure (e.g. to identify which quadrant contains a given vertex).*

Figure 5.11 shows the result of the filtering stage on a two-dimensional computational domain. We can clearly see (right-hand side) the quadrant deformation due to the relocation of the tree vertices. Although many extra intersection points have been discarded, the initial boundary discretization is not preserved (i.e., some of the intersection points still remain).

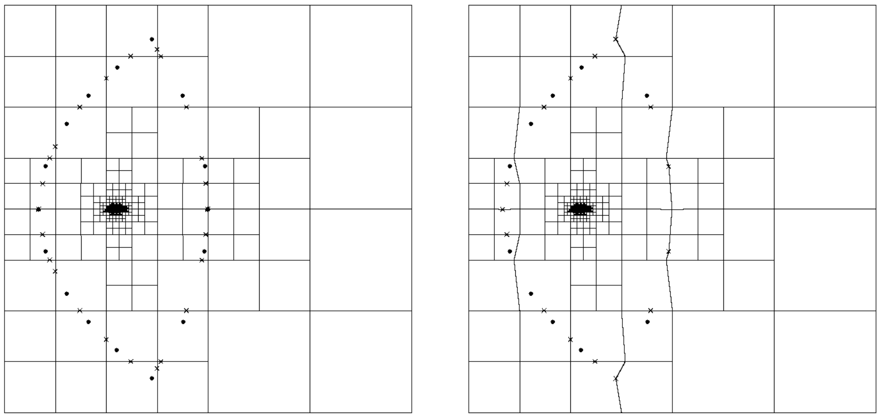


Figure 5.11: *Initial decomposition of a two-dimensional computational domain (left-hand side) and resulting decomposition after the filtering of the intersection points (right-hand side). Notice the deformation of the spatial decomposition structure.*

Remark 5.13 *As the topological consistency of the tree representation is rather difficult to preserve (especially in three dimensions), in some of the proposed approaches points are not filtered, [Frey, Maréchal-1998]. In this case, the resulting mesh elements that may have an unacceptable size or quality are further processed, during the optimization stage.*

The filtering stage marks the end of the processing on the tree structure. At completion of this stage, the tree structure remains unchanged and the following steps are exclusively based on tree traversals, especially the element creation stage.

Vertex and element creation

At this stage of the procedure, we have to go from a spatial decomposition structure to a mesh of the domain. To this end, the basic idea of this approach is to mesh the leaves of the tree independently of one another. The final mesh will be the union of all the elements created at the cell level, the global conformity being ensured by the conformity of the cell side discretization.

Vertex creation and insertion. The creation of the mesh vertices consists of sequentially inserting each cell corners and each intersection point in the mesh structure. From a practical point of view, it may be noticed that the number of mesh vertices is known beforehand and corresponds to the sum of the number of boundary vertices, the number of the cell corners and the number of intersections points (if any). This allows the data structures to be allocated correctly at the beginning of the program. This step is based on a tree traversal to find all terminal cells. Here, we encounter again the need to have an efficient and flexible data structure.

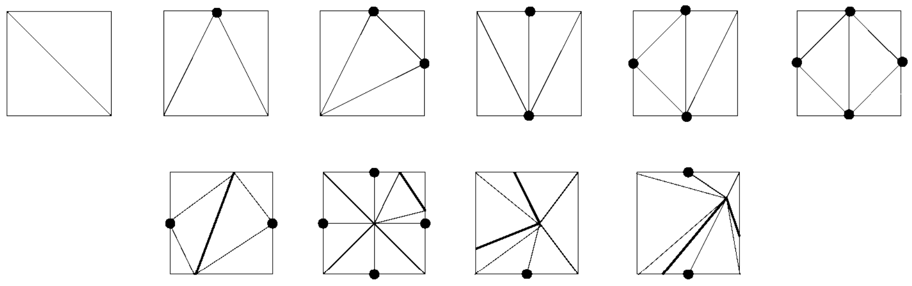


Figure 5.12: *A set of six plausible patterns to triangulate the internal quadrants (the other patterns can be retrieved using the rotational symmetry properties) and four patterns used to mesh boundary quadrants (in the last two patterns, the intersection point sees the other quadrant points).*

Mesh element generation. The creation of the mesh topology (i.e., the mesh elements) is a more tedious operation. First notice that the balancing rule led to a substantial reduction in the number of possible configurations for a given cell, as compared with its neighbors. Therefore, knowing of the sizes of the adjacent cells is sufficient to be able to mesh a terminal cell. However, it is necessary to distinguish the case of boundary cells from that of internal cells.

The case of an internal cell is indeed very simple as it involves a set of predefined patterns (the so-called *templates*) used to get the list (the connectivity) directly of the elements that form a conforming covering up of the cell. Global mesh conformity is automatically ensured, as seen above.

In two dimensions, one has to consider $2^4 = 16$ possible configurations for each internal cell. From the computational point of view, this number can be reduced to only six templates using the various symmetry properties. Figure 5.12 depicts a set of templates used to mesh internal cells.

In three dimensions, $2^{12} = 4,096$ possible configurations must be considered for each internal cell, this number could be reduced to only 78 (according to

[Yerry, Shephard-1984]) using the various symmetry and rotational properties, as in two dimensions. As the full enumeration of these cases is error-prone, a simpler approach is preferred to mesh any internal octant. In practice, any octant having all its neighbors equally sized is decomposed into six tetrahedra, otherwise, the octant is decomposed into six pyramidal elements, the vertices of which are the octant corners and the centroid of the octant. These pyramids are then decomposed into tetrahedral elements according to the neighboring octant configurations (cf. Figure 5.13).

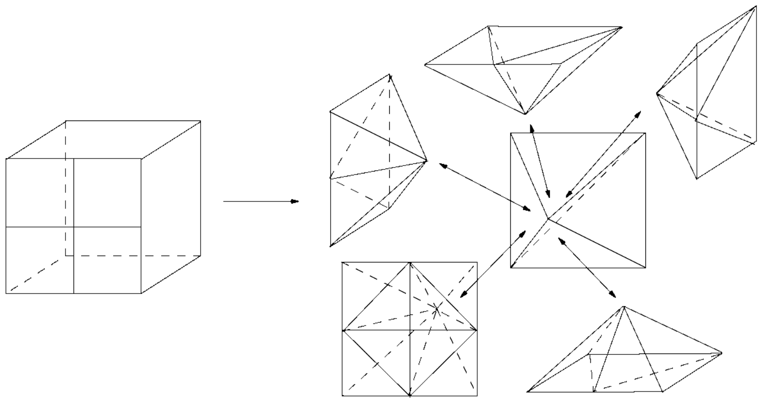


Figure 5.13: *Automatic triangulation of an internal octant. The octant is subdivided into six pyramids that are then meshed according to the neighboring octant sizes.*

When dealing with boundary cells, the procedure requires more attention. Mesh elements are created in two steps [Grice *et al.* 1988]:

- the sides of the cell are first discretized so as to ensure conformity with the neighboring cells,
- the barycenter of all points within the cell (boundary, intersection points and corners) is inserted and then connected to these points to form tetrahedral elements, in three dimensions.

Obviously, it is necessary to check that the edges and faces corresponding to the boundary discretization are correctly formed in this procedure. In particular, two intersection points issued from the same edge must be connected with a mesh edge (to preserve the topology of the domain).

From the algorithmic point of view, the procedure is based on a tree traversal (a depth-first or a breadth-first traversal, the order not being significant at this stage). The data structure should allow the quick identification of the neighbors of a given leaf. Moreover, the topology of the resulting mesh is explicitly given as a list of elements, each element being described by the references of its vertices. Therefore, a mesh data structure can be very simple, as it is not used for neighborhood searching purposes, for instance. This is indeed a difference with the advancing-front type methods, for which the current mesh is always searched to determine the candidate points in the vicinity of a given point (Chapter 6).

Remark 5.14 *The proposed method leads to a simplicial mesh of the domain (with triangular or tetrahedral elements). In two dimensions, it is also possible to obtain a mixed mesh (composed of triangles and quadrilaterals); other conforming patterns then need to be defined.*

Removal of external elements. The spatial decomposition obtained at completion of the previous stage is a covering up of the bounding box of the domain rather than a covering of the domain. The resulting mesh is then a mesh of the bounding box of the domain (similar to that obtained by constrained Delaunay methods; see Chapter 7). The boundary discretization is present in the current mesh, thus making it easy to identify internal elements.

A very clever and simple algorithm, based on a coloring scheme (see Chapter 2), is used to mark the different connected components of the domain. It is then possible to keep the mesh of one specific component only. This algorithm can be summarized as follows:

1. Assign the value $v = -1$ to all bounding box mesh elements and set $c = 0$.
2. Find and stack an element having a value equal to -1.
3. Pop an element, while the stack is not empty,
   - if the value associated with the element is different from -1, go to Step 4,
   - assign the value $v = c$ to the element,
   - stack the three (resp. four) adjacent elements if the common edge (resp. face) is not a boundary item,
   - back to Step 3.
4. Set $c = c + 1$ and while there exists an element having a value $v = -1$, go back to Step 2.

Other techniques can be used to achieve the same result, all based on the identification of the various connected components of the domain.

Boundary points relocation. So far, we have considered the items of the boundary discretization to be straight segments. Thus, the introduction of intersection points during the tree construction does not pose any particular problems. However, if the boundary discretization is a piecewise polygonal (resp. polyhedral) approximation of the true boundary, it is fundamental that the resulting mesh vertices all belong to the true underlying curve (resp. surface). Indeed, for numerical simulation purposes, it is desirable to have a polygonal (resp. polyhedral) representation of the curve (resp. surface) for which the gap between the underlying geometry is bounded by a given tolerance value ε.

If a geometric model is available, queries to the modeling system provide the exact position of vertices corresponding to the intersections of the tree cells and the curve (resp. surface). This operation usually concerns a small number of vertices and does not lead to a substantial increase in the computational cost.

When the boundary discretization is the only data available, it is possible to construct a G^1 continuous geometric support that reasonably emulates the features of a geometric modeling system (especially, given a point, to find the closest surface point in a given direction; see Chapter 19).

Notice that, if the initial boundary discretization is a geometric one (for which the edge lengths are proportional to the local curvature of the surface; see Chapters 14 and 15), the final position of a point onto the geometric support is usually very close to its initial position. However, if the discretization is arbitrary, the node relocation can lead to the creation of invalid elements. That is why, in some cases, the relocation may be refused. Figure 5.14 illustrates the notion of geometric approximation for a given tolerance value (see also Section 5.4.1).

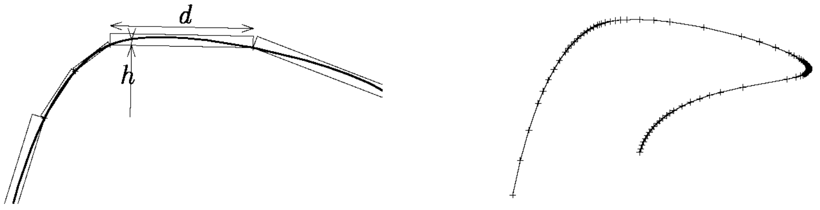


Figure 5.14: *Geometric approximation using a polygonal segment with respect to a given tolerance value $\varepsilon = 0.08$ (left-hand side) and $\varepsilon = 0.01$ (right-hand side). The relative tolerance is such that $h/d < \varepsilon$, with h the distance between a point and the supporting edge and d the edge length.*

Optimization

When all decomposition points have been inserted and all elements have been created within the terminal cells, we obtain a mesh of the domain that can be optimized. The goal is to obtain a finite element mesh and the optimization criterion must reflect this objective. A quality measure for a mesh element K is given by the following formula (see also Chapter 18):

$$\mathcal{Q}_K = \alpha \frac{h_{max}}{\rho_K}, \tag{5.1}$$

with h_{max} the diameter of K and ρ_K the radius of the inscribed circle (sphere). This leads to a raw value at the mesh level:

$$\mathcal{Q}_{\mathcal{T}} = \max_{K \in \mathcal{T}} \mathcal{Q}_K . \tag{5.2}$$

The optimization stage aims at minimizing this value. Notice, however, that this quality measure has not been explicitly taken into account during the tree construction or during the element creation step. Moreover, the mesh vertices corresponding to the cell corners confer a certain rigidity to the resulting mesh. Therefore, the optimization stage concerns, *a priori*, all mesh elements, noticing, however, that the elements created in internal cells are usually well-shaped.

Optimization procedures. The basic idea is to locally modify the mesh so as to progressively and iteratively improve its global quality. We commonly identify two categories of local modifications (see Chapter 18):

- topological modifications that preserve the point positions and modify the connections between mesh vertices and
- metric modifications that change the point positions while keeping their connections.

Several operators allow these local mesh modifications to be carried out, and, more precisely, to:

- move the points (for example using a weighted barycentrage technique),
- remove points,
- remove edges by merging their endpoints,
- flip (swap) edges (in two dimensions) or flip edges and faces (generalized swap in three dimensions).

The quality of the final mesh is improved by successive iterations, a judicious combination of these operators allowing a gain in quality. Notice also, that this optimization stage is (almost) identical to the one involved in other simplicial mesh generation methods (see Chapters 6 and 7).

Remark 5.15 *If the filtering stage has not been carried out, the optimization stage can be more time-consuming. In particular, it is necessary to deal first with the mesh elements which have a size that is not compatible with the given size specification and resulting from the intersections with the cell sides.*

Computational issues

As clearly appears in the general scheme, the efficiency of a spatial decomposition method is strongly related to the criteria used during the tree construction stage.

Basic algorithms. A careful analysis of the general scheme shows three key points. They are related to the tree construction and the tree structure management, the insertion of the boundary discretization (intersection problems) and the implementation of predefined patterns for triangulating terminal cells.

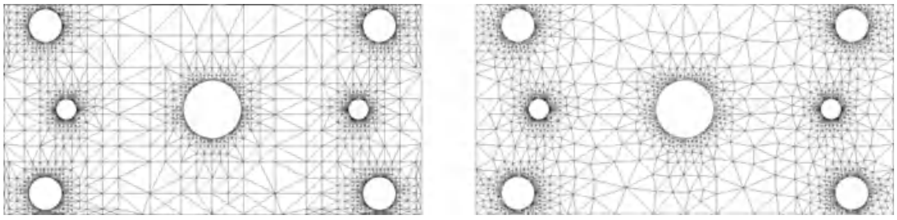


Figure 5.15: *Two dimensional mesh optimization. The initial raw quadtree mesh (left-hand side) and the final mesh after optimization (right-hand side).*

- Construction and management of the tree structure.

The tree structure is a dynamic structure that is updated during the insertion of the boundary discretization items and during the balancing stage. The use of a suitable data structure adapted to such operations (tree insertion, point searching operation, neighborhood, etc.) greatly facilitates the tree construction. A data structure enabling a direct access to the adjacent cells of a given cell should be favored so as to reduce tree traversals.

The search for a cell containing a given point is a common operation based on numerical checks. During tree construction, all cells have a similar shape (a square or a cube), thus allowing a bounding box test to be used to find the cell (the point coordinates being bounded by the cell extrema). However, if a filtering stage has been carried out, the shape of the cells is no longer preserved, thus making the search operation more tedious.

- Insertion of the boundary discretization.

The insertion of the boundary discretization involves numerical (metric) tests as well as topological tests (related to the neighborhood). Tree-like data structures are usually well suited to searching for items using adjacency relationships[5] However, the tree construction can lead to the division of a boundary item (for instance, because of its proximity to another item). In this case, numerical information has to be propagated in all related cells. We have already mentioned that it could be useful to slightly modify the coordinates of a point so as to avoid numerical problems when inserting it into the tree. The intersection checks concern the pairs edge-cell (hence, edge-edge), edge-face and face-face.

- Template implementation.

In order to improve the efficiency of the method, we have mentioned the use of predefined patterns to mesh the terminal cells (internal or external). From a practical point of view, each terminal cell is analyzed so as to determine its neighborhood and the result leads to a classification. A given pattern is associated

[5]Moreover, they are used to this end in other mesh generation methods (see Chapters 6 and 7).

with each class corresponding to a list of elements forming a conforming covering up of the cell.

Decomposition-related issues. We have already noticed that boundary integrity is usually not preserved with such methods. Moreover, the domain decomposition is related to its geometry as well as to its relative position in the classical frame of coordinates. In fact, each geometric transformation (translation, rotation) applied on the domain leads to different decompositions, thus to different meshes. Figure 5.16 shows different meshes obtained by rotating a rectangle, initially axis aligned, of angles of 10, 20, and 30 degrees respectively.

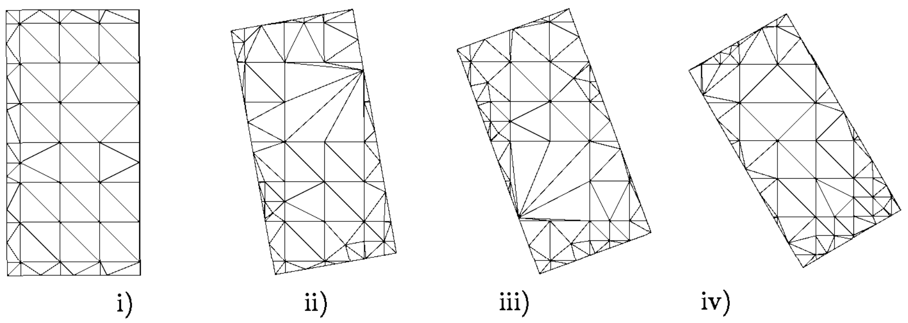


Figure 5.16: *Decompositions associated with a unique geometry: initial domain i); rotations of angles* $\alpha = 10°$ *ii);* $\alpha = 20°$ *iii); and* $\alpha = 30°$ *iv). Notice that the resulting meshes are dramatically different.*

In two dimensions, no convergence problem is faced. The only tedious problem is related to the proximity of two sides (almost tangent) that the tree construction will try to separate. This problem can be solved for instance, by slightly distorting a cell so as to insert one of its corners in between the two items. The same strategy can be used in three dimensions to avoid convergence problems.

Notice finally that tree-based methods allow isotropic meshes to be created. The generation of anisotropic meshes (in arbitrary directions, Chapter 21) appears almost impossible (except in some specific cases, for instance, in boundary layers, [McMorris, Kallinderis-1997]).

Memory requirements and data structures. Memory requirements correspond to the data structures needed to store the tree and the mesh. The main structures must allow access to:

- the list of vertex coordinates,
- the boundary items,
- the list of element vertices,
- the element adjacency relationships (in the optimization operations),

- the cell adjacency relationships,
- the list of boundary items per cell,
- other local resources.

5.3 Governed tree-based method

In numerous applications, the resulting meshes must satisfy specific properties, for instance, related to the local element sizes, to the density of the elements, etc. The construction scheme then involves using a so-called *governed* or *controlled* mesh generation method that makes it possible to match the desired requirements.

Such a control can be defined using an element size distribution function. This input data is in fact common to most unstructured mesh generation methods. A widely used technique consists of using a grid or a background mesh in which the elements (cells) keep track of the information related to desired local element size. From the discrete information (associated with the vertices of this covering up), an interpolation scheme to construct a continuous size distribution function can be used. This approach consists of defining a control space (Chapter 1). In our case, the control structure is obviously a tree.

Control space

Suppose the size distribution function is known in a discrete manner, for instance, at the vertices of a background mesh. From this discrete specification, a continuous size map is constructed using an interpolation scheme.

Tree-based construction of the size distribution function. Let P be an arbitrary point in the domain. The size $h(P)$ at this point can be interpolated from the sizes $h(P_i)$, $i = 1, ..d$ (with d the space dimension) at the vertices P_i of the element enclosing P. Suppose that a decomposition tree is available (generated, for instance, according to the classical method described in the previous section). The points of the structure (the cell corners) will be assigned the size values. More precisely:

- For a given corner Q of a tree cell,
  - find the element K of the background mesh enclosing Q,
  - calculate $h(Q)$ using a P^1-interpolation of the sizes $h(P_i)$ at the vertices of K.

With this technique, we obtain a value $h(Q_i)$ associated with each corner of the tree cells. Then, to find the size at any point P in the domain, we can simply use a Q^1 interpolation based on these sizes:

- For a given a point P,
  - find the cell containing P,

- calculate $h(P)$ using a Q^1 interpolation of the sizes $h(Q_i)$ at the cell corners.

When the background mesh is an empty mesh (with no internal point), these techniques result in a size distribution function that is exclusively related to the boundary discretization. However, if no background mesh is provided, it is still possible to construct a continuous size distribution function after the tree construction stage. In this case, the average value of the side lengths of the cell surrounding a corner is associated with each cell corner. A so-called *implicit* size map is then defined, which takes into account the boundary discretization and the domain geometry.

Remark 5.16 *Notice that the discrete evaluation of the sizes as described above may result in an undesirable filtering effect if the cell sizes are not well adjusted.*

Governed tree construction

Once the control space has been defined, it is possible to modify the classical method so as to create size compatible elements with respect to the given specifications. As the elements sizes are related to the cell sizes they are created in, the basic idea is to modify the classical tree construction procedure to take these specifications into account.

The first stage of the construction remains unchanged. The boundary discretization items are successively inserted into the current tree. The resulting tree is thus adapted to the domain geometry (more refined in high curvature regions or when two sides are close and, conversely, less refined far from the boundary). Then, the tree is used together with the vertices of the background mesh to control the cell sizes.

At this point two approaches are possible. The first one consists of constructing the size distribution function as described above. It is necessary to check that the size of each terminal cell (a square in two dimensions and a cube in three dimensions) is compatible with the average sizes associated with its corners. If not, the cell is recursively refined until the length of each sub-cell becomes less than or equal to the desired average length.

An alternative approach consists of initializing the sizes at each cell corner with an average value between the length of the adjacent cells (implicit map). Then, each vertex of the background mesh is embedded into the tree. Given the size $h(P)$ at a mesh vertex P, this size is compared with the size $h'(P)$ obtained using a Q^1 interpolation at the cell corners. If $h(P) < h'(P)$, the cell is refined and the process is repeated on the sub-cell enclosing the point P. Notice that the values associated with the cell corners are updated during the cell refinement.

In both cases, the tree is balanced according to the [2:1] rule.

Governed optimization

As mentioned in the classical case, it is usually very useful to optimize the mesh resulting from a governed spatial decomposition method.

Edge length. Let AB be a mesh edge and let $h(t)$ be the size variation along the edge, such that $h(0) = h(A)$ and $h(1) = h(B)$. The normalized edge length l_{AB} is defined as (Chapters 1 and 10):

$$l_{AB} = d_{AB} \int_0^1 \frac{1}{h(t)}\, dt\,,$$

with d_{AB} the Euclidean distance between the two endpoints A and B. Then, the edge AB is said to conform with the size specification if its length is such that:

$$\frac{1}{\sqrt{2}} \leq l_{AB} \leq \sqrt{2}\,. \tag{5.3}$$

The coefficient $\sqrt{2}$ can be retrieved when comparing the length of the initial edge with the lengths of the two sub-edges after splitting the original edge. We are looking for the configuration minimizing the error in distance to the unit length.

Local modifications. As for the classical case, the mesh quality is improved iteratively. The goal is to enforce unit edge lengths with respect to the specified metric (that is locally conforming to the size specification) for all mesh elements. Moreover, well-shaped elements are still aimed at. Therefore, a two-step optimization method is carried out, that first attempts to remove ill-shaped elements before optimizing the mesh in a global fashion. The second stage tends to improve shape quality as well as size quality using mesh modification operators (Chapters 17 and 18).

For practical reasons, the mesh elements are first optimized based on a size criterion and then the mesh shape quality is improved.

Application examples

In this section, several mesh examples obtained using a spatial decomposition method (classical or governed case) are proposed, in two and three dimensions. Figures 5.17 and 5.18 illustrate these examples. Table 5.1 gives some characteristic numerical values for different computational domains.

-	np	nf	ne	Q_M	t (sec)
Example 1	465	678	1,629	4.3	12.
Example 2	2,548	4,028	8,761	8.2	174.3
Example 3	5,792	7,506	22,681	8.2	346.5
Example 4	10,705	9,412	48,667	6.	115.8

Table 5.1: Statistics for different meshes obtained using a spatial decomposition method (the boundary discretization is the only data available to construct these meshes).

-	np	ne	t (sec)	v	t_B	t_T
Example 1	1,060	3,558	24.9	8,580	12.	2.
Example 2	4,089	15,947	74.9	12,780	30.1	2.2
Example 3	25,887	124,213	278.5	26,760	99.8	13.
Example 4	27,686	134,826	375.4	21,540	115.8	42.
Example 5	79,820	380,183	823.8	27,689	305.7	36.5
Example 6	104,015	498,509	1,100	27,191	400.6	47.5

Table 5.2: Efficiency and scalability of a spatial decomposition method, in three dimensions.

In Table 5.1, np, nf and ne denote respectively the number of vertices, the number of surface triangles and the total number of mesh elements, $\mathcal{Q}_M$ is the quality of the worst shaped element (after optimization) and t is the CPU time (Sun Ultra-1 Sparc workstation) required to mesh the domain. Table 5.2 shows the efficiency of the method, v is the average speed of the method (i.e., the number of elements created per minute). After the tree construction, the terminal (internal and external) cells are triangulated using templates and the boundary cells are meshed in a fully automatic manner. In this table, t_B and t_T denote the computational times devoted to tree construction (insertion of the boundary discretization) and the triangulation of non-boundary leaves, respectively.

The results clearly show that the speed v ranges from $8,500$ to $27,000$ elements per minute. The difference between the lower and upper bound can be explained by noting that as the number of mesh elements increases, the part devoted to tree construction (boundary discretization) becomes dominant, but the triangulation of the non-boundary leaves remains quasi-constant in time. The variations can also be explained by the geometric complexity of the computational domains that affect the tree construction stage.

As a conclusion, it could be observed that the theoretical complexity of tree-based mesh generation methods is not easy to calculate. However, it can be noticed that the computational cost of meshing process for the internal cells is proportional to the number of elements created in these cells and so to the number of internal cells in the tree decomposition. The latter number is related to the geometry of the domain, which is not the case for advancing-front or Delaunay-type mesh generation methods. The complexity of this category of methods is theoretically $\mathcal{O}(n \log(n))$, with n being the number of mesh elements [Schroeder, Shephard-1989]. The term in $\log(n)$ can be explained by the use of a tree structure (in neighborhood and searching operations). In practice, the speeds observed are even closer to $\mathcal{O}(n)$ for the overall process.

5.4 Other approaches

In this section, we briefly mention several specific applications of the tree-based mesh generation method. In particular, we emphasize combined approaches (like Delaunay-octree, advancing-front-octree).

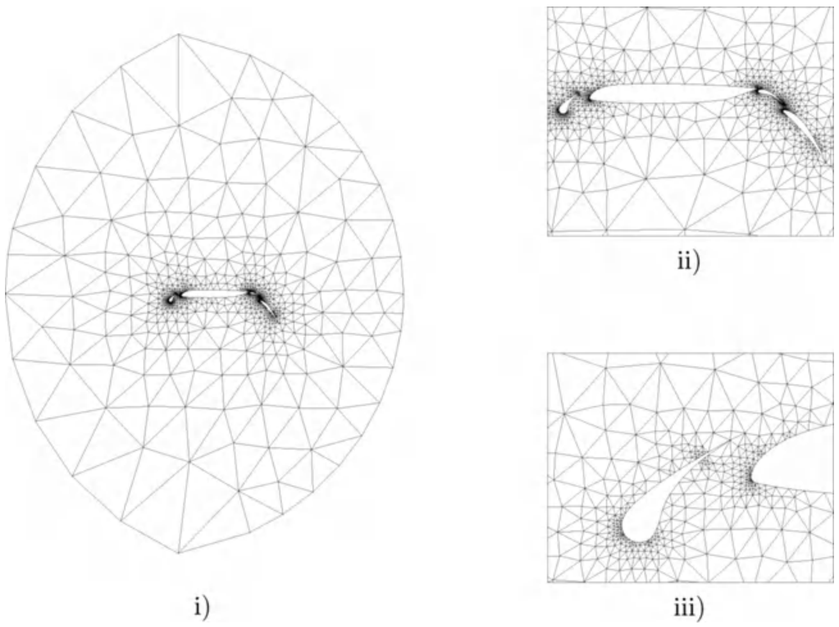


Figure 5.17: *Two-dimensional mesh of a computational domain and close-ups around the wing profile.*

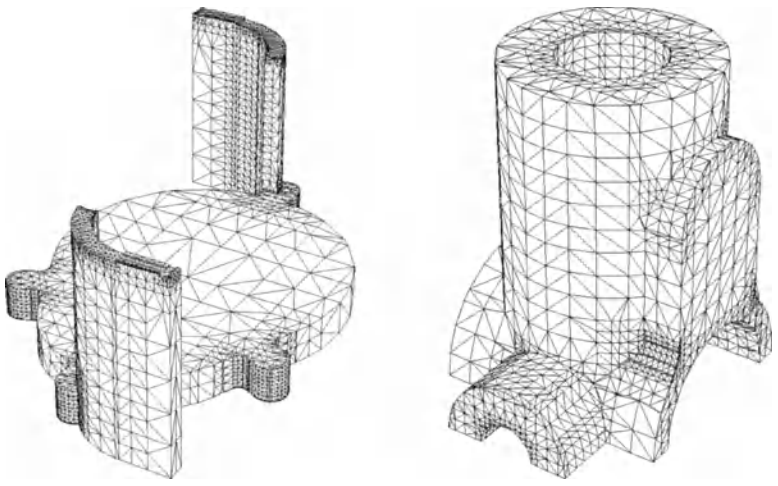


Figure 5.18: *Examples of meshes obtained using an octree-based method in three dimensions (data courtesy of the MacNeal-Schwendler Corp.).*

Combined approaches

The strong point with tree-based decomposition methods is related to their ability to combine a control space and a neighborhood space in a single data structure. On the other hand, the main drawback of such methods is the lack of a real strategy for internal point creation based explicitly on the element (shape) quality. Indeed, it should be recalled that the mesh vertices correspond to the terminal cell corners, which actually tends to rigidify the resulting mesh[6]. Moreover, numerous numerical problems are related to intersection computations in the vicinity of boundary items. This situation is due to the fact that the boundary is somehow embedded into the tree structure without any specific geometric consideration.

This is why some authors have found it useful to use the tree structure as a control space (to get the element sizes information) as well as a neighborhood space (to improve the searching operations), while using other approaches to create internal points and to connect them (i.e., the mesh generation, strictly speaking).

- Delaunay/octree method.

Mesh generation based on a Delaunay-type algorithm generates a mesh point by point, each point possibly allowing the construction of several elements simultaneously. More precisely, the Delaunay kernel (Chapter 7) is a procedure to connect the mesh vertices together (i.e., to generate the mesh topology). First, the decomposition tree is built as in the governed case described above. The element sizes are then associated with the cell corners, the cell sizes being locally compatible with the desired element sizes. The cell corners as well as possible internal points (whenever the size variations are too great) are then inserted into the current mesh using the Delaunay kernel procedure.

Remark 5.17 *Notice that the Delaunay algorithm used in this approach must be constrained so as to preserve the topology and the discretization of the boundary (cf. [Schroeder, Shephard-1988], [Sapidis, Perucchio-1993]).*

- Advancing-front/octree method.

In this case, the internal points are created and inserted into the domain using an advancing-front technique (Chapter 6). The optimal location of a candidate point is determined from the local sizes associated with the cell corners (these possibly being corrected so as to enforce a given mesh gradation). This technique has been successfully applied in mesh generation for viscous flow simulations in computational fluid dynamics [McMorris, Kallinderis-1997].

[6]In fact, it can be stated that this type of method is not really a mesh generation method [Schroeder, Shephard-1990].

Other approaches

The spatial decomposition techniques can be applied to the mesh generation with quadrilateral or hexahedral elements. In the approach suggested notably by [Schneiders *et al.* 1996] in three dimensions, the root of the tree is subdivided into 27 cells[7] which are then recursively subdivided until a given size criterion has been satisfied.

5.5 Extensions

In this section, we will mention the curve and surface mesh generation issues and make some comments on the use of a spatial decomposition method in the context of mesh adaptation and image encoding or compression.

Curve and surface mesh generation

In the previous sections, we have described the general scheme of the tree-based mesh generation method, under the assumption that the boundary discretization is provided. However, it is possible to modify this scheme so as to construct the boundary discretization as well, this stage possibly being followed by the classical mesh generation procedure. This approach assumes that an adequate representation of the domain geometry is available (for instance, provided by a geometric modeling system).

Problem statement. Surface mesh generation is a renowned complex problem notably related to the nature of the surfaces. Two approaches are commonly used to generate surface meshes. Whichever approach is used, the goal is to obtain a so-called geometric mesh such that the gap between the mesh edges and the mesh elements and the underlying surface is bounded. This control can be expressed in terms of size specifications (i.e., the edge lengths are proportional to the principal radii of curvature; see Chapters 14 and 15).

The indirect approach is used for parametric surfaces. It consists of meshing the surface via a parametric space using a classical mesh generation technique (for instance, an advancing-front or a Delaunay-type method). The characteristic and the advantage of this approach is that the problem is reduced to a purely two-dimensional problem. The surface can be meshed using a two-dimensional mesh generation technique in the parametric space, then projecting the resulting mesh onto $\mathbb{R}^3$, via a (sufficiently smooth) transformation. However, to obtain a suitable surface mesh, it is necessary to use a mesh generation method able to generate anisotropic meshes, the metric map definition being based on the intrinsic properties of the surface. Unfortunately, this feature immediately excludes the tree-based method.

[7]This 27-cell pattern has been chosen because of the relative ability with which a hexahedral mesh can be extracted from the tree structure.

On the other hand, if the surface is not considered via a parametric representation, the following tree-based technique can be used to create a surface mesh directly, without any transformation ([Grice *et al.* 1988], [Shephard, Georges-1991]).

Control of the geometric approximation. The depth p of the tree can be related to the edge length h and to the size b of the bounding box $\mathcal{B}(\Omega)$ by the relation $p = \log_2(b/h)$. As the edge size h is proportional to the minimum of the principal radii of curvature, denoted ρ (cf. Chapter 15), we thus obtain a lower bound for the tree depth:

$$p \geq \log_2\left(\frac{b}{\rho\,\alpha}\right),$$

where the coefficient of proportionality α is such that $\alpha = 2\sqrt{\varepsilon(2-\varepsilon)}$, where ε is a given tolerance value. More precisely, let h be the distance from a point to the supporting edge, and let d be the length of this edge, it is then necessary to enforce the relation:

$$\frac{h}{d} < \varepsilon,$$

ε being the given relative tolerance value (cf. Figure 5.14, left-hand side).

Tree construction. The proposed method follows the classical general scheme, with the difference that the tree construction stage is now based on an iterative algorithm for inserting the items of the geometric model (and not the items of the boundary discretization). The points (in fact, the singular points and more generally all points of C^0 discontinuity) are first inserted into the tree from the initial bounding box decomposition, according to the classical construction rules. Then, at each cell level, the geometric model is questioned (using a system of geometric queries) so as to determine the intersections between the model edges and the cell sides. Depending on the number of such intersections, the current cell may be refined. The model faces are in turn inserted into the tree.

The local intrinsic properties of the surface (principal radii of curvature, vertex normals, etc.) are used to construct the tree. In practice, the cells are refined until the geometric approximation criterion is satisfied.

Main difficulties. The difficulties related to this kind of technique are essentially the numerical problems encountered during the calculation of the intersections between the geometric model entities and the tree cells.

The following example emphasizes the complexity of the tree construction stage. Consider a cell intersected by at least two model edges. In Figure 5.19, the curved edge is inserted into the current cell, thus leading to the creation of a segment AB. Then, the edge PQ is inserted and results in the creation of the straight segment PQ, which partially overlaps segment AB. This kind of problem can be solved either by subdividing the two curved edges into several straight segments (Figure 5.19, middle) or by refining directly the cell (same figure, right-hand side).

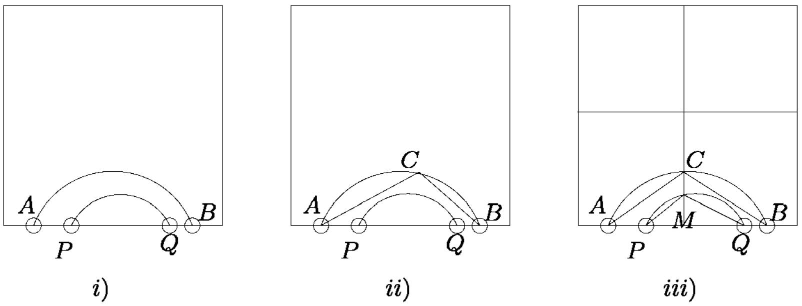


Figure 5.19: *Insertion of two curved edges into the same tree cell. i); problem of overlapping straight segments coming from the discretization. ii); the problem is solved by splitting the curved edge AB into two sub-segments. iii); the problem is solved by refining the quadrant.*

The difficulty is even greater in three dimensions. The intersections between the model face edges and the cell sides are calculated. The resulting intersection points are then connected, each octant face being processed separately, so as to approximate the intersection curves [Shephard *et al.* 1996].

Remark 5.18 Finite Octree *meshing method, for instance, is a complete representation of the geometric model that has been partitioned. It maintains a hierarchical understanding of the portions of the model intersected by an octant. However, problems may occur, mostly related to the geometric tolerance, which may result in ambiguous configurations; Figure 5.20 shows such a problem: the sides intersected by the curve and that intersected by the discretization are not the same. Therefore, if the boundary discretization is built at the same time, this configuration can lead to consistency problems when the intersections between the edges and the quadrants are propagated into the tree structure.*

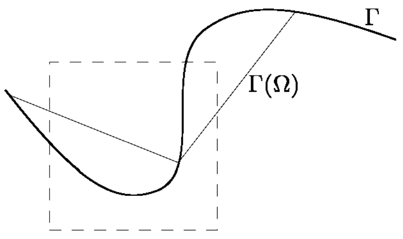


Figure 5.20: *Potential problem related to too great a tolerance value, in two dimensions. The sides of a quadrant intersected by the discretization and that resulting from the intersection of the underlying curve are not the same.*

Mesh adaptation

The objective of the computational adaptation scheme is to ensure the reliability of the solutions in the numerical simulation based on finite element methods. From a theoretical point of view, adapting the meshes to the physical properties of the considered problem should improve the accuracy of the numerical solutions and should guarantee the convergence of the computational scheme. The adaptation is based on a governed mesh generation method, in which the size map is provided by an *a posteriori* error estimate.

Quadtree-octree based methods can thus be used in this context, see Chapter 21.

Image and quadtree

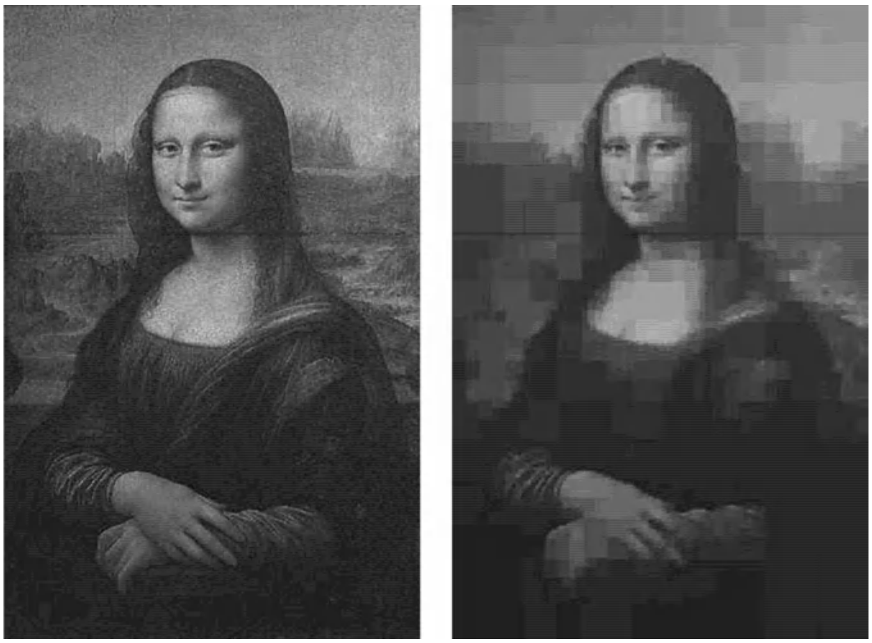


Figure 5.21: *Image partitioning: The "Joconde" by Leonardo da Vinci. Left hand-side: Initial image, a* 289×440 *matrix of pixels. Right hand-side: image after the quatree-based partitioning with a given compression factor. The quadtree is made up of about* 2,800 *cells.*

The quadtree structure and the corresponding mesh can be used to encode an image with a given compression ratio. For instance, in Figure 5.21, the initial image is a bitmap with 289×440 pixels, which can be seen, roughly speaking, as a quad mesh with $126,432$ elements while the resulting quadtree only includes a limited number of quads.

Given the initial image, a $n \times m$ matrix of pixels, a quadtree is constructed where the refining criterion (thus the stopping criterion) is related to the difference between two "consecutive" pixels. Considering a series of pixels in a given cell, this quad is subdivided into four sub-cells if the colour changes more than a threshold value from a cell's corner to the examined pixel. This simple application of the hierarchical aspect of the quadtree structure is clearly not competitive with modern method (such as in the jpeg format) but just an exercise about quadtrees.

★
★ ★

Mesh generation methods based on a spatial tree decomposition technique are robust, efficient and flexible. Such methods allow the construction of classical as well as governed meshes (conforming to a prescribed size map). The use of the spatial decomposition structure as a control space is also an idea common to other methods (and applies in other fields of applications such as image processing). Moreover, the boundary discretization of the computational domain can be created using this approach. However, this kind of technique is not able to create anisotropic meshes (in which the stretching directions of the elements are arbitrary). In addition, the use of the cell corners to define the mesh vertices rigidify the resulting meshes (the optimization stage is usually more computationally expensive than with advancing-front or Delaunay-type mesh generation methods; see Chapters 6 and 7).

Mesh Generation : Application to Finite Elements
Pascal Jean Frey & Paul-Louis George
Copyright © 2008, ISTE Ltd.

Chapter 6

Advancing-front Technique for Mesh Generation

The advancing-front technique for mesh generation has been investigated for more than 35 years since the pioneering work of [A.George-1971], who studied a two-dimensional case. The classical advancing-front method, in its current form, was first described by [Lo-1985] and [Peraire *et al.* 1987]. Numerous improvements in this technique have been proposed over the years, first by [Löhner, Parikh-1988], [Golgolab-1989] and more recently by [Mavriplis-1992] and [Shostko, Löhner-1995]. This approach is now a very powerful and mature technique for generating high-quality unstructured meshes composed of simplices (triangles or tetrahedra) for domains of arbitrary shape. Variants of this technique have even been proposed to generate quadrilaterals or hexahedra in two and three dimensions (cf. [Blacker, Stephenson-1991], [Blacker, Meyers-1993]).

Classical advancing-front approaches start from a discretization of the domain boundaries as a set of edges in two dimensions or a set of triangular faces in three dimensions. The name of this class of methods refers to a strategy that consists of creating the mesh sequentially, element by element, creating new points and connecting them with previously created elements, thus marching into as yet unmeshed space and sweeping a *front* across the domain. The process stops when the front is empty, i.e., when the domain is entirely meshed. The *front* is the region(s) separating the part (or parts) of the domain already meshed from those that are still unmeshed. Hence, depending of the strategy, the front can have multiple connected components (Figure 6.1) or not (Figure 6.2).

One of the critical features of any advancing-front approach is the creation of internal points. This procedure should result in elements that, usually, satisfy size and shape quality criteria. Such a size criterion, and thus the corresponding element-size distribution function, prescribes the desired element shapes and sizes (review the notion of control space in Chapter 1).

The main advantage of current advancing-front techniques is their heuristic mesh generation strategy, which tends to produce high-quality elements and nicely graded meshes. Moreover, in contrast to some other automatic methods, bound-

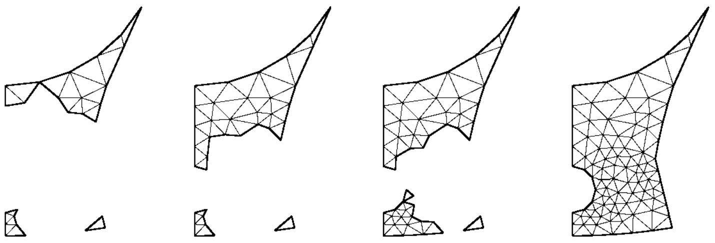


Figure 6.1: *Various stages illustrating the multiple components of an evolving front in two dimensions depending of the selected strategy.*

ary integrity is always preserved, as the discretization of the domain boundary constitutes the initial front, which is not the case for some other mesh generation methods (for instance, see Chapters 5 and 7). On the other hand, convergence

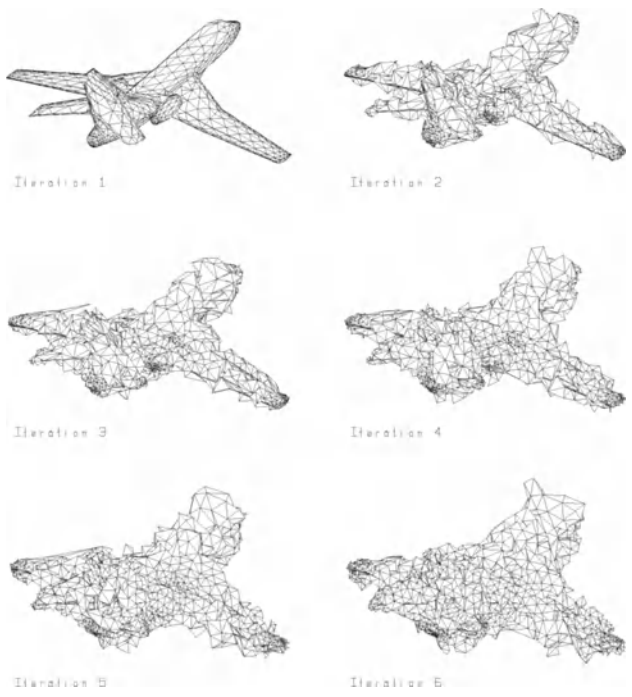


Figure 6.2: *A different strategy, here by means of inflating the current front. The domain to be meshed is the volume between an aircraft (whose surface mesh is the initial front) and a bounding box.*

problems can occur, especially in three dimensions, as it is not always clear how to complete a mesh of the entire domain. In other words, some regions can be found which are not easy to fill up with elements (mostly in three dimensions).

★

★ ★

In the first section of this chapter, the framework of a classical advancing-front method is outlined, which takes into account most features of this class of techniques. The critical features of the approach are described with emphasis on specific topics such as optimal point creation, geometric checks and mesh gradation. Some indications are also given about convergence issues. The second section explains how robustness and reliability issues can be addressed using a control space. Then, in the third section, some details are discussed about a governed method where the control corresponds to the data of a control space. The coupling of the advancing-front method with a Delaunay technique is discussed in the fourth section and the capabilities of combined or hybrid approaches are raised. Finally, possible extensions of the method to anisotropic mesh generation, surface mesh generation and mesh adaptation (for the mesh adaptativity problems discussed in Chapter 21) are introduced in the fifth section.

6.1 A classical advancing-front technique

An advancing-front technique is generally part of a mesh generation procedure including three successive steps:

Step 1: mesh parameterization: boundary description. specification or construction of a function defining element size distribution, etc.

Step 2: boundary discretization.

Step 3: the creation of the field vertices and elements, in other words, the advancing-front method itself.

This general scheme which will be found in other methods such as the Delaunay type method (Chapter 7), is slightly different from that of a *quadtree*-based method (Chapter 5). Indeed, in contrast to the latter case, here the boundary mesh is an input data that must be preserved.

Following a classical advancing-front approach (Step 3 of the above scheme), the mesh vertices are created as part of the procedure of mesh element creation. Numerous variants of this technique have been proposed by [Bykat-1976], [Peraire *et al.* 1988], [Löhner, Parikh-1988] [Dannelongue, Tanguy-1991] and also [Jin, Tanner-1993] and [Farestam, Simpson-1993]. Despite several differences between these approaches, a classical advancing-front method is, inherently, based on a local insertion mesh generation strategy relying on geometric criteria.

Although the fundamental steps of mesh generation are the same in two and three dimensions, the development of robust, reliable and efficient algorithms in three dimensions is much more tedious than that in two dimensions.

General scheme

Formally speaking, the advancing-front procedure is an iterative procedure, starting from a given boundary discretization, which attempts to fill the as yet unmeshed region of the domain with simplices. At each step, a front entity is selected and a new (optimal) vertex is created and possibly inserted in the current mesh should it form a new well-shaped simplex. The main steps of the classical advancing-front algorithm can be summarized as follows:

Step 1: Preliminary definitions.

- data input of the domain boundaries (i.e., the boundary mesh $\mathcal{T}$)
- specification of an element-size distribution function (governed mesh generation) or the best possible construction[1] of such a function (classical problem).

Step 2: Initialization of the front $\mathcal{F}$ with $\mathcal{T}$

Step 3: Analysis of the front $\mathcal{F}$ as long as $\mathcal{F}$ is not empty:

(a) select a front entity f based on a specific criterion,

(b) determine a best-point position P_{opt} for this entity,

(c) identify if a point P exists in the current mesh that should be used in preference to P_{opt}. If such a point exists, consider P as P_{opt},

(d) form an element K with f and P_{opt},

(e) check if the element K intersects any mesh entity. If this check fails, pick a new point P (if any) and return to 3.*d*.

Step 4: Update the front and the current mesh:

- remove entity f from the front $\mathcal{F}$ and any entity of $\mathcal{F}$ used to form K,
- add the entities of the newly created element K members of the new front,
- update the current mesh $\mathcal{T}$.

Step 5: If the front $\mathcal{F}$ is not empty, return to Step 3.

Step 6: Mesh optimization.

In the following sections, we give some details about these steps while discussing the main difficulties involved in the process.

[1]See Sections 6.2 and 1.6.

Main difficulties. The above scheme gives the principles of the method but does not reveal a series of delicate issues that must be addressed. In particular, there is no guarantee of robustness or convergence in three dimensions[2]. In fact, the key steps of the method are primarily concerned with:

- the selection of one entity in the front,
- the front analysis and the various possibilities for defining the optimal points,
- the element validation at the time an element is constructed,
- the use of appropriate data structures.

The three-dimensional implementation of this approach requires a great deal of attention to ensure a robust algorithm (especially regarding convergence, i.e., the guarantee that the method really completes the mesh) with a certain extent of efficiency. However, the main difficulties of an advancing-front type meshing process are relatively easy to identify. They are primarily related to

- the proper identification of the local situation at some neighborhood of a point or a region,
- the adequate intersection identification (for edges, faces, etc.) used, for instance, to validate one element before inserting it or to detect if a point falls outside the domain, etc,
- the definition of an optimal location for a point or the selection of an existing point when elements must be created based on a given front entity.

Clearly, any newly-created mesh point must result in valid and well-shaped element(s), and must prevent, as much as possible, any configuration from further meshing difficulties.

Remark 6.1 *The difficulties are then related to the methodology (optimal point definition) or to numerical aspects (intersection checks, efficient access to some neighborhood of an entity).*

Preliminary requirements

The proper achievement of an advancing-front method relies on a certain number of assumptions (about the data input) and on a well-suited choice about the necessary data structures.

[2]Although no specific difficulty is encountered in two dimensions.

About the domain boundary mesh. The boundary mesh is supplied together with an orientation convention. In a manifold case, this means, in two dimensions, that the polygonal discretization (the boundary mesh) consists of segments defined and traversed in a given direction (based on an orientation convention, for instance, the domain is in the region on the "right" of the given boundary). Similarly, in three dimensions, the faces members of the domain boundaries must be orientated in an adequate way, the domain still being defined by its position with respect to these faces. Furthermore, this requirement is used to identify the relative position of the domain with respect to a segment (a face) of the boundary (indeed, with respect to the half-spaces separated by this entity[3]).

The non-manifold case is more subtle in the sense that two situations may arise. There is an edge (a series of edges) which is not a boundary of a connected component of this domain (such an edge could be an isolated edge or may have only one endpoint located in a boundary) or, conversely, such an edge belongs to the interface between two connected components. In the first case, there is no longer an orientation problem, as the domain is on one side of the entity and is also on the other side of it. In the second case, the entity separates two components and therefore has a different orientation for each of these components. Figure 6.3 depicts a case where the boundary discretization includes several connected components.

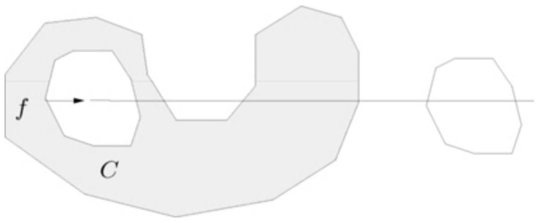


Figure 6.3: *Orientation of a boundary connected component C. Entity f must be re-orientated in the case where the region to be meshed is the shaded part.*

Data structures. Based on the nature of the operations needed in an advancing-front method, it is relevant to define some specialized data structures, [Löhner-1988], [Bonet, Peraire-1991]. The objective is twofold: to make the management of the front fast and easy once the process progresses and, on the other hand, to provide easy access to all the entities present in some neighborhood of a given point or a given region.

The front management requires a topological data structures that enables the suppression or the insertion of an entity: an edge in two dimensions, a face in three dimensions; cf. Chapter 2.

Access to some neighborhood of a point is made using a data structure like a neighborhood space (cf. Chapter 1).

[3]Actually, an orientation convention can be retained such that the face normals are orientated towards the interior of the domain.

Analysis of the front

The mesh elements are created based on the edges (faces) of the current front. The way to do this is related to the way in which the front is analyzed and thus to the choice of a particular front entity.

The identification of a front entity from the generation front is a non-negligible task that can be performed in various ways depending on what benefit is expected. In general, the mesh generation does not proceed locally, but rather bounces randomly from one side of the domain to another. It seems obvious that this chaotic behavior has several undesirable side effects [Frykestig-1994]:

- unnecessary complexification of the front,
- loss of the possibility of using the previously collected loacl information,
- overall degradation of the performances (due notably to an increase in cache memory management: see Chapter 2).

It is always stated that these types of problems should not arise with a smooth element-size distribution function. In three dimensions, however, this is not really the case for various reasons. First of all, the front generally presents a chaotic aspect (as previously mentioned), even in a simple geometric context and for smooth size functions (such as when a constant size is specified). Moreover, the desired size distribution can be tedious to follow (for instance, in CFD problems where size chocks and/or boundary layers must be considered).

More schematically, according to [George, Seveno-1994], the front entity can be chosen in a set of mesh entities (a subset of all front entities) corresponding to:

- all entities satisfying specific properties (angles, areas, lengths, etc.),
- an offset of a part of the initial front (at the beginning) and of the current front after,
- all the front entities.

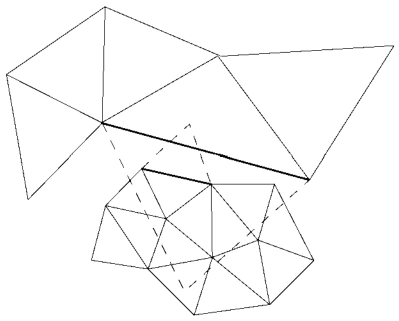


Figure 6.4: *Merging fronts of very dissimilar sizes is a possible case of failure. The dashed lines illustrate the optimal elements that can be formed based on the selected front entities.*

An ordering can be defined over this set using several criteria. For instance, the selected front entity can be the entity leading to the smallest element, in order to avoid large elements crossing over regions of small elements [Löhner-1996b]. This can be useful in the case of two merging fronts, if the element size varies rapidly over the region between these fronts (as illustrated in Figure 6.4). Other approaches consist of using the shortest edge (of a face, in three dimensions) [Peraire *et al.* 1992], the smallest entity [Jin, Tanner-1993] or even ordering the entities with respect to their areas [Möller, Hansbo-1995]. The selection of the front entity can also take the relative position of this entity with respect to its neighbors (incident entities). These approaches are usually carried out using a heap structure [Löhner-1988], for instance, based on binary trees (Chapter 2).

As the objective is to generate a good-quality mesh, another technique aims at locally creating the best element possible based on the worst front entity. This approach clearly makes sense in three dimensions since in two dimensions it is *a priori* always possible to form an equilateral triangle based on an arbitrary edge (provided the point to be defined is inside the domain). This is true, specifically, only in the configurations with corners where, in two dimensions, it is impossible to connect two adjacent edges having a rather acute angle to each other [Jin, Tanner-1993]. Finally, the approach suggested by [Rassineux-1997] and [Seveno-1997] tends to minimize the size of the front and to take advantage of the (local) information previously collected. As a consequence, the number of intersection checks is also reduced, thus impacting favorably the overall meshing time. This approach is claimed to make mesh generation easier as it seems to reduce the number of degenerate cases. In practice, the example illustrated in Figure 6.8, becomes highly unlikely to happen as the algorithm tends to *convexify* the front. The approach starts from a given front entity and deals successively with any neighboring front entity that has not already been considered, thus resulting in a "spiral-like" progression.

Exercise 6.1 *Implement the spiral procedure using a coloring algorithm and a FIFO (first in-first out) data structure (see Chapter 2). Show that the previous algorithm proceeds by connected components.*

Remark 6.2 *Regardless of the approach selected, the main concern should always be to design and implement a robust algorithm (i.e., for which convergence can be guaranteed) resulting in high-quality meshes.*

Field point creation

Central to the advancing-front technique, the creation and insertion of an optimal point from a front entity requires a lot of attention. Let us assume that a front entity has been selected and that the desired element size can be known locally[4]. The objective is to discretize the domain in a set of well-shaped elements of arbitrary gradation. This can be carried out in the following steps:

[4]See Chapter 1 for the use of a control space to locally determine the desired element size.

- a local stepsize, denoted h, (e.g. a node spacing function or an element size value) is associated with each mesh vertex, based on an average value of the edge lengths (surface areas) sharing the vertex for a boundary vertex and computed using function h for the mesh vertices after their creation,
- for a given entity, a point is defined (see hereafter) which is then connected with this entity,
- this choice is validated in accordance with the geometry and the size function. In the case of a point rejection, the previous step is repeated while picking another point.

Three types of different points are tried as candidates for element vertices:

- the optimal point with respect to the given front entity,
- some already existing points in the current mesh which are close (in some sense) to the front entity under analysis,
- some (guest) points created in a given neighborhood of the above optimal point.

As will be seen, these points will be the possible *candidates* from which the element vertices will be chosen.

Optimal point. Several construction schemes have been proposed for determining the location of the optimal point associated with a front entity. In two dimensions, a purely geometric idea defines as an optimal point that which allows for the construction of an equilateral triangle when combined with the given front edge. In three dimensions, the optimal point is that which allows the most regular tetrahedral element possible to be built from a given front face (see Figure 6.5). An approach that takes advantage of the desired local size $h(P)$ (i.e., the edge length) in any point P is also a widely used solution. Specifically, this is a *natural* solution for a governed mesh generation method (as will be seen later). Thus, in three dimensions, the classical technique advocates positioning the point along the normal direction passing through the centroid of the front entity at a distance related to the magnitude of the desired element size (i.e., the average distance from this point to the vertices of the front entity leads to the desired element size) [Peraire *et al.* 1992], [Jin, Tanner-1993].

Let K be a triangle of the front, whose centroid is denoted G, and let $h(G)$ be the desired element size at point G. If $\vec{n}_K$ represents the inward normal to K, the position of the optimal point P_{opt} can be calculated as follows:

$$\overrightarrow{GP_{opt}} = \alpha h(G).\vec{n}_K \,, \tag{6.1}$$

where α is a normalization coefficient such that if K is an equilateral triangle, the resulting tetrahedron will be regular (if the size map h is constant).

Exercise 6.2 *Determine the value of the coefficient α (hint: look at the specific above configuration).*

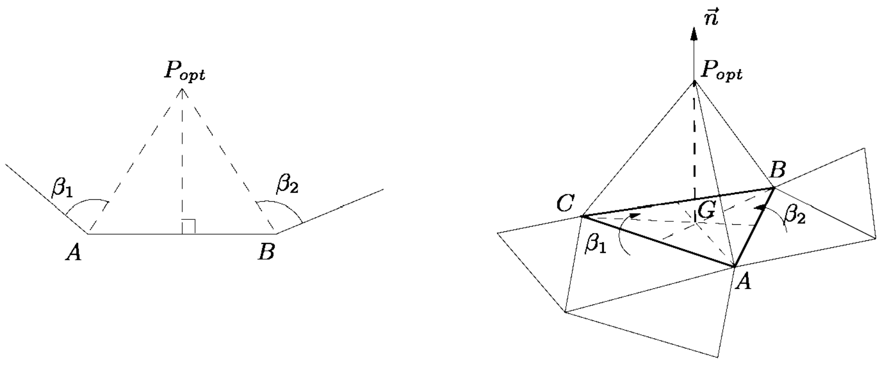


Figure 6.5: *Optimal point creation in two and three dimensions.*

Remark 6.3 *This construction assumes that the selected front entity is well-shaped. Nevertheless, a correction procedure has been proposed to account for distorted entities [Frykestig-1994], which basically consists of performing the computation in a* normalized *space, obtained by scaling the coordinates by the inverse of the desired mesh size value locally.*

Remark 6.4 *Once the optimal point P_{opt} has been located, the element size $h(P_{opt})$ at P_{opt} can be used to relocate P_{opt} so that the element size matches the desired element-size specification (cf. Section 6.2).*

The optimal point P_{opt} (with respect to a given front entity) is not directly inserted in the mesh. Indeed, it is usually preferable to use an existing vertex as the optimal point rather than introducing a new one. Therefore, the other possible candidates are identified, collected and possibly ordered according to their suitability to form well-shaped elements with the selected front entity.

Potential candidates. The identification of other candidate points is carried out according to a distance criterion. The closeness of a point is related to the local element size specification $h(P_{opt})$ at the optimal point P_{opt}. A point P can be considered as a candidate (i.e., is sufficiently close) if it belongs to the disk (ball) of radius $h(P_{opt})$ centered at P_{opt}, thus satisfying the following relationship:

$$||\overrightarrow{P_{opt}P}|| \leq h(P_{opt}). \tag{6.2}$$

Remark 6.5 *Given a front entity K, the points of the adjacent front entities K_i are automatically added to the set of admissible points, if the angles β_i (the dihedral angles in three dimensions) between K and K_i are less than a given threshold value (cf. Figures 6.5 and 6.6) irrespective of their distances to P_{opt}.*

Exercise 6.3 *Explain how a neighborhood space (Chapter 1) makes it possible to efficiently identify the mesh entities closely spaced to the given optimal point.*

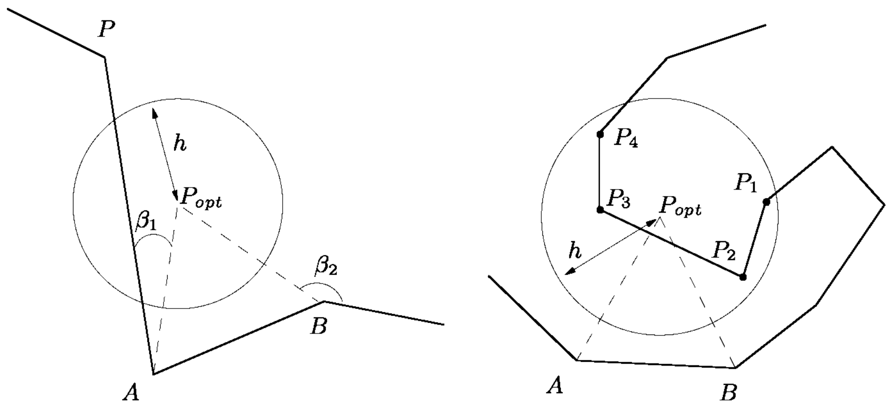


Figure 6.6: *Identification of all potential candidates for optimal point creation in two dimensions. Left: no candidate other than P_{opt} exists. Right: the P_i's represent all the possible candidates.*

Remark 6.6 *Several authors [Jin, Tanner-1993], [Frykestig-1994] advocate the use of* tree points *that are constructed on predetermined positions (close to P_{opt}) to resolve complex front configurations.*

Remark 6.7 *The input of specified points at the beginning of the construction (added to the boundary discretization) is also a frequently encountered case that induces some slight modifications of the algorithm.*

The set of potential candidates P_i is ordered according to the increasing distance from the optimal point P_{opt}. They will be considered for element generation with this ordering. The optimal point is also inserted in this set and will be considered prior to some of these points, according to the distance criterion.

Filtering. As the computation of the optimal point location P_{opt} does not explicitly account for existing vertices, an existing mesh vertex P can incidentally be closely spaced to P_{opt} which must then be discarded to the advantage of P.

Validation

The candidate points, including the optimal point, must of course satisfy certain *validity criteria* prior to being considered as potential mesh vertices. The purpose of the criteria is to ensure that the resulting mesh:

- is topologically and geometrically valid,
- is of good quality,
- conforms to the specified element-size distribution function.

Definition 6.1 *A mesh is topologically and geometrically valid if all mesh elements are conforming (Definition 1.2) and have positive surfaces (volumes).*

In practice, for a given candidate point P_i, conformity can be ensured by checking that none of the edges (faces) of the expected element intersects the front and that no existing mesh element contains point P_i.

Exercise 6.4 *Show that this geometric check can be used to determine whether a point P_i belongs to the domain (to the relevant connected component) or not.*

Definition 6.2 *A candidate element is a valid element if none of its edges (faces in three dimensions) intersect any front entity and if it does not contain any mesh entity (for instance, a vertex or an element).*

Notice that this latter case (where an element exists) is a somewhat specious case. For instance, in two dimensions, we encounter one element (or a series of elements) fully included in the element under analysis.

All this leads to performing a certain number of validation checks which reduce to intersection checks. The intersection checks are relatively straightforward to carry out (as the elements are supposedly straight-sided). However, if the geometry checks are not carefully implemented, they can be impressively time-consuming. For instance, [Löhner, Parikh-1988] reported that more than 80% of the meshing time can be spent checking the intersections. Hence, the overall meshing speed is strongly related to the algorithm used to check whether the elements intersect or not. Therefore, several efficient solutions have been proposed reducing the time to a mere 25% of the total meshing time [Löhner-1988], [Jin, Tanner-1993]. In short, one uses data structures (binary tree or quadtree/octree, etc.) and filters so as to reduce the number of entities that must be checked (Figure 6.7).

Remark 6.8 *The complexity of the intersection checks is almost constant (i.e., does not depend on the mesh size). Indeed, the complexity is proportional to the number of neighboring elements collected, which is itself related to the size of the neighborhood.*

Additional checks. From the computational point of view, the notion of validity as envisaged so far is not sufficiently reliable. Indeed, it is certainly not enough to consider exclusively the new element to be created, as its insertion may affect the front configuration in such a way that further insertions may be virtually impossible[5]. Therefore, a practical approach consists of analyzing the distances between the edges of the element to be inserted and the front edges. This check makes it possible to control the space left after the element has been inserted.

[5]This situation is especially acute in three dimensions.

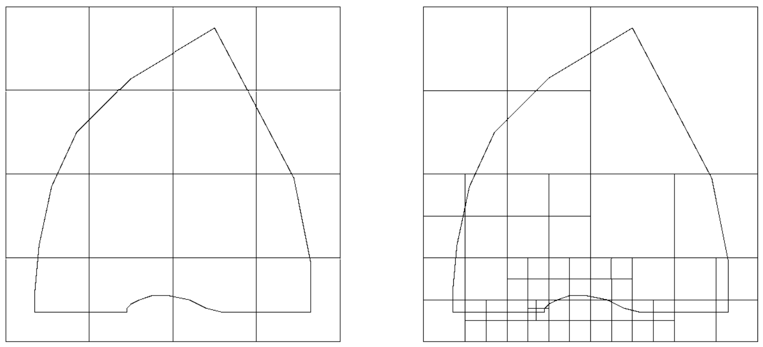


Figure 6.7: *For this example, a PR-quadtree structure (right-hand side) is better fitted to the domain geometry than an uniform grid (left-hand side) as a neighborhood space.*

In three dimensions exclusively, the worst problem encountered involves a configuration in which all front faces delimit a polyhedron that cannot be meshed without introducing a point. In this case, the problem comes down to determining a possible location for the internal point (Figure 6.8). Convex polyhedra are easy to triangulate, but additional points may be needed to mesh a non-convex polyhedron [Lennes-1911], [Schönhardt-1928].

This naturally leads us to convergence issues of advancing-front algorithms. So far, to continue the discussion, we assume that such a problem is not an issue, i.e., we assume that in the set of the candidate points there is always a non-empty set of *a priori* admissible points (for topologic and geometric validity).

Quality checks. At this stage of the evaluation procedure, a candidate element has been declared acceptable based only on geometric considerations. As the aim is to create well-shaped elements (as regular as possible), the introduction of a quality check may be desirable. A candidate point that successfully passed the geometric checks is nevertheless discarded if the resulting element shape quality is poor. Several quality measures have been proposed (Chapter 18), a *natural* measure for a simplex K being:

$$\mathcal{Q}_K = \alpha \frac{h_{max}}{\rho_K}, \tag{6.3}$$

where h_{max} is the length of the longest edge of K, ρ_K is the in-radius of K and α a normalization coefficient (such that $\mathcal{Q}_K = 1$ if K is the regular element, the quality function $\mathcal{Q}_K$ ranging from 1 to ∞). A candidate element K is thus discarded if its quality exceeds a user-specified threshold value η, i.e.,

$$\mathcal{Q}_K > \eta\,. \tag{6.4}$$

The η value is not necessarily fixed during the process, which leads to different strategies. It is possible to modify this threshold value (and degrade the shape

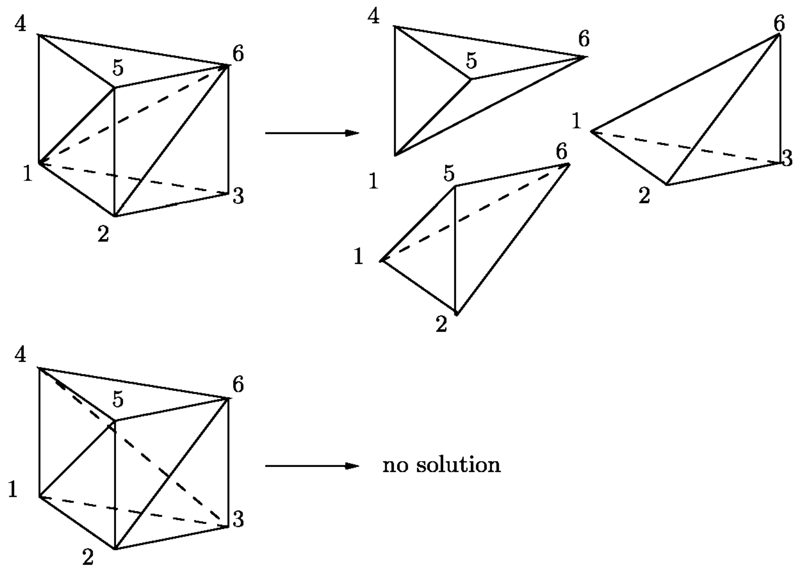


Figure 6.8: *Schönhardt polyhedron: valid and non-decomposable (without adding an internal point) constrained triangulation of a regular prism. If this pattern localy occurs, it is necessary to find as candidate a point a priori not found in the current strategy.*

quality) just to find a valid solution. It has been pointed out that a too strict a value leads to good quality elements but, on the other hand, may have some negative effect on the convergence, in some case.

An experimental analysis. As seen at the beginning of this discussion, the principle of any advancing-front method is relatively simple (and admitted) but several strategies can be used in an actual implementation, all of them being based on heuristics. Thus, it seems to be of interest, since no theoretical issues can be involved, to look more closely at the behavior of such or such a strategy. To this end, it is necessary to have several implementations available so as to be able to make some comparisons. This fact is clearly unlikely to be realistic. Nevertheless, after [Seveno-1997], some statistics can be provided that give some interesting ideas. The method used relies on the above scheme and the element validation is performed using the tools previously described. A large enough set of examples has been obtained and the nature of the retained connections for element creation has been reported. This appears to be a useful piece of information obtained *a posteriori*, to give information about which strategy could be adopted (and also on how to optimize its implementation).

Table 6.1 shows the frequency of the various connections between a front face and an entity when forming an element. In this table, *np*, *ne* and *nf* represent the numbers of vertices, edges and faces, respectively created for a given front face.

Face K connected with	np	ne	nf	frequency %
1 adjacent face	0	1	2	59.4
2 adjacent faces	0	0	1	23.7
optimal point	1	3	3	13.4
other cases	0	1	3	3.5

Table 6.1: Frequencies of the connections with adjacent mesh entities, given a front face K in three dimensions.

It appears that the front face is directly connected to an existing mesh vertex from an adjacent entity in about 80% of cases. Hence, point creation represents at most 20% of the whole element creation process. Moreover, the table shows that, on average, two faces are created when constructing one new element.

These statistics give us some indications. The first leads us to prefer, particularly in three dimensions, an existing vertex instead of a new point. Another issue related to convergence argues again for this choice that, in practice, reduces the number of delicate situations. The second indication, as proposed in [Golgolab-1989], leads to ordering the candidate points based on their probability of being connected with a given front face.

Convergence issues

The notion of a convergence for a mesh generation method concerns its capability to complete a mesh in all situations. Here, it reduces simply to entirely "filling up" the domain. As previously indicated, this point is not really relevant in two dimensions. Indeed, the following theorem holds (as proven in [George, Borouchaki-1997], Section 3.3).

Theorem 6.1 *In two dimensions, any arbitrary domain defined by a polygonal non-crossing contour can be triangulated without adding an (internal) point.*

This interesting result has a direct consequence on the convergence of the iterative advancing-front scheme.

Corollary 6.1 *In two dimensions, an iterative advancing-front mesh generation technique always converges in a finite number of steps.*

Actually, at each time, the iterations of the method can be stopped and the remaining unmeshed region(s) can be filled up.

In three dimensions, there is no similar issue. Indeed, irrespective of the numerical problems and provided an adequate strategy is used, the existence of a local configuration like that depicted in Figure 6.8 indicates that there is no guarantee of convergence.

Remark 6.9 *It is worth noticing that this negative issue is still present in a Delaunay type method where it appears in a different way. With an advancing-front method, the point is to complete the mesh, while with a Delaunay approach, the* *delicate point is related to the enforcement of a given boundary entity (edge or face).*

Nonetheless, there are numerous examples (in three dimensions) of advancing-front based software which converge in most cases. As indicated, the strategies used try to minimize the number of delicate situations. However, some authors propose, when faced with a bottle-necked situation, removing some previously created elements in some neighborhood so as to suppress the locked configuration while taking care not to produce the same pathology again or to provoke a cyclic process of creation-removal.

Point insertion and front updating

Given a front entity f, once a point has been identified (among P_{opt}, the candidate points, the trying points, etc.), and having successfully passed the different tests, it can be inserted in the current mesh. This stage consists of creating *one element* based on f and this point.

Remark 6.10 *As an optimization, specific cases allow multiple elements to be created during the insertion of a single optimal point (cf. Figure 6.9, in two dimensions).*

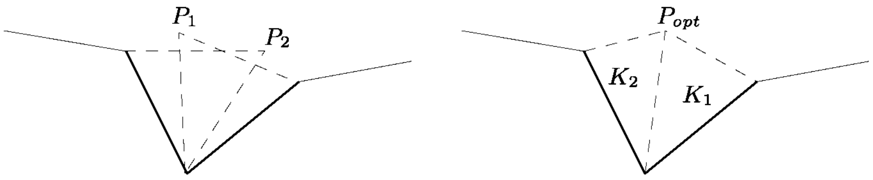


Figure 6.9: *Creation and insertion of a single optimal point P_{opt} resulting in the creation of two elements K_1 and K_2 in two dimensions (right-hand side).*

Updating the mesh simply involves adding this (these) element(s) into the list of elements already constructed.

The last stage of the iterative procedure corresponds to the front update. In practice, this procedure:

- removes the (current) front entity from the front which is now a member of the element constructed, if a new point has been introduced or
- adds to the current front the edges (faces) of the constructed element not shared by any other element.

We meet here the fact that the data structure devoted to front storage must be such that adding or suppressing any entity is made easy and fast.

Optimization

Once all the points have been inserted and once the corresponding elements have been constructed, we have a mesh that could be optimized to some extent. The purpose is then to optimize a quality criterion related to the future application (a finite element style computation for instance).

As previously mentioned, a well-suited quality measure for an element K in the mesh is:

$$\mathcal{Q}_K = \alpha \frac{h_{max}}{\rho_K}, \tag{6.5}$$

which, for the whole mesh, leads to a global value like:

$$\mathcal{Q}_{\mathcal{T}} = \max_{K \in \mathcal{T}} \mathcal{Q}_K. \tag{6.6}$$

The aim is then to minimize these values (including the global value). In fact, it could be observed that the method retained when choosing the points and thus when constructing (validating) the mesh elements have already considered a quality criterion. Nevertheless, for some convergence reasons, this criterion may have been violated in some cases. Also, the strategy generally leads to meshes with a majority of good quality elements (according to the threshold value η). Hence, in principle, the optimization stage is mostly of interest for a reduced number of elements.

Optimization procedures. The current mesh is optimized by means of local modification tools. Two local techniques can be used (Chapter 18). In the first, the point positions are maintained and their connections are affected. In the other, the connections are unchanged and the node locations are improved.

By using one or other of these techniques, we can use local optimization operators that make it possible to suppress some edges (by merging their endpoints), to swap (in two dimensions) an edge or to swap some edges or faces (this is the generalized swapping procedure in three dimensions). To end, notice that this optimization phase is common to all methods resulting in simplicial meshes (see Chapters 5 to 7).

Practical issues

As is now clear, any advancing-front method includes a large number of heuristics.

About basic algorithms. A precise examination of the general scheme indicates that there are three points that must be carefully handled. They are concerned with the front management (initialization and updating), the visit of a given neighborhood of the front entity under examination (to collect the neighboring entities and find the corresponding sizing information) and the validation checks, essentially based on intersection checks. They need to find the neighboring entities and, from a numerical point of view, to minimize the necessary CPU cost while guaranteeing valid results.

- Front management and selection of a front entity.

The front is a list of edges (faces) that is updated at each step by inserting or removing some edges (faces). Its management is then made easier by using a data structure well-suited to such operations (structures like a table or a dynamic linked list, see Chapter 2).

Depending on the strategy used, choosing a front entity is related to a metric type criterion (thus a sort) or a topological style criterion (neighborhood). In the first case, the criterion (length, surface area, etc.) must be considered when dealing with the front which is easy using a heap or a tree type structure.

In the second case, an easy access to the neighbors (edges or faces) of a given front entity is rather useful and we return to the notion of a neighborhood space that allows for this. Notice, in three dimensions, that the initial front is a triangular mesh whose vicinity relationships (in terms of edge adjacencies) are easy to obtain (Chapter 2).

- Neighborhood searching.

Already mentioned above, the search for the entities (in the mesh and not only in the front) in some neighborhood of a given entity f is necessary in order to find the points and validate the elements resulting from the connection of f with these points. As a consequence, data structures like a tree (quadtree/octree) or a uniform grid (in view of *bucket sorting*) allow easy access to the entities close (in terms of distance) to the given entity (the edges (resp. faces) in the mesh are encoded in this structure to make this searching task easy).

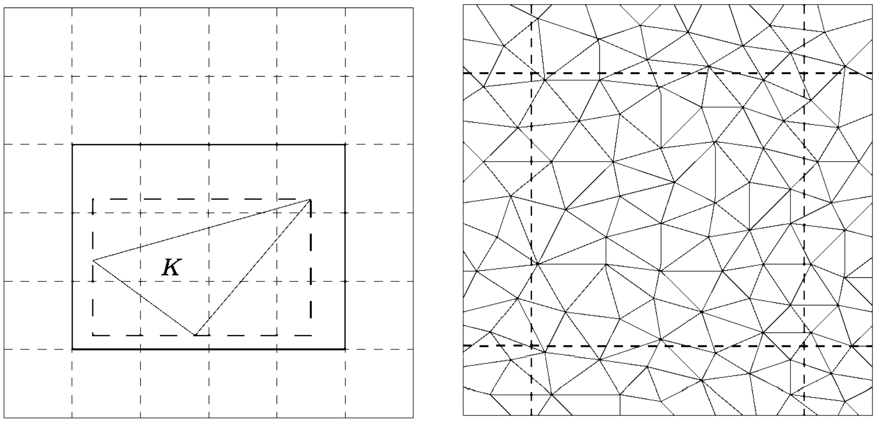


Figure 6.10: *Illustration of how to store and to retrieve a face in a grid. Range searching. Left-hand side: face K is coded in each cell it intersects. Right-hand side: a common problem with a regular grid, the size of the grid cell is not compatible with the size of faces K_i.*

- Remarks about intersection checks.

The neighborhood space (or any equivalent structure) is the key support for fast access to the entities close to the region under analysis. The use of filters then makes it possible to quickly reject the entities that are clearly not relevant (in terms of intersection). Indeed, it is obvious that a segment $[AB]$ such that $x_A < x_B$ does not intersect the segment $[CD]$ such that $x_C < x_D$ if, for instance, $x_C > x_B$. Thus, the intersection checks related to some adequate enclosing boxes show whether the corresponding entities may intersect or not. In the first case, an exact intersection check is made to give the decision, otherwise a rapid reject leads to a reduction in the cost. Intersection checks and tests are concerned with the following pairs:

- (box - box),
- (edge - edge),
- (edge - face),
- (point - element).

Remark 6.11 *A case where there is not really an intersection but where the two entities in comparison are rather close to each other one must, in general, be considered as an intersection case. In fact, constructing an element in such a configuration will lead, for instance, to defining some edges very close to this element and then a rather small yet unmeshed space will be formed, which will furthermore be rather difficult to fill up.*

Convergence issues. In two dimensions, a theoretical convergence result holds (Theorem (6.1)). From a practical point of view, the only trouble that is expected is when two fronts meet and the edge sizes are rather different. This potential drawback can be avoided if the sizes are globally considered, which means that such a situation is avoided.

In three dimensions, the convergence results from a strategy which tends to minimize the number of impossible cases and, in the case of such a pattern, a procedure able to remove some elements. This consists of going backwards so as to modify the local context and, in this way, remove the bottleneck.

Memory resources and data structures. Necessary memory resources correspond to the data structure used to store the useful values (vertices, mesh elements, etc.) and to the implementation of the basic algorithms (front management, vicinity, etc.).

The main internal data structures must be suitable to contain the following information[6]:

[6]Their internal organization depends on the programming language used.

- point coordinates,
- point sizes,
- the vertices of the front entities,
- element vertices,
- adjacent elements to a given element (in terms of edge or face adjacencies) and
- some other resources, *a priori* with a small memory requirements as compared with the previous ones (for instance, to encode the neighborhood space).

6.2 Governed advancing-front method

Constructing meshes that satisfy some specific properties (variable mesh density from one region to another, i.e., variable element sizes) is required in numerous applications. The construction scheme then leads to using a governed mesh generation method where the desired control is specified, as described below.

The specification of the desired element-size distribution function is indeed the desired information. Notice that this data, common to most mesh generation algorithms (see, for instance, Chapters 5 and 7), can be rather delicate to provide. The earliest approaches used a grid or a background mesh whose elements encode the desired sizing values. This data is then known everywhere in the domain. In fact, the purpose of this grid (mesh) was to provide a covering-up of the domain with which a continuous element-size distribution function is associated. This is exactly how a control space can be defined (Chapter 1). In terms of the spatial aspect, the control structure may be defined in various ways: using a regular grid, a quad- and octree decomposition or a background mesh (usual mesh or a mesh with no internal point, for instance, as will be seen in Chapter 7). In the coming sections, we briefly recall what a control space is.

Control space

Let us consider a background mesh of the domain as a control space[7]. The element-size function is known at the vertices of this background mesh. From the discrete size specification map, a continuous size function can be obtained using an interpolation scheme.

Direct use of a background mesh. Let P be an arbitrary point in the domain, the size $h(P)$ at P is computed as the P_1-interpolate of the sizes $h(P_i)$, $i = 1,..d+1$ (d being the space dimension) at the vertices P_i of the mesh element containing P (Figure 6.11). Schematically, the sizes $h(P_i)$ are computed as follows:

[7]For instance, a background mesh can be obtained using a Delaunay-based mesh generation algorithm, after the insertion of the boundary points and the boundary recovery (Chapter 7).

- For all grid vertices P_i:
  - find the element enclosing the cell vertices P_i,
  - compute $h(P_i)$ as the P_1-interpolate of the vertices of the element.

Using a grid built on the background mesh. Again, let P be an arbitrary point in the domain (cf. Figure 6.11). We are given a uniform grid and we note P_j its vertices. Schematically, the size value $h(P)$ is found after two steps. At first:

- For a given point P_j, a grid corner:
  - find the element within which P_j falls,
  - compute $h(P_j)$ as the P^1-interpolate of the sizes $h(P_i)$ at the vertices P_i of this element,

which simply reduces to using the above algorithm (so as to equip the grid corners with a size value). Then, for a given point P:

- find the grid box within which P falls,
- compute $h(P)$ as the Q^1-interpolate of the sizes $h(P_j)$ at the corners P_j of this cell.

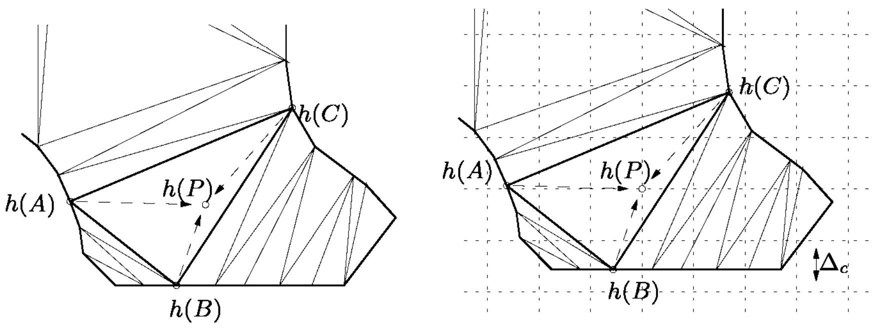


Figure 6.11: *Construction of the control space. Left: The size $h(P)$ at P is obtained by interpolation between the sizes $h(A)$, $h(B)$ and $h(C)$ at A, B and C, respectively. Right: a regular grid is combined with the background mesh, which can be discarded later.*

Remark 6.12 *In such an approach, the size of the grid cells must be in accordance with the size of the elements in the background mesh. If not, a filtering effect can arise leading to a lack (dilution) of information (cf. Chapter 5).*

Remark 6.13 *Notice also that the grid can be used as a neighborhood space only. For a given point P, the cell enclosing it is found. This cell acts as a* pointer *on an vertex in the background mesh, thus making it easy to quickly find an element enclosing P. The algorithm given at first can be then used to evaluate the desired size at P.*

Remark 6.14 *In the case where the background mesh is an "empty" mesh, these approaches result in a size distribution function strongly related to the boundary discretization. Indeed, the h's (size values) are "guessed" from this sole data.*

In place of a uniform grid, a decomposition like a PR-quadtree can be used that provides *a priori* a way to have as the spatial part of the control space a covering up that is better adapted to the geometry (Figure 6.12). The size of the tree cells is then related to the values of the size function and to local information (source points, etc.) if any. Therefore, the control structure is finer in high curvature regions and coarser elsewhere, specifically inside the domain (say, far away from its boundaries) [Shephard, Georges-1991], [Rassineux-1995]. Thus, the above filtering effect is avoided to some extent. Other methods for size prescriptions can be found in [Löhner-1996b] which combine a general aspect and finer information.

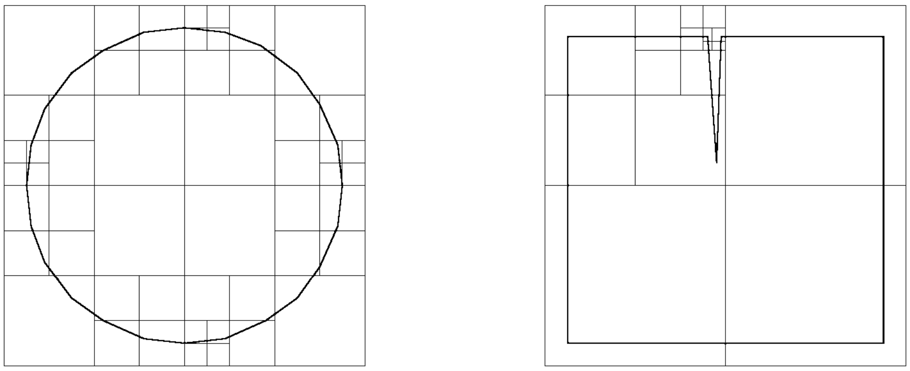


Figure 6.12: *Computing an implicit size map using a quadtree type decomposition. Left-hand side: the cell sizes increase as the distance from the boundary becomes greater. Right-hand side: the cell sizes is related to the domain geometry. The small cells make it possible to separate two entities belonging to two close boundary sides.*

Field point creation

Once the control space is ready, it is possible to develop a robust and flexible procedure for the field point creation designed in accordance with the information encoded in this space. Recall that for a given front entity, the goal is to create an optimal point such that the resulting element is well-shaped and its size conforms to the desired size. It can be seen (Chapter 1) that such conditions can be satisfied if the edges of the optimal element are unit edge (close to one, in terms of length) with respect to the size map. A mesh whose edges are unit edges is said to be a *unit mesh.*

Edge length. Let AB be a mesh edge and $h(t)$ be the size specification along AB such that $h(0) = h(A)$ and $h(1) = h(B)$, the normalized length of AB, denoted

l_{AB}, with respect to $h(t)$ has been defined as (Chapters 1 and 10)

$$l_{AB} = d_{AB} \int_0^1 \frac{1}{h(t)}\, dt\,, \tag{6.7}$$

where d_{AB} stands for the Euclidean distance between A and B. Edge AB is said to be conforming[8] to the size specification if:

$$\frac{1}{\sqrt{2}} \leq l_{AB} \leq \sqrt{2}\,, \tag{6.8}$$

and the goal is then naturally to create an optimal element (based on the front entity and the optimal candidate point) having all its edges conforming to the desired size.

Optimal point creation. The optimal point creation strategy is very similar to the classical case. In addition to the quality check, the algorithm attempts to create unit edge lengths for this element. In practice, an iterative algorithm is designed (both in two and three dimensions), which, given a front face ABC, its normal $\overrightarrow{n_{ABC}}$ and its centroid, reads:

1. Compute the optimal point location P_{opt} (via the standard procedure) as

   $$\overrightarrow{GP_{opt}} = \alpha h(G).\overrightarrow{n_{ABC}}\,,$$

   where α is a normalization coefficient leading to a regular tetrahedron. Set $it = 1$, the number of iterations.

2. Query the desired size $h(P_{opt})$ at P_{opt} via the control space (an interpolation scheme can be used to find $h(P)$ based on the size specifications at the vertices of the control space element enclosing P_{opt}).

3. Compute the normalized edge lengths $l_{AP_{opt}}$, $l_{BP_{opt}}$ and $l_{CP_{opt}}$ of the optimal element.

4. Compute the locations of the *pseudo-optimal* points P'_A, P'_C, P'_C along each edge AP_{opt}, BP_{opt} and CP_{opt} such that $l_{AP'_A} = 1$, $l_{BP'_B} = 1$, $l_{CP'_C} = 1$ (close to 1, Relation (6.8)).

5. Compute P'_{opt} the weighted barycenter of points P'_A, P'_B, P'_C.

6. Move P_{opt} towards P'_{opt}, using a relaxation coefficient ω (Figure 6.13),

   $$P'_{opt} = P_{opt} + \omega \overrightarrow{P_{opt}P'_{opt}}\,.$$

7. Set $P_{opt} = P'_{opt}$, fix $it = it + 1$ and, if $it < itmax$, return to Step 2.

[8]The coefficient $\sqrt{2}$ is related to the fact that an edge can be split if the lengths of the two sub-edges minimize the error distance to the unit length as compared with the initial length.

Remark 6.15 *The procedure converges rapidly (practically itmax < 5) toward an optimal point P_{opt}. The purpose of using an iterative approach is to take into account the potential gradation shocks ahead of the front face (for instance, when merging two fronts of widely differing sizes).*

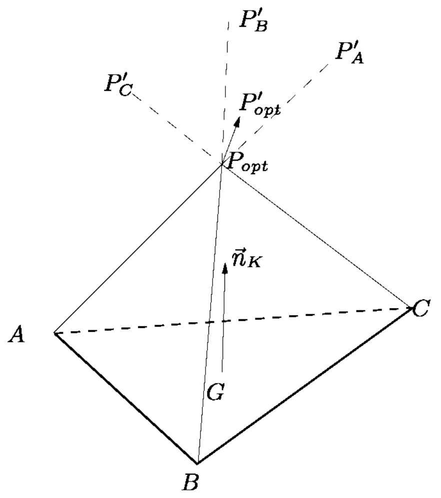


Figure 6.13: *Optimal point P_{opt} creation, given a front face ABC, the resulting tetrahedron K is considered optimal as it has conforming edge lengths.*

Remark 6.16 *The field point creation algorithm must also check the shape quality of the new element. In fact, the method tends to create conforming edges with respect to the control space. In three dimensions, however, this condition is not sufficient (e.g., the volume of a* sliver *is close to* 0*, even though all edge lengths are conforming; see Chapter 18).*

Optimization

As seen in the classical mesh generation problem and *a fortiori* in a governed case, it is usually useful to optimize the mesh resulting from the advancing-front governed algorithm. The goal is to produce the best regular element possible in accordance with the control space. Notice that the assumption of optimal field point generation has governed the entire procedure. Nevertheless, a two-stage optimization is applied to remove the worst elements in the mesh. This consists of improving the element quality (the shape as well as the size quality) using local topological mesh modifications (see Chapter 17, for more details). It seems to be efficient to optimize at first the size criterion and then to turn to the shape aspect (considering both criteria is probably elegant but tedious to do and is unlikely to be more efficient).

6.3 Application examples

In this section, several application examples of meshes in two and three dimensions, obtained using classical as well as governed advancing-front methods are proposed. Figures 6.14 and 6.15 illustrate a few meshes in two and three dimensions and Table 6.2 reports some characteristic values related to these examples.

-	np	ne	$\mathcal{Q}_{th}$	$\mathcal{Q}_M$	$1-2$	t (sec)
case 1	1,955	7,254	10.2	14.3	75	11
case 2	4,028	15,205	4.5	1.6	83	16
case 3	8,125	36,517	7.7	9.1	84	47
case 4	21,942	95,740	4.	5.3	91	139
case 5	33,373	170,330	3.2	7.7	93	212

Table 6.2: Statistics related to the selected meshes (the boundary discretization is the sole data used to generate the isotropic meshes).

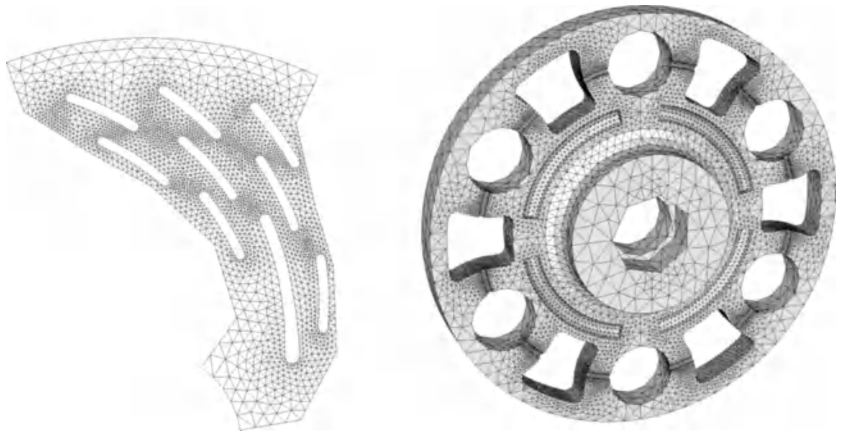


Figure 6.14: *Two- and three-dimensional advancing-front meshes (left: data courtesy of TELMA, right: data courtesy of ANSYS, Corp.).*

The statistics relative to the different meshes are provided in Table 6.2, where np and ne represent the number of mesh vertices and mesh elements respectively, $\mathcal{Q}_M$ denotes the shape quality of the worst-shaped element in the mesh, after the optimization stage. In two dimensions, the mesh quality should be close to one, regardless of the domain boundary discretization[9]. In three dimensions however, the mesh quality must be compared with $\mathcal{Q}_{th}$ which represents the quality of the best element that can be theoretically created based on the worst face of the boundary mesh. The row $1-2$ represents the percentage of elements having a quality ranging between 1 and 2 (according to Relation (6.3)) and t is the CPU

[9]Provided the optimal point belongs to the domain!

time (DEC ALPHA 2100/500 workstation) required to complete the final mesh (including the i/o).

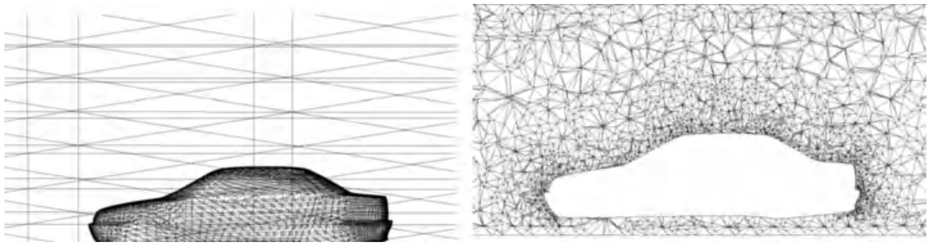


Figure 6.15: *Three-dimensional mesh of a car used for computational fluid dynamics (CFD) computations (data courtesy PSA). Left: computational domain (boundary mesh, local enlargement). Right: mesh in the vicinity of the car (section).*

In addition, Table 6.3 is given so as to appreciate the efficiency and the scalability of the advancing-front algorithm, where v represents the number of elements created per minute.

-	np	ne	t (sec., DEC 2100/500)	v
case 1	20,674	107,085	192	33,464
case 2	36,856	194,663	422	27.677
case 3	72,861	421,444	807	31,334
case 4	220,177	1,234,457	1,682	44,035
case 5	327,092	1,832,038	2,153	51,055

Table 6.3: Efficiency and scalability of the advancing-front meshing algorithm.

The results show that the speed-up ranges from $25,000$ to around $50,000$ elements per minute. This difference can be explained by two factors:

- the size of the underlying data structure used as neighborhood space (here a regular grid has been used) which may be inappropriate with regard to the element sizes (for instance, in case 2), or
- the small part devoted to intersection checking in some examples (for instance, when the computational domain is geometrically simple as in case 5 or for domains with high volume to surface area).

To conclude, notice that the theoretical complexity of any advancing-front method is not so obvious to investigate. We have already observed that the computational cost of the intersection checks is proportional to the number of entities close (in some sense) to a given entity and, for this reason, does not *a priori* depend on the mesh size. Also, we have pointed out that the geometrical aspect of the domain is not a parameter in terms of cost (since, after a few fronts, the remaining region is an arbitrarily shaped region, except for very specific cases where, in addition, a regular gradation is possible.

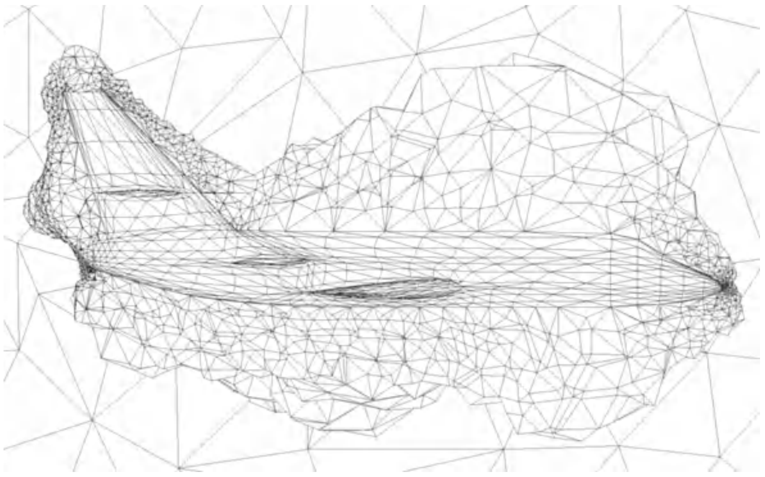


Figure 6.16: *Front evolution during the construction (cut through a three-dimensional mesh after a few iterations).*

Nevertheless, in [Peraire *et al.* 1988] and [Bonet, Peraire-1991], for instance, a complexity in $\mathcal{O}(n \log(n))$ is reported where n is the number of elements in the mesh. In these references, the factor in $\log(n)$ results from the use of a tree structure (an alternated digital tree). However, if the generation of a single element could be made in a constant time, then the triangulation of the whole domain will be linear in time. [Krysl-1996] also showed that a linear complexity is obtained in two specific problems, namely front management (using a hashing technique) and the search for the entities neighboring a given entity (using a neighborhood space, Chapter 1).

6.4 Combined approaches

The main drawback of the (standard or governed) advancing-front approach mainly relates to its efficiency (especially when compared to a Delaunay method). The intersection checking routine is a relatively expensive procedure for ensuring the acceptability of a new element.

In addition, advancing-front approaches often have problems with robustness, mainly because of the lack of a correct definition of the region of interest (i.e., the neighboring elements of a mesh entity)

An approach combining the Delaunay criterion (Chapters 1 and 7) and the advancing-front technique is a solution that has proved especially useful to overcome some difficulties. Using this it is possible to construct several elements at a time and, moreover, to allow the proper merging of two fronts of different length scales, which represents a classical type of failure in the standard algorithm (especially in three dimensions).

Advancing-front Delaunay approach

The Delaunay based methods are regarded as faster than the advancing-front style methods mainly because the mesh is constructed point by point and each point allows for the creation of several elements. From a local point of view, the number of elements constructed from a point is a function of the local situations, running from a few elements to several tens of elements (see, in Chapter 7, the notion of a cavity). However, since the elements are constructed and removed during the algorithm, a more precise analysis is needed to make sure that the global efficiency is related to this point.

The idea suggested by [Mavriplis-1992] leads to defining an advancing-front strategy that automatically locates the new points and constructs as elements those which conform to the Delaunay criterion so as to achieve the efficiency of these methods[10]. The algorithm mainly consists of identifying the three following configurations:

- some neighborhood of P_{opt} is empty (no point is included inside this region) or not,
- this zone is empty but there are some circles (spheres) circumscribing some elements of the current mesh that enclose the selected candidate point,
- this zone is empty and such circles (spheres) do not exist.

Then, for each of these patterns, the collected information allows us to anticipate the nature of the local context.

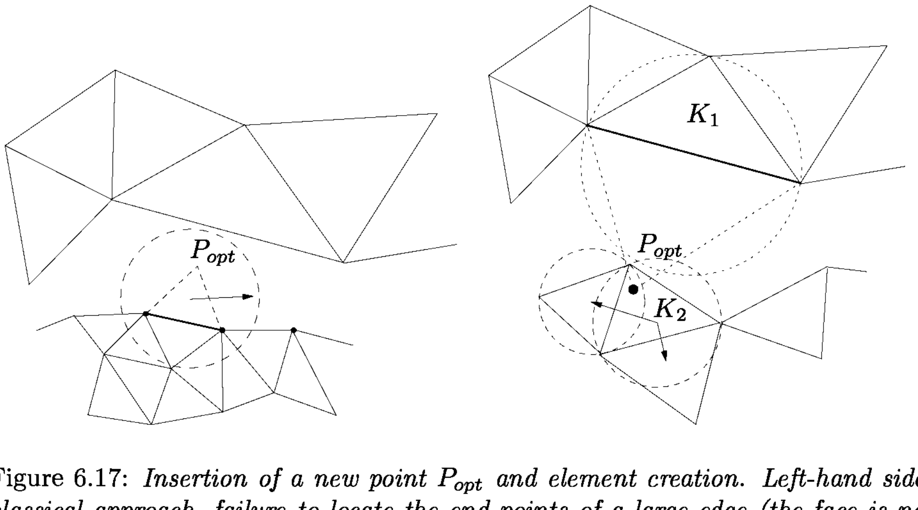


Figure 6.17: *Insertion of a new point P_{opt} and element creation. Left-hand side: classical approach, failure to locate the end points of a large edge (the face is not coded in all cells of the background space). Right-hand side: the circumcircles of K_1 and K_2 are intersected, neighbors of K_2 are searched and one is found to have its circumcenter intersected.*

[10]This point of view is also suggested in [Merriam-1991], [Muller *et al.* 1992] and [Rebay-1993].

Remark 6.17 *The inverse coupling of Delaunay advancing-front type is also a possible solution. The Delaunay method is used to connect the vertices while an advancing-front strategy is used to locate the field points; see [Frey* et al. *1998] and Chapter 7.*

Computational issues. The efficiency of the combined method is related to the fact that additional information is available, provided by the Delaunay criterion. In particular, this means that we automatically have certain knowledge of the neighborhood of the considered area, which, in the classical advancing-front approach, can only be obtained by a tremendous computational effort and through relatively complex data structures. The robustness of the method is also improved following this approach, as the Delaunay criterion makes it possible to anticipate and to avoid potential collisions between two fronts, to use a criterion to decide the deletion of conflicting elements and, finally, to handle and create several elements simultaneously. Moreover, if the internal points are well located, the Delaunay criterion offers a certain guarantee of the resulting element (shape) quality.

Advancing-front with preplaced interior points

As the advancing-front methods work from the boundary towards the interior of the domain, special care must be taken regarding the generation of mesh elements in the vicinity of the boundary. However, for domains with high volume to surface area, the internal point creation stage can be improved.

The approach suggested by [Rassineux-1997] consists of combining the advancing-front method, used to create internal points close to the boundary, with a quadtree-octree based spatial decomposition method to generate the internal elements (return to Chapter 5). More precisely, the general scheme of this technique consists of:

- building the spatial decomposition of the domain using a tree structure (quadtree in two dimensions and octree in three dimensions),
- marking the internal cells and the boundary cells,
- meshing the internal cells using predefined patterns (*templates*),
- meshing the boundary cells using an advancing-front approach.

An optimization step is then carried out to improve element (shape) quality, especially in the transition zones, between the regions where different meshing techniques have been used.

Notice that in this approach, the mesh gradation is governed by the tree balancing rule (element sizes are directly related to cell sizes). Moreover, the cell corners give the resulting mesh vertices, thus providing the final mesh with a certain rigidity.

Hybrid approaches

It is often desirable to generate stretched meshes for viscous flow computations, particularly in boundary layers and wake regions. This can be achieved using a hybrid technique which consists of creating a structured mesh in the boundary layers and wake regions combined with an unstructured mesh elsewhere.

The structured mesh consists of prisms (which can later be subdivided into tetrahedra, if necessary) obtained by offsetting the surface triangles along the normal directions at the vertices, thus defining a sort of homothetic surface to the initial one. The thickness of the layers is related to the vertex normals, the distance between the nodes being related to the Reynolds number and to the other parameters of the given problem. However, the stretching of the elements is that of the initial mesh.

The unstructured mesh is then created using a classical advancing-front technique. The internal points are calculated so as to generate well-shaped elements, which are as regular as possible. The mesh gradation is controlled so as to achieve a smooth transition of element sizes between the structured and unstructured meshes.

The tedious part of this approach concerns the proper definition of the boundary layer features and other regions where the elements have to be stretched, as well as the definition of the parameters (thickness, number of nodes, node stretching function). It is also necessary to ensure a smooth transition between the structured and unstructured meshes.

This kind of approach has been primarily studied in computational fluid dynamic applications, for instance, by [Muller *et al.* 1992], [Johnston, Sullivan-1993], [Pirzadeh-1994], [Hassan *et al.* 1996] and [Löhner *et al.* 1992].

6.5 Extensions

In this section, we mention some peculiar applications of the advancing-front method, that allow it to increase its potential field of application to non-conventional meshing problems. In particular, we mention anisotropic mesh generation, which is then extended to surface mesh generation. Finally, we comment briefly on the use of an advancing-front technique within an adaptive mesh generation scheme.

Anisotropic mesh generation

So far, the general scheme of the advancing-front method has been designed to handle isotropic meshes. However, most of the proposed approaches have been conceived for isotropic meshing purposes and are not able to handle high-aspect ratio element generation (such as those involved in Navier-Stokes computations, for example).

The general classical scheme of the advancing-front method can be modified to generate such elements. More precisely, the internal point creation stage is adapted to the creation of anisotropic elements.

Edge length in an anisotropic metric. We use here a more general formulation of the normalized edge length calculation (cf. Relation (6.7)). Let AB be an edge, defined using a parameterization such that $AB(t) = A + t\overrightarrow{AB}, \quad t \in [0,1]$ and let $\mathcal{M}(M(t))$ be the 2×2 matrix (in two dimensions) defined by:

$$\mathcal{M}(M(t)) = \begin{pmatrix} \frac{1}{h^2(t)} & 0 \\ 0 & \frac{1}{h^2(t)} \end{pmatrix}, \tag{6.9}$$

(resp. a 3×3 matrix in three dimensions), where $h(t)$ denotes the expected size at point $M(t)$. The length $l_{\mathcal{M}}(AB)$ of the segment AB in the metric corresponding to the matrix $\mathcal{M}$ is then defined by (see also Chapter 10):

$$l_{\mathcal{M}}(AB) = \int_0^1 \sqrt{{}^t\overrightarrow{AB}\,\mathcal{M}(A + t\,\overrightarrow{AB})\,\overrightarrow{AB}}\,dt. \tag{6.10}$$

Optimal point creation. The optimal point creation based on a given front item in order to create an anisotropic element is carried out by replacing the edge length computation in the classical scheme by an edge length calculation in the anisotropic metric. More precisely, we use the previous definition of the normalized edge length, replacing the matrix $\mathcal{M}(M(t))$ by the matrix (in two dimensions):

$${}^t\mathcal{R}(t) \begin{pmatrix} \frac{1}{h_1^2(t)} & 0 \\ 0 & \frac{1}{h_2^2(t)} \end{pmatrix} \mathcal{R}(t). \tag{6.11}$$

The matrix $\mathcal{R}(t)$ makes it possible to specify two privileged directions at point $M(t)$ and $h_1(t)$ (resp. $h_2(t)$) indicates the desired size in the first (resp. second) direction at this point. The adjusting strategy of the optimal point location is then identical to that of the governed isotropic case.

Optimizations. As for the governed isotropic case, an optimization stage is carried out to improve the size and the shape qualities of the resulting anisotropic mesh. This stage is based on local topological and geometrical modifications.

Surface mesh generation

The generation of finite element surface meshes is a topic that has received a lot of attention over the last few years. The reason is manifold. At first, surface meshes are important because of their effect on the accuracy of the numerical solutions (partly related to the boundary conditions) and the convergence of the computational scheme in numerical simulations based on finite (or boundary) element methods. On the other hand, three-dimensional meshing techniques often rely on surface (boundary) meshes. This is especially the case for quadtree-octree (Chapter 5), advancing-front (see above) and Delaunay-type methods (Chapter 7). Two approaches, direct and indirect, can be envisaged for surface meshing.

Direct surface meshing. This approach consists of applying a governed meshing technique (here the advancing-front) directly to the body of the surface, without using any kind of mapping related to any arbitrary parameterization. The mesh element sizes and shapes can be controlled by monitoring the surface variations. The approach proceeds by first discretizing the curves representing the surface boundary, then it triangulates the surface. The reader is referred to [Nakahashi, Sharov-1995] and [Chan, Anatasiou-1997], among others, for more details.

The general scheme of surface meshing using a direct approach is based on the same steps as the classical advancing-front technique. The main difference lies in the iterative algorithm used to find an optimal point location given a front edge.

More precisely, given an edge AB, the optimal point P is computed to construct a triangle determined by an angle α between the stretching direction and the tangent at the midpoint M of AB (or using an average tangent plane and setting α to zero). The point P is then projected onto the surface, its location being obtained through a query to a geometric modeler [Steger, Sorenson-1980]. Candidate points are identified as those which lie in the circle of center at P and radius $\kappa \times \delta$, where κ is a positive coefficient (e.g. 0.7 according to [Nakahashi, Sharov-1995]). The size of the triangle is locally adapted to the surface curvature (see Chapter 15). An optimization stage is required to globally improve the element shapes (cf. Chapter 19).

Notice that the edge lengths are calculated based on straight segments; however, points are further moved onto the true surface and the computed length can thus be quite different from the true edge length. It is therefore quite a tedious approach to implement, especially when the surface contains large curvature variations.

Parametric surface meshing. The indirect approaches essentially concern parametric surface meshing. The generation of a mesh for such a surface is obtained through a parametric space, which in fact is equivalent to a purely two-dimensional problem (although it uses surface related information). Let Ω be a domain of $\mathbb{R}^2$ and σ a sufficiently smooth function; then the surface Σ defined by the application $\sigma : \Omega \longrightarrow \mathbb{R}^3$, $(u,v) \longmapsto \sigma(u,v)$ can be meshed using a two-dimensional meshing technique in Ω, then mapping this mesh via σ onto $\mathbb{R}^3$. To this end, it is necessary to use a mesh generation technique with anisotropic features, the metric map being based on the intrinsic properties of the surface. The general scheme of this approach is based on three successive steps: the parameterization of curves and surfaces, the unstructured mesh generation in Ω and, finally, the mapping of this mesh from the parametric space to the real space so as to obtain the final mesh. An optimization stage can complete the mesh generation process.

Several authors have used an advancing-front for parametric surface meshing (see, for instance, [Peraire *et al.* 1987], [Samareh-Abolhassani, Stewart-1994], and also [Rypl, Krysl-1994] [Möller, Hansbo-1995], [Löhner-1996b] and [Marcum-1996]). Notice that the intersection calculations can be directly performed in the parametric space [Frykestig-1994].

Mesh adaptation

The basic idea of mesh adaptation is to improve the accuracy of the numerical solutions as well as to reduce the computational cost of the numerical operations. It requires a quasi-optimal node distribution at each iteration. The mesh adaptation problem will be discussed in greater detail in Chapter 21.

The mesh adaptation is based on a governed mesh generation technique, the size map is supplied by an *a posteriori* error estimate. The governed advancing-front strategy can be applied, almost without modification, to the creation of adapted meshes[11]

★
★ ★

Advancing-front type methods are efficient, robust and polyvalent. They allow classical or governed meshes (i.e., conforming a prescribed size map) to be constructed. They are also of interest in the context of adaptation problems (Chapter 21) and, in addition, allow surface meshes to be constructed (taking advantage of the anisotropic features).

[11]The adaptation is based on the regeneration of the whole mesh and not on a mesh optimization.

Chapter 7

Delaunay-based Mesh Generation Methods

Delaunay triangulation and the construction methods resulting in this triangulation have been extensive fields of research for a very long time. In particular, these topics are one of the major concerns in computational geometry (CG for short). It is therefore not really surprising to find a great deal of literature about Delaunay triangulation, starting with the pioneering paper by Delaunay himself, [Delaunay-1934]. Relevant references include [Shamos, Preparata-1985], [Joe-1991], [Fortune-1992], [Rajan-1994], [Boissonnat, Yvinec-1995] together with [Ruppert-1995] among various others. Delaunay triangulation problems are of interest for a number of reasons. Firstly, numerous theoretical issues can be investigated. Then, a wide range of applications in various disciplines exists including many engineering problems where theoretical results are used or revisited so as to obtain concrete algorithms.

Delaunay triangulation problems are of great interest as they can serve to support efficient and flexible mesh generation methods. In this respect, people concerned with engineering applications have investigated Delaunay-based mesh generation methods. The main references for this topic include [Lawson-1977], [Hermeline-1980], [Watson-1981], [Bowyer-1981] in the early 1980s and many others in the next decade such as [Weatherill-1985], [Mavriplis-1990], together with [George, Borouchaki-1997].

★
★ ★

This chapter includes six parts. The first recalls some theoretical issues regarding Delaunay triangulation. The second discusses the notion of a constrained triangulation. We then show how to develop a Delaunay-type mesh generation method. The fourth part briefly introduces several variants. Finally, extensions are proposed. We explain how to complete a mesh conforming to a pre-specified size map and how to generate anisotropic meshes (used, in particular, when dealing with parametric surface; see Chapter 15). Comments are added about weighted and anisotropic diagrams and triangulations together with potential applications.

7.1 Voronoï diagram and Delaunay triangulation

The Delaunay triangulation can be introduced in various ways (depending on the context of application). A convenient way is to use the dual of this triangulation, the Voronoï diagram.

The Voronoï diagram

Let $\mathcal{S}$ be a finite set of points $(P_i)_{i=1,\dots,n}$ in d dimensions. The Voronoï diagram for $\mathcal{S}$ is the set of cells, V_i, defined as:

$$V_i = \{P \quad \text{such that} \quad d(P, P_i) \le d(P, P_j), \quad \forall j \ne i\} \tag{7.1}$$

where $d(.,.)$ denotes the usual Euclidean distance between two points. A cell V_i is then the set of the points closer to P_i than any other point in $\mathcal{S}$. The V_i's are closed (bounded or not) convex polygons (polyhedra in three dimensions, d-polytopes in d dimensions); these non-overlapping cells tile the space, and constitute the so-called Voronoï diagram associated with the set of points $\mathcal{S}$ in $\mathbb{R}^d$.

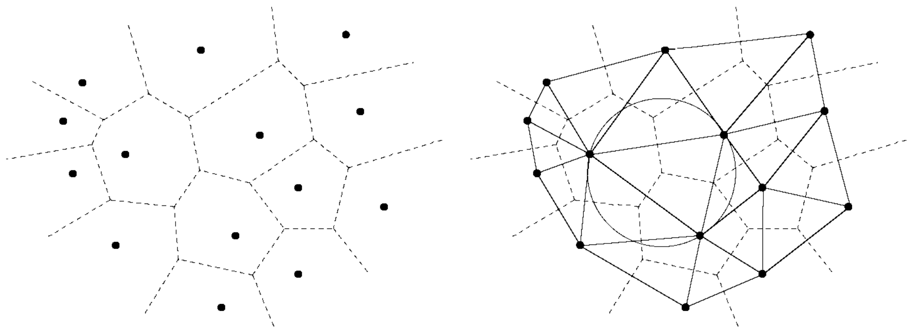


Figure 7.1: *Left-hand side: Voronoï diagram (in two dimensions). Right-hand side: corresponding Delaunay triangulation.*

Delaunay triangulation and Voronoï diagram

A triangulation problem typically concerns the construction of a triangulation of the convex hull of the P_i's such that the P_i's are element vertices. The construction of the Delaunay triangulation of this convex hull can be achieved by considering that this triangulation is the dual of the Voronoï diagram associated with $\mathcal{S}$.

Based on Definition (7.1), each cell V_i of the Voronoï diagram is a non-empty set and is associated with one point in $\mathcal{S}$. From these V_i's, the dual can be constructed, which is the desired Delaunay triangulation. For instance, in two dimensions, the cell sides are midway between the two points they separate, thus, they are segments that lie on the perpendicular bisectors of the edges of the triangulation. In other words, joining the vertices in S belonging to two adjacent cells results in the desired triangulation. The latter is unique and consists of simplices (triangles or tetrahedra according to d) provided the points in S are locally in general position (cf. Chapter 1 for this notion). Otherwise, elements other than simplices can be constructed which can be easily split into simplices (thus resulting in several solutions for a unique set of points).

Theoretical issues

This section recalls some classical theoretical issues about the Delaunay triangulation. In this respect, a fundamental theorem, the so-called *"lemme général de Delaunay"*, will be given. But first, we need to provide the definition of the well-know *empty sphere criterion.*

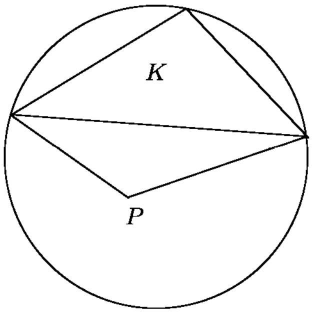


Figure 7.2: *B.N. Delaunay and his famous criterion. In this two-dimensional example, the "empty sphere" criterion is violated as the open disc of triangle K encloses point P. Note that this example is special as the point P is the vertex of a triangle adjacent to K which is opposite the common edge.*

The empty sphere criterion. In two dimensions, this definition refers to the open disk circumscribing a triangle while in three dimensions it concerns the open ball circumscribing a tetrahedron. This criterion is also referred to as the *Delaunay criterion.* Note that the criterion is referred to as the empty sphere criterion while would be better referred to as the empty ball criterion.

The "*lemme général de Delaunay*" can be enounced as follows[1]

[1]Published in French in 1934, see [Delaunay-1934], the original lemma is, *in extenso*, as follows: *"Soient T des tétraèdres tout à fait arbitraires qui partagent uniformément l'espace à n dimensions étant contigus par des faces entières à n-1 dimensions et tels qu'un domaine quelconque limité (c'est-à-dire à diamètre limité) ait des points communs seulement avec un nombre limité de ces tétraèdres, alors la condition nécessaire et suffisante pour qu'aucune sphère*

General lemma. *Let $\mathcal{T}$ be a given arbitrary triangulation of the convex hull of a set of points $\mathcal{S}$. If for each and every pair of adjacent simplices in $\mathcal{T}$, the empty sphere criterion holds, then this criterion holds globally and $\mathcal{T}$ is a Delaunay triangulation.*

Remark 7.1 *This lemma provides a rather simple characterization of the Delaunay triangulation. Note that a local property about the Delaunay criterion for two adjacent elements results in a global property for the entire triangulation.*

The proof of this lemma can be achieved in several ways and can be found in numerous references. For the sake of simplicity, we assume, in the following sections, that the given points are locally in general position. With this background, we discuss one method (among many others) that allows the construction of the Delaunay triangulation of the convex hull of a given set of points.

Incremental method. Given $\mathcal{T}_i$, the Delaunay triangulation of the convex hull of the first i points in $\mathcal{S}$, we consider P the $(i+1)^{th}$ point of this set.

The purpose of the incremental method is to construct $\mathcal{T}_{i+1}$, the Delaunay triangulation including P as an element vertex, from $\mathcal{T}_i$. To this end, we introduce a procedure, the so-called *Delaunay kernel* which can be simply written as:

$$\mathcal{T}_{i+1} = \mathcal{T}_i - \mathcal{C}_P + \mathcal{B}_P\,, \tag{7.2}$$

where $\mathcal{C}_P$ is the *cavity* and $\mathcal{B}_P$ is the *ball* associated with point P. Without loss of generality, we assume that P is included[2] in $\mathcal{T}_i$; then:

- cavity $\mathcal{C}_P$ is the set (the union) of elements in $\mathcal{T}_i$ whose open circumballs contain point P and
- ball $\mathcal{B}_P$ is the set of elements formed by joining P with the external faces of the above cavity.

From a practical point of view, the directly usable decisive result is that the cavity is a star-shaped set with respect to point P.

Theorem 7.1 *Let $\mathcal{T}_i$ be a Delaunay triangulation and let P be a point enclosed in $\mathcal{T}_i$. The above construction scheme completes $\mathcal{T}_{i+1}$, a Delaunay triangulation including P as an element vertex.*

There are several ways to prove this theorem. Here, we give two different proofs based on what is assumed and what must be proved (a proof, using the Voronoï duality, can be found in various references and in [George, Borouchaki-1997]).

circonscrite à un tel tétraèdre ne contienne dans son intérieur aucun sommet d'aucun de ces tétraèdres est que cela ait lieu pour chaque paire de deux de ces tétraèdres contigus par une face à n-1 dimensions, c'est-à-dire que dans chaque telle paire le sommet d'un de ces tétraèdres ne soit intérieur à la sphère circonscrite à l'autre, et réciproquement."

[2]In fact, three situations are possible including this case. The other cases are when P is outside all elements although it belongs to a circumscribing ball and the case where P is outside all elements and balls. In such cases, the definition of the cavity is slightly different but the same construction scheme holds.

Proof 7.1.a. shows that $\mathcal{T}_{i+1}$ is a Delaunay triangulation and establishes that $\mathcal{B}_P$ conforms to the previous definition. Proof 7.1.b. shows that Relation (7.2) where $\mathcal{B}_P$ is defined as above results in a Delaunay triangulation since $\mathcal{T}_i$ is Delaunay.

Proof (7.1.a.) This proof is completed in two parts. First, $\mathcal{T}_{i+1}$, a Delaunay triangulation, exists as the dual of the corresponding Voronoï diagram. Moreover, for the same reason, $\mathcal{T}_{i+1}$ is unique. Thus, the only thing we have to show is that $\mathcal{T}_{i+1}$ is the triangulation as defined by Relation (7.2), meaning that $\mathcal{C}_P$ and $\mathcal{B}_P$ are exactly the same as the previously introduced sets.

As the elements that violate the Delaunay criterion are those of $\mathcal{C}_P$ (and only those), the remaining part of $\mathcal{T}_i$ remains unchanged and this part becomes a part of triangulation $\mathcal{T}_{i+1}$. Then, we just have to establish that $\mathcal{B}_P$ is the appropriate construction to replace $\mathcal{C}_P$.

Let $\mathcal{R}_P$ be the re-triangulated cavity in $\mathcal{T}_{i+1}$. We will show that $\mathcal{R}_P$ is the above $\mathcal{B}_P$. The set $\mathcal{R}_P$ is a set of elements having P as a vertex. This is proved by contradiction. To this end, let us assume that there exists one element in $\mathcal{R}_P$ without P as one of its vertex, this element is then necessarily a member of $\mathcal{C}_P$ and thus violates the Delaunay criterion. Therefore, all elements in $\mathcal{R}_P$ have P as a vertex. As a consequence, they can be written as (P, f) where f is a face. Assume that f is not an external face of $\mathcal{C}_P$, then there exists an element in $\mathcal{R}_P$ that shares the face f with the element (P, f). In other words, this element can be written as (Q, f), where Q is different from P. This leads to a contradiction, hence f is necessarily an external face of $\mathcal{C}_P$. This enables us to conclude: the solution exists, is unique and $\mathcal{R}_P = \mathcal{B}_P$ is a valid way to replace the cavity. This particular solution is then the desired solution. □

We now turn to a different proof. Before giving it, however, we recall a fundamental lemma.

Lemma 7.1 *The Delaunay criterion for a pair of adjacent elements is symmetric.*

We consider a pair of adjacent simplices sharing a face and P (respectively Q) the vertex in these simplices opposite that face. Then,

$$Q \notin B_P \Leftrightarrow P \notin B_Q \tag{7.3}$$

where B_P (resp. B_Q) denotes the ball associated with the simplex having P (resp. Q) as a vertex.

Proof (7.1.b). In this discussion, we do not infer the duality between the Voronoï diagram and the Delaunay triangulation (so as to prove the existence of a solution). Consider $\mathcal{T}_i$, a Delaunay triangulation, and a point P contained in some elements but not a vertex element. We would like to show that the above construction completes $\mathcal{T}_{i+1}$, a Delaunay triangulation, with P as an element vertex.

At first, P is an element vertex of $\mathcal{T}_{i+1}$ according to the definition of $\mathcal{B}_P$. Then, we just have to establish that $\mathcal{T}_{i+1}$ is a valid Delaunay triangulation.

We first establish that the triangulation is valid (regarding the topology) as $\mathcal{C}_P$ is a connected set of elements. Assume that $\mathcal{C}_P$ consists of two connected components, one of these including an element, denoted by K_0, which separates this connected component from that enclosing P. Then define the segment joining the centroid of this element to P. This segment intersects one face of K_0, we consider then the element, say K_1, sharing this face with K_0. By definition, before introducing point P, the pair K_0 and K_1 complies with the Delaunay criterion (as members of $\mathcal{T}_i$). Thus, the vertex of K_1 opposite the common face is outside the circumball of K_0. As a consequence, the circumball of K_1 necessarily encloses P and thus K_1 is a member of $\mathcal{C}_P$. Repeating the same discussion, it is shown that all elements between the two connected components of the cavity are in fact members of this set. Hence the cavity is a connected set. The triangulation of $\mathcal{B}_P$ is then valid, in terms of its connections.

Moreover, since the external faces of $\mathcal{C}_P$ are visible by P, this triangulation is valid (regarding the geometry). The reason is obvious in two dimensions. We proceed by adjacency from triangle K_0, the triangle within which point P falls. Then, the three edges of K_0 are visible from P. Let K_1 a triangle sharing an edge f_1 with K_0. Then if K_1 violates the Delaunay criterion, it is in the cavity and the faces of K_1 other than f_1 are visible by P. This is due to the fact that P is inside the circumball of K_1 and that f_1 separates P and the vertex of K_1 opposite f_1. Thus, applying the same discussion makes possible the result for all the triangles in the cavity. However, the same argument does not extend in three dimensions as a face does not have the required separation property. Indeed, following the same construction from K_0, it is possible to meet as tetrahedron K_1 an element whose faces other than f_1, the face common with K_0, include one face, g, which is not visible by P. In this case, K_2, the element sharing the face g with K_1 is necessarily in the cavity, thus leading to the desired property.

Exercise 7.1 *Given the above situation, prove that K_2 is a member of the cavity of point P. Hint: examine the region within which P falls.*

To complete the proof, we have to show that $\mathcal{T}_{i+1}$ is a Delaunay triangulation. To this end we use the above general lemma. Then, the only thing which must be established is that the empty sphere criterion holds for all and every pair of adjacent elements. To account for all possible configurations of such pairs, the elements in $\mathcal{T}_{i+1}$ are classified into three categories:

i) those in $\mathcal{B}_P$,

ii) those having one element outside $\mathcal{B}_P$ and sharing an external face of $\mathcal{C}_P$,

iii) the remaining pairs.

Obviously, the elements falling in the third category conform to the Delaunay criterion. Those of category *ii*) are Delaunay too. Actually, while their circumballs do not enclose P, all the vertices opposite the face shared with the cavity are not inside the circumballs associated with the elements in $\mathcal{B}_P$. This is due to the symmetric property of the Delaunay criterion. Those of category *i*) are also

Delaunay. The proof is again obtained by contradiction. We consider an external face of the cavity, say f, and we consider the element of this cavity having this face (a former element of $\mathcal{T}_i$). Let K_{old} be this element, together with the new element constructed with this face, K_{new}. Then, assume that the circumball of K_{new} includes a vertex, that is necessarily outside the circumball of K_{old} and is therefore outside the cavity. While the elements outside the cavity are Delaunay, this results in a contradiction. Thus, the entire proof is completed. □

Practical issues

In this section, we briefly consider some practical issues that can be derived from the above theoretical background.

First, the incremental method can be used to define a constructive triangulation method even in the case where the given points are not in general position (i.e., when four or more co-circular, or five or more co-spherical points are in the initial set with corresponding empty circumballs, which is not likely to be plausible for realistic engineering applications, apart from a case where all the points in the set are co-spherical).

Then, after replacing the problem of constructing a triangulation of the convex hull of the given set of points by that of triangulating a convex "box" enclosing all the initial points, we can compute a solution using the same incremental method. Indeed, the box defines a convex hull problem for set $\mathcal{S}$ enriched with the corners of this box. As a consequence, all vertices fall within this box.

In the previous discussion, we did not account for numerical problems that can arise such as those related to round-off errors. As the key to the method is the proper definition of the cavity, any wrong decision when determining whether an element is in this set may lead to an invalid cavity. In other words, due to round-off errors, the above construction may fail, thus resulting in an unsuitable triangulation. Hence, at the cavity construction step, a correction is applied to ensure the star-shapedness of the cavity; see [George, Hermeline-1992]. Basically, this means that we explicitly check the visibility property (which is equivalent to the star-shapedness property) and, in case of a failure, we modify the cavity accordingly.

Remark 7.2 *In the case where such a correction is applied, the resulting triangulation could be non-Delaunay.*

The correction step is typically due to the numerical problems necessarily encountered when encoding the triangulation algorithm (and is representative of the difficulty of encoding geometric algorithms). This merits the following comments.

A typical numerical problem. The cavity construction is done by adjacency given the base as initialization (note that other methods exist). However, this solution offers a guarantee about the connectivity of the set; indeed, it prevents the obtaining of a multi-connected set.

The question is to decide if a given element is a member of the cavity of the current point P. Let K be the visited element, let O_K be its circumcenter (i.e., the center of its circumcircle (circumsphere)) and let r_K be the corresponding circumradius. Theoretically speaking, it is merely necessary to consider the ratio $\alpha(P,K) = \frac{d(P,O_K)}{r_K}$, (called the Delaunay measure) and to check if

$$\alpha(P,K) < 1\,.$$

The relevant check leads to comparing $d(P,O_K)$ and r_K. As these two quantities are not precisely valued, this check may be inaccurate, specifically, if the region in which P falls is close to the boundary of the disk (the ball) C_K of K. This uncertainty may result in dramatic results and the Delaunay kernel, (Relation 7.2), may result in a non-valid triangulation. These ambiguous configurations can fall in two classes:

- the cavity is not empty, meaning that there exists at least one vertex of a previously created element inside the cavity. This default is usually due to a proximity problem,
- the cavity is not a connected set. In general, this denotes a cocyclic (co-spherical) configuration.

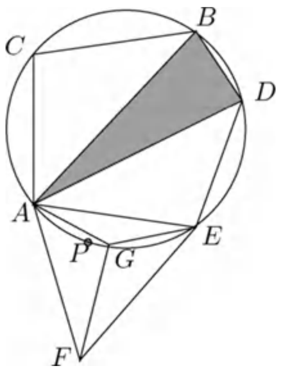


Figure 7.3: *The two ambiguous configurations.*

The first case leads to a vertex being missed (the resulting triangulation is still valid albeit wrong in this respect). The second case leads to a triangulation having overlapping regions. Figure 7.3 depicts these two situations.

- The first case of failure, due to imprecise computations, corresponds to the case where all the triangles in the figure are picked; point G is then strictly included in the cavity.
- The second case of failure, also caused by imprecise calculations, corresponds to the case where all the triangles of the figure except triangle (ADB) are selected, thus resulting in a non-connected cavity.

To overcome these problems, several solutions have been investigated. They consist of

- not introducing any point causing a problem,
- (slightly) perturbing all points leading to a problem,
- introducing a threshold value, ε, in the comparisons,
- performing exact computations,
- or, finally, suppressing the ambiguity using a different formulation of the algorithm.

The first solution requires that the current point is placed on a stack, such that its insertion will be done later when the local context is modified.

The second approach moves the point upon insertion and modifies the quantities involved in the construction, thereby expecting to remove the ambiguity.

The third solution, which introduces a threshold value ε in the comparison, has been investigated by numerous authors but does not lead to satisfactory results. An adequate value ε for a given case is not suitable for other cases.

The fourth approach implicitly assumes integer-type coordinates for the vertices and is not based on the Delaunay measure (meaning that the circumcenters and the circumradii are not computed or updated). Instead, it is related to the equivalent formulation (let us consider the two-dimensional case)

$$\Delta_K(x_P, y_P) < 0$$

where $\Delta_K(x, y)$ is the *inCircle* predicate of Chapter 2. This inequality includes quantities in the range of a length to the power $d + 2$ which involve additions (subtractions) and multiplications only. Consequently, a restriction is imposed on the vertex coordinate's range. In other words, the minimal distance between two points is limited. Indeed, if b is the number of bits of the mantissa of a double memory word, the largest value (denoted as l) that can be expressed in the above expression must satisfy the following relation

$$l \leq 2^{\frac{b}{d+2}} .$$

Assuming that the vertex coordinates start from the origin, this relation states that these coordinates must range from 0 to $l \leq 4096$ in two dimensions and from 0 to $l \leq 1024$ in three dimensions and, on the other hand, that the distance between two points is at least 1 with a typical computer[3] for which $b = 50$. These limits give both the maximal possible number of points according to the d directions as well as the minimal distance between two points. We have introduced, *de facto*, the *separating power* or the *resolution* of the method. The limit resulting from this discussion is obviously too restrictive and, consequently, while *a priori* elegant, this method is not adequate in general.

A determinant evaluation method can be found in [Avnaim *et al.* 1994], which overcomes this limit at the expense of increased complexity.

[3]Double precision words are employed, with *a priori* 51 significative bits. For safety reason, we limit ourselves to 50 bits. It should be noted that this limit depends on the technology; actually, 128 bit computers are widely used.

Another way to avoid this limit is to introduce an extended arithmetic and, more specifically, to use infinite precision[4] in the computations. See for instance, [Guibas *et al.* 1989], [Fortune, Van Wyk-1993] (among others) or [Perronnet-1988b] and [Peraire, Morgan-1997] for meshing applications.

The fifth method is the one we would like to recommend. We assume that the vertex coordinates are of integer type and we propose a new formulation for the Delaunay kernel resulting in a robust and exact algorithm in this context. The discussion of this method is the aim of the following paragraph. Briefly, the assumptions about the coordinates allow us to find the base exactly. This base enables us to define an approximated cavity which is furthermore corrected so as to ensure the expected properties (emptyness, connecteness and star-shapedness). This method will result in a valid triangulation which will not strictly be Delaunay.

Cavity correction A way to prevent a failure in the construction is to explicitly check what is needed in terms of properties. This motivated the following so-called cavity correction algorithm.

The problem centers on expressing the Delaunay kernel in such a way as to obtain an efficient constructive algorithm despite the round-off errors that may occur in the actual computation scheme. As already mentioned, the given coordinates are assumed to be of integer type ensuring exact surface (or volume) evaluations (obviously, to this end, we compute twice the surface area or six times the volume so as to avoid the division needed for an exact value). In this context, a two part algorithm is proposed. This algorithm includes the above method serving to initialize the cavity, the latter being wrong in some cases. The process is completed by a new algorithm, referred to as the *correction algorithm.* Let P be the current point to be inserted and let $\mathcal{T}_i$ be the current triangulation; the first stage of the method leads to

- using the Delaunay measure to construct the cavity associated with P, $\mathcal{C}_P$, by adjacency, given the base.

As this algorithm can result in a non-valid cavity, a correction step is needed as the second part of the process. This correction relies in removing some elements from $\mathcal{C}_P$ to meet the desired properties again. Thus, the correction algorithm can be described as follows

- if a vertex of $\mathcal{T}_i$ falls in the cavity, find one of the simplices[5] in $\mathcal{C}_P$, not in the base, having this point as vertex and remove this element from $\mathcal{C}_P$,

- if there is a $(d-1)$ boundary face of $\mathcal{C}_P$ not visible by P, pick and remove the simplex having this face from the cavity,

[4]This approach requires some comments. Indeed, if we consider the example of a surface of a triangle strictly positive when valued in infinite precision, it is not obvious that the same surface will be computed in the same way when used in a different software package.

[5]We can select as a simplex candidate the first element found that can be removed or select one of the possible simplices, enjoying a desired property.

- repeat this process as long as the number of elements in the cavity changes. One iteration results in starting the whole analysis of the elements remaining in the cavity again. This is done either from the base and proceeding by adjacency or this can be done by considering the last element not affected by the actual process.

Note that this correction algorithm converges. Indeed, in the worst case, the cavity is reduced to the base thus leading to the convergence. Also, using adjacency relationships in the process ensures that the cavity is a connected set; as the base is necessarily included in the cavity, the latter contains point P. Finally the visibility checks (surface or volume computations according to d) guarantee the star-shapedness property of the cavity.

In summary, the proposed algorithm is constructive and the computations are integer in nature (and thus are exact) or such that only surface (volume) evaluations, or equivalent computations, have been used. Thus, it is possible to obtain a computationally efficient and robust algorithm with a limit of application, as discussed above, partly extended. Indeed, the limit is now $l \leq 2^{\frac{b}{d}}$, leading to $l \leq 33554432$ in two dimensions and $l \leq 65536$ in three dimensions. Obviously, an order of magnitude has been obtained and the separation power of the method is increased. Hence, this method is usually well-suited. The l value gives the separation power of the method and indicates the maximal number of points in each direction (the minimum distance from point to point being 1).

Actually while assuming integer coordinates in the discussion, the same idea of using explicit validation works well with real coordinates.

7.2 Constrained triangulation

As pointed out in Chapter 1, a constrained triangulation problem concerns a triangulation problem of a set of points in the case where some constrained entities (edges or faces) are specified that must be present in the resulting triangulation.

In this section, we discuss three aspects related to a constrained triangulation. We show how to maintain such an entity (edge or face) when it exists at some step, then we turn to a method suitable for entity enforcement in two and three dimensions.

7.2.1 Maintaining a constrained entity

A triangulation procedure such as the incremental method (Relation (7.2)) can be constrained so as to preserve a specified edge (a face) which is introduced at some step of the incremental scheme and must be retained. To this end, the construction of the cavity is modified.

When visiting the elements by adjacency, we do not pass through the specified edges (faces) which have been created at some previous step. This means that two elements are not regarded as adjacent if they share a specified entity (edge or face) created in the triangulation at a previous step. Hence the cavity construction is modified in this way. This results in a triangulation where some edges (faces) are specified. Due to this constraint, this triangulation is no longer a Delaunay triangulation (it is referred to as a constrained Delaunay triangulation for which the Delaunay criterion is not required between a pair of elements separated by a constrained item).

Theorem 7.2 *Let $\mathcal{T}_i$ be an arbitrary triangulation and let P be a point enclosed in $\mathcal{T}_i$, then Relation (7.2) determines $\mathcal{T}_{i+1}$, a valid triangulation having P as element vertex.*

This theorem just means that Relation (7.2) (considered together with a correction algorithm in some cases; see the above discussion) still results in a valid triangulation even when the initial triangulation is an arbitrary triangulation and some constraints are present.

Proof. This proof is obvious since the cavity involved in the construction is a star-shaped region due to the way in which it is constructed (starting from the base and completed by adjacency while, at the same time, the required visibility property is explicitly achieved by a correction stage). Then, the definition of the ball is valid and the resulting triangulation is valid as well. □

The previous construction is not, in general, a solution to ensure the existence of a pre-specified set of edges (edges and faces in three dimensions) in a triangulation at the time the endpoints of these items have been inserted in the triangulation. Thus, other methods must be developed which work well in this case.

Remark 7.3 *In three dimensions, the above discussion holds for a constrained face but is not a solution for a constrained edge. We can remove an edge by turning around it by means of face adjacencies.*

7.2.2 Enforcing a constraint

Constraints in two dimensions

We consider a series of edges whose endpoints are in set $\mathcal{S}$ and we want to make sure that these items are edges of the triangulation at the time all the points in $\mathcal{S}$ have been inserted. In general, this property does not hold, as can be seen in Figure 7.4 where a simple example is depicted.

The problem we face is then to enforce[6] the missing edges. In two dimensions, a rather simple procedure can be used to get this result, the so-called *diagonal swapping* (see also Chapter 18). Given a pair of adjacent triangles sharing an edge, we consider the quadrilateral formed by these two triangles. If this polygon is convex then it is possible to swap its diagonal (the former common edge) so as to create the alternate diagonal. In this way, we have removed an edge (while a new one is created). A repeated use of this procedure enables us to delete all the edges

[6]This is an *a posteriori* approach to the constrained triangulation problem. Note that an *a priori* solution can be also envisaged, as will be discussed hereafter and in Chapter 9.

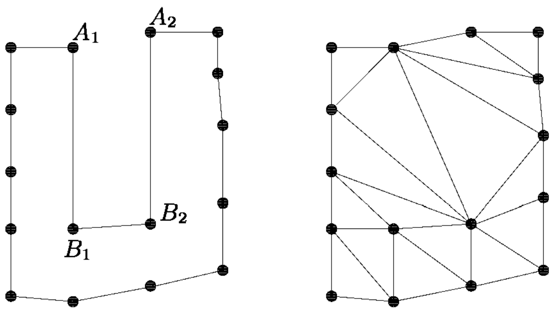


Figure 7.4: *In this simple two-dimensional example, we have displayed the triangles which are intersected or close to two missing edges (in particular, some triangles, part of the triangulation of the convex hull, are not shown). Actually, edges A_1B_1 and A_2B_2 are missing in the triangulation although their endpoints are element vertices.*

that intersect a specified segment (actually, an edge that must be constructed) and results in the desired solution.

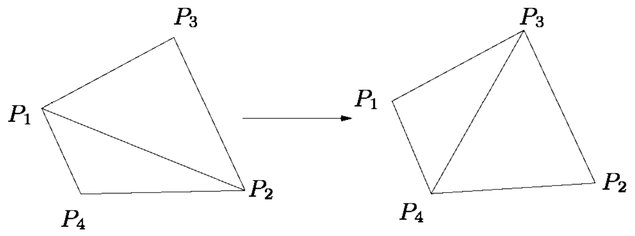


Figure 7.5: *Diagonal swapping. The quadrilateral formed by this pair of adjacent triangles is a convex region, so its diagonal can be swapped.*

Applied to each missing entity, this procedure computes the desired triangulation which, however, is obviously not a Delaunay triangulation.

Two theoretical issues can be invoked to justify the above method.

Theorem 7.3 *Given a set of segments, there exists a triangulation incorporating these segments as element edges.*

Theorem 7.4 *Given an arbitrary triangulation and a set of segments (whose endpoints are vertices of this triangulation), a triangulation where these entities are edges can be computed using only the diagonal swapping operator.*

In fact, given an arbitrary triangulation, it is always possible to obtain a specified triangulation having the same vertices, by means of diagonal swapping only.

Thus, a non-cyclic process of diagonal swapping is a solution to the above problem (note that the diagonal swapping is a reversible process). Nevertheless, the previous theorems hold in two dimensions only.

Constraints in three dimensions

The same problem is much more difficult in three dimensions. Actually, the constrained entities could be a series of edges and faces (assumed to be triangular) and, at the time all the endpoints of these entities have been inserted, some of these edges and faces might not be present in the triangulation.

The problem is split into two parts. At first we enforce[7] the missing edges and, once this has been completed, we enforce the missing faces (indeed, there exist geometrical configurations where the three edges of a face exist while the face itself is not formed, meaning that one or more edges of the current triangulation pass through the triangle whose edges are the three above edges).

Theoretical results can be put forward to prove that an edge can be enforced in a triangulation by means of generalized swapping and, if necessary, by creating some points, the *Steiner points*, to overcome the situations where no more swaps can be successfully done. This result can be seen as an extension of Theorem (7.4) to three dimensions. Regarding the constraint by a face, the situation is not so clear. In practice, heuristics must be used.

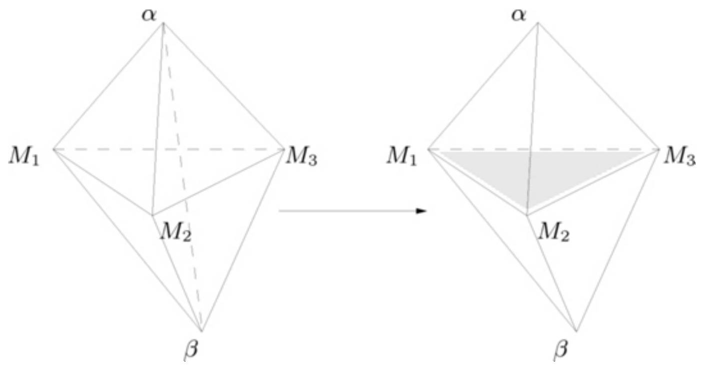


Figure 7.6: *The polyhedron consisting of the two tets $M_1M_2M_3\alpha$ and $\beta M_1M_2M_3$ (right) is convex and can be re-meshed using the three tets $M_1\alpha\beta M_2$, $M_2\alpha\beta M_3$ and $M_3\alpha\beta M_1$. Conversely, this three element configuration can be re-meshed by means of two elements. In the first transformation, a face has been removed while in the second an edge has been removed.*

Generalized swapping procedure. It is appealing to extend the two-dimensional diagonal swapping by considering the pattern formed by a pair of adjacent tetrahedra. This leads to a face swapping procedure whose converse application results in removing an edge. In fact, the latter operator is only a simple occurrence of a more general operator dealing with a general pattern, the so-called *shell* (Chapter 2). A shell is the set of tetrahedra sharing a given edge. Then, the

[7]See the above footnote about *a posteriori* or *a priori* solutions for the problem.

three-dimensional version of the swap operator can be seen as remeshing this polyhedron by suppressing the common edge (Figures 7.6, 7.7 and Chapter 18).

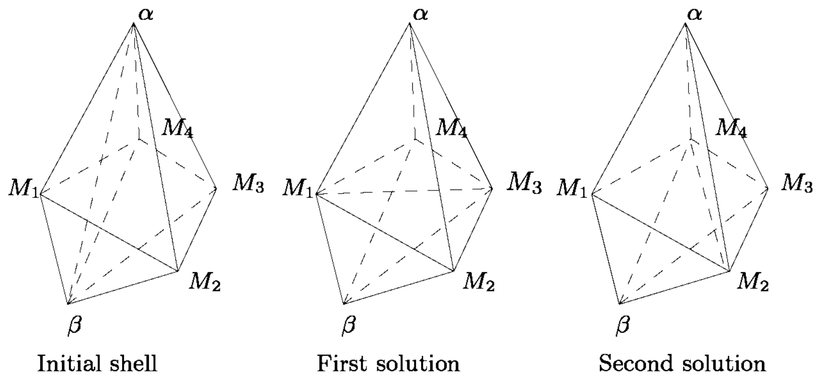


Figure 7.7: *The initial pattern is the shell associated with edge $\alpha\beta$. Two alternate remeshings of this polyhedra are possible if it is convex.*

Steiner points. A rather obvious example (Figure 7.8) shows that it is not always possible to triangulate a region. Nevertheless, adding a single point in the region depicted at the bottom of the figure leads to a solution. As a result, we could expect to meet such situations when considering a constraint in a triangulation and would like to use a similar method to obtain a valid solution. The question is then how to detect such a pathology and how many points are strictly needed to overcome the difficulty and where this (these) point(s) must be located. Actually, this leads to finding the *visibility kernel* of a given polyhedron.

Following the previous discussion, we propose a heuristic method to enforce a set of constraints. First, we deal with the problem of edge enforcement, then we turn to the face enforcement problem. The key idea is to locally modify the current triangulation by means of the generalized swapping operator, creating some Steiner points when the previous operator fails.

Edge enforcement. The elements in the current triangulation that are intersected by a missing edge are identified (such a set is called a *pipe*). Then we meet two situations. Either only faces of these elements are intersected by the missing edge or the latter intersects, at least, one edge of the current triangulation. The first case leads to applying the generalized swapping to every pair of adjacent tetrahedra (in the case where the thus formed region is convex) while the second situation leads to modifying the shell(s) of interest. Steiner points are created when no more swaps can be successfully completed.

Except for the numerical problems, the above idea makes it is possible to regenerate all the missing edges. It is now possible to consider the missing faces (if any, since most of them exist at the time their edges are present in the mesh).

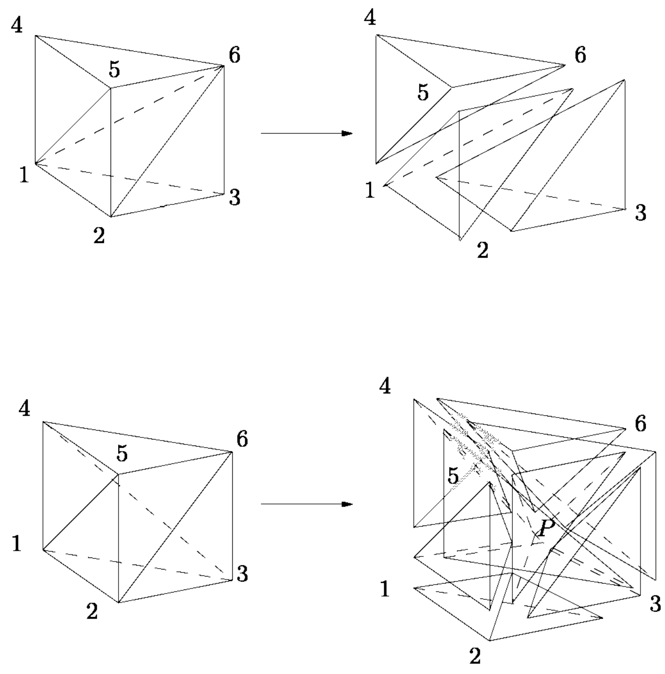


Figure 7.8: *The prism at top can be partitioned by simply using three tetrahedra. The prism at the bottom, the so-called Schönhardt polyhedron, cannot be split with three tetrahedra. This prism differs from the previous one in the way in which its quadrilateral faces are decomposed into triangular faces. To find a valid mesh, a point must be created in the visibility kernel of this polyhedron. This point is then joined with all the external faces, thus resulting in a suitable mesh.*

Face enforcement. A similar procedure is used. The set of elements corresponding to a missing face are exhibited. In this set, a series of edges exists which intersect the missing face. These edges are then swapped, using Steiner points in some cases, until a missing face is intersected by only one edge. Then an ultimate generalized swap results in the desired solution (assuming that the corresponding pattern is convex).

This heuristic, while not numerically proved as, theoretically speaking, we use arguments like "there exists a non-empty visibility kernel", has proved to work well in most concrete situations.

7.3 Classical Delaunay meshing

Delaunay triangulation algorithms serve as a basis for designing a meshing method, a so-called Delaunay type mesh generation method.

In practice, the problem we face is now rather different. Up to now, we have discussed a triangulation problem. This means a problem of triangulating the convex hull of a given set of points, but, for a typical meshing problem, the input is a closed polygonal curve (polyhedral surface) defining a region (or several such curves (surfaces) defining a multiply connected region). The problem is then to generate a set of vertices in this not necessarily convex domain and to make sure that the above (curve or surface) discretization is present in the resulting triangulation. This means that we meet a problem of constrained triangulation as a series of edges and faces must be present in the mesh.

Despite these differences, some of the previous material on Delaunay triangulation, possibly with some extensions, can be applied to meshing problems when the domain is supplied via a boundary discretization.

The intention that the mesh be suitable for applications takes several forms. For classical Delaunay meshing, as discussed in this section, it is necessary that the mesh elements should be well-shaped while their sizes should be adequate with regard to the sizing information available in this case (basically, the sizes of the boundary items serving as input data). However, applications typically require meshes in which the element sizes, and even their shapes, vary across the mesh according to a given specification. This point will be discussed in further sections.

A Delaunay type meshing method is generally one step of a mesh generation procedure including three successive steps:

Step 1: the mesh parameterization (boundary description, specification or construction of a function defining the element size distribution, etc.),

Step 2: the boundary discretization,

Step 3: the creation of the field vertices and elements, in other words, the Delaunay type method itself.

This general scheme is close to that found in other methods such as the advancing-front type method (Chapter 6) and is slightly different from that of a method based on an *quadtree* (Chapter 5) where the boundary mesh can be constructed during the domain mesh construction.

In a Delaunay type method (Step 3 of the above scheme) the resulting mesh is a mesh of the box enclosing the domain. Field points are created and inserted in the current mesh so as to form the elements by means of the Delaunay kernel.

General scheme

The previous material can be now used to develop a Delaunay type mesh generation method. Here is a scheme for such a method:

- Preparation step.
  - Data input: point coordinates, boundary entities and internal entities (if any).
  - Construction of a bounding box and meshing of this box by means of a few elements.

- Construction of the box mesh.
  - Insertion of the given points in the box mesh using the Delaunay kernel.
- Construction of the empty mesh (which is boundary conforming).
  - Search for the missing specified entities.
  - Enforcement of these items.
  - Definition of the connected components of the domain.
- Internal point creation and point insertion.
  - (1) Internal edges analysis, point creation along these edges.
  - Point insertion via the Delaunay kernel and return to (1) until edge saturation.
- Domain definition.
  - Removal of the elements exterior to the domain.
  - Classification of the elements with respect to the connected components.
- Optimization.

Note that the domain definition is only achieved at the end of the process. In this way, the convex mesh of the box is present throughout the process which facilitates the necessary searching operations.

In the following sections, we describe the different stages of this general scheme and we focus on the main difficulties expected.

Preliminary requirements. In contrast to advancing-front methods (Chapter 6), no specific assumption is made on the nature of the input data related to the boundary discretization. In particular, the orientation of the boundary items is not required and does not offer any specific interest (whereas it may increase the processing speed in quadtree-octree methods).

7.3.1 Simplified Delaunay type triangulation method

Without loss of generality, we define a convex "box" which is large enough to enclose the domain. In this way we again encounter a situation where the previously described incremental method can be used.

In fact, introducing a box, enables us to return to a convex hull problem where the set S consists of the vertices of the given boundary discretization and four (eight) additional points (the corners of the introduced box, a square in two dimensions, a cube in three dimensions).

Once the box has been triangulated by means of two triangles (five or six tetrahedra), we find a situation where all the points in S (apart from the box corners) are strictly included in this initial triangulation. Due to this simple property, which will be maintained throughout the meshing process, the construction

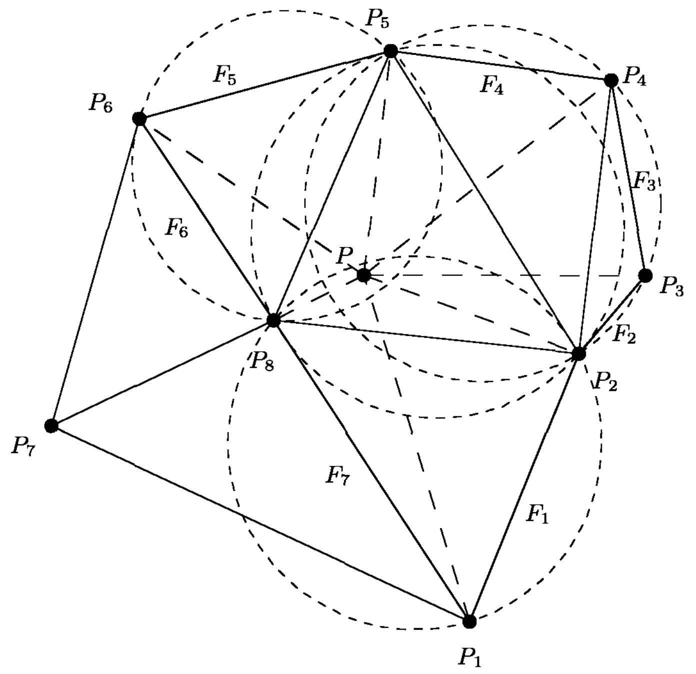


Figure 7.9: *Inserting point P (P included in the current mesh). In this two-dimensional example, only the triangles close to point P are displayed. The base is reduced to the triangle ($P_2P_5P_8$). The cavity is formed by the triangles in solid lines. The external faces of this cavity are denoted by $F_1, F_2, ..., F_7$. The ball consists of the elements in dotted lines. It is formed by joining P with the F_is.*

method reduces to the case where the points that must be inserted are always inside the current triangulation.

Thus, the construction method relies on properly defining the cavity associated with the point to be inserted, knowing that this point necessarily falls within a mesh element. This construction, for a given point P, is sequentially achieved as follows:

1. we search in the current mesh for the element within which point P falls. As a result we obtain a set of elements, the so-called *base* associated with P. This base can be reduced to one element, two elements when P is located on an edge (a face in three dimensions) or more when, in three dimensions, P falls on one edge.

2. starting with the elements in the base, we visit by adjacency the current mesh so as to determine the elements whose circumballs contain point P. This results (possibly after a correction stage) in the desired cavity.

Then, this cavity is replaced by the corresponding ball and the mesh with P as vertex is completed. This simple procedure is applied to all the points known at this stage (typically, the boundary points). At completion, we have created a mesh of the box enclosing the domain and *not* a mesh of this domain.

7.3.2 Boundary integrity and domain identification

Boundary integrity

The mesh resulting from the above method is a mesh of the box enclosing the domain. As previously seen, the boundary entities defining the domain, are not necessarily present in this box mesh. In other words, we face a constrained meshing problem. Typically, two approaches can be envisaged to solve this problem, one being an *a priori* approach and the other an *a posteriori* approach. In the first case, the boundary discretization is such that it naturally appears in the mesh, in the second case, some boundary entities are missing in the current mesh which need to be enforced.

Delaunay-conforming boundary mesh. Before constructing the box mesh, we analyze the boundary discretization to see whether it is Delaunay or not. At this time, this notion simply means that the boundary entities are automatically present in the mesh based on their endpoints. If the given discretization is not Delaunay, then we modify it (see Chapter 9) so as to meet this property. Hence, boundary integrity is no longer a problem.

Boundary enforcement. We are given a mesh where some edges (faces) are missing. In this approach, we return to the method discussed for the constrained triangulation, and, by means of local mesh modifications, we modify the current mesh in such a way as to ensure the existence of all boundary entities and to obtain the desired boundary integrity.

Identifying the elements in the domain

When the boundary entities of a given domain are present in the mesh, it is possible to identify the elements of the mesh which lie within this domain. Bear in mind that we have triangulated a "box" enclosing the domain and that we now need to discover this domain.

A rather simple algorithm, based on coloring (Chapter 2), can be used to discover the connected component(s) of the domain. In this way the internal elements can be determined where, furthermore, it will be possible to create the field points. The scheme of this algorithm is as follows:

1. Assign the value $v = -1$ to all elements of the box mesh (where the boundary entities now exist) and set $c = 0$, c being seen as a color.

2. Find an element having as a vertex one of the box corners and set it to the value $v = c$, put this element in a list.

3. Visit the three (four) elements adjacent by an edge (a face in three dimensions) to the elements in the list:
    - if the color of the visited element is not -1, the element has been already colored, thus return to 3;
    - if the face (edge) common with the visited element and the current element in the list is not a boundary entity, assign the value $v = c$ to this element, put it in the list and return to 3;
    - if the common face (edge) is a boundary member, return to 3.
4. Set $c = c + 1$, empty the list and if an element with $v = -1$ exists, put it in the list and return to 3. □

Variants of this algorithm can be used to complete the same task. Nevertheless, at completion of such a procedure the elements are classified according to the different connected components of the domain.

7.3.3 Field point creation

We now have a mesh for the domain where the element vertices are typically the boundary points, meaning that no (or few[8], in three dimensions) internal vertices exist in the mesh. Thus, to fulfill the numerical requirements (well-shaped elements and adequate sized elements), we have to create some field points in the domain.

Several methods can be envisaged to achieve this; among these, we focus here on one method using the edges of the current mesh as a spatial support for the field points.

Preliminary requirements. At first, a stepsize h is associated with all the boundary vertices (by means of the average of the lengths (surface areas) of the edges (faces) sharing a boundary vertex).

Edge analysis. The key idea is to consider the current mesh edges and to construct a set of points on them. The process is then repeated as long as the creation of a point on an edge is needed. In other words, as long as the edges are not *saturated*. This iterative process starts from the mesh obtained after the domain definition (see above), constructs a first series of points, inserts them into the current mesh and repeats the processing on the resulting mesh.

Then, the current mesh edges are examined and their lengths are compared with the stepsizes related to their endpoints. The goal of the method is to decide if one or several points must be created along the visited edge. If so, both the number of required points, n, and their location must be determined. The objective is twofold. We want to introduce suitably spaced points along the edges in order to saturate them and to obtain a smooth point distribution.

We demonstrate an arithmetic type of point distribution for an edge AB. If

[8]The necessary Steiner points.

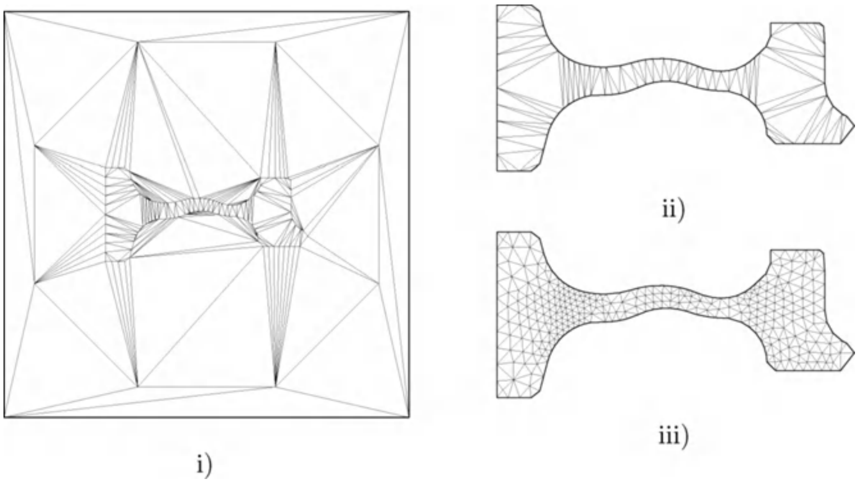


Figure 7.10: *i) Mesh of the box enclosing a domain for a mesh generation problem in two dimensions. The four corners used to define the box enclosing the domain can be seen together with some extra points defined between the box and the domain for efficiency reasons. ii) Corresponding empty mesh. This mesh is nothing other than a coarse discretization of the domain resulting from the previous coloring algorithm. Actually, this mesh displays the edges where a first wave of field points will be created. iii) Final mesh after internal point insertion and optimization.*

- $h(0) = h_A$ denotes the stepsize associated with $P_0 = A$, one of the endpoints,
- $h(n+1) = h_B$ is that related to $P_{n+1} = B$, the other endpoint,

we can define a sequence α_i (and thus the corresponding P_is) as:

$$\left\{ \begin{array}{lcl} \alpha_0 & = & h(0) + r \\ \alpha_n & = & h(n+1) - r \\ \alpha_i & = & d(P_i, P_{i+1}) \end{array} \right. \tag{7.4}$$

where $d(P_i, P_{i+1})$ is the (Euclidean) distance between P_i and P_{i+1}, while r is the ratio of the distribution. This requires us to solve the system:

$$\left\{ \begin{array}{lcl} \sum\limits_{i=0}^{n} \alpha_i & = & d \\ \alpha_{i+1} & = & \alpha_i + r \end{array} \right. \tag{7.5}$$

to find both r and n. The solutions are:

$$n = \frac{2d}{h(0) + h(n+1)} - 1 \quad \text{and} \quad r = \frac{h(n+1) - h(0)}{n+2} .$$

As n must be an integer value, the solution is rescaled so as to obtain an exact discretization of the edge AB in terms of n and r. The α_is and thus the sequence of points is determined at the time n and r are established. Then, with each thus-defined point is associated a value, h, derived from the h's of the supporting edge. This means that the control space[9] is completed on the fly.

The process is repeated for all the current mesh edges and the series of points created in this way is then filtered, simply using a (structured) grid (cf. Chapter 1). This treatment is related to the fact that the vertices are well-positioned along one edge but this property does not hold globally. For instance, one may observe the case of all the edges emanating from one point. The retained points are then inserted using the Delaunay kernel (Relation (7.2)) and the entire process is iterated as long as some mesh edges need to be subdivided, i.e., are still not saturated.

It could be noted that this rather simple method is independent of the spatial dimension.

7.3.4 Optimization

Once the field points have been inserted, we have constructed a mesh for the domain that needs to be optimized to some extent. The goal is to optimize the mesh with respect to a quality criterion which is suitable for our purpose (a finite element style computation). Indeed, while being Delaunay (in most of the domain, in specific far from the boundaries), the mesh quality is not necessarily what is needed. This means that the Delaunay criterion (i.e., a bound about the angles) is not, *stricto sensu*, a quality criterion.

Up to now, we have considered a classical mesh generation problem. The aim is then to produce well-shaped elements, in other words isotropic elements that are as regular as possible (equilateral triangles in two dimensions and regular tetrahedra in three dimensions[10]). In terms of sizes, we have very little information about what is expected, so we try to conform as best we can to the sizes defined at the boundaries and, elsewhere, to have a reasonably smooth variation.

While various quality measures have been proposed (Chapter 18), a "natural" measure for the quality of a simplex is:

$$\mathcal{Q}_K = \alpha \frac{h_{max}}{\rho_K} \tag{7.6}$$

where α is a normalization factor such that the quality of a regular element is one, h_{max} is the longest edge of the element, i.e., its *diameter* and ρ_K is its inradius. This quality adequately measures the shape or aspect ratio of a given element. It ranges from 1, for an equilateral triangle (regular tetrahedron), to ∞, for a totally flat element; to return to a range of variation from 0 to 1, the inverse of $\mathcal{Q}_K$ could be used. Based on the above element quality, the quality of a mesh, $\mathcal{T}$, is given by:

$$\mathcal{Q}_M = \max_{K \in \mathcal{T}} \mathcal{Q}_K. \tag{7.7}$$

[9]The control space (see Chapter 1) is the current mesh considered together with the hs of its vertices.

[10]A regular tetrahedron is an element with equilateral triangular faces.

The aim is then to minimize this value. It could be observed that the point placement method results, in principle, in a good location for the field points. If so, then good quality elements may be expected. While effective in two dimensions, this result is not so easily attained in three dimensions, mainly due to slivers.

Optimization procedures. The aim is to optimize the current mesh by means of local modifications. In this respect, two categories of optimization techniques can be identified (Chapter 18), the topological techniques that preserve the point coordinates and modify their connections and the metric techniques that move the points while preserving vertex connectivity.

The local optimization operators associated with these techniques make it possible to move the nodes (for example, using a weighted barycentrage); to remove points; to remove edges (for example, by merging their endpoints) and to flip edges (in two dimensions) or edges and faces (generalized edge swapping, in three dimensions).

7.3.5 Practical issues

In this short section, we would like to give some indications regarding computer implementation of the above scheme.

In terms of basic algorithms. Four steps of the previous scheme require careful computer implementation. The major effort concerns an efficient implementation of the Delaunay kernel, then the boundary enforcement problem as well as the optimization process must be considered together with the point creation step.

Regarding the Delaunay kernel, the operations that are involved are:

- a fast searching procedure so as to define the *base*,
- a convenient way of passing from one element to its neighbors to complete the *cavity* by adjacency,
- an inexpensive evaluation of the circumcenters and the circumradii of the mesh elements to evaluate the Delaunay criterion,
- a low cost update of a mesh when inserting a point.

Regarding the point creation step, the creation itself proves to be inexpensive while the filter which is needed can be relatively time-consuming. A grid is then constructed to minimize the cost of this task (for instance, using a *bucket sorting* algorithm; see Chapter 2). Moreover, a cloud of points is inserted randomly (especially when the clusters contain a large number of points) in order to make the process more efficient[11].

Regarding the boundary problem, local modification operators must be carefully implemented. Note that this is also of interest for the optimization step as the required operators are basically the same.

[11]Note that other more subtle strategies can be employed to minimize the overall cost of this point insertion process.

Remark 7.4 *In the previous computational issues, we have not really mentioned accuracy problems or round-off errors. Indeed, the only thing we need is to be sure that a surface area (volume) is positive (to ensure the visibility criterion for a cavity as already discussed or that an element is valid (at the boundary or optimization step) which return to the same type of control.*

In terms of memory resources and data structures. First, it could be noted that a simple data structure probably provides a good chance of reaching a desirable level of efficiency. Moreover, the memory resources must be minimized as much as possible, which is the case with a simple structure.

Thus the internal data structure used must store (and maintain) the following (according to the facilities of the programming language):

- the point coordinates,
- the element vertices,
- the element neighbors (in terms of edge (face) adjacency),
- the element circumcenters,
- the element circumradii,
- some extra resources (for instance, for the grid used for the above filter).

7.3.6 Application examples

In this section, we give some application examples in both two dimensions and three dimensions. Some statistics are also presented.

In two dimensions, a mesh quality, i.e., $\mathcal{Q}_M$ of Relation (7.7), close to one can be expected regardless of the polygonal discretization of the domain boundary (meaning that equilateral triangles can be constructed[12] whatever the size of the given edges serving at their basis). In three dimensions, the expected value for $\mathcal{Q}_M$ depends on how good a tetrahedron can be constructed for each given surface triangle. Hence, the three-dimensional quality is related to the quality of the surface mesh serving as data.

Table 7.1, recorded in 1997, gives some statistics about a selected series of examples of different geometries. In this table *np* is the number of vertices, *ne* is the number of tetrahedra, *target* is the targeted value for the tetrahedron with the worst quality while $\mathcal{Q}_M$ is the value obtained. The row $1 - 2$ indicates the percentage of elements for which $1 < \mathcal{Q}_K < 2$, thus with a nice aspect ratio (i.e., close to a regular tetrahedron) while t is the CPU time (HP 9000/735 at 100 MHz) required to achieve the mesh (including i/o).

It should be noted that $\mathcal{Q}_M$ is indeed in the range of *target* and that the number of nicely shaped elements is greater, in proportion to the total number, if the domain volume is large.

[12]Obviously, if the point leading to this triangle falls within the domain.

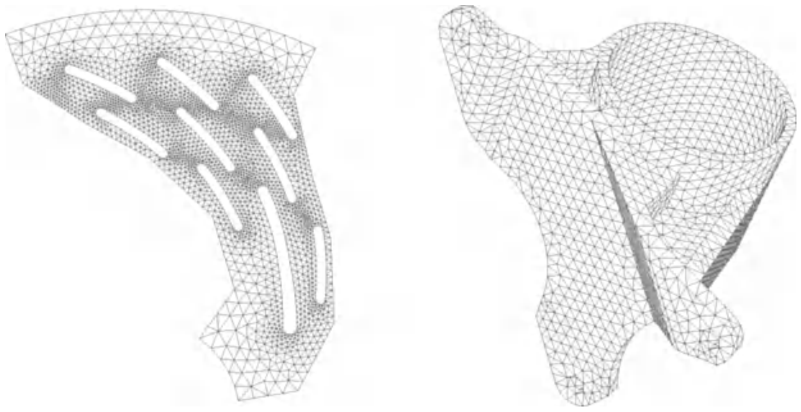


Figure 7.11: *Left-hand side: a two-dimensional geometry. The mesh contains* 2,445 *vertices and* 4,340 *triangles. The worst quality is* 1.12. *Right-hand side: a three-dimensional geometry (Data courtesy of MSC). The resulting mesh includes* 13,001 *tetrahedra and* 3,703 *vertices. The quality is* 5.83 *while the CPU cost is* 1.73 *seconds (HP9000/C180).*

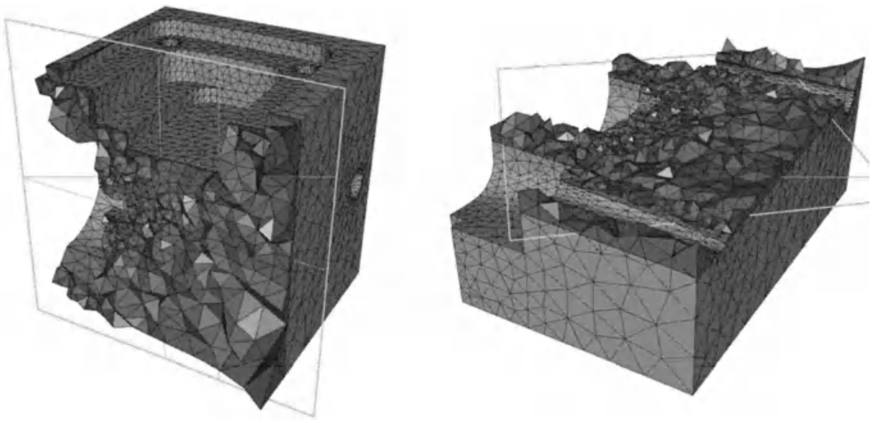


Figure 7.12: *Two cuts through a three-dimensional mesh for a view of the shape and gradation of the internal tets.*

-	np	ne	Q_{th}	Q_M	$1-2$	t
Case 1	105	286	7.38	9.61	66	0.36
Case 2	880	2,658	6.01	7.05	78	1.19
Case 3	1,917	7,230	10.24	11.65	76	2.75
Case 4	62,304	369,304	38.06	42.06	91	53.30

Table 7.1: Statistics related to the four selected examples (classical isotropic meshes constructed solely from the data of a discretization of the corresponding domain boundaries).

To appreciate more precisely the efficiency of the mesh generation algorithm, we can see Table 7.2, also recorded in 1997, where v is the number of elements constructed within a minute, for six selected examples with various sizes. Moreover, we denote by *Del* the part, as a percentage, of the mesh generation algorithm devoted solely to the triangulation process (the point insertion process).

It can be seen that v is directly related to *Del* and that v is less for the small meshes than for the large ones.

-	np	ne	t (sec., HP 9000/735)	v (ne/mn)	*Del*
Example 1	1,014	3,601	1.54	140,000	17
Example 2	36,252	191,279	28.26	406,000	49
Example 3	62,495	369,304	53.30	415,000	44
Example 4	214,184	1,243,871	179.	417,000	46
-	np	ne	t (sec., HP PA 8000)	v	*Del*
Example 5	518,759	3,067,937	373.62	492,000	54
Example 6	1,452,192	8,700,574	1125.	464,000	49

Table 7.2: Mesh generation algorithm efficiency.

To conclude, notice that the theoretical complexity of Delaunay type methods is not easy to calculate. We have mentioned that the geometry of the domain influences only a moderate amount of the global computational cost of the algorithm. However, the boundary enforcement procedure can be costly in some cases. Nevertheless, the theoretical complexity is mostly in $O(n \log(n))$, n being the number of mesh elements, [George, Borouchaki-1997], as in the triangulation algorithms.

Recent computers and recent advances in the way in which algorithms are encoded and cache miss are considered allow, with no surprise, for better performances as can be seen in Table 7.3. By comparison, we indicate that a triangulation algorithm allow for more than $500,000$ elements within a second!

7.4 Other methods

A particular Delaunay type mesh generation method has been discussed based on the scheme given in the previous sections. However, most of the steps included

-	np	ne	t (sec., Mac G5)	v (ne/mn)	*Del*
Example 1	5,031	23,566	1.65	910,000	12
Example 2	15,237	85,393	2.58	2,000,000	20
Example 3	338,129	1,952,673	45.38	2,650,000	32

Table 7.3: Mesh generation algorithm efficiency.

in this scheme can be modified, giving rise to various different approaches. In the following paragraphs, we give some indications about the main variants that have been suggested.

7.4.1 Point insertion methods

Using an enclosing box is not strictly required. For instance, it is possible, given a set of points, to sort them so as to ensure that a given point, P_{i+1}, is always *exterior* to the current mesh (based on the i previous points). Applying such a trick to the boundary points results in the mesh of the convex hull of this set of points. Then, internal points can be created and inserted returning to the classical point insertion method or an equivalent method. For instance, in two dimensions, a point is inserted in its *base* by splitting it and some diagonal swapping completes a Delaunay type mesh.

7.4.2 Field point creation

Some popular methods which are different from those based on the edge analysis given above, also prove useful for point creation. Among them are:

- the creation, under some conditions (size or shape), of the centroid of the elements in the current mesh, [Hermeline-1980],
- the use of the circumcenters as internal points. In two dimensions, this leads to a lower bound on the angles of the triangles, [Holmes, Snyder-1988], [Chew-1989b]. Devised in the early 1980s, this method was retained only as one of the possible solution and not more (indeed, it does not give a rigorous control about any prescribed size or anisotropic needs; see below) but, surprisingly, reappeared 20 years after in various methods, called Delaunay refinement methods,
- an advancing-front point placement strategy as discussed in Chapter 6,
- the use of the "variogram" [Tacher, Parriaux-1996]; such a method considers inserting the point maximazing the distance to the already inserted vertices, thus meeting ideas like medial axis,
- and many other methods, for instance, using a pre-determined set of points, [Weatherill-1985], [Baker-1986], [Mavriplis-1990], etc.

We briefly return to an interesting approach that combines an advancing-front point insertion technique with the insertion of the set of points using the Delaunay kernel.

Combined advancing-front-Delaunay approach. In this approach, the internal points correspond to optimal points created from a front, the elements being created during the insertion of these points into the current mesh using a Delaunay approach. This technique has been suggested in two and three dimensions (see [Merriam-1991], [Mavriplis-1992], [Muller *et al.* 1992], [Rebay-1993] and [Marcum, Weatherill-1995], [Frey *et al.* 1998], for example).

Here, the *front* represents the interface between two elements, one being classified as acceptable and the other not. As with the classical advancing-front approach, the initial front is the given boundary discretization. A first mesh of the bounding box of the domain is created that contains no (or very few) internal points and the boundary integrity is enforced (cf. Sections 7.3.2 and 7.3.3). The tetrahedral elements in this mesh are classified into two categories; *accepted* (that of quality compatible with the target quality) and those to be *processed*. By assumption, all external tetrahedra are accepted. The *active* elements, adjacent to accepted elements, serve to define the current front. The front is then composed of the set of faces that separate an accepted element from an unaccepted one. The algorithm stops when all elements are accepted.

For each front item, an optimal point is computed so that the element formed by this point and the front item would be optimal. If an optimal point is located outside the circumscribed disk (resp. ball) associated with the active element it should be connected with, this point is rejected because, during its insertion using the Delaunay kernel, the corresponding cavity will not contain the active element to be deleted. The remaining points are then filtered and inserted into the current mesh using the Delaunay kernel.

The final mesh is then optimized in the same way as in the classical approach.

7.4.3 Boundary enforcement

In practice, variations can be developed to carry out the boundary enforcement step included in a Delaunay type meshing method.

- The boundary (and more generally the specified entities) are enforced after the field points have been inserted.

- The specified entity not present at the time their endpoints are inserted are split so as to ensure that the resulting partition is automatically formed (we encounter here the notion of a *Delaunay admissible entity* or something more or less equivalent (say protecting balls) as already seen and as will be discussed in detail in Chapter 9).

- The missing boundary entities are split by means of intersecting points (an entity is missing because another one intersects it) therefore geometrically recovered before removing at the best the intersection points. This constructive method, described in [George, Borouchaki-2002], also gives an existence proof. The sketch of the method is shown in Figure 7.13 where a rather simple example in two dimensions is depicted.

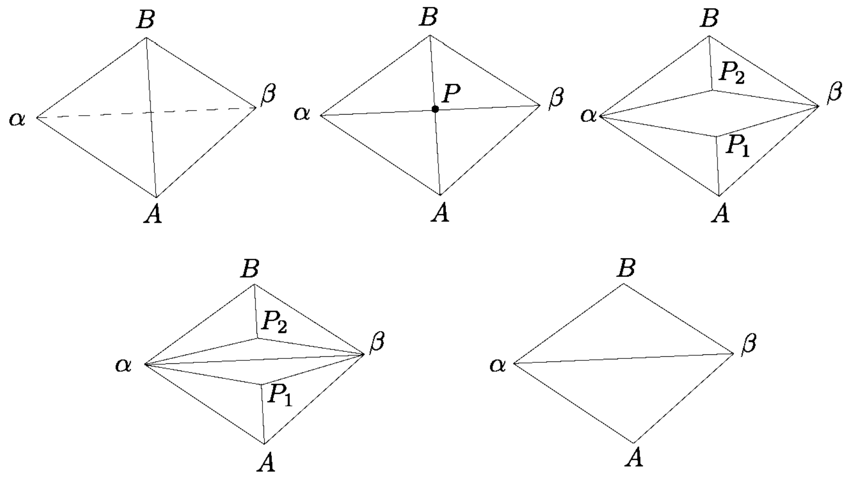


Figure 7.13: *Top left: the initial pattern (as created after the insertion of the four vertices) where edge AB exists while edge $\alpha\beta$ is expected. Top middle: point $P = AB \cap \alpha\beta$ is inserted in the pattern leading to four triangles. Top right: Point P is duplicated in P_1 and P_2, edge AP is replaced by edge AP_1, edge BP is replaced by edge BP_2. Bottom left: polygon $\alpha P_1 \beta P_2$ is such that edge $\alpha\beta$ exists. Bottom right: Vertex P_1 together with vertex P_2 are removed leading to the expected solution, e.g. edge $\alpha\beta$ exists in the current mesh.*

7.5 Isotropic governed Delaunay meshing

Up to now, the mesh generation problem, the so-called classical problem, has been to mesh as best we can a domain using a discretization of its boundaries as the sole data input. We now turn to a different meshing problem.

We consider a domain provided through an adequate boundary discretization and we assume that the sizes (and, possibly, the directional features) of the elements that must be constructed are given. The problem becomes how to construct a mesh conforming to the above specifications. In the case where the information only specifies the element sizes, we have an *isotropic* mesh generation problem whereas in the case where directional information is specified, we have an *anisotropic* meshing problem.

In principle the scheme proposed in Section 7.3.1 is still valid while it is needed to replace some of its components. At first, it can be proved that the Delaunay kernel is an adequate technique, without modification, to insert a point in a mesh. Actually, the main difference lies in the way the field points must be created so as to conform to the given size specification. On the other hand, slight variations in the optimization procedures must be also considered. A rather elegant way to achieve the desired solution is to introduce the notion of a *unit mesh* as already seen in the previous chapters and to use this when analyzing the edges. Moreover, some modifications must be made at the optimization phase.

Control space

The specification of a size distribution function enables us to define a control space. Notice that this data, common to most mesh generation algorithms (see, for instance, Chapters 5 and 6), is not so obvious to construct. A popular approach uses a grid or a background mesh whose cells (elements) encode the information about the desired element size. In practice, a size distribution function is associated with this domain covering-up. This simply means that a space control (Chapter 1) is defined in this way. If, from the spatial aspect, the control structure is a background mesh without internal points, then the size distribution depends only on the sizes of the boundary entities. In the following section, we will see how to use this control space.

Using a background mesh. Provided with a background mesh[13], the size function is known in a discrete way, say at the vertices of this background mesh. Using this discrete map, a continuous size map can be obtained by means of interpolation. Let P be an arbitrary point in the domain, the size $h(P)$ at P can be computed using the sizes $h(P_i)$, $i = 1,..d$ at the vertices P_i of the element enclosing P by means of a P^1 type interpolation, i.e.,

- we find the element K within which point P falls,
- we compute $h(P)$ as the P^1-interpolate of the sizes $h(P_i)$ at this element vertices P_i.

In the case of an empty background mesh, the size distribution function is strongly related to the boundary discretization. Indeed, the sizes are computed based on this data alone.

Remark 7.5 *An alternative solution is to define the background mesh by a uniform grid or a quadtree-octree type structure.*

Field point creation

Provided the space control is well defined, it is now possible to turn to the creation of the field points. This is done based on the analysis of the edges of the current mesh. The aim is to construct good quality elements whose size is in accordance with the given specification. This latter requirement is satisfied if the edge lengths are close to one in the corresponding metric (Chapter 1). Bear in mind that a

[13]For instance, a classical mesh (the mesh obtained at the previous iteration step in an adaptive process; cf. Chapter 21) or the boundary mesh completed by the boundary enforcement phase.

mesh whose elements conform to a size specification is nothing other than a *unit mesh.*

Edge length. Let AB be an edge in the mesh. If, t lying between 0 and 1, $h(t)$ stands for a sizing function defined for this edge ($h(0) = h(A)$, $h(1) = h(B)$), then the length of edge AB with respect to $h(t)$ is:

$$l_{AB} = d_{AB} \int_0^1 \frac{1}{h(t)}\, dt \tag{7.8}$$

where d_{AB} is the usual (Euclidean) distance. Then, by definition, an edge AB conforms to the size specification if:

$$\frac{1}{\sqrt{2}} \leq l_{AB} \leq \sqrt{2}\,.$$

The key idea of the point placement strategy is to construct the points so that each segment defined by a pair of neighboring points is of length one (or close this value). It could be observed that this notion is consistent. Indeed, if $h(t)$ is a constant function, say $h(t) = h, \forall t$, then $l_{AB} = 1$ means that $d_{AB} = h$ which is the expected value.

Field point creation. Now, based on this notion, the field points can be created by modifying the classical point creation method accordingly. Note that the edge-based method is therefore a rather elegant method to which the above concept applies without major difficulty.

In practical terms, an iterative procedure is used, which aims at saturating the internal edges of the current mesh. The edges are analyzed and their normalized lengths are calculated using an approximate method described below. The goal is to construct optimal elements (that are as regular as possible) of unit size. Let δ be a fixed threshold (usually $\delta < 1$, $\delta = 0.5$ for example). If $l_{AB} < \delta$, the edge AB is not split. Otherwise, the middle point Q_1 is introduced, and the process is iterated onto each of the sub-segments AQ_1 and Q_1B. We then find a series of points Q_i such that $l_{Q_iQ_{i+1}} < \delta$ and that:

$$l_{AB} = \sum_i l_{Q_iQ_{i+1}}\,.$$

The total number of points is then known along each edge. The next step consists of finding the location of the points along the edges. To this end, an index i is identified such that:

$$l_{0,i} = \sum_{j=0}^{j=i} \delta_j > 1$$

and the point P_1 is introduced, corresponding to the average value between the points Q_i and Q_{i+1}, weighted by the difference to one of this sum:

$$P_1 = Q_i + \omega \overrightarrow{Q_iQ_{i+1}}\,,$$

where $\omega = \frac{1-l_{0,i-1}}{l_{i-1,i}} d_{i-1,i}$, l_{i-1} denotes the normalized length of segment $Q_{i-1}Q_i$, $d_{i-1,i}$ being the Euclidean distance between the points Q_{i-1}, Q_i. The process is iterated until all edges are saturated.

Similar to the classical case and for the same reason, the points are filtered and inserted into the current mesh using the Delaunay kernel. Moreover, to obtain an optimal mesh, a constraint related to the areas (volumes) of the elements is added. The latter is not strictly required in two dimensions, a triangle having all of its edges of unit lengths is necessarily optimal. However, in three dimensions, the same property does not hold, and explicit control of the volumes is thus required[14].

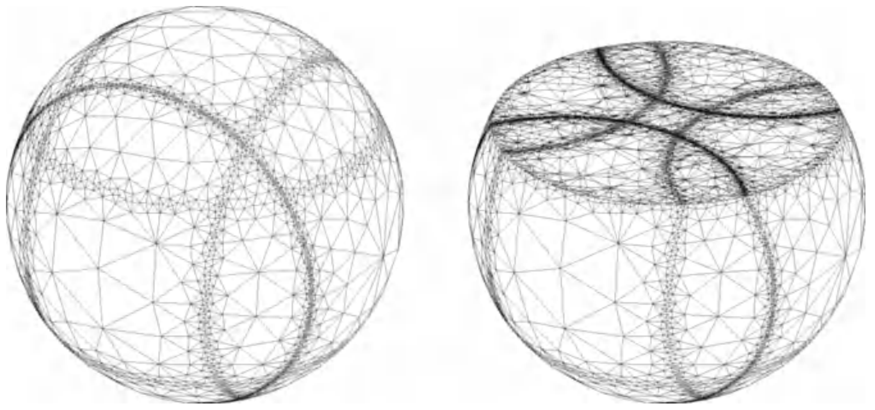


Figure 7.14: *An academic example of an isotropic governed mesh (see Chapter 21 for realistic cases) where the size function is analytic.*

Optimizations. Similarly to the classical case, it is generally useful to optimize the resulting mesh. The goal is to achieve well-shaped elements, which are as regular as possible, and whose sizes are compatible with the given specifications. The optimization stage is based on the same procedures as in the classical case, with a size-based criterion. For practical reasons, this procedure is performed in two steps, the mesh elements are first optimized with respect to the size criterion and then optimized with respect to the shape criterion.

In this discussion, the size prescription is assumed to be known analytically, the real case will be discussed in Chapter 21.

7.6 Extensions

A weighted Delaunay triangulation can be seen as a Delaunay triangulation where weights are associated with the given set of points.

[14]A *sliver* is an element of almost null volume although its edges could be almost of unit length and the aspect ratio of its four faces is nice.

In this section, we briefly discuss weighted triangulation before turning to cases where the information associated with the elements concerns sizes as well as directions, where the construction step aims at creating *anisotropic* meshes, that is meshes in which the elements are stretched (for instance, those found in computational fluid dynamic simulations). Therefore, we briefly analyze the construction of anisotropic meshes, the construction of surface meshes. We then give some indication of how to use the Delaunay approach for creating meshes in an adaptation scheme.

7.6.1 Weighted Delaunay triangulation

We consider a set of points $\mathcal{S}$ where a positive or zero weight ω_P^2 is associated with P, for all P in $\mathcal{S}$. Constructing a weighted triangulation of $\mathcal{S}$ relies on constructing a Delaunay triangulation where the (classical) distance from point to point is replaced by a power taking into account the given weights [Aurenhammer-1987].

The pair (P, ω_P^2) can be seen as a sphere of radius ω_P^2 centered in P. Given another point Q, we define $\Pi(P, \omega_P, Q) = d(P,Q)^2 - \omega_P^2$ where $d(P,Q)$ is the usual Euclidean distance. Thank to this definition, we define the so-called power diagram[15] (as we did for the Voronoï diagram):

$$V_i = \{P \quad \text{such that} \quad \Pi(P_i, \omega_{P_i}, P) \leq \Pi(P_j, \omega_{P_j}, P), \quad \forall j \neq i\} \tag{7.9}$$

If, for all pair, $\|\omega_P^2 - \omega_Q^2\| \leq d(P,Q)^2$ holds, both the diagram and its duality, the weighted triangulation, exist.

The construction of weighted triangulations and mainly the way to insert a weighted point relies in applying the Delaunay kernel described in the classical case by replacing the distance d by the power Π and then most of the classical practical issues apply.

The main interest is that weights modify the cells in the diagram as well as the triangulation is the sense that, in the diagram, the separators from point to point can be pushed towards one or the other (instead of being located at the midplace) and, therefore, the connections in the triangulation are affected accordingly.

An interesting application of such a construction resulting in hybrid meshes made up of arbitrary polyhedra (eg the cells of the power diagram) can be found in [Borouchaki *et al.* 2005] where the goal was reservoir simulations.

7.6.2 Anisotropic Delaunay meshing

Similarly, the approach adopted for the classical scheme can be followed but, in addition, it is necessary to modify the method used to create the field points (such as in the previous isotropic governed method), and the Delaunay kernel[16] itself must be extended. These two specific aspects are now briefly presented (see [George, Borouchaki-1997] for a detailed discussion).

[15]Also referred to as the Laguerre diagram.

[16]See the theoretical remark at the end of this chapter.

Edge length with respect to a metric. First, we give a more subtle form of Relation (7.8). Again let AB be an edge, which can be defined as: $AB(t) = A + t\overrightarrow{AB}, \quad t \in [0,1]$. Let $\mathcal{M}(M(t))$ be a two by two matrix defined by:

$$\mathcal{M}(M(t)) = \begin{pmatrix} \frac{1}{h^2(t)} & 0 \\ 0 & \frac{1}{h^2(t)} \end{pmatrix}, \tag{7.10}$$

if we consider a two-dimensional problem (and a similar three by three matrix in three dimensions) where $h(t)$ is the specified size expected at point $M(t)$, the distance between A and B with respect to the metric $\mathcal{M}$ is then:

$$l_{\mathcal{M}}(AB) = \int_0^1 \sqrt{{}^t\overrightarrow{AB}\,\mathcal{M}(A + t\,\overrightarrow{AB})\,\overrightarrow{AB}}\,dt\,, \tag{7.11}$$

which is nothing more than a more general expression of Relation (7.8).

Field point creation. Internal point creation for an anisotropic mesh is performed by replacing the length calculation of the classical scheme. More precisely, we use the previous definition of the normalized lengths, by replacing the matrix $\mathcal{M}(M(t))$ with the matrix (in two dimensions):

$${}^t\mathcal{R}(t) \begin{pmatrix} \frac{1}{h_1^2(t)} & 0 \\ 0 & \frac{1}{h_2^2(t)} \end{pmatrix} \mathcal{R}(t)\,. \tag{7.12}$$

In this expression, $\mathcal{R}(t)$ specifies the two directions that are expected at point $M(t)$ while $h_1(t)$ (respectively $h_2(t)$) indicates the desired sizes at the same point following the first (resp. the second) direction previously mentioned. The point placement strategy is then identical to that of the previous isotropic case.

Point insertion scheme. As mentioned earlier, the classical Delaunay kernel is no longer suitable for an anisotropic mesh generation problem. Therefore, the classical construction (Relation (7.2)),

$$\mathcal{T}_{i+1} = \mathcal{T}_i - \mathcal{C}_P + \mathcal{B}_P\,, \tag{7.13}$$

where $\mathcal{C}_P$ is the *cavity* and $\mathcal{B}_P$ is the *ball* associated with point P, must be extended to the present context.

The idea is to define $\mathcal{C}_P$ so as to conform to the anisotropy which is desired. To this end, we detail the condition which implies that a given element is a member of $\mathcal{C}_P$. Indeed, in the classical context, an element K is in the cavity if:

$$\alpha(P,K) = \frac{d(P,O_K)}{r_K} < 1\,, \tag{7.14}$$

(d being the usual distance), with O_K the center of the circumcircle (circumsphere) corresponding to K and r_K the radius of this circle (sphere). We first replace this relation by:

$$\alpha_{\mathcal{M}}(P,K) = \frac{l_{\mathcal{M}}(P,O_K)}{r_K} < 1 \tag{7.15}$$

where now,

- O_K stands for the point equidistant to the vertices of K, in terms of $l_{\mathcal{M}}$ the length related to the matrix (metric) $\mathcal{M}$, and
- r_K is $l_{\mathcal{M}}(P_1,O_K)$, P_1 being one of the vertices of K.

As a result, the cavity is evaluated following the anisotropic context. In practice, a computer implementation of the above relationship is not practical since a non-linear system is involved. Thus, an approximate solution must be found. A possible answer could be to solve the following system:

$$\begin{cases} l_{\mathcal{M}(P)}(O_K,P_1) &= l_{\mathcal{M}(P)}(O_K,P_2) \\ l_{\mathcal{M}(P)}(O_K,P_1) &= l_{\mathcal{M}(P)}(O_K,P_3) \end{cases} . \tag{7.16}$$

In this way, O_K is obtained (P_i being the vertices of K). Then we check if:

$$\alpha_{\mathcal{M}(P)}(P,K) < 1\,, \tag{7.17}$$

where $\mathcal{M}$ is approached by $\mathcal{M}(P)$, the metric associated with the point under insertion. The following theorem, which holds in two dimensions, shows that the previous characterization is constructive.

Theorem 7.5 *The relationship $\alpha_{\mathcal{M}(P)}(P,K) < 1$ results in a valid Delaunay kernel.*

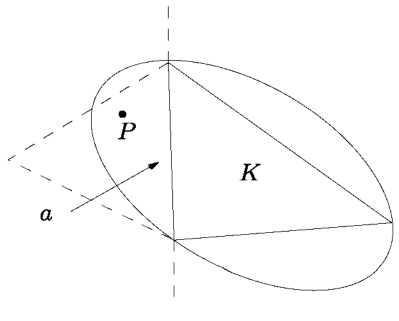


Figure 7.15: *A schematic for the proof of Theorem 7.5.*

The proof of the above theorem consists of checking that the so-defined cavity is a star-shaped region with respect to P. The cavity is initialized with the base of P, the latter being obviously star-shaped with respect to P. Then, this cavity is enriched by adjacency. Given a star-shaped cavity including a certain number of elements, we simply have to prove that adding one element, using the above criterion, preserves the star-shapedness of the resulting cavity. As every point in the ellipse $\alpha_{\mathcal{M}(P)}(P,K) = 1$ is visible by P and as the edge (denoted by a in Figure 7.15, where the triangle being processed is depicted by a continuous line while the triangle in dotted lines is still in the cavity. The ellipse circumscribing triangle K has been evaluated using the metric of P), common to the current cavity and the element K in question, separates the ellipse into two non-connected components. Thus, P is visible by the two other edges of K, edges that will be part of the new cavity.

Unfortunately, as the above separation property does not hold in three dimensions, this result does not extend to this case and a more subtle method must be used. Indeed, we consider exactly the same construction but we have to verify explicitly that the current cavity is a star-shaped region (this is completed by an algorithm that corrects a possibly invalid cavity, thus returning to the classical cavity correction algorithm).

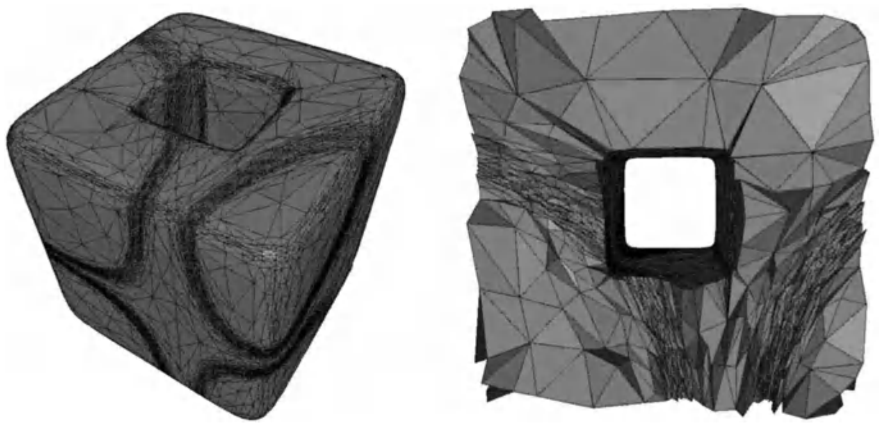


Figure 7.16: *An academic example of an anisotropic governed mesh (see Chapter 21 for realistic cases) where the metric function is analytically defined.*

To conclude, it should be noted that other approximate solutions can be used resulting in the desired Delaunay type kernel. These solutions infer point P and some other points naturally present in the context.

Optimizations. Similar to the governed anisotropic case, an optimization stage aims at optimizing the size and shape quality of the resulting mesh. This stage is based on local topological and geometrical operations governed by the metrics in hand.

A theoretical remark. As discussed in this section, a Delaunay type method has been advocated to handle anisotropic meshing problems. In this respect, the classical Delaunay kernel (as itself or after an approximation) has been used to connect the vertices.

A purely theoretical view of the problem could refute our point of view. In fact, due to the metric map supplied as a sizing (directional) information, the basic notion of a distance between two points is no longer Euclidean. Indeed, the distance measure varies from one point to the other. In terms of the Voronoï diagram that can be associated with a set of points in such a metric context, we do not return to the classical diagram. For instance, the perpendicular bisector of two points (in $\mathbb{R}^2$) is not a straight line but a curve. Thus, the Delaunay triangulation, the dual of this diagram, is not composed of *affine* (straight-sided) triangles (considering again a two-dimensional problem). Indeed we face a Riemannian problem where the curve of minimum length between two points, the so-called *geodesic*, is no longer a straight segment.

Actually, a proper definition of the diagram is far from being evident and exhibiting a dual which will be called a Delaunay triangulation is not certain apart from being sure that this triangulation is valid. The classical diagram (Relation 7.1):

$$V_i = \{P \quad \text{such that} \quad d(P,P_i) \leq d(P,P_j), \quad \forall j \neq i\}$$

would be replaced by:

$$V_i = \{P \quad \text{such that} \quad l_{\mathcal{M}}(P,P_i) \leq l_{\mathcal{M}}(P,P_j), \quad \forall j \neq i\} \qquad (7.18)$$

where $l_{\mathcal{M}}(.,.)$ denotes the Riemannian distance between two points. The definition used for the above triangulation construction suggests considering:

$$l_{\mathcal{M}}(AB) = \int_0^1 \sqrt{{}^t\overrightarrow{AB}\,\mathcal{M}(A + t\,\overrightarrow{AB})\,\overrightarrow{AB}}\,dt\,,$$

while we were interested by affine triangles. Theoretically speaking, the distance between two points is not measured along a straight line segment but along a curve, suggesting considering a formula like:

$$l_{\mathcal{M}}(AB) = \int_0^1 \sqrt{{}^t\overrightarrow{\gamma'(t)}\,\mathcal{M}(\gamma(t))\,\overrightarrow{\gamma'(t)}}\,dt\,, \qquad (7.19)$$

with $\gamma(0) = A$ and $\gamma(1) = B$ and γ an unknown curve.

Using such a definition is computationally unrealistic and numerical approximations (quadratures) may result in a solution but may cause failure in the theoretical analysis. Thus, the hypothetical diagram and its hypothetical dual are far from being well defined and usable. Some works discuss these points such as [Leibon, Letscher-2000] and [Labelle, Shewchuk-2003] where a theoretical approach is presented showing some *a priori* surprising properties of the cells as they are defined and the way to obtain some consistency in terms of the dual.

Nevertheless, in our context, the only objective is to construct an affine triangulation and therefore the purely theoretical discussion above is not an issue. In fact, we use the Delaunay background as a basic scheme that enables us to design a constructive mesh generation method (which is no longer Delaunay).

7.6.3 Surface meshing

The generation of surface meshes is generally considered to be a difficult problem, for which two approaches are *a priori* possible, a direct and an indirect approach.

Direct approaches consist of performing a classical meshing technique directly onto the surface, without using any mapping. The element sizes and shapes are controlled by taking into account the local variations of the surface intrinsic properties. The boundary curves are first discretized before meshing the domain using a classical scheme. According to this scheme, a Delaunay type approach does not seem well-suited to performing this task. However, the indirect approach is widely possible.

Parametric surfaces. The generation of parametric surface meshes can be obtained via the mesh of the parametric space. This technique has been suggested by several authors (see for instance [George, Borouchaki-1997] or Chapter 15 for a detailed overview of this approach). The main interest of this approach is to reduce the problem to a purely two-dimensional problem. Let Ω be a domain in $\mathbb{R}^2$ and let σ be a sufficiently smooth function, then the surface Σ defined by the application

$$\sigma : \Omega \longrightarrow \mathbb{R}^3,\ (u,v) \longmapsto \sigma(u,v)$$

can be meshed using a two-dimensional mesh generation method in Ω, then mapping this mesh, via σ, onto $\mathbb{R}^3$. The sole constraint is to have a method that allows anisotropic meshes to be created. The metric map is indeed based on the intrinsic properties of the surface. The Delaunay method appears to be particularly well-suited to this kind of problem.

★
★ ★

Delaunay type mesh generation methods prove to be efficient, robust and flexible enough to handle classical as well as governed mesh creation (Chapter 21).

In addition, it could be noted that anisotropic Delaunay type meshing methods will be a key ingredient for parametric surface mesh generation (Chapter 15).

Chapter 8

Other Types of Mesh Generation Methods

This chapter briefly discusses some mesh generation methods which do not belong to the classical methods covered in the previous chapters. The fact that there is such a wide variety of methods is some indication of the richness of the subject but also indicates that there is no one method[1] that is a universal solution for all possible situations. Thus, for instance, a semi-automatic method is sometimes more powerful, more flexible or easier to use for some cases compared with a purely classical automatic method. Moreover, some domains are defined by means of data input that is not directly suitable for an automatic approach.

★
★ ★

The first section discusses product methods, which represent an elegant meshing method when the geometry of the domain has the required aspect, i.e., a cylindrical analogy in terms of topology. Grid or pattern-based methods are then discussed which complete a mesh of a given domain starting from a given mesh composed of a simple grid or a partition covering the whole domain by means of a predefined pattern which is repeated. Therefore, the grid or the given patterns in the domain (or in a portion of this domain) can be found. A mesh generation method by means of optimization is briefly presented which is capable of handling problems with large deformation (such as forming processes). Moreover, this approach allows for more specific applications.

Constructing a quad mesh (in two dimensions) is then discussed in Sections 8.4 and 8.5 where indirect approaches (by means of triangle combination) and direct approaches are given respectively. The extension to hex mesh construction in three dimensions can be found in Section 8.6. Then, to conclude (without claiming exhaustivity), we mention some mesh generation methods that are well-suited to some specific applications. In particular, we focus on methods resulting in radial meshes and methods that make use of a recursive domain partitioning.

[1]As least, as far as we know.

8.1 Product method

The aim of a product method consists of creating elements in d dimensions from the data, on the one hand, of elements in p dimensions, $0 < p \leq d$, mapped in the space of d dimensions, and, on the other hand, of a meshed line serving as a *generation line.*

Basic principles

Thus, locally, a point (i.e., an element reduced to a point, a 0-dimensional item) defined in the p-dimensional space and seen in a d-dimensional space produces a series of segments (item of 1-dimension) defined in the d-dimensional space. Similarly, a segment (item of 1-dimension) defined in the p-dimensional space produces quadrilaterals in d dimensional space. A triangle serves as a support for creating pentahedra, while a quadrilateral produces hexahedra (Figure 8.1).

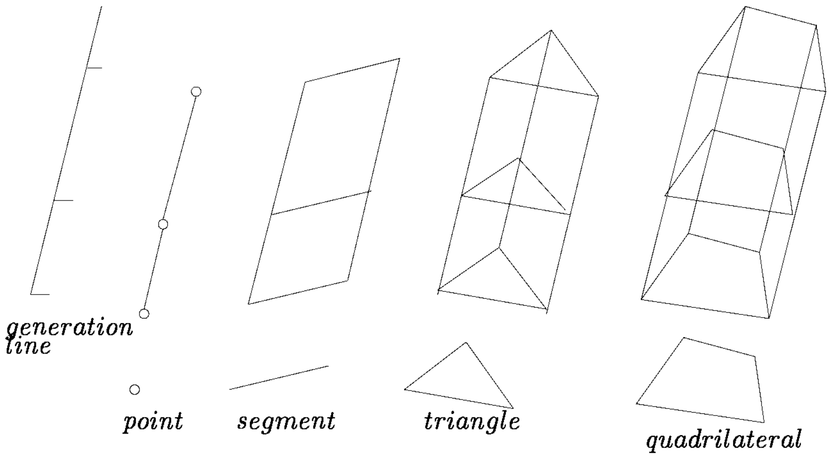


Figure 8.1: *Correspondences. On the left we show the generation line, then the construction by a product method associated with a point, a segment, a triangle and a quad.*

It could be observed that the basic mesh appears, in terms of topology, as a layer of the resulting mesh, such a layer being defined for each discretization step in the generation line. In this way, we obtain a cylindrical topology (which makes it clear in which cases the method applies).

Degeneracies may be encountered for some special positions of the generation line with respect to the given mesh serving as data. Such a phenomenum can be seen as the result of a merging operation where two vertices collapse due to some property of invariance. In this case segments may produce, not only quadrilaterals, but also triangles (two vertices collapse then resulting in a three vertex element), triangles may produce degenerated pentahedra and quadrilaterals may produce degenerated hexahedra. Depending on the geometry, the degenerate elements are valid (or not) in the usual finite element context. In this respect, for example, hexahedra may degenerate into pentahedra, which are admissible elements, or into non-admissible elements (Figure 8.2) with regard to the expected usage of these elements, while pentahedra may lead to the creation of tetrahedra or non-admissible elements (same figure and again related to the future application).

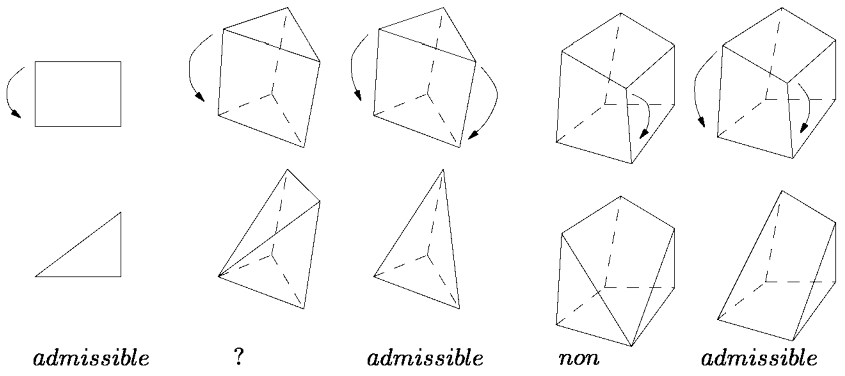


Figure 8.2: *Correspondences and possible degeneracies. On the left is a quad that degenerates into a suitable triangle. Then two forms of degeneracies for a pentahedral element are given, one of which is admissible while the other is suitable or not depending on the envisaged application. Finally, two patterns related to hex degeneracies are displayed, only one of the patterns being valid.*

Computational issues

In what follows, we discuss two methods that allow for the computation of the vertices in the various layers of the desired mesh. Then, we add some details about how to implement a product method.

Computing the intermediate vertices ("semi-manual" method). The vertices of the various layers are defined in a semi-automatic manner through a description of these layers. The possible choices, [George-1993], correspond to the following data input:

- a constant stepsize or a number of layers between the initial section (the basis) and the final one,
- a variable stepsize from one section to the next,
- the full definition of a function that associates a vertex position for each section with the current vertex position in the initial section,
- the full definition, for a given section (the source), of the way in which the next section (the target) must be constructed,
- etc.

Computing the intermediate vertices ("semi-automatic" method). More automatic, this method completes the vertices in the sections lying between two reference sections, [Staten *et al.* 1998], the first serving as a *source*, the other being the *target*. The first section is the initial section or basis (once the entire domain is considered). The target section is associated with an intermediary section or with the final section (the last target section). At the same time, we give how many sections must be constructed between the source and the target and we describe the side of the "cylinder" contained between these two particular sections by means of discretized edges meshed with consistency from one to the other. The source and the target sections are assumed to be known in a parametric form, then the domain could be subdivided into several portions and an automatic algorithm can be applied in each of them in order to complete the intermediary points based on the available data input. The computational scheme is as follows:

- a background mesh is constructed for the source section in its parametric space based on the points lying on the sides on this section. As assumed, this mesh is also a background mesh for the target section and thus will be used when the various intermediate sections are dealt with,

- the (internal) points in the source section are classified with respect to the background mesh (we find the triangle that contains such a point and we obtain its barycentric coordinates) and the same is done for the points in the target section. At completion, two sets of three values[2] $(a_i, b_i, c_i)^{source}$ and $(a_i, b_i, c_i)^{target}$ are available for the point P_i considered in the source (resp. target) section,

- the distances between these internal points (in $\mathbb{R}^3$) are computed with respect to the triangles whose vertices (denoted by S_i in what follows) are the images of the three vertices of the triangles in the background mesh containing these points (now considered in the parametric space). To this end, we use the following formula, for point P_i:

$$d_i = \|a_i\, S_1 \;+\; b_i\, S_2 \;+\; c_i\, S_3 \;-\; P_i\,\|\,,$$

  with respect to the source section for the points in this section and using a similar formula for the points in the target section with respect to this section. In this way, we obtain two distances d_i^{source} and d_i^{target} that measure the distortion[3] at point P_i for the two sections bounding the domain being processed.

This information is then used to compute the position of the internal points in the sections contained between the source section and the target section. This calculus, for the point P_i in a given section, includes a linear interpolation between

[2]If the shape (the geometry) of the source and the target sections are identical, these two sets are the same. Otherwise, for instance, the triangle containing the source point is not necessarily that containing the corresponding target point assumed to be in front. In this case, to retrieve the same element, negative barycentric coordinates are allowed.

[3]If, for example, the source section is planar, we have $d_i^{source} = 0$.

$(a_i, b_i, c_i)^{source}$ and $(a_i, b_i, c_i)^{target}$, a function of the section (defined through the points supplied in the sides of the above mentioned cylinder) and a correction step that modifies this value based on d_i^{source} and d_i^{target} (after an interpolation).

Implementation remarks The position of the intermediate points, the case where a degeneracy occurs or when the final section becomes identical to the first section (for instance, in the case of a rotation) having been successfully established, the computational aspect does not induce any difficulty. In practice, the mesh generation problem comes down to numbering the vertices of the different layers or sections corresponding to the discretization of the generation line. This numbering step is rather similar to that already seen in a multibloc method (see Chapter 4). Once this numbering has been done, the element vertex enumeration is trivial.

Limits

Product methods (also referred to as *extrusion* methods) can be applied to domains which have the desired properties, i.e., one of the following topological types:

- cylindrical topology: the domain can be described via the data of a two-dimensional mesh and a generation line defining layers with which the three-dimensional elements are associated,
- hexahedral topology: the domain can be described via the data of a one-dimensional mesh and a generation line to construct a two-dimensional mesh which, in turn, is coupled with a second generation line and produces the desired three-dimensional elements.

Note, as a conclusion about the limits, that the validity (positive surface areas or volumes, absence of auto-intersections (inter-penetrations or overlapping elements)) of the resulting mesh must be checked explicitly, which may increase the computational cost.

Application examples

Figures 8.3 and 8.4 display two examples constructed by a product method applied respectively to a meshed line and a two-dimensional mesh serving as data and a rotation used as a generation line. In the first example, the generation line (a circle) is not connected with the data and classical expected elements are constructed (quads), while, in the second example, the generation line is partially common to some edges of the data and pentahedra (instead of hexahedra) are produced in this region.

The examples in Figure 8.5 have been completed using a method described by [Staten *et al.* 1998] which is used in ANSYS. A mesh resulting from translation and rotation (left-hand side) and a mesh resulting from translation (right-hand side) can be seen. The latter example has been completed in three steps whose respective meshes, after merging, provide the final mesh. This partition into three parts has been made so as to make the product method suitable for each of them.

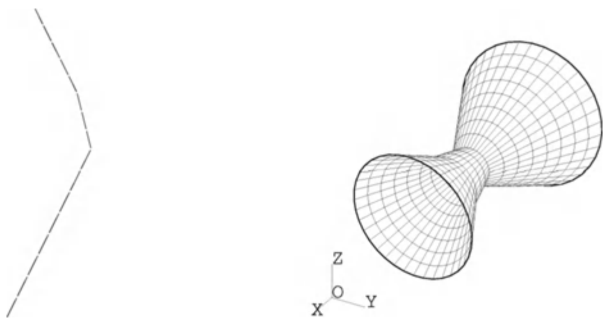


Figure 8.3: *Line serving as data (left-hand side) and resulting mesh (right-hand side) obtained by a product method corresponding to a rotation.*

This partition is defined so as to permit a compatible mesh merge (see Chapter 17 on how to merge two meshes).

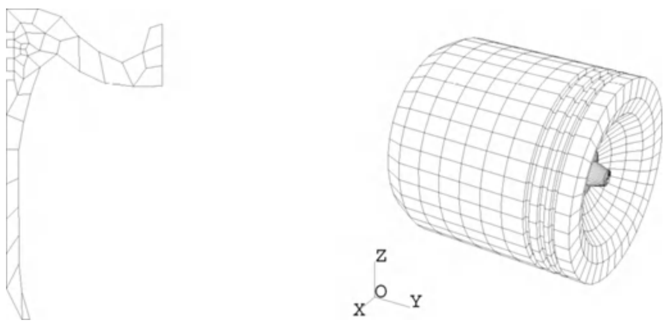


Figure 8.4: *2D mesh serving as data (left-hand side) and resulting mesh (right-hand side) obtained by a product method corresponding to a rotation.*

8.2 Grid or pattern-based methods

To some degree, these methods recall some of the constructions seen in Chapter 5 and also some ideas that will be seen in Chapter 16. The domain of interest is merged into a regular grid G composed of uniform squares (in two dimensions) or uniform cubes (in three dimensions). The cell size is a function of the smallest distance between two contour points of the domain and also depends on available memory resources. A meshing process based on the (easily) so-defined grid can be developed. It involves the following phases:

Step 1: the removal of the cells of G which do not intersect the domain.

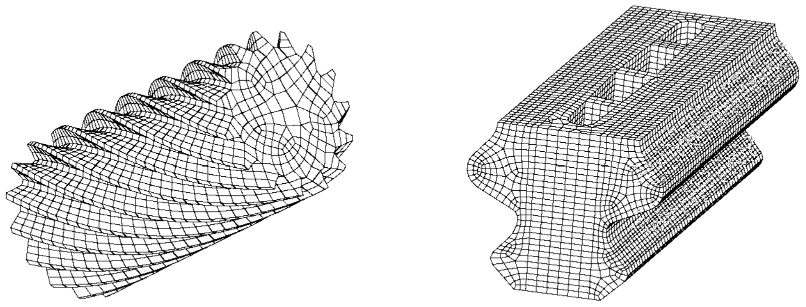


Figure 8.5: *Examples of meshes resulting from the product method incorporated in ANSYS. Left-hand side: the various sections result from the basis after a translation combined with a rotation. Right-hand side: the domain is split into three parts and, for each of them, a translation of the corresponding basis is used.*

Step 2: the processing of the cells of G containing a portion of the boundary of the domain; two variations can be advocated:

- a cell whose intersection with the boundary is not empty is considered as a mesh element; in this way, the final mesh will only be a piecewise linear approximation of the given domain (the accuracy of this approximation depends on the stepsize of the grid cells),
- or this type of cell is modified in such a way that the boundary is better approximated.

Step 3: the enumeration of the mesh elements:

- a purely internal cell becomes a mesh element (or can be split into simplices);
- a cell intersecting the boundary is considered as above. It defines a mesh element or it can be split into one or several triangular or quad elements, or tetrahedral or hex elements, such a splitting being based on the analysis of the different possible configurations and the use of the corresponding splitting pattern (Chapters 17 and 18).

Simple in principle, this method is unlikely to be suitable when the boundary discretization, serving as data, includes widely different items (edges in two dimensions) in terms of size while this situation generally requires a very fine stepsize and then may lead to an overflow in the number of cells of grid G, and consequently in the number of elements of the resulting mesh. As such, except for some specific problems, more flexible methods must be used (such as the quadtree-octree methods discussed in Chapter 5). Nevertheless, for certain specific applications (for example, for forecast computation), a relatively regular mesh is a source of simplification at the computational level.

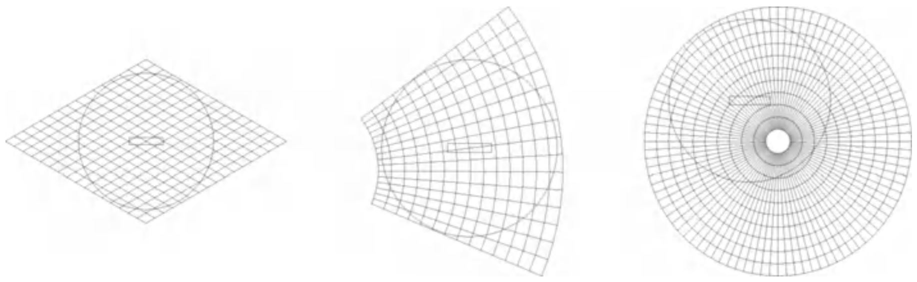


Figure 8.6: *The grid made of parallelograms (left), the portion of the ring made of radial quadrangles (middle) and the ring made of radial quads (right).*

The above approach can be followed not with a regular grid, but when a pre-specified mesh is given which encloses the domain. In this way, using as pre-specified mesh a mesh composed of pre-defined patterns results in a final mesh where most of the elements conform to this pattern. Thus, most of the domain is covered with the pre-specified patterns and the remaining regions (i.e., the regions intersected or close to the boundary where the given pattern is not suitable) can be dealt with either using a general purpose mesh generation method or a specific processing of the patterns that intersect the domain boundary.

Figures 8.6 and 8.7 demonstrate the principle of this method. In the first figure, one can see the domain in question, the part inside a circle having a rectangular hole at its centroid. Three enclosing meshes are depicted. On the left-hand side, the pattern is a parallelogram, in the middle, we have a part of a ring covered by quads and, finally, on the right-hand side, we have a ring covered with radial quads which are excentered with respect to the domain. In the other figures, we illustrate a plausible scenario as to what the construction could be. We see successively the patterns with (at least) one portion inside the domain, i), and in ii), the retained patterns, i.e., among the previous, we have discarded those partially included in the domain and those judged to be too close to the domain boundaries to define a large enough region in the vicinity of these boundaries. Then, in iii), the not yet meshed regions are meshed, this point concerns the regions close to the boundaries. Finally, in iv), the resulting mesh is visible and is formed by the patterns that have been retained and the mesh of the boundary regions as completed previously.

Various choices about the pattern are possible. For instance, in two dimensions, we may think of the pattern related to the union of six equilateral triangles (a regular hexagon) or to the configuration composed by elements whose vertices are located in a radial manner with respect to a given point.

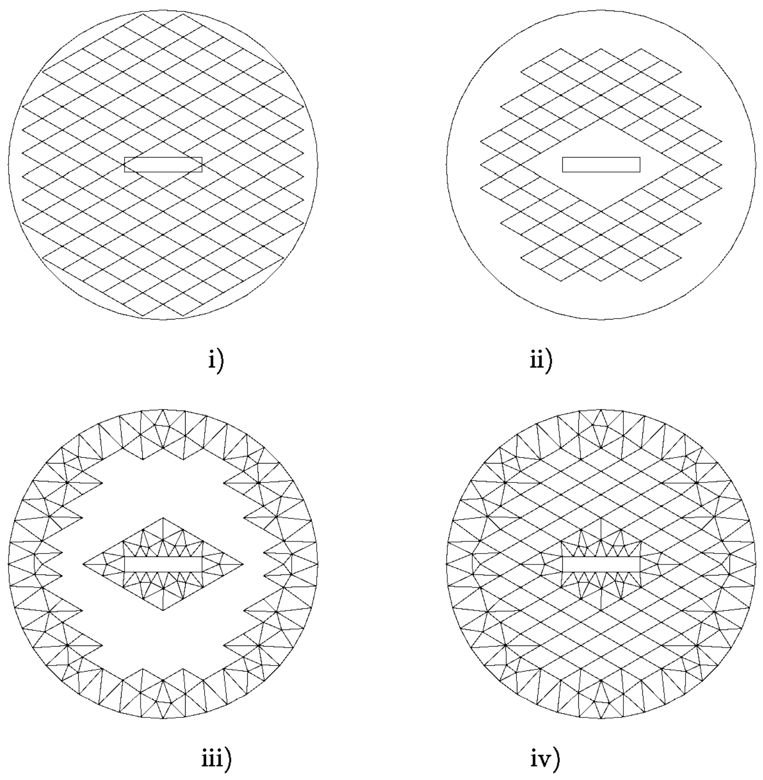


Figure 8.7: *From i) to iv), one can see the quads strictly included in the hollow disc, those included in this disc that are "consistent" with the boundary, the triangle mesh resulting from an automatic mesh generator and the final mesh of the disk.*

8.3 Optimization-based method

Proposed by [Coupez-1991] in the context of forming processes where the geometry can change dramatically from one time step to another, this method can also be used as a mesh generation method, a mesh optimization method or even a mesh correction method. Before going further in the discussion, we now introduce the theoretical background which serves at the basis of this method.

Theoretical background

For the sake of simplicity, we consider a two-dimensional problem where the elements are triangles only.

Notations and definitions. A triangle K is an oriented triple of (distinct) integers. A mesh edge is a pair of (distinct) integers. The boundary of K, denoted by δK, is defined by the three edges of K. Now, let $\mathcal{T}$ be a set of triangles, we denote by $\mathcal{E}$ the set of the edges of the elements in $\mathcal{T}$, by $\mathcal{S}$ the set of the element vertices and by $\delta\mathcal{S}$ the set of vertices in $\delta\mathcal{T}$, the boundary of $\mathcal{T}$, i.e., both $\mathcal{S}$ and $\delta\mathcal{S}$ are a set of integers (or indices). With these notations, we first introduce the notion of a so-called *mesh topology.*

Definition 8.1 *A given mesh $\mathcal{T}$ conforms to a mesh topology if $\mathcal{E}$, the corresponding set of edges, is such that*

$$\forall e \in \mathcal{E}\,, 1 \leq card\{\, K \in \mathcal{T} \text{ such that } e \in \delta K\} \leq 2\,.$$

In this definition, $card\{.\}$ stands for the number of elements in set $\{.\}$. In other words, any edge in $\mathcal{E}$ is shared by two triangles (manifold case in three dimensions) or is a boundary edge.

Given a mesh topology $\mathcal{T}$, its boundary $\delta\mathcal{T}$ consists of the edges in $\mathcal{E}$ which belong to only one element.

Remark 8.1 *Unlike a mesh (as defined in Chapter 1), a mesh topology relies only on topological properties. In this respect a mesh can be defined as a mesh topology whose corresponding vertex coordinates result in a valid mesh (in the usual sense) while a mesh topology is valid only in terms of neighboring relationships.*

Following this remark, we can give a rough idea of the present mesh generation method. First, a mesh topology is constructed, then it is modified so as to produce a (valid) mesh. To this end, a local operator is introduced.

A simple local operator. Let $\mathcal{T}$ be a mesh topology and $\mathcal{S}$ the corresponding vertex set, let P be a point, then joining P with the edges of $\delta\mathcal{T}$ results in a new mesh topology as can be easily seen. In what follows, this operator will be referred to as $Op(P, \delta\mathcal{T})$. A more precise analysis of this operator leads to examining two situations:

- either $P \in \delta\mathcal{S}$ (point P is identified with index P),
- or $P \notin \delta\mathcal{S}$.

In the first situation, the operator basically joins the vertices in $\delta\mathcal{S}$, apart from P and its two immediate neighboring vertices, with P. In the second configuration, the operator leads to connecting all the vertices in $\delta\mathcal{S}$ with a given point P which could be inside or outside the region whose boundary is $\delta\mathcal{T}$. In Figure 8.8, one can see the first situation (cases iii) and iv)) as well as the second situation (cases i) and ii)).

Remark 8.2 *If $\mathcal{T}$ is a convex region, then applying $Op(P, \delta\mathcal{T})$ with P inside $\mathcal{T}$ or P is a member of $\delta\mathcal{T}$ results in a (valid) mesh following Definition (1.7).*

However, when $\mathcal{T}$ is an arbitrary mesh topology and not a valid mesh, applying $Op(P, \delta\mathcal{T})$ does not result, in general, in a (valid) mesh. The question is then to find some criteria to govern the application of the operator, e.g., how to repeat this single operator with appropriate points P so as to complete a (valid) mesh.

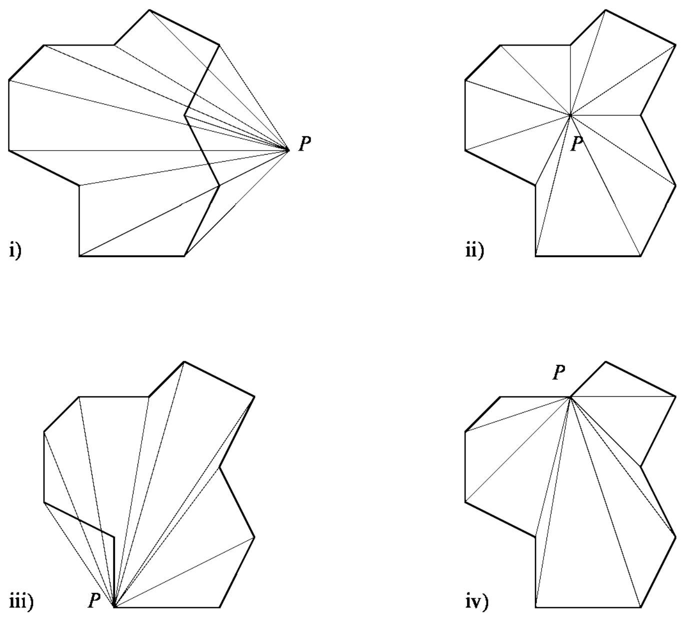


Figure 8.8: *Various applications of the $Op(P, \delta\mathcal{T})$ operator. The point P is outside $\mathcal{T}$, case i), P inside $\mathcal{T}$, case ii), P is a boundary point, cases iii) and iv).*

Two criteria are introduced: one is based on the element surface area while the other concerns the element shape quality.

Given $\mathcal{T}$, the surface area of this set of elements, $Sur(\mathcal{T})$, is the sum of the (non-oriented) surface areas of the triangles in $\mathcal{T}$. Similarly, the quality of $\mathcal{T}$, $Qua(\mathcal{T})$, is the quality of the worst element in this set. The quality measure is any of the measures discussed in Chapter 18 where the surface area involved in the expression is considered as positive (actually, we take the absolute value of the real surface area).

Some basic observations. A mesh topology $\mathcal{T}$ is not a valid mesh if, at least, one of its elements is negative, thus resulting in an overlapping region. Hence, the surface of $\mathcal{T}$ is larger than (or equal to) the surface of the corresponding domain. As a consequence, a mesh is a mesh topology whose surface area is minimum. This observation leads to the first idea to govern the operator $Op(P, \delta\mathcal{T})$: *minimize the surface area of the topology.*

A second observation is as follows. Given a mesh topology $\mathcal{T}$, we can apply the $Op(P, \delta\mathcal{T})$ operator for different points P and, in this way, various new mesh topologies can be constructed. In this situation, several new topologies may be equivalent in terms of surface areas. Thus, we need a criterion to decide whether a given result is better than another. This is why a quality-based criterion is introduced. Thus, the second key of the method is: *retain the transformation that minimizes the surface area and, at the same time, optimizes the quality criterion.*

Now that we have specified the basic ideas of the method, we can turn to the issue of a meshing algorithm based on these simple ideas.

Synthetic algorithm

In short, given a domain discretization, we initialize a mesh topology and then we modify it by repeated use of the above operator until a valid mesh is completed. To this end, the current mesh topology is analyzed and the regions with negative or badly shaped elements are identified. These regions are considered one after the other. Given a region, we define the corresponding (sub) mesh topology and we apply the operator Op to this (sub) mesh topology.

A natural (sub) topology is the *ball* of a given vertex (see Chapter 2), i.e., the set of elements that share this point. If P is a given vertex, then its ball is denoted by $\mathcal{B}_P$. In fact, any connected set of elements can be considered as a (sub) topology. For instance, the two triangles sharing a given edge is a natural candidate.

Given a mesh topology $\mathcal{T}$ and its boundary $\delta\mathcal{T}$, we define $\mathcal{V}$ the set of points P such that

$$\mathcal{V} = (P \in \mathcal{P}\ ,\ Sur(Op(P, \delta\mathcal{T})) = \min_{Q \in \mathcal{P}} Sur(Op(Q, \delta\mathcal{T})))\,, \tag{8.1}$$

in other words, $\mathcal{V}$ is the set of points (indices) that, after using Op, minimizes the surface area of the resulting mesh topology. In the above expression, $\mathcal{P} = \delta\mathcal{S} \cup G$ where G could be an arbitrary point (see below).

We also introduce $\mathcal{Q}$ the set of points P such that

$$\mathcal{Q} = (P \in \mathcal{P}\ ,\ Qua(Op(P, \delta\mathcal{T})) = \max_{Q \in \mathcal{P}} Qua(Op(Q, \delta\mathcal{T})))\,, \tag{8.2}$$

i.e., $\mathcal{Q}$ is the set of points (indices) that, after using Op, optimizes the quality of the resulting mesh topology.

Now, the mesh generation algorithm is as follows:

- Initialization: $\mathcal{T} = Op(P, \delta\mathcal{S})$ where P is the point in $\delta\mathcal{S}$ which is the nearest point to the centroid[4] of $\delta\mathcal{S}$.
- (A) $opt = .false.$
- (B) Loop over the points of $\mathcal{T}$, let P be the current point, then
  - consider the ball $\mathcal{B}_P$ as a sub mesh topology,
  - $\mathcal{P}$ is the set of vertices in $\delta\mathcal{B}_P$,
  - we consider a point G which is the centroid of the points in $\delta\mathcal{B}_P$, i.e., $G = \frac{\sum P_i}{n_P}$, where P_i denote a vertex in $\mathcal{T}_P$ other than P and n_P is the number of such vertices.
  - $\mathcal{P} = \mathcal{P} \cup \{G\}$,

[4]This is one possible candidate, in fact, any point in $\delta\mathcal{S}$ could be used.

- we apply $Op(P, \delta\mathcal{B}_P)$ where $P \in \mathcal{P}$ and we define the corresponding sets $\mathcal{V}$ and $\mathcal{Q}$,
- if $P \in \mathcal{Q} \cap \mathcal{V}$, return to (B),
- otherwise, pick the point P in $\mathcal{Q} \cap \mathcal{V}$ and retain the corresponding mesh topology, the current topology is now the following:

$$\mathcal{T} = \{\mathcal{T} - \mathcal{B}_P\} \cup Op(P, \delta\mathcal{B}_P)$$

fix $opt = .true.$ and return to (B).

- End of Loop
- if $opt = .true.$, then return to (A), otherwise the current state is stable.

On completion, a stable mesh topology is obtained. The issue is to make sure that this state corresponds to a valid mesh.

Since the topology is valid (due to the way it is constructed), the (global) mesh validity is ensured if the (signed) surface of all elements is strictly positive. In fact, if there is a triangle with a negative surface, then an overlapping element exists (a zero surface implies that there is a completely flat element while a very "small" surface, with regard to the edge lengths, indicates the existence of a bad quality element).

If the stable situation corresponds to a valid mesh (in the above sense), then the resulting mesh is a suitable mesh possibly after an optimization stage. If not, we have a stable situation which is invalid. This case deserves some comments. The stability is related to the choice of the sub-topology used in the method. In the case of a stable situation which is an invalid mesh, the only thing we can try is to modify the sub-topologies in order to make it possible to continue the process. In this way, the surface series still continues to decrease since the mesh is modified. As an example of other types of sub-topologies, we can consider the pairs of adjacent triangles (or some other patterns) instead of the balls used in the previous algorithm.

In theory, such a process converges since we face a decreasing series whose targeted bound (the right surface of the domain) is known in advance and because the number of combinations (sub-topologies) that can be constructed is a finite number. Thus, the convergence holds because, in the worst case, it is sufficient to examine all these possibilities (irrespective of the cost of such a method related to the underlying combinatorial aspect).

Computational issues

Various remarks about the computational aspects of the above algorithm can be made.

First, maintaining a (valid) topology relies on checking that the corresponding definition holds and, in addition, that the intersection of the initial topology except the examined ball (the sub-topology, in the case where these sets are the balls) with the sub-topology resulting from operator Op is nothing more than the boundary $\delta\mathcal{B}_P$ of the ball $\mathcal{B}_P$. In other words,

$$(\mathcal{T} - \mathcal{B}_P) \cap Op(P, \delta\mathcal{B}_P) = \delta\mathcal{B}_P \,.$$

Figure 8.9 illustrates a case where this result is not satisfied. Triangle ANB will be formed twice if the sub-topology $Op(N, \delta\mathcal{B}_P)$ is retained.

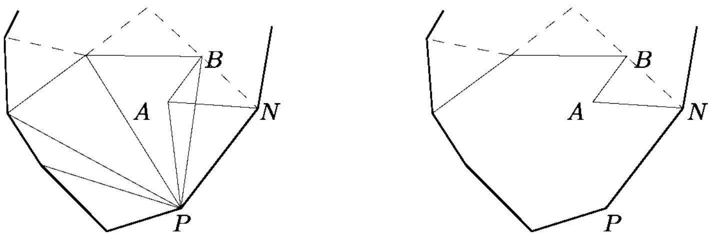


Figure 8.9: *Left-hand side: the initial sub-topology, right-hand side: joining point N with the other points in the boundary of the ball of P again results in triangle ANB which is already an element inside the ball in question.*

Leading on from these observations, it is of interest to define a suitable strategy regarding the way in which the operations are carried out and the choice of the sub-topologies used. For instance, it is proposed a first stage without any point insertion (by a pertinent choice of Op) and then to start again while point insertions may be used in this stage.

Extension in three dimensions

The above approach extends to three dimensions. In this case, the elements are only tetrahedra, i.e., an oriented quadruple of integers. The set $\mathcal{E}$ is the set of element faces and Definition (8.1) still holds (where e is a triangular face).

The operator $Op(P, \delta\mathcal{T})$ acts on the boundary of the mesh topology $\mathcal{T}$, meaning that point P is connected with the triangles that form this boundary. The two above criteria are also used, the first criterion concerns the volumes of the elements under consideration while the second remains unchanged.

The synthetic scheme of the method is as above while the sub-mesh topologies that can be dealt with include the balls of the current points, the shell of the current edges, i.e., (see Chapter 2) the tets that share an edge or, finally, any connected set of tets.

Application examples

As indicated at the beginning, two different applications of this meshing method can be envisaged. First, the method can be seen as a mesh construction method for a domain starting from a discretization of the boundaries of this domain.

Then, this method can be used as a mesh optimization method or even as a mesh correction procedure that applied to a mesh being modified during a computational process. Another more "exotic" application can be also envisaged that makes it possible to develop an algorithm resulting in the boundary enforcement (in the context of a Delaunay type method; see Chapter 7).

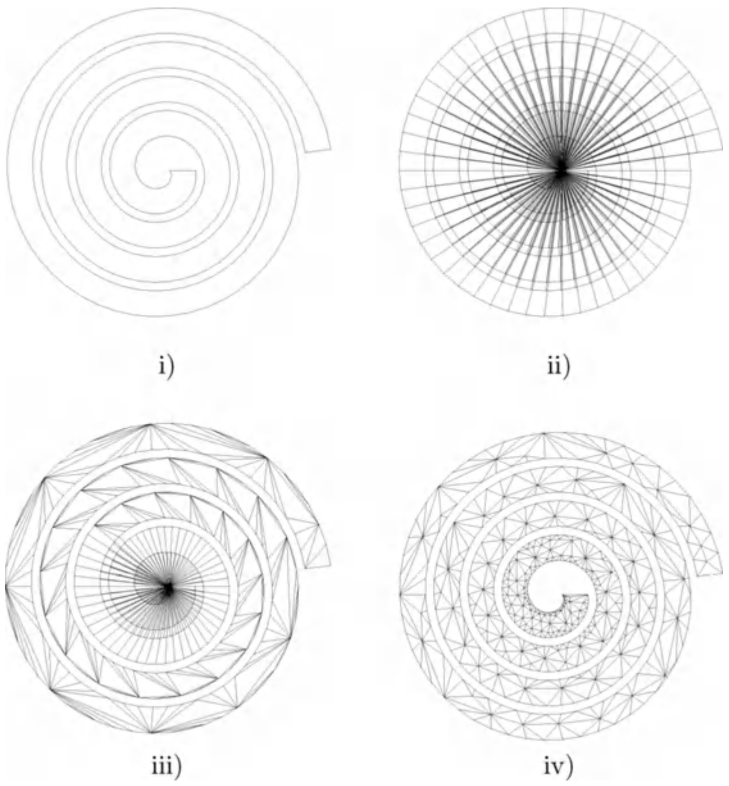


Figure 8.10: *i) the domain boundary, ii) the mesh after connecting one point (close to the domain centroid) with all the others, iii) the mesh during the optimization stage and iv) the resulting mesh. Indeed, the current topology is now a valid mesh.*

We now give some examples of meshes resulting from the present method. Figure 8.10 depicts several steps of a two-dimensional example. Figure 8.11 gives a tridimensional case. The geometry is a node, a kind of cylinder (or a pipe) that closes in on itself. Numerous examples, including some very impressive ones, can be found in [Coupez-1991], [Coupez-1996] and [Coupez-1995] concerning forming problems. An initial configuration whose geometry is rather simple is modified by stamping since a complex shape is obtained. The initial mesh is used as the initial topology and, due to the large deformations that are applied, this mesh becomes invalid. The method is then used to maintain a valid mesh at each state of the domain deformation.

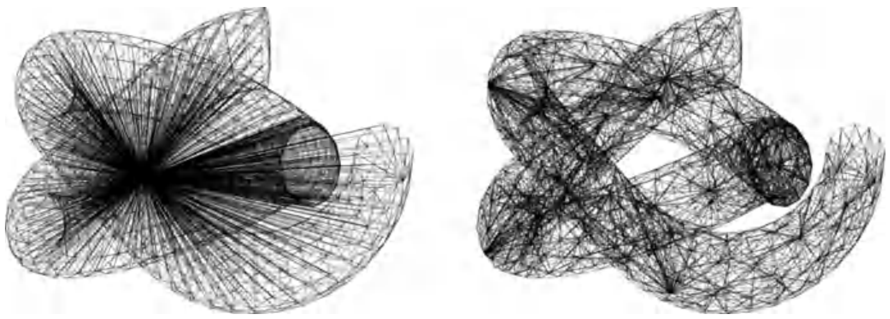


Figure 8.11: *The initial mesh topology resulting from the connexion of one boundary point with all the others (left-hand side) and the resulting mesh (right-hand side). Indeed, the current topology is now a true mesh.*

8.4 Quads by means of triangle combination

Constructing a quad mesh for an arbitrarily shaped domain (in Chapter 4, we saw the case where the domain geometry was suitable for a direct construction, either using an algebraic method or a PDE-based method or, finally, by means of a multiblock approach) is a tedious problem which can be handled in two ways. The first approach, a so-called *indirect approach* is based on using a triangular mesh and then creating the quad by triangle combination. The second approach, a *direct approach*, consists of designing a method that directly results in quads. In this section, the first approach is considered while the second approach will be discussed in the following section.

Before going further, we give two preliminary remarks. The first remark is a fundamental observation which concerns the existence of a solution to the problem in question. The second remark, of purely practical interest, mentions the expected cost of either approach.

Remark 8.3 *In general, the existence of a mesh composed only of quads is not guaranteed for an arbitrarily shaped domain. Indeed, provided with a polygonal discretization of a domain boundary, it is clear that, at least, an even number of edges is required if we want to obtain a solution. If not, a certain number of triangles (one for each connected component) may be necessary in order to cover the domain entirely.*

Remark 8.4 *The approach by means of combination starts from a triangular mesh (assumed to be conforming) of the domain and thus is nothing more than a post-processing of an existing mesh. As a consequence, the design and the validation (robustness, reliability) are only related to this processing that is, in principle, faster than validating an entire mesh construction method.*

Having made these remarks, we now describe an approach based on combination. We assume a favorable case (where the boundary includes an even number of segments) and we consider a domain in the plane (the surface case will be seen below).

Two basic combinations

The simplest pattern corresponds to two adjacent triangles that form a convex polygon. These two triangles make it possible to define one quad. A different pattern is formed by one triangle and its three neighbors. The corresponding polygon then has six sides. It allows, *a priori*, for the construction of three quads (a point being introduced, for instance, at the centroid of the initial triangle and then joined with the sides of the above polygon, thus giving the three quads in question). Figure 8.12 demonstrates these two examples of triangle combination.

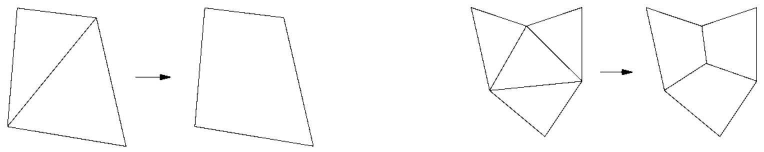


Figure 8.12: *Left-hand side: the combination of two triangles gives one quad; right-hand side, the combination concerns four triangles and results in three quads.*

Selection of a combination

Given one or several combination patterns, the question is to define the pairs (or the sets) of triangles that must be combined to define the quads. The basic technique consists of finding in the triangular mesh a path that makes it possible to pass from one triangle to the other. This path, obtained by visiting the triangles in some way, is stored in a stack. Thus, the triangles are put in a stack and pushed once the quads are constructed.

The construction of the stack (the path) follows some rules. A starting triangle is selected (having two boundary edges if such an element is found). This triangle is the first in the stack.

The current triangle is colored and one of its edges is selected. The neighbor (through this edge) is put into the stack if it is not already colored and if the quad thus formed is valid (actually, this quad is not really constructed). The stacked triangle is the *child* of the initial triangle. When it is no longer possible to continue, the triangles are removed from the stack. When carrying out such an operation we encounter two situations.

There is an edge which has not yet been used. The corresponding neighbor is put into the stack as the child of the related triangle, otherwise, the quad construction begins.

Let us assume that a certain number of triangles has been put into the stack and that some quads exist. Suppose we are at triangle K_1 while going from its neighbor after having passed through edge e_1 of triangle K_1 (e_1 is then the edge common to the two triangles in question). We try to pass through the edge e_2, edge next to e_1 while in the direct sense. This is possible if, on the one hand, edge e_2 is not a boundary edge and, on the other hand, the triangle neighbor of K_1 through this edge has not already been processed. If the edge cannot be traversed, we consider the next (using the same convention), in this way we find one triangle, say K_2, or this branch of the stack is no longer active (the edge is a boundary edge or the neighboring element is already in the stack). In the first case, K_2 is put into the stack and we report that this element follows K_1. In the second case, we count the number of *children* of K_1 not already involved in a quad. The children of a given triangle are its neighbors that have not already been taken into account. Therefore, a given element may have zero, one or two children.

If K_i is the last triangle in the stack, we use the following algorithm:

- if the number of children of K_i is zero, we remove K_i from the stack and we return to triangle K_{i-1},

- if the number of children of K_i is one and if the triangle is still *active* (i.e., has not been used to construct a quad), we construct a quad by combining K_i and its child while verifying that this element is convex. K_i is then removed from the stack. If the combination of K_i and its child forms a non-convex polygon (or a quad with two consecutive aligned sides), the triangle child will remain and we try to form a quad using K_i and its father if this pattern is adequate,

- if the number of children of K_i is two and if the triangle is still active, we form three quads from K_i and its three neighbors (its two children and its parent), then K_i is removed from the stack.

This algorithm continues as long as the stack is not empty. Once it is empty, the quad mesh is obtained. Nevertheless, some isolated triangles can be present in this mesh. Thus, the resulting mesh is a *mixed* mesh.

Remark 8.5 *The thus-constructed path may not pass through all the triangles in the domain. In such a case, we start again from one triangle not already visited and one path by the thus-defined component can be obtained.*

A solution resulting in a mesh without any triangles, consists of applying the combination strategy on triangles whose size is twice the desired size (in terms of edge length) and then splitting the resulting quad into four elements. The triangles that can still exist are subdivided into three by introducing centroids and edge midpoints (if the edge is a boundary edge, we have to consider the geometry of this boundary to locate the midpoint). In this way, the resulting mesh has only quads and, due to the method, is a conforming mesh.

Optimization

The resulting meshes, in most cases, are not of good quality everywhere in the domain. Adding strict criteria about this quality during the process is possible *a priori* but impedes the quad creation. In such a case, numerous triangles necessarily exist since the corresponding combinations are rejected. Therefore, it is advisable to leave some degree of freedom in the construction and, since, bad quality elements may appear, to apply an optimization stage *a posteriori*. The optimization tools are the local tools capable of carrying out local configurations (a set of adjacent elements). Chapter 18 contains a detailed description of general purpose optimization tools, so, here, we will just give some brief indications about tools acting on quad elements. Among these tools, we can find some operators related to certain typical configurations:

- an internal vertex only common to two quads or common to one quad and one triangle can be removed. We then successively obtain one quad or one triangle whose quality is necessarily better than that of the initial configuration,
- a vertex shared by three quads is in general related to a bad quality quad. The initial pattern can then be replaced by two quads which are better *a priori*,
- a quad with two (internal) opposite vertices common to only three elements can be removed (the two points are merged and the quad disappears),
- the edge common to two quads can be removed by using the alternative connexion with the two other vertices (we find here the edge swapping operator described in Chapter 18),
- a set of adjacent quads forming a six side polygon may be replaced using only two quads, etc.

An optimization operator is retained if the resulting quality is better (the initial context is analyzed according to a quality criterion or a topological criterion). Note that the relaxation, on average, of the degree of the internal vertices (see again Chapter 18) is a promising operation for this type of mesh.

Alternate method

In this approach, we examine, for all triangles in the initial mesh, the quality of the quad formed by combining the triangle in question, K, with its neighbors, K_i. Therefore, there are *a priori* three possibilities. Each possible quad is identified by the edge common to K and K_i. With each of these edges is associated a quality value which involves the angle in the quad at the two vertex endpoints of the edge in question.

Let $[ABCD]$ be a quad whose vertices are defined in the direct sense. In the classical isotropic case, the quality measure of a quad is defined at vertex P by means of the formula:

$$Q_P = \begin{cases} \dfrac{2\theta}{\pi} & \text{if} \quad 0 \leq \theta < \frac{\pi}{2} \\ 2 - \dfrac{2\theta}{\pi} & \text{if} \quad \frac{\pi}{2} \leq \theta < \pi \\ 0 & \text{if} \quad \pi \leq \theta \end{cases}, \tag{8.3}$$

where θ is the angle between three consecutive vertices, P_iPP_j, in the quad. This function varies linearly between 0 (flat quad) and 1 (rectangle). The quad quality is then defined as the minimum value of the quality values in its vertices:

$$\mathcal{Q} = \min_{P\in\{A,B,C,D\}} \mathcal{Q}_P \,. \tag{8.4}$$

This quality measure can be extended to the anisotropic case. To this end, it is just necessary to define the dot product with regard to a given metric.

Dot product in a given metric. The dot product of two vectors $\vec{u}$ and $\vec{v}$ in the Euclidean space characterized by a metric $\mathcal{M}_2(X)$ is defined as:

$$\langle \vec{u}, \vec{u} \rangle_{\mathcal{M}_2(X)} =^t \vec{u}\mathcal{M}_2(X)\vec{v} \,.$$

The norm of a vector w is then given by:

$$\|\vec{w}\|_{\mathcal{M}_2(X)} = \sqrt{^t\vec{w}\mathcal{M}_2(X)\vec{w}} \,.$$

Angle measurement in a metric. Let A, B and C be three points such that $\overrightarrow{BC} \wedge \overrightarrow{BA} \geq 0$, the notation $\wedge$ stands for the cross product in the classical Euclidean metric. The value of the angle (in radians) between the vectors $\overrightarrow{BC}$ and $\overrightarrow{BA}$ with respect to the metric $\mathcal{M}_2(X)$ is given by:

$$\theta_{B\mathcal{M}_2(X)} = \arccos\left(\frac{\langle \overrightarrow{BC}, \overrightarrow{BA} \rangle_{\mathcal{M}_2(X)}}{\|\overrightarrow{BC}\|_{\mathcal{M}_2(X)} \|\overrightarrow{BA}\|_{\mathcal{M}_2(X)}} \right) . \tag{8.5}$$

Quad quality in a metric. The quality function defined in the isotropic case (Relationship (8.3)) can be extended to an anisotropic metric in the following way:

$$(\mathcal{Q}_P)_{\mathcal{M}_2(X)} = \begin{cases} \dfrac{2\theta_{\mathcal{M}_2(X)}}{\pi} & \text{if} \quad 0 \leq \theta_{\mathcal{M}_2(X)} < \frac{\pi}{2} \\ 2 - \dfrac{2\theta_{\mathcal{M}_2(X)}}{\pi} & \text{if} \quad \frac{\pi}{2} \leq \theta_{\mathcal{M}_2(X)} < \pi \\ 0 & \text{if} \quad \pi \leq \theta_{\mathcal{M}_2(X)} \end{cases} ,$$

and, thus:

$$\mathcal{Q} = \min_{(X,P)\in\{A,B,C,D\}} (\mathcal{Q}_P)_{\mathcal{M}_2(X)} \,. \tag{8.6}$$

Remark 8.6 *The quad quality computation in a Riemannian case (anisotropic metric) requires evaluating 16 quality values in the Euclidean case.*

A simplified quality function can be used. If $ABCD$ is a quad, its quality can be defined as the value of an angle associated with the edge AC (common to the two triangles) as:

$$\mathcal{Q}_{\mathcal{M}_2(X)} = \min\left([\theta_A]_{\mathcal{M}_2(A)}, [\theta_A]_{\mathcal{M}_2(C)}, [\theta_C]_{\mathcal{M}_2(A)}, [\theta_C]_{\mathcal{M}_2(C)}\right) . \tag{8.7}$$

General scheme. The edges are sorted based on the decreasing quality value. The quads are formed by combining the triangles following this order. Once a quad is constructed, the corresponding edge is removed from the list of possible edges. A data structure such as a *heap* is used to store the triangles that remain to be combined. Indeed, at each quad creation, the list of the edges to be dealt with must be updated.

With this strategy, the quads that are constructed are, in some senses, the best possible. However, the number of isolated triangles is not minimal. Several ideas can be advocated so as to minimize this number of triangles or to avoid them altogether. On completion, there are still some triangles that have not been combined. These elements are then subdivided into three quads after introducing the edge midpoints and the centroids. This leads to propagating the refinement to some adjacent elements (cf. Figure 8.13). In such a case, the size of the elements in the final quad mesh is half the size of the triangles in the initial mesh (assumed to conform to a given size map). To solve this problem, we can use a technique based on the *twice size* (as previously discussed).

Regarding the way in which the triangles to be combined are governed by the quality of the resulting quads, two particular drawbacks must be avoided:

- combining a triangle with some edges aligned with one of a neighbor,
- leaving too many isolated triangles. Indeed, the vertex introduced at the centroid of the element is of degree three and thus rigidifies the mesh, thus making the quality optimization more delicate.

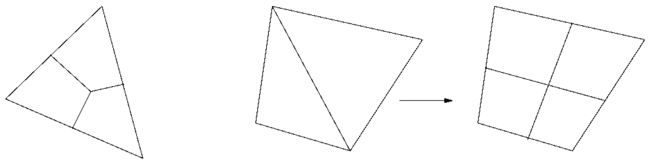


Figure 8.13: *Triangle combination so as to form quads. Left-hand side: an isolated triangle is subdivided into three quads. Right-hand side: the quad resulting from the combination of two triangles is split into four quads.*

Remark 8.7 *In the isotropic case, it is possible to use a different quality measure (see Chapter 18):*

$$\mathcal{Q} = \alpha \frac{S_{min}}{h_{max}\sqrt{\sum_{i=1,4} d_i^2}}\,, \tag{8.8}$$

where h_{max} is the largest length of the sides and the two diagonals, d_i is the distance between any two points in the quad (i.e., sides or diagonals), S_{min} is the minimum of the four (triangle) surfaces that can be associated with the quad and α is a normalization factor such that the square quality value is one.

Remark 8.8 *A procedure which is easy to implement, based on a frontal type propagation, can also be used to combine the triangles. Given a triangle, the triangles adjacent to an edge are examined and combined in this order (the list of triangles to be processed is no longer sorted according to a quality criterion, thus simplifying the algorithm).*

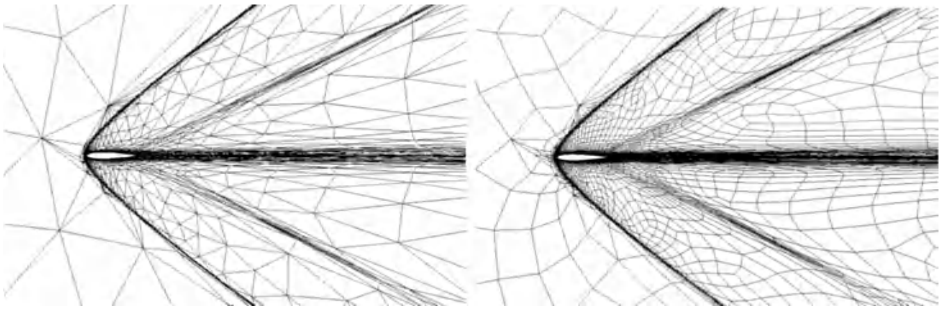


Figure 8.14: *Example of a quad mesh by triangle combination. Left-hand side, initial anisotropic triangular mesh. Right-hand side, resulting quad mesh. Note that the size of the quads is half that of the triangles.*

Dealing with a surface

Using a triangular surface mesh, it is also possible to obtain a quad mesh of this surface. The method is rather similar to that previously described.

In this approach, we assume a discrete parametric surface provided in the form $\sigma(u, v) = (x, y, z)$, the parameters u, v being defined in a rectangle. The metric $\mathcal{M}_2$ introduced in the general case corresponds here to the (anisotropic) metric of the principal radii of curvature or to the (isotropic) metric of the minimal radii of curvature, these metrics being defined in the parametric space. The definition of these different metric maps is described in Chapter 13.

8.5 Quads by means of a direct method

Unlike the previous schemes, various direct methods can be envisaged to obtain a quad mesh (without the temporary help of a triangular mesh). Three of these have received some attention:

- an advancing-front type method,
- a special understanding of the above grid superposition type method,
- a domain decomposition combined with an algebraic method in each subdomain.

Advancing-front type method

This type of method consists of generating the domain *paving* by going through the domain starting from its boundaries (in this respect, we return to the advancing front type methods as seen in Chapter 6) [Blacker, Stephenson-1991].

More recently, [Cass *et al.* 1996] advocated the use of this approach in the case of parametric surfaces while using the tangent planes together with the radii of curvature.

Another method in this class makes use of the STC, the *spatial twist continuum*, [Murdoch, Benzley-1995]. Given a quad mesh, the STC can be constructed. This structure is made of chords which link the quads. It can be seen as the dual of the mesh (in some sense, this structure is what the Voronoï diagram is for a triangular mesh). Conversely, given a STC, say a series of chords inside the domain which follow some rules, it is possible to define a quad mesh.

Remark 8.9 *In essence, this approach is sensitive to the domain (surface) boundary discretization. In addition, the number of segments in the boundary mesh must be even.*

Grid Superposition

Here, we follow an idea close to that seen in the section devoted to mesh construction by means of a grid (a set of quad or square boxes) or by using predefined patterns (quads in this context). Figures 8.6 and 8.7 give one academic example of the aspect of the meshes completed by using such an approach.

Using the medial axis

In Chapter 9, we will see the precise definition of the medial axis of a domain in two dimensions together with a method suitable for such a construction (actually, a discrete approximation of this line). Briefly, we assume the data of a discretization of the domain boundaries and an approximation of its medial axis. This information can then be used to subdivide the domain into regions with a simple geometry. These regions are constructed in such a way that certain properties hold, ensuring it is possible to cover them by means of quads. Depending on the case, the mesh is completed by using an algebraic type method or by following a midpoint subdivision method.

Domain partitioning. The domain's boundaries and its medial axis make it possible to define a domain partition composed of a certain number of regions that are bounded by a portion of this line, one of several portions of the boundary and one or several "cuts" joining these two categories of sides.

Mid-point subdivision. This technique makes it possible to cover a polygon by means of quadrilateral regions. An internal point is constructed, for instance, the centroid G of the initial polygon then the edge midpoints are introduced (for the edge boundary of the polygon). It is then sufficient to join point G with two consecutive midpoints to obtain the quad regions that are sought.

The regions resulting from the domain partition are dealt with in this way and a first quad covering-up is thus obtained.

Quad construction. A repeated use of the above algorithm results in the final mesh completion. Another technique uses an algebraic method (as in Chapter 4) in each region while ensuring the conformity of the output mesh (we encounter again the consistency constraints of the multiblock methods about the way in which the subdivision parameters propagate from one region to another).

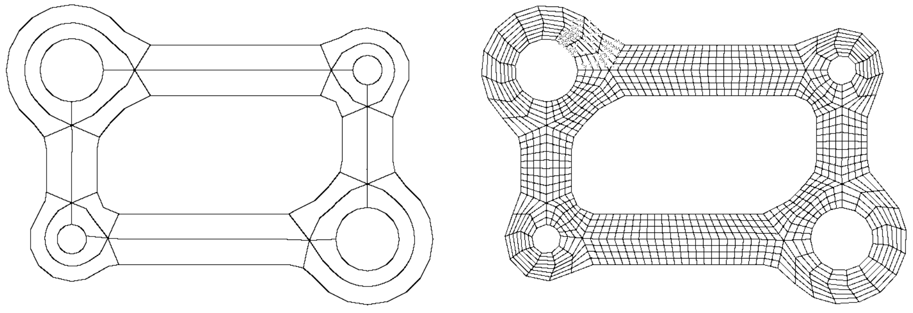


Figure 8.15: *Partitioning method based on the medial axis: approximate skeleton of the domain and cut lines (left-hand side), quad mesh resulting from this partition (right-hand side).*

8.6 Hex meshing

We consider here only the case of an arbitrarily shaped domain. Indeed, if the domain geometry is adequate, we already know some methods capable of constructing a hex mesh (algebraic method, PDE-based method, product method, etc.). First, it could be observed that the triangle combination method for quad meshing in two dimensions does not extend to three dimensions as it is tedious to define a tet combination resulting in a hex that uses all or, at least, the majority of the elements in the mesh.

On the other hand, splitting the elements of a given tetrahedral mesh by means of 4 hexes results in poorly shaped elements and bad connectivities (or vertex valence; see Chapter 18).

Thus, a direct construction method must be considered and, in principle, we return to the three methods that have been introduced for quad meshing purposes in two dimensions, the advancing-front method, grid-based method or use of midsurface:

- advancing-front based method. We encounter here the principles of any classical advancing-front method (Chapter 6). Nevertheless, the expected

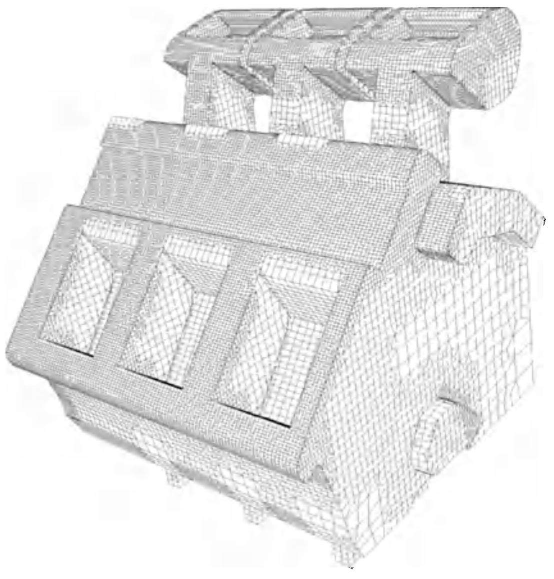


Figure 8.16: *Mechanical part automaticaly meshed with Hexotic. The mesh is made up of 152,719 elements (96.8% hexes and 3.2% pyramids). Data Courtesy of Part Solution.*

difficulties are now more critical. Indeed, it is tedious, given a face, to find the various points candidate for hex creation. Several techniques have been proposed, [Blacker, Meyers-1993], [Murdoch, Benzley-1995] and also [Folwell, Mitchell-1998], which, in some cases, introduce some non-hexahedral elements (pyramids, prisms, tets), thus leading to *mixed* meshes;

- grid-based method. The chosen grid could be uniform or hierarchical (an octree, or even an octree whose cells are subdivided into 27 octants (and not 8 as in the classical case)). The main difficulty then consists of dealing properly with the octants that intersect the boundary or that are close to it. The main references in this area are [Schneiders *et al.* 1996] and [Schneiders-1996a]. Note that full hex meshes are tedious to obtain while hex dominant meshes offer more flexibility, [Marechal-2001]; see Figure 8.16;

- method using the midsurface. A midsurface (approximated or having the same topology as the exact midsurface) is constructed (see Chapter 9). It then serves to partition the domain in terms of regions whose topology is simpler so that a direct method can be envisaged, [Price *et al.* 1995], [Price, Armstrong-1997].

8.7 Miscellaneous

In this section, we mention some methods which can be suitable in certain situations and which have not been covered in the previous discussion.

Radial meshes

A radial mesh can be a source of benefit for some numerical simulations that can account with this specificity. Radial meshes can be developed based on various methods. For instance, a product type method is a natural candidate for such a result. An alternative method is the pattern based mesh generation method as briefly discussed above. Using a classical method (such as an advancing-front or a Delaunay-based method) with pre-placed vertices may also be a solution.

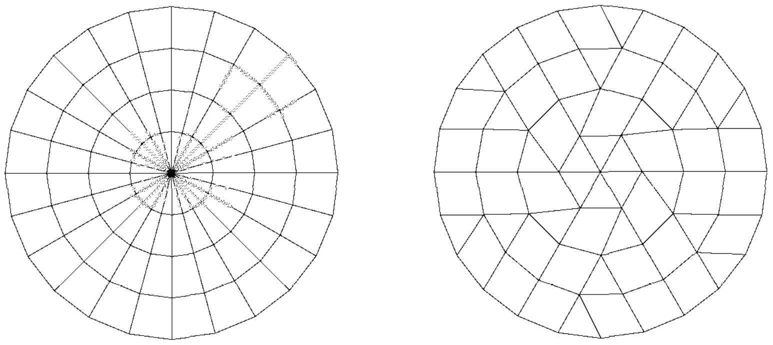


Figure 8.17: *Classical radial mesh as completed by a product method (left-hand side) and non-classical radial mesh as resulting from a series of vertex and edge collapsing (right-hand side).*

In Figure 8.17 (left), we can see the classical result obtained by a product type method where the data consists of a discretized segment and a generation line that reduces to a rotation. We should note that the mesh elements near the center of the domain are very badly shaped due to the rigidity of the method which imposes a constant number of subdivisions along each section. Thus, in Figure 8.17 (right), we can see the mesh obtained using a variation developed by [Hecht, Saltel-1989] which consists of balancing the element sizes (note that some triangles remain in the resulting mesh). In short, the classical product mesh is modified by means of edge and vertex collapsing (see Chapter 18) so as to prevent the construction of elements that are too small. In this way, the number of subdivisions may vary from one section to another, thereby reducing, to some extent, the rigidity of the classical approach.

Recursive partition

The key idea is to make use of the "divide and conquer" paradigm (see Chapter 2) to construct the mesh of a domain using its boundary discretization as input data.

Given a domain by means of its boundary discretization (a series of polygonal segments in two dimensions, or a triangular surface mesh in three dimensions), a mesh generation method can be designed based on a repeated partition into two sub-domains of the current domain. Once a polygonal (polyhedral) sub-domain is composed of only three (or a small number of) segments or of four (or a small number of) triangular faces, it is then possible to mesh it by means of one or a small number of triangles (tets).

Let us restrict ourselves to a two-dimensional problem. We consider the polygonal line serving as boundary discretization. A boundary vertex is selected and a line passing through this point is constructed in such a way as to separate the domain into two sub-domains. This line is furthermore subdivided by means of segments and, thus, two polygonal contours are defined. Thus, the initial meshing problem is replaced by two similar sub-problems. When a polygonal contour reduces to three (four or a meshable pattern), the subdivision is no longer pursued.

From a practical point of view, several issues must be carefully addressed including:

- the choice of a candidate vertex, the basis of the separating line,
- this line by itself and its appropriate subdivision so as to reflect the initial boundary stepsizes,
- the determination of meshable patterns.

Finding a separating line leads to selecting two points on the boundary such that the line whose extremities are these points is fully inside the domain with such a boundary. Therefore, we find an intersection problem that is, on the one hand, very sensitive to numerical errors (round-off errors) and, on the other hand, one which could be time-consuming. Then, provided with such a line, it is generally necessary to subdivide it into segments so as to reflect the discretization of the initial boundary. A technique, close to that used when splitting the edges when creating the internal points in a Delaunay type method (see Chapter 7), is then a possible solution. A sizing value is associated with the line endpoints, the length of this line is computed and, based on the desired point distribution, the line is subdivided into several adequately sized segments. Notice, in passing, that the separatrice lines will be present in the final mesh (as they stand or slightly modified if a point relocation step is performed with a view to optimization).

Remark 8.10 *The data of a size map (i.e., a control space) serves as input information that makes the above subdivision possible in accordance with some size specifications.*

A few remarks about this type of approach may be given. The choice of the separators strongly affects the resulting solution and a strategy must be defined to ensure a nice solution. In this respect, priorities in the selection of the basic points of the partition can be introduced (in a way, we return to some ideas close to those used in an advancing-front method, using the local aspects (such as angles), partition balancing, etc.).

In three dimensions, the same principle applies, at least formally speaking. However, the numerical problems are relatively much greater, in particular those related to intersection problems (the separatrice surface must remain inside the domain). Also, preventing the construction of ill-shaped elements (flat tets) and the definition of non-meshable regions (such as the Schönhardt polyhedron) must be carefully considered.

In conclusion, this type of method, which is widely used in some commercial software packages, is a solution that enables automatic mesh construction for complex domains. Nevertheless, various numerical troubles may be expected and, in addition, the complexity (in terms of CPU cost) is far from easy to evaluate *a priori.*

★
★ ★

Do methods other than those covered in the previous chapters and more briefly in the present chapter exist? The answer is not so clear. It is likely that solutions do exist for some particular configurations and the answer would certainly be yes again if we consider some applications related to a "domain" other than the finite element domain (computer graphics, physical analogy, image processing, etc.), while observing that some of these methods developed in a specific context (spring analogy, molecular dynamics, simulated annealing, genetic algorithms, etc.) can be seen as occurrences of some previously known methods (although possibly under a different name) or as variations of some "classical" methods. It is then necessary to remain attentive and open-minded in such judgments while retaining a (evenly positive) critical view about any *a priori* new or novel method or any method presented as a new one.

Chapter 9

Delaunay Admissibility, Medial Axis and Applications

Mid-surface or *medial surface* (*medial axis* in two dimensions) construction for a given domain is a very promising field of research that has multiple applications such as domain simplification, spatial dimension reduction and also as a first step to designing an algorithm for quad mesh generation. Similarly, the mid-surface associated with a three dimensional domain can serve the same purposes.

Several methods can be advocated to construct the medial entity of a domain, also referred to as the domain *skeleton* or *neutral fiber.* A brief survey of the possible methods can be found in [Turkiyyah *et al.* 1997]. In this respect, quadtree-octree based methods, tracing algorithms, adequate PDE solutions as well as Voronoï-based methods can be considered. Among these methods, we focus here on the last approach. Since this method considers the Voronoï cells of a given domain, it can be based on Delaunay triangulation of the domain. This opens up the discussion as to what is termed a *Delaunay admissible* boundary discretization (a polygon in two dimensions, a polyhedral simplicial surface in three dimensions).

Based on such a boundary discretization, it is possible to complete an "empty" Delaunay triangulation[1] of the domain of interest. From that, algorithms can be develope, resulting in the construction of the corresponding medial entity (actually, a discrete approximation of the entity). Thus, before considering medial entity construction and the various applications that can be derived from it, the first sections focus on Delaunay admissible boundary discretization.

★
★ ★

The case of Delaunay admissible edges in a two-dimensional space is considered, then we turn to the same edge problem in three dimensions before going on to the

[1]This triangulation is termed empty in the sense that it does not include any vertex inside the domain. The sole element vertices are those serving at the boundary discretization.

case of (triangular) faces. We then discuss the construction of the desired medial entity and, finally, we briefly introduce various applications.

9.1 Delaunay-admissible set of segments in $\mathbb{R}^2$

Let $\mathcal{S}$ be a set of points[2] in $\mathbb{R}^2$ and let $\mathcal{F}$ be a set of segments whose endpoints are members of $\mathcal{S}$. The question is to make sure that applying a Delaunay triangulation algorithm (see Chapter 7) to $\mathcal{S}$ results in a mesh where the segments of $\mathcal{F}$ exist as element edges. In other words, given any segment f in $\mathcal{F}$ whose endpoints are A and B, we wish to have f a triangle edge at the time the points in $\mathcal{S}$ (in particular A and B) have been inserted by the Delaunay triangulation method.

Remark 9.1 *Actually, f will be formed if there exists a point P such that a Delaunay triangle fP exists. Another naive equivalent condition is that there is no pair of points, P and Q, located each on one "side" of f such that edge PQ will be formed that intersects f.*

If $\mathcal{F}$ is such that the previous property holds for all of its members, then $\mathcal{F}$ is said to be *Delaunay-admissible* (or *Delaunay-conforming*) and, for the time being, we just retain this naive definition for this notion.

Now, given $\mathcal{F}$ a set of segments, it is not guaranteed that $\mathcal{F}$ is Delaunay admissible. The problem is then to characterize this property, to find the condition(s) the segments in $\mathcal{F}$ must have to meet it and, in the case where $\mathcal{F}$ is not Delaunay admissible, to modify $\mathcal{F}$ in such a way as to obtain a new set of segments which conforms to the desired property[3].

Terminology and notations

Before going further, we introduce some notations (see Figure 9.1 and Table 9.1). Given $\mathcal{F}$, a set of segments, $f = AB$ denotes a segment of this set, $\mathcal{S}$ stands for the set of the endpoints of the fs and $\mathcal{L}_f$ denotes the line supporting f, this line defines two subspaces, $\mathcal{H}_f^+$ and $\mathcal{H}_f^-$. Given f, B_f^{min} is the open ball whose diameter is f (this ball corresponds to the smallest circle that can be constructed passing through the endpoints of f). Finally, given a triangle, say a triple ABC or a triple like fM where f is a segment (an edge) and M is a point, B_{ABC} (resp. B_{fM}) is the open ball circumscribing the triangle ABC (resp. fM).

[2]The points are not assumed to be in general position. Indeed, in what follows, a co-circular pattern, possibly after a swap, does not lead to difficulty.

[3]A precise definition of the notion of a Delaunay admissibility of an set of $k-$faces can be found in the *ad-hoc* literature and, for instance, in [George, Borouchaki-1997]. Such a theoretical definition involves the Voronoï cells, dual of the Delaunay triangulation. Actually, using such a duality is not strictly required in two dimensions if we want to prove some theoretical issues (see below). Nevertheless, this argument will be reviewed later since it leads to a rather elegant proof when the dual problem (Voronoï) is easier to solve than the primal problem (Delaunay).

f or AB	*segment in question*
$\mathcal{P}roj_f(P)$	*projection of point P on f*
$\mathcal{L}_f$	*line support of f*
$\mathcal{H}_f^+$, $\mathcal{H}_f^-$	*two half-planes related to $\mathcal{L}_f$*
B_f^{min}, B_{ABC}, B_{fP}	*small ball of f, circle circumscribing $ABC(fP)$*

Table 9.1: Notations for the entities involved in the two-dimensional edge problem.

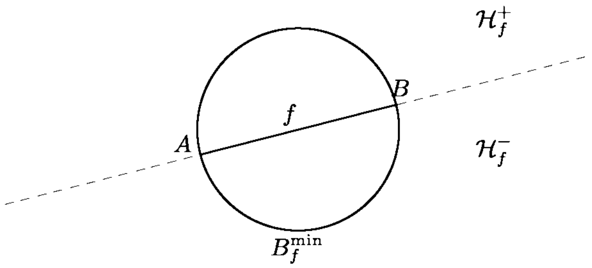


Figure 9.1: *A segment $f = AB$, member of $\mathcal{F}$, the corresponding ball B_f^{min} and the two associated half-spaces $\mathcal{H}_f^+$ and $\mathcal{H}_f^-$.*

Classification

Given a set $\mathcal{F}$ (and the corresponding set $\mathcal{S}$ where, to make sense, we assume that more than three non-aligned points exist), we need to analyze the configurations associated with all any f in $\mathcal{F}$ and, depending on the case, to see whether or not $\mathcal{F}$ is suitable or must be modified. This analysis concerns the various configurations that can be encountered. Let us consider an edge f, then the following cases can be found [Pebay-1998b]:

Case 0: $\mathcal{H}_f^+$ (or $\mathcal{H}_f^-$) is empty,

Case 1: B_f^{min} is empty,

Case 2: B_f^{min} contains one or several points (other than A and B, remember that we are considering open sets).

Analysis of the three configurations

Obviously, a segment f falling within *(Case 0)* will be formed and then retrieved as an element edge. So it is for a segment f corresponding to *(Case 1)* as proved in what follows. Let A and B be the endpoints of segment f. Then, a condition to have AB an element edge is that there exists a point P in $\mathcal{S}$ such that $B_{fP} = \emptyset$. Since such a point exists, a segment in *(Case 1)* is Delaunay.

Proof. Let P be an arbitrary point in $\mathcal{S}$, for instance, in $\mathcal{H}_f^-$. Since $P \notin B_f^{min}$, the circumball B_{fP} is such that $B_{fP} \cap \mathcal{H}_f^+ \subset B_f^{min}$ then $B_{fP} \cap \mathcal{H}_f^+ = \emptyset$. If $B_{fP} \cap \mathcal{H}_f^- = \emptyset$, then $B_{fP} = \emptyset$ and AB is Delaunay. Otherwise, at least one point, say Q, exists in $B_{fP} \cap \mathcal{H}_f^-$. Replace P by Q and repeat the same process (i.e., return to "since"). At completion we have found a point P such that $B_{fP} = \emptyset$ and thus AB is Delaunay admissible. Actually, the solution point is that for which angle $\widehat{APB}$ is maximal. □

Note that, given f, the condition "B_f^{min} is empty" is a *sufficient condition* of Delaunay admissibility which, in fact, is too demanding. We meet here the notion of a *protecting ball* which is, surprisingly, very popular in some disciplines while being, as indicated, too restrictive (e.g. omitting considering *(Case 2))* .

Now, we turn to *(Case 2.)*. In this case, one or more points are located in B_f^{min}. Let P be the point such that angle $\widehat{PAB}$ is maximal. Then,

- if B_{fP} is empty, triangle fP is Delaunay and thus will be formed (and f will be formed),
- otherwise, one or several points exist in B_{fP} and triangle fP is not Delaunay, meaning that f is not Delaunay.

We then have a condition of Delaunay admissibility for f. It involves the point in set $\mathcal{S}$ with maximal angle and the simple examination of the Delaunay criterion for the corresponding triangle makes it possible to decide whether or not f is Delaunay.

Delaunay-admissibility and Voronoï cells. The Delaunay admissibility of a given segment in $\mathbb{R}^2$ can be also easily expressed in terms of the properties the corresponding Voronoï cells must have.

Thus, following Chapter 7 and more precisely Relationship (7.1), the Voronoï cell associated with a point A is the region V_A such that:

$$V_A = \{M \quad \text{such that} \quad d(M, A) \leq d(M, N), \quad \forall N \in \mathcal{S}\} \tag{9.1}$$

where $d(.,.)$ is the usual distance between two points. Now, consider four points in $\mathcal{S}$, the two endpoints of segment $f = AB$ under consideration and two other arbitrary (non-aligned with the previous) points, P and Q, that can impede the construction of AB. Four Voronoï cells can be defined, successively V_A, V_B, V_P and V_Q. It is obvious that these four regions fall within one of the three situations depicted in Figure 9.2.

Now, $f = AB$ is Delaunay-admissible if edge AB exists in the Delaunay triangulation. In terms of Voronoï cells, AB is a Delaunay edge if V_A and V_B share a (Voronoï) edge or a (Voronoï) vertex[4]. Thus, AB will be formed in case i), in case ii) and not in case iii), as displayed in Figure 9.2. Thus, we have a condition about the Voronoï cells that allows us to see whether or not a given segment is Delaunay.

[4]In such a case, the four points under consideration for the local analysis are co-circular and AB can be easily obtained (at least after a swap; see Chapter 18).

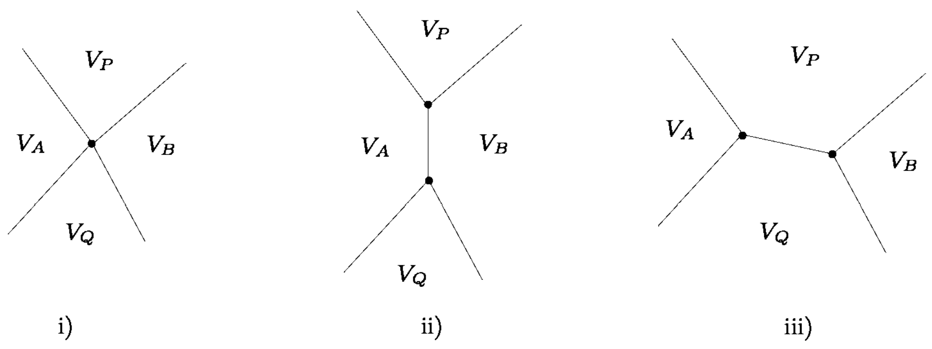


Figure 9.2: *The three possible patterns that can be encountered when considering the Voronoï cells related to four (non-aligned) points in* $\mathbb{R}^2$.

Exercise 9.1 *Write the fact that V_A and V_B share an entity in terms of properties about the balls involved in the context. On the fly, retrieve the above conditions by showing that*

$$V_A \cap V_B \cap AB \neq \emptyset \iff B_f^{min} \textit{ empty},$$

$$V_A \cap V_B \neq \emptyset \textit{ and } V_A \cap V_B \cap AB = \emptyset \iff B_{fP} \textit{ empty},$$

where P is such that angle $\widehat{APB}$ is maximal.

Construction of a Delaunay-admissible set of edges

Given $\mathcal{F}$ a set of segments in $\mathbb{R}^2$ and the corresponding set $\mathcal{S}$, we want to see whether or not this set is Delaunay and, if not, how to modify the unsuitable segments so as to form a Delaunay set of edges. The segments in $\mathcal{F}$ are examined. Only those falling within *(Case 2)* are considered. Let f be such a segment.

We first examine the case where only one segment exists in $\mathcal{F}$ and where only one point, P, falls within B_f^{min}, for instance, in $\mathcal{H}_f^-$ (see Figure 9.3):

- if $B_{fP} \cap \mathcal{H}_f^+$ is empty, triangle fP is Delaunay and thus will be formed (and f will be formed),
- otherwise one or several points exist in B_{fP} not in B_f^{min} and f is not formed[5]. However, a point M exists along $f = AB$ such that $B_{f_1}^{min}$ and $B_{f_2}^{min}$, where $f_1 = AM$ and $f_2 = MB$, are empty and we return to *(Case 1)* for these two new segments. Thus, removing f from $\mathcal{F}$ and replacing it by f_1 and f_2 is a suitable solution.

[5]It can be easily proved that there is no point Q, other than the points falling within B_{fP}, such that the triangle fQ is Delaunay.

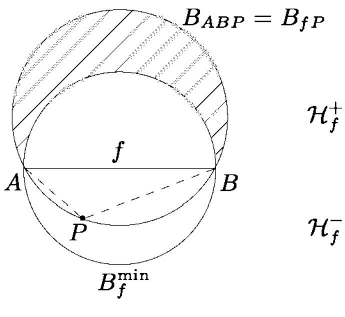


Figure 9.3: *In this pattern, B_f^{min} contains only one point, P, then assuming that $P \in \mathcal{H}_f^-$, the circumball C_{fP} may comprise one or several points in the region of $\mathcal{H}_f^+$ which is shaded in the figure.*

Proof. To prove this result, we just have to define the above point M. One solution is to define M as the projection of point P on f, the point denoted by $\mathcal{P}roj_f(P)$ in Figure 9.4 (left). Then, after replacing f by the above f_1 and f_2, we analyze these two new situations. Obviously, $B_{f_1}^{min}$ and $B_{f_2}^{min}$ are empty. This is due to the fact that relationships

$$B_{f_1}^{min} \subset B_f^{min} \quad \text{and} \quad B_{f_2}^{min} \subset B_f^{min} \tag{9.2}$$

hold, combined with the fact that the possible points of $\mathcal{H}_f^+$ that can interact are outside B_f^{min} (see Figure 9.3) and thus outside $B_{f_1}^{min}$ and $B_{f_2}^{min}$ (see Figure 9.4, left-hand side). □

Remark 9.2 *Relationship (9.2) is the basic key to the problem.*

Remark 9.3 *Following Figure 9.4 (right), we could observe that using as point M, the midpoint of AB, may lead to a solution only after several subdivisions.*

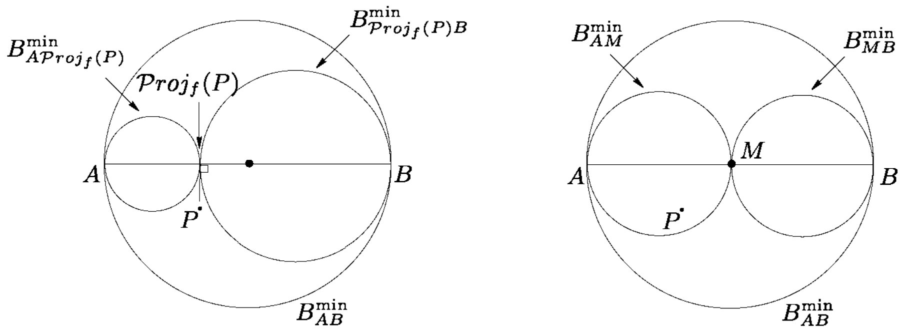


Figure 9.4: *Left-hand side: P in B_f^{min} no longer belongs to $B^{min}_{A\mathcal{P}roj_f(P)}$ and $B^{min}_{\mathcal{P}roj_f(P)B}$. Right-hand side: P in B_f^{min} falls within B^{min}_{AM} where M is the midpoint of AB.*

To pursue, we have to examine the case where we have only one f and several points in B_f^{min}. The following scheme completes the desired solution:

- pick the point P in B_f^{min} such that $\widehat{APB}$ is maximal,
- if B_{fP} is empty, f is Delaunay, END,
- otherwise, apply the above construction. Consider the new segments f_1 and f_2 and repeat the process.

Proof. Actually, Relationship (9.2) gives the key. The radii of the relevant minimal balls decrease and then the number of points in these balls decreases. As the number of points is finite, the solution is yielded. □

Remark 9.4 *Note that the above construction is based on the projection of the point with maximal angle while another point impeding the construction could be used. This means that the resulting solution is not necessarily optimal (in specific, it is tedious to decide whether or not a given f has been split too much).*

Now, by means of exercises, we consider the case where the set $\mathcal{F}$ includes more than one segment.

Exercise 9.2 *Discuss the validity of the above proof in the case of two segments AB and AC sharing point A. Hint: first, observe that if angle $\widehat{BAC}$ is obtuse, segments AB and AC can be decoupled in some sense. Otherwise, if $\widehat{BAC}$ is acute, the two segments are coupled. Adding a point in AB may lead to adding a point for AC and vice versa. Nevertheless, among the points impeding the construction of AB and AC, one can be retained that further decouples the resulting situation.*

Exercise 9.3 *In the same configuration, find a counter-example where subdividing edge AB using the midpoint results in a solution for edge AB which is no longer valid when considering edge AC. Hint: assume the angle between the two segments to be acute and show that adding a midpoint may lead to repeating to infinity the same pathology.*

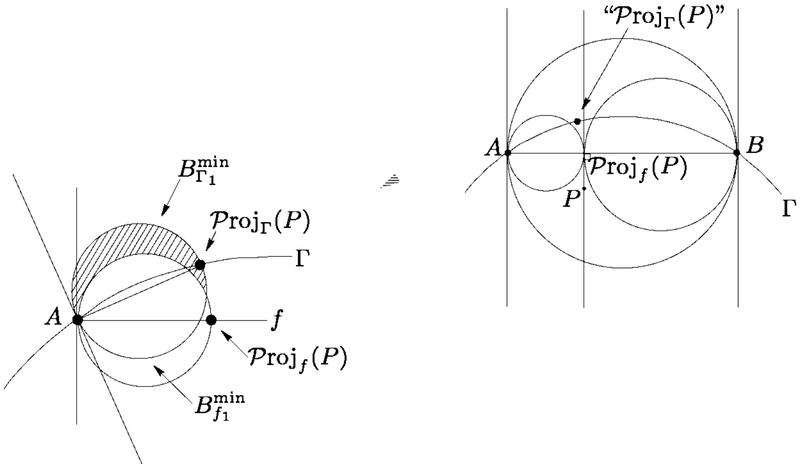
Exercise 9.4 *Turn now to a non-manifold boundary. Consider the case of a point A where several edges meet and where every angle between two segments is acute. Hint: examine the vertices surrounding point A. Find the smallest distance between these points and A. Introduce a point on all edges emanating from A at a distance less than the above value.*

Remark 9.5 *As previously mentioned, we have some flexibility when a projection is demanded. In the above construction, a specific point has been chosen and then projected but other projections may lead to the desired property. Thus a more subtle analysis can be made, specifically if some criteria must be followed. For instance, it could be of interest to ensure that the resulting f_1 and f_2 are not too different in terms of length.*

Exercise 9.5 *Look at the case where the open balls are replaced by the closed balls. Note that these cases may result in some small changes in the previous discussion.*

Delaunay-admissible boundary discretization

The previous material can be used for a slightly different problem. Given a polygonal discretization of a closed curve Γ boundary of a given domain, we want to see whether or not this discretization is Delaunay conforming. If not, we want to modify it so as to complete such a Delaunay conforming discretization.



$B_{\Gamma_1}^{\min}$
$\mathcal{P}\text{roj}_\Gamma(P)$
Γ
A
f
$\mathcal{P}\text{roj}_f(P)$
$B_{f_1}^{\min}$
"$\mathcal{P}\text{roj}_\Gamma(P)$"
A
B
$\mathcal{P}\text{roj}_f(P)$
P
Γ

Figure 9.5: *Right-hand side: the context when a point P must be projected, the initial edge f and the curve Γ. The small ball of f, those of f_1 and f_2 resulting from the "theoretical" method. Left-hand side: the small ball that is not strictly included in the initial one when P is projected on Γ and not on f.*

Then, the above approach can be retained and, specifically, the construction of the projection of a point that impedes the Delaunay process can be used. Nevertheless, in contrast to the above theoretical framework, such a point must be precisely located on Γ and not on an edge f of Γ, which leads to a different location when Γ is a curved boundary. In other words, if $f \in \Gamma$, then f_1 and f_2 as defined above are not necessarily close to Γ and we wish to avoid such a case.

Now, following Figure 9.5, Relationship (9.2) may be not satisfied. Thus, the convergence of the method is an issue. Actually, based on the strict decreasing of the radii of the small balls that are involved, the same result as above can be obtained.

9.2 Delaunay-admissible set of segments in $\mathbb{R}^3$

Following [George, Borouchaki-1997], dealing with the non-Delaunay admissible segments included in a set of three-dimensional segments leads to the same result as in two dimensions and the same sufficient condition holds. Such a segment can be split, if necessary, using the adequate projections, as previously detailed in two dimensions. Similar reasons ensure the convergence. Indeed, the spheres involved in the construction are strictly enclosed in the former spheres they replace.

Let $f = AB$ be a segment in $\mathbb{R}^3$. Segment AB is Delaunay if the Voronoï cells V_A and V_B share a Voronoï entity (vertex, edge or facet). Thus, B_{AB} empty is a *sufficient condition* for Delaunay admissibility.

Proof. Since B_{AB} is empty, the midpoint of AB is in $V_A \cap V_B$, thus we have $V_A \cap V_B \neq \emptyset$. □

Remark 9.6 *The solution may be obtained after a swap (in the case where $V_A \cap V_B$ reduces to a single point).*

Now, assume that B_{AB} is not empty. Then one or several points exist in B_{AB}. One of these points, say P_1, is such that angle $\widehat{AP_1B}$ is maximal. Now, continue the proof by means of the following exercises.

Exercise 9.6 *Show that for any point P in B_{AB} such that $\widehat{APB} < \widehat{AP_1B}$, we have $P_1 \in B_{APB}$ where B_{APB} is the ball whose great circle is the circumcircle of triangle APB.*

Exercise 9.7 *Show that if face ABP_1 is not created, then AB cannot exist.*

As a consequence, the above point P_1 is a natural candidate for forming a face with AB. Then, "B_{ABP_1} empty" is a condition for AB to be Delaunay.

Proof. Since B_{ABP_1} is empty, the center of the circle passing through A, B and P_1 is in $V_A \cap V_B \cap V_{P_1}$ then $V_A \cap V_B \cap V_{P_1} \neq \emptyset$. □

The above result can be obtained by considering the triangulation point of view. The segment AB is Delaunay admissible if, obviously, there exists one Delaunay tet (a series of Delaunay tets) with AB as an edge (to make sense, set S must include more than four points, including A and B, and then the number of tets is at least two).

Let P_1 be the above point (that in B_{AB} such that $\widehat{AP_1B}$ is maximal). B_{ABP_1} empty is a condition that ensures the existence of a Delaunay tet with face ABP_1 and thus with edge AB. Such a tet exists.

Proof. Let r_{init} be the radius of the disc ABP_1, then, we consider a family of balls passing through A, B and the above P_1 whose radius varies from r_{init} to infinity. For a value of this radius, the ball touches a point, let P_2 be this point. Then, tet P_1ABP_2 is Delaunay since, by construction, its ball is empty. □

Exercise 9.8 *After element P_1ABP_2, find the other elements sharing AB. Show that they are Delaunay. Hint: use the same method to find the desired points based on the disc of the face common to the previous element. Then show that each resulting tet is Delaunay.*

Another less demanding condition is now introduced, still based on ball ABP_1 which, now, is not assumed to be empty. Among the points in this ball, we select one, say P_2, such that the solid angle between P_2 and triangle ABP_1 is maximal, then if tet P_1ABP_2 is Delaunay, AB will be formed. In this way, we have a condition based on two specific points.

Remark 9.7 *Above tet P_1ABP_2 is the tet of maximal (circum)ball among all the tets with a vertex in B_{ABP_1}.*

Now, to complete the discussion, we have to follow the same scheme as in the above section starting from a simple situation and going further to the general case (and, specifically, a case where several edges emanate from one point).

9.3 Delaunay-admissible set of triangular faces

We now face the same problem for $\mathcal{F}$, a given set of triangular faces, where one face is denoted as f. The main difficulty that is expected is related to the fact that Relationship (9.2) does not extend to a face in three dimensions. Given a triangle f in $\mathbb{R}^3$ along with B_f^{min} its minimum ball having as *great circle*, gc_f, the circle circumscribing f, then any subdivision of f results into sub-triangles and (see Figure 9.6), in general, we do not have

$$B_{f_1}^{min} \subset B_f^{min} , \qquad (9.3)$$

where f_1 denotes one of these sub-triangles. Actually, introducing a subdivision point inside the initial face f or along one of its edges leads to great circles that are not included in the initial great circle. Moreover, in the case of a surface mesh, splitting a face may lead to splitting the neighboring faces so as to preserve the conformity of the mesh.

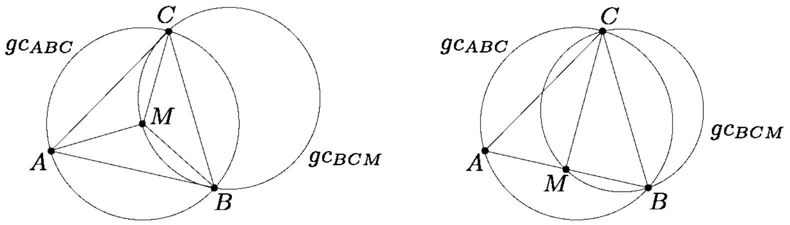


Figure 9.6: *The great circles, $gc_{...}$, (thus the corresponding balls) after subdivision are not strictly included in the initial circles (the initial ball). Moreover, some radii can be significantly greater than the initial one. Left: the face $f = ABC$ is subdivided based on an internal point M. Right: the subdivision point M is located along one edge of f.*

Despite these observations, we want to find a solution[6]. First, we establish the catalogue of the possible patterns, then we propose a method while discussing the convergence issues.

[6]As in Section 9.1, a naive approach to the problem is to make sure that a Delaunay tetra-

Notations

As for the two-dimensional case, we define some notations (see Figure 9.7 and Table 9.2). $\mathcal{F}$ is now a set of triangular faces, $f = ABC$ denotes such a face and $\mathcal{S}$ stands for a set of points including the extremities of the f's. Π_f is the plane supporting f. It defines two subspaces, $\mathcal{H}_f^+$ and $\mathcal{H}_f^-$. Given f, B_f^{min} is the open ball whose great circle, gc_f, is the circumcircle of f (this ball corresponds to the smallest sphere that can be constructed passing through the vertices of f). Finally, given a tetrahedron, say a quadruple $ABCD$ or a quadruple like fM where f is a face (a triple) and M is a point, B_{ABCD} (resp. B_{fM}) is the open ball circumscribing the tetrahedron $ABCD$ (resp. fM).

f *or* ABC	*the face in question*
Π_f	*the plane supporting* f
$\mathcal{H}_f^+$, $\mathcal{H}_f^-$	*the two sub − spaces related to* Π_f
B_f^{min}	*the small ball of* f
B_{AB}^{min}	*the small ball of segment* AB
B_{-C}^{min}	*the small ball of segment* AB (*opposite* C)
gc_f	*the great circle of* f $(gc_f = B_f^{min} \cap \Pi_f)$
B_{ABCD} , B_{fP}	*the circumball of* $ABCD$ (*resp.* fP)

Table 9.2: Notations for the entities involved in the face problem.

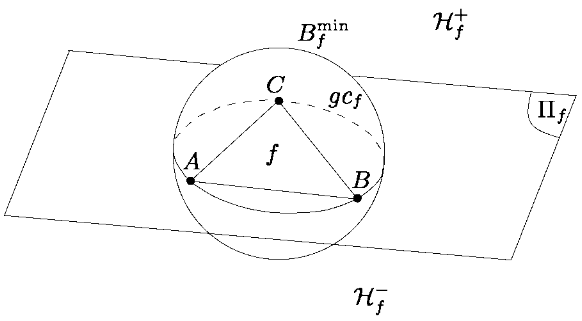


Figure 9.7: *A face* $f = ABC$, *member of* $\mathcal{F}$, *the corresponding ball* B_f^{min}, *the great circle (disk)* gc_f, *the plane* Π_f *and the two associated half-spaces* $\mathcal{H}_f^+$ *and* $\mathcal{H}_f^-$.

hedron exists with f as a face. Note that the cases where a face f does not exist do not reduce to a situation where an edge PQ, where P and Q are two points, one on each of the half-spaces separated by a plane supporting f, intersects f.

Classification

***A priori* classification.** Given a set $\mathcal{F}$ (and the set $\mathcal{S}$), we want to see whether or not $\mathcal{F}$ exists in $\mathcal{D}el(\mathcal{S})$ the Delaunay triangulation of $\mathcal{S}$ (to make sense we assume that more than four points compose $\mathcal{S}$). If not, we construct a new set $\mathcal{F}'$ (and the related set $\mathcal{S}'$) such that $\mathcal{D}el(\mathcal{S}')$ has the desired property (i.e., the faces in $\mathcal{F}'$ exist in the triangulation based on $\mathcal{S}'$).

Thus we need to analyze the configurations associated with any f in $\mathcal{F}$. For a given triangular face f, the following cases can be encountered:

(Case 0): $\mathcal{H}_f^+$ (or $\mathcal{H}_f^-$) is empty, more precisely we are in *(Case 0.1)* if gc_f is empty or in *(Case 0.2)*, otherwise,

(Case 1): B_f^{min} is empty

(Case 2): B_f^{min} is not empty, which leads to several situations: *(Case 2.1)* if gc_f is empty or *(Case 2.2)* otherwise.

Reduced classification. A more precise analysis of the above classification leads us to remove *(Case 0.2)* as well as *(Case 2.2)*. In other words it is always possible to ensure that the gc_f's are empty.

Assume a configuration where gc_f is not empty. If P is a point in gc_f (thus in Π_f), it is immediately clear to see that this point will impede the construction whatever the context (with respect to B_f^{min}, $\mathcal{H}_f^+$ or $\mathcal{H}_f^-$). Indeed, the circumball of any tet with f as a face will include point P and thus such a tet is not Delaunay. Actually we face a problem in two dimensions (in plane Π_f). Given an arbitrary triangulation in a plane (the plane Π_f here), it is possible to achieve a Delaunay triangulation by means of edge swapping. As a consequence the cases where a gc_f is not empty can be removed after edge swapping, thus resulting in a new set $\mathcal{F}$ with empty gc_f's. Thus, the classification reduces to:

- *(Case 0)*: $\mathcal{H}_f^+$ (or $\mathcal{H}_f^-$) is empty,
- *(Case 1)*: B_f^{min} is empty,
- *(Case 2)*: B_f^{min} is not empty,

while, at the same time, the gc_f's are empty. Based on this classification, we want to see if $\mathcal{F}$ is suitable or must be modified so as to obtain what is needed.

Analysis of the three configurations

We examine the three above situations (see [Pebay-1998a] for more details). A face in *(Case 0)* is Delaunay.

Proof. Assume that $\mathcal{H}_f^+$ is empty, then the points candidate for connection with f are in $\mathcal{H}_f^-$. Let Q be the point in $\mathcal{H}_f^-$ such that the solid angle formed with f is maximal, then the ball of tet $ABCQ$ is empty. □

Since gc_f is empty, the sole points that can impede the construction would be exactly located on the boundary of gc_f (and thus co-circular with A, B and C). In this case, f is easy to obtain, possibly after a swap.

Actually, above point Q could be obtained based on what follows:

- pick a point, say P, in $\mathcal{H}_f^-$. Consider the tet $ABCP$ and its ball B_{ABCP}:
  - if B_{ABCP} is empty, then $Q = P$,
  - otherwise, a point M exists in B_{ABCP}, set $P = M$ and repeat the same process.

The solution results from the fact that

$$\text{if} \quad M \in B_{ABCP} \cap \mathcal{H}_f^- \quad \text{then} \quad B_{ABCM} \cap \mathcal{H}_f^- \subset B_{ABCP} \cap \mathcal{H}_f^- . \tag{9.4}$$

Now, a face in *(Case 1)* is also Delaunay.

Proof. First, if P is a point in $\mathcal{H}_f^+$, not in B_f^{min}, for tet $ABCP$ we have $B_{ABCP} \cap \mathcal{H}_f^- \subset B_f^{min}$ thus $B_{ABCP} \cap \mathcal{H}_f^-$ is empty. If P is in $\mathcal{H}_f^-$, we have a similar conclusion, $B_{ABCP} \cap \mathcal{H}_f^+$ is empty.

Now, consider a point P in $\mathcal{H}_f^+$, not in B_f^{min}, form the tet $ABCP$. If $B_{ABCP} \cap \mathcal{H}_f^+$ is empty, set $Q = P$. Otherwise, a point M exists in $B_{ABCP} \cap \mathcal{H}_f^+$, set $P = M$ and repeat the process. At completion, we have found a point Q such that $B_{ABCQ} \cap \mathcal{H}_f^+$ is empty.

Since $B_{ABCQ} \cap \mathcal{H}_f^- \subset B_f^{min}$, then B_{ABCQ} empty holds. □

It may be noted that we again encounter the conditions of Delaunay admissibility of a given face which are similar to those of an edge in $\mathbb{R}^2$. Now we have to examine the remaining case. Let us recall that in *(Case 2)*, for a given face f, we have a situation where:

$$B_f^{min} \neq \emptyset \ , \mathcal{H}_f^- \neq \emptyset \ , \mathcal{H}_f^+ \neq \emptyset \text{ and } gc_f = \emptyset , \tag{9.5}$$

in other words, B_f^{min} encloses one or several points, some in $\mathcal{H}_f^+$, some in $\mathcal{H}_f^-$. Let P be the point in B_f^{min} also in $\mathcal{H}_f^+$ such that B_{ABCP} is maximal, $B_{ABCP} \cap \mathcal{H}_f^+$ is then empty. Thus, a condition for f to be Delaunay is that $B_{ABCP} \cap \mathcal{H}_f^- = \emptyset$. Since the same holds when considering the points in $\mathcal{H}_f^-$, the condition is the same for the point in one or the other half-spaces which maximizes the corresponding circumball.

To summarize, unless the above condition holds (which is actually tedious to check), the faces in *(Case 2)* are not Delaunay.

Construction of a Delaunay-admissible set of faces

Based on the previous classification, some faces are Delaunay admissible, whereas others require specific processing. As previously seen, only faces falling within *(Case 2)* must be examined. We have a situation where B_f^{min} encloses at least one point, say P, for instance, in $\mathcal{H}_f^-$, and the ball circumscribing the quadruple fP encloses at least one point, say Q, in $\mathcal{H}_f^+$ (if the above circumball is empty, face f is Delaunay) while the great circle related to f is empty.

The only case where this particular situation is not the situation encountered would be the following:

- a point P is in B_f^{min} as above, for instance, in $\mathcal{H}_f^-$ and, at the same time, one point (or more) exists in the boundary of gc_f and a Delaunay tet could be formed based on P and this (these) point(s). In this case, a swap may be required to retrieve f.

For the sake of simplicity, we discard this particular case and we first discuss the case where only two such points exist (P and Q). Before going further, bear in mind that any subdivision by given patterns of an unsuitable face may result in replacing this face by several Delaunay admissible faces but at the same time, due to conformity reasons, may lead to splitting some neighboring faces (if we consider a surface mesh) for which the corresponding small spheres may be as large as we want.

Towards a general scheme. After the above remark, the solution cannot be directly completed by "subdividing" the faces. Therefore, the expected result can be obtained only by using a multiple step scheme where, at each step, some *criterion* is enhanced. The scheme we propose is as follows:

- (A) we process the non-Delaunay face edges,
- doing so, we introduce some new faces as well as some new edges,
- as long as, among these new edges, there are some non-Delaunay edges, we return to (A),
- then we process the faces (and, now, all face edges are Delaunay-admissible).

Edge enforcement. The edges of the given faces that are not Delaunay admissible are dealt with using the previous method. In this way, they are subdivided, if judged useful, and an admissible partitioning is completed.

Nevertheless, when subdividing an edge, for instance, into two sub-edges, a new edge is introduced. In fact, the problem does not reduce to considering stand alone edges in $\mathbb{R}^3$ but edges that are face edges. Thus, it is necessary to maintain a topological face structure.

At completion, we have a set of faces all of whose edges exist, and we then turn to the face enforcement step.

Face enforcement. Following footnote 6 above, we examine more precisely a configuration where the given face is missing due to two points P and Q, one in $\mathcal{H}_f^-$, the other in $\mathcal{H}_f^+$ (this point may be also in B_f^{min} or not), such that edge PQ is Delaunay admissible and impedes the construction of face f. Some occurrences of such a pattern are depicted in Figure 9.8.

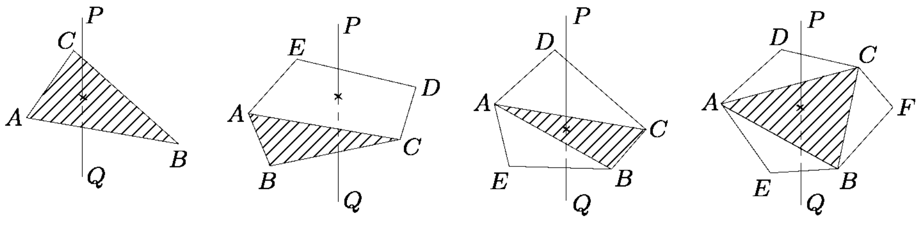


Figure 9.8: *A face $f = ABC$, a member of $\mathcal{F}$, is not Delaunay due to an edge PQ. Several tets sharing PQ exist whose vertices, apart from P and Q, could be those shown in the figure.*

If edges AB, AC and BC exist, we are faced with a situation where PQ intersects ABC, then only tets $PQAB$, $PQAC$ and $PQBC$ exist as can be easily proved (left-hand side of the figure). Thus, the corresponding circumballs are empty. Then, observe that the only way to prevent the creation of the three above tets is to violate the Delaunay criterion. In other words, a solution is to introduce a point in the intersection of these circumballs.

We first consider the case where P, in B_f^{min} is such that the sphere passing through P and f is larger than that passing through Q and f. Let M be the projection of P onto the face f. *A priori*, based on the position of P with regard to f, M is either internal to f or is located on one of the edges of f. Since PQ cuts f, it is easy to see that M is necessarily inside f and falls within the three above balls, therefore this particular point will prevent the creation of edge PQ.

Using the thus-defined point M, the face f is subdivided by means of three sub-faces. The new edges are processed so as to obtain Delaunay-admissible edges (via the above method or using some edge swap in the plane of f) and we consider the set of resulting faces.

At completion, three results hold. First, point P is no longer inside any balls associated with the sub-faces replacing face f. Next, the radii of the balls related to the edges and those related to the newly created faces decrease. Thus, point P has been discarded from the balls of the new faces. Indeed, point M acts as a "wall" between P and Q which are no longer coupled together (i.e., P is not closer to Q than any other point in $\mathcal{S}$).

This ensures the convergence of the process.

We now turn to a case where there are several points in B_f^{min}, such that several edges intersect the face f. Among these points, we pick the one forming the maximal solid angle with f (the sphere passing through f and this point is maximal). We project this point onto f and we repeat the same process, then we consider again the B_f^{min} (the above point being discarded), we repeat the same process as long as a point exists inside one of these new balls.

Remark 9.8 *Notice that the only impeding edges are those which cut the face (thus, in Figure 9.8, three cases correspond to this situation while the fourth case is not of interest).*

Remarks about the convergence. We postulate that the global convergence holds. For the face edges, this results from the previous discussion. Regarding the faces, the same result holds if we can make sure that the process has no loop. This proof remains to be properly established, as far as we know (unless some particular assumptions are assumed about, for instance, the value of the angles between two edges).

From a practical point of view, we can find in [Pebay-1998a] an algorithm based on the previous issues which is completed by some heuristics. The key idea is to enforce the edges prior to considering the resulting faces. Furthermore, to each of these faces, the following process is applied:

- a subdivision based on the a choice about the three possible cases (one point being added on one of the three edges) such that the retained pattern maximizes the minimum angle between the resulting triangles,
- an edge swap (in the case of coplanar faces, this being exactly or approximatively verified).

This heuristic strategy has proved to be adequate in numerous significant examples (as seen in the previous reference).

Delaunay-admissible boundary discretization

We face the same problem as in two dimensions. Given a discretization of a closed surface Σ boundary of a given domain, we want to see whether or not this discretization is Delaunay conforming. If not, we want to modify it so as to complete such a discretization.

The above approach can then be employed and, specifically, any point, edge or face creation required to ensure the desired property must be such that this entity is precisely located on Σ and not on a face f of Σ.

9.4 Medial axis

First, we recall the definition of the medial axis of a domain. Given a domain Ω in $\mathbb{R}^2$, the medial axis of Ω is defined as follows:

Definition 9.1 *The medial axis of a domain is the locus of the centers of the circles of maximum radius that can be inscribed in the domain.*

Delaunay triangulation and medial axis construction

Following on from the above, it is possible to find the *medial axis* or the *skeleton* of a given *polygonal* domain based on its boundary discretization. From the above definition, the medial axis of a domain is the locus of the center of a circle of maximal diameter as it rolls inside this domain.

Given a Delaunay admissible discretization of a domain boundary, it is easy to obtain a Delaunay triangulation of this domain whose sole vertices are the endpoints of the edges of the above boundary discretization. In terms of the Voronoï diagram corresponding to the above triangulation, we encounter two types of Voronoï entities: the *Voronoï edges* and the *Voronoï vertices.* A Voronoï edge is equidistant from two Delaunay vertices while a Voronoï vertex is equidistant from three such vertices[7]. The Voronoï edges dual to a boundary edge are removed and the resulting edges form a polygonal line (which can reduce to a point). Under appropriate assumptions, this line enjoys some good properties and thus gives a rough idea of what the medial axis is. Note that among the Voronoï edges that are removed, we encounter first the *non-finite* Voronoï edges[8] (also referred to as unbounded edges) as well as some finite edges related to concave parts of the boundary.

A theoretical issue can be written as the following theorem:

Theorem 9.1 *If h tends to 0, where h is the length of the longest edge of the discretization of the boundary of Ω, then the union of the finite internal Voronoï edges associated with the triangulation of Ω based on its boundary vertices approaches the medial axis of Ω.*

In other words, the medial axis is basically obtained once h tends towards 0, by joining the centers of the circumcircles of the triangles in the Delaunay mesh.

First, we give a sketch of the proof, then we discuss the convergence issue of the proposed method (i.e., the finite internal Voronoï edges).

Proof. Since h tends to 0, the boundary discretization is Delaunay admissible. The triangulation of Ω based on the boundary vertices is then Delaunay. As a consequence, the Voronoï edges are easy to obtain based on the dual of the current triangulation. Now, among these edges, those which are not related to a boundary Delaunay edge are inside the domain and the circles centered in these edges are inside the domain. Moreover, the above circles have maximal radii. Thus, the thus-selected Voronoï edges form the medial axis of Ω.

The first three points are obvious. Given an arbitrary boundary discretization, it is possible to modify it so as to complete a Delaunay admissible discretization. Moreover, such a discretization can be uniform which will further simplify matters. Let h_{min} be the smallest distance between two non-consecutive boundary vertices, then $h \leq \frac{h_{min}}{2}$ is a suitable value for the definition of the distance between two consecutive boundary points. Indeed, a uniform mesh of the boundary can be

[7]Or more in the case of co-circular points.

[8]Indeed, such edges intersect a boundary edge.

constructed with boundary edges of length h. This mesh satisfies the condition of Delaunay requirement. As a consequence, a Delaunay triangulation of the domain can be easily constructed whose vertices are the above uniformly spaced boundary points.

The dual of this triangulation can be constructed. The edges of the dual that are not related to a Delaunay boundary edge are inside the domain. The latter property results from an appropriate choice of h as can be easily seen.

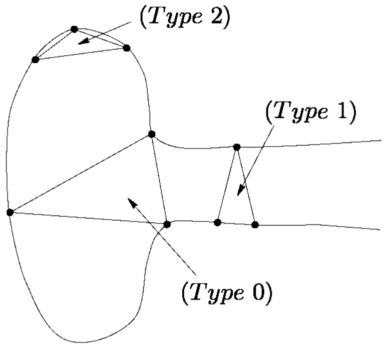


Figure 9.9: *The three types of triangles that can be encountered in an "empty" simplicial mesh in two dimensions.*

It still remains to prove that, when h tends to 0, the circles centered on these edges are inside the domain and are maximal. From a computational point of view, three types of Delaunay triangles exist based on the number of boundary entities they have. A Delaunay triangle may have (see Figure 9.9):

- no boundary edge, termed a *(Type 0)* triangle,
- one boundary edge for a *(Type 1)* triangle,
- two boundary edges[9] and the triangle is termed a *(Type 2)* triangle.

Accordingly, the medial axis can be found by considering the way the different triangles contribute to it. First, we consider the case of a *(Type 0)* triangle before discussing a pair of two adjacent triangles.

- *(Type 0)* triangle.

Let Γ be the boundary of domain Ω and let ABC be the triangle in question. The circumcircle of this triangle touches the boundary at three vertices. It is maximal. If h is small enough, the circle is internal to Ω as it satisfies the Delaunay criterion. At A, B or C, it is tangent to Γ if this boundary is at least of class C^1. Otherwise, we have a simple contact, the contact point being a corner (Γ is

[9]The case where the three edges of a triangle are boundary edges is a specific case where the domain approximation reduces to this single triangle. Thus a (Type 3) triangle, corresponding to this case, is not really interesting.

only of class C^0 at this point). Therefore, the circumcenter (which is inside the triangle) belongs to the medial axis. This point is a *special node* or a *critical node* of the axis in which three Voronoï edges meet. Based on the configuration related to the three adjacent triangles (see below), such a node is the beginning endpoint of three branches of the axis (which can be reduced to this point in some cases, for instance, in the case of co-circular points).

To know the regularity of the contact between this circle and the boundary, it is necessary to analyze the edges of Γ in one or other side of this contact vertex. If Γ is at least of class C^1, these two edges tend with h towards the (unique) tangent and the contact is of class C^1. If Γ is of class C^0, the two edges form an angle (other than Π) and we have a simple contact, with no tangency property.

Note, in addition, that whatever h, a *(Type 0)* triangle does not vanish (its surface area does not generally tend to 0 with h).

Now, to obtain the entire medial axis, apart from the critical nodes, we have to examine the contribution of all pairs of adjacent triangles. First, we discuss the case where one of the two triangles is of *(Type 0)*, thus leading *a priori* to three possibilities, then we turn to the case where one triangle is of *(Type 1)* resulting in two new cases (while a pair *(Type 2)-(Type 2)* does not make sense).

- Case of a *(Type 0)-(Type 0)* pair.

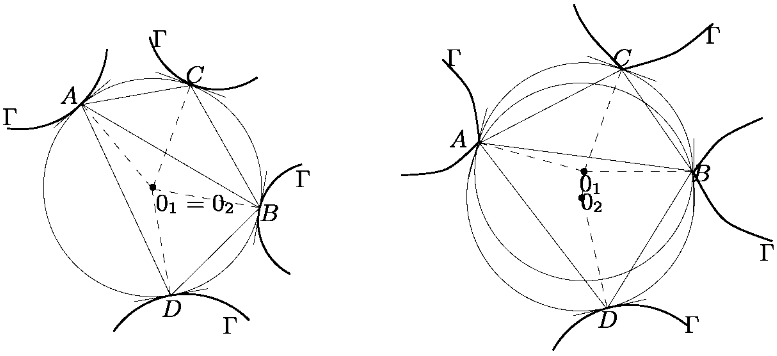


Figure 9.10: *The contribution of a (Type 0)-(Type 0) pair. Left: the four vertices are co-circular. Right: the four vertices are not co-circular.*

A combination *(Type 0)-(Type 0)* is depicted in Figure 9.10. We denote by O_1 the center of the circumcircle of triangle ABC and by O_2 that of circle of ADB, AB being the common edge. If the four vertices are not co-circular, the centers O_1 and O_2 are distinct. The segment O_1O_2 is a part of the medial axis. Indeed, the circle centered at O_1, as well as that centered at O_2 obviously conforms to the definition and, moreover, all circle centered on O_1O_2 and passing through the endpoints A and B of the common edge is maximal, inside the domain and touches this domain at two points. The point O_1 is, potentially, the origin of three branches of the axis (see below) among which is the branch O_1O_2, so it is for the point O_2. The case where the four vertices are co-circular implies that $O_1 = O_2$ and this critical node is, potentially, the origin of four branches of the medial axis, as will be seen shortly.

- Case of a *(Type 0)-(Type 1)* pair.

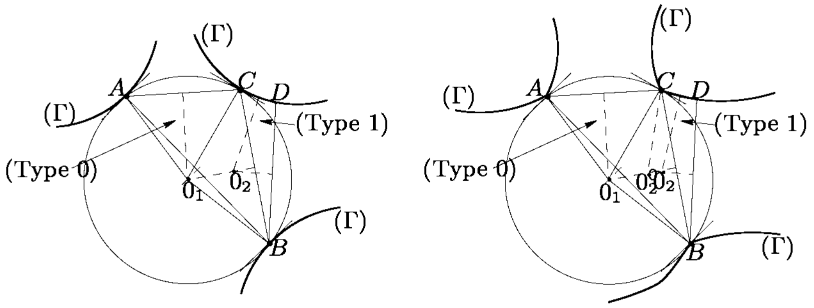


Figure 9.11: *The contribution of a (Type 0)-(Type 1) pair. Left: the case of a C^1 boundary at point C, O_2 tends towards O_1 with h. Right: the case of a C^0 boundary at point C, O_2 tends towards O_2^0 with h and, in general, O_2^0 is distinct from O_1.*

Figure 9.11 illustrates a *(Type 0)-(Type 1)* combination. Let ABC be the triangle of *(Type 0)* and BDC be the triangle of *(Type 1)* which is adjacent to the previous through the edge CB. Edge CD is a boundary Delaunay edge whose size is h. As above, we notice O_1 and O_2 the centers of the circles circumscribing the two triangles in question. If CD tends with h towards the tangent at C at Γ, then O_2 tends towards O_1. Otherwise, O_2 tends towards a point O_2^0 distinct from O_1. If h is small enough (non-zero), in the first case, any point belonging to O_1O_2 is a point of the medial axis and thus O_1O_2 is a portion of this axis. In the second case, it is the same but the medial axis will have a discontinuity reflecting the corner in the boundary at C (see the remark below). Notice again that the various points O involved in the construction are or are not inside the corresponding triangles. The contribution to the medial axis of a pair of triangles is not necessarily located inside these triangles. To conclude, if D is co-circular with A, B and C, then the contribution to the axis is reduced to the center-point of the circle of ABC.

Remark 9.9 *The boundary regularity induces the medial axis smoothness. A point like O_2^0 results in a local regularity of type C^0 in the axis. Otherwise, this line is locally at least of class C^1.*

- Case of a *(Type 0)-(Type 2)* pair.

A *(Type 0)-(Type 2)* junction is *a priori* possible. Nevertheless, if h is small enough, this pattern does not exist. Indeed, when h vanishes, a *(Type 2)* triangle reduces to one point. Thus, this case is only feasible from a discrete point of view and does not participate to the theoretical discussion. Note that for numerical reasons, this case can arise for a sufficiently small (but non-zero) h.

- Case of a *(Type 1)-(Type 1)* pair.

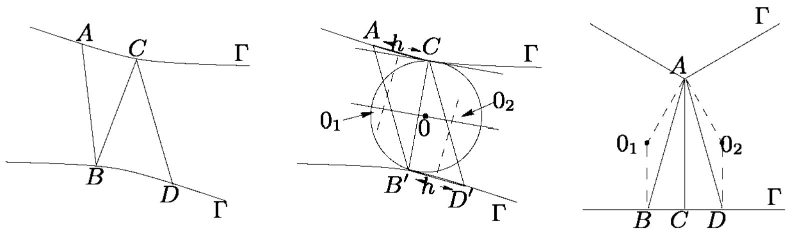


Figure 9.12: *The contribution of a (Type 1)-(Type 1) pair. Left and middle: first possible situation. Right: second case where a beam is encountered.*

In the following (Figure 9.12), a *(Type 1)-(Type 1)* junction is discussed. Indeed, two such junctions can be found based on the geometry of the domain boundary. Let ABC and BDC be the two triangles sharing the Delaunay edge BC which traverses the domain. The first category of *(Type 1)-(Type 1)* junction is encountered when AC and BD are the Delaunay boundary edges, while the second corresponds to the case where the Delaunay boundary edges are BC and CD, meaning that point A is the base of a *beam* of edges traversing the domain from one boundary to the other.

Let us consider the first case, when h tends to 0; let us assume that C remains unchanged, meaning A tends towards C. Then, the circles passing through C tangent to the boundary are centered on the "right" normal[10] at the boundary at vertex C. Among this family of circles, let us pick the one which is tangent to the opposite boundary and let O be its center. If B' is the point where this property holds, the retained circle is centered on the "right" normal at the boundary at this vertex B'. Let us examine[11] triangle $AB'C$ as well as triangle $B'D'C$, which are Delaunay triangles. The circumcenters of these two triangles, O_1 and O_2, define the Voronoï edge O_1O_2 and O is located on this edge. Since h tends to 0, the three points O, O_1 and O_2 tend towards the same point, actually point O. Now, from a discrete point of view, if h is small enough, O_1O_2 is the approximate medial axis as contributed by the two triangles in question.

Now we turn to the second possible situation, namely a beam corresponding to one or two concave parts of the boundary. A close look at Figure 9.12 (right-hand side) leads to proving that the circumcenters of any triangles in the beam tend towards O_1 (resp. O_2) with h based on the position of the triangle under examination. Depending on the boundary regularity, these two points are distinct or not, which is also visible about the axis regularity (which may contain a portion of a parabola whose links with the adjacent segments give the resulting smoothness).

- Case of a *(Type 1)-(Type 2)* pair.

[10]This notion of "right" normal is due to the fact that point C tends towards point A from the "right" side.

[11]Depending on the boundary curvature near points C and B', the Delaunay triangles that are formed are $AB'C$ and $B'D'C$ or $AB'D'$ and $B'D'C$. Nevertheless, the discussion is the same whatever the situation (while using the "left" normals).

A *(Type 1)-(Type 2)* junction is now discussed. As for the *(Type 0)-(Type 2)* case, the *(Type 2)* triangle vanishes, thus contributing with its terminal node and the corresponding Voronoï edge (which may not exist in the case of co-circularity). The medial axis ends at the center of the circle of the triangle of *(Type 2)*, to complete it, this point can be linked with the terminal node.

To summarize, we define the critical nodes (related to the *(Type 0)* triangles) then the branches between two such nodes. These branches result from the contribution between a *(Type 0)* triangle and its neighbor of *(Type 1)*, and then from those of the pairs *(Type 1)-(Type 1)* until a contribution of a pair *(Type 1)-(Type 0)* or *(Type 1)-(Type 2)*, in which case the medial axis ends at a terminal node (except in a case of co-circularity where the axis ends before this point). Thus the proof of Theorem 9.1 holds. □

After this discussion, let us mention a few remarks.

Remark 9.10 *From a practical view point, h is a small value (and thus does not tend towards* 0*) and the above discussion only gives a general idea about the expected result.*

Remark 9.11 *If we consider a polygonal approximation of a curved boundary, the circles circumscribing (Type 2) triangles tend towards the osculating circles of the boundary curve.*

Remark 9.12 *As previously seen, the topology of the medial axis is defined based on the critical and terminal nodes. The branches in the axis are defined between two such nodes.*

Remark 9.13 *One should define the medial axis as the locus, when h tends towards* 0, *of the circumcenters of the Delaunay triangles. This method is good* a priori *but is nevertheless badly suited to the case where the boundary is not sufficiently smooth. Indeed, two neighboring centers do not necessarily converge toward a unique point, thus leading to a discontinuity (a hole) in the axis. Examples of such cases have been seen in the above discussion and can be found in [Turkiyyah* et al. *1997].*

Voronoï cells of a set of points and medial axis

The data input of points located on the domain boundaries which are dense enough, while not explicitly defining the corresponding boundary edges, make it possible to retrieve the same result. A Delaunay triangulation is built, based on the insertion of these points and the corresponding Voronoï cells are considered so as to find the medial axis of this cloud of points. If this cloud is dense enough, this axis is that of the domain that is implicitly defined by this type of input data.

Computational issues

In practice, three types of numerical difficulties generally arise. First, the case where there are co-circular points leads to an imprecise definition of the axis (which should be reduced to one point) because the Delaunay triangulation is not uniquely defined in such a situation. The second issue concerns the choice of the h's in such a way as to ensure a suitable regularity of the line completed by the construction. A third problem, which is immediate, is related to numerical errors. For instance, one case of such a problem leads to finding *(Type 2)* triangles and thus branches in the axis while such triangles (branches) do not exist in theory. Moreover, the two following exercises demonstrate some interesting issues.

Exercise 9.9 *Find the relationship between the hs on the boundary discretization and the lengths of the portions of lines in the approximate medial axis.*

Exercise 9.10 *Find the relationship between the hs on the boundary discretization and the boundary curvature in order to guarantee a good enough smoothness in the approximated medial axis.*

Notice that the aim is to minimize the size of the mesh by choosing variable hs, the simplest solution clearly being to consider a constant and small value for these hs.

Finally, a rough idea of the medial axis is given by the following.

Remark 9.14 *Joining the midpoints of the internal Delaunay edges allows a line to be constructed whose topology, when h is appropriately sized, is similar to that of the medial axis. Notice that defining the element centroid as a bifurcation point ((Type 0) triangle) is an easy solution (since this point falls within the triangle) but, depending on the context, could be a rather coarse approximation of the exact result.*

9.5 Mid-surface

Constructing the mid-surface of a domain using a Delaunay triangulation based only on surface boundary points adequately distributed on this surface is more tedious. Before giving a few remarks about this approach, we recall the formal definition of what a mid-surface is (similar to Definition 9.1).

Definition 9.2 *The medial surface of a domain is the locus of the centers of the spheres of maximal radius that can be inscribed in the domain.*

Find the mid-surface of an empty Delaunay triangulation (with no internal point) of the domain appeals an immediate comment. If the domain boundary mesh is not Delaunay-admissible, there is no guarantee (Chapter 7) that such an empty triangulation exists (due to the Steiner points). Moreover, when h tends towards 0, a situation assumed in theory, this phenomenum is not relevant. Thus, we can discard this difficulty, for the moment.

We therefore assume the existence of the desired triangulation. Then, the equivalent of Theorem 9.1 (which we prefer not to attempt to formulate and prove here) is simply assumed. Note, without going into detail, that the proof is necessarily harder than in two dimensions. Specifically, if we want to use a classification of the tets, we clearly find many more types than above.

Exercise 9.11 *Classify the tets according to how many of their edges or faces belong to the domain boundary (while their four vertices are assumed to be on this boundary).*

This being done, it is shown that there are tets having only boundary edges (and no boundary faces) and the precise examination of their neighboring elements is necessary in order to find a possible contribution to the mid-surface. Let us recall that in two dimensions, the contributions to the medial axis corresponded to the dual of the triangulation, say some Voronoï edges. In three dimensions, the situation is much more complex. regarding the dual, we find some Voronoï faces and some Voronoï edges whose presence indicates the way in which the underlying tet touches the boundary.

A few papers discuss, in greater or lesser detail, these aspects and, among these, the reader is referred to [Yu *et al.* 1991] and [Armstrong *et al.* 1993].

Medial axis (mid-surface) transform

This is a process commonly called the "*Medial Axis Transform*" (MAT for short), see for instance, [Price *et al.* 1995], [Armstrong *et al.* 1995] or [Sheehy *et al.* 1996]. This transformation, used in a meshing context, has also long been used for graphic purposes [Blum-1967].

The key idea is that the data of the medial axis, together with some additional information (radii, branches, etc.), make it possible to retrieve the boundary of the domain in question.

9.6 Applications

Numerous applications take advantage of the medial axis (resp. mid-surface) of a domain in $\mathbb{R}^2$ (resp. $\mathbb{R}^3$). A first application consists of partitioning a domain into several sub-domains. Furthermore, this approach allows quad (hex) mesh generation methods to be applied in these sub-regions. On the other hand, the analysis of the entities in the medial axis (mid-surface) provides some indications about the domain geometry which, in turn, allows for some different operations such as dimensional reduction (a domain in $\mathbb{R}^3$ is seen as a surface, a domain in $\mathbb{R}^2$ is seen as a line), or a simplification of geometry details where a detail is found to be based on too "small" an entity in the axis.

Domain partitioning

In two dimensions, using medial axis identification, it is possible to split the domain into geometrically simple regions. This subdivision is based on the types of the triangles which indicate whether a branch exists or not. The existence of a branch (a triangle of *(Type 0)*) reflects the fact that several paths on the domain are possible starting from this branch. The types also indicate the local concavity or convexity of the domain boundary.

Following these observations, it is possible to define some *cut lines* that separate the domain. A *(Type 2)* triangle corresponds to a convex region. A *(Type 1)* triangle traverses the domain from one boundary to another and the examination of the elements neighbor of this triangle indicates the geometric nature of the domain boundary.

A first method. The vertex of a *(Type 2)* triangle common to its two boundary edges is a terminal node of the approximated axis and a line emanating from this point defines a cut. This line is followed, passing through some triangles with *(Type 1)* and *(Type 0)* until a *(Type 2)* triangle is found (by vicinity). In this way, we define a graph which allows for the definition of a partition of the domain by means of polygonal regions (assuming the data of a polygonal boundary).

Provided the domain has no hole and assuming the boundary edges between two vertices in a *(Type 2)* triangle are considered as one side, assuming also that the edges included between such a vertex and a node in a *(Type 0)* triangle and also those between two nodes with *(Type 0)* as one side, then the number of sides in the thus-defined polygon is the number of the triangles of *(Type 0)* traversed in the path plus two. In this way, the resulting polygons have three, four, five or six sides[12].

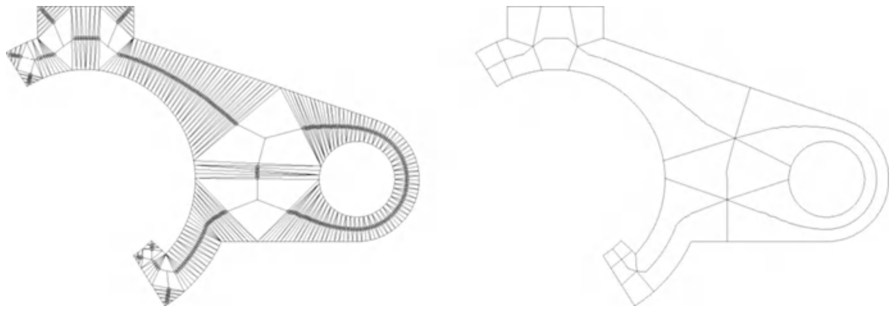


Figure 9.13: *Domain partitioning based on its medial axis, in two dimensions. Delaunay triangulation and medial axis approximation (left-hand side) and partition resulting from method 2 (right-hand side).*

Another method. The above method can be replaced by a method which relies more explicitly on *(Type 0)* triangles. We construct the three lines joining the node in the triangle and its three vertices. Then, while traversing the axis, we join this node to that of the next *(Type 0)* triangle. This results in a partition of the domain into a set of quads and/or triangles only (Figure 9.13).

Either approach provides a partition of the given domain. Notice that numerous variations can be used in order to optimize such a partition, in particular, by

[12]This result can be found in the literature. Is it possible to find more sides? Presumably not.

explicitly taking into account the concave points (or, at least, some of them based on the angles) so as to convexify the resulting regions as well as possible.

Whatever the method, the resulting partition can be used with a view to quad construction.

The same partitioning principle can be used in three dimensions. We use the faces in the mid-surface and their type (i.e., the type of the underlying tets) to find the cutting surfaces in the domain. However, it is clear that the actual implementation is much more complex and, have yet to be thoroughly mastered (the examples that can be seen in the literature are, for the most part, rather academic).

The process begins by analyzing the corners, the edges and the faces in the mid-surface. Based on this classification, we consider the way in which a sphere of maximal radius centered in one entity of the mid-surface touches the domain boundary. In accordance with the situations that arise, we find whether it is possible or not to complete a particular (polyhedral) region. The goal is then obtained once the domain has been entirely subdivided by means of such regions.

In conclusion and with no further discussion about this problem, we think that some good research issues are likely to be found and various new programming developments may be expected in this area.

Quad mesh construction

First, it is possible to use a partition in polygons (each having a small number of sides) and to deduce from it a (coarse) quad partition. If the polygon is not a quad itself, the midpoint subdivision method completes what is expected.

The latter technique consists of inserting one point in the region (for instance, its centroid) and one point in each of its sides (for instance, the midpoint for a free side, i.e., a side related to a portion of the medial axis and one point, close in some sense, for a boundary side). It is easy to see that only quads are obtained in this way. The number of quads is the number of sides (a triangle results in three quads, etc.). This subdivision is made globally and results in a first coarse conforming covering-up of the domain (Figure 9.14, middle).

The resulting covering-up serves as a basis for the construction of the final mesh. To complete this mesh, there are two approaches. The first consists of repeating the midpoint subdivision process. The second method uses a classical algebraic mesh generation method (Chapter 4).

Whichever approach is used, the global conformity of the mesh must be maintained. This leads to imposing consistency about the number of subdivision of a given coarse element by taking into account its neighboring elements and, therefore, this imposes some constraints that propagate from element to element.

Finally, given the subdivision parameters, there is no special reason for having the points of the given boundary follow this point distribution. Thus, new points are constructed in the boundary and it is these points that define the new geometry of the resulting mesh and thus that of the final discretization of the domain (Figure 9.14, right-hand side).

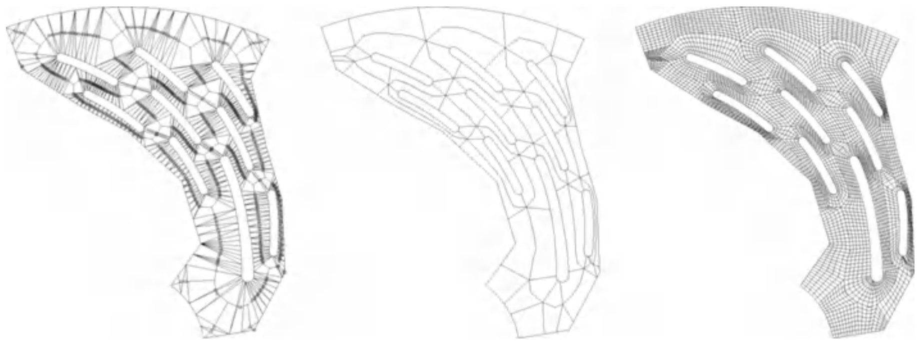


Figure 9.14: *Examples of quad meshes based on the medial axis, in two dimensions: Delaunay triangulation of the domain (left-hand side), domain partitioning using the approximated medial axis (middle) and domain mesh resulting from an algebraic method (right-hand side).*

Hex mesh construction

As can be imagined, this topic is in principle the extension to three dimensions of what we did in two dimensions. In practice, the expected difficulty is much greater. Notice that two mesh generation methods can be used to mesh a region resulting from the partition:

- as in two dimensions, a midpoint subdivision method (possibly repeated) or an algebraic type method (Chapter 4) or a similar method, [Price *et al.* 1995], [Price, Armstrong-1997],
- solely in three dimensions, an approach by extrusion (the product method as seen in Chapter 8) which consists in using a mesh of the considered region of the mid-surface and to extrude it in a third direction (towards the domain boundary),

while ensuring compatibility at the region interface level.

Other applications

Among the other applications using the medial axis or the mid-surface of a domain, we can envisage dimensional reduction [Donaghy *et al.* 1996] or domain simplification [Armstrong *et al.* 1995].

Dimensional reduction. In essence, this operation consists of replacing a computation in two dimensions by one in one dimension (in three dimensions by a computation based on a surface). Observe that the shell case is a case where the domain is a three-dimensional domain but where the computation is made based on a surface (after some assumptions and using some data values that allow for the three-dimensional aspect of the problem in question).

Dimensional reduction is mostly used in two situations:

- the problem, in its intrinsic dimension, can be approximated by a problem posed in a space with a lesser dimension (assuming some more or less restrictive hypotheses),
- the solution of the reduced problem provides an indication, albeit relatively coarse, about the solution of the exact problem and thus already makes it possible to know some important parameters for a reduced CPU cost.

Here again, the medial axis (in two dimensions) serves at a basis for the dimensional reduction procedure. However, the medial axis, by itself, is not usually the ideal solution. Indeed, the adequate solution can be formed by a combination of regions (entities) in zero dimension (some points), in one dimension (some edges) and, in some places, in two dimensions, all of them being adequately linked to as to permit the further usage of the reduced model. Notice also that a simplification process (see below) enables us, in some cases, to obtain a more complete reduction.

Geometry simplification. Simplifying the geometry of a domain is an operation that has proved useful in various contexts (and it will be discussed again later, in particular, in the surface mesh case; see Chapter 15). For the moment, we use the properties of the medial entity in order to simplify the geometry. The key idea is to remove some details that are judged useless (in other words, too small in some sense) in the geometry while preserving the general shape and the topological structure of the geometric model.

Remark 9.15 *Detail removal must be made in accordance with the targeted application. Indeed, during the solving of the problem, a small detail may be the source of a singularity in the solution and thus removing it may alter the computed solution. Thus it is recommended to remove details whose influence remains local. For graphical purposes, the size of a detail is the only factor to be considered.*

The medial axis and the corresponding radii (in two dimensions) indicate the size of a given detail as compared with its neighborhood.

Visiting the medial axis, we detect the possible holes and the possible loops, we then evaluate the size of these holes (by observing the path visited in the axis by comparison with the average radius of a maximal circle traversing the same path). In this way, it is possible to decide whether a hole can be suppressed or not (edge collapsing, see Chapter 18, is then a suitable solution).

The edges are then examined. With each edge is associated a value defined as the ratio between the edge length and the average radius of the maximal circle that touch it. Again, this value allows for the decision. In this way it is possible to simplify the geometry by suppressing the fillets, the small notches, the small protrusions or the small stepsizes.

In three dimensions, suppressing a face is much more a delicate operation and (see Chapter 19), such an operation will be made using a series of edge collapsing while maintaining topological coherence and smooth enough regularity for the thus-simplified surface.

Mesh Generation : Application to Finite Elements
Pascal Jean Frey & Paul-Louis George
Copyright © 2008, ISTE Ltd.

Chapter 10

Quadratic Forms and Metrics

As the perspicacious reader will have already noticed in the presentation of the main governed mesh generation methods (Chapters 5 to 7) and as will also be seen in the chapters devoted to curve and surface meshing (Chapters 14 and 15), as well as in the sections dealing with h, p and hp-methods (Chapters 21 and 22), lengths, distances and other metric-like relations play an important role and are key features in numerous mesh generation and evaluation algorithms.

From a mathematical point of view, the definition of the length of a given vector (resp. a segment) or, similarly, of the distance between two points is based on an adequate definition of the dot product. Algebraic results indicate that this product is related to a quadratic form (associated with a bilinear form). Depending on the objectives, various definitions of these notions can be exhibited, therefore leading to various definitions of a *metric*.

★
★ ★

This chapter begins with some elementary reviews of quadratic forms (which can be found in textbooks), then the notion of length and metric are introduced and explained. The definition of the *unit length* is introduced and discussed in detail as a simple way to measure the length of a given item (segment, vector, etc.) with respect to a given metric. Examples of metrics are given to emphasize the different types of control that can be applied based on the previous notions.

Different metric-related operators are then suggested. They allow us to apply the various metric manipulations usually involved in a mesh generation or mesh modification context. To this end, we briefly discuss the simultaneous reduction, the interpolation and the intersection of two given quadratic forms. We then focus on the metric smoothing problem when these metrics present discontinuities or variations that are too great.

Finally, we briefly discuss a way of constructing metrics suitable for surface meshing and numerical simulations based on finite element methods with a control of the error (of interpolation, for instance).

10.1 Bilinear and quadratic forms

The goal of this first section is to recall some classical definitions and mathematical results of linear algebra, related to linear, bilinear and quadratic forms. These results will notably serve to compute the edge lengths of the given meshes.

Linear and bilinear forms

Let E be a vector space on a field K. We recall that a *basis* of E is a part $a = (a_1, a_2, ..., a_n)$ of E such that each vector $u = (u_1, ..., u_n)$ of E can be written in a unique fashion as:

$$u = \sum_{i=1}^{n} a_i\, u_i\,.$$

All bases of E have the same number of elements, the so-called *dimensions* of E. Now, let consider a vector space E on a commutative field K of characteristics different from 2.

Definition 10.1 *A linear application f, defined on E with value in K, is called a* linear form.

Each linear form has the two following properties:

$$\left\{\begin{array}{rcll} f(u+v) & = & f(u)+f(v) & \forall u, v \in E \\ f(\lambda u) & = & \lambda f(u) & \forall u \in E, \forall \lambda \in K \end{array}\right. .$$

The set $\mathcal{L}(E, K)$ of the all linear applications from E to K is an *additive group* by defining, if f and $g \in \mathcal{L}(E, K)$, the *sum* $f + g$ and the *opposite* $-f$ as:

$$(f+g)(u) = f(u) + g(u) \quad \forall u \in E$$

$$(-f)(u) = -f(u) \quad \forall u \in E\,.$$

Definition 10.2 *We call* bilinear form *on $E \times F$ any bilinear application f from $E \times F$ to K satisfying the two following conditions, $\forall u_j \in E$, $\forall v_j \in F$:*

1. $f(\lambda_1 u_1 + \lambda_2 u_2, v) = \lambda_1 f(u_1, v) + \lambda_2 f(u_2, v), \quad \forall \lambda_j \in K,$

2. $f(u, \mu_1 v_1 + \mu_2 v_2) = \mu_1 f(u, v_1) + \mu_2 f(u, v_2), \quad \forall \mu_j \in K.$

In other words, a bilinear form on $E \times F$ is an application f from $E \times F$ to K which is linear on K in each of its parameters u and v when the other is fixed. The set of bilinear forms on $E \times F$ is a sub-space of K, denoted as $\mathcal{L}_2(E, F)$.

An example of such a bilinear form extensively used in our context is the dot product, defined as the bilinear form f on $(\mathbb{R}^d, \mathbb{R}^d)$ to $\mathbb{R}$:

$$f(u, v) = \langle u, v \rangle = \sum_{k=1}^{d} u_k\, v_k\,,$$

with u_k (resp. v_k) stands for the k^{th} component of vector u (resp. v).

Definitions. Two vectors $u \in E$ and $v \in F$ are said to be *orthogonal* and we write $u \perp v$ if and only if $f(u,v) = 0$. When $E = F$, a bilinear form f is said to be, $\forall (u,v) \in E^2$:

- *symmetric* if and only if: $f(u,v) = f(v,u)$,
- *antisymmetric* if and only if: $f(u,v) = -f(v,u)$,
- *alternate* if and only if: $f(u,u) = 0$,
- *definite symmetric* if and only if: $f(u,u) = 0 \Longleftrightarrow u = 0$.

Let f be a symmetric bilinear form on E. A family $(u_i)_{i \in I}$ (where I is a set of indices) of vectors of E is said to be, $\forall (i,j) \in I^2$:

- *orthogonal,* if and only if $(i \neq j) \Longrightarrow f(u_i, u_j) = 0\,,$
- *orthonormal,* if and only if $f(u_i, u_j) = \delta_{ij} \quad$ (where $\delta_{ij} = 0\,, \delta_{ii} = 1$).

Matrix form of a bilinear form

Let $a = (a_1, ..., a_n)$ and $b = (b_1, ..., b_p)$ be two bases of E and F, then, we can write, for $i \in [1,n]$, $j \in [1,p]$:

$$f(u,v) = f\Big(\sum_i a_i u_i, \sum_j b_j v_j\Big) = \sum_{i,j} f(a_i, b_j) u_i v_j\,. \qquad (10.1)$$

Definition 10.3 *We call (representative) matrix of the bilinear form f of $\mathcal{L}_2(E,F)$ in the basis a and b, the matrix $\mathcal{M} = [m_{ij}]$ determined by $m_{ij} = f(a_i, b_j)$.*

If U and V are two column-matrices of the components of the vectors $u \in E$ and $v \in F$ in the basis a and b, Relation (10.1) can be expressed as a product of matrices, $\forall (u,v) \in E \times F$:

$$f(u,v) = f(aU, bV) =^t U f(a,b) V = {}^tU\,\mathcal{M}\,V\,. \qquad (10.2)$$

$\mathcal{M}$ is *non-degenerate* if and only if $\mathcal{M} = [m_{ij}]$ is *invertible.*

Definition 10.4 *A basis of E is orthogonal (resp.* orthonormal*) if and only if $\mathcal{M}$ is diagonal (resp. equal to the unit matrix of order n).*

Quadratic forms

Definition 10.5 *A* quadratic form *q on a space vector E is an application from E to its field K such that for each $\alpha \in K$ and $(u,v) \in E^2$:*

1. *$q(\lambda u) = \lambda^2 q(u)$,*
2. *the application F from $E \times E$ to K, $(u,v) \longrightarrow F(u,v) = q(u+v) - q(u) - q(v)$ is bilinear*[1].

[1]Moreover, the application F is symmetric.

If q_1 and q_2 are two quadratic forms on E and if α_1 and $\alpha_2 \in K$, the application $u \longrightarrow \alpha_1 q_1(u) + \alpha_2 q_2(u)$ is also a quadratic form. Therefore, the set $\mathcal{Q}(E)$ of all quadratic forms on E has a structure of vector space.

Let f be an arbitrary bilinear form on E. The function $q(u) = f(u,u)$ is a quadratic form on E. The sole data of q makes it possible to retrieve the symmetric part of f:

$$q_f(u+v) - q_f(u) - q_f(v) = f(u,v) + f(v,u)\,.$$

If $K = \mathbb{R}$, the form F can be replaced by the symmetric bilinear form $f = \frac{1}{2}F$. Thus, we have the following relation:

$$f(u,v) = \frac{1}{2}\left(q(u+v) - q(u) - q(v)\right)\,, \tag{10.3}$$

with $f(u,u) = q(u)$. Therefore, the data of a quadratic form q is equivalent to the data of a symmetric bilinear form $f = \frac{1}{2}\left(q(u+v) - q(u) - q(v)\right)$; the data of the form f determines the form q by the relation $q(u) = f(u,u)$.

The three following relations are commonly used in practice:

1. $q(u+v) = q(u) + q(v) + f(u,v) + f(v,u)$,
2. $q(u-v) = q(u) + q(v) - f(u,v) - f(v,u)$,
3. $q(u+v) - q(u-v) = 2(f(u,v) + f(v,u))$.

The restriction of q to the sub-space $\mathcal{S}_2(E)$ of the symmetric bilinear forms on E induces an isomorphism of $\mathcal{S}_2(E)$ on $\mathcal{Q}(E)$. The inverse image of a quadratic form q by this isomorphism is then called the *polar form* of q. In other words, the symmetric bilinear form defined by Relation (10.3) is the so-called polar form of q. Hence, we have the two following identities (which can be easily deduced from the previous relations):

$$\begin{cases} 2f(u,v) &= q(u+v) - q(u) - q(v)\,,\\ 4f(u,v) &= q(u+v) - q(u-v)\,. \end{cases}$$

Let q be a quadratic form of polar form f. We also have the two following inequalities, for each $(u,v) \in E^2$:

- *Cauchy-Schwartz's inequality*:
  if q is positive, then:$f^2(u,v) \leq q(u)\,q(v)$, the equality is obtained if u and v are colinear and only when the form is positive definite.

- *Minkowski's inequality*:
  if q is positive, then: $\sqrt{q(u+v)} \leq \sqrt{q(u)} + \sqrt{q(v)}$, with equality if $v = 0$ or $\exists \alpha \in \mathbb{R}_+$ such that $x = \alpha v$ and only if the form is positive definite.

Distances and norms

Let us now consider the field of real numbers, $\mathbb{R}$. We call *distance* any application $d : \mathbb{R} \times \mathbb{R} \longmapsto \mathbb{R}_+$ such that $\forall (x,y) \in \mathbb{R}^2$:

$$\begin{array}{rcl} x = y & \Longleftrightarrow & d(x,y) = 0 \\ d(x,y) & = & d(y,x) \\ d(x,y) & \leq & d(x,z) + d(z,y)\,. \end{array}$$

On the other hand, we call *norm* on $\mathbb{R}$ any application N from $\mathbb{R}$ to $\mathbb{R}_+$ such that $\forall (x,y) \in \mathbb{R}^2$:

$$\begin{array}{rcl} N(x) = 0 & \Longleftrightarrow & x = 0 \\ N(x+y) & \leq & N(x) + N(y) \\ N(\lambda x) & = & \lambda\, N(x)\,. \end{array}$$

Each vector of norm 1 is a so-called *unit vector.*

For instance, the three following applications are norms on $\mathbb{R}$:

1. $N_1(x) = \sum\limits_{i=1}^{n} |x_i|$,
2. $N_2(x) = \sqrt{\sum\limits_{i=1}^{n} |x_i|^2}$,
3. $N_\infty(x) = \sup |x_i|$,

and we write $N_i(x) = \|x\|$. We will then show which relation exists between a symmetric positive bilinear form and a norm or a distance.

Norm associated with a quadratic form. Let us consider a positive symmetric bilinear non-degenerate form, f on $\mathbb{R}$ and let q be the associated quadratic form. We are now trying to establish that the application $\sqrt{q(x)}$ is a norm on $\mathbb{R}$. We have already noticed that (Minkowski's inequality):

$$\sqrt{q(x+y)} \leq \sqrt{q(x)} + \sqrt{q(y)}\,.$$

Moreover, by definition the following relation holds:

$$q(\lambda\, x) = \lambda^2\, q(x)$$

we can then deduce that:

$$\sqrt{q(\lambda\, x)} = |\lambda|\, \sqrt{q(x)}\,.$$

As the form f is non-degenerate, we have a third relation:

$$q(x) = 0 \Longleftrightarrow x = 0\,,$$

and therefore we can deduce that the application $x \longmapsto \sqrt{q(x)}$ is a norm on $\mathbb{R}$.

Dot product. Any non-degenerate symmetric positive bilinear form f on $\mathbb{R}$ can be written in an orthonormal basis:

$$f(x,y) = \sum_{i=1}^{n} x^i\, y^i\,, \qquad \text{or also} \qquad q(x) = f(x,x) = \sum_{i=1}^{n} (x^i)^2\,.$$

Among all the possible forms, we pick one particular form f the so-called *dot product* on $\mathbb{R}$ which is written as:

$$f(x,y) = \langle x, y\rangle\,.$$

Thus introduced, the norm $x \longrightarrow \sqrt{\langle x,x\rangle}$ is called the *Euclidean norm.* This obviously means that the associated quadratic form q is the square of the norm:

$$q(x) = \langle x,x\rangle = \|x\|^2\,.$$

Distance between two points. Using the Euclidean norm, we can define the distance between two points $(P,Q) \in \mathbb{R}^2$ as follows:

$$d(P,Q) = \|P - Q\|\,. \tag{10.4}$$

By extension, we also say that $\|x\|$ is the *length* of vector x.

We have thus the two following classical results in $\mathbb{R}$:

- *Cauchy-Schwartz's inequality*: $\|x+y\| \leq \|x\| + \|y\|$ and
- *Pythagorus's theorem*: $\left\|\sum_{i=1}^{n} x_i\right\|^2 = \sum_{i=1}^{n} \|x_i\|^2\,.$

Matrix form of a quadratic form

A quadratic form q on E is so-called *positive* (resp. *strictly positive*) and denoted $q \geq 0$ (resp. $q > 0$), if $q(u) \geq 0$ (resp. $q(u) > 0$) for each u (resp. $u \neq 0$) of E.

Hence, whatever the basis $a = (a_1, ..., a_n)$ of E, the determinant of the matrix $\mathcal{M} = [f(a_i, a_j)]$ of q in this basis is positive or zero (resp. strictly positive) if $q \geq 0$ (resp. $q > 0$).

A symmetric matrix $\mathcal{M} = [f_{ij}]$ on (the ordered field) K is called *positive* if for any column-vector $U \neq 0$:

$$q(U) =^t U\mathcal{M}U = \sum_{i,j} f_{ij}u_iu_j = \sum_{i} f_{ii}(u_i)^2 + 2\sum_{i<j} f_{ij}x_iu_j$$

is positive. In other words, $\mathcal{M}$ is the matrix of a positive quadratic form q on K.

Notice also that the eigenvalues of a symmetric operator are real numbers. Any symmetric operator is diagonalizable in an orthogonal basis. This means that there is always an orthogonal basis of eigenvectors for any symmetric operator. We will see an important application of this property concerning the diagonalization of two quadratic forms.

Having recalled these classical results of linear algebra, it is now possible to introduce the notions of length and metric which are commonly involved in mesh generation and mesh modification algorithms.

10.2 Distances and lengths

We have seen in Section 10.1 that the data of a symmetric bilinear form $f(u,v) = \frac{1}{2}(q(u+v) - q(u) - q(v))$ is equivalent to the data of a quadratic form q on $\mathbb{R}$. From the notion of metric space, which allows us to define the distance between two points, we will show how to compute the length of a segment.

Classical length

The notion of length in a metric space is related to the notion of metric and, thus, to a suitable definition of the dot product in the given vector space.

Notion of metric. Assume that at any point P of $\mathbb{R}^d$ a metric tensor is given, as a $(d \times d)$ symmetric definite positive matrix $\mathcal{M}(P)$, (i.e., non-degenerate). For example, in two dimensions, we consider:

$$\mathcal{M}(P) = \begin{pmatrix} a & b \\ b & c \end{pmatrix}, \tag{10.5}$$

such that $a > 0$, $c > 0$ and $ac - b^2 > 0$, for $a, b, c \in \mathbb{R}$ (notice that these values depend on P, i.e., $a = a(P)$, etc.). If the field of tensors thus defined is known, it induces a Riemannian structure over $\mathbb{R}^d$.

Remark 10.1 *In the case where $\mathcal{M}(P)$ does not depend on P, the matrix thus defined has real coefficients and we again find the classical Euclidean case (where the metric is independent of the position).*

Dot product. The dot product of two vectors in the classical Euclidean space for a given metric $\mathcal{M}(P)$ can be defined as:

$$\langle u, v\rangle_{\mathcal{M}(P)} = u\,\mathcal{M}(P)\,v\,, \tag{10.6}$$

and therefore, considering the Euclidean norm introduced in the previous section, the norm of a vector u is given by the relation:

$$\|u\| = \sqrt{\langle u, u\rangle_{\mathcal{M}(P)}} = \sqrt{{}^t u\,\mathcal{M}(P)\,u}\,. \tag{10.7}$$

Notion of length (general case). Having recalled the notions of metric and dot product, we will now introduce the notion of the length of a vector. In the Euclidean space $\mathbb{R}^2$ or $\mathbb{R}^3$, supplied with the Euclidean norm, we have seen that:

$$f(u,v) = \langle u, v\rangle \quad \text{i.e.,} \quad q(u) = \langle u, u\rangle = \|u\|^2\,,$$

which allows us to see that $\mathcal{M} = I_d$ (as compared with the above definition of the dot product). This will allow us to compute the length of any vector u, which is indeed the distance between the two endpoints of this vector, using the norm:

$$\|u\| = \sqrt{{}^t u\,\mathcal{M}(P)\,u}\,. \tag{10.8}$$

To summarize, given a quadratic form q, computing a length consists of using the dot product associated to q. Hence, the dot product can be formally written as: $\langle .,. \rangle_q$ or also as $\langle .,. \rangle_{\mathcal{M}}$ where $\mathcal{M}$ is the matrix (symmetric positive definite) associated with the quadratic form q.

Notion of length (particular case). The length notion can be also retrieved in differential geometry, as will be seen in the following chapter. Let us consider the space supplied with a Riemannian structure induced by a metric $\mathcal{M}_\gamma$. We consider the curve γ that is the shortest path between two given points A and B. Such a curve is a so-called *geodesic*. Assume a parameterization $\gamma(t)$ of the arc γ of class C^k ($k \geq 1$) is known, such that $\gamma(0) = A$ and $\gamma(1) = B$. Then, the *length* $L(\gamma)$ of the arc is defined as:

$$L(\gamma) = \int_0^1 \|\gamma'(t)\| dt = \int_0^1 \sqrt{{}^t\gamma'(t)\, \mathcal{M}_\gamma\, \gamma'(t)}\, dt\,. \qquad (10.9)$$

We then call the *distance* between two points, the lower bound of the length of the curves connecting these points. Computing $L(\gamma)$ requires knowing $\gamma(t)$, which turns out to be difficult in practice. That is why we consider the case where the metrics are independent of the position (which reduces the problem to the classical Euclidean case, cf. Remark (10.1)), for which the geodesics are straight (line) segments.

So, the restriction of a paremetrized arc $\gamma(t)$, $t \in [a,b]$ to a vector $\overrightarrow{AB}$ with the parameterization $\gamma(t) = A + t\overrightarrow{AB}$, $t \in [0,1]$ and $\gamma(0) = A, \gamma(1) = B$ allows us to write the length $L(\gamma)$ of the segment as:

$$L(\gamma) = \int_0^1 \sqrt{{}^t\gamma'(t)\, \mathcal{M}_\gamma\, \gamma'(t)}\, dt \qquad (10.10)$$

where M_γ represents the metric specification along γ. Hence, noticing that $\gamma'(t) = \overrightarrow{AB}$, we have:

$$L(\gamma) = \int_0^1 \sqrt{{}^t\overrightarrow{AB}\, \mathcal{M}_\gamma\, \overrightarrow{AB}} dt\,.$$

Writing $\mathcal{M}_\gamma = \mathcal{M}$ (i.e., the metric is independent of the position), we obtain the relation:

$$L(\gamma) = \sqrt{{}^t\overrightarrow{AB}\, \mathcal{M}\, \overrightarrow{AB}}\,.$$

And, in the particular case where $\mathcal{M}_\gamma = I_d$, we finally have:

$$L(\gamma) = \int_0^1 \|\gamma'(t)\| dt = \sqrt{\langle \overrightarrow{AB}, \overrightarrow{AB} \rangle} = \|\overrightarrow{AB}\|\,,$$

which is obviously the expected result.

Unit length

The key is now to define a way to compute the lengths in the case where various metrics (i.e., different from the above classical Euclidean case) are specified. Thus, we want to change the definition of q (or that of $\mathcal{M}$) resulting in a different expression for $\langle .,\rangle_q$ (or $\langle .,.\rangle_{\mathcal{M}}$).

Notice firstly that two situations will be of particular interest. They are related to the way q (or $\mathcal{M}$) is defined. The first case corresponds to the Euclidean case where q is defined in a global fashion. On the other hand, cases where q depends on the spatial position (say $q = q_t$ or, similarly, $\mathcal{M} = \mathcal{M}(t)$) lead to a Riemannian context.

Unit length. Let us consider a given basis $a^k, k = 1, d$ of unit vectors in $\mathbb{R}^d$ and d positive real values λ_k. We want to define a metric $\mathcal{M}$ where the norm, denoted by $||.||_{\mathcal{M}}$, is such that satisfying the relation $||u||_{\mathcal{M}} = 1$ means that vector u conforms to the pairs (λ_k, u^k)'s.

To make these notions more precise, we first give some simple examples, then we introduce the general notion of unit length.

In the first example, we want to define segments (vectors) of constant length h, irrespective of the direction. This problem is *isotropic* by nature. Indeed, the geometric locus of all points P distant from h from a given point O is a circle (a sphere), centered at O of radius h.

In practice, we want to define metric $\mathcal{M}$ such that:

$$||\overrightarrow{OP}||_{\mathcal{M}} = 1 \iff ||\overrightarrow{OP}|| = h\,.$$

Notice (cleverly) that the diagonal matrix Λ having all its coefficients λ_k equal to $\frac{1}{h}$ leads to a matrix:

$$\mathcal{M} = \Lambda^2 = \frac{1}{h^2} I_d\,,$$

which is a solution of the previous equation. Indeed, using Relation (10.8), with $u = \overrightarrow{OP}$ and $||\overrightarrow{OP}|| = h$, we obtain:

$$||\overrightarrow{OP}||_{\mathcal{M}} = \sqrt{{}^t\overrightarrow{OP}\,\mathcal{M}\,\overrightarrow{OP}} = \sqrt{{}^t\overrightarrow{OP}\,\Lambda^2\,\overrightarrow{OP}} = \sqrt{{}^t\overrightarrow{OP}\,\frac{I_d}{h^2}\,\overrightarrow{OP}} = \frac{||\overrightarrow{OP}||}{h} = 1\,.$$

In fact, according to Relation (10.2), we can observe that the metric defined in this way corresponds to a circle (a sphere). Let consider the bilinear form

$$f(u,v) = \frac{{}^t uv}{h^2}\,.$$

Then, for example in $\mathbb{R}^2$, we have the relation:

$$f(u,u) = \frac{u_x^2 + u_y^2}{h^2}\,,$$

which, in the case where $f(u,u) = 1$ defines a circle of radius h.

Let us now consider an *anisotropic* example. More precisely, if e_k $(k = 1, d)$ denotes a vector of the canonical basis of $\mathbb{R}^d$, we want to define segments (vectors) of length h_k in the direction e_k.

Let Λ be the $d \times d$ diagonal matrix in which all diagonal coefficients λ_k are set to $\frac{1}{h_k}$. Then, we define a transformation $\mathcal{T}$ such that

$$\mathcal{T}(e_k) = e_k \,, \text{for} \;\; k = 1, .., d\,.$$

We introduce the matrix $\mathcal{M} = {}^t\mathcal{T}\Lambda\Lambda\mathcal{T}$ and we define

$$\|u\|_{\mathcal{M}} = \sqrt{{}^tu\,\mathcal{M}\,u}\,.$$

Hence, given a point O, the points P such that $\|\overrightarrow{OP}\|_{\mathcal{M}} = 1$ are within an ellipse (an ellipsoid) centered at O, aligned with the e_ks, whose radii are the h_ks. Similarly to the isotropic case, we find, for instance, in two dimensions:

$$f(u, u) = \frac{u_x^2}{h_x^2} + \frac{u_y^2}{h_y^2}\,,$$

which is actually the expected classical result.

Finally, the last example corresponds to the general anisotropic case. In this case, segments (vectors) of length h_k are desired in the direction a_k. Following the same scheme as in the previous example, the transformation $\mathcal{T}$ is now such that:

$$\mathcal{T}(a_k) = e_k \,, k = 1, .., d \quad \text{and similarly,} \quad \mathcal{T}^{-1}(e_k) = a_k \,, k = 1, .., d\,.$$

Then, the relation $\|\overrightarrow{OP}\|_{\mathcal{M}} = 1$ defines an ellipse (an ellipsoid) centered at O, aligned with the a_k and such that the radii are the given h_k's.

A simple way of proving this is to associate a point P' with each point P using the relation $OP = \mathcal{T}^{-1}\,OP'$. Then, $\|\overrightarrow{OP}\|_{\mathcal{M}} = 1$ leads to writing the relations:

$$1 = \sqrt{{}^t\overrightarrow{OP}\mathcal{M}\overrightarrow{OP}} = \sqrt{{}^t\overrightarrow{OP}\,{}^t\mathcal{T}\Lambda^2\mathcal{T}\overrightarrow{OP}} = \sqrt{{}^t\overrightarrow{OP'}\,{}^t\mathcal{T}^{-1}\,{}^t\mathcal{T}\Lambda^2\mathcal{T}\mathcal{T}^{-1}\,\overrightarrow{OP'}}\,,$$

which can be reduced to the relation:

$$1 = \sqrt{{}^t\overrightarrow{OP'}\Lambda^2\,\overrightarrow{OP'}}\,,$$

which is indeed the relation showing that point P' belongs to an ellipse (in two dimensions) centered at O, aligned with the e_k and whose radii are the h_k. Then, using the relation relating points P and P', it is easy to see that P belongs (in two dimensions) to the ellipse with the same center and the same radii, but now aligned with the a^k.

Remark 10.2 *It could be noticed that the above general form can be reduced to the first two cases, provided a suitable choice of Λ and of $\mathcal{T}$. So, in the first example, $\mathcal{T}$ is the identity matrix I_d and $\Lambda^2 = \frac{I_d}{h^2}$. In the second example, we again have $\mathcal{T} = I_d$ while Λ is the diagonal matrix whose coefficients are h_k^{-1}.*

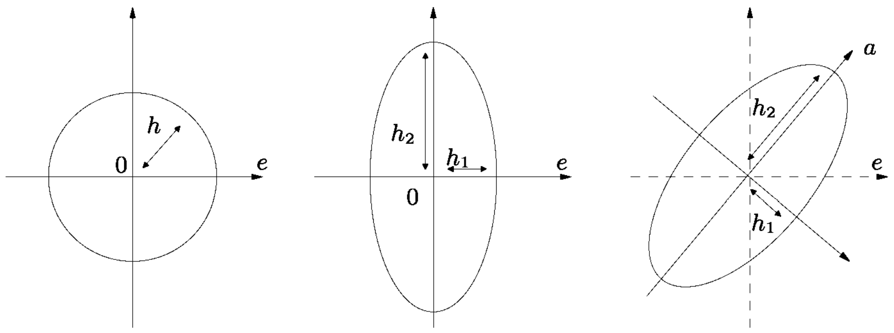


Figure 10.1: *The geometric interpretation of the various metrics. An isotropic metric leads to a circle (left-hand side), an anisotropic metric leads to an ellipse (aligned with the canonical basis, middle, or aligned with any arbitrary orthogonal vectors, right-hand side).*

A global definition. In this case, the metric (i.e., q or $\mathcal{M}$) is globally defined, thus meaning that the notion of unit length is the same at any point location. Computing a length is then easy since the matrix involved in such a calculation is a constant one.

A local definition. In this case, the metric varies from point to point and the notion of unit length is different according to the spatial position. Actually, if we consider the matrix $\mathcal{M}$, this matrix is a function of the current position and thus can be expressed as $\mathcal{M}(t)$, where t is a parameter value. Unlike the previous case, the calculation of a length is more tedious. In practice, the matrix involved in the formula is no longer constant, thus leading us to consider approximate solutions for the length calculation.

Applications

Let us consider a given specification. This can be expressed in terms of sizes or in terms of directional features and related sizes. Then, the previous material enables us to characterize this specification as a unit length defined in a suitable space.

Actually, two categories of metric specifications can be exhibited, each of which includes two classes of definitions. Independently of this classification, the metrics are either isotropic or anisotropic by nature.

We have already mentioned that the case where the metric is constant over the space is equivalent to the classical Euclidean situation. When the metric varies from point to point (i.e., is not constant from one point to another), the context of the study is then Riemannian (actually, the field of tensors induces a Riemannian structure over $\mathbb{R}^d$). The length calculation then requires a special effort.

10.3 Metric-based operations

In this section, we will discuss various metric-based manipulation methods. Indeed, in many applications, several metrics can be defined at any point location (of the computational domain). We will then consider different ways of going back to the specification of a unique metric. Actually, the metric-based operations are considered from the point of view of operations dealing with the associated quadratic forms.

Simultaneous reduction of two metrics

Given two metrics $\mathcal{M}_1$ and $\mathcal{M}_2$ (or similarly two quadratic forms q_1 and q_2), the problem here is to express these two metrics in a basis where the associated matrices are both diagonal. In general, no such basis e exists that is orthogonal for both q_1 and q_2. Such a basis is one of eigenvectors for the operator $f_1^{-1} f_2$ (f_1 and f_2 being the linear applications associated with q_1 and q_2), hence a basis allowing us to diagonalize this operator.

To this end, we can discuss either from the quadratic forms or from the matrices related to the given metrics. For the sake of convenience, we will follow the second approach.

Let us denote by $\mathcal{M}_1$ and $\mathcal{M}_2$ the two $d \times d$ matrices related to the two given quadratic forms. The simultaneous reduction of two positive quadratic forms is possible as soon as one of them is defined, which means that the associated matrix is invertible. Assume then that $\mathcal{M}_1$ is positive definite[2].

To obtain a basis where both $\mathcal{M}_1$ and $\mathcal{M}_2$ are diagonal, we introduce the matrix $\mathcal{N}$ defined as:

$$\mathcal{N} = \mathcal{M}_1^{-1} \mathcal{M}_2 .$$

The matrix $\mathcal{N}$ being $\mathcal{M}_1$-symmetric, it can thus be diagonalized. Let e_1 and e_2 be the two eigenvectors of $\mathcal{N}$. These vectors define a basis of $\mathbb{R}^d$ and we can write:

$${}^t e_2 \, \mathcal{M}_1 \, e_1 \; = \; {}^t e_2 \, \mathcal{M}_2 \, e_1 = 0 \, ,$$

thus meaning that e_1 and e_2 are $\mathcal{M}_i$-orthogonal ($i = 1, 2$). To establish this property, we consider λ_1 and λ_2 the two eigenvectors associated with the two previously defined eigenvectors. Then, we have the relations:

$$\mathcal{N} e_1 = \lambda_1 e_1 \quad \text{as well as} \quad \mathcal{N} e_2 = \lambda_2 e_2 \quad \text{i.e.,}$$

$$\mathcal{M}_1^{-1} \mathcal{M}_2 \, e_1 = \lambda_1 e_1 \quad \text{and} \quad \mathcal{M}_1^{-1} \mathcal{M}_2 \, e_2 = \lambda_2 e_2 \, .$$

By applying $\mathcal{M}_1$ to the left, the previous relations become:

$$\mathcal{M}_2 \, e_1 = \lambda_1 \mathcal{M}_1 \, e_1 \quad \text{and} \quad \mathcal{M}_2 \, e_2 = \lambda_2 \mathcal{M}_1 \, e_2 \, . \tag{10.11}$$

From the first relation we deduce:

$${}^t e_2 \mathcal{M}_2 \, e_1 = \lambda_1 \, {}^t e_2 \mathcal{M}_1 \, e_1 \, ,$$

[2]Trivially, we will not consider the case where the two matrices are linked by a relation like $\mathcal{M}_1 = \alpha \mathcal{M}_2$.

and from the second one:

$$^t e_1 \mathcal{M}_2 \, e_2 = \lambda_2 \, ^t e_1 \mathcal{M}_1 \, e_2 \,,$$

whose transpose is:

$$^t e_2 \mathcal{M}_2 \, e_1 = \lambda_2 \, ^t e_2 \mathcal{M}_1 \, e_1 \,,$$

then, by identification, we obtain:

$$\lambda_1 \, ^t e_2 \mathcal{M}_1 \, e_1 = \lambda_2 \, ^t e_2 \mathcal{M}_1 \, e_1 \,,$$

which finally implies that:

$$^t e_2 \mathcal{M}_1 \, e_1 = \, ^t e_2 \mathcal{M}_2 \, e_1 = 0 \,.$$

Moreover, Relation (10.11) leads to:

$$^t e_1 \mathcal{M}_2 \, e_1 = \lambda_1 \, ^t e_1 \mathcal{M}_1 \, e_1 \quad \text{and} \quad ^t e_2 \mathcal{M}_2 \, e_2 = \lambda_2 \, e_2 \mathcal{M}_1 \, e_2 \,,$$

which can also be written as follows:

$$q_2(e_1) = \lambda_1 q_1(e_1) \quad \text{and} \quad q_2(e_2) = \lambda_2 q_1(e_2) \,,$$

by using the two corresponding quadratic forms.

Expression of a vector in the eigenbasis. In the basis defined by $[e_i, e_2]$, any vector v can be written as $v = x_1 e_1 + x_2 e_2$. The two quadratic forms $q_1(v)$ and $q_2(v)$ are represented by two diagonal matrices. This result is left as an exercise:

Exercise 10.1 *Define $\alpha_{i,j}$ the coefficients such that:*

$$^t v \mathcal{M}_1 v = \sum_{i,j} \alpha_{i,j} x_i x_j \,.$$

Prove that $\alpha_{i,j} = 0$ for $i \neq j$ and show that $\alpha_{i,i} = \, ^t e_i \mathcal{M}_1 \, e_i$. Similarly, reconsider the same exercise with the matrix $\mathcal{M}_2$.

Expression of the matrices in the eigenbasis. Let $\mathcal{M}_1^e$ and $\mathcal{M}_2^e$ be the matrices obtained by replacing $\mathcal{M}_1$ and $\mathcal{M}_2$ by the corresponding forms in the eigenbasis.

Exercise 10.2 *Express the transformation defined by the matrix $\mathcal{R} = (e_1, e_2)$, find the expression of $\mathcal{M}_1^e = \, ^t \mathcal{R} \mathcal{M}_1 \mathcal{R}$ and verify that a diagonal matrix results from this operation. Similarly, consider the coefficients of the matrix $\mathcal{M}_2^e$.*

Remark 10.3 *Notice that $\mathcal{M}_1^e$ is I_d, the identity matrix, if we normalize e_1 and e_2 with respect to $\mathcal{M}_1$. In this case we have, for the corresponding quadratic forms:*

$$q_1(e_1) = q_1(e_2) = 1 \quad \textit{as well as} \quad q_2(e_1) = \lambda_1 \quad \textit{and} \quad q_2(e_2) = \lambda_2 \,.$$

Metric interpolation

For the sake of clarity, we only consider here the two dimensional case. Let P_1 and P_2 be two points in $\mathbb{R}^2$ and let $\mathcal{M}_1$ and $\mathcal{M}_2$ be two metrics associated with these points. The problem here is to find a metric $\mathcal{M}(t)$ defined on the segment $[P_1, P_2] = [P_1 + t\,(P_2 - P_1)]$ for each t in $[0,1]$. Moreover, the desired metric must be such that:

$$\mathcal{M}(0) = \mathcal{M}_1 \quad \text{and} \quad \mathcal{M}(1) = \mathcal{M}_2\,,$$

and must vary monotonously between these two values.

Achieving such a metric is actually equivalent to performing a *metric interpolation.* To this end, various techniques can be considered.

An intuitive method. To give the idea of this kind of method, we consider the isotropic situation. The desired solution can then be obtained trivially. Indeed, if the metrics are simply λI_d and μI_d, then the expected sizes are respectively $h(0) = 1/\sqrt{\lambda}$ for $\mathcal{M}_1$ (at point P_1) and $h(1) = 1/\sqrt{\mu}$ for $\mathcal{M}_2$ (at point P_2). Hence, assuming that an arithmetic (linear) size distribution is specified, the interpolation function is defined as follows:

$$\mathcal{M}(t) = \frac{1}{(h(0) + t(h(1) - h(0)))^2}\, I_d\,, \qquad 0 \le t \le 1\,, \tag{10.12}$$

with $\mathcal{M}(0) = \mathcal{M}_1$ and $\mathcal{M}(1) = \mathcal{M}_2$.

Notice that other types of distributions can be considered, for instance, a geometric distribution (see below).

In the anisotropic case, several approaches can be considered. By analogy with the isotropic case where the metric is usually written as $\mathcal{M} = \frac{1}{h^2}\, I_d$, we observe that the variation related to the h's is "equivalent" to the variation related to the $\mathcal{M}^{-1/2}$'s. Hence, we obtain the following interpolation scheme:

$$\mathcal{M}(t) = \left((1-t)\mathcal{M}_1^{-1/2} + t\mathcal{M}_2^{-1/2}\right)^{-2}, \qquad 0 \le t \le 1\,. \tag{10.13}$$

Computing $\mathcal{M}^{-1/2}$ requires evaluating the eigenvalues of $\mathcal{M}$, which is tedious. To avoid this problem, we can consider the interpolation as:

$$\mathcal{M}(t) = ((1-t)\mathcal{M}_1^{-1} + t\mathcal{M}_2^{-1})^{-1}\,, \qquad 0 \le t \le 1\,, \tag{10.14}$$

and notice that this relation emphasizes the smallest sizes (i.e., the weakest values of h).

The interpolation scheme based on a metric exponent is properly defined. Actually:

- if $\mathcal{M}$ is a metric, then $t\mathcal{M}^{\alpha}$ is also a metric, when $t > 0$ and α are two arbitrary real values;
- if $\mathcal{M}_1$ and $\mathcal{M}_2$ are two metrics, $\mathcal{M}_1 + \mathcal{M}_2$ is also a metric.

Proving these results requires only making sure that, in each case, the resulting matrices are symmetric and positive definite.

Notice, however, that this kind of intuitive interpolation presents some weaknesses. In particular, the variations in terms of h cannot be explicitly controlled. Thus, in the following, we consider the *simultaneous reduction* of two metrics. Conversely (see below) the intuitive method gives a solution when the interpolation is performed on a triangle and not only along a line.

A method based on the simultaneous reduction. The interpolation metric is obtained after a two-step algorithm:

Step 1: using the above simultaneous reduction, we write both $\mathcal{M}_1$ and $\mathcal{M}_2$ in a diagonal form.

Step 2: according to the interpolation between P_1 and P_2, we complete the desired interpolation between the metrics.

Thus, let e_1 and e_2 be the two eigenvectors of $\mathcal{N} = \mathcal{M}_1^{-1}\mathcal{M}_2$, the eigenvalues of the metric $\mathcal{M}_1$ are the λ_i's such that $(\lambda_i = {}^te_i\mathcal{M}_1e_i)_{i=1,2}$ and that of the metric $\mathcal{M}_2$, the μ_i's such that $(\mu_i = {}^te_i\mathcal{M}_2e_i)_{i=1,2}$. Any vector $X = x_1e_1 + x_2e_2$ in $\mathbb{R}^2$, written in the basis $[e_1, e_2]$, is such that:

$$ {}^tX\mathcal{M}_1X = \lambda_1x_1^2 + \lambda_2x_2^2 \quad \text{and} \quad {}^tX\mathcal{M}_2X = \mu_1x_1^2 + \mu_2x_2^2 . $$

Now, we define $(h_{1,i} = \frac{1}{\sqrt{\lambda_i}})_{i=1,2}$ and $(h_{2,i} = \frac{1}{\sqrt{\mu_i}})_{i=1,2}$. Then, the value $h_{1,i}$ (resp. $h_{2,i}$) is the unit length in the metric $\mathcal{M}_1$ (resp. $\mathcal{M}_2$) along the axis e_i and the interpolation metric between $\mathcal{M}_1$ and $\mathcal{M}_2$ is defined using the formula:

$$ \mathcal{M}(t) = {}^t\mathcal{P}^{-1} \begin{pmatrix} \dfrac{1}{H_1^2(t)} & 0 \\ 0 & \dfrac{1}{H_2^2(t)} \end{pmatrix} \mathcal{P}^{-1} \qquad t \in [0,1], $$

where $\mathcal{P}$ is the matrix formed by the column-vector (e_1, e_2), and $(H_1(t), H_2(t))$ are two monotonous continuous functions such that $H_i(0) = h_{1,i}$ and $H_i(1) = h_{2,i}$ for $i = 1, 2$. To complete the definition of this interpolation, we have still to express the terms $H_i(t)$.

Depending on the expected result, various choices can be made. In practice, we can consider the following interpolation functions:

- a linear function: $H_i(t) = h_{1,i} + t\,(h_{2,i} - h_{1,i})$,
- a geometric function: $H_i(t) = h_{1,i} \left(\dfrac{h_{2,i}}{h_{1,i}}\right)^t$,
- a sinusoïdal function: $H_i(t) = \frac{1}{2}\Big(h_{1,i} + h_{2,i} + (h_{1,i} - h_{2,i})\cos(\pi\, t)\Big)$.

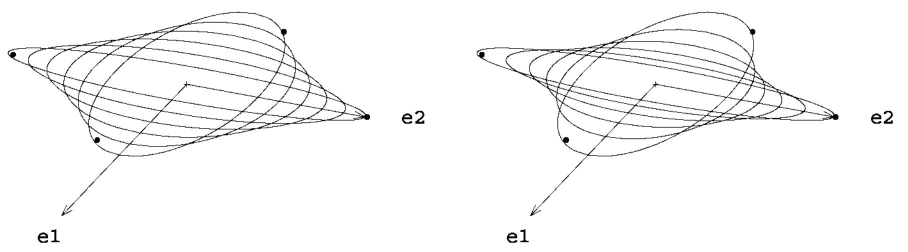


Figure 10.2: *Metric interpolation: continuous variation of the metric between $\mathcal{M}_1$ and $\mathcal{M}_2$. Left-hand side, linear interpolation; right-hand side, geometric interpolation.*

Notice that this interpolation is only controlled along the directions of the axes e_1 and e_2. As an example, we present Figure 10.2 which illustrates the two initial metrics (represented with small dots) and the interpolated metrics in the case of a linear function (left-hand side) and of a geometric function (right-hand side).

Note also that the previous discussion, in two dimensions, extends to three dimensions.

Remark 10.4 *The metric interpolation method by means of simultaneous reduction is unlikely to be suitable when looking for the solution inside a triangle. Indeed, it is well suited to "edge"-type interpolation, i.e., the metrics at the edge endpoints are given and the metric at any point of this edge is sought. For a triangle interpolation, more intuitive methods give reasonable solutions.*

Metric intersection

Now we face a different kind of problem. Given a point P, we assume that several metrics $\mathcal{M}_i$ are supplied at this point. The problem here is to find a unique metric $\mathcal{M}$ that somehow reflects, in a sense that we will specify, the nature of the initial metrics.

For the sake of simplicity, we consider only the two-dimensional case, while noting, however, that the relations that will be established also apply in three dimensions (replacing a circle by a sphere, an ellipse by an ellipsoid).

First, we discuss the case where two metrics are supplied, and we consider the unit circles (in fact, ellipses) associated with the two original metrics. The desired solution is then a metric associated with the intersection of these two ellipses. As in general, the result is not an ellipse, we can consider one of the ellipses that fits in this intersection area. In this way, we define a so-called *intersection metric*. According to the choice of the ellipse contained in this intersection region, different solutions can be obtained. One solution consists of considering the largest ellipse, while another attempts to preserve some features (for instance, directional) of one of the two initial ellipses. This leads to two solutions which are discussed below.

Metric intersection using the simultaneous reduction scheme. The simultaneous reduction of the two quadratic forms corresponding to two metrics leads to defining the intersection metric related to the two initial metrics as explained in the previous section. Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be two metrics, the two corresponding unit circles can be expressed in the base associated with the simultaneous reduction of the matrices $\mathcal{M}_1$ and $\mathcal{M}_2$:

$$^tX\mathcal{M}_1X = \lambda_1 x^2 + \lambda_2 y^2 = 1 \quad \text{and} \quad ^tX\mathcal{M}_2X = \mu_1 x^2 + \mu_2 y^2 = 1 \tag{10.15}$$

the intersection metric $(\mathcal{M}_1 \cap \mathcal{M}_2)$ is then defined as:

$$(\mathcal{M}_1 \cap \mathcal{M}_2) = {}^t\mathcal{P}^{-1} \begin{pmatrix} \max(\lambda_1, \mu_1) & 0 \\ 0 & \max(\lambda_2, \mu_2) \end{pmatrix} \mathcal{P}^{-1} \tag{10.16}$$

where $\mathcal{P}$ is the matrix mapping the canonical basis to that associated with the simultaneous reduction of the two metrics. Figure 10.3 (left-hand side) depicts the metric intersection of two given metrics based on the simultaneous reduction.

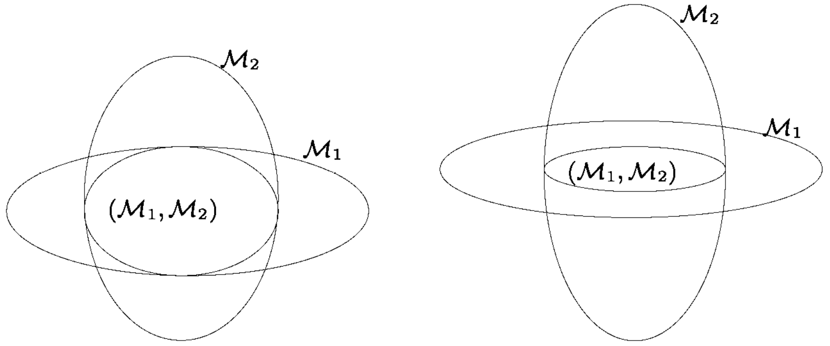

$\mathcal{M}_2$
$\mathcal{M}_1$
$(\mathcal{M}_1, \mathcal{M}_2)$
$\mathcal{M}_2$
$\mathcal{M}_1$
$(\mathcal{M}_1, \mathcal{M}_2)$

Figure 10.3: *Intersection of two metrics $\mathcal{M}_1 \cap \mathcal{M}_2$ based on the simultaneous reduction of the metrics (left-hand side) and preserving the directions of the metric $\mathcal{M}_1$ (right-hand side).*

When several metrics $(\mathcal{M}_i)_{1 \leq i \leq q}$ are specified at a given point, the resulting intersection metric can be defined using the following formula:

$$(\mathcal{M}_1 \cap \cdots \cap \mathcal{M}_q) = ((...((\mathcal{M}_1 \cap \mathcal{M}_2) \cap \mathcal{M}_3) \cap ...) \cap \mathcal{M}_q) \,. \tag{10.17}$$

Exercise 10.3 *Prove that $(\mathcal{M}_1 \cap \mathcal{M}_2)$ defines a metric (hint: check that the relevant properties hold).*

Exercise 10.4 *Is the intersection scheme an associative or commutative scheme?*

Metric intersection preserving specific directions. The previously described method consists of finding the maximal ellipse included in the intersection region of the initial ellipses. Hence, this requirement does not preserve, in any way, the directions of one or the other of the given metrics. As this latter property can be of great interest[3], we are suggesting a different method, this leads to construct a metric having its directions identical to those of one of the initial metrics. Then, a maximal ellipse with particular directions will be found. In the case depicted in Figure 10.3 (right-hand side), the directions specified in metric $\mathcal{M}_1$ are preferred. The intersection metric $(\mathcal{M}_1 \cap \mathcal{M}_2)$ is defined by

$$(\mathcal{M}_1 \cap \mathcal{M}_2) = \omega\, \mathcal{M}_1 \quad \text{with} \quad \omega = \max\left(\frac{\mu_1}{\lambda_1}, \frac{\mu_2}{\lambda_2}, 1\right) \tag{10.18}$$

if we want to preserve the shape of the metric $\mathcal{M}_1$ (μ_i and λ_i also denoting the eigenvalues of the matrices).

Metric smoothing

Given a metric and irrespective of its nature (i.e., related to a geometry or resulting from the physics of the problem considered), there is no guarantee that a mesh strictly based on this metric will conform to the whole set of requirements. Several undesirable features can be encountered. It seems indeed obvious that in two dimensions, it is not possible to obtain a mesh composed of equilateral triangles if the given metric presents great size variations.

Variation and shock of a metric. In the isotropic case, the metric at an arbitrary point P can be written, as seen before, as:

$$\mathcal{M}(P) = \frac{1}{h(P)^2} I_d\,,$$

where I_d is the unit matrix of dimension d and $h(P)$ is the desired size at P. Hence, for an edge AB where we want to have $h(A)$ at A, $h(B)$ at B, the length of the segment is:

$$l(AB) = \|\overrightarrow{AB}\| \int_0^1 \frac{1}{h(t)}\, dt$$

where $h(t)$ represents a continuous interpolation function defined on $[0,1]$ such that $h(0) = h(A)$ and $h(1) = h(B)$. This is equivalent to parameterizing the edge AB by $(1-t)\,A + t\,B$ and to denoting in a similar fashion $h(t)$ and $h(P)$, the value of h at the current point P parameterized by t.

Remark 10.5 *The function h being chosen, the previous expression allows us to go from discrete data (in A and B only) to continuous data (along the whole segment AB).*

As mentioned, $h(A)$ and $h(B)$ can be more or less "compatible" with the Euclidean length of AB. To be able to evaluate this notion numerically, we introduce the following definitions.

[3]For instance, when triangulating some surfaces.

Definition 10.6 *The* h-variation, *denoted v_h, related to an edge AB is defined as:*

$$v_h(AB) = \frac{|h(B) - h(A)|}{\|\overrightarrow{AB}\|}.$$

The h-shock, *denoted χ_h, related to AB is defined as:*

$$\chi_h = \max\left(\frac{h(B)}{h(A)}, \frac{h(A)}{h(B)}\right)^{\frac{1}{l(AB)}}.$$

In other words, the h-variation v_h along AB, when B tends towards A, represents an approximation of the gradient of the function h. The h-shock measures the distortion of h along AB.

For a mesh, these values are defined at each vertex.

Definition 10.7 *Let A be a given mesh vertex and let P_i be the endpoints of the edges incident to A, not equal to A. We set:*

$$v_h(A) = \max_{P_i} v_h(AP_i) \quad \text{and} \quad \chi_h(A) = \max_{P_i} \chi_h(AP_i).$$

This discrete definition enables us to characterize a mesh, according to a given metric field. The h-variation and h-shock values are then defined as the extrema of the values related to these quantities at the mesh vertices.

In the anisotropic case, we define the same notions based on the direction of the given edge. This is equivalent to finding the intersection of the metric in A (resp. in B) with the edge AB and then using the same scheme with $h(A)$ (resp. $h(B)$ defined by $h(A) = \|\overrightarrow{AA_1}\|$ (resp. $h(B) = \|\overrightarrow{BB_1}\|$) where A_1 (resp. B_1) is the intersection point of AB with the circle related to the (anisotropic) metric $\mathcal{M}(A)$ (resp. the circle of $\mathcal{M}(B)$).

Notice, however, that in the anisotropic case, the resulting Riemannian structured is not able to constrain a size variation in each direction.

Metric smoothing using a correction scheme. Given a mesh and a field of metrics defined, in a discrete fashion, at each mesh vertex, the smoothing procedure aims at constructing a (new) field satisfying a given regularity specified *a priori*, whenever the initial field is not compliant. This is especially the case when the size variation is too great or discontinuous.

The new metric is used to reconstruct a new mesh of the domain[4], that is better adapted to the given specifications. In particular, the quality of the resulting mesh is improved, the created elements being more regular (equilateral triangles, for instance).

Let us consider the isotropic case. The problem consists here of bounding the h-variation v_h of an edge AB by a given threshold ε, $v_h < \varepsilon$, by changing the

[4]The given mesh is seen here as a background mesh and forms, along with the smoothed metric, a control space (Chapter 1).

size specifications $h(A)$ and/or $h(B)$. The new specifications are then determined using the following formulas:

$$\begin{aligned} h(A) &= \min(h(A), h(B) + \varepsilon\|\overrightarrow{AB}\|) \\ \text{and} \quad h(B) &= \min(h(B), h(A) + \varepsilon\|\overrightarrow{AB}\|) . \end{aligned}$$

Notice that only one of the size specifications is affected (the largest one).

The procedure is quite similar in the anisotropic case. To this end, it is simply necessary to extend the operator min, related to the sizes, to an operator related to metrics. This operator is, as expected, the metric intersection operator previously described. Notice that in this case, the correction applied to an edge does not account for the metric interpolation along the edge. Moreover, this operation may affect the shape of the corresponding ellipses (ellipsoids).

This procedure, based on the notion of h-shock, in the isotropic and anisotropic cases can be found in [Borouchaki *et al.* 1998].

Examples of metric smoothing. Figure 10.4 illustrates the effects of the metric smoothing and correction procedures on a surface mesh. Figure 10.5 represents a prediction of an unstationary transonic flow around a wing profile. The flow parameters are $Re = 10^7$, $Mach = 0.775$ and the angle of incidence $\alpha = 4°$.

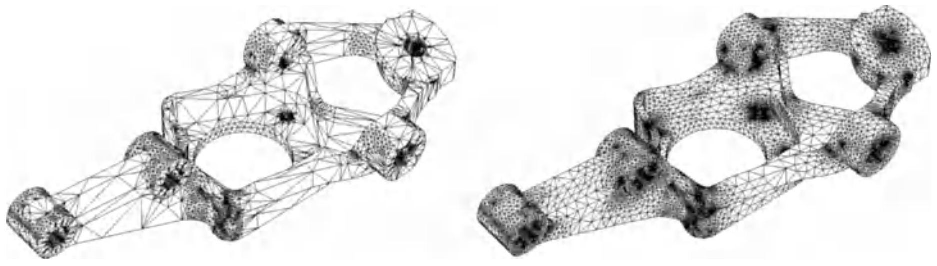


Figure 10.4: *Example of metric correction related to a surface mesh. Left, geometric mesh without correction (data courtesy of the MacNeal-Schwendler Corp.). Right, geometric mesh after a metric correction by a given value* $\varepsilon = 1.5$.

In these examples, the influence of the metric correction procedure is clearly visible. This procedure is even more important in the numerical computations, as the difficulties usually encountered are related to the possible lack of information during the interpolation of the solutions (in an adaptation scheme) and to the capture of the critical regions.

10.4 Metric construction

In the previous sections, we have largely discussed metrics and related operations, on the assumption that these latter were supplied. Now, we give some details on

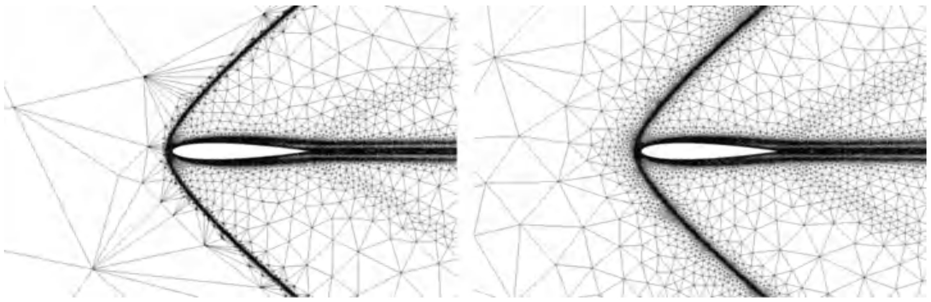


Figure 10.5: *Example of a metric correction on an adapted mesh in a computational fluid simulation. Left, the mesh is adapted without correction. Right, the mesh has been adapted with a metric correction by a given value $\varepsilon = 2$.*

how to define such metrics, especially for surface meshes and regarding a computational scheme based on the finite element method.

10.4.1 Parametric surface meshing

Let Σ be a surface, let σ be its parameterization and let Ω be its parametric space. We want to obtain a surface mesh conforming to some given specifications (in particular, related to the element sizes as well as to the intrinsic properties of Σ, (i.e., to conform to the geometry), from a mesh of Ω. In other words, the goal is to control the mesh of Σ by controlling the mesh of Ω. Chapter 15 will deal with this approach more thoroughly.

Assume now that a metric is given on the surface. Let $\mathcal{M}_3$ be the current associated matrix, of dimensions 3×3. The problem is to find the relationship between a length on Σ and the corresponding length in Ω. Thus defined, the problem can be reduced to that of finding the matrix $\mathcal{M}_2$, a 2×2 matrix, related to $\mathcal{M}_3$. To this end, we use the metric induction.

Metric induction. Given a point $X \in \Omega$, the matrix $\mathcal{M}_2(X)$ is the *metric induced* by $\mathcal{M}_3(P)$, $P \in \Sigma$ on the tangent plane to the surface at P. By denoting $\Pi(P)$ the transition matrix from the canonical basis of $\mathbb{R}^3$ to the local basis at the current point P, the desired metric $\mathcal{M}_2(X)$ is defined by the matrix:

$$\mathcal{M}_2 = \lfloor {}^t\Pi \mathcal{M}_3 \Pi \rfloor_2 ,$$

where the symbol $\lfloor\rfloor_2$ means that we consider the first two columns and the first two lines of the matrix ${}^t\Pi\mathcal{M}_3\Pi$. Given a matrix $\mathcal{M}_3$ we then find by induction a matrix $\mathcal{M}_2$ that enables the lengths on the surface to be controlled via a control of the edge lengths in Ω.

Choice of the surface metrics. The control of the gap between an edge and the surface is obtained using the metric $\mathcal{M}_3$, which has yet to be explained. Hence, to govern the mesh of Σ according to $\mathcal{M}_3$, consists of governing the mesh of Ω with respect to an induced metric $\mathcal{M}_2$. As will be seen in Chapter 15, a judicious choice of $\mathcal{M}_3$ makes it possible to bound the gap between any edge and the surface by a given threshold value ε. A matrix of the form:

$$\mathcal{M}_3(P)_{\rho_1,\rho_2} = {}^t\mathcal{D}(P) \begin{pmatrix} \frac{1}{\alpha^2\,\rho_1^2(P)} & 0 & 0 \\ 0 & \frac{1}{\beta^2\,\rho_2^2(P)} & 0 \\ 0 & 0 & \lambda \end{pmatrix} \mathcal{D}(P)\,, \tag{10.19}$$

where $\mathcal{D}(P)$ corresponds to the principal directions at P, α and β are suitable coefficients and $\lambda \in \mathbb{R}$, enables us to have an *anisotropic* control of the gap between the geometric, which accounts for the two principal radii of curvature ρ_1 and ρ_2 and for the two principal directions.

We can also consider the case $\rho = \min(\rho_1, \rho_2)$ which leads to a so-called metric of the minima radius of curvature for which the matrix $\mathcal{M}_3(P)$ can be expressed as:

$$\mathcal{M}_3(P)_\rho = \begin{pmatrix} \frac{1}{h^2(P)} & 0 & 0 \\ 0 & \frac{1}{h^2(P)} & 0 \\ 0 & 0 & \frac{1}{h^2(P)} \end{pmatrix}, \tag{10.20}$$

where the variable $h(P) = \alpha\rho(P)$ is related to the position and α is a suitable coefficient, related to the geometric approximation (i.e., to the gap between the edges of the discretization and the surface).

10.4.2 Finite element simulation with error control

The previously defined fields of metrics are related to the intrinsic properties of the surface, hence to its geometry. We will now focus on metric fields met during the numerical calculations based on finite element methods, in particular in the case of mesh adaptation (Chapters 21 and 22).

Construction of a computational metric. The goal is to construct a metric that allows an homogenous distribution of the error related to the interpolation of the solutions. The error estimation[5] is analyzed by studying the behavior of $||u - \Pi_h u||$, $\Pi_h u$ being the solution of the discrete model and $||\ \ ||$ being a suitable norm.

Let us assume that a solution $\Pi_h u$ has already been computed on a former mesh. If the interpolation scheme is linear and piecewise continuous (P^1-type), then the interpolation error can be related to the variations of the variables of the problem and, in particular, to their successive derivatives (gradient, Hessian, etc.).

To show this result, we first focus on a one-dimensional problem with only one unknown u.

[5]As well as the convergence of the numerical approximation.

Interpolation error for a one-dimensional P^1 problem. Let us consider the segment $[\Pi_h(a), \Pi_h(b)]$, which is the function Π_h that is the linear approximation of the function u between a and b (between $u(a)$ and $u(b)$, in fact).

In order to evaluate the interpolation error between u and $\Pi_h u$, we will perform a local analysis. In other words, we assume that the computed solution is quite close to the real solution. This evaluation is based on various Taylor's formulae. One such formula, when applied to a regular enough function $f(x)$ from $I = [a, b]$ to $\mathbb{R}$, can be written as:

$$f(a) = f(x) = h\, f'(x) + \frac{h^2}{2}\, f''(x) \,+\, \mathcal{O}(h^3)\,, \tag{10.21}$$

with $x = a + h$. This formula and, in general, formulas of the same type are actually not well suited for our purpose since h is not necessarily small. Thus, we prefer a formula like:

$$f(a) = f(x) = (a - x)\, f'(x) + \frac{h^2}{2}\, f''(x + t(a - x))\,, \tag{10.22}$$

where t, in $[0, 1]$, is a function of both x and a. For our purposes, we fix the function f to be the function that, for $x \in I$ associates $(u - \Pi_h u)(x) = u(x) - \Pi_h u(x)$, thus assuming that $\Pi_h u(a) = u(a)$ and $\Pi_h u(b) = u(b)$. From the previous discussion, regarding u and $\Pi_h u$, we have:

$$u(a) - \Pi_h u(a) = u(x) - \Pi_h u(x) + (a - x)\,(u'(x) - \Pi_h u'(x)) + \frac{(a - x)^2}{2}\, u''(x + t_1(a - x))\,,$$

where now t_1 is a function of a and x. As $u(a) - \Pi_h u(a) = 0$ and as we look for an extremum, x, where $u'(x) - \Pi_h u'(x) = 0$, then we have:

$$0 = (u - \Pi_h u)(x) + \frac{(a - x)^2}{2} u''(x + t_1(a - x))\,.$$

Then, we write the same, based on b. For the above x, we have:

$$0 = (u - \Pi_h u)(x) + \frac{(b - x)^2}{2} u''(x + t_2(b - x))\,.$$

Adding these two relations gives:

$$0 = 2\,(u - \Pi_h u)(x) + \frac{(a - x)^2}{2} u''(x + t_1(a - x)) + \frac{(b - x)^2}{2} u''(x + t_2(b - x))\,.$$

If M is a majorant of u'' in I, then:

$$|(u - \Pi_h u)(x)| \leq \frac{1}{2}\,(\frac{(a - x)^2}{2} + \frac{(b - x)^2}{2})M\,.$$

Then:

$$|(u - \Pi_h u)(x)| \leq \frac{1}{2}\, max_{x \in I}(\frac{(a - x)^2}{2} + \frac{(b - x)^2}{2})\, M\,.$$

And finally, $\forall x \in I,$:

$$|u(x) - \Pi_h u(x)| \leq \frac{(b-a)^2}{8} M \,. \tag{10.23}$$

The goal is then to see what is the value of $|u(x) - \Pi_h u(x)|$, which means finding the value of $\frac{(b-a)^2}{8} M$ and then to compare this value with a given value ε representing the maximum allowed gap between the function u and the linear approximation $\Pi_h u$.

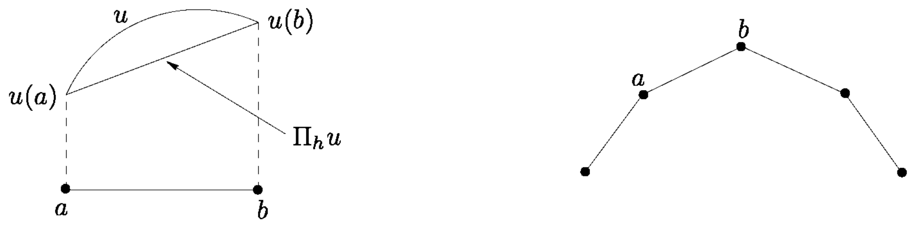


Figure 10.6: *Piecewise linear interpolation in one dimension. Left-hand side: the segment $I = [a,b]$, $u(a)$ and $u(b)$, the function $\Pi_h u$, the segment $[u(a), u(b)]$ and the presumed function u. Right-hand side: the segment ab and its neighbors.*

From a geometric point of view, if x is a point of $[a,b]$, point $(x, \Pi_h u(x))$ describes the segment $[u(a), u(b)]$ while point $(x, u(x))$ describes the (unknown) "curve" u. As u is assumed to be sufficiently regular along $[a,b]$, we will then replace the curve by a parabola. Thus, a method allowing us to reach the expected result consists of:

- constructing a parabola going through the point $\Pi_h u(a) = u(a)$ and through the point $\Pi_h u(b) = u(b)$ and
- evaluating u'' on I, based on the parabola.

This enables us to find the desired value. If the latter is of the desired order, the meshing step is correct according to point a. If for the given ε, we find a larger or smaller value, we can then compute the h that would give the right value and then deduce the metric to be enforced:

$$\mathcal{M} = \frac{I_d}{h^2} \quad \text{with} \quad h^2 = \frac{8\,\varepsilon}{M} \,.$$

In practice, when segment $I = [a,b]$ is analyzed, it is also of interest to look at the neighboring segments so as to guess M. Indeed, in practice, the problem is to find this maximum. The use of a parabola to simulate the function u can then lead to a solution from which the expected size h can be deduced.

Extension to a two-dimensional solution. We also consider a P^1 interpolation in two dimensions. The desired function u is approached by a solution $\Pi_h u$ computed at the triangles vertices of the mesh. Then, we look at the gap (or any other norm) between u et $\Pi_h u$ in each triangle. There are *a priori* two ways of controlling this error:

- either via a control by the edges and we come back to the previous discussion with, however, a second derivative (the Hessian) reduced to these edges,
- or via a control by the gap (or more precisely the L_∞ norm) between the triangle corresponding to the three known values and the (presumed) surface going through these three points. We then face a similar problem to that of the surface meshing.

The first solution is obviously quite rough. By analogy with a surface meshing problem, it consists of controlling the gap between a mesh triangle and the surface based on the sole evaluation of the relative gaps between the triangle edges and the surface. However, this solution gives an initial idea of the control.

The other approach is clearly smarter. It corresponds to what has been suggested in one dimension and, in its principle, can be seen as the direct extension of this approach.

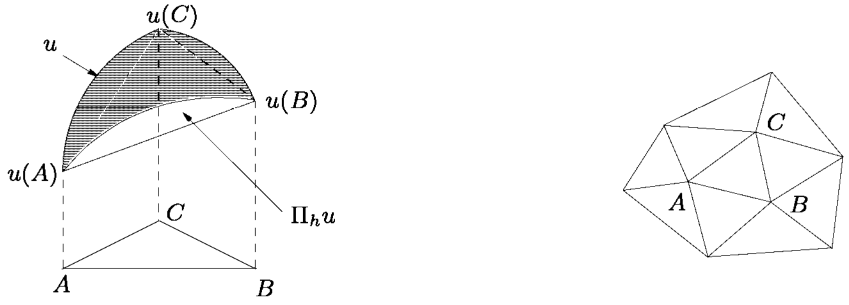


Figure 10.7: *Piecewise linear interpolation in two dimensions. Left-hand side: the triangle $K = [A, B, C]$, $u(A), u(B)$ and $u(C)$, the function $\Pi_h u$, the triangle $[u(A), u(B), u(C)]$ and the function presumed u. Right-hand side: the triangle ABC and its neighbors.*

Let us consider a function f from K to $\mathbb{R}$, where K is an interval of $\mathbb{R}^2$, actually the triangle of vertices A, B and C. For such a function, denoted $f(X)$, we look at how Relation (10.22) writes. We vary X over the triangle K. Under the same assumptions as previously, the local analysis based on a development around X, as seen from A, leads to:

$$f(A) = f(X) + \langle \overrightarrow{XA}, \nabla f(X) \rangle + \frac{1}{2} \langle \overrightarrow{XA}, H_f(X + t_1 \overrightarrow{XA}) \overrightarrow{XA} \rangle , \qquad (10.24)$$

with $\overrightarrow{XA}$ the displacement around A in the triangle K. In this expression, ∇ represents the gradient and $\mathcal{H}_f$ is the Hessian of f.

As in one dimension, we now consider as f the function $u - \Pi_h u$ and we are trying to write the previous relation with respect to u and $\Pi_h u$. Thus, we have:

$$(u-\Pi_h u)(A) = (u-\Pi_h u)(X)+\langle\overrightarrow{XA}\,,\,\nabla(u-\Pi_h u)(X)\rangle+\frac{1}{2}\langle\overrightarrow{XA}\,,\,H_u(X+t_1\overrightarrow{XA})\,\overrightarrow{XA}\rangle\,, \tag{10.25}$$

as $\mathcal{H}_f = \mathcal{H}_u$. Now, let us turn to the term in ∇. To this end, let us assume that X where the extremum occurs falls[6] inside K. Then,

$$\nabla(u-\Pi_h u)(X) = 0\,,$$

and, the above relation reduces to:

$$0 = (u-\Pi_h u)(X) + \frac{1}{2}\langle\overrightarrow{XA}\,,\,H_u(X+t_1\overrightarrow{XA})\,\overrightarrow{XA}\rangle\,. \tag{10.26}$$

Now, we write similar expressions for the same X as now expressed from B and C.

$$0 = (u-\Pi_h u)(X) + \frac{1}{2}\langle\overrightarrow{XB}\,,\,H_u(X+t_2\overrightarrow{XB})\,\overrightarrow{XB}\rangle\,,$$

$$0 = (u-\Pi_h u)(X) + \frac{1}{2}\langle\overrightarrow{XC}\,,\,H_u(X+t_3\overrightarrow{XC})\,\overrightarrow{XC}\rangle\,.$$

Adding these three relations leads to:

$$0 = 3(u-\Pi_h u)(X)+\frac{1}{2}\langle\overrightarrow{XA}\,,\,H_u(X+t_1\overrightarrow{XA})\,\overrightarrow{XA}\rangle+\frac{1}{2}\langle\overrightarrow{XB}\,,\,H_u(X+t_2\overrightarrow{XB})\,\overrightarrow{XB}\rangle$$
$$+\frac{1}{2}\langle\overrightarrow{XC}\,,\,H_u(X+t_3\overrightarrow{XC})\,\overrightarrow{XC}\rangle\,.$$

Let M now be such that:

$$M = max_{Y\in K}\left(\max_{\overrightarrow{vec}\in R^2}\frac{|\langle\overrightarrow{vec}\,,\,H_u(Y)\overrightarrow{vec}\rangle|}{\|\overrightarrow{vec}\|^2}\right)\,.$$

Then:

$$|(u-\Pi_h u)(X)| \le \frac{1}{6}\left(\|\overrightarrow{AX}\|^2+\|\overrightarrow{BX}\|^2+\|\overrightarrow{CX}\|^2\right)M\,.$$

Each point X of K can be written using a linear combination of A, B and C:

$$X = \lambda_a A + \lambda_b B + \lambda_c C\,,$$

with $\lambda_a + \lambda_b + \lambda_c = 1$. It is easy to see that:

$$\overrightarrow{AX} = \lambda_b\overrightarrow{AB}+\lambda_c\overrightarrow{AC}\,,\quad \overrightarrow{BX} = \lambda_c\overrightarrow{BC}+\lambda_a\overrightarrow{BA}\quad\text{and}\quad \overrightarrow{CX} = \lambda_a\overrightarrow{CA}+\lambda_b\overrightarrow{CB}\,,$$

and thus:

$$\|\overrightarrow{AX}\|^2+\|\overrightarrow{BX}\|^2+\|\overrightarrow{CX}\|^2 \le (\lambda_a^2+\lambda_b^2)\|\overrightarrow{AB}\|^2+(\lambda_a^2+\lambda_c^2)\|\overrightarrow{AC}\|^2+(\lambda_b^2+\lambda_c^2)\|\overrightarrow{BC}\|^2$$

[6]If not, X necessarily belongs to an edge of K, and the one-dimensional result holds.

$$+2\lambda_a\lambda_b\,|\langle \overrightarrow{CA}\,,\overrightarrow{CB}\rangle| + 2\lambda_a\lambda_c\,|\langle \overrightarrow{BA}\,,\overrightarrow{BC}\rangle| + 2\lambda_b\lambda_c\,|\langle \overrightarrow{AB}\,,\overrightarrow{AC}\rangle|\,.$$

If L is the length of the largest edge in K, then:

$$\|\overrightarrow{AX}\|^2+\|\overrightarrow{BX}\|^2+\|\overrightarrow{CX}\|^2 \leq 2\,\left(\left(\lambda_a^2+\lambda_b^2+\lambda_c^2\right)+\lambda_a\lambda_b+\lambda_a\lambda_c+\lambda_b\lambda_c\right)L^2.$$

It is easy to see that the extremum is obtained for $\lambda_a = \lambda_b = \lambda_c = \frac{1}{3}$, and therefore, we have:

$$|(u-\Pi_h u)(X)| \leq \frac{2}{9}\,L^2\,M\,. \tag{10.27}$$

The majorant that we give below is certainly not optimal and, specifically, it is isotropic. To improve this majoration, we go back to Formula (10.26).

To this end, we assume that X, the point where the maximum holds, is closer to A than to the two other vertices of K. Then, we write:

$$0=(u-\Pi_h u)(X)+\frac{1}{2}\langle\overrightarrow{AX}\,,\,H_u(X')\,\overrightarrow{AX}\rangle\,, \tag{10.28}$$

where X' lies on AX. Let us introduce A' the point intersection of AX with BC the side opposite vertex A. Then, due to the assumption about X, we have $\lambda \leq \frac{2}{3}$ where λ is such that:

$$\overrightarrow{AX}=\lambda\,\overrightarrow{AA'}\,.$$

The above relation becomes:

$$|(u-\Pi_h u)(X)|=\frac{1}{2}\,\|\langle\overrightarrow{AX}\,,\,H_u(X')\,\overrightarrow{AX}\rangle\,\|\,,$$

$$|(u-\Pi_h u)(X)|=\frac{\lambda^2}{2}\,\|\langle\overrightarrow{AA'}\,,\,H_u(X')\,\overrightarrow{AA'}\rangle\,\|\,,$$

thus,

$$|(u-\Pi_h u)(X)|\leq\frac{2}{9}\,\|\langle\overrightarrow{AA'}\,,\,H_u(X')\,\overrightarrow{AA'}\rangle\,\| \tag{10.29}$$

holds. As a consequence, we have obtained an anisotropic estimate. Note that the same reasoning can be made for B and C and a combination of the corresponding results is used to find the metric information needed for error control and mesh adaptation.

From a geometric point of view, if X is a point of K, point $(X,\Pi_h u(X))$ covers the triangle $[u(A),u(B),u(C)]$ while point $(X,u(X))$ covers the (unknown) "surface" u. As u is supposed sufficiently regular over K, we will replace this surface by a paraboloid. Hence, a method providing the desired result consists of:

- constructing a paraboloid going through the points $\Pi_h u(A)=u(A)$, $\Pi_h u(B)=u(B)$ and $\Pi_h u(C)=u(C)$ and

- this surface being known, evaluating $\mathcal{H}$ on K and finding the desired majorants.

Therefore we vary X to see the majoration values needed to define a suitable $\mathcal{H}^*$ in K using the various $\mathcal{H}$ we have computed[7]. We deduce the values of λ_1 and of λ_2 that correspond to the principal directions of this matrix. The required metric consists of making sure that, given ε, each point X is such that:

$$\langle \overrightarrow{AX}, |\mathcal{H}^*| \overrightarrow{AX} \rangle = \varepsilon .$$

We then find a metric of the form:

$$\mathcal{M} = \mathcal{R} \begin{pmatrix} |\lambda_1| & 0 \\ 0 & |\lambda_2| \end{pmatrix} \mathcal{R}^{-1} ,$$

which, in the coordinate system defined by $\mathcal{R}$, corresponds to the ellipse:

$$\frac{|\lambda_1|}{\varepsilon} x^2 + \frac{|\lambda_2|}{\varepsilon} y^2 = 1 .$$

If X covers or is internal to this ellipse, the interpolation error is smaller than or equal to ε in any direction.

Extension to a three-dimensional solution. The same reasoning applies and the conclusions, in terms of the control, remain unchanged. The constant of error is then $\frac{9}{32}$, see [Frey, Alauzet-2005]. However, the calculations are considerably more technical and the geometric interpretation relies on a surface of $\mathbb{R}^4$ associated with the four vertices of a tetrahedron.

Remark 10.6 *Notice, to conclude this discussion, that the control of the interpolation error suggested here is purely geometric. It does not involve the physics of the problem via its operator. Other operators analyze this point of view. Moreover, other norms may be chosen for the error.*

Hessian computation. Finally, we briefly mention one of the potential numerical problems. One of the difficulties is actually to compute the Hessians involved in the estimate. Such a calculation can be carried out by inventing a surface assumed to represent the desired solution or by more directly numerical methods (Green's formula, for example). More precisely, this approach using *generalized finite differences* is based on Green's formula:

$$\int_\Omega u\, \partial_i v \, d\Omega = - \int_\Omega \partial_i u \, v d\Omega + \int_\Gamma u\, v\, \nu_i \, d\Gamma ,$$

with the usual notations (ν_i being the normal in the i-direction, [Ciarlet-1991]). Setting $u = \partial_i u$, we obtain:

$$\int_\Omega \partial_i u \, \partial_i v \, d\Omega = - \int_\Omega \partial^2_{ii} u \, v d\Omega + \int_\Gamma \partial_i u \, v\, \nu_i \, d\Gamma ,$$

[7]Note that this step is not obvious.

that will allow us to find the diagonal coefficients of the Hessian of u and, similarly, setting $u = \partial_j u$, we find:

$$\int_\Omega \partial_j u \, \partial_i v \, d\Omega = - \int_\Omega \partial^2_{ij} u \, v d\Omega + \int_\Gamma \partial_j u \, v \, \nu_i \, d\Gamma ,$$

which leads to the non-diagonal coefficients of the Hessian.

For an internal point, the boundary term is zero and the integral is calculated on the set of elements sharing the vertex (the ball of this vertex) where the Hessian is sought. Given some simplifying assumptions (Hessian constant per ball), we find, for example, for the coefficient $_{ii}$:

$$\partial_{ii} u = - \frac{\sum_k \int_K \partial_i u \, \partial_i v \, dK}{\sum_k \int_K v \, dK} ,$$

where K is the element of index k in the ball at hand. Hence,

$$\partial_{ii} u = - \frac{\sum_k \alpha_k \, \alpha'_k \, S_k}{\sum_k V_k} ,$$

with:

- α_k the derivative in i of the interpolation of the function u (a plane here, in P^1, the plane going through the points $u(A), u(B)$ and $u(C)$ where A, B and C are the vertices of the considered triangle),
- α'_k the derivative in i of the surface defined over the triangle considered of zero height except at the vertex concerned where the height is equal to 1,
- S_k the area of the triangle of index k,
- V_k the volume of the tetrahedron having the triangle of index k for base and corresponding to the hat function v_k for which the value is 1 at the considered vertex and 0 at the other vertices.

Similarly, we evaluate the terms in $_{ij}$. If the point at which the Hessian is sought is a boundary point, we compute the boundary contribution in the same fashion or, simply, we use an average of the Hessians at the neighboring internal vertices of the current point.

Problem with multiple unknowns. Metric intersection. When the physical problem has many unknowns that are all used to control the adaptation, each of them is used to construct a metric. We therefore face a context in which several fields of metrics are given. The previous section enables us to retrieve the case of a sole metric map, using the metric intersection scheme.

On the other hand, for surface meshes, the geometric metric (of the radii of curvature, for instance) can be intersected with one (or several) computational metrics. Thus, the geometric approximation of the surface can be preserved.

Mesh adaptation. Mesh adaptation (using a h-method; see Chapter 21) can be based on the same principles as discussed previously, i.e., by controlling the interpolation error. Once a solution has been computed, it is analyzed using an error estimate. The previous method provides a way of carrying out this analysis by noticing that the local analysis performed here is meaningless unless a more global analysis justifies its full validity.

Mesh Generation : Application to Finite Elements
Pascal Jean Frey & Paul-Louis George
Copyright © 2008, ISTE Ltd.

Chapter 11

Differential Geometry

Mesh generation of curves and surfaces[1] is an operation known to be tedious to carry out, in a robust fashion, in the context of numerical simulations based on a finite element method as well as in other types of applications. The accuracy of the results in finite element numerical simulations is partly related to the quality of the geometric approximation (i.e., the mesh). Therefore, the mesh of the boundary of a two- or three-dimensional arbitrary domain must have certain properties that are directly related to the geometry it represents.

The construction of a mesh of a curve (resp. surface) requires, in particular, knowledge of the local intrinsic characteristics of the curve (resp surface) such as the curvature, the normal(s), the tangent (the tangent plane), etc. at any point of this geometric support. These geometric characteristics have a significance that will be made precise in this chapter. This analysis is based in practice on a limited expansion of the function, γ or σ, which gives a local approximation of the corresponding curve or surface. The local behavior and the main features of the function can be deduced from this approximation. Thus, in a mesh generation context, the analysis of curves and surfaces can be deduced from γ or σ as well as from their successive derivatives.

Differential geometry was introduced in the early 18th century and then established in the 19th century as a way of defining a general theoretical framework for the local study of curves and surfaces. The fundamental contribution of Gauss[2] consisted of using a parametric representation of the surfaces and of showing the intrinsic nature of the total curvature. The breakthrough came with Riemann[3] who gave a global mathematical definition of curves and surfaces, introducing the notion of n-dimensional manifold.

★
★ ★

[1]Or, more generally, any mesh of the boundary of a domain.
[2]*Disquisitiones circa superficies curvas* (1827).
[3]*Sur les hypothèses qui servent de fondement à la géométrie* (1854).

The purpose of this chapter is to review the elementary notions of differential geometry necessary to understand the chapters related to the modeling as well as the mesh generation of curves and surfaces (Chapters 12 to 15). If some of these notions may appear obvious to the initiated reader[4], we believe it advisable to recall these results so as to introduce the terminology and the notations that will be used subsequently. This chapter should not be considered as a substitute for the various references in this domain ([do Carmo-1976], [Lelong-Ferrand, Arnaudies-1977], [Berger-1978], [Farin-1997], among others).

We limit ourselves here to the study of curves and surfaces embedded in the Euclidean space in two or three dimensions. The definitions require the use of the implicit function theorem. Therefore, we introduce the notion of a parameterized arc, and we conduct a local study. We also define surfaces and we introduce the two fundamental forms and the total curvature which plays an important role in the local or global behavior of the surfaces. Finally, the last section is devoted to practical aspects of the calculations related to curves and surfaces.

11.1 Metric properties of curves and arcs

This section briefly recalls the different features used for the study of curves. In particular, we outline the notions of curvilinear abscissis, of arc length and we introduce the tangent and normal vectors as well as the Frénet frame. These definitions will allow us to compute quantities like the local curvature, the radius of curvature, the osculating circle as well as the local torsion and the relevant radius of torsion. Table 11.1, given at the end of the section devoted to curves, contains the values of the main geometric features encountered.

We denote by $\mathcal{E}^d$ a Euclidean affine space of dimension d and by E^d the corresponding vector space (in practice, $E = \mathbb{R}$). We then study the properties of curves and geometric arcs of $\mathcal{E}^d$ in this Euclidean context.

Let us recall that the structure $\mathcal{E}^3$ of Euclidean space of $\mathbb{R}^3$ is defined by the choice of the usual dot product (Chapter 10), for which the canonical basis

$$[e_1 = (1,0,0)\,, e_2 = (0,1,0)\,, e_3 = (0,0,1)] \quad \text{is orthonormal.}$$

The norm of a vector X of components (x, y, z) in the space $\mathbb{R}^3$ is simply:

$$\|X\| = \sqrt{x^2 + y^2 + z^2}\,.$$

Arc length

We recall that a curve Γ of a normed vector space E corresponds to a continuous application $\gamma : I \longrightarrow E$, defined on an interval I, which associates to the parameter $t \in I$ the value $\gamma(t)$. A curve is of class C^k if the application γ is of class C^k. If in addition, the interval I is compact, the curve Γ is said to be compact.

[4]Who might prefer to skip part of this chapter.

We will define the notion of length in an arbitrary normed vector space E. To this end, we introduce an approximation of the curve (the arc) γ by a set of *inscribed polygonal lines.*

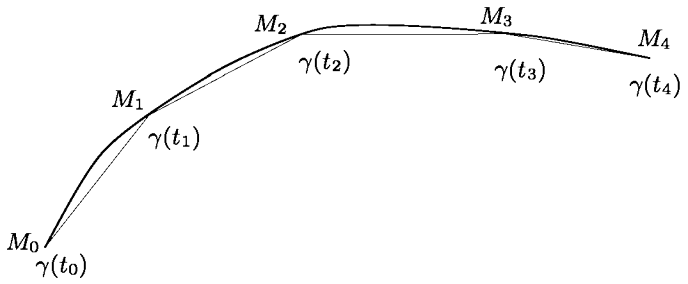


Figure 11.1: *Approximation of a curve Γ described by a function γ using an inscribed polygonal line.*

General notion of a rectifiable arc. With any subdivision $sub = (t_0, t_1, ..., t_p)$ of $[a, b]$ (with $a = t_0$ and $b = t_p$, $t_{i+1} \geq t_i$) into p segments, we associate the *polygonal line* $(M_0, M_1, ..., M_p)$ having $M_i = \gamma(t_i)$ as vertices. This subdivision is said to be *inscribed* into Γ.

Definition 11.1 *The length of this line is the number:*

$$L_{sub,\Gamma} = \sum_{i=0}^{p-1} \|\overrightarrow{M_i M_{i+1}}\| = \sum_{i=0}^{p-1} \|\gamma(t_{i+1}) - \gamma(t_i)\| \, .$$

An arc Γ of a normed vector space is said to be *rectifiable* if the upper bound $L(\Gamma)$ of the lengths $L_{sub,\Gamma}$ of the polygonal lines inscribed in Γ is *finite.* In this case, $L(\Gamma)$ is a positive real number called the *length* of Γ. Obviously we have:

$$L(\Gamma) \geq \|\gamma(b) - \gamma(a)\| \, .$$

Remark 11.1 *The length of a rectifiable arc depends on the norm chosen.*

Remark 11.2 *The fact that a curve is continuous does not necessarily imply that its length is bounded. A counter-example is that of fractal curves.*

Theorem 11.1 *In a complete normed vector space E, any compact arc Γ (of endpoints $A = \gamma(a)$ and $B = \gamma(b)$) of class C^k ($k \geq 1$) is* rectifiable. *If in addition, $\gamma : [a, b] \longrightarrow E$ is a parameterization of Γ, the length of Γ is:*

$$L(\Gamma) = \int_a^b \|\gamma'(t)\| dt \, . \tag{11.1}$$

Proof. The proof, left as an exercise, consists of showing that Γ is rectifiable, then posing:

$$\Delta_{sub} = \int_a^b \|\gamma'(t)\|dt - L_{sub,\Gamma} = \sum_{i=0}^{p-1} \left(\int_{t_i}^{t_{i+1}} \|\gamma'(t)\|dt - \|\gamma(t_{i+1}) - \gamma(t_i)\| \right) ,$$

it is sufficient to prove that, for each $\varepsilon > 0$, there exists a subdivision *sub* such that $\Delta_{sub} \leq \varepsilon$. □

In the affine Euclidean space $\mathcal{E}^d$, with the usual norm, we then find, using Relation (11.1):

$$L(\Gamma) = \int_a^b \sqrt{\gamma_1'^2(t) + ... + \gamma_d'^2(t)}dt \text{ that is, for } d = 3 \text{ , } \int_\gamma \sqrt{dx^2 + dy^2 + dz^2} .$$

For example in $\mathbb{R}^2$, the length L of the circle of center O and of radius ρ represented by $x = \rho \cos t , y = \rho \sin t , t \in [0, 2\pi]$, is (obviously) equal to:

$$L = \int_0^{2\pi} \rho\sqrt{\sin^2 t + \cos^2 t}dt = \rho \int_0^{2\pi} dt = 2\pi\rho .$$

Curvilinear abscissa, normal parameters and characteristics

Definition 11.2 *Let Γ be an arc of class C^1. We call* normal parameterization *of Γ any parameterization $\gamma : I \longrightarrow \mathcal{E}^d$ such that, $\forall t \in I$, we have: $\|\gamma'(t)\| = 1$.*

If the arc Γ is simple and *oriented* in the sense of the ascending t, the number:

$$s(t) = \int_{t_0}^{t} \|\gamma'(\theta)\|d\theta$$

is called *curvilinear abscissa* of the point $M = \gamma(t)$, measured from the point M_0, that is $\gamma(t_0)$, the *origin*. In other words, the length of the portion of curve joining two points of this curve can be expressed as a sum of arc lengths. Hence, we are able to write:

$$\|\gamma'(t)\| = s'(t) , \qquad \text{thus} \quad ds = \|\gamma'(t)\|\, dt .$$

If the parameterization is normal, we have $\|\gamma'(t)\| = 1$ and thus, $ds = dt$. In other words, for such parameterizations, s and t are identical.

Tangent vector. Let Γ be a regular oriented arc of class D^2 defined by a normal parameterization $\bar{\gamma} : I \longrightarrow \mathcal{E}^d$, $s \longmapsto \bar{\gamma}(s)$.

Definition 11.3 *The function $\vec{\tau} : I \longrightarrow \mathcal{E}_d$, $s \longmapsto \bar{\gamma}'(s)$ defines the* unit tangent vector *to Γ at any s.*

This tangent vector indicates the direction of the tangent to the curve.

Remark 11.3 *If $M = \bar{\gamma}(s)$ is not a simple point of Γ, the vector $\vec{\tau}(s)$ admits a value to the left and a value to the right that are different.*

If we consider a regular parameterization, such that, $\|\gamma'(t)\| \neq 0$, for each t, the unit tangent vector $\vec{\tau}(t)$ to Γ at t is defined by:

$$\vec{\tau}(t) = \frac{d\gamma(t)}{ds} = \frac{d\gamma(t)}{dt}\frac{dt}{ds} = \frac{\gamma'(t)}{\|\gamma'(t)\|}$$

and, if the parameterization is normal, we simply have $\vec{\tau} = \gamma'(s)$.

Principal normal. Curvature. The hyperplane denoted as Π_n going through M and orthogonal to $\vec{\tau}(s)$ is called *normal* to Γ in M. This plane (Figure 11.4) contains the various normals to the curve. By extension, any line going through M and orthogonal to $\vec{\tau}(s)$ is a *normal* to Γ in M.

As in the case of a normal parameterization, $\|\bar{\gamma}'(s)\| = 1$, a derivation shows that: $\langle \bar{\gamma}'(s), \bar{\gamma}''(s)\rangle = 0$ and thus, these two vectors are orthogonal to each other. Differently written, we have the relationship:

$$\langle \vec{\tau}(s), \frac{d\vec{\tau}(s)}{ds}\rangle = 0 .$$

Therefore, there exists a (scalar) function, $C : s \longmapsto C(s)$, such that:

$$\frac{d\vec{\tau}(s)}{ds} = C\,\vec{\nu}(s) ,$$

where $\vec{\nu}(s)$ is the vector supported by $\frac{d\vec{\tau}(s)}{ds}$, which is a unit vector. This vector is called unit oriented *normal vector* to Γ and function $C(s)$ is the *curvature* of Γ in s.

The curvature function of Γ in s can be also expressed by the relation:

$$C(s) = \|\bar{\gamma}''(s)\| .$$

Remark 11.4 *In the case of planar curves, as with the oriented plane, we can define an* algebraic curvature. *To this end, we define, from $\vec{\tau}(s)$, the vector $\vec{\nu}_1(s)$ by rotation of value $\pi/2$ and $\frac{d\vec{\tau}(s)}{ds} = C\vec{\nu}_1(s)$, which implies that C is signed.*

We note that the curvature is zero[5] if and only if the point $M = \gamma(s)$ is an *inflection* point. If the arc Γ has no inflection point, the vector $\vec{\nu}(s)$ is defined at

[5]Except for the trivial case of lines in the plane!

any s. Moreover, the *radius of curvature* of Γ in M is the number $\rho(s) = \frac{1}{C(s)}$. The point O defined by $\overrightarrow{MO} = \rho(s)\,\vec{\nu}(s)$ is the *center of curvature* of Γ in M.

If Γ is a simple regular arc having the point O as the center of curvature in M, the circle of center O passing through M and contained in plane Π defined by the basis $[\vec{\tau}, \vec{\nu}]$ is the *osculating circle* of the curve in M (Figure 11.2, right-hand side). The latter plane is the *osculating plane* to Γ in M.

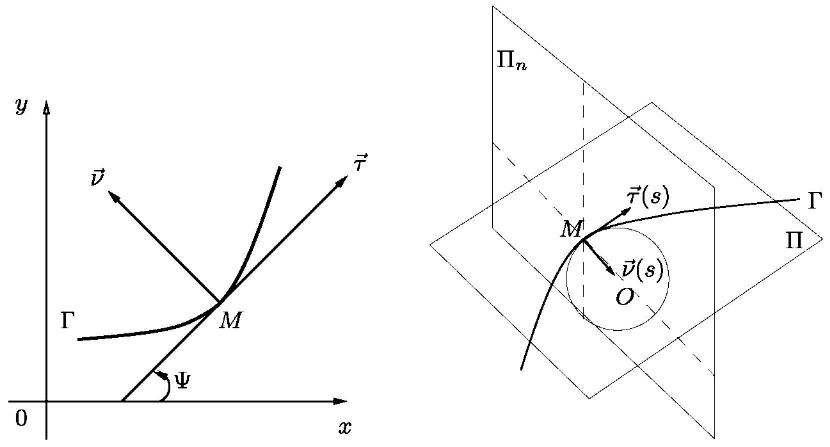


Figure 11.2: *Left-hand side, unit normal vector to the arc Γ in M. Right-hand side, the intersection of planes Π and Π_n is a line supporting the normal $\vec{\nu}$ and containing the point O, the center of curvature to Γ in M.*

Arbitrary parameterization. Let γ be a parameterization of at least class D^2 defining an oriented arc Γ, which is regular and without any inflection point. We go back to a normal parameterization $\bar{\gamma}$ using the change of parameter $t \longmapsto s(t)$, such that $ds/dt = \|\gamma'(t)\|$. As $\gamma(t) = \bar{\gamma}(s(t))$, we have:

$$\gamma'(t) = \bar{\gamma}'(s)\frac{ds}{dt}\,; \quad \gamma''(t) = \bar{\gamma}'(s)\frac{d^2(s)}{dt^2} + \bar{\gamma}''(s)\left(\frac{ds}{dt}\right)^2,$$

from which we have the formula (depending on $\vec{\tau}$ and on $\vec{\nu}$):

$$\gamma''(t) = \frac{d^2s}{dt^2}\vec{\tau}(s) + C(s)\left(\frac{ds}{dt}\right)^2 \vec{\nu}(s)\,, \tag{11.2}$$

which can also be expressed as follows:

$$\gamma''(t) = \frac{\langle\gamma'(t), \gamma''(t)\rangle}{\|\gamma'(t)\|}\vec{\tau}(s) + C(s)\|\gamma'(t)\|^2\vec{\nu}(s)\,.$$

Of all the normals, some have privileged directions. The intersection between Π and Π_n is a line oriented along the vector $\vec{\nu}(t)$ (Figure 11.2, right-hand side).

This normal is of particular interest in the study of the curves; it contains in fact, as already mentioned, the center of curvature of the curve (see for instance, [Lelong-Ferrand, Arnaudies-1977]).

Practical calculation of the curvature. In practice, it is not always possible to express the curvature of the formula $C(s) = \|\bar{\gamma}''(s)\|$ (i.e., the expression of the normal parameterization can be too complex to be practical). The calculation will thus be based on the relation:

$$C = \|\gamma''(s)\| = \left\| \frac{d\vec{\tau}(s)}{ds} \right\| .$$

We then consider an arbitrary parameterization of Γ. If M denotes the point $\bar{\gamma}(s) = \gamma(t)$, with $s = s(t)$. We start from the relation:

$$\frac{d\vec{\tau}(s)}{ds} = \frac{d\vec{\tau}(t)}{dt}\,\frac{dt}{ds} = \frac{1}{\|\gamma'(t)\|}\,\frac{d\vec{\tau}(t)}{dt} ,$$

but, $$\frac{d\vec{\tau}(t)}{dt} = \frac{\gamma''(t)\|\gamma'(t)\| - \frac{d}{dt}(\|\gamma'(t)\|)\,\gamma'(t)}{\|\gamma'(t)\|^2} ,$$

as, $\|\gamma'(t)\| = \sqrt{\langle \gamma'(t), \gamma'(t)\rangle}$, we have:

$$\frac{d}{dt}(\|\gamma'(t)\|) = \frac{\langle \gamma'(t), \gamma''(t)\rangle}{\|\gamma'(t)\|} ,$$

hence: $$\frac{d\vec{\tau}(t)}{ds} = \frac{1}{\|\gamma'(t)\|^3}\left(\gamma''(t)\|\gamma'(t)\| - \gamma'(t)\frac{\langle \gamma'(t), \gamma''(t)\rangle}{\|\gamma'(t)\|} \right) ,$$

which finally leads to the well-known formula:

$$\frac{d\vec{\tau}(t)}{ds} = \frac{\gamma'(t) \wedge \gamma''(t)}{\|\gamma'(t)\|^3} , \tag{11.3}$$

where $\wedge$ represents the cross product (to obtain this result, we use the relation $(a \wedge b) \wedge c = \langle c, a\rangle\, b - \langle c, b\rangle\, a$). Thus, we have,

$$C(t) = \frac{\|\gamma'(t) \wedge \gamma''(t)\|}{\|\gamma'(t)\|^3} . \tag{11.4}$$

Notice that in the case of a normal parameterization, this general expression gives the known result, that is:

$$C(s) = \|\gamma''(s)\| . \tag{11.5}$$

Particular case. Let us assume that we are using Cartesian coordinates to calculate this. Let $(x(t), y(t))$ be the coordinates of $\gamma(t)$ in an orthonormal frame. We can write:

$$\frac{ds}{dt} = \sqrt{x'^2(t) + y'^2(t)} \qquad \text{and} \qquad \gamma'(t) \wedge \gamma''(t) = x'(t)y''(t) - y'(t)x''(t)$$

thus we deduce the algebraic curvature of Γ at the point $M = \gamma(t)$:

$$C = \frac{x'(t)y''(t) - y'(t)x''(t)}{\sqrt{x'^2(t) + y'^2(t)}^3} .$$

Frénet's frame and formula

The two orthogonal vectors $\vec{\tau}(s)$ and $\vec{\nu}(s)$ define the *osculating plane.* From these vectors, we define the *unit binormal vector* $\vec{b}(s)$ as:

$$\vec{b}(s) = \vec{\tau}(s) \wedge \vec{\nu}(s) ,$$

and the triple $[\vec{\tau}(s), \vec{\nu}(s), \vec{b}(s)]$ forms a basis that defines *Frénet's frame* at the point M of abscissa s:

$$\mathcal{F}_{frenet} = [M, \vec{\tau}, \vec{\nu}, \vec{b}] .$$

Notice that if the orientation of Γ changes, $\vec{\tau}$ and $\vec{\nu}$ are then both changed to their opposite.

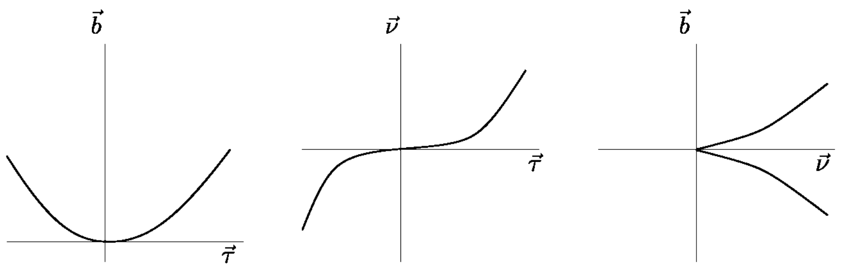


Figure 11.3: *Projections of a curve on the three planes of Frénet's frame. Left-hand side: regular point, middle: inflection point, right-hand side: cusp point.*

By derivating the relation $\langle \vec{\tau}(s), \vec{\nu}(s) \rangle = 0$, we obtain:

$$\left\langle \frac{d\vec{\tau}}{ds}(s), \vec{\nu}(s) \right\rangle + \left\langle \vec{\tau}(s), \frac{d\vec{\nu}}{ds}(s) \right\rangle = 0 ,$$

so, when introducing the curvature, the relation:

$$\left\langle \vec{\tau}(s), \frac{d\vec{\nu}(s)}{ds} \right\rangle = -C .$$

The two formulae:

$$\frac{d\vec{\tau}}{ds} = C\vec{\nu} \qquad \text{and} \qquad \frac{d\vec{\nu}}{ds} = -C\vec{\tau} , \tag{11.6}$$

are the *Frénet formulae* for the arc Γ.

Local behavior of a planar curve

Let Γ be a simple regular arc of a sufficient class, defined by an arbitrary parameterization γ or a normal parameterization $\bar{\gamma}$. We wish to study the behavior of the curve Γ in the vicinity of a point $M_0 = \gamma(t_0) = \bar{\gamma}(s_0)$. We go back to the case $t_0 = s_0 = 0$ using a translation. According to the form of the parameterization, we will study the local behavior of the curve in the local basis of the vectors $[\gamma', \gamma'']$ in $t = 0$ or in the Frénet frame $[\vec{\tau}, \vec{\nu}]$ in $s = 0$. So, we consider a small increase (a neighborhood) Δt or Δs in t or in s sufficiently small.

Applying a Taylor series at the first order to γ, we have:

$$\gamma(\Delta t) = \gamma(0) + \Delta t \gamma'(0) + \mathcal{O}(\Delta t^2)$$

and for $\bar{\gamma}$, we have similarly:

$$\bar{\gamma}(\Delta s) = \bar{\gamma}(0) + \Delta s \bar{\gamma}'(0) + \mathcal{O}(\Delta s^2),$$

with $\gamma'(0) = \vec{\tau}(0)$. This local study corresponds to an approximation of the curve using a line supported by the tangent at the point M_0. To show the gap between this approximation and the curve, we use a Taylor series at order 2:

$$\gamma(\Delta t) = \gamma(0) + \Delta t \gamma'(0) + \frac{\Delta t^2}{2}\gamma''(0) + \mathcal{O}(\Delta t^3)$$

and for $\bar{\gamma}$, we have similarly:

$$\bar{\gamma}(\Delta s) = \bar{\gamma}(0) + \Delta s \bar{\gamma}'(0) + \frac{\Delta s^2}{2}\bar{\gamma}''(0) + \mathcal{O}(\Delta s^3),$$

with $\gamma'(0) = \vec{\tau}(0)$ and $\gamma''(0) = \vec{\nu}(0)/\rho(0) = C(0)\,\vec{\nu}(0)$ where $\rho(0)$ is the radius of curvature ($C(0)$ the curvature) at point M_0. In these expressions, the second order term measures the desired gap. This local study corresponds to an approximation of the curve by a parabola which, in the former case, is defined in a non-orthonormal frame, and in the latter case, is defined in the orthonormal Frénet frame. The gap between this parabola and the curve can be obtained by pursuing the expansion at a higher order. A Taylor series at order 3 applied to γ gives, on the one hand:

$$\gamma(\Delta t) = \gamma(0) + \Delta t \gamma'(0) + \frac{\Delta t^2}{2}\gamma''(0) + \frac{\Delta t^3}{6}\gamma'''(0) + \mathcal{O}(\Delta t^4)$$

and, on the other hand:

$$\bar{\gamma}(\Delta s) = \bar{\gamma}(0) + \Delta s \bar{\gamma}'(0) + \frac{\Delta s^2}{2}\bar{\gamma}''(0) + \frac{\Delta s^3}{6}\bar{\gamma}'''(0) + \mathcal{O}(\Delta s^4).$$

As the local study is more intuitive in the Frénet's frame and thus in the case of a normal parameterization, we will examine this case more carefully. Let us recall that $\bar{\gamma}'(0) = \vec{\tau}(0)$ and that $\bar{\gamma}''(0) = \vec{\nu}(0)/\rho(0) = C(0)\,\vec{\nu}(0)$ and let us express $\bar{\gamma}'''(0)$ in the Frénet's frame:

$$\bar{\gamma}'''(s) = \frac{d\dfrac{\vec{\nu}(s)}{\rho(s)}}{ds} = -\frac{\vec{\tau}(s) + \rho'(s)\,\vec{\nu}(s)}{\rho^2(s)}.$$

The previous limited expansion can then be expressed as follows:

$$\vec{\gamma}(\Delta s) = \vec{\gamma}(0) + \Delta s \vec{\tau}(0) + \frac{\Delta s^2}{2\,\rho(0)}\vec{\nu}(0) - \frac{\Delta s^3}{6\,\rho^2(0)}\left(\vec{\tau}(0) + \rho'(0)\vec{\nu}(0)\right) + \mathcal{O}(\Delta s^4)\,.$$

The term $\dfrac{\Delta s^3}{6\,\rho^2(0)}(\vec{\tau}(0) + \rho'(0)\vec{\nu}(0))$ measures the gap to the parabola.

The coordinates $x(\Delta s), y(\Delta s)$ of the point M of abscissa Δs are such that:

$$\left\{\begin{array}{lcl} x(\Delta s) & = & \Delta s - \dfrac{\Delta s^3}{6\,\rho^2(0)} + \mathcal{O}(\Delta s^4) \\ y(\Delta s) & = & \dfrac{\Delta s^2}{2\,\rho(0)} - \dfrac{\Delta s^3\,\rho'(0)}{6\,\rho^2(0)} + \mathcal{O}(\Delta s^4) \end{array}\right. .$$

Then,

- if, in $s = 0$, the curvature is zero $C(0) = 0$ (i.e., the radius of curvature $\rho(0)$ is infinite), we have the case of an inflection point (Figure 11.3, middle). The arc crosses its tangent at point M_0. Notice that a line segment has no inflection point.

- if $C(0) \neq 0$, arc Γ is on the same side of its tangent as that of its center of curvature in M_0. We can specify the position of Γ by considering a circle Γ_a of radius $|a|$, tangent to Γ in M_0 defined by the normal parameterization:

  $$X(s) = a\sin\left(\frac{s}{a}\right), \quad Y(s) = a\left(1 - \cos\left(\frac{s}{a}\right)\right). \text{ And, for } \quad a = \frac{1}{C(0)} = \rho(0)\,,$$

  the circle Γ_a is the osculating circle to Γ. Using a limited expansion of the sine and cosine, we find:

  $$x(\Delta s) - X(\Delta s) = \mathcal{O}(\Delta s^4) \quad \text{and} \quad y(\Delta s) - Y(\Delta s) = -\frac{\Delta s^3\,\rho'(0)}{6\,\rho^2(0)} + \mathcal{O}(\Delta s^4)\,.$$

  So, if $\rho'(0) \neq 0$, the sign of $y(\Delta s) - Y(\Delta s)$ changes with that of Δs and the arc crosses its osculating circle in M_0.

Geometric interpretation of the osculating circle. Let us consider the circle Γ_a defined by the former parameterization, where a is an arbitrary scalar value. This circle passes through the point $M = (x(s), y(s))$ if and only if $a = \lambda(s)$, where

$$\lambda(s) = \frac{x^2(s) + y^2(s)}{2y(s)} = \frac{x^2(s)}{2y(s)} + \frac{1}{2}y(s)\,.$$

So, we can deduce:

$$\lim_{s\to 0}\lambda(t) = \lim_{t\to 0}\frac{x^2(t)}{2y(t)} = \frac{1}{C(0)} = \rho(0)\,.$$

Then, for each $M \in \Gamma$, close to M_0, there exists a unique circle Γ_M tangent to Γ in M_0 passing through M. The osculating circle to Γ in M_0 is the limit of Γ_M when M tends toward M_0 along Γ.

Arcs in $\mathbb{R}^3$. Frénet's frame and Serret-Frénet's formulae

Let us assume that the arc Γ has no inflection point (i.e., $C(s) \neq 0, \forall s \in I$). *Frénet's frame* $[M, \vec{\tau}(s), \vec{\nu}(s), \vec{b}(s)]$ is, as already seen, the direct orthonormal frame of $\mathbb{R}^3$ defined by the vectors $\vec{\tau}(s)$, $\vec{\nu}(s)$and the vector $\vec{b}(s) = \vec{\tau}(s) \wedge \vec{\nu}(s)$ that is the *binormal vector* to Γ. The tangent plane passing through M, of main vectors $\vec{\nu}(s)$ and $\vec{b}(s)$, is called the *normal plane* to Γ in M and the plane passing through M, of main vectors $\vec{\tau}(s)$ and $\vec{b}(s)$ is called the *rectifying plane* to Γ in M (Figure 11.4).

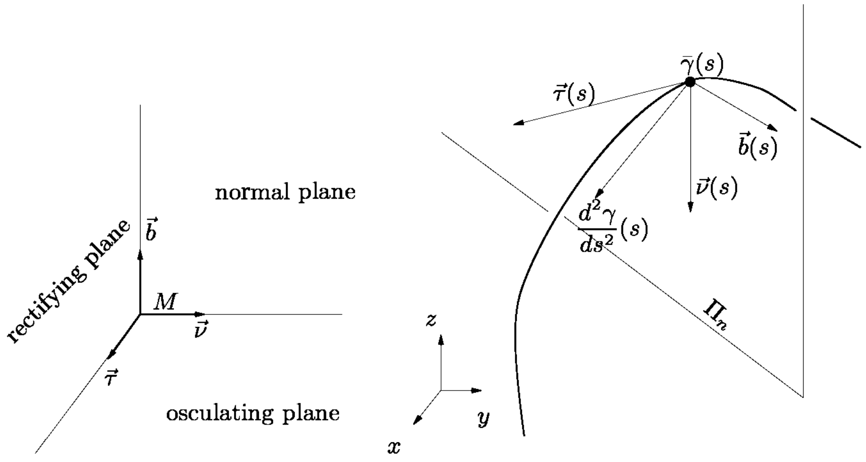


Figure 11.4: *Geometrical entities related to the parameter s at a point M of* Γ.

Serret-Frénet's formulae. Torsion. By assumption, the functions $\vec{\tau}$, $\vec{\nu}$ and $\vec{b}$ are such that:

$$\|\vec{\tau}(s)\| = \|\vec{\nu}(s)\| = \|\vec{b}(s)\| = 1,$$

$$\langle \vec{\tau}(s), \vec{\nu}(s) \rangle = 0, \qquad \langle \vec{b}(s), \vec{\tau}(s) \rangle = 0, \qquad \langle \vec{b}(s), \vec{\nu}(s) \rangle = 0.$$

In order to establish Serret-Frénet's formulae, we will express the derivatives of these three vectors of Frénet's basis in this same basis. For $\vec{\tau}(s)$, we know that:

$$\frac{d\vec{\tau}(s)}{ds} = C(s)\, \vec{\nu}(s). \tag{11.7}$$

For $\vec{\nu}$, we write (and we look for a_0, a_1 and a_2):

$$\frac{d\vec{\nu}(s)}{ds} = a_0 \vec{\tau}(s) + a_1 \vec{\nu}(s) + a_2 \vec{b}(s).$$

Obviously $a_1 = 0$. To find a_0 we look at the dot product by $\vec{\tau}(s)$:

$$a_0 = \left\langle \frac{d\vec{\nu}(s)}{ds}, \vec{\tau}(s) \right\rangle = -\left\langle \frac{d\vec{\tau}(s)}{ds}, \vec{\nu}(s) \right\rangle = -C,$$

to find a_2, we consider the product by $\vec{b}(s)$:

$$a_2 = \left\langle \frac{d\vec{\nu}(s)}{ds}, \vec{b}(s) \right\rangle ,$$

which is called the *torsion*[6], $T(s)$. So, we have:

$$\frac{d\vec{\nu}(s)}{ds} = -\,C(s)\vec{\tau}(s) \,+\, T(s)\,\vec{b}(s)\,. \tag{11.8}$$

For $\vec{b}(s)$, we write again:

$$\frac{d\vec{b}(s)}{ds} = a_0\vec{\tau}(s) + a_1\vec{\nu}(s) + a_2\vec{b}(s)\,.$$

Here, $a_0 = a_2 = 0$ and $a_1 = \left\langle \frac{d\vec{b}(s)}{ds}, \vec{\nu}(s) \right\rangle = -\left\langle \frac{d\vec{\nu}(s)}{ds}, \vec{b}(s) \right\rangle$, that is $-T$, and thus:

$$\frac{d\vec{b}(s)}{ds} = -\,T(s)\vec{\nu}(s)\,. \tag{11.9}$$

By combining Relations (11.7), (11.8) and (11.9), we obtain *Frénet*'s (or *Serret-Frénet*'s) formulae:

$$\frac{d\vec{\tau}}{ds} = C\vec{\nu}\,, \qquad \frac{d\vec{\nu}}{ds} = -C\vec{\tau} + T\vec{b}\,, \qquad \frac{d\vec{b}}{ds} = -T\vec{\nu}\,. \tag{11.10}$$

the number $\rho(s) = 1/C(s)$ being called the *radius of curvature* of Γ at the point $M = \vec{\gamma}(s)$. If, in addition, $T(s) \neq 0$, the number $R_T(s) = 1/T(s)$ is called the *radius of torsion* of Γ at the point M. With these two notations, Serret-Frénet's formulae can be written as:

$$\frac{d\vec{\tau}}{ds} = \frac{\vec{\nu}}{\rho}\,, \qquad \frac{d\vec{\nu}}{ds} = -\frac{\vec{\tau}}{\rho} + \frac{\vec{b}}{R_T}\,, \qquad \frac{d\vec{b}}{ds} = -\frac{\vec{\nu}}{R_T}\,. \tag{11.11}$$

These relations indicate the fact that the matrix $\mathcal{M}$ of the vectors $d\vec{\tau}/ds$, $d\vec{\nu}/ds$, $d\vec{b}/ds$ in the basis $[\vec{\tau}(s), \vec{\nu}(s), \vec{b}(s)]$ is:

$$\mathcal{M} = \begin{bmatrix} 0 & C(s) & 0 \\ -C(s) & 0 & T(s) \\ 0 & -T(s) & 0 \end{bmatrix} = \begin{bmatrix} 0 & \dfrac{1}{\rho(s)} & 0 \\ \dfrac{-1}{\rho(s)} & 0 & \dfrac{1}{R_T(s)} \\ 0 & \dfrac{-1}{R_T(s)} & 0 \end{bmatrix}\,. \tag{11.12}$$

This matricial relation is useful when looking at the behavior of the curve in space when the parameter changes.

[6]Or second curvature.

Computing curvature and torsion

If Γ is defined by a normal parameterization, Frénet's formulae allow us to evaluate the curvature and torsion of Γ by means of the derivatives $\bar{\gamma}'$, $\bar{\gamma}''$ and $\bar{\gamma}'''$. We have already seen, in fact, the case of the curvature. For a normal parameterization, it is Relation (11.5): $C(s) = \|\gamma''(s)\|$. For an arbitrary parameterization, it is Relation (11.4):

$$C(t) = \frac{\|\gamma'(t) \wedge \gamma''(t)\|}{\|\gamma'(t)\|^3}\,.$$

The torsion is then obtained using the previous formulae. The easiest way of computing the torsion is to start from Relation (11.8):

$$\frac{d\vec{\nu}(s)}{ds} = -\,C(s)\vec{\tau}(s) + T(s)\,\vec{b}(s)$$

and to apply the dot product with $\vec{b}(s)$. Hence, we find (omitting the sign):

$$T(s) = \left\langle \frac{d\vec{\nu}(s)}{ds}, \vec{b}(s) \right\rangle = \left\langle \frac{d\vec{\nu}(s)}{ds}, \vec{\tau}(s) \wedge \vec{\nu}(s) \right\rangle = det \left| \frac{d\vec{\nu}(s)}{ds}, \vec{\tau}(s), \vec{\nu}(s) \right|\,.$$

We have $\vec{\tau}(s) = \bar{\gamma}'(s)$, $\vec{\nu}(s) = \frac{1}{C(s)}\frac{d\vec{\tau}(s)}{ds} = \frac{1}{C(s)}\bar{\gamma}''(s)$ and it is then sufficient to express $\frac{d\vec{\nu}(s)}{ds}$ to find the desired result.

$$\frac{d\vec{\nu}(s)}{ds} = \frac{d}{ds}\left(\frac{1}{C(s)}\bar{\gamma}''(s)\right),$$

$$\text{however} \quad \frac{d}{ds}\left(\frac{1}{C(s)}\bar{\gamma}''(s)\right) = \frac{1}{C(s)^2}\left(C(s)\bar{\gamma}'''(s) - \bar{\gamma}''(s)\frac{dC(s)}{ds}\right)$$

$$\text{and as} \quad \frac{dC(s)}{ds} = 2.\,\langle\bar{\gamma}'''(s), \bar{\gamma}''(s)\rangle = 0\,,$$

then, $\frac{d\vec{\nu}(s)}{ds}$ simply reads $\frac{1}{C(s)}\gamma'''(s)$ and the torsion is (omitting the sign):

$$T(s) = \frac{1}{C(s)^2} det\,|\bar{\gamma}'''(s)\ \ \bar{\gamma}'(s)\ \ \bar{\gamma}''(s)| = \frac{1}{C(s)^2} det\,|\bar{\gamma}'(s)\ \ \bar{\gamma}''(s)\ \ \bar{\gamma}'''(s)|\,. \tag{11.13}$$

We find again the relation (without the sign):

$$det\,|\bar{\gamma}'(s), \bar{\gamma}''(s), \bar{\gamma}'''(s)| = -C^2(s)\,T(s)\,.$$

Remark 11.5 *These formulae are not always obvious to compute. If the function $ds/dt = \|\gamma'(t)\|$ admits a simple expression, it may be interesting to compute the components of the vectors $\vec{\tau}, \vec{\nu}$ and $\vec{b}$ at the point M, which is equivalent to applying Frénet's formulae without specifying the normal parameter.*

So, we can express the torsion as a function of t. We have:

$$T(t) = det \left| \frac{d\vec{\nu}(t)}{ds}\ \ \vec{\tau}(t)\ \ \vec{\nu}(t) \right|\,.$$

In this determinant, we now have:

$$\vec{\tau}(t) = \frac{\gamma'(t)}{\|\gamma'(t)\|},$$

$$\vec{\nu}(t) = \frac{1}{C(s)}\frac{d\vec{\tau}(t)}{ds} = \frac{1}{C(s)}\frac{\gamma'(t)\wedge\gamma''(t)}{\|\gamma'(t)\|^3} = \frac{\gamma'(t)\wedge\gamma''(t)}{\|\gamma'(t)\wedge\gamma''(t)\|},$$

$$\text{and for } \frac{d\vec{\nu}(t)}{ds}, \text{ we obtain: } \frac{d\vec{\nu}(t)}{ds} = \frac{\vec{\nu}'(t)}{\|\gamma'(t)\|}.$$

It remains to express $\vec{\nu}'(t)$. We have:

$$\vec{\nu}'(t) = \left(\frac{1}{C(s)}\frac{\gamma'(t)\wedge\gamma''(t)}{\|\gamma'(t)\|^3}\right)',$$

and the calculation gives:

$$\vec{\nu}'(t) = \frac{\gamma'(t)\wedge\gamma'''(t)}{\|\gamma'(t)\wedge\gamma''(t)\|},$$

so, we find:

$$T(t) = det\left|\frac{1}{\|\gamma'(t)\|}\frac{\gamma'(t)\wedge\gamma'''(t)}{\|\gamma'(t)\wedge\gamma''(t)\|}\;\frac{\gamma'(t)}{\|\gamma'(t)\|}\;\frac{\gamma'(t)\wedge\gamma''(t)}{\|\gamma'(t)\wedge\gamma''(t)\|}\right|,$$

which gives the expression of the torsion:

$$T(t) = \frac{det|\gamma'(t),\gamma''(t),\gamma'''(t)|}{\|\gamma'(t)\wedge\gamma''(t)\|^2}. \tag{11.14}$$

Use of the tangent indicatrix. The indicatrix of the tangents γ_1 is defined by the parameterization:

$$\gamma_1 : I \longrightarrow E^3, \quad t \longmapsto \frac{\gamma'(t)}{\|\gamma'(t)\|}.$$

The unit tangent vector to γ_1 is $\vec{\tau}_1 = \vec{\nu}$ and the curvature C of γ is:

$$C(s) = \|\gamma_1'(t)\|\frac{dt}{ds}.$$

Summary table. Table 11.1 summarizes the various vector functions and definitions introduced in this section, along with their notations.

Local metric study of arcs in $\mathbb{R}^3$

Let us denote by $(x(\Delta s), y(\Delta s), z(\Delta s))$ the coordinates of $\bar{\gamma}(\Delta s)$ in Frénet's frame of Γ (assumed to be regular in the vicinity of M_0) at point M_0. The study of the curve's behavior in the vicinity of 0, so for small Δs, leads to looking at a Taylor

function	notat.	depending on t	depending on s
curvilinear abscissa	s	$s(t) = \int_{t_0}^{t} \|\gamma'(\theta)\| \, d\theta$	-
tangent vector	$\vec{\tau}$	$\dfrac{\gamma'(t)}{\|\gamma'(t)\|}$	$\gamma'(s)$
curvature	$C = \frac{1}{\rho}$	$\dfrac{\|\gamma'(t) \wedge \gamma''(t)\|}{\|\gamma'(t)\|^3}$	$\|\bar{\gamma}''(s)\|$
normal vector	$\vec{\nu}$	$\dfrac{1}{C(t)} \dfrac{d\vec{\tau}(t)}{dt}$	$\dfrac{\bar{\gamma}''(s)}{\|\bar{\gamma}''(s)\|}$
torsion	T	$\dfrac{det\|\gamma'(t), \gamma''(t), \gamma'''(t)\|}{\|\gamma'(t) \wedge \gamma''(t)\|^2}$	$\dfrac{det\|\bar{\gamma}'(s), \bar{\gamma}''(s), \bar{\gamma}'''(s)\|}{C^2(s)}$

Table 11.1: Notations and characteristic values related to a curve.

series at an adequate order in the vicinity of 0. By noticing that order 3 is required to have an analysis that is not restricted to the plane defined by the tangent and the normal, we write the expansion at this order:

$$\bar{\gamma}(\Delta s) = \bar{\gamma}(0) + \Delta s \bar{\gamma}'(0) + \frac{\Delta s^2}{2} \bar{\gamma}''(0) + \frac{\Delta s^3}{6} \bar{\gamma}'''(0) + \mathcal{O}(\Delta s^4),$$

with $\bar{\gamma}(0) = M_0$, $\bar{\gamma}'(0) = \vec{\tau}(0)$, $\bar{\gamma}''(0) = C(0)\vec{\nu}(0)$. Posing M the point $\bar{\gamma}(\Delta s)$ and returning to the Frénet formula for $\vec{\nu}'0)$, we have:

$$\begin{aligned} M &= M_0 + \Delta s \vec{\tau}(0) + \tfrac{\Delta s^2}{2} C(0)\vec{\nu}(0) \\ &+ \tfrac{\Delta s^3}{6} \left(-C^2(0)\vec{\tau}(0) + C'(0)\vec{\nu}(0) + C(0)T(0)\vec{b}(0) \right) + \mathcal{O}(\Delta s^4), \end{aligned}$$

and, in terms of the Frénet's frame, this reads

$$\left\{ \begin{array}{lcl} x(\Delta s) &=& \Delta s - \frac{C^2(0)\Delta s^3}{6} + \mathcal{O}(\Delta s^4) \\ y(\Delta s) &=& \frac{C(0)\Delta s^2}{2} + \frac{C'(0)\Delta s^3}{6} + \mathcal{O}(\Delta s^4) \\ z(\Delta s) &=& \frac{C(0)T(0)\Delta s^3}{6} + \mathcal{O}(\Delta s^4) \end{array} \right. \tag{11.15}$$

These relations give the behavior of the projections γ_1, γ_2 and γ_3 of γ on the osculating, rectifying and normal planes to Γ in M_0. To do so, it is sufficient to choose the corresponding plane and the two associated components. For example, to study the behavior of γ_2, we check whether $T(0) < 0$ or whether $T(0) > 0$ (cf. Figure 11.5).

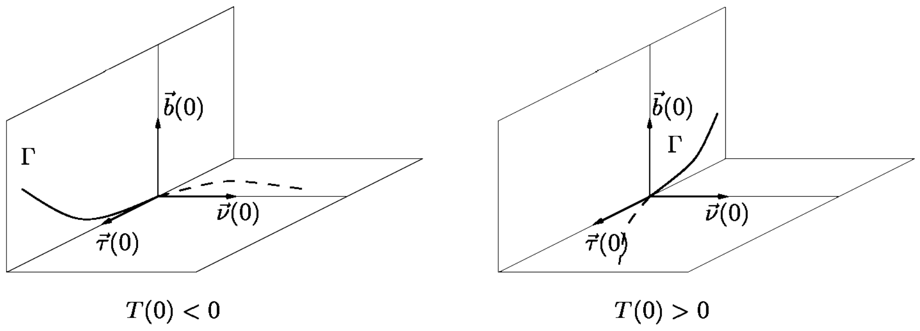


Figure 11.5: *Behavior of the curve γ_2 (projection of Γ on the rectifying plane) according to the sign of the torsion $T(0)$.*

Osculating spheres. Let Γ be a simple arc, which is regular, without any inflection point and of class C^k with $k \geq 3$. Let M_0 be a point of Γ. Then:

Definition 11.4 *There exists an infinity of* osculating spheres *to Γ in M_0, which are all the spheres passing through the osculating circle*[7] *to Γ in M_0.*

Remark 11.6 *The radii of the osculating spheres are greater than or equal to the radius of curvature.*

Parameterization of arcs

In this short section, we are interested in various parameterizations of simple arcs and curves, especially the arcs defined by a Cartesian parameterization and the implicit curves of $\mathbb{R}^2$.

Arcs defined by a Cartesian parameterization. We consider an affine space of finite dimension d.

Definition 11.5 *A* Cartesian parameterization *is such that, in an adequate frame, the parameter is equal to one of the coordinates.*

For example, in the affine plane $\mathbb{R}^2$, the parabola defined by

$$x \longmapsto (x, kx^2)\,,\ (k = Cte,\ x \in \mathbb{R})\,,\ \text{is a simple arc.}$$

[7]This is equivalent to saying that such a sphere contains the circle of curvature of γ at this point.

In a broader sense, in $\mathbb{R}^d$, we consider a parameterization f of the form:

$$x_1 = t\,, \quad x_2 = f_2(t)\,, \quad \ldots, \quad x_d = f_d(t) \quad (t \in I)\,,$$

the functions f_i $(2 \le i \le d)$ being of class C^k on I. An arc of class C^k with $k \ge 1$ admitting such a parameterization is *simple* and *regular*. Let us consider the parameterization f of the form $(t, f(t))$ in the affine plane $\mathbb{R}^2$. By applying Formula (11.1), we obtain a simplified expression of the length ds of an arc:

$$ds = \|f'(t)\|dt = \sqrt{1 + f'^2(t)}\,.$$

Similarly, the tangent vector $\vec{\tau}$ is obtained by the formula:

$$\vec{\tau} = \frac{1}{\sqrt{1+f'^2(t)}} \begin{pmatrix} 1 \\ f(t) \end{pmatrix},$$

and the (signed) curvature is given by $C = \dfrac{f''(t)}{(1+f'^2(t))^{3/2}}$.

Implicit curves in $\mathbb{R}^2$. Let $f : \Omega \longrightarrow \mathbb{R}$ be a function of class C^k $(k \ge 1)$ defined on an open set Ω of the affine plane $\mathbb{R}^2$ and let Γ_f be the set of points $M \in \Omega$ such that $f(M) = 0$.

Definition 11.6 *The pair* (f, Γ_f) *is the so-called* implicit curve $f = 0$.

Using the implicit function theorem, we can show that the line tangent at M_0 to Γ_0 (arc of class C^k whose local support is the curve Γ_f) is defined by an equation such as:

$$(x - x_0) f'_x(x_0, y_0) + (y - y_0) f'_y(x_0, y_0) = 0\,,$$

where $f(x,y)$ denotes the value of f at point $M(x,y)$, i.e., we write in the same way both $f(M)$ and $f(x,y)$. Refer to Chapter 16 for more details about implicit curves and functions as well as computing their intrinsic properties.

We will now move on to the study of the metric properties of surfaces.

11.2 Metric properties of a surface

This section recalls some basic notions useful for the study of surfaces. The aim is once again to give a brief overview of the classical results of differential geometry that can suit our purposes. In this section, we denote by $\mathcal{E}$ an oriented Euclidean affine space of dimension 3 and by E the associated Euclidean vector space. We recall (by analogy with the curves of E) that a parametric surface [8] of class C^k of $\mathcal{E}$ is an application of class C^k of a domain of $\mathbb{R}^2$ into $\mathcal{E}$.

Let Σ be a regular surface defined by the parameterization σ:

$$\sigma : \Omega \longrightarrow \mathbb{R}^3, \quad (u,v) \longmapsto \sigma(u,v)\,,$$

[8] Also called a parametric sheet.

where Ω denotes a domain of $\mathbb{R}^2$ and σ is a function of class C^k $(k \geq 2)$. By analogy with the study of the local behavior of a curve using a limited expansion, we first indicate what such a development is (at order 2) in the case of a surface.

Let $M = \sigma(u,v)$ be the point of parameters (u,v). We consider the limited expansion at order 2 of σ at the parameters (u,v) for a small increase $(\Delta u, \Delta v)$:

$$\sigma(u+\Delta u, v+\Delta v) = \sigma(u,v) + \underbrace{\sigma'_u \Delta u + \sigma'_v \Delta v}_{\text{order 1}} + \underbrace{\frac{1}{2}\sigma''_{uu}\Delta u^2 + \sigma''_{uv}\Delta u \Delta v + \frac{1}{2}\sigma''_{vv}\Delta v^2}_{\text{order 2}} \tag{11.16}$$

where $\sigma'_u(u,v)$ represents $\dfrac{\partial \sigma}{\partial u}(u,v)$, $\sigma'_v(u,v)$ denotes $\dfrac{\partial \sigma}{\partial v}(u,v)$, $\sigma''_{uu}(u,v)$ denotes $\dfrac{\partial^2 \sigma}{\partial u^2}(u,v)$, $\sigma''_{uv}(u,v)$ is $\dfrac{\partial^2 \sigma}{\partial u \partial v}(u,v)$ and $\sigma''_{vv}(u,v)$ is $\dfrac{\partial^2 \sigma}{\partial v^2}(u,v)$. According to the depth of the limited expansion (at order 1 or 2), we obtain two approximations of the surface allowing to obtain its intrinsic features. These approximations involve the successive derivatives of σ and the fundamental forms of surface Σ.

First fundamental quadratic form

At point $M = M(u,v)$, the tangent vector plane T_M is directed by the vectors:

$$\frac{\partial M}{\partial u} = \sigma'_u(u,v)\,, \qquad \frac{\partial M}{\partial v} = \sigma'_v(u,v)\,,$$

which are respectively noted by $\vec{\tau_1}$ and $\vec{\tau_2}$. Any vector $\vec{V}$ of T_M can be written as:

$$\vec{V} = \lambda\,\vec{\tau_1} + \mu\,\vec{\tau_2}\,.$$

So, we have : $\|\vec{V}\|^2 = \|\lambda\vec{\tau_1} + \mu\vec{\tau_2}\|^2 = \lambda^2\|\vec{\tau_1}\|^2 + 2\,\lambda\,\mu\langle\vec{\tau_1},\vec{\tau_2}\rangle + \mu^2\|\vec{\tau_2}\|^2$. Posing,

$$E = \|\vec{\tau_1}\|^2\,, \quad F = \langle\vec{\tau_1},\vec{\tau_2}\rangle \quad \text{and} \quad G = \|\vec{\tau_2}\|^2\,,$$

this expression can be written: $\|\vec{V}\|^2 = E\lambda^2 + 2F\lambda\mu + G\mu^2$. Thus, we have introduced the first fundamental form of the surface, its precise definition being:

Definition 11.7 *Let T_M be the tangent vector plane to a surface Σ at a point M. The restriction to T_M of the quadratic form $\vec{V} \longmapsto \|\vec{V}\|^2$, $(\vec{V} \in E)$, is called the* first fundamental quadratic form *of Σ at M.*

This form is usually denoted by Φ_1^M.

Expression of Φ_1. The usual expression of Φ_1^M in the basis $[\vec{\tau_1},\vec{\tau_2}]$ is then:

$$\Phi_1^M(\vec{V}) = E\lambda^2 + 2F\lambda\mu + G\mu^2\,, \tag{11.17}$$

which can also be written as a "differential" form:

$$\Phi_1^M(dM) = Edu^2 \,+\, 2Fdudv \,+\, Gdv^2\,, \tag{11.18}$$

where $dM = \dfrac{\partial M}{\partial u}du + \dfrac{\partial M}{\partial v}dv$. Posing $\Lambda = \begin{pmatrix} \lambda \\ \mu \end{pmatrix}$ and $\mathcal{M}_1(M) = \begin{pmatrix} E & F \\ F & G \end{pmatrix}$, we can write this fundamental form using a matrix expression:

$$\Phi_1^M(\vec{V}) = {}^t\Lambda \, \mathcal{M}_1(M) \, \Lambda .$$

Remark 11.7 *The first fundamental form defines the* metric of the tangent plane *to Σ which will be used to govern the calculation of lengths (as will be seen later).*

We then introduce the function $H = \sqrt{EG - F^2}$ which allows[9] us to write:

$$H = \|\vec{\tau}_1 \wedge \vec{\tau}_2\| .$$

Case of a Cartesian parameterization. We now assume that Σ is defined by the following Cartesian equation $z = f(x, y)$, where f is a function of class C^k on a planar domain. If $M(x, y)$ denotes the point of coordinates $(x, y, f(x, y))$, we will have the following expressions of E, F and G:

$$E = \left\| \frac{\partial M}{\partial x} \right\|^2 = 1 + f_x'^2(x,y) \, , \; F = \left\langle \frac{\partial M}{\partial x}, \frac{\partial M}{\partial y} \right\rangle = f_x'(x,y) f_y'(x,y) \, ,$$

$$\text{and } G = \left\| \frac{\partial M}{\partial y} \right\|^2 = 1 + f_y'^2(x,y) \, .$$

Area of a surface. The quantities E, F and G are used to define a surface element Δa of a surface, corresponding to a small variation Δu and Δv around point $M(u, v)$:

$$\Delta a = \left\| \frac{\partial M}{\partial u} \Delta u \wedge \frac{\partial M}{\partial v} \Delta v \right\| ,$$

then:

$$\Delta a = \sqrt{\langle (\vec{\tau}_1 \Delta u \wedge \vec{\tau}_2 \Delta v), (\vec{\tau}_1 \Delta u \wedge \vec{\tau}_2 \Delta v) \rangle} \, .$$

Hence, using the previously described relations, we reach the expression:

$$\Delta a = \sqrt{EG - F^2} \Delta u \, \Delta v = H \Delta u \, \Delta v \, .$$

The area of a small piece of surface in the vicinity of point M is then (Figure 11.6):

$$a = \iint H \Delta u \, \Delta v \, .$$

We now analyze how to compute the length of an arc traced on a surface. We will see that such a calculation involves the previous fundamental form.

[9]This function H is notably used to compute the *area* of Σ.

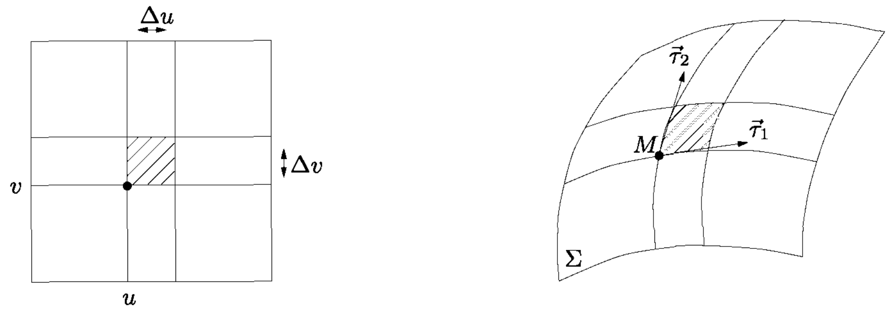


Figure 11.6: *Element of surface defined using (u, v) and $(u + \Delta u, v + \Delta v)$ where Δu (resp. Δv) represents a small increase in u (resp. v).*

Length of an arc traced on Σ. Let us assume that Γ is the image by the parameterization σ of the planar arc Γ defined by the parameterization: $t \longmapsto (u(t), v(t))$. So, Γ is defined by the parameterization σ: $t \longmapsto \sigma(u(t), v(t))$ and thus we have:

$$\sigma' = u'(t)\frac{\partial M}{\partial u} + v'(t)\frac{\partial M}{\partial v},$$

allowing us to write:

$$||\sigma'(t)||^2 = Eu'^2(t) + 2Fu'(t)v'(t) + Gv'(t).$$

Indeed, if $[a, b]$ is an interval, the length of Γ is:

$$L(\Gamma) = \int_a^b \sqrt{Eu'^2 + 2Fu'v' + Gv'^2 dt},$$

and finally (posing $du = u'(t)$ and $dv = v'(t)$):

$$ds^2 = Edu^2 + 2Fdudv + Gdv^2 \text{ or also } ds = \sqrt{Edu^2 + 2Fdudv + Gdv^2}.$$

Remark 11.8 *We can also express the length of an arc traced on Σ with respect to t:*

$$s(t) = \int_a^b \sqrt{Edu^2 + 2Fdudv + Gdv^2}.$$

Normal. Local frame. Darboux's frame

We will then focus on the notion of normal and oriented normal. Let Σ be a simple surface of class C^k $(k \geq 1)$ of $\mathcal{E}$ defined by the parameterization $\sigma : (u, v) \longrightarrow M(u, v) = \sigma(u, v)$.

Normal. We pose:

$$\vec{N}(u, v) = \vec{\tau}_1 \wedge \vec{\tau}_2,$$

with $\vec{\tau}_1$ and $\vec{\tau}_2$ the vectors of the basis of the tangent plane previously introduced. With the notations introduced before, we can write:

$$\|\vec{N}(u,v)\| = H(u,v) \quad \text{with} \quad H = \sqrt{EG - F^2}\,.$$

The vector $\vec{N}(u,v)$ thus defined is called the *normal vector* to Σ at M associated with the given parameterization σ. The affine line passing through M and directed by vector $\vec{N}$ (orthogonal to the tangent plane) is called the *normal* to Σ at M.

Oriented normal. The unit vector

$$\vec{n}(u,v) = \frac{\vec{N}(u,v)}{H(u,v)} = \frac{\vec{\tau}_1 \wedge \vec{\tau}_2}{\|\vec{\tau}_1 \wedge \vec{\tau}_2\|}$$

does not depend on the orientation of Σ with respect to the given parameterization. This vector is the *unit normal vector* at M to Σ.

Local frame. At point $M = \sigma(u,v)$, the tangent plane T_M can be defined by means of the parametric equation: $M + \lambda\vec{\tau}_1 + \mu\vec{\tau}_2$. The normal $\vec{\tau}_1 \wedge \vec{\tau}_2$ to the plane T_M coincides with the normal to Σ at M. So, the unit normal

$$\vec{n} = \frac{\vec{\tau}_1 \wedge \vec{\tau}_2}{\|\vec{\tau}_1 \wedge \vec{\tau}_2\|}$$

and the vectors $\vec{\tau}_1$ and $\vec{\tau}_2$ form a local system of coordinates:

$$\mathcal{F}_{loc} = [M, \vec{\tau}_1, \vec{\tau}_2, \vec{n}]\,,$$

the so-called *local frame* at M (Figure 11.7, left-hand side).

Remark 11.9 *Notice also that the axes $\vec{\tau}_1$ and $\vec{\tau}_2$ usually form only an affine system. This frame is the analog of the Frénet frame for the curves.*

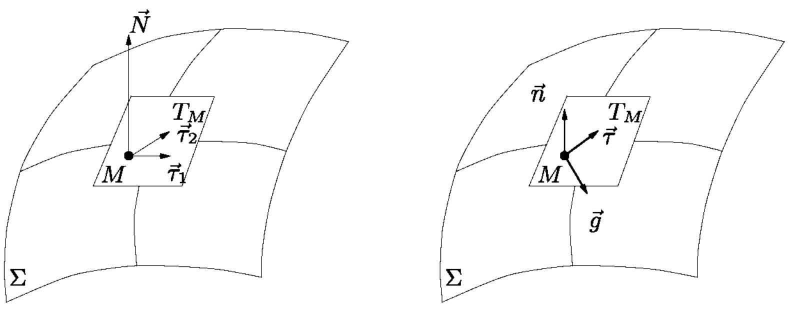


Figure 11.7: *Left-hand side: the local frame at point $M = \sigma(u,v)$ of Σ. Right-hand side: the Darboux frame (moving frame) associated with point $M = \sigma(u,v)$ of Σ and with the tangent $\vec{\tau}$.*

Darboux's frame. The *Darboux frame* $[M, \vec{\tau}, \vec{g}, \vec{n}]$ associated with the pair $(M, \vec{\tau})$ is defined by $\vec{\tau}$, a unit tangent vector at M to Σ, by $\vec{n}$, the unit normal vector and by the vector $\vec{g} = \vec{n} \wedge \vec{\tau}$ (Figure 11.7, right-hand side).

We now consider a regular and oriented arc Γ traced on Σ. The moving frame

$$\mathcal{F}_{moving} = [\gamma(s), \vec{\tau}(s), \vec{g}(s), \vec{n}(s)]$$

is the *Darboux frame* of Γ (associated with a normal parameterization $s \longmapsto \gamma(s)$). Moreover, the vector $\vec{g}(s)$ is called the *geodesic normal vector* to Γ at the point $M = \gamma(s)$.

Given a curve Γ traced on Σ, the tangent $\vec{\tau}$ to Γ at M is one of the vectors of T_M and a moving frame can be associated with this particular tangent vector.

The study of the various curves passing through a point M allows us to capture the behavior of the surface in the vicinity of M. For instance, the intersection of Σ with any of the planes containing the unit normal $\vec{n}$ defines one of the curves we are interested in. Such a plane Π_n is a *normal section* to the surface at M.

Normal curvature, curvature and geodesic torsion

The *normal curvature* of Γ is defined by the function:

$$\kappa_n : s \longmapsto \kappa_n(s) = \left\langle \vec{n}, \frac{d\vec{\tau}}{ds} \right\rangle = -\left\langle \vec{\tau}, \frac{d\vec{n}}{ds} \right\rangle .$$

The *geodesic curvature* of Γ is defined by the function:

$$\kappa_g : s \longmapsto \kappa_g(s) = \left\langle \vec{g}, \frac{d\vec{\tau}}{ds} \right\rangle = -\left\langle \vec{\tau}, \frac{d\vec{g}}{ds} \right\rangle .$$

Finally, the *geodesic torsion* of Γ is defined by the function:

$$T_g : s \longmapsto T_g(s) = \left\langle \vec{g}, \frac{d\vec{n}}{ds} \right\rangle = -\left\langle \vec{n}, \frac{d\vec{g}}{ds} \right\rangle .$$

Using all these definitions, the *Darboux formulae* are the following:

$$\frac{d\vec{\tau}}{ds} = \kappa_g \vec{g} + \kappa_n \vec{n} , \quad \frac{d\vec{g}}{ds} = -\kappa_g \vec{\tau} - T_g \vec{n} , \quad \frac{d\vec{n}}{ds} = -\kappa_n \vec{\tau} + T_g \vec{g} .$$

Second fundamental form

At point $M = M(u, v)$, any vector $\vec{\tau}$ of the tangent plane T_M can be written as $\vec{\tau} = \lambda \vec{\tau}_1 + \mu \vec{\tau}_2$. Let $\vec{n}$ be the unit normal vector to T_M at M defined above. Using the same notations as in the previous section, we have:

$$\kappa_n = \left\langle \vec{n}, \frac{d\vec{\tau}}{ds} \right\rangle = -\left\langle \vec{\tau}, \frac{d\vec{n}}{ds} \right\rangle \quad \text{where} \quad \vec{n}(u, v) = \frac{\vec{N}(u, v)}{H(u, v)} .$$

Developing the first formula, we obtain a relationship of the following form:

$$\kappa_n = L \left(\frac{du}{ds} \right)^2 + 2M \frac{du}{ds} \frac{dv}{ds} + N \left(\frac{dv}{ds} \right)^2 , \qquad (11.19)$$

where:

$$L = \left\langle \vec{n}, \frac{\partial^2 M}{\partial u^2} \right\rangle \qquad M = \left\langle \vec{n}, \frac{\partial^2 M}{\partial u \partial v} \right\rangle \qquad N = \left\langle \vec{n}, \frac{\partial^2 M}{\partial v^2} \right\rangle .$$

Formula (11.19) indicates that the normal curvature of Γ at M depends only on the direction of the unit tangent vector:

$$\vec{\tau} = \frac{\partial M}{\partial u}\frac{du}{ds} + \frac{\partial M}{\partial v}\frac{dv}{ds} .$$

Thus we introduce the second fundamental form of the surface.

Definition 11.8 *Let T_M be the tangent vector plane to the surface Σ at M, the* second fundamental quadratic form *of Σ at M is the quadratic form:*

$$\Phi_2^M(\vec{V}) = L\lambda^2 + 2M\lambda\mu + N\mu^2 ,$$

the values L, M, N being given by the formulae of Relation (11.19).

Expression of Φ_2. Φ_2^M can also be written as a differential form:

$$\Phi_2^M(dM) = Ldu^2 + 2Mdudv + Ndv^2 ,$$

with $dM = \frac{\partial M}{\partial u}du + \frac{\partial M}{\partial v}dv$. Posing

$$\mathcal{M}_2(M) = \begin{pmatrix} L & M \\ M & N \end{pmatrix} ,$$

we can write Φ_2^M as a matrix form (as we did for Φ_1^M):

$$\Phi_2^M(\vec{V}) = ^t\Lambda\, \mathcal{M}_2(M)\, \Lambda .$$

Remark 11.10 *The second fundamental form measures the deviation between the tangent plane and the surface at M.*

Case of a Cartesian parameterization. Let us assume that the surface Σ is parameterized by the Cartesian equation $z = \sigma(x, y)$, where σ is of class C^2 at least. Let $M(x, y)$ be the point of coordinates $(x, y, \sigma(x, y))$, the second fundamental form is:

$$\Phi_2(dM) = \frac{1}{H}(rdx^2 + 2sdxdy + tdy^2) ,$$

with

$$dM = \frac{\partial M}{\partial x}dx + \frac{\partial M}{\partial y}dy \text{ and } H = \sqrt{1 + p^2 + q^2} ,$$

posing: $p = \sigma'_x$, $q = \sigma'_y$, $r = \sigma''_{x^2}$, $s = \sigma''_{xy}$ and $t = \sigma''_{y^2}$.

Computing curvatures and geodesic torsion

The normal curvature κ_n at point M of Σ is related to a tangent vector of the form $\vec{V} = \lambda\vec{\tau}_1 + \mu\vec{\tau}_2$ by the relation:

$$\kappa_n(\vec{V}) = \frac{\Phi_2^M(\vec{V})}{\Phi_1^M(\vec{V})}\,.$$

Whenever $\vec{V}$ varies, this function admits two extrema, the *principal curvatures* to which are associated the two *principal radii of curvature* and the two *principal directions.*

Remark 11.11 *We consider the curve traced on the surface and defined by the segment joining (u,v) to $(u+\lambda, v+\mu)$. We have seen previously that this curve could be approached by a parabola with a limited expansion truncated at order 2. Using Pythagorus's theorem, we can then write:*

$$\left(R(\vec{V}) - \frac{1}{2}\Phi_2^M(\vec{V})\right)^2 + \Phi_1^M(\vec{V}) = R(\vec{V})^2\,,$$

and, neglecting the terms at order 2 before those at order 1, we have:

$$R(\vec{V}) = \frac{\Phi_1^M(\vec{V})}{\Phi_2^M(\vec{V})}\,.$$

All this information can be used to control the quality of the geometric polyhedral approximation of the surface (Chapter 15).

Curvatures. Let κ_1 be the minimal curvature and let κ_2 be the maximal curvature. We define the *Gauss curvature*[10], the *mean curvature* and the *absolute curvature* respectively as:

$$\kappa_{Gauss} = \kappa_1\kappa_2\,,\quad \kappa_{Mean} = \frac{\kappa_1+\kappa_2}{2} \text{ and } \kappa_{Abs} = |\kappa_1| + |\kappa_2|\,.$$

Geodesic torsion. Let Γ be an arc defined by a normal parameterization and traced on Σ. The geodesic torsion of Γ is given by the formula:

$$T_g = \frac{1}{H}\left\langle \frac{\partial M}{\partial u} \wedge \frac{\partial M}{\partial v}\,,\ \vec{\tau} \wedge \frac{d\vec{n}}{ds}\right\rangle\,,$$

which can also be written as:

$$T_g = \frac{1}{H}\begin{vmatrix} \left\langle \vec{\tau}, \dfrac{\partial M}{\partial u}\right\rangle & \left\langle \vec{\tau}, \dfrac{\partial M}{\partial v}\right\rangle \\ \left\langle \dfrac{d\vec{n}}{ds}, \dfrac{\partial M}{\partial u}\right\rangle & \left\langle \dfrac{d\vec{n}}{ds}, \dfrac{\partial M}{\partial v}\right\rangle \end{vmatrix}\,.$$

[10] Also called the *total curvature.*

Meusnier's circle and theorem

We assume that the normal curvature κ_n is not zero and we denote by $\Omega_n = M + R_n\vec{n}$ the center of normal curvature. If Γ is a simple and regular arc traced on Σ and tangent to Σ at M, the center of curvature Ω of Γ at M is defined by:

$$\Omega = M + R\vec{\nu}\,, \qquad \text{with} \quad R = \frac{1}{C} = \frac{\cos\alpha}{\kappa_n} = R_n \cos\alpha\,.$$

where α denotes the angle between the normal $\vec{\nu}$ to Γ and the normal $\vec{n}$ to Σ at M. The point Ω is the projection of Ω_n onto the principal normal to γ at M.

The relation $\kappa_n(\vec{\tau}) = Ccos\alpha$ has an interesting application. If Γ is the intersection of Σ by a normal section, we find that the circle of diameter $\kappa_n(\vec{\tau})$, the *Meusnier circle*, is the geometric locus of points M, endpoints of the segments PM, such that $\|PM\| = C$, the curvature of a curve Γ whose normal $\vec{\nu}$ makes an angle α with the unit normal $\vec{n}$ to Σ. In other words, $\kappa_n(\vec{\tau})$ and α make it possible to obtain the curvature of various curves under this angle.

Local behavior of a surface

In the local study of curves (Section 11.1), we have seen that the curvature gives an indication of the deviation of the curve with respect to the tangent in the vicinity of a point M. Similarly, we will analyze the behavior of a surface with respect to its tangent planes.

Let Σ be a simple and regular surface of class at least C^2, defined by a parameterization σ. We wish study the behavior of the surface Σ in the neighborhood of a point $M = \sigma(u, v)$.

We consider a small increase $\Delta M = (\Delta u, \Delta v)$ in u and v in the neighborhood of M, assumed to be sufficiently small. Applying a Taylor series at order 1 to σ, we have:

$$\sigma(u + \Delta u, v + \Delta v) = \sigma(u, v) + \Delta u\,\frac{\partial M}{\partial u} + \Delta v\,\frac{\partial M}{\partial v} + \mathcal{O}(\|\Delta M\|^2)\,.$$

This local study corresponds to an approximation of the surface by a plane (the tangent plane), noted T_M, directed by the vectors $\vec{\tau}_1$ and $\vec{\tau}_2$ at point M.

To estimate the gap between the surface and the approximation, we pursue the development at order 2 (for a surface smooth enough):

$$\begin{aligned}\sigma(u + \Delta u, v + \Delta v) &= \sigma(u, v) + \Delta u\,\vec{\tau}_1 + \Delta v\,\vec{\tau}_2\\ &+\frac{1}{2}\left(\Delta u^2\,\frac{\partial^2 M}{\partial u^2} + 2\,\Delta u\,\Delta v\,\frac{\partial^2 M}{\partial u\,\partial v} + \Delta v^2\,\frac{\partial^2 M}{\partial v^2}\right) + \mathcal{O}(\|\Delta M\|^3)\,.\end{aligned}$$

In this expression, the term of order 2 enables us to measure the desired gap. The local analysis corresponds to an approximation of the surface by a paraboloid (a quadric) $\mathcal{P}$. Notice that in this development, we can exhibit the terms L, M and N previously introduced by observing the projection on $\vec{n}$, we have:

$$\langle\Delta u^2\,\frac{\partial^2 M}{\partial u^2} + 2\,\Delta u\,\Delta v\,\frac{\partial^2 M}{\partial u\,\partial v} + \Delta v^2\,\frac{\partial^2 M}{\partial v^2}\,,\,\vec{n}\rangle \;=\; \Delta u^2\,L\, + 2\Delta u\Delta v\,M\, +\, \Delta v^2\,N\,,$$

that makes use of the second fundamental form Φ_2 of the surface at M. The nature of the form Φ_2 corresponds to a local geometric property of Σ at the point M.

In the Taylor series at order 2, the second order term allows us to determine whether the tangent plane T_M intersects the surface in the neighborhood of M.

Dupin's indicatrix. Using the polar coordinates, $r = \sqrt{\rho}$ and θ, a point P of the tangent plane T_M at M can be written as:

$$x = \sqrt{\rho}\cos\theta \qquad \text{and} \qquad y = \sqrt{\rho}\sin\theta\,,$$

where θ denotes the angle between one of the principal directions (the other being orthogonal) and the line MP. We can then use the Euler relation [do Carmo-1976]:

$$\kappa_n = \kappa_1 \cos^2\theta + \kappa_2 \sin^2\theta\,,$$

relating the minimal and maximal curvatures to the normal curvature and, then, we can write:

$$\kappa_1 x^2 + \kappa_2 y^2 = \frac{x^2}{\rho_1} + \frac{y^2}{\rho_2} = \pm 1\,. \tag{11.20}$$

which corresponds to the equation of a conic section, called the *Dupin indicatrix.* Notice, by the way, that a change in the orientation of the normal does not change the sign of ρ, thus the symbol $\pm$ in the relation.

Let us now consider a plane T_ε, parallel to the tangent plane T_M at M and located at a distance ε of T_M. Considering the Taylor series at order 2 of σ, the trace of the quadric in this plane is such that:

$$\langle \Delta u^2 \frac{\partial^2 M}{\partial u^2} + 2\,\Delta u\,\Delta v \frac{\partial M}{\partial u\,\partial v} + \Delta v^2 \frac{\partial^2 M}{\partial v^2}\,,\,\vec{n}\rangle = \varepsilon\,,$$

The idea is here to consider that the Dupin indicatrix represents the intersection of the surface (locally approached by the paraboloid) and the plane T_ε (within a scale factor ζ, such that $\varepsilon = \frac{1}{2\,\zeta^2}$).

We study the behavior of the curve defined by Equation (11.20) in the plane T_ε, according to the curvatures κ_1 and κ_2, by noticing that the conic is a curve not depending on the parameterization of Σ. This curve is (Figure 11.8):

- an ellipse, when $\kappa_1\,\kappa_2 > 0$,
- two parallel lines (at a distance of $2\rho_1$ or $2\rho_2$), when $\kappa_1 = 0$ or $\kappa_2 = 0$,
- a pair of hyperbolas, when $\kappa_1\,\kappa_2 < 0$.

From a geometric point of view, when the distance ε between T_ε and T_M tends towards 0, the curve is reduced down to a contact point only. When decreasing the distance, the curve is scaled up continuously, we observe that this curve approaches an ellipsis (in the case of an elliptical point) whose center is the contact point.

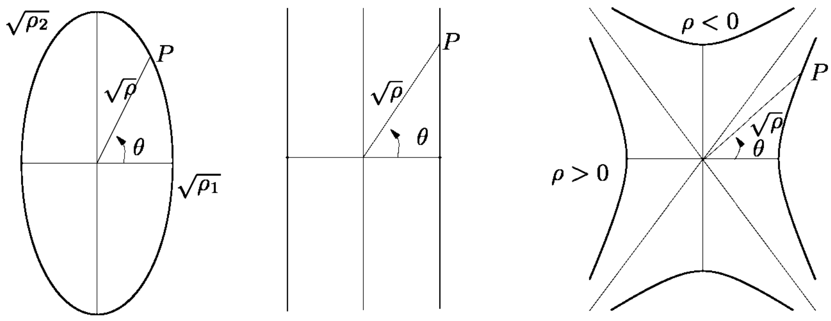


Figure 11.8: *Dupin's indicatrix: elliptical point, $\kappa_1 \kappa_2 > 0$ (left-hand side), parabolic point, $\kappa_1 = 0$ or $\kappa_2 = 0$ (middle) and hyperbolic point, $\kappa_1 \kappa_2 < 0$ (right-hand side).*

11.3 Computational issues about surfaces

Here, we will focus on problems which arise when dealing with surfaces.

Curvature computation

Let Σ be a simple regular and oriented surface of class C^k $(k \geq 2)$ defined by a parameterization $(u,v) \longmapsto M(u,v)$. Consider a curve Γ traced in a normal section Π_n at a point M to the surface.

The curvature of Γ at a point M characterizes the geometry of the curve in the neighborhood of this point. The *normal curvature* characterizes the surface in the same neighborhood. The expression of the curvature of Γ shows the tangent vector $\vec{\tau}$ and the unit normal $\vec{\nu}$ as follows:

$$\frac{d\vec{\tau}(t)}{ds} = C\vec{\nu}(t) \quad \text{or} \quad C = \left\| \frac{d^2\gamma(t)}{ds^2} \right\| ,$$

whereas the normal curvature of the surface at M is characterized by:

$$\kappa = \kappa_n(\vec{\tau}) = \left\langle \frac{d\vec{\tau}(s)}{ds}, \vec{n} \right\rangle = \left\langle \frac{d^2\gamma(s)}{ds^2}, \vec{n} \right\rangle .$$

In fact, $\kappa_n(\vec{\tau})$ measures the curvatures of the surface with respect to its unit normal $\vec{n}$ at M in the direction $\vec{\tau}$. Thus, we have: $\kappa_n(\vec{\tau}) = C \cos\alpha$ where α is the angle between $\vec{\nu}$ and $\vec{n}$.

Remark 11.12 *If $\vec{n} = \vec{\nu}$ (when these two vectors are colinear), the curve Γ is called a* geodesic. *In other words, at point M, the unit normal $\vec{n}$ to Σ is the* geodesic normal *of γ.*

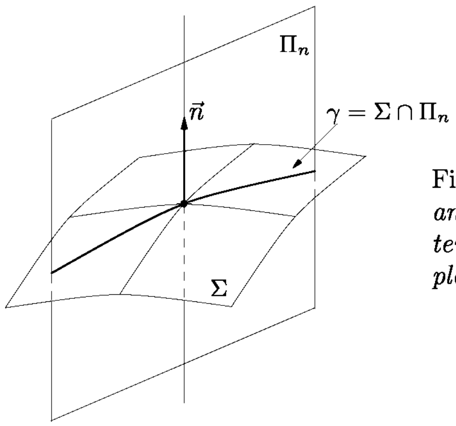


Figure 11.9: *A normal section* Π_n *and the curve* Γ *representing the intersection of the surface with this plane.*

As we assumed a parameterization σ of Σ, the relations

$$\gamma(t) = \sigma(u(t), v(t)) \quad \text{or} \quad \gamma(s) = \sigma(u(s), v(s)),$$

are established, the characteristics of Σ (i.e., its normal curvatures) can be obtained with respect to σ and to its derivatives.

As $\langle \vec{n}, \vec{\tau} \rangle = 0$,

$$\kappa = \left\langle \frac{d\vec{\tau}(s)}{ds}, \vec{n} \right\rangle \quad \text{is also} \quad \kappa = -\left\langle \vec{\tau}, \frac{d\vec{n}}{ds} \right\rangle .$$

Denoting $\vec{n} = \frac{\vec{\tau}_1 \wedge \vec{\tau}_2}{H}$, we have $H\vec{n} = \vec{\tau}_1 \wedge \vec{\tau}_2$ and using again $\langle \vec{n}, \vec{\tau} \rangle = 0$,

$$\left\langle \vec{\tau}, \frac{dH\vec{n}}{ds} \right\rangle = \left\langle \vec{\tau}, \frac{dH}{ds}\vec{n} \right\rangle + H \left\langle \vec{\tau}, \frac{d\vec{n}}{ds} \right\rangle$$

is finally reduced to:

$$\left\langle \vec{\tau}, \frac{dH\vec{n}}{ds} \right\rangle = H \left\langle \vec{\tau}, \frac{d\vec{n}}{ds} \right\rangle ,$$

hence

$$\kappa = -\left\langle \vec{\tau}, \frac{d\vec{n}}{ds} \right\rangle = -\frac{1}{H} \left\langle \vec{\tau}, \frac{d(\vec{\tau}_1 \wedge \vec{\tau}_2)}{ds} \right\rangle .$$

Now, depending on u and v, as

$$\frac{d(\vec{\tau}_1 \wedge \vec{\tau}_2)}{ds} = \left(\frac{d\vec{\tau}_1}{ds} \wedge \vec{\tau}_2) + (\vec{\tau}_1 \wedge \frac{d\vec{\tau}_2}{ds} \right)$$

with

$$\frac{d\vec{\tau}_1}{ds} = \frac{\partial \vec{\tau}_1}{\partial u}\frac{du}{ds} + \frac{\partial \vec{\tau}_1}{\partial v}\frac{dv}{ds} \quad \text{and} \quad \frac{d\vec{\tau}_2}{ds} = \frac{\partial \vec{\tau}_2}{\partial u}\frac{du}{ds} + \frac{\partial \vec{\tau}_2}{\partial v}\frac{dv}{ds}$$

we thus have:

$$\frac{d(\vec{\tau}_1 \wedge \vec{\tau}_2)}{ds} = \left(\left(\frac{\partial \vec{\tau}_1}{\partial u}\frac{du}{ds} + \frac{\partial \vec{\tau}_1}{\partial v}\frac{dv}{ds} \right) \wedge \vec{\tau}_2 \right) + \left(\vec{\tau}_1 \wedge \left(\frac{\partial \vec{\tau}_2}{\partial u}\frac{du}{ds} + \frac{\partial \vec{\tau}_2}{\partial v}\frac{dv}{ds} \right) \right) .$$

Finally, the expression of the curvature κ_n is given by the following formula:

$$\frac{-1}{H}\left\langle\left(\vec{\tau}_1\frac{du}{ds}+\vec{\tau}_2\frac{dv}{ds}\right),\left(\left(\frac{\partial\vec{\tau}_1}{\partial u}\frac{du}{ds}+\frac{\partial\vec{\tau}_1}{\partial v}\frac{dv}{ds}\right)\wedge\vec{\tau}_2\right)+\left(\vec{\tau}_1\wedge\left(\frac{\partial\vec{\tau}_2}{\partial u}\frac{du}{ds}+\frac{\partial\vec{\tau}_2}{\partial v}\frac{dv}{ds}\right)\right)\right\rangle$$

then, for instance, L, the coefficient depending on $(du/ds)^2$ is:

$$L=-\frac{1}{H}\left\langle\vec{\tau}_1,\left(\frac{\partial\vec{\tau}_1}{\partial u}\wedge\vec{\tau}_2+\vec{\tau}_1\wedge\frac{\partial\vec{\tau}_2}{\partial u}\right)\right\rangle=-\frac{1}{H}\left\langle\vec{\tau}_1,(\sigma''_{uu}\wedge\vec{\tau}_2)\right\rangle=\frac{1}{H}\left\langle\vec{\tau}_1\wedge\vec{\tau}_2,\sigma''_{uu}\right\rangle$$

which is simply:

$$\left\langle\frac{\vec{\tau}_1\wedge\vec{\tau}_2}{H},\sigma''_{uu}\right\rangle \quad \text{i.e.,} \quad \left\langle\vec{n},\sigma''_{uu}\right\rangle,$$

previously defined. Similarly, we can demonstrate that:

$$M=\left\langle\vec{n},\sigma''_{uv}\right\rangle \quad \text{and} \quad N=\left\langle\vec{n},\sigma''_{vv}\right\rangle.$$

Exercise 11.1 *Establish the previous results for M and N (hint: find the terms in $du/ds\,dv/ds$ and $(dv/ds)^2$, respectively).*

Using these values, the normal curvature is given by:

$$\kappa=L\left(\frac{du}{ds}\right)^2+2\,M\frac{du}{ds}\frac{dv}{ds}+N\left(\frac{dv}{ds}\right)^2 \quad \text{or} \quad \kappa=\frac{Ldu^2+2\,Mdudv+Ndv^2}{ds^2}$$

i.e.,

$$\kappa_n(\vec{\tau})=\kappa=\frac{Ldu^2+2\,Mdudv+Ndv^2}{Edu^2+2\,Fdudv+Gdv^2}=\frac{\Phi_2^M(\vec{\tau})}{\Phi_1^M(\vec{\tau})} \tag{11.21}$$

according to a given direction $\vec{\tau}$ such that $\vec{\tau}=du\vec{\tau}_1+dv\vec{\tau}_2$. Thus, we find the relation between the two fundamental forms (evaluated in $\vec{\tau}$) introduced previously.

Normal curvature analysis

Let λ and μ be two parameters and consider $\vec{\tau}=\lambda\vec{\tau}_1+\mu\vec{\tau}_2$. Using the previously established result, we can write:

$$\kappa_n(\vec{\tau})=\kappa_n(\lambda,\mu)=\frac{L\lambda^2+2\,M\lambda\mu+N\mu^2}{E\lambda^2+2\,F\lambda\mu+G\mu^2},$$

the aim is to determine how this function varies. To this end, we study its variation with respect to the two parameters. We start by searching the extrema of this function, which are given by the relations:

$$\frac{\partial\kappa_n(\lambda,\mu)}{\partial\lambda}=0 \quad \text{and} \quad \frac{\partial\kappa_n(\lambda,\mu)}{\partial\mu}=0.$$

These two relations lead to a unique equation:

$$(FL-EM)\lambda^2+(GL-EN)\lambda\mu+(GM-FN)\mu^2=0. \tag{11.22}$$

Notice that this relation can also be written in the following form:

$$det \begin{vmatrix} \mu^2 & -\lambda\mu & \lambda^2 \\ E & F & G \\ L & M & N \end{vmatrix} = 0 . \tag{11.23}$$

If the three coefficients of the relations are zero, then the two fundamental forms are proportional, irrespective of the values of $\vec{\tau}$ (i.e., for all (λ, μ)) and the curvature is constant along the normal sections. In this case, the constant curvature is then:

$$\kappa_n(\vec{\tau}) = \kappa_n = \frac{L}{E} = \frac{M}{F} = \frac{N}{G} .$$

In the other cases, this equation admits two distinct solutions (i.e., two pairs (λ_1, μ_1) and (λ_2, μ_2)). With each of these pairs is associated a vector, respectively:

$$\vec{V_1} = \lambda_1 \vec{\tau_1} + \mu_1 \vec{\tau_2} \quad \text{and} \quad \vec{V_2} = \lambda_2 \vec{\tau_1} + \mu_2 \vec{\tau_2} .$$

$\vec{V_1}$ and $\vec{V_2}$ are orthogonal, i.e., $\langle \vec{V_1}, \vec{V_2} \rangle = 0$. They define the *two principal directions.* The vectors

$$\vec{W_1} = \frac{\vec{V_1}}{||\vec{V_1}||} \quad \text{and} \quad \vec{W_2} = \frac{\vec{V_2}}{||\vec{V_2}||}$$

form with $\vec{n}$ an orthonormal basis called the *local principal basis* at point M and denoted as:

$$\mathcal{B}_M = [M, \vec{W_1}, \vec{W_2}, \vec{n}] .$$

With each principal direction is associated a *principal curvature*, which is the solution of:

$$det \begin{vmatrix} \kappa E - L & \kappa F - M \\ \kappa F - M & \kappa G - N \end{vmatrix} = 0 . \tag{11.24}$$

To summarize, the surface Σ at point M is characterized by its local principal basis $\mathcal{B}_M$, its two principal curvatures $\kappa_1 = \kappa_n(\lambda_1, \mu_1)$ and $\kappa_2 = \kappa_n(\lambda_2, \mu_2)$ and thus by its two principal radii of curvature $\rho_1 = 1/\kappa_1$ and $\rho_2 = 1/\kappa_2$ whose variations allow us to determine locally the geometry of the surface.

Remark 11.13 *By comparing this result with the expression of the geodesic torsion defined previously, we again find the fact that the two principal directions are the directions for which the geodesic torsion is zero.*

Remark 11.14 *All the functions introduced in this section will serve to define the requisites needed to control the meshing process of surfaces (Chapter 15).*

Relationship between curvatures and fundamental forms

We suggest here a different way of establishing the results given above. To this end, let us consider the two metrics associated with the two fundamental forms. According to the previous sections, we already know that:

$$\mathcal{M}_1(M) = \begin{pmatrix} E & F \\ F & G \end{pmatrix} ,$$

is attached to the first fundamental form and that:

$$\mathcal{M}_2(M) = \begin{pmatrix} L & M \\ M & N \end{pmatrix},$$

corresponds to the second fundamental form.

Let us consider the matrix $\mathcal{N} = \mathcal{M}_1^{-1}\mathcal{M}_2$. This matrix is diagonalizable (as it is a $\mathcal{M}_1$-symmetric matrix). Let $\vec{W}_1$ and $\vec{W}_2$ be the two unit eigenvectors of $\mathcal{N}$. Then, for each vector $\vec{V}$ of T_M, expressed according to $\vec{W}_1$ and $\vec{W}_2$, we have:

$$\Phi_1(\vec{V}) = \alpha^2 + \beta^2 \quad \text{and} \quad \Phi_2(\vec{V}) = \kappa_1\alpha^2 + \kappa_2\beta^2$$

leading to:

$$\kappa_n(\vec{V}) = \frac{\Phi_2(\vec{V})}{\Phi_1(\vec{V})} = \frac{\kappa_1\alpha^2 + \kappa_2\beta^2}{\alpha^2 + \beta^2}.$$

The extrema of $\kappa_n(\vec{V})$ are obtained for $\alpha = 0$ or $\beta = 0$. Hence, κ_1 and κ_2 are naturally extrema (in the previous expression). The corresponding directions (i.e., the $\vec{V}$'s) are $\vec{W}_1$ (for $\beta = 0$) and $\vec{W}_2$ (for $\alpha = 0$).

Remark 11.15 *In fact, finding the principal directions is only a geometric interpretation of the problem of the simultaneous reduction of two quadratic forms (Chapter 10).*

Local behavior of a surface

The analysis of the local principal basis when M varies on Σ allows us to capture the local behavior of the surface. Indeed, the extremal values of κ, the curvature related to a point, characterize the type of the surface at point M. We can also write:

$$\kappa^2 - (\kappa_1 + \kappa_2)\kappa + \kappa_1\kappa_2 = 0,$$

or

$$\kappa^2 - 2\,\kappa_{Mean}\kappa + \kappa_{Gauss} = 0. \tag{11.25}$$

Thus we have:

$$\kappa_{Mean} = \frac{1}{2}\frac{NE - 2MF + LG}{EG - F^2} \quad \text{and} \quad \kappa_{Gauss} = \frac{LN - M^2}{EG - F^2}.$$

When (cf. Figure 11.10):

- $\kappa_{Gauss} > 0$, the point M is said to be *elliptic*,
- $\kappa_{Gauss} < 0$, the point M is said to be *hyperbolic*,
- $\kappa_{Gauss} = 0$, the point M is said to be *parabolic*.

Obviously, when κ_{Gauss} and κ_{Mean} both tend towards 0, the surface is *planar* at M.

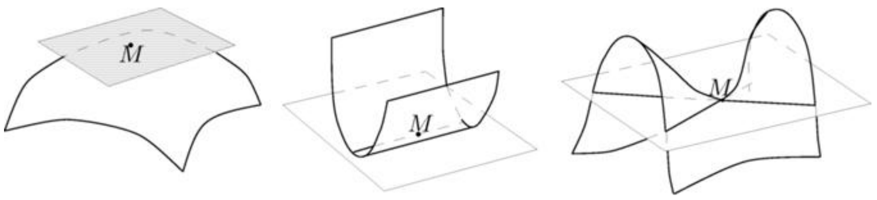


Figure 11.10: *Position of a surface with respect to a tangent plane. Elliptic point (left-hand side), parabolic point (middle) and hyperbolic point (right-hand side).*

Remark 11.16 *For a point M of a simple and regular surface to be* elliptic, *it is necessary and sufficient that the second fundamental form is defined (positive or negative). For a point M to be* hyperbolic, *it is necessary and sufficient that the second fundamental form Φ_2^M is non-degenerated and non-defined. Finally, for a point M to be* parabolic, *it is necessary and sufficient that Φ_2^M is degenerated.*

11.4 Non-linear problems

The definitions of curves and surfaces usually leave out some problems. For instance, the intersection procedures between two curves, two pieces of surfaces (patches such as those defined in Chapter 13) theoretically lead to solving a non-linear equation. In the general case, these problems can also be solved explicitly and therefore require the development of numerical approximation methods. Some of these methods are presented in this section.

Non-linear issues

Let Γ_1 and Γ_2 be two parametric curves associated with the functions $\gamma_1(t)$ and $\gamma_2(t)$. The intersection of these curves is defined by the set of pairs of values:

$$(t_1, t_2) \text{ such that } \gamma_1(t_1) = \gamma_2(t_2)$$

leading to a non-linear equation. Similarly let us consider Σ_1 and Σ_2 two bi-parametric patches, associated with the functions $\sigma_1(u_1, v_1)$ and $\sigma_2(u_2, v_2)$. The intersection between Σ_1 and Σ_2 is defined by the set of pairs of vectors:

$$(u_1, v_1, u_2, v_2) \text{ such that } \sigma_1(u_1, v_1) = \sigma_2(u_2, v_2)$$

also leading to a non-linear equation. Finally, let us consider the intersection between a patch and a curve; we then have the following (non-linear) equation to solve:

$$(u_1, v_1, t) \quad \text{such that} \quad \sigma_1(u_1, v_1) = \gamma_1(t_1)\,.$$

Moreover, the projection and the search for the closest point on such a curve from a given point according to a specific direction leads to a non-linear equation of the following type:

$$t \text{ such that } ||M\gamma(t)|| \text{ is minimum, or also } \frac{d||M, \gamma(t)||}{dt} = 0\,.$$

In such cases, the problems can be solved with the algorithms described in the following sections. However, we should also mention here the algebraic methods. This type of method is usually based on an implicit definition of $\vec{F}$ ($\vec{F}$ being a vector with d components representing the problem to be solved), $\vec{F} = 0$. More details about such methods can be found in the literature, for example in [Sederberg-1987]. We will not spend more time here on these issues as they are seldom used in this context.

Newton-Raphson type algorithms

Let us consider $\vec{F}(x_1, x_2, ...x_n) = \vec{0}$ as the problem to be solved, for which $\vec{F}$ is a vector with m components. We are searching for the best vector (i.e., the optimal vector, in a certain sense) $\vec{x} = (x_1, x_2, ...x_n)$ such that the equation is guaranteed. The basic idea of a *Newton-Raphson*-type method is to identify the zeros of the vector function $\vec{F}$, using the following algorithm:

Algorithm 11.1 *Newton-Raphson algorithm*

```
Initialize $\vec{x}$ randomly,
WHILE $\|F(\vec{x})\| \geq \varepsilon$
    in first approximation, consider the asymptotic development
    of $\vec{F}$:
        $F_j(\vec{x} + d\vec{x}) = F_j(\vec{x}) + \frac{\partial F_j}{\partial x_1}dx_1 + \frac{\partial F_j}{\partial x_2}dx_2 + ... + \frac{\partial F_j}{\partial x_n}dx_n$
          where $j \in [1, m]$
    solve the linear system $F_j(\vec{x} + d\vec{x}) = 0$ with $d\vec{x}$ as unknown
    set $\vec{x}$ to the value $\vec{x} + d\vec{x}$
    compute $F(\vec{x})$
END WHILE.
RETURN $\vec{x}$
```

Remark 11.17 *This algorithm yields a unique value for $\vec{x}$. This solution greatly depends on the value used to initialize $\vec{x}$. If more than one solution exists, only one is returned by the algorithm. Hence, the solution found may not be the global minimum of the function.*

Figure 11.11 illustrates the behavior of the algorithm for a function of one parameter. Solution x_3 is the unique solution returned by the algorithm, whereas three solutions can be shown.

Remark 11.18 *In some cases, the system to be solved is under-determined, i.e., $m \leq n$. In such cases, new equations need to be added, depending on the context, to find the best direction for the vector $\vec{F}$.*

Divide and conquer algorithm

Newton-Raphson-type algorithms do not allow a complete analysis in the interval in which the solution is assumed to be. The *divide and conquer* strategy attempts

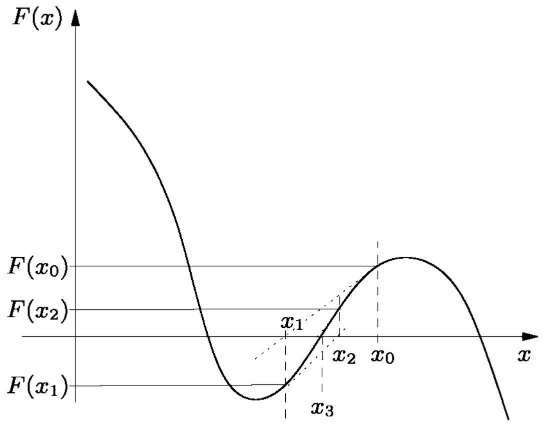


Figure 11.11: *Resolution of an equation using a Newton-Raphson method.*

to identify the different sub-domains where a solution might exist. The main idea is to bound the values of $\vec{F}$ in an interval. If the interval contains 0, the latter is then subdivided into m (usually $m = 2$) sub-domains. This process is then repeated on each sub–domain. Each branch of the binary tree structure associated with this method is analyzed and recursively subdivided until a given minimal size is reached.

Remark 11.19 *Such an algorithm does not strictly speaking solve the initial problem. In fact, it identifies the various sub-domains where a solution may be found. From a practical point of view, a Newton-Raphson algorithm is then employed to find the solution in each interval.*

Remark 11.20 (Method based on a grid) *This type of method starts by defining a grid* a priori *over the computational domain. Then, a linear interpolation of $\vec{F}$ is constructed. The non-linear problem is solved using the linear interpolation of $\vec{F}$. Seen from this point of view, this method is another way of subdividing the domain in such a way so as to localize the various possible solutions. From a practical point of view, the grid used is a regular grid in the space of $\vec{x}$.*

Chapter 12

Curve Modeling

A curve can be defined using various categories of methods. Indeed, there are parametric, implicit or explicit curves. Using classical notations, a parametric curve is given by a function γ and a parameter t. Then, given t in some interval, the curve is defined by $\gamma(t)$ where $\gamma(t) \in \mathbb{R}^d$, $d = 2$ or 3. Implicit curves in $\mathbb{R}^2$ are given via a relation like $f(x, y) = 0$ where x and y denote the coordinates. Explicit curves in $\mathbb{R}^2$ are defined by the pair $(x, y = f(x))$. In $\mathbb{R}^3$, non-parametric curves are defined as the intersection of two surfaces.

In principle, function f or γ depends on modeling parameters $(p_i)_{i\in[1,np]}$. These parameters can be defined in the following ways:

- through a direct definition of the design variables: with this approach, the user of a CAD system directly defines the values of the parameters p_i using a graphic interface. This way of processing is actually the simplest but it requires reasonably precise knowledge about the underlying models and their intrinsic variables,

- through the specification of the curve characteristics: the curve is defined by means of high level characteristics but the model, as completed by the CAD system, does not necessarily store this information. For instance, a circle can be defined while a NURBS type curve is generated inside the system. This type of curve specification includes the curve generation method based on a set of constraints,

- through an interpolation method: a set of points P_i and, for some of these methods, a set of derivatives at these points are supplied which form the parameters (the p_i's). Then, an interpolation procedure constructs a curve that passes through all of these points,

- through a smoothing technique: a set of points P_i (and, as above some derivatives for some methods) is supplied. A smoothing algorithm then completes a curve that passes through the given points (or only some of them) while ensuring that a given criterion is minimized,

- through a combination of several of the above approaches.

★
★ ★

In this chapter we discuss the methods that are the most widely used in practice (in CAD systems), namely a parametric definition. Implicit curves will be discussed in Chapter 16 while explicit curves are mostly of academic interest or are used only in some particular cases.

In the first section, we introduce the main ideas of the interpolation or smoothing based methods for parametric curve modeling. In the following sections, we describe the models that make use of a control polygon which form the basis of most curve definition systems.

Curves are defined from a set of points and various types of functions. First, a global definition can be constructed, which means that a single function is used to define the curve from point P_0 to point P_n, irrespective of the range of n. Second, the curve from P_0 to P_n is defined using several segments whose junctions verify some regularity. In the first case, the curve passes through P_0 and P_n and passes through the P_is, $(i = 1, n-1)$, and we encounter an interpolation method or does not pass through these points and we encounter an extrapolation or a smoothing based method.

Among interpolation methods we first find methods such as Lagrange interpolates using the control points as input data and Hermite interpolates where derivatives (tangents at the control points) are also involved. We can find other types of definitions such as the well-known Bézier method based on Bernstein polynomials of degree n and the rational Bézier method using rational polynomials also of degree n.

While good in terms of regularity, such global definitions may lead to some problems of a different nature (complexity, unnecessary computational effort, oscillations, etc.). As a consequence, when n is large, methods with a low degree (such as 3 or 4) have been introduced. Then, whatever the value of n, a series of segments is defined where a low degree method is developed. We then obtain a good local regularity and the issue is to make sure that the junctions between two consecutive segments is smooth enough to obtain sufficient global regularity. In this class of methods we can first use low degree Bézier (or similar) definitions or *Spline*-based methods leading to B-spline or more generally to NURBS-based methods.

The first sections aim at briefly introducing the methods that have been mentioned above. We do not claim to be exhaustive and for a complete view of curve definitions we refer the reader to the *ad-hoc* literature; i.e., [Mortenson-1985], [Bartels *et al.* 1987], [Bartels *et al.* 1988], [Léon-1991] or [Farin-1997] among many others. At the end of the chapter, some numerical problems regarding curve manipulation are discussed based on some comprehensive examples.

Notations. In what follows, t is the parameter and $\gamma(t)$ is the parametric expression of the curve we are interested in (the curve is usually noted by Γ). If need be, we do not distinguish between $\gamma(t)$ (which for a given value of t is a point) and Γ (the whole curve).

12.1 Interpolation and smoothing techniques

As we are concerned with parametric curves, the function γ defining such a curve has the form:

$$\gamma(t) = \gamma(p_1, p_2, ..., p_{np}, t)\,,$$

where the p_i's are the parameters defining the shape of model γ (i.e., that of curve Γ) and t is a real value parameter.

In the following we consider $n+1$ control points $P_0, P_1, ..., P_n$ in $\mathbb{R}^d$. The problem is to define a curve using a function and these points, then various questions must be addressed including how to define these points and what type of functions must be constructed so as to obtain a suitable and easy to manipulate curve.

The purpose of any interpolation or smoothing method is to find the values of the parameters p_i involved in the definition, such that the resulting curve Γ is representative, in some way, of the given set of points $P_0, P_1, ..., P_n$. At first, we are not interested in what the functions γ used to define the curve are. Then, for a specific interpolation or smoothing method (using a parameterization of the whole set of points), there exists a unique parameter that can be associated with any point in this set. The given points define a *polyline* and an interpolation or smoothing technique must complete a curve close to this line.

Parameterization of a set of points

The issue is to associate the parameters t_i with the points in the set P_i. At least, we must achieve a certain correspondence[1] between $\gamma(t_i)$ and P_i. Once the points have been sorted, the first assumption is that the t_i's are ordered in an increasing order: $t_i \leq t_{i+1}$.

This being satisfied, any set of t_i's is valid *a priori*. Nevertheless, for simplicity and simple automatization, the most frequent choices correspond to:

- a uniform parameterization: the parameters are uniformly spaced in the given interval,
- a parameterization related to distances between the points: the parameters are then spaced in accordance with the length of the segments $[P_i, P_{i+1}]$.

Uniform parameterization. If the interval of parameters is $[t_{min}, t_{max}]$, then the t_i are defined following the rule:

$$t_i = t_{min} + \frac{i}{n} \times (t_{max} - t_{min})$$

Remark 12.1 *Notice that the curvilinear abscissa along the approached polyline does not follow, in general, the distribution of the t_i's. For instance, the parameter value $\dfrac{t_{min} + t_{max}}{2}$, the middle of the interval $[t_{min}, t_{max}]$ is not, in general, the parameter of the midpoint of this polyline.*

[1]Such a parameterization is not strictly required. The purpose is to find the np parameters. The t_is are only additional parameters. If, in addition, there are more equations than unknowns, then the parameterization allows us to reduce the number of parameters by fixing some t_i.

Parameterization conforming to the length ratios. The parameters t_i's can be defined in such a way as the ratio between two successive parameter values is exactly the ratio between the lengths of the two corresponding consecutive points:

$$\frac{t_{i+1} - t_i}{t_{max} - t_{min}} = \frac{\|P_{i+1} - P_i\|}{\sum\limits_{j=0}^{n-1} \|P_{j+1} - P_j\|} \quad ; \quad t_0 = t_{min}$$

where t_{min} and t_{max} are the bounds of the interval. In this case, the series of the t_i's can be obtained using the following algorithm.

Algorithm 12.1 *Parameterization following the length ratios.*

```
$t_0 = t_{min}$
$L = \sum_{j=0}^{n-1} \|P_{j+1} - P_j\|$
FOR $i = 1$ to $n$
    $t_i = t_{i-1} + (t_{max} - t_{min}) \times \frac{\|P_i - P_{i-1}\|}{L}$
END FOR $i$.
```

Remark 12.2 *Taking these length ratios allows for a better match between the variation in curvilinear abscissa and that of the parameters but, however, does not lead to the proportional ratio between these two values, therefore* $\frac{ds}{dt} \neq cte$.

Interpolation based methods

The interpolant properties of any curve definition method are related to the fact that the distance between the curve and any point P_i is zero. The best way to ensure this property is to impose that:

$$\gamma(p_1, p_2, ..., p_{np}, t_i) = P_i \, , \forall \, i \, \in \, [0, n] \, . \tag{12.1}$$

Since the values of the t_i's must be chosen so as to be suitable for all the previously mentioned methods, we have $(n + 1) \times d$ equations with np unknowns. Then we encounter three cases:

- $np \leq (n + 1) \times d$ and the interpolation problem is *over-determined*: System (12.1) cannot be solved,
- $np = (n + 1) \times d$ and the interpolation problem is well-posed: System (12.1) can be solved. However, since γ is non-linear with respect to the parameters p_i's, the system to be solved is a *non-linear* system,
- $np \geq (n + 1) \times d$ and the interpolation problem is *under-determined*: System (12.1) must be completed with additional conditions, for instance, about tangencies, curvature values, etc.

Exercise 12.1 *Interpolate a circle from the following series of data points (when this makes sense):*

- $\{(0,0);(5,8)\}$,
- $\{(0,0);(5,8);(3,1)\}$ *and*
- $\{(0,0);(5,8);(3,1);(-4,2)\}$.

Hint: in the case of two points, fix one parameter t_1 or t_2 so as to define a specific circle (there is an infinity of circles passing through two points). With 3 points, the t_is result from a non-linear system (which must be solved using an adequate technique; cf. Chapter 11). With 4 points, there is no longer an interpolation solution in most cases.

Smoothing based methods

Interpolation techniques are not suitable when the corresponding system is over-determined. In this case, smoothing techniques allow for the definition of a curve which is close, in some sense, to the set of points. This notion of a proximity is evaluated with respect to a criterion C:

$$C(p_1, p_2, ..., p_{np}, P_0, P_1, ..., P_n)\,.$$

The aim is then to minimize this criterion C for the set of p_is. To this end, we construct the system:

$$\frac{\partial C}{\partial p_i} = 0\,.$$

In this system, there is still the same number of parameters and equations. Thus, this system can be solved. However, it could be non-linear, meaning that appropriate methods must be used. The reader may refer to Section 11.4 where some methods suitable for this purpose are discussed.

Remark 12.3 (Linear regression by means of a least square method) *In numerous cases, the square of the distances is used in the definition of C:*

$$C(p_1, p_2, ..., p_{np}, P_0, P_1, ..., P_n) = \sum_{i=0}^{n} \|P_i - \gamma(p_1, p_2, ..., p_{np}, t_i)\|^2\,.$$

If the curve corresponding to this series of points is a line, γ is linear with its parameters. Thus, C is a polynomial of degree 2 and the resulting system is a linear system. Indeed, we turn to the system of the classical linear regression.

Remark 12.4 *When the p_is have been fully defined, $C(p_1, p_2, ..., p_{np}, P_0, P_1, ..., P_n)$ is used to judge the quality of the curve in its approximation of the set of points.*

Exercise 12.2 *Generate a circle approximating the series of points in Exercise 12.1 when a least square criterion is prescribed.*

Combined methods

When the number of constraints, interpolation points, tangency conditions, curvatures, etc., is less than the number of parameters, a smoothing method can be combined with an interpolation technique. Then, the minimization of criterion C must be made under this interpolation constraint.

12.2 Lagrange and Hermite interpolation

Lagrange's interpolation scheme

After this general survey, we turn to the first class of interpolation methods, called the *Lagrange* type method. In this case, the control points are associated with a series of parameter values t_i, $(i = 0, n)$, where t_i corresponds to P_i. The modeling parameters, the p_is, are the P_is and the parameter t consists of the set of t_is. Then the interpolation function:

$$\gamma(t) = \sum_{i=0}^{n} \phi_i(t) P_i \quad \text{with} \quad \phi_i(t) = \prod_{l=0}^{n} \frac{t - t_l}{t_i - t_l} \quad \text{for } l \neq i \tag{12.2}$$

defines a curve, called the *Lagrange interpolate* of degree n. This curve is governed by the $n+1$ given points and *passes through* these points: in fact, since $\phi_i(t_j) = \delta_{ij}$ where δ_{ij} is the Kronecker delta, we have $\gamma(t_i) = P_i$.

Remark 12.5 *The ϕ_is form a basis of all polynomials of degree n. Moreover, they sum to 1.*

Remark 12.6 *$\gamma(t)$ can be written by means of the monomial basis. Following this form, we have*

$$\phi_i(t) = \sum_{i=0}^{n} a_i t^i \quad \text{or} \quad \gamma(t) = \sum_{i=0}^{n} A_i t^i \,,$$

where the A_is are indeed combinations of the given P_is.

This interpolation function is C^∞ which is, at the same time, good but probably unnecessary. On the other hand, when n is large this curve definition may produce oscillations and lead to an expensive computational effort. Thus, this type of polynomial function is mostly of theoretical interest or used in practice as a component of another curve definition.

Recursive form for a Lagrange interpolation

The above Lagrange interpolate can be written in a recursive manner. We introduce a polynomial of degree 1, which corresponds to the case $n = 1$ of the above general definition. This polynomial, defined in the interval $[t_i, t_{i+1}]$, is given by

$$A_i^1(t) = \frac{t_{i+1} - t}{t_{i+1} - t_i} P_i + \frac{t - t_i}{t_{i+1} - t_i} P_{i+1}; \quad i = 0, ..., n-1 \,.$$

Using this relation[2], we construct a recursion. To this end, we assume that we have found a polynomial, say $A_0^{n-1}(t)$, that interpolates to the n first P_is (i.e., $i = 0, ..., n-1$) along with a polynomial $A_1^{n-1}(t)$ that interpolates to the n last P_is (i.e., $i = 1, ..., n$). With this material, we define the following sequence:

$$A_0^n(t) = \frac{t_n - t}{t_n - t_0} A_0^{n-1}(t) + \frac{t - t_0}{t_n - t_0} A_1^{n-1}(t) \, .$$

We then have $A_0^n(t_i) = P_i$.

Proof. First, for $t = t_0$, we have $A_0^n(t_0) = A_0^{n-1}(t_0)$ which leads to having $A_0^n(t_0) = A_0^1(t_0)$. Then, following the definition of the A_1^is with $i = 0$ and $t = t_0$, we have

$$A_0^n(t_0) = \frac{t_1 - t_0}{t_1 - t_0} P_0 = P_0 \, .$$

Similarly, for $t = t_n$, we have $A_0^n(t_n) = A_1^{n-1}(t_n)$, thus $A_0^n(t_n) = A_{n-1}^1(t_n)$, i.e.,

$$\frac{t_n - t_{n-1}}{t_n - t_{n-1}} P_n = P_n \, .$$

For the other t_is, we just have to follow the way in which the recursion was defined. Indeed, we have $A_0^{n-1}(t_i) = A_1^{n-1}(t_i) = P_i$, then $A_0^n(t_i) = P_i$. □

The above recursion can be generalized in:

$$A_i^r(t) = \frac{t_{i+r} - t}{t_{i+r} - t_i} A_i^{r-1}(t) + \frac{t - t_i}{t_{i+r} - t_i} A_{i+1}^{r-1}(t); \; i = 0, ..., n-r \quad r = 1, ..., n$$

the so-called Aitken algorithm.

As a conclusion, the Lagrange interpolate can be written as initially stated or by means of the above Aitken algorithm, namely

$$\gamma(t) = A_0^n(t) \, .$$

Matrix form for a Lagrange interpolation

In practice, given an adequate set of t_is, the Lagrange interpolate can be expressed, for each component, in a simple matrix form:

$$\gamma(t) = [\mathcal{T}(t)][\mathcal{M}][\mathcal{P}]$$

with:

$$[\mathcal{T}(t)] = [t^n \quad t^{n-1} \quad ... \quad t \quad 1]$$

$$[\mathcal{P}] = {}^t[P_0 \quad P_1 \quad .... \quad P_n]$$

where $[\mathcal{M}]$ is a $(n+1) \times (n+1)$ matrix.

[2]Where the notation A is short for Aitkens, as will be justified subsequently.

Lagrange forms of degree 1 and 2

In the case where parameter t is chosen as: $t_i - t_{i-1} = \frac{1}{n}$, i.e., the sequence t_i is uniform, matrix $[\mathcal{M}]$ is written as follows[3]:

$$[\mathcal{M}] = \begin{pmatrix} -1 & 1 \\ 1 & 0 \end{pmatrix}$$

for $n = 1$ (assuming $[t_0, t_1] = [0, 1]$). A similar expression, whatever n is, can be obtained (obviously unlikely to be suitable when n is large). For instance, for $n = 2$, $t_0 = 0$, $t_2 = 1$ and $t_1 = 0.5$, we find:

$$[\mathcal{M}] = \begin{pmatrix} 2 & -4 & 2 \\ -3 & 4 & -1 \\ 1 & 0 & 0 \end{pmatrix}.$$

Proof. Using the general formula for $n = 2$, we have:

$$\gamma(t) = \frac{(t-t_1)(t-t_2)}{(t_0-t_1)(t_0-t_2)}P_0 + \frac{(t-t_0)(t-t_2)}{(t_1-t_0)(t_1-t_2)}P_1 + \frac{(t-t_0)(t-t_1)}{(t_2-t_0)(t_2-t_1)}P_2\,,$$

since we set $t_0 = 0$, $t_1 = 0.5$ and $t_2 = 1$, we have:

$$\gamma(t) = (2t^2 - 3t + 1)\,P_0 + (-4t^2 + 4t)\,P_1 + (2t^2 - t)\,P_2\,,$$

thus, $[\mathcal{M}]$ is the above expression. □

Proof. Using the Aitken formula (under the same assumptions), we have successively:

$$A_0^2(t) = \frac{t_2 - t}{t_2 - t_0}A_0^1(t) + \frac{t - t_0}{t_2 - t_0}A_1^1(t)\,,$$

$$A_0^1(t) = \frac{t_1 - t}{t_1 - t_0}P_0 + \frac{t - t_0}{t_1 - t_0}P_1\,,$$

$$A_1^1(t) = \frac{t_2 - t}{t_2 - t_1}P_1 + \frac{t - t_1}{t_2 - t_1}P_2\,,$$

$$A_0^2(t) = \frac{(t-t_1)(t-t_2)}{(t_0-t_1)(t_0-t_2)}P_0 + \frac{(t-t_0)(t-t_2)}{(t_1-t_0)(t_1-t_2)}P_1 + \frac{(t-t_0)(t-t_1)}{(t_2-t_0)(t_2-t_1)}P_2\,,$$

which is the same expression as above, thus leading to the same matrix. □

Without restriction on the t_is, we have:

$$[\mathcal{M}] = \begin{pmatrix} \frac{1}{(t_0-t_1)(t_0-t_2)} & \frac{1}{(t_1-t_0)(t_1-t_2)} & \frac{1}{(t_2-t_0)(t_2-t_1)} \\ \frac{-t_1-t_2}{(t_0-t_1)(t_0-t_2)} & \frac{-t_0-t_2}{(t_1-t_0)(t_1-t_2)} & \frac{-t_0-t_1}{(t_2-t_0)(t_2-t_1)} \\ \frac{t_1 t_2}{(t_0-t_1)(t_0-t_2)} & \frac{t_0 t_2}{(t_1-t_0)(t_1-t_2)} & \frac{t_0 t_1}{(t_2-t_0)(t_2-t_1)} \end{pmatrix}$$

[3]Our interest in this simple case will be made precise hereafter.

Hermite interpolation scheme

Another class of interpolation methods uses *Hermite* type interpolants. In addition to the above $n+1$ pair of control points and parameter values, we assume that the $n+1$ derivatives $\dot{P}_0, \dot{P}_1, ..., \dot{P}_n$ are provided. Then, the function:

$$\gamma(t) = \sum_{i=0}^{n} \tilde{\phi}_i(t) P_i + \sum_{i=0}^{n} \varphi_i(t) \dot{P}_i \tag{12.3}$$

where

$$\tilde{\phi}_i(t) = \{1 - 2\,\phi_i'(t_i)(t-t_i)\}\,\phi_i(t)^2 \text{ and } \varphi_i(t) = (t-t_i)\phi_i(t)^2$$

defines a curve which is governed by the $n+1$ given points and their tangents. This curve, called the *Hermite interpolate*, is such that:

$$\gamma(t_i) = P_i \quad \textit{and} \quad \gamma'(t_i) = \dot{P}_i$$

as can be shown, moreover it is C^∞. Nevertheless, we return to the previous remarks both in terms of how to define the parameter values and in terms of complexity and oscillations (if n is large).

The cubic Hermite form

We reduce to the case where $n = 1$ and we assume that P_0 and P_1 along with $\dot{P}_0$ and $\dot{P}_1$ are supplied. Then, since, after a variable change, $t_0 = 0$ and $t_1 = 1$ are assumed, we have:

$$\begin{aligned} \phi_0(t) &= (1-t) && \text{and} && \phi_0'(t) = -1 \\ \phi_1(t) &= t && \text{and} && \phi_1'(t) = 1 \\ \tilde{\phi}_0(t) &= (1+2t)(1-t)^2 && \text{and} && \tilde{\phi}_1(t) = (1-2(t-1))\,t^2 \\ \varphi_0(t) &= t(1-t)^2 && \text{and} && \varphi_1(t) = (t-1)t^2 \end{aligned}$$

thus:

$$\gamma(t) = (1+2t)(1-2t+t^2)\,P_0 + \left(t^2 - 2t^3 + 2t^2\right) P_1 + \left(t - 2t^2 + t^3\right) \dot{P}_0 + \left(t^3 - t^2\right) \dot{P}_1\,,$$

which can be expressed in a matrix form as:

$$\gamma(t) = [\mathcal{T}(t)][\mathcal{M}][\mathcal{P}] \tag{12.4}$$

where now:

$$[\mathcal{T}(t)] = [t^3 \quad t^2 \quad t \quad 1]$$

$$[\mathcal{P}] =^t [P_0 \quad P_1 \quad \dot{P}_0 \quad \dot{P}_1]$$

$$[\mathcal{M}] = \begin{pmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

which is known as the cubic Hermite interpolation or the cubic Coons basis.

Remark 12.7 *Actually, the cubic Hermite form concerns a third degree polynomial considered with a set of 4 data values (2 points and 2 derivatives). Thus, this simple example will be of interest when composite curves are seen (see below).*

Remark 12.8 *Higher order Hermite type interpolations can be constructed if higher order derivatives are supplied as input data.*

12.3 Explicit construction of a composite curve

In this section, we give a method suitable for constructing a composite curve using one of the previous approaches. Indeed, after the remark about the cost and the possible existence of oscillations when the number of points n is large, we want to define the curve Γ by a series of sub-curves with a low degree such that their junctions are under control. Moreover, this construction uses as input data a discrete approximation of the curve under investigation.

This approximation is indeed a *polyline* composed of a series of segments $[P_i, P_{i+1}]$. The construction is completed segment by segment using the available information (tangents, corners, etc.). Depending on the amount of information provided as input data, a polynomial based definition of degree 1, 2 or 3 can be easily obtained in any member of the polyline. In what follows, we consider one segment, say $[P_i, P_{i+1}]$, and we denote this segment by $[A, B]$.

Giving 4 input data, A, B and the tangents at A and at B, $\vec{\tau_A}$ and $\vec{\tau_B}$, we can obtain a function like:

$$\gamma(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 \,, \tag{12.5}$$

where $a_i \in \mathbb{R}^d \quad (i = 0, 3)$ and t is a parameter ranging from 0 to 1. For the moment, we know nothing more about the meaning of t.

First, we consider the case where no tangent is provided. Using the only two data we have, we look for a function γ of the above type with $a_2 = a_3 = 0$, i.e., a function of degree one. The constraints we want to conform to are $\gamma(0) = A$ along with $\gamma(1) = B$. Then, we must solve the system:

$$\left\{ \begin{array}{lcl} a_0 & = & A \\ a_1 & = & B - A \,, \end{array} \right. \tag{12.6}$$

meaning that $\gamma(t) = A + t(B - A)$, thus the Lagrange form of degree 1; see Section 12.2.

Remark 12.9 *Since the derivative is $\gamma'(t) = \vec{AB}$, we encounter two cases. First, if $||\vec{AB}|| = 1$, t is nothing more than s, the curvilinear abscissa (see Chapter 11), while, if $||\vec{AB}|| \neq 1$, t is an arbitrary parameter. In the previous case, a curvilinear abscissa can be defined by $s = t\, ||\vec{AB}||$.*

When either of the tangents is not provided, we look for a function of degree two, i.e., we fix $a_3 = 0$. Then, if $\vec{\tau_A}$ is known, we have:

$$\left\{ \begin{array}{lcl} a_0 & = & A \\ a_1 & = & \vec{\tau_A} \\ a_2 & = & (B - A) - \vec{\tau_A} \,, \end{array} \right. \tag{12.7}$$

$\vec{\tau_A}$	$\vec{\tau_A}$	.	.	A_{-1}	$\vec{\tau_A}$	A_{-1}	A_{-1}	.
A	A	A	A	A	A	A	A	A
B	B	B	B	B	B	B	B	B
$\vec{\tau_B}$	.	$\vec{\tau_B}$	.	$\vec{\tau_B}$	B_{+1}	B_{+1}	.	B_{+1}
3	2	2	1	3	3	3	2	2

Table 12.1: Degree of the approximation (bottom line) according to the data type (in columns).

while if only $\vec{\tau_B}$ is known, we obtain:

$$\left\{\begin{array}{lcc} a_0 & = & A \\ a_1 & = & 2(B-A)-\vec{\tau_B} \\ a_2 & = & -(B-A)+\vec{\tau_B}\,. \end{array}\right. \tag{12.8}$$

Then, we assume that we know the tangents at A and at B, $\vec{\tau_A}$ and $\vec{\tau_B}$, this information then allows us to define a curve of degree 3. The four given information elements lead to the system:

$$\left\{\begin{array}{ccc} a_0 & = & A \\ a_0+a_1+a_2+a_3 & = & B \\ a_1 & = & \vec{\tau_A} \\ a_1+2a_2+3a_3 & = & \vec{\tau_B}\,, \end{array}\right. \tag{12.9}$$

whose solution is:

$$\left\{\begin{array}{lcc} a_0 & = & A \\ a_1 & = & \vec{\tau_A} \\ a_2 & = & 3(B-A)-2\vec{\tau_A}-\vec{\tau_B} \\ a_3 & = & -2(B-A)+\vec{\tau_A}+\vec{\tau_B}\,. \end{array}\right. \tag{12.10}$$

Then, $\gamma(t)$ is nothing other than the cubic Hermite form given in Section 12.2.

Remark 12.10 *It is advisable to make sure that $||\vec{\tau}||$ is something like $||AB||$. Indeed $||\vec{\tau}||$ acts through two aspects. First we naturally encounter a directional aspect and then the module of the tangent controls the locality of this directional control.*

Other categories of situations are encountered when A and/or B are not corner(s) but when we know a point "before" A, denoted as A_{-1}, and/or a point "after" B, denoted as B_{+1}, then we can define a function of degree three or two by returning to the previous cases. The tangents at AB in A and/or B are evaluated by using the points A_{-1} and/or B_{+1}. For instance, $\vec{\tau_A} = A - A_{-1}$ or a similar expression. The final type that can be observed is when the data combine the different possibilities.

Remark 12.11 *Apart from the first case where t could be s or s could be easily defined resulting in a normal parameterization of AB, such a parameterization is not obtained in the other situations.*

Table 12.1 shows, according to the data categories, the degree that can be expected for the approximation of AB.

A few remarks. In general, the function $\gamma(t)$ constructed from two points and the two related tangents may offer different aspects depending on the type of this information. Some aspects may obviously be undesirable for the type of applications envisaged.

Thus, the presence of a loop in one segment, i.e., when there are two different values t_1 and t_2 for which $\gamma(t_1) = \gamma(t_2)$ necessarily corresponds to ill-suited data. In fact, a nice property to guarantee what we need, is to have a function g of parameter t such that

- $g(t)$ is a strictly increasing function,

where $g(t) = d(A, proj_{AB}(\gamma(t)))$ in segment AB. In this expression, d denotes the usual distance while $proj_{AB}(P)$ is the projection of point P onto AB (cf. Figure 12.1).

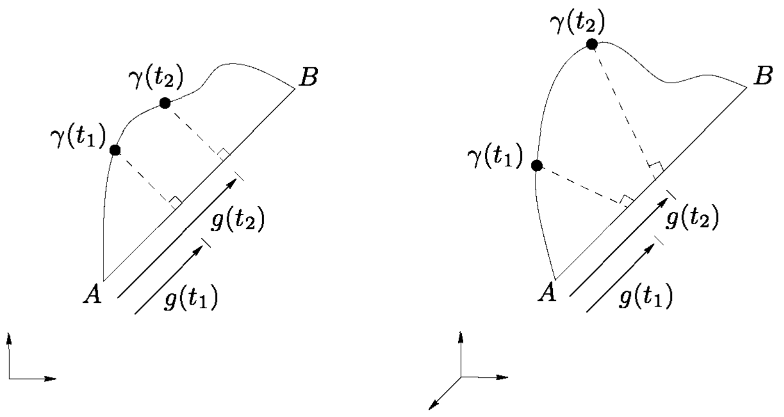


Figure 12.1: *$\gamma(t)$ and $g(t)$ for a curve in two dimensions (left-hand side) and for a curve in three dimensions (right-hand side).*

The assumed property implies in particular that the curve has no loop (between A and B). From a practical point of view, we can also assume that the distance between AB and the curve is bounded by a reasonably small threshold value. In other words, the segment AB is close to the curve, meaning that the segments provided as input already correspond to a reasonable approximation of the geometry of the real curve.

12.4 Control polygon based methods

We now turn to a class of methods not based on an interpolation technique. Here, we are interested in methods that make use of a control polygon whose vertices act through some contributions defined via various functions.

Control polygon and curve definition

The set of control points enables us to define a *control polygon.* As a general statement, it is possible to define a curve in terms of the following equation:

$$\gamma(t) = \sum_{i=0}^{n} \varphi_i(t) P_i \tag{12.11}$$

Once the basis functions $\varphi_i(t)$s have been chosen, the curve is uniquely defined by the points P_is in the control polygon. At a glance, any basis of functions makes this curve definition possible. However, some properties are usually required so as to ensure that the curve "looks like" its control polygon. These assumptions are indeed assumed to facilitate curve manipulation when used in a CAD system, particularly, if the user has only limited knowledge of the underlying geometric models.

General properties

The following presents some of the properties commonly assumed in most of the models based on a control polygon.

Cauchy identity. This relationship is related to a normalization purpose. Then, we have:

$$\sum_{i=0}^{n} \varphi_i(t) \equiv 1 \quad \forall t\,.$$

This implies that Equation (12.11) is equivalent to:

$$\gamma(t) = \frac{\sum\limits_{i=0}^{n} \varphi_i(t) P_i}{\sum\limits_{i=0}^{n} \varphi_i(t)}\,. \tag{12.12}$$

This expression of the curve clearly shows the barycentric form of the latter. The current point $\gamma(t)$ is the centroid of the set of points P_i associated with the weights $\varphi_i(t)$.

Remark 12.12 (Curve translation) *We look at the construction of a new polygon resulting from the translation of the control polygon P_i by a vector v. The coordinates of the points of this new control polygon are denoted by $P_i + v$. The new curve γ' associated with this new polygon can be written as:*

$$\gamma'(t) = \sum_{i=0}^{n} \varphi_i(t)(P_i + v) = \sum_{i=0}^{n} \varphi_i(t) P_i + \underbrace{\left(\sum_{i=0}^{n} \varphi_i(t)\right)}_{\equiv 1} v = \gamma(t) + v$$

If the Cauchy identity were not assumed, then the form of the curve would be a function of the position of the control polygon. Conversely, this identity implies that the curve is only related to the shape of its control polygon.

Remark 12.13 (Linear transformation of a curve) *We now turn to a method that makes it possible to construct a new curve resulting from a linear transformation of the curve $\gamma(t)$. Then, this new curve, $\gamma'(t)$, conforms to the general expression:*

$$\gamma'(t) = \begin{pmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{pmatrix} \gamma(t)\,.$$

Thus, we have:

$$\gamma'(t) = \begin{pmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{pmatrix} \left[\sum_{i=0}^{n} \varphi_i(t) \begin{pmatrix} P_{i_x} \\ P_{i_y} \\ P_{i_z} \end{pmatrix} \right]$$

$$= \sum_{i=0}^{n} \varphi_i(t) \left[\begin{pmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} P_{i_x} \\ P_{i_y} \\ P_{i_z} \end{pmatrix} \right],$$

where the second equality is true only if the Cauchy condition is assumed. Thus, any linear transformation of a curve (for instance, a rotation, a scaling, etc.) is easy to perform directly in the control polygon.

Positive functions. Inside the definition interval of γ, the functions $\varphi_i(t)$ are assumed to be larger than or equal to zero:

$$\varphi_i(t) \geq 0\,.$$

Provided with the Cauchy identity, the current point $\gamma(t)$ is the centroid of the set of the P_is associated with *positive weights*, the $\varphi_i(t)$s. Hence, all any convex region bounding the set of points P_i encloses the curve.

Extremity conditions. If $[t_{min}, t_{max}]$ stands for the definition interval of γ, the functions φ_i satisfy the following:

$$\varphi_0(t_{min}) = 1 \qquad \text{and} \qquad \varphi_n(t_{max}) = 1\,.$$

Corollary 12.1 *The Cauchy condition and the previous relationships with positive functions imply that:*

$$\underbrace{\varphi_i(t_{min})}_{i \neq 0} \equiv 0 \quad \textit{and} \quad \underbrace{\varphi_i(t_{max})}_{i \neq n} \equiv 0\,.$$

Hence, at $t = t_{min}$, the sole point with an action on the curve is point P_0. Moreover, we have the relationships $\gamma(t_{min}) = P_0$ and $\gamma(t_{max}) = P_n$. As a consequence, the curve endpoints are the endpoints of the corresponding control polygon.

Derivatives at curve endpoints. In addition, the following conditions can be assumed. At t_{min}, we impose:

$$\frac{d\varphi_0}{dt}(t_{min}) = -\frac{d\varphi_1}{dt}(t_{min}) \quad \text{and} \quad \underbrace{\frac{d\varphi_i}{dt}(t_{min}) \equiv 0}_{i>1}$$

and at parameter t_{max}:

$$\frac{d\varphi_m}{dt}(t_{max}) = -\frac{d\varphi_{m-1}}{dt}(t_{max}) \quad \text{and} \quad \underbrace{\frac{d\varphi_i}{dt}(t_{max}) \equiv 0}_{i<m-1}.$$

Hence, for $t = t_{min}$, we obtain the following equation:

$$\frac{d\gamma}{dt}(t_{min}) = \sum_{i=0}^{m} \frac{d\varphi_i}{dt}(t_{min})P_i = \frac{d\varphi_0}{dt}(t_{min})(P_1 - P_0).$$

Thus, the tangent at t_{min} is directed by the first segment in the control polygon. Similarly, the tangent at t_{max} has the same direction as the last segment in this polygon.

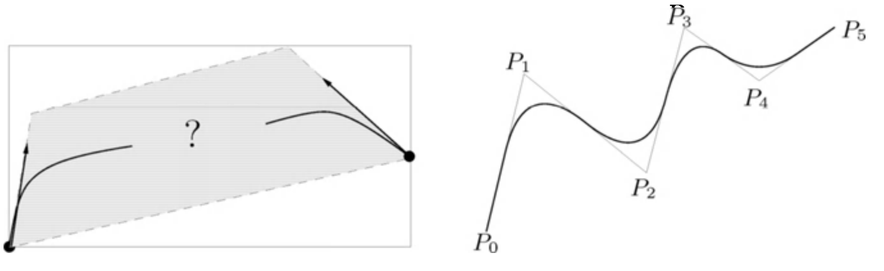


Figure 12.2: *Basis properties related to the definition of a control polygon. The shape of the curve is close to this governing polyline (broken line).*

Figure 12.2 depicts the various properties of the curves governed by a control polygon. At first, $\gamma(t)$ passes through P_0 and P_n while the other points are only control points. Moreover, the tangents at the curve endpoints follow the first and the last segments in the polygon. In addition, the curve lies inside any convex region including its control polygon.

In this figure, the polygon in dashed line is the smallest of the convex polygons containing the control polygon. CAD systems usually use the enclosing box shown in the figure by the rectangle to bound the region enclosing the curve. Apart from the characteristics at the curve endpoints, the only thing we can say *a priori* about the shape of the curve is that it "looks like" its control polygon.

12.5 Bézier curves

A popular method based on a control polygon makes use of *Bézier curves*.

Form of a Bézier curve

Provided with the $n+1$ control points, we define a Bézier curve using the following algebraic relation

$$\gamma(t) = \sum_{i=0}^{n} C_n^i t^i (1-t)^{n-i} P_i \,. \tag{12.13}$$

with $C_n^i = \frac{n!}{(n-i)!i!}$ for $0 \leq i \leq n$ and $C_n^i = 0$ otherwise[4]. In other words, a Bézier curve relies on Bernstein polynomials . Indeed, these polynomials are defined by

$$B_{i,n}(t) = C_n^i t^i (1-t)^{n-i}$$

where t ranges from 0 to 1 and $i \in [0, n]$ while $B_{i,n}(t) = 0$ elsewhere. Thus:

$$\gamma(t) = \sum_{i=0}^{n} B_{i,n}(t)\, P_i \,.$$

Exercise 12.3 *Show that the above properties hold.*

Remark 12.14 *The parameter value t ranges from 0 to 1. It is possible to use a different set of parameter values where t varies from a given t_0 to a given $t_n > t_0$. To return to an interval between 0 to 1, we simply have to introduce a variable change such as $\tilde{t} = \frac{t-t_0}{t_n-t_0}$. The way in which the t_is vary allow for some flexibility in the curve definition.*

This curve definition is C^∞. Nevertheless, for a large n, the above remarks remain true. In addition, while more flexible than the previous approaches, some classical curves, such as conics are poorly represented by this Bézier form. To deal with such a problem, two solutions can be envisaged. On the one hand, a different global curve definition can be used (see for instance, the rational Bézier) or, on the other hand, a low degree Bézier can be employed leading to using a composite definition for the curve (see below).

About Bernstein polynomials

First, it can be proved that Bernstein polynomials form a basis of all polynomials of degree n. In this respect, Bernstein polynomials can be written as:

$$B_{i,n}(t) = \sum_{j=i}^{n} (-1)^{j-i} C_n^j C_j^i \, t^j \,.$$

On the other hand, Bézier curves offer some facility for various computations. Indeed, since these curves are defined by means of Bernstein polynomials, various

[4]Note that coefficient C_n^i has various notations. For instance, the notation ${}^nC_i$ as well as $\begin{pmatrix} n \\ i \end{pmatrix}$ can be found.

recursions about these polynomials allow us to elegantly manipulate these curves. First, the C_n^is conform to the following recursion:

$$C_n^i = C_{n-1}^{i-1} + C_{n-1}^i \,.$$

In other words, the C_n^is can be obtained by the Pascal triangle rule as illustrated in Table 12.2. The Bernstein polynomials can then be obtained by recursion. Actually, we have:

$$B_{i,n}(t) = t\,B_{i-1,n-1}(t) \;+\; (1-t)\,B_{i,n-1}(t)\,. \tag{12.14}$$

Proof. Merging the recursion about the C_n^is in $B_{i,n}(t) = C_n^i t^i (1-t)^{n-i}$, we obtain successively:

$$B_{i,n}(t) = C_{n-1}^{i-1} t^i (1-t)^{n-i} + C_{n-1}^i t^i (1-t)^{n-i}\,,$$

$$B_{i,n}(t) = C_{n-1}^{i-1} t\, t^{i-1} (1-t)^{n-1-(i-1)} + C_{n-1}^i t^i\,(1-t)\,(1-t)^{n-1-i}\,,$$

$$B_{i,n}(t) = t\,C_{n-1}^{i-1} t^{i-1} (1-t)^{n-1-(i-1)} + (1-t)\,C_{n-1}^i t^i\,(1-t)^{n-1-i}\,,$$

and the above recursion holds. □

-	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	....
$n=0$	1						
$n=1$	1	1					
$n=2$	1	2	1				
$n=3$	1	3	3	1			
$n=4$	1	4	6	4	1		
$n=5$	1	5	10	10	5	1	
...							

Table 12.2: The Pascal triangle rule.

Obviously we have:

$$B_{i,n}(t) = B_{n-i,n}(1-t)\,.$$

Moreover,

$$B_{0,n}(t) = (1-t)\,B_{0,n-1}(t) \text{ thus } B_{0,n}(t) = (1-t)^n\,,$$

$$B_{n,n}(t) = t\,B_{n-1,n-1}(t) \text{ and } B_{n,n}(t) = t^n\,.$$

A recursion about the derivatives can be easily found. It is as follows:

$$B'_{i,n}(t) = n\,(B_{i-1,n-1}(t) \;-\; B_{i,n-1}(t))\,.$$

Various recursions related to high order derivatives, integrals and many other relationships can be exhibited. To conclude this discussion about Bernstein polynomials, it should be noted that all of the above recursions are also useful in the case of rational Bézier, composite Bézier, etc., curve representations.

De Casteljau form for a Bézier curve

Defining $D_i^0(t) = P_i$ for $i = 0, ..., n$, the recursion[5]:

$$D_i^r(t) = (1-t)\, D_i^{r-1}(t) + t\, D_{i+1}^{r-1}(t)$$

for $r = 1, n$ and $i = 0, n-r$ (and $D_i^r(t) = 0$ otherwise) is the so-called De Casteljau algorithm. Using this recursion, the curve defined by:

$$\gamma(t) = D_0^n(t)\,,$$

is a practical way to construct a Bézier curve which does not directly involve the use of Bernstein polynomials.

Bézier curve of degree 3

Any Bézier curve can be written in the matrix form already introduced. For example, for $n = 3$, the general expression or the De Casteljau algorithm results in the matrix $[\mathcal{M}]$ given by:

$$[\mathcal{M}] = \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

with

$$[\mathcal{T}(t)] = [t^3 \quad t^2 \quad t \quad 1]$$

$$[\mathcal{P}] =^t [P_0 \quad P_1 \quad P_2 \quad P_3]$$

Proof. Using Bernstein polynomials, we have successively:

$$B_{0,3}(t) = (1-t)^3\,,$$

$$\begin{aligned}
B_{1,3}(t) &= t\, B_{0,2}(t) + (1-t)\, B_{1,2}(t)\,, \\
B_{1,2}(t) &= t\, B_{0,1}(t) + (1-t)\, B_{1,1}(t)\,, \\
B_{1,2}(t) &= t\,(1-t) + (1-t)\,t = 2t\,(1-t)\,, \\
B_{1,3}(t) &= t\,(1-t)^2 + 2t\,(1-t)^2 = 3t\,(1-t)^2\,, \\
B_{2,3}(t) &= t\, B_{1,2}(t) + (1-t)\, B_{2,2}(t)\,, \\
B_{2,3}(t) &= 2t^2\,(1-t) + (1-t)\,t^2 = 3t^2\,(1-t)\,, \\
B_{3,3}(t) &= t^3\,,
\end{aligned}$$

then:

$$\gamma(t) = B_{0,3}(t)\,P_0 + B_{1,3}(t)\,P_1 + B_{2,3}(t)\,P_2 + B_{3,3}(t)\,P_3\,,$$

$$\gamma(t) = (1-t)^3\,P_0 + 3t\,(1-t)^2\,P_1 + 3t^2\,(1-t)\,P_2 + t^3\,P_3\,,$$

and we retrieve the above matrix. □

[5]The notation D stands for De Casteljau.

Proof. Using De Casteljau algorithm, we have successively:

$$D_0^1(t) = (1-t)\,D_0^0(t) \,+\, t\,D_1^0(t)\,,$$
$$D_0^1(t) = (1-t)\,P_0 \,+\, t\,P_1\,,$$
$$D_1^1(t) = (1-t)\,D_1^0(t) \,+\, t\,D_2^0(t)\,,$$
$$D_1^1(t) = (1-t)\,P_1\,t\,P_2\,,$$
$$D_0^2(t) = (1-t)\,D_0^1(t) \,+\, t\,D_1^1(t)\,,$$
$$D_0^2(t) = (1-t)^2\,P_0 \,+\, 2t\,(1-t)\,P_1 \,+\, t^2\,P_2\,,$$
$$D_1^2(t) = (1-t)\,D_1^1(t) \,+\, t\,D_2^1(t)\,,$$
$$D_1^2(t) = (1-t)^2\,P_1 \,+\, 2t\,(1-t)\,P_2 \,+\, t^2\,P_3\,,$$

then:

$$D_0^3(t) = (1-t)^3\,P_0 + 3t\,(1-t)^2\,P_1 + 3t^2\,(1-t)\,P_2 + t^3\,P_3\,,$$

and we still retrieve the above matrix. □

Degree elevation of a Bézier curve

Degree elevation of a Bézier curve is a very useful process for various purposes (including the problem of finding some degree of continuity when composite curves (surfaces) are considered).

Given a Bézier curve of degree n corresponding to $n+1$ control points P_i, degree elevation leads to defining the same curve as a curve of degree $n+1$ based on $n+2$ control points, the Q_i. The issue is to find these control points.

Let $\gamma(t) = \sum_{i=0}^{n} B_{i,n}(t)\,P_i$ be the given curve, which we want to be written as $\gamma(t) = \sum_{i=0}^{n+1} B_{i,n+1}(t)\,Q_i$. The solution is as follows:

$$Q_0 = P_0 \quad \text{and} \quad Q_{n+1} = P_n \quad \text{with}$$
$$Q_i = \frac{i\,P_{i-1} \,+\, (n+1-i)\,P_i}{n+1}\,, \quad i = 1, ..., n\,.$$

Proof. Since $B_{i,n}(t) = (1-t)\,B_{i,n}(t) + tB_{i,n}(t)$ holds where the $B_{i,n}(t)$s are defined by $B_{i,n}(t) = \frac{n!}{i!(n-i)!}t^i(1-t)^{n-i}$, we have:

$$B_{i,n}(t) = \frac{n+1-i}{n+1}B_{i,n+1}(t) + \frac{i+1}{n+1}B_{i+1,n+1}(t)\,,$$

then,

$$\gamma(t) = \sum_{i=0}^{n} B_{i,n}(t)\,P_i = \sum_{i=0}^{n}\left(\frac{n+1-i}{n+1}B_{i,n+1}(t)\,P_i + \frac{i+1}{n+1}B_{i+1,n+1}(t)\,P_i\right)\,,$$

and, after rearranging the terms, we obtain:

$$\gamma(t) = B_{0,n+1}\,P_0 \,+\, \sum_{1=0}^{n} B_{i,n+1}\left(\frac{i}{n+1}P_{i-1} + \frac{n+1-i}{n+1}P_i\right) \,+\, B_{n+1,n+1}\,P_n\,,$$

thus leading to the definition of the Q_is. □

12.6 From composite curves to B-splines

The main idea is to use locally one of the above curve definitions where a low degree is assumed. In this way, a curve is defined by a series of sub-curves. The issue is then to insure the desired smoothness between the different parts of the entire curve. This technique actually leads to constructing a *composite* curve.

Composite Bézier curves

Composite Bézier curves correspond to Bézier curves. We define a *knot sequence* of t_js and a series of segments where Bézier polynomials of degree m are employed. Here, segment j, for $j \geq 1$, is denoted by seg_j. It is simply the interval:

$$seg_j = [t_j, t_{j+1}] \; .$$

With this background a composite Bézier curve is:

$$\gamma(t) = \sum_{j=1}^{n_m} \sum_{i=0}^{m} B^j_{i,m}(t)\, P_{i,j} \, , \tag{12.15}$$

where (to return to the case where t ranges from 0 to 1 for each segment) the $B^j_{i,m}(t)$s are written by means of Bernstein polynomials as follows:

$$B^j_{i,m}(t) = B_{i,m}\left(\frac{t - t_j}{t_{j+1} - t_j}\right) \quad \text{if } t \in seg_j$$

$$B^j_{i,m}(t) = 0 \quad \text{otherwise}$$

and

$$P_{i,j} = P_{m\,(j-1)+i}$$

meaning that a segment runs from

$$P_{0,j} = P_{m\,(j-1)} \quad \text{to} \quad P_{m,j} = P_{mj}$$

and, finally n_m depends both on n and m. In fact, it is necessary to have $n = (m+1) \times n_m$.

The curve only passes through the $P_{m\,(j-1)}$s, the other points really being control points.

The definition of the t_js can be made in several ways thus leading to some extent of flexibility. First, t_0 is naturally associated with P_0, then t_1 must be associated with P_m and so on. In fact t_j is related to $P_{m\,(j-1)}$. On the other hand, the values of the t_js can be arbitrarily defined.

First, as already indicated, the entire curve is defined by a series of segments, in other words by means of local Bézier definitions and locally, the regularity in C^∞. Second, the global regularity is insured by an adequate definition of the junction from segment to segment. In this respect a G^1 or even a C^1 continuity can be obtained, say for $m = 3$, by acting on the control points previous and next to a segment endpoint. Actually, the three above points must be colinear (thus leading to a G^1 smoothness) and, in addition, must be in a certain ratio (to reach a C^1 property). Similarly a G^2 or a C^2 continuity, for $m = 4$, can be completed by acting on the two previous and the two next control points of a given segment endpoint. These requests allow for the desired regularity when the entire curve is considered but, on the other hand, imply some constraints that can impede the curve definition. A greater flexibility is then obtained by introducing the curve representation discussed in the next section.

Remark 12.15 *The above composite curve definition involves locally a single curve function. Thus, global continuity is obtained at some price. To overcome this problem local functions acting as basis functions may themselves be composite (see below).*

B-spline curves

B-spline curves correspond to De Boor spline curves. Before going further in B-splines, we introduce the notion of a spline.

Given a sequence of $m+2$ knots $t_0 \leq t_1 \leq ... \leq t_{m+1}$, a general basis polynomial spline f of degree m is a function that satisfies the following properties:

- f is a polynomial of degree m on all intervals $[t_i, t_{i+1}]$, this restriction being denoted by f_i,
- f is such that $f_i(t_{i+1}) = f_{i+1}(t_{i+1})$,
- f is C^{m-1} at the junction points (i.e., at the t_is) when these knots are not multiple. At a node of multiplicity r, the continuity is C^{m-r}.

Remark 12.16 *Note that the above composite Bézier curve definition is included in this category of definitions while, in this case, the corresponding f_is reduce to a single (non-composite) function and the third condition is not automatically insured.*

Given a knot sequence of $m+2$ t_js like $t_i, t_{i+1}, ..., t_{i+m+1}$, the basis spline $N_{i,m}$ related to these knots is constructed[6] using a recursion, in terms of index k. For $k = 0$, it leads to:

$$\begin{cases} N_{i,0}(t) &= 1 \quad \text{if} \quad t_{i-1} \leq t < t_i \quad \text{and} \\ N_{i,0}(t) &= 0 \quad \text{else} \ , \end{cases}$$

and, for $k = 1, 2, ..., m$, we have:

$$N_{i,k}(t) = \frac{t - t_{i-1}}{t_{i-1+k} - t_{i-1}} N_{i,k-1}(t) + \frac{t_{i+k} - t}{t_{i+k} - t_i} N_{i+1,k-1}(t) \, ,$$

meaning that the composite function $N_{i,k}(t)$ is non-zero only if $t \in [t_{i-1}, t_{i+k}]$.

[6]In the following, we discuss only one of the various possible ways to define a B-spline.

The basis polynomials have a local and minimal support. Moreover, they are linearly independent. Indeed, they form a basis.

In the above recursion, if, for instance, $t_{i-1+k} - t_{i-1} = 0$, meaning that multiple knots are used, then the term in $N_{i,k-1}(t)$ does not contribute since $N_{i,k-1}(t) = 0$.

Remark 12.17 *Provided with an adequate choice of the t_is, the above relation reduces to Relation (12.14) meaning that, in this particular case, the $N_{i,k}(t)$s are nothing other than Bernstein polynomials. Indeed, we have to set $t_i = 0$ for $i = 0, 1, ..., k$ and $t_{k+1} = ... = t_{2k+1} = 1$.*

With this background (leaving aside the previous remark), given $n+1$ control points ($n \geq m$), a composite curve of $n - m + 1$ polynomial curves of degree m is defined as

$$\gamma(t) = \sum_{j=i+1-m}^{i+1} N_{j,m}(t)\, P_j\,. \tag{12.16}$$

Remark 12.18 *Since the initial condition of the definition by recursion is satisfied for $N_{j,m}(t) \equiv 0$ when $j \notin [i+1-m, i+1]$, we also have:*

$$\gamma(t) = \sum_{j=0}^{m} N_{j,m}(t)\, P_j\,. \tag{12.17}$$

where we can retrieve the general equation of a model based on a control polygon.

Exercise 12.4 *Study the properties of Section 12.4 in the case of the basis functions $N_{j,m}(t)$.*

This curve, the so-called B-spline, uses $n+1$ basis splines and thus needs a sequence of $n+m+2$ knots, namely $t_0, t_1, ..., t_{n+m+1}$. The first curved segment that fully uses the first $m+1$ control points, is the combination $N_{0,m}\, P_0 + N_{1,m}\, P_1 + ... + N_{m,m}\, P_m$, which is defined in $[t_{m-1}, t_m]$. Similarly, segment number k is the combination $N_{k,m}\, P_k + N_{k+1,m}\, P_{k+1} + ... + N_{k+m,m}\, P_{k+m}$, which is defined in $[t_{k+m-1}, t_{k+m}]$. Then the last segment is defined in $[t_{n-1}, t_n]$ corresponding to the combination $N_{n-m,m}\, P_{n-m} + N_{n-m+1,m}\, P_{n-m+1} + ... + N_{n,m}\, P_n$.

Following on from what has just been said, the curve does not pass through the control points if the knots are all distinct. Using multiple knots enables us to pass through some control points. Boundary conditions can be also achieved by using multiple or phantom (fictitious) control points.

In the following three examples, we assume that the t_is are all distinct.

Degree 1 B-spline

In this case $m = 1$ is assumed, then using the general recursion with $k = m = 1$, we have:

$$N_{i,1}(t) = \frac{t - t_{i-1}}{t_i - t_{i-1}} N_{i,0}(t) + \frac{t_{i+1} - t}{t_{i+1} - t_i} N_{i+1,0}(t)\,. \tag{12.18}$$

This B-spline consists of two components. In other words, it lives in two intervals. This composite function enables us to define the curve $\gamma(t)$ as:

$$\gamma(t) = \sum_{j=i}^{i+1} N_{j,1}(t)\, P_j \,. \tag{12.19}$$

The first interval that is well defined corresponds to $j = 0$ and then to $i = 0$. Hence, in $[t_0, t_1]$, both $N_{0,1}(t)$ and $N_{1,1}(t)$ have a contribution. Since only $N_{1,0}(t) \neq 0$, we simply have (for $i = 0$):

$$N_{0,1}(t) = \frac{t_1 - t}{t_1 - t_0}\,,$$

while (for $i = 1$):

$$N_{1,1}(t) = \frac{t - t_0}{t_1 - t_0}\,.$$

Then, following Relation (12.19), in $[t_0, t_1]$, we have:

$$\gamma(t) = \frac{t_1 - t}{t_1 - t_0}\, P_0 + \frac{t - t_0}{t_1 - t_0}\, P_1 \,.$$

Using a uniform distribution of t_is (for instance, $t_i = i$), a simple variable change allows us to find a definition in $[0, 1]$. Indeed, we return to the classical linear interpolation function $\gamma(t) = (1 - t)\, P_0 + t\, P_1$.

Similarly, to define the curve in $[t_1, t_2]$, we need to know both $N_{1,1}(t)$ and $N_{2,1}(t)$ which act through their components related to $N_{2,0}(t)$. Thus, we consider the general recursion and we fix $i = 1$ to obtain the contribution of $N_{1,1}(t)$:

$$N_{1,1}(t) = \frac{t_2 - t}{t_2 - t_1}\,,$$

and we fix $i = 2$ to find the contribution of $N_{2,1}(t)$:

$$N_{2,1}(t) = \frac{t - t_1}{t_2 - t_1}\,,$$

hence, in $[t_1, t_2]$, we have:

$$\gamma(t) = \frac{t_2 - t}{t_2 - t_1}\, P_1 + \frac{t - t_1}{t_2 - t_1}\, P_2 \,.$$

This definition (with the above assumptions) leads to having $\gamma(t) = (1-t)\, P_1 + t\, P_2$.

Remark 12.19 *In practice, for a uniform node distribution, the B-spline can be defined everywhere using only the factors* $(1 - t)$ *and* t. *To this end, a variable change is done to reduce the interval of interest to* $[0, 1]$.

Remark 12.20 *The B-spline is symmetric (see the case where* t *is replaced by* $1 - t$*).*

As expected, this curve is $C^{m-1} = C^0$. Indeed, in the first interval we have $\gamma(1) = P_1$ as well as in the second interval where $\gamma(0) = P_1$. Note also that the definition is reversible. The curve defined using P_0, P_1, P_2 and the curve defined by P_2, P_1, P_0 are identical. More generally, the sequences $P_0, P_1, P_2, ..., P_n$ and $P_n, P_{n-1}, P_{n-2}, ..., P_0$ lead to the same curve.

Degree 2 B-spline

Using the recursion about the $N_{i,k}$ in the case where $k = m = 2$, we have:

$$N_{i,2}(t) = \frac{t - t_{i-1}}{t_{i+1} - t_{i-1}} N_{i,1}(t) + \frac{t_{i+2} - t}{t_{i+2} - t_i} N_{i+1,1}(t) . \tag{12.20}$$

This B-spline, comprising three components, lives in three sub-intervals. The curve defined by:

$$\gamma(t) = \sum_{j=i-1}^{i+1} N_{j,2}(t)\, P_j \,, \tag{12.21}$$

is well defined since the interval that corresponds to $i = 1$. Hence, in $[t_1, t_2]$. In this interval, three B-splines have a contribution, namely $N_{0,2}(t)$, $N_{1,2}(t)$ and $N_{2,2}(t)$. Merging Relationship (12.18) into Relationship (12.20), we find:

$$\begin{aligned} N_{i,2}(t) &= \frac{t - t_{i-1}}{t_{i+1} - t_{i-1}} \left(\frac{t - t_{i-1}}{t_i - t_{i-1}} N_{i,0}(t) + \frac{t_{i+1} - t}{t_{i+1} - t_i} N_{i+1,0}(t) \right) \\ &+ \frac{t_{i+2} - t}{t_{i+2} - t_i} \left(\frac{t - t_i}{t_{i+1} - t_i} N_{i+1,0}(t) + \frac{t_{i+2} - t}{t_{i+2} - t_{i+1}} N_{i+2,0}(t) \right) \end{aligned}$$

and, in terms of $N_{j,0}(t)$, for $j = i, i+1, i+2$,

$$\begin{aligned} N_{i,2}(t) &= \frac{t - t_{i-1}}{t_{i+1} - t_{i-1}} \frac{t - t_{i-1}}{t_i - t_{i-1}} N_{i,0}(t) \\ &+ \left(\frac{t - t_{i-1}}{t_{i+1} - t_{i-1}} \frac{t_{i+1} - t}{t_{i+1} - t_i} + \frac{t_{i+2} - t}{t_{i+2} - t_i} \frac{t - t_i}{t_{i+1} - t_i} \right) N_{i+1,0}(t) \\ &+ \frac{t_{i+2} - t}{t_{i+2} - t_i} \frac{t_{i+2} - t}{t_{i+2} - t_{i+1}} N_{i+2,0}(t) . \end{aligned}$$

The desired contributions are obtained, in interval $[t_1, t_2]$, by looking at the coefficients of $N_{2,0}(t)$ which are successively related to $i = 2$, $i = 1$ and $i = 0$. Then, we have:

$$N_{2,2}(t) = \frac{t - t_1}{t_3 - t_1} \frac{t - t_1}{t_2 - t_1} ,$$

$$N_{1,2}(t) = \frac{t - t_0}{t_2 - t_0} \frac{t_2 - t}{t_2 - t_1} + \frac{t_3 - t}{t_3 - t_1} \frac{t - t_1}{t_2 - t_1} ,$$

$$N_{0,2}(t) = \frac{t_2 - t}{t_2 - t_0} \frac{t_2 - t}{t_2 - t_1} .$$

Relation (12.21), in $[t_1, t_2]$, is

$$\gamma(t) = \frac{t_2 - t}{t_2 - t_0} \frac{t_2 - t}{t_2 - t_1} P_0 + \left(\frac{t - t_0}{t_2 - t_0} \frac{t_2 - t}{t_2 - t_1} + \frac{t_3 - t}{t_3 - t_1} \frac{t - t_1}{t_2 - t_1} \right) P_1 + \frac{t - t_1}{t_3 - t_1} \frac{t - t_1}{t_2 - t_1} P_2 ,$$

This relation, when uniformly spaced t_is are used (i.e., $t_i = i$ or $t_i = \frac{i}{3}$), leads to:

$$\gamma(t) = \frac{(2-t)^2}{2} P_0 + ((2-t)\frac{t}{2} + (3-t)\frac{t-1}{2}) P_1 + \frac{(t-1)^2}{2} P_2 ,$$

$$\begin{tikzcd}
& & & & & & & N_{0,3} \arrow[dllll] \arrow[drrrr] & & & & & & & \\
& & & N_{0,2} \arrow[dll] \arrow[drr] & & & & & & & & N_{1,2} \arrow[dll] \arrow[drr] & & & \\
& N_{0,1} \arrow[dl] \arrow[dr] & & & & N_{1,1} \arrow[dl] \arrow[dr] & & & & N_{1,1} \arrow[dl] \arrow[dr] & & & & N_{2,1} \arrow[dl] \arrow[dr] & \\
N_{0,0} & & N_{1,0} & & N_{1,0} & & N_{2,0} & & N_{1,0} & & N_{2,0} & & N_{2,0} & & \boldsymbol{N_{3,0}}
\end{tikzcd}$$

Figure 12.3: *Relationships between the $N_{i,k}$ from the root $N_{0,3}$ used to compute* $N_{0,3}$.

and, in $[0,1]$, it is simply:

$$\gamma(t) = \frac{(1-t)^2}{2} P_0 + \frac{1+2t-2t^2}{2} P_1 + \frac{t^2}{2} P_2 ,$$

and we return to the two previous remarks. The curve is $C^{m-1} = C^1$. Indeed, $\gamma(1) = \frac{P_1+P_2}{2}$ and $\gamma'(1) = -P_1 + P_2$ in the first interval where this curve is defined ($[t_1, t_2]$). These values are identical to $\gamma(0)$ and to $\gamma'(0)$ respectively in the next interval ($[t_2, t_3]$).

Degree 3 B-spline

In this case, the curve Γ is defined by:

$$\gamma(t) = \sum_{j=i-2}^{i+1} N_{j,3}(t)\, P_j . \tag{12.22}$$

Thus, the first interval where $\gamma(t)$ is well defined is $[t_2, t_3]$. To simplify the expression of $\gamma(t)$, we first consider the relationships between the $N_{i,k}$s which are used in the present definition, i.e., the $N_{i,k}$s of interest when expanding the relation

$$N_{i,3}(t) = \frac{t - t_{i-1}}{t_{i+2} - t_{i-1}} N_{i,2}(t) + \frac{t_{i+3} - t}{t_{i+3} - t_i} N_{i+1,2}(t) .$$

The diagram relating these coefficients when evaluating the contribution of $N_{0,3}(t)$ is depicted in Figure 12.3. Since only $N_{3,0}(t)$ is non-zero, we just have to visit the branches (Figure 12.3) from the root to terminal node $N_{3,0}(t)$. We obtain successively:

$$N_{0,3}(t) = \frac{t_3 - t}{t_3 - t_0} N_{1,2}(t) ,$$

$$N_{0,3}(t) = \frac{t_3 - t}{t_3 - t_0} \frac{t_3 - t}{t_3 - t_1} N_{2,1}(t) ,$$

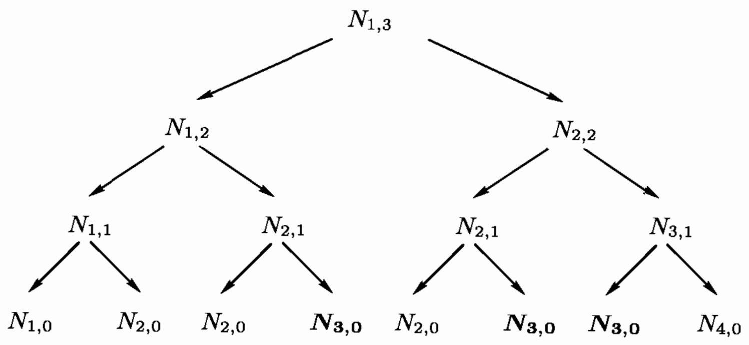


Figure 12.4: *Relationships between the $N_{i,k}$ from the root $N_{1,3}$ used to compute this value.*

$$N_{0,3}(t) = \frac{t_3 - t}{t_3 - t_0}\frac{t_3 - t}{t_3 - t_2} N_{3,0}(t) \quad \text{i.e.,} \quad \frac{t_3 - t}{t_3 - t_0}\frac{t_3 - t}{t_3 - t_2}\,.$$

For uniformly spaced t_is, this relation reduces to

$$N_{0,3}(t) = \frac{(3-t)^3}{6}\,,$$

and, in the interval $[0, 1]$, we obtain

$$N_{0,3}(t) = \frac{1}{6}\,(1-t)^3\,.$$

To obtain the contribution of $N_{1,3}(t)$, we define the diagram with root $N_{1,3}(t)$ (Figure 12.4) and we examine the different branches leading to terminal node $N_{3,0}(t)$.

Thus, considering this tree, we obtain (after certain effort):

$$N_{1,3}(t) = \frac{t - t_0}{t_3 - t_0}\frac{t_3 - t}{t_3 - t_1}\frac{t_3 - t}{t_3 - t_2} + \frac{t_4 - t}{t_4 - t_1}\left(\frac{t - t_1}{t_3 - t_1}\frac{t_3 - t}{t_3 - t_2} + \frac{t_4 - t}{t_4 - t_2}\frac{t - t_2}{t_3 - t_2}\right).$$

For a uniform spacing in t_i, we obtain:

$$N_{1,3}(t) = \frac{t\,(3-t)^2}{6} + \frac{4-t}{3}\left(\frac{(t-1)(3-t)}{2} + \frac{(4-t)(t-2)}{2}\right),$$

resulting, if $t \in [0, 1]$, in:

$$N_{1,3}(t) = \frac{1}{6}\,(4 - 6t^2 + 3t^3)\,.$$

Similarly, we can obtain:

$$N_{2,3}(t) = \frac{1}{6}\,(1 + 3t + 3t^2 - 3t^3) \quad \text{and} \quad N_{3,3}(t) = \frac{1}{6}\,t^3\,.$$

These expressions assume that t lives in $[0, 1]$ for any sub-interval. To complete this range of variation, a change variable must be used. For instance, let us consider the interval $[t_0, t_1, ..., t_4]$ where the curve is defined by the above relationships. Then, for interval $[t_0, t_1]$, we shift by t_0 and we scale by the factor $(t_1 - t_0)^{-1}$ to obtain the parameter t used in $N_{0,3}(t)$. For interval $[t_1, t_2]$, we shift by t_1 and we scale by the ratio $(t_2 - t_1)^{-1}$ to return to $N_{1,3}(t)$, and so on. Then $\gamma(t)$ is defined accordingly. Note that $\gamma(t)$ is C^2.

Exercise 12.5 *Check, for the three above B-splines, the C^{m-1} continuity.*

Specific controls

As previously suggested, some specific controls can be used. Basically, this relies on a proper definition of the nodes.

Multiple control points. Multiple vertices can be used to achieve the end termination of the curve definition (i.e., γ passes through P_0 and P_n). They can also serve to control the shape of the curve at some neighborhood of a control point.

As an example, the uniform B-spline of degree 3 defined by the control points P_i for $i = 0, n$ does not pass through P_0 nor P_n. To complete this feature, we can define the sequence

$$P_0, P_0, P_0, P_1, P_2, ..., P_{n-1}, P_n, P_n, P_n \,,$$

as vertices where P_0 and P_n both have a triple multiplicity.

Exercise 12.6 *Check that this definition of the control points insures that the curve passes through P_0 and P_n. Discuss the tangent at these endpoints.*

Phantom vertices. Phantom control points can be also defined in such a way as to obtain the previous property. In this way, fictitious vertices are defined before P_0 and after P_n. It is also possible to control a tangent in this way.

Multiple knots. Multiple knots (obviously, in a non-uniform curve definition, see below for an example of such a definition) can also serve to control the corresponding curve. These multiple knots act in such a way as to modify the expression of the functions involved in the curve definition.

Each time a node t_i is repeated, the continuity level decreases by one at parameter t_i.

A Bézier curve by means of a B-spline

In numerous applications, it is of interest to take a Bézier curve and to consider it as a B-spline curve. As the B-spline model is more general than the Bézier model, this operation is rather easy. Indeed, the control polygon associated with a Bézier curve is exactly the same if this curve is seen as a B-spline. The only point to be ensured is that the nodes in the B-spline are properly defined. To this end, we could use the following node sequence:

$$\{\underbrace{0,0,...,0}_{n\,\text{times}},\underbrace{1,1,...,1}_{n\,\text{times}}\}\,.$$

Remark 12.21 *The Bézier curve is a polynomial of degree n in $[0,1]$. Then, this continuity is infinite after $n-1$ derivations, we obtain a zero value at any parameter t. If the above sequence of nodes is selected, consisting of two multiple nodes repeated n times, the continuity at $t=0$ and at $t=1$ is -1. In fact, the B-spline curve defined in this way stops at these parameter values and there is no continuity around $t=0$ and $t=1$.*

Exercise 12.7 *Show that in this case the B-spline is a composite curve composed of one portion that is precisely the corresponding Bézier curve. Check that the recursion about a B-spline curve is, in this case, equivalent to the recursion about a Bézier curve (based on the De Casteljau algorithm).*

Relationships between the parameters of a B-spline

Throughout this section, some useful basis relationships are recalled which can be used when considering B-spline curves. The following expressions provide a form for these relationships that make a B-Spline definition coherent. In these expressions,

- nk stands for the number of nodes in the node sequence,
- $\mathcal{C}(t)$ is the continuity range of the curve at parameter t. A negative value indicates that the continuity is not known at t,
- $\mathcal{M}(t)$ denotes the multiplicity of a parameter t in the node sequence. If the value of t is not in the node sequence, then $\mathcal{M}(t)=0$. If the B-spline is uniform $\mathcal{M}(t_i)=1$, for all t_i in the node sequence,
- np stands for the number of points defining the control polygon and
- m is the degree of the B-spline.

Thus, we have:

$$order = m+1$$

$$nk = m+1+np$$

$$\mathcal{C}(t) = order - \mathcal{M}(t) - 1$$

$$np \geq m+1$$

12.7 Rational curves

To begin, we state what an homogenous projection is.

Definition 12.1 *The* homogenous projection *is an application T_h from $\mathbb{R}^{d+1}$ to $\mathbb{R}^d$ defined by:*

$$\{x_1\ ,x_2\ ,\dots\ x_d\ ,\ h\} \mapsto \begin{cases} \textit{if } h=0 \textit{ the point is reported at infinity} \\ \qquad \textit{in the direction } \{x_1\ ,\ x_2\ ,\ \dots\ x_d\} \\ \textit{otherwise } \{\frac{x_1}{h}\ ,\ \frac{x_2}{h}\ ,\ \dots\ \frac{x_d}{h}\} \end{cases}$$

The main interest of such a projection is to allow the definition of a division by means of a projection. Hence, a rational curve in $\mathbb{R}^d$ is the image through such a projection of a polynomial curve in $\mathbb{R}^{d+1}$. The coordinate ω is called the *homogenous coordinate* of the point under consideration.

Definition based on a control polygon

A *rational curve* Γ is defined by an equation like:

$$\gamma(t) = \frac{N(t)}{Q(t)}$$

where N and Q are some polynomials that can be described in terms of curves based on a control polygon. Thus, we have:

$$\begin{aligned} N(t) &= \sum_{i=0}^{m} \omega_i\,\varphi_i(t) P_i \qquad \text{and} \\ Q(t) &= \sum_{i=0}^{m} \omega_i\,\varphi_i(t) \end{aligned} \quad ,$$

which can also be written as:

$$\left\{ \begin{array}{c} N(t) \\ Q(t) \end{array} \right\} = \sum_{i=0}^{m} \varphi_i(t) \left\{ \begin{array}{c} \omega_i P_i \\ \omega_i \end{array} \right\} .$$

This equation enables us to write Γ as:

$$\gamma(t) = T_{\omega_i}\left(\sum_{i=0}^{m} \varphi_i(t) \left\{ \begin{array}{c} \omega_i P_i \\ \omega_i \end{array} \right\} \right) .$$

As a consequence, a rational curve is the homogenous projection of a curve in $\mathbb{R}^{d+1}$ that is based on a control polygon in $\mathbb{R}^d$.

Rational Bézier curve

Following the above remark regarding conics, a rational Bézier definition can be defined. This representation involves the previous input ($n+1$ control points that form a control polygon and the corresponding parameter values) along with a sequence of weights ω_i. It is written as

$$\gamma(t) = \frac{\sum\limits_{i=0}^{n} \omega_i\, B_{i,n}(t)\, P_i}{\sum\limits_{i=0}^{n} \omega_i\, B_{i,n}(t)}\,. \tag{12.23}$$

Remark 12.22 *The question is how to define the t_is as well as the ω_is. For the t_is, we return to the previous remarks. Regarding the weights, they are mostly used to give some preferences to some control points. A weight can be seen as a shape parameter. Actually, increasing the value of ω_i leads to pulling the curve towards the corresponding P_i.*

Remark 12.23 *If the weights are equal (for example, equal to one), we find again the classical definition of a non-rational Bézier curve (as $\sum\limits_{i=0}^{n} B_{i,n}(t) = 1$).*

Exercise 12.8 *Prove that the $B_{i,n}(t)$s sum to 1 (hint: return to the recursion about the $B_{i,n}(t)$s).*

As for Bézier curves, recursion formulas can be found for rational Bézier curves. Nevertheless, these recursions are a little complex (at least, in terms of notations). For instance, a rational Bézier curve may be evaluated by applying the De Casteljau form to both numerator and denominator of the above general expression.

While suitable for handling conics, rational Bézier curves have the same drawbacks as standard Bézier curves. In particular, a large value of n leads to the same remarks as above. For this reason, other functions have been developed, based on local definitions that enjoy the nice properties of the above *global* definitions while avoiding their disadvantages. A first approach involves using a standard (or rational) Bézier definition locally leading to a so-called composite method.

Rational B-spline curve (NURBS)

Similarly, a rational B-spline can be constructed using locally rational spline curves. This definition involves a sequence of weights ω_i and is written as

$$\gamma(t) = \frac{\sum\limits_{i=0}^{n} \omega_i\, N_{i,m}(t)\, P_i}{\sum\limits_{j=0}^{n} \omega_j N_{j,m}(t)}\,. \tag{12.24}$$

Note that NURBS stands for non-uniform rational B-splines where the non-uniformity concerns the t_is. Such a distribution, unlike a uniform distribution, allows for a greater flexibility and may also be used for other kinds of curve representation.

In the following, we give an example of a quadratic NURBS. To this end, we return to a quadratic B-spline before defining the NURBS.

Non-uniform quadratic B-spline with multiple nodes. We return to the case of a quadratic B-spline whose explicit form has been established previously in the case of distinct knots (see Section 12.6). We now review the general formula for the $N_{i,j}$s involved in this B-spline.

$$N_{i,2}(t) = \frac{t-t_{i-1}}{t_{i+1}-t_{i-1}}\frac{t-t_{i-1}}{t_i-t_{i-1}}N_{i,0}(t)$$
$$+ \left(\frac{t-t_{i-1}}{t_{i+1}-t_{i-1}}\frac{t_{i+1}-t}{t_{i+1}-t_i} + \frac{t_{i+2}-t}{t_{i+2}-t_i}\frac{t-t_i}{t_{i+1}-t_i}\right)N_{i+1,0}(t)$$
$$+ \frac{t_{i+2}-t}{t_{i+2}-t_i}\frac{t_{i+2}-t}{t_{i+2}-t_{i+1}}N_{i+2,0}(t)\,.$$

In the case of multiple nodes, this general relation reduces, for example, if $t_i = t_{i-1}$, we have $N_{i,0}(t) = 0$ and

$$N_{i,2}(t) = \left(\frac{t-t_{i-1}}{t_{i+1}-t_{i-1}}\frac{t_{i+1}-t}{t_{i+1}-t_i} + \frac{t_{i+2}-t}{t_{i+2}-t_i}\frac{t-t_i}{t_{i+1}-t_i}\right)N_{i+1,0}(t)$$
$$+ \frac{t_{i+2}-t}{t_{i+2}-t_i}\frac{t_{i+2}-t}{t_{i+2}-t_{i+1}}N_{i+2,0}(t)\,.$$

and so on. For instance, if $t_{i+2} = t_{i+1} = t_i$, we have simply:

$$N_{i,2}(t) = \frac{t-t_{i-1}}{t_{i+1}-t_{i-1}}\frac{t-t_{i-1}}{t_i-t_{i-1}}N_{i,0}(t)\,.$$

We define a sequence of knots t_i for $i = 0, ..., 9$ as follows:

$$t_i = [\,0,0,0,1,1,2,2,3,3,3\,]\,.$$

As a consequence we have successively $N_{1,0}(t) = N_{2,0}(t) = N_{4,0}(t) = N_{6,0}(t) = N_{8,0}(t) = N_{9,0}(t) = 0$. It is easy to obtain the $N_{i,2}$s. In our example, we have, for $i = 0$, $N_{0,2}(t) = 0$. For $i = 1$, we find:

$$N_{1,2}(t) = \frac{t_{i+2}-t}{t_{i+2}-t_i}\frac{t_{i+2}-t}{t_{i+2}-t_{i+1}}N_{3,0}(t) \;=\; (1-t)^2\,N_{3,0}(t)\,.$$

For $i = 2$ and the following value of i, we have:

$$N_{2,2}(t) = \left(\frac{t-t_{i-1}}{t_{i+1}-t_{i-1}}\frac{t_{i+1}-t}{t_{i+1}-t_i} + \frac{t_{i+2}-t}{t_{i+2}-t_i}\frac{t-t_i}{t_{i+1}-t_i}\right)N_{3,0}(t) = 2\,t(1-t)\,N_{3,0}(t)$$

$$N_{3,2}(t) = \frac{t-t_{i-1}}{t_{i+1}-t_{i-1}}\frac{t-t_{i-1}}{t_i-t_{i-1}}N_{3,0}(t) \;+\; \frac{t_{i+2}-t}{t_{i+2}-t_i}\frac{t_{i+2}-t}{t_{i+2}-t_{i+1}}N_{5,0}(t)\,.$$

$$N_{3,2}(t) = t^2\,N_{3,0}(t) \;+\; (2-t)^2\,N_{5,0}(t)\,.$$

$$N_{4,2}(t) = \left(\frac{t-t_{i-1}}{t_{i+1}-t_{i-1}}\frac{t_{i+1}-t}{t_{i+1}-t_i} + \frac{t_{i+2}-t}{t_{i+2}-t_i}\frac{t-t_i}{t_{i+1}-t_i}\right)N_{5,0}(t)$$

$$N_{4,2}(t) = 2\,(t-1)(2-t)N_{5,0}(t)$$

$$N_{5,2}(t) = \frac{t-t_{i-1}}{t_{i+1}-t_{i-1}}\frac{t-t_{i-1}}{t_i-t_{i-1}}N_{5,0}(t) \;+\; \frac{t_{i+2}-t}{t_{i+2}-t_i}\frac{t_{i+2}-t}{t_{i+2}-t_{i+1}}N_{7,0}(t)\,.$$
$$N_{5,2}(t) = (t-1)^2\,N_{5,0}(t) \;+\; (3-t)^2\,N_{7,0}(t)\,.$$
$$N_{6,2}(t) = \left(\frac{t-t_{i-1}}{t_{i+1}-t_{i-1}}\frac{t_{i+1}-t}{t_{i+1}-t_i} + \frac{t_{i+2}-t}{t_{i+2}-t_i}\frac{t-t_i}{t_{i+1}-t_i}\right)N_{7,0}(t)$$
$$N_{6,2}(t) = 2\,(t-2)(3-t)\,N_{7,0}(t)$$
$$N_{7,2}(t) = \frac{t-t_{i-1}}{t_{i+1}-t_{i-1}}\frac{t-t_{i-1}}{t_i-t_{i-1}}N_{7,0}(t)$$
$$N_{7,2}(t) = (t-2)^2\,N_{7,0}(t)\,.$$

Written in terms of the interval $[0,1]$, we obtain:

$$N_{1,2}(t) = (1-t)^2\,N_{3,0}(t)\,.$$
$$N_{2,2}(t) = 2\,t(1-t)\,N_{3,0}(t)$$
$$N_{3,2}(t) = t^2\,N_{3,0}(t) \;+\; (1-t)^2\,N_{5,0}(t)\,.$$
$$N_{4,2}(t) = 2\,t(1-t)N_{5,0}(t)$$
$$N_{5,2}(t) = t^2\,N_{5,0}(t) \;+\; (1-t)^2\,N_{7,0}(t)\,.$$
$$N_{6,2}(t) = 2\,t(1-t)\,N_{7,0}(t)$$
$$N_{7,2}(t) = t^2\,N_{7,0}(t)\,.$$

The first part of the curve definition involves the sequence $[\,0,0,0,1\,]$, the second uses $[\,0,0,1,1\,]$, then we have successively $[\,0,1,1,2\,]$, $[\,1,1,2,2\,]$, $[\,1,2,2,3\,]$, $[\,2,2,3,3\,]$ and finally $[\,2,3,3,3\,]$.

An example of quadratic NURBS. Using the above B-spline, we give an example of a quadratic NURBS. In addition to the above sequence of knots, we consider a sequence of weights ω_i for $i = 1,...,7$ defined as:

$$\omega_i = [\,1,\frac{1}{2},1,\frac{1}{2},1,\frac{1}{2},1\,]\,.$$

Then, given the P_i,s, we consider the first segment where the NURBS is defined, say the portion corresponding to $[\,0,0,0,1\,]$. Following Relation (12.24), we obtain

$$\gamma(t) = \sum_{i=1}^{3}\left(\frac{\omega_i\,N_{i,2}(t)\,P_i}{\sum\limits_{j=1}^{3}\omega_j N_{j,2}(t)}\right), \tag{12.25}$$

which is:

$$\gamma(t) = \frac{(1-t)^2\,P_1 + t\,(1-t)\,P_2 + t^2\,P_3}{1-t+t^2}\,.$$

As P_is, we take the following control polygon, $P_1 = {}^t(0,0)$, $P_2 = {}^t(1,0)$ and $P_3 = {}^t(\frac{1}{2},\frac{\sqrt{3}}{2})$. Then, $\gamma(t)$ passes through P_1 and P_3, and $\gamma(t)$ is nothing other than a segment of the circle centered at point ${}^t(0,\frac{\sqrt{3}}{3})$ whose radius is $r = \frac{\sqrt{3}}{3}$.

Exercise 12.9 *Check that $\gamma(t)$ is the above portion of the previous circle.*

Indeed, we have defined a 120° sector of the full circle. To obtain the entire circle, we use the following control points P_3, P_4, P_5 and $P_5, P_6, P_7 = P_1$ as shown in Figure 12.5.

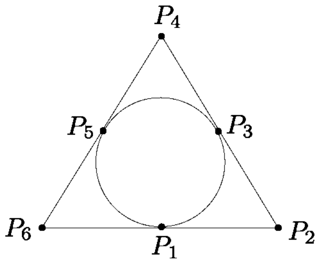


Figure 12.5: *The control polygon used to define a circle by a quadratic NURBS.*

12.8 Curve definitions and numerical issues

Depending on the way the curve Γ is defined, the quantities we are interested in (length, tangent, normal, curvature, etc.) are more or less easy to compute. Moreover, only approximate values are obtained which are more or less accurate. This aspect is familiar to anyone who has made use of a curve (a surface) in a computer program and is clearly shown in the following with the help of some simple examples. In practice, Γ can be defined in terms of a parameter t, thus using a function $\gamma(t)$ or Γ can be known as a function of the curvilinear abscissa s by a relation like $\gamma(s)$. It could be observed that, for a given Γ, these two functions, while denoted similarly, have different expressions.

An exact parameterization of a curve

We consider a very simple example. Let Γ be the quarter of a circle defined in terms of a parameter t ranging from 0 to $\frac{\pi}{2}$. Obviously the function[7]:

$$\gamma(t) = {}^t(r\,cos(t) \quad r\,sin(t))\,,$$

is an exact parameterization of the circle of radius r whose center is the origin. Then using the previous material (Chapter 11), we will compute the length of Γ and, for instance, its radius of curvature. We have:

$$\gamma'(t) = {}^t(-r\,sin(t) \quad r\,cos(t))\,,$$

then $||\gamma'(t)|| = r$ and, following Formula (11.1): $s = \int_0^{\frac{\pi}{2}} r d\theta = r\frac{\pi}{2}$. To compute the curvature, we need to evaluate $\gamma''(t)$.

$$\gamma''(t) = {}^t(-r\,cos(t) \quad -r\,sint(t))\,,$$

[7]The notation ${}^t(a \ \ b)$ stands for the vector whose components are a and b.

then $\gamma'(t) \wedge \gamma''(t) = {}^t(0\ 0\ r^2)$ whose norm is r and the curvature is $C(t) = \frac{1}{r}$, meaning that the radius of curvature is nothing other than r, as expected. In conclusion, the *a priori* expected values, for both s and C, are obtained without any difficulty.

A given parameterization of a curve

We consider the same curve Γ as above. We define now an approximate representation of this curve using one of the methods discussed in this chapter (a composite curve). To this end, we assume that $A = {}^t(r\ 0)$, $B = {}^t(0\ r)$ along with[8] $\tau_A = {}^t(0\ r\sqrt{2})$ and $\tau_B = {}^t(-r\sqrt{2}\ 0)$ are supplied and correspond to a parameter $t \in [0, 1]$, i.e., A (resp. B) and τ_A (resp. τ_B) correspond to $t = 0$ (resp. $t = 1$). Then, following Relationship (12.5), we can define a curve representation $\gamma(t)$ by:

$$\gamma(t) = A + \tau_A\, t + (3(B - A) - 2\tau_A - \tau_B)\ t^2 + (-2(B - A) + \tau_A + \tau_B)\ t^3\,,$$

then:

$$\gamma'(t) = \tau_A\ + 2\ (3(B - A) - 2\tau_A - \tau_B)\ t + 3\ (-2(B - A) + \tau_A + \tau_B)\ t^2\,,$$

which are such that $\gamma(0) = A$, $\gamma(1) = B$, $\gamma'(0) = \tau_A$ and $\gamma'(1) = \tau_B$. To obtain s between 0 and 1, the length of Γ, we have to compute $s(t) = \int_0^1 ||\gamma'(\theta)||\, d\theta$ which is:

$$\int_0^1 \sqrt{\langle \gamma'(\theta), \gamma'(\theta)\rangle}\, d\theta\,.$$

This expression involves the square root of a polynomial of degree four, which is therefore tedious to compute exactly. Thus, a quadrature may be used. Using a low degree formula, like the trapezoidal rule, leads to the following approximation

$$\int_0^1 \sqrt{\langle \gamma'(\theta), \gamma'(\theta)\rangle}\, d\theta = \frac{1}{2}\left(\sqrt{\langle \gamma'(0), \gamma'(0)\rangle} + \sqrt{\langle \gamma'(1), \gamma'(1)\rangle}\right) = r\sqrt{2}\,,$$

which is very imprecise (in particular, this length is equal to the length of segment AB) as might have been expected. By introducing the mid-value (say $t = \frac{1}{2}$) in the quadrature, we obtain:

$$s = \frac{1}{4}\left(\sqrt{\langle \gamma'(0), \gamma'(0)\rangle} + 2\sqrt{\langle \gamma'(\frac{1}{2}), \gamma'(\frac{1}{2})\rangle} + \sqrt{\langle \gamma'(1), \gamma'(1)\rangle}\right)\,,$$

i.e., about $1.517\,r$ which is again quite a bad evaluation. Introducing the quarters, we have now, for the length, a value close to $1.533\,r$ which again is not suitable. Using more nodes in the quadrature or using a more precise formula probably results in a better evaluation of the curve length but, on the other hand, leads to a large computational effort.

[8] τ is short for $\vec{\tau}$.

-	-	2 nodes	3 nodes	5 nodes
2 control points	$[0, \frac{\pi}{2}]$	1.414	1.517	1.533
3 control points	$[0, \frac{\pi}{4}]$	1.5307	1.5598	1.5663
4 control points	$[0, \frac{\pi}{6}]$	1.5529	1.5661	1.5692
5 control points	$[0, \frac{\pi}{8}]$	1.5607	1.5682	1.57004
9 control points	$[0, \frac{\pi}{32}]$	1.5683	1.5701	1.57063

Table 12.3: Evaluation of the length of Γ as a function of its parameterization. The targeted value is $\frac{\pi}{2} \approx 1.5707$.

-	1 curve	2 curves	3 curves	4 curves	8 curves
$t = 0$	.63	.86	.93	.96	.9904
$t = .25$	1.04	1.01	1.008	1.004	1.0012
$t = .5$	1.31	1.07	1.03	1.019	1.0048

Table 12.4: Evaluation of the radius of curvature of Γ as a function of its parameterization. This radius must be constant and the targeted value is 1.

Indeed, the definition of the above control points and tangents leads to a curve which is not close enough to the real curve[9] and thus, the above rough approximation results in quite bad evaluations for the quantities we are interested in (the length in this case).

One curve parameterization with more precise definitions

We consider the same curve Γ and the same method of parameterization. But we now define this curve using its midpoint. We fix $r = 1$. In fact, we define half of the curve (i.e., from 0 to $\frac{\pi}{4}$ in terms of angle). Then, we obtain for the length of the initial curve the values 1.5307, 1.5598 and 1.56633 based on the number of nodes in the quadrature. Splitting the initial curve into three composite curves (the first from 0 to $\frac{\pi}{6}$) we now have 1.5529, 1.5661 and 1.5692. Using four composite portions (the first from 0 to $\frac{\pi}{8}$), we have 1.5607, 1.5682 and 1.57004. The final example concerns eight composite curves (the first from 0 to $\frac{\pi}{32}$).

All these values are given in Table 12.3 in terms of the number of composite curves used (i.e., how many control points and tangents have been used) and, for a given composite curve, in terms of the number of nodes used in the quadrature.

Similarly, using Relationship (11.4), we can evaluate the curvature of Γ as obtained when a particular number of composite curves are used to discretize the curve. To this end, we need to compute $\gamma''(t)$. For our example, this is simply

$$\gamma''(t) = 2\ (3(B - A) - 2\tau_A - \tau_B) + 6\ (-2(B - A) + \tau_A + \tau_B)\ t\,.$$

Table 12.4 presents the radius of curvature as a function of t for the different above parameterizations.

[9]Anyway, we are trying to approach a circle by a polynomial of degree three!

To sum up, both the lengths and the curvatures are obtained with a greater or lesser degree of accuracy. Approximate tangents and normals can also be obtained in the same way.

Using a polyline

The same exercise in the case where a polyline is given as the discrete definition of a curve Γ is now discussed. Let P_i, P_{i+1} be a segment of the polyline, $i = 0, n-1$, $\gamma(t)$ is defined as a curve passing through the P_is. The length of $\gamma(t)$ is nothing other than the sum of the length of the $n+1$ segments constituting the polyline. As a function of n, we return to Table 12.3 (first row) to have an approximate value of the length of Γ. The other characteristics of Γ can be obtained using the P_i, P_{i+1}s.

If Γ is a curve in $\mathbb{R}^2$, a simple construction enables us to find approximate curvature, tangent and normal at some t.

Note that t can be chosen so that $t = 0$ for P_i and $t = 1$ for P_{i+1}, meaning that t is local to the segment under consideration. Another possible t definition globally considers the different segments and is based on their lengths, then we can obtain (after scaling) $t = 0$ for P_0 and $t = 1$ for P_n.

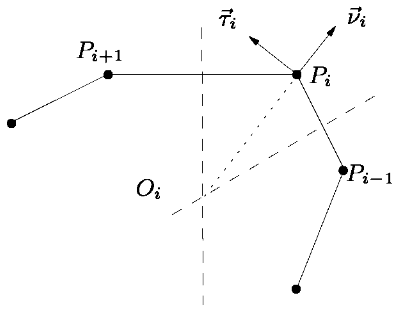


Figure 12.6: *Curvature, tangent and normal that can be obtained from a simple polyline (in two dimensions).*

We consider three consecutive vertices, P_{i-1}, P_i and P_{i+1}. We construct the perpendicular bisector of P_{i-1}, P_i and that of P_i, P_{i+1}. Provided the two segments are not aligned, they intersect in a point denoted as O_i, we define $r_i = ||O_i P_i||$ and $\vec{\nu}_i = \frac{\vec{P_i O_i}}{||\vec{P_i O_i}||}$, then we introduce $\vec{\tau}_i$ as the unit vector perpendicular to $\vec{\nu}_i$. Indeed, using this simple construction results in approximate values for the center of curvature of Γ at point P_i, say O_i, the radius of curvature at this point (r_i), the unit normal and tangent at P_i ($\vec{\nu}_i$ and $\vec{\tau}_i$). Note that these quantities are not well defined at P_0 and P_n for an open polyline.

Thus, $\gamma(t)$ can be described in terms of these discrete quantities. In other words, local knowledge of the presumed curve can be accessed. For instance, we can use the osculating circle at point P_i and say that:

$$\gamma(t) \approx P_i + (t - t_i)\,\vec{\tau}_i + \frac{r_i}{2}(t - t_i)^2\,\vec{\nu}_i$$

in some vicinity of P_i, i.e., for t close to t_i the parameter value associated with P_i.

To obtain a continuous definition of the curve, it is necessary to choose a mathematical representation. For instance, we can return to the above parameterization. In this case, a given characteristic can be evaluated not only at the segment endpoints but anywhere else as well (for any arbitrary value of parameter t).

For a curve in $\mathbb{R}^3$, we can also evaluate these characteristics. Indeed, the above geometric construction in two dimensions can be written in a more general form that extends to three dimensions.

First, we consider again two consecutive segments[10], P_{i-1}, P_i and P_i, P_{i+1}. We define the approximate normal at P_i as

$$\vec{\nu}_i \approx \frac{\dfrac{\overrightarrow{P_iP_{i-1}}}{||\overrightarrow{P_iP_{i-1}}||} + \dfrac{\overrightarrow{P_iP_{i+1}}}{||\overrightarrow{P_iP_{i+1}}||}}{\left|\left|\dfrac{\overrightarrow{P_iP_{i-1}}}{||\overrightarrow{P_iP_{i-1}}||} + \dfrac{\overrightarrow{P_iP_{i+1}}}{||\overrightarrow{P_iP_{i+1}}||}\right|\right|}. \tag{12.26}$$

Then we can obtain an approximation of r_i the radius of curvature at P_i. Actually, we can define:

$$\rho_i \approx \min_j \frac{1}{2} \frac{\langle \overrightarrow{P_iP_j}, \overrightarrow{P_iP_j} \rangle}{\langle \vec{\nu}_i, \overrightarrow{P_iP_j} \rangle}, \tag{12.27}$$

where $j = i-1$ or $i+1$. Note that a formula like:

$$\rho_i \approx \frac{1}{4} \left(\frac{\langle \overrightarrow{P_iP_{i-1}}, \overrightarrow{P_iP_{i-1}} \rangle}{\langle \vec{\nu}_i, \overrightarrow{P_iP_{i-1}} \rangle} + \frac{\langle \overrightarrow{P_iP_{i+1}}, \overrightarrow{P_iP_{i+1}} \rangle}{\langle \vec{\nu}_i, \overrightarrow{P_iP_{i+1}} \rangle} \right), \tag{12.28}$$

also provides an approximation of ρ_i.

Proof. Let M be the midpoint of P_iP_{i-1}, we construct O the point where $\vec{\nu}_i$ and the perpendicular bisector of P_iP_{i-1} intersect (thus a line passing through M). Let α be the angle formed between $\vec{P_iM}$ and $\vec{\nu}_i$, we have $\langle \overrightarrow{P_iM}, \vec{\nu}_i \rangle = ||\overrightarrow{P_iM}|| \cos\alpha$ and $\cos\alpha = \frac{||\vec{P_iM}||}{r_i}$ from which Relationship (12.27) yields. □

Remark 12.24 *Relationships (12.26) and (12.27) extend to curves in three dimensions. They can also be extended in the case of a curve traced on a surface. In this case, a mean normal can be computed along with the minimum of the various radii of curvature that can be defined by considering the various P_j neighboring P_i.*

12.9 Towards a "pragmatic" curve definition?

Due to the number and the variety of possible curve definitions and bearing in mind some numerical troubles that can happen, we now attempt to define a context which is well-suited to the meshing application at hand. Basically, we are interested

[10]We can consider the plan where P_{i-1} and P_{i+1} pass through.

in curves from this viewpoint. Indeed, we want to mesh a curve (as will be seen in Chapter 14), that is in general the boundary of a domain (in the plane or in space). The aim is then to take this curve discretization as input data for the construction of a mesh of the domain of which this curve is a boundary.

It is then clear that a meshing process developed for this purpose must be as general and as stand-alone as possible while being reasonably simple. Thus, it seems unrealistic to guess that a meshing algorithm will know in detail *all* the curve representations that may be encountered (if so, we will face a "huge monster", tedious to design, hard to update, and so on.). For this reason, we want to simplify the context to that which is strictly necessary for the implementation of a curve meshing algorithm (the same is true for a surface meshing algorithm).

Curve definitions and meshing topics

As previously seen, we encounter two types of representations for a curve. One approach makes use of a parametric definition while the other makes use of a discrete definition (i.e., a mesh). In the first case, the curves result from a CAD system and actually exist only if it is possible to query this system in order to collect the useful geometric characteristics (by default to have the precise representation of the underlying model). In the second case, the discrete data is directly usable (given some assumptions) so as to collect the desired characteristics (or, at least, to find their approximate values).

Whether one or the other of these approaches is adopted, we can see that a meshing problem requires (for simplicity, let us just look at a curve meshing problem):

- access to the CAD modeler used when defining the curve, in the first case, and
- an internal choice of a "CAD" enabling us to find the useful information from a discrete set of values, say a polyline, in the second case.

Key ideas

The approach we would like to follow is to mesh a curve without explicitly using its CAD definition in order to be independent (thus obtaining some extent of generality and computational efficiency).

Thus, we adopt a rather different point of view from a classical approach. Our goal is to be independent of a particular curve definition (and thus of the CAD system the user is familiar with) when a meshing task is demanded. To this end, the key idea is as follows:

- the curve is defined by a sequence of control points and *possibly* by some other information,
- the curve passes through these points (in this sense, an interpolation type is retained),

- afterwards, a mathematical support is defined, using the points (and the extra information, if any) by means of a unique composite curve definition of low degree,
- then this mathematical definition becomes the curve definition and is used to define what is necessary for our purpose.

Following these principles, the user's responsibility (and thus that of the CAD system) is limited to providing a sequence of points which makes the construction of a sufficiently precise definition of the geometry possible.

From this point of view, a reasonable synthetic scheme for curve definition could be composed of two different steps:

Step 1: (user's responsibility) use the CAD system available to construct a polyline while giving the corners[11] if any.

Step 2: (pre-meshing process responsibility) use the previous polyline to construct a fixed geometric definition or use this polyline directly.

In this way, the process of meshing a curve (see Chapter 14) is based solely on the above geometric definition (and the original CAD definition is no longer considered).

Construction of a well-suited discrete definition

In practice, this issue is the key to obtaining a curve definition that is sufficiently accurate. The problem reduces to finding an appropriate polyline to represent the geometry.

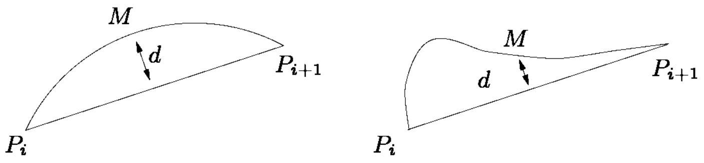


Figure 12.7: *Controlling the gap between a polyline and a curve.*

A rather obvious method can be used to control the accuracy of a "polylinization" of a curve. Using his favorite CAD system, the user defines one member of the desired polyline. Let P_i, P_{i+1} be this segment. Let t_i and t_{i+1} be the two parameter values corresponding to P_i and P_{i+1}. It is easy to obtain point M corresponding to $t = \frac{t_i+t_{i+1}}{2}$. Then, d the distance between M and P_i, P_{i+1} is computed. Let δ be a threshold value; if $d < \delta \, ||P_i, P_{i+1}||$, then the current segment is judged correct. Otherwise, point M is used to define the two segments P_i, M and M, P_{i+1} which are in turn analyzed in the same way to decide whether or not additional subdivisions are required.

[11]If the corners are not explicitly supplied, it is nevertheless possible to guess them. To this end, an analysis of the angles between two consecutive segments is made. Note that a threshold value must be used which may lead to certain ambiguities in some cases.

Note that, in general, the maximum gap is not necessarily obtained at the midpoint as defined above (see Figure 12.7). Thus, to prevent a bad capture of the curve, sample points can be tested to decide whether the current segment is sufficiently fine.

★
★ ★

As we have just seen, there are various curve definitions each of which offers both advantages and weaknesses. It is therefore no easy matter to decide which definition is the most appropriate in general and, anyway, such a discussion is beyond the scope of this book. However, if one idea is to be retained after this discussion, we think it is clear that the adopted meshing techniques for curves (surfaces, in a similar way) must be, whenever possible, developed without regard to the model used during the design of the curve[12] (the surface).

On the other hand, the few examples of actual computation regarding curves defined by one or other of the models seen above have brought to light some perverse numerical effects which will require further consideration.

[12]In specific, we would like to start from a minimal interface under the responsibility of the CAD system.

Chapter 13

Surface Modeling

The aim of this chapter is to review some methods for surface definition. A surface can be defined using various categories of methods, and, in particular, we encounter parametric, implicit or explicit surfaces. A parametric surface, given u and v two parameters living in some interval, is the data of a function σ whose values $\sigma(u, v)$ describe the surface. Implicit surfaces are given by a relation like $f(x, y, z) = 0$ while explicit surfaces take the form $z = f(x, y)$.

In the first sections of this chapter we discuss the methods extensively used in practice (for instance, in CAD systems), i.e., parametric surfaces. Most methods developed for surface definition are derived from methods used for curve definition. Thus, the material discussed in Chapter 12 will be reviewed here and extended to the case of surfaces.

★
★ ★

First, we introduce the basic notions related to surface definitions (as we did for curves in Chapter 12) to give a rough idea of the various methods that can be used to define (construct) a surface. It is not our intention to be exhaustive, and we refer the reader to the *ad-hoc* literature. In this respect, references listed in Chapter 12 for curve topics remain relevant.

Surface definitions can be classified into several categories depending on what the surface looks like or what the geometry of the surface must be. Particular surfaces are briefly discussed including surfaces of *revolution*, *ruled* surfaces and *sweep* surfaces. We then turn to surfaces that can be defined by means of a tensor product and we discuss surfaces related to an interpolation scheme based on a set of points (for instance, Coons patches). Next, we look at tensor product based patches based on a control polyhedron. We briefly examine rational patches before turning to patches that are not included in the above categories. Specifically, we mention the case of patches with an arbitrary topology or those whose control differs from the classical form. To conclude, as we did for the curves, we consider the case of *composite* surfaces which are widely used in CAD systems. The initial surface is split into different portions and each of these is defined using one of the surface representation methods. In addition, some specific properties about regularity can be required at the junction of two such portions.

Notations. Parameters used for surface definitions are commonly denoted by u and v (together with w in the case of a triangular patch) unlike those used for curve definitions which was denoted by t in Chapter 12. Nevertheless, in this chapter a curve parameter will be denoted by u (or v) as well since such a curve is now seen as a component of a surface with such parameters. Recall that curves, Γ, are noted as $\gamma(.)$ while surfaces, Σ, will be denoted by $\sigma(.,.)$. While, for instance, $\sigma(u,v)$ is a point on surface Σ, for a given pair (u,v), in some cases, we make no distinction between $\sigma(u,v)$ (where (u,v) varies) and the surface Σ.

13.1 Specific surfaces

Numerous surfaces commonly used in industry have a specific geometric nature. This results from the fact that they must be manufactured. Owing to this, the surfaces must respect certain constraints regarding their manufacture, e.g. have a degree of detail that is compatible with the precision of the tool used.

Surfaces of revolution

Some particular surfaces can be defined by means of the revolution of a two-dimensional entity (point[1], line, planar curve, closed or open polygon) around an axis in space. The position of the line (curve) with respect to the axis leads to various surfaces such as cylinders, truncated cones, solid discs, hyperboloids of one sheet, cones with a cylindrical hole, spheres, ellipsoids, torus, etc.

The entity to be rotated is a function of one parameter and the rotation is determined by another parameter (namely the rotation angle), so a *surface of revolution* is a biparametric function since a point on this surface is specified by two parameters.

Figure 13.1 (left-hand side) demonstrates an example of a surface of revolution. We can see the curve $\gamma(u)$, the rotation axis and, for a given value v of the angle, the point $\sigma(u,v)$ in the surface.

Trimmed surfaces

Given a point $P(u)$ function of a parameter u and a vector $\vec{V}(u)$ such that $\vec{V}(u) \neq 0, \forall u$, then the relationship:

$$\sigma(u,v) = P(u) + v\,\vec{V}(u)\,, \tag{13.1}$$

defines a *ruled* surface.

Given two curves Γ_1 and Γ_2 and the corresponding parameterizations, $\gamma_1(u)$ and $\gamma_2(u)$, a surface can be defined as

$$\sigma(u,v) = (1-v)\,\gamma_1(u) \;+\; v\,\gamma_2(u)\,.$$

[1]In this case a curve results from the revolution.

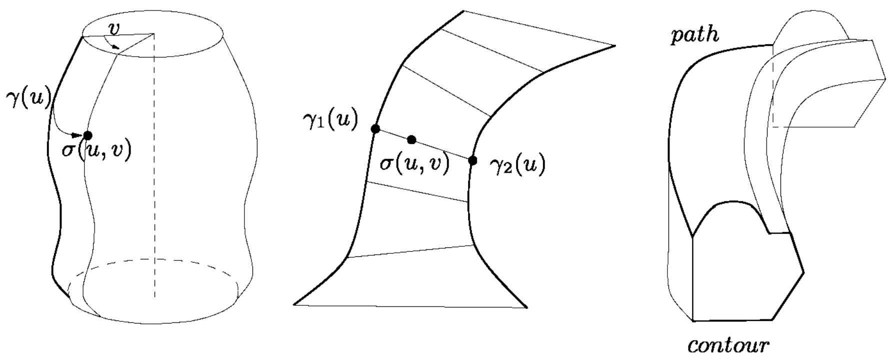


Figure 13.1: *Examples of particular surfaces. Left-hand side, a surface of revolution, middle, a ruled surface; right-hand side, an extruded surface.*

This surface is also a *ruled surface.* In fact, it conforms to the general definition of a ruled surface with $P(u) = \gamma_1(u)$ and $\vec{V}(u) = \gamma_2(u) - \gamma_1(u)$.

In Figure 13.1 (middle) is an example of a ruled surface. We can see, for instance, the two curves $\gamma_1(u)$ and $\gamma_2(u)$ together with the net linking them, i.e., for a given v, the point $\sigma(u, v)$ of the thus-defined surface.

Extruded or sweeping surfaces

Given a path and a two-dimensional entity, a surface is obtained by traversing this entity along the given path (a line, a curve). In this way, the defined surface is a *sweeping surface* or an *extruded surface.* Note that an entity like a line results in a ruled surface, as defined in the previous section. A simple example of an extruded surface is given in Figure 13.1 (right-hand side).

13.2 Interpolation-based surfaces

We turn now to arbitrary surfaces which nevertheless are suitable for a tensor product or an interpolation based definition. The easiest way to define a surface is to extend the material used for curve definition (Chapter 12). This leads to quadrilateral patches that can be considered as the *tensor product* of two curve definitions. Based on the way the curves are defined, we obtain various surface definitions. To give a basic feeling of a tensor product based method, we take a rather simple case, namely a bilinear interpolation.

Tensor product based patches (introduction)

We consider a parametric space in $\mathbb{R}^2$ whose parameters u and v live in $[0, 1]$. Given four points in $\mathbb{R}^3$, $P_{i,j}$ for $i = 0, 1$ and $j = 0, 1$ where $P_{i,j}$ corresponds to

$u = i$ and $v = j$, we can define four lines:

$$\text{for } v = 0,\ \gamma(u) = (1-u)\,P_{0,0} + u\,P_{1,0} = \sum_{i=0}^{1} B_i^1(u)\,P_{i,0}\,,$$

$$\text{for } v = 1,\ \gamma(u) = (1-u)\,P_{0,1} + u\,P_{1,1} = \sum_{i=0}^{1} B_i^1(u)\,P_{i,1}\,,$$

$$\text{for } u = 0,\ \gamma(v) = (1-v)\,P_{0,0} + v\,P_{0,1} = \sum_{j=0}^{1} B_j^1(v)\,P_{0,j}\,,$$

$$\text{for } u = 1,\ \gamma(v) = (1-v)\,P_{1,0} + v\,P_{1,1} = \sum_{j=0}^{1} B_j^1(v)\,P_{1,j}\,,$$

with $B_0^1(u) = 1 - u$, $B_1^1(u) = u$ and similar expressions for $B_j^1(v)$ and where the ^{1} in B_j^1 refers to the degree of the function. Combining these curves, we can construct a surface $\sigma(u, v)$ defined by:

$$\sigma(u,v) = \sum_{i=0}^{1}\sum_{j=0}^{1} B_i^1(u)\,B_j^1(v)\,P_{i,j}\,, \qquad (13.2)$$

which is the tensor product based on the above curve definitions. In matrix form, such a curve Γ can be written, for each component, as:

$$\gamma(t) = [\mathcal{U}][\mathcal{M}][\mathcal{P}] \quad \text{or} \quad \gamma(t) = [\mathcal{B}(u)][\mathcal{P}]\,,$$

where $[\mathcal{U}] = [1-u, u]$ is the basis functions of the representation,

$$[\mathcal{M}] = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

is the coefficient matrix of the function in the above basis (for one of the above curves), $[\mathcal{P}] = {}^t[P_{0,0}\,, P_{1,0}]$ is the row of the control points (for the first curve) and, finally, $[\mathcal{B}(u)] = [\mathcal{U}][\mathcal{M}]$.

Similarly, the surface of Equation (13.2) can be seen in matrix form as:

$$\sigma(u,v) = [\mathcal{U}][\mathcal{M}][\mathcal{P}]\,{}^t[\mathcal{M}]\,{}^t[\mathcal{V}] \quad \text{or again} \quad \sigma(u,v) = [\mathcal{B}(u)][\mathcal{P}]\,{}^t[\mathcal{B}(v)]\,, \qquad (13.3)$$

where, now, we have:

$$[\mathcal{P}] = \begin{bmatrix} P_{0,0} & P_{0,1} \\ P_{1,0} & P_{1,1} \end{bmatrix},$$

and $[\mathcal{B}(u)]$ is the above matrix form of the (curve) function. As a consequence, any tensor product surface definition can be expressed in a similar matrix form based on the curve matrix form.

Remark 13.1 *In general, the matrix form of a tensor product surface is:*

$$\sigma(u,v) = [\mathcal{B}^n(u)][\mathcal{P}]\,{}^t[\mathcal{B}^m(v)]\,, \tag{13.4}$$

where the two $[\mathcal{B}]$s may be different. Such a case is encountered, for instance, when the network of control points includes $n+1$ points in one direction (the u-direction) and $m+1$ points in the other, n and m also referring to the degree of the representation.

Depending on what the B_i^n's are, this kind of method results in various surface definitions. Indeed, both the value of n and the nature of the coefficients in the B_i^n's lead to different definitions. We return, as for the curve case, to interpolation methods where the surface passes through the given (control) points (Lagrange type methods) or passes through the given points while, at the same time, matching some derivatives at these points (Hermite methods) or again to extrapolation methods such as Bézier, B-spline and many others.

Interpolation-based patches

The "simplest" tensor product that can be used corresponds to the definition of an interpolation method at the curve level. We then encounter the two methods previously seen, the Lagrange or the Hermite interpolation.

Lagrange interpolation

In this case, given an $(n+1)\times(m+1)$ array[2] of data points, $P_{i,j}$, the problem is one of finding the coefficients of the matrices $[\mathcal{B}]$ such that the surface:

$$\sigma(u,v) = [\mathcal{B}^n(u)][\mathcal{P}]\,{}^t[\mathcal{B}^m(v)]$$

passes through the $P_{i,j}$'s. In other words, if $u = u_i$ and $v = v_j$ are the parameters of point $P_{i,j}$, we want to have:

$$P_{i,j} = [\mathcal{B}^n(u_i)][\mathcal{P}]\,{}^t[\mathcal{B}^m(v_j)] \quad \text{for all} \quad i \quad \text{and} \quad j\,.$$

Using as $[\mathcal{B}]$ the Lagrange interpolate of Chapter 12 gives the solution. For example, for $n = m = 2$, given 9 control points, the above equation with $[\mathcal{B}^n(u)] = [\mathcal{U}][\mathcal{M}]$ in which:

$$[\mathcal{U}] = [u^2, u, 1] \quad \text{and}$$

$$[\mathcal{M}] = \begin{pmatrix} 2 & -4 & 2 \\ -3 & 4 & -1 \\ 1 & 0 & 0 \end{pmatrix},$$

and similar expressions for $[\mathcal{B}^m(v)]$ defines a Lagrange tensor product resulting in a surface passing through the 9 data points.

[2]Where n and m are not necessarily equal.

Hermite interpolation. As for the curve case, a Hermite interpolation involves derivatives of the control points along with these points. The tensor product used to define the surface leads to the same discussion as above. In particular, an equation like Equations (12.3) or (12.4) characterizes the surface using a coefficient matrix based on the basic curves of the tensor product. For instance, the popular bicubic Hermite patches is one example of such a method:

$$\sigma(u,v) = [\mathcal{B}^3(u)]\,[\mathcal{P}]\,{}^t[\mathcal{B}^3(v)]\,,$$

where $[\mathcal{P}]$ the matrix of the control points contains both the control points and their derivatives. It is written as:

$$[\mathcal{P}] = \begin{bmatrix} P_{0,0} & P_{v,0,0} & P_{v,0,1} & P_{0,1} \\ P_{u,0,0} & P_{uv,0,0} & P_{uv,0,1} & P_{u,0,1} \\ P_{u,1,0} & P_{uv,1,0} & P_{uv,1,1} & P_{u,1,1} \\ P_{1,0} & P_{v,1,0} & P_{v,1,1} & P_{1,1} \end{bmatrix},$$

where, for instance,

$$P_{u,i,j} = \frac{\partial P_{i,j}}{\partial u} \quad \text{and} \quad P_{uv,i,j} = \frac{\partial^2 P_{i,j}}{\partial u \partial v}\,.$$

In this control matrix, the data relative to one control point are grouped into a 2×2 sub-matrix. Now, the coefficient matrix $[\mathcal{B}^3(u)]$ ($[\mathcal{B}^3(v)]$ following the same matrix form) is:

$$[\mathcal{B}^3(u)] = [u^3, u^2, u, 1] \begin{bmatrix} 2 & 1 & 1 & -2 \\ -3 & -2 & -1 & 3 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}.$$

Remark 13.2 *The above coefficient matrix is the cubic Hermite matrix of Chapter 12 in the case where the control point row is defined as* ${}^t[P_0, \dot{P}_0, \dot{P}_1, P_1]$ *instead of* ${}^t[P_0, P_1, \dot{P}_0, \dot{P}_1]$ *as we did above. The present notation is motivated by the notations of the* $[\mathcal{P}]$ *matrix for the surface case so as to retrieve the tensor product of the curve definition for the surface case. Also, the interpretation of the control points and derivatives as arranged in* $[\mathcal{P}]$ *is then much more intuitive.*

Transfinite interpolation (Coons patches)

The so-called Coons patches are based on transfinite interpolation. Introduced in Chapter 4 as a mesh generation method able to carry out some particular geometries, the transfinite interpolation is now seen as a method for surface definition (construction). Referring to Chapter 4, we recall the formula of Equation (4.5):

$$\begin{aligned} F(\xi,\eta) \quad = \quad & (1-\eta)\,\phi_1(\xi) + \xi\,\phi_2(\eta) + \eta\,\phi_3(\xi) + (1-\xi)\,\phi_4(\eta) \\ & - \left((1-\xi)(1-\eta)\,a_1 + \xi(1-\eta)\,a_2 + \xi\eta\,a_3 + (1-\xi)\,\eta a_4\right). \end{aligned}$$

Now, to conform to the current notations, this formula is written as:

$$\begin{aligned} \sigma(u,v) \quad = \quad & (1-v)\,\phi(u,0) + u\,\phi(1,v) + v\,\phi(u,1) + (1-u)\,\phi(0,v) \\ & - \left((1-u)(1-v)\,P_{0,0} + u(1-v)\,P_{1,0} + uv\,P_{1,1} + (1-u)\,vP_{0,1}\right). \end{aligned}$$

which can be expressed as:

$$\begin{aligned}\sigma(u,v) &= [1-u,u]\begin{pmatrix}\phi(0,v)\\ \phi(1,v)\end{pmatrix} + (\phi(u,0),\phi(u,1))\begin{bmatrix}1-v\\ v\end{bmatrix}\\ &- [1-u,u]\begin{pmatrix}\phi(0,0) & \phi(0,1)\\ \phi(1,0) & \phi(1,1)\end{pmatrix}\begin{bmatrix}1-v\\ v\end{bmatrix}. \qquad (13.5)\end{aligned}$$

The first part of this expression corresponds to a ruled surface in terms of u, the second denotes a ruled surface in terms of v, while the third part is the correction term resulting in the desired properties (see below).

Thus, we have a surface definition which is rather different from the previous ones. The patches that can be dealt with are defined via their boundaries and these boundaries can be seen as a series of four logical sides[3]. Actually, the input data consists of the boundary of the patch, i.e., ϕ. In other words, from a discrete point of view, the control points are located on the four curves defining the patch boundaries. Functions $1-u$ and u are the so-called *blended functions.*

It is easy to check, as in Chapter 4, that the corner identities are satisfied and that the surface interpolates to the four boundary curves. For instance, we have $\sigma(u,0) = \phi(u,0), \sigma(u,1) = \phi(u,1)$ and $\sigma(0,v) = \phi(0,v)$ together with $\sigma(1,v) = \phi(1,v)$.

Changing the blending functions leads to generalizing the Coons patch. If $f_1(u)$ and $f_2(u)$ along with $g_1(v)$ and $g_2(v)$ are two pairs of blending functions, then:

$$\begin{aligned}\sigma(u,v) &= [f_1(u),f_2(u)]\begin{pmatrix}\phi(0,v)\\ \phi(1,v)\end{pmatrix} + (\phi(u,0),\phi(u,1))\begin{bmatrix}g_1(v)\\ g_2(v)\end{bmatrix}\\ &- [f_1(u),f_2(u)]\begin{pmatrix}\phi(0,0) & \phi(0,1)\\ \phi(1,0) & \phi(1,1)\end{pmatrix}\begin{bmatrix}g_1(v)\\ g_2(v)\end{bmatrix}, \qquad (13.6)\end{aligned}$$

is the form of a generalized Coons patch. Note that the blending functions must enjoy certain properties to result in a consistent definition. In practice, the f_is as well as the g_is sum to one and continuity must be ensured at the corners, $f_1(0) = g_1(0) = 0$ together with $f_1(1) = g_1(1) = 1$.

An example of blending functions resulting in a powerful surface definition consists of cubic Hermite polynomials. In this case, the surface is defined by:

$$\begin{aligned}\sigma(u,v) &= [H_0^3(u),H_1^3(u),H_2^3(u),H_3^3(u)]\begin{pmatrix}\phi(0,v)\\ \phi_u(0,v)\\ \phi_u(1,v)\\ \phi(1,v)\end{pmatrix}\\ &+ (\phi(u,0),\phi_v(u,0),\phi_v(u,1),\phi(u,1))\begin{bmatrix}H_0^3(v)\\ H_1^3(v)\\ H_2^3(v)\\ H_3^3(v)\end{bmatrix} \qquad (13.7)\end{aligned}$$

[3]Following the transfinite interpolation for a triangle of Chapter 4, it is possible to develop triangular patches as well.

$$- \quad [\mathcal{H}^3(u)] \begin{pmatrix} \phi(0,0) & \phi_v(0,0) & \phi_v(0,1) & \phi(0,1) \\ \phi_u(0,0) & \phi_{uv}(0,0) & \phi_{uv}(1,0) & \phi_u(1,0) \\ \phi_u(1,0) & \phi_{uv}(1,0) & \phi_{uv}(1,1) & \phi_u(1,1) \\ \phi(1,0) & \phi_v(1,0) & \phi_v(1,1) & \phi(1,1) \end{pmatrix} {}^t[\mathcal{H}^3(v)] \,,$$

where the H_i^3 are the cubic Hermite polynomials of Section 12.2 and $\mathcal{H}^3$ is the corresponding line.

In fact, depending on the input data, using the previous patch definition requires a certain technique. When only the curves ϕ are supplied, quantities involving the necessary derivatives must be approached. When the ϕ are given along with the directional derivatives, the uv-derivatives must be evaluated. Various solutions may then be used, leading to various controls of the thus-defined surface.

13.3 Tensor product and control polyhedron

In this section, we discuss the patches that are widely used in CAD software packages. These make it possible to define surfaces with an arbitrary shape (geometry).

Control polyhedron

Let $P_{i,j}$ be a quadrilateral lattice of points. By analogy with the case of curves based on a control polygon, we can introduce, for a given surface Σ, the following definition:

$$\sigma(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{n} \Phi_{i,j}(u,v) P_{i,j} \,. \tag{13.8}$$

The choice of the basis functions $\Phi_{i,j}$ is left to the user. However, some characteristics are usually imposed in order to relate the surface to its control polyhedron (the $P_{i,j}$s).

Tensor product. The first assumption is that the surface can be written as a tensor product. In order to ensure this property, the contribution of u and that of v must be separated in the definition of $\Phi_{i,j}(u,v)$. So, we have:

$$\Phi_{i,j}(u,v) = \phi_i(u)\, \psi_j(v) \,.$$

The main interest of such a definition for $\Phi_{i,j}$ is that Equation (13.8) becomes:

$$\sigma(u,v) = \sum_{i=0}^{m} \phi_i(u) \underbrace{\left(\sum_{j=0}^{n} \psi_j(v) P_{i,j} \right)}_{S_i(v) = \gamma_1(v)} = \sum_{i=0}^{m} \phi_i(u)\, S_i(v) \,. \tag{13.9}$$

Thus, a current point on the surface is defined by the tensor product of two curves based on a definition by a control polygon. The points $S_i(v)$ can be obtained as current points of the curve $\gamma_1(v)$ related to the control polygon $(P_{i,j}(v))_{j \in [0,m]}$ and, these points being calculated, any point $\sigma(u,v)$ is calculated as the current point of a new curve related to the control polygon $(S_i(v))_{i \in [0,n]}$.

Characteristics of $\phi_i(u)$ and $\psi_j(v)$ inherited from the curve definition. To be consistent with the definition of curves, the surfaces based on a control polyhedron generally use the same basis functions for $\phi_i(u)$ and for $\psi_j(v)$ as the curves. Hence, if we use Bernstein polynomials, we define Bézier patches and if we take Splines functions, we get B-Splines type patches. By adding an homogenous coordinate to each control point of the polyhedron, we find rational patches, i.e., rational Bézier or B-Splines patches, commonly called *NURBS* when the associated sequence of nodes is non-uniform. Therefore, it should be borne in mind that the features of patches inherit the characteristics of the underlying curves.

Cauchy identity. Let us look at the value of $\mathcal{S} = \sum_{i=0}^{m} \sum_{j=0}^{n} \Phi_{i,j}(u,v)$. We have:

$$\mathcal{S} = \sum_{i=0}^{m} \sum_{j=0}^{n} \phi_i(u)\psi_j(v) = \sum_{i=0}^{m} \phi_i(u) \underbrace{\sum_{j=0}^{n} \psi_j(v)}_{\equiv 1} = \underbrace{\sum_{i=0}^{m} \phi_i(u)}_{\equiv 1} \equiv 1\,.$$

Hence, the Cauchy identity is also true in this type of surface definition. Thus, we have:

$$\sigma(u,v) = \frac{\sum_{i=0}^{m} \sum_{j=0}^{n} \Phi_{i,j}(u,v) P_{i,j}}{\sum_{i=0}^{m} \sum_{j=0}^{n} \Phi_{i,j}(u,v)}\,.$$

The point of parameters (u,v) is the barycenter of the system of $(n+1) \times (m+1)$ points $P_{i,j}$ associated with the weights $\Phi_{i,j}(u,v)$.

Remark 13.3 *As for the curves, the Cauchy identity ensures that any linear transformation of the characteristic lattice produces the same transformation onto the surface.*

Positive functions. As the functions $\phi_i(u)$ and $\psi_j(v)$ are positive functions, the functions $\Phi_{i,j}(u,v)$ are positive in the domain of definition of the (u,v)s.

If we assume the Cauchy identity to be verified, the current point $\sigma(u,v)$ is the barycenter of the points $P_{i,j}$ weighted with positive values. Thus, any convex volume enclosing the set of points $P_{i,j}$ encloses the whole patch.

Remark 13.4 *An application of this feature consists of taking as the bounding box of the patch the box defined via the minima and the maxima of the coordinates of the points of the control polyhedron.*

Relations at the endpoints. Let $[u_{min}, u_{max}] \times [v_{min}, v_{max}]$ be the interval of variation of $u \times v$. If we assume that ϕ_i and ψ_j verify the relation of Section 12.4 related to the curves, it is easy to show that:

$$\Phi_{0,0}(u_{min}, v_{min}) = 1 \;;\; \Phi_{n,0}(u_{max}, v_{min}) = 1\,,$$

$$\Phi_{n,m}(u_{max}, v_{max}) = 1 \;;\; \Phi_{0,m}(u_{min}, v_{max}) = 1\,.$$

Hence, the four corners of the polyhedron define the four points of the patch.

Boundary curves. Let us place $v = v_{min}$ in Equation (13.9); the equation of a curve governed by the parameter u is thus:

$$\sigma(u, v_{min}) = \sum_{i=0}^{m} \phi_i(u) S_i(v_{min}) = \sum_{i=0}^{m} \phi_i(u) P_{i,0} \,.$$

Hence, the boundaries of the control polyhedron define the control polygons of the curves bounding the patch.

Tangent planes at the corners. As the boundary curves are defined by the boundary of the control polyhedron, we can obtain two particular tangents at any corner by taking the tangents to the endpoints of the boundary curves. These tangents are supported by the segments of the control polyhedron sharing the given corner.

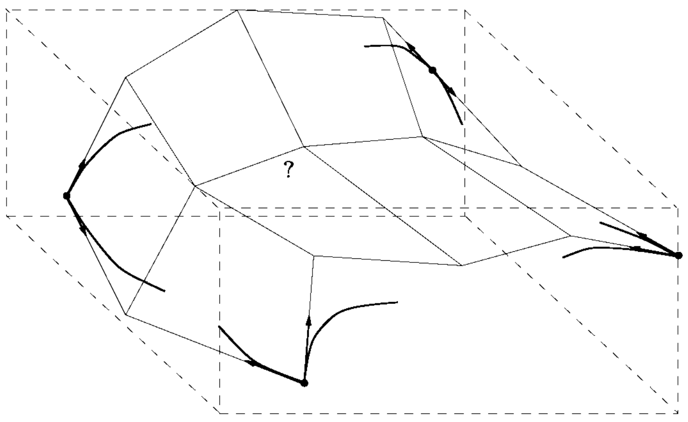


Figure 13.2: *One patch, its control polyhedron and the tangent planes at the corners.*

Figure 13.2 shows a control polyhedron. The corners of this polyhedron are particular points of this patch. The tangent planes at these corners are defined by the segments incident to these points. Within the parametric space, we can only affirm that the surface "is similar to" its control polyhedron and that the patch is enclosed in any box enclosing this control polyhedron.

Bézier quads

Bézier quad patches are constructed using Bernstein polynomials as ϕ_i and ψ_j. As a consequence, we have:

$$\sigma(u, v) = \sum_{i=0}^{n} \sum_{j=0}^{m} B_{i,n}(u)\, B_{j,m}(v)\, P_{i,j} \,, \qquad (13.10)$$

with $B_{i,n}(u)$ (resp. $B_{j,m}(v)$) the classical Bernstein polynomial, i.e., for example,

$$B_{i,n}(u) = C_n^i u^i (1-u)^{n-i} \,.$$

Notice that m can be different to n. The degree of the patch is not necessarily the same in the two directions u and v.

On the other hand, the data of a lattice of control points $P_{i,j}$ whose structure is a quadrilateral grid ($(n+1) \times (m+1)$ control points) makes it possible to define a surface via Equation (13.10).

Properties already seen in Chapter 12 about Bézier curves (and Bernstein polynomials) apply here in Bézier quadrilateral patches.

As for the curves, various recursions allow us to compute different quantities of interest, for instance, for point evaluations, for derivative computations, for degree elevations, and many others. In this way, it will be possible to find the conditions that ensure some degree of continuity when composite surfaces are discussed (see below).

Also the De Casteljau algorithm extends to the surface case when[4] $n = m$. The curve algorithm:

$$D_i^r(t) = (1-t)\, D_i^{r-1}(t) + t\, D_{i+1}^{r-1}(t)\,,$$

is replaced by:

$$D_{i,j}^{r,r}(u,v) = [1-u, u] \begin{pmatrix} D_{i,j}^{r-1,r-1}(u,v) & D_{i,j+1}^{r-1,r-1}(u,v) \\ D_{i+1,j}^{r-1,r-1}(u,v) & D_{i+1,j+1}^{r-1,r-1}(u,v) \end{pmatrix} \begin{bmatrix} 1-v \\ v \end{bmatrix}$$

for $r = 1, ..., n$ and i, j in $[0, n-r]$. This algorithm is initialized by $D_{i,j}^{0,0} = P_{i,j}$ which is, for consistency, noted by $D_{i,j}^{0,0}(u,v)$ while, obviously u and v do not appear in the D at the initialization step. The surface is then equivalently written as:

$$\sigma(u,v) = D_{0,0}^{n,n}(u,v)\,.$$

As previously indicated, a Bézier patch can be formulated in terms of a matrix form. Indeed, we return to Relation (13.3), i.e.,

$$\sigma(u,v) = [\mathcal{B}^n(u)]\,[\mathcal{P}]\,{}^t[\mathcal{B}^m(v)] \quad \text{or} \quad [\mathcal{U}][\mathcal{M}][\mathcal{P}]\,{}^t[\mathcal{N}]\,{}^t[\mathcal{V}]$$

where

$$[\mathcal{U}] = [u^n, u^{n-1}, ..., u^2, u, 1]$$

$$[\mathcal{M}] = [M_{i,j}] \quad \text{with} \quad M_{i,j} = (-1)^{n-i-j}\, C_n^i\, C_{n-i}^j$$

and similar expressions for both $[\mathcal{V}]$ and $[\mathcal{N}]$.

Then, for $n = 3$ we return to the matrix of Chapter 12, i.e.,

$$[\mathcal{M}] = \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}.$$

[4]If $n \neq m$, the De Casteljau algorithm is more subtle.

Degree elevation. As for a Bézier curve (see Chapter 12), degree elevation of a Bézier patch is useful for various purposes. In practice, curve degree elevation can be used while the degree is elevated in the u-direction and, this task being completed, it is elevated in the v-direction. Then, following the technique for a Bézier curve, we first fix j and m and we seek to obtain a $(n+1, m)$ patch (starting from a (n, m) patch). The surface:

$$\sigma(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} B_{i,n}(u)\, B_{j,m}(v)\, P_{i,j}$$

is seen as:

$$\sigma(u,v) = \sum_{j=0}^{m} \left(\sum_{i=0}^{n} B_{i,n}(u)\, P_{i,j} \right) B_{j,m}(v)$$

and, with

$$Q^*_{i,j} = \frac{i\, P_{i-1,j} \;+\; (n+1-i)\, P_{i,j}}{n+1}\,,$$

we obtain:

$$\sigma(u,v) = \sum_{j=0}^{m} \sum_{i=0}^{n+1} B_{i,n+1}(u) B_{j,m}(v)\, Q^*_{i,j}\,,$$

applying the same in terms of m, we obtain the desired result, i.e.,

$$\sigma(u,v) = \sum_{i=0}^{n+1} \sum_{j=0}^{m+1} B_{i,n+1}(u) B_{j,m+1}(v)\, Q_{i,j}\,,$$

where, now, $Q_{i,j} = \dfrac{j\, Q^*_{i,j-1} \;+\; (m+1-j)\, Q^*_{i,j}}{m+1}$, thus, in terms of the $P_{i,j}$s:

$$Q_{i,j} = \frac{1}{n+1} \frac{1}{m+1} [i, n+1-i] \begin{pmatrix} P_{i-1,j-1} & P_{i,j-1} \\ P_{i-1,j} & P_{i,j} \end{pmatrix} \begin{bmatrix} j \\ m+1-j \end{bmatrix}.$$

B-spline patches

These patches take a form similar to Relation (13.10) in which the polynomials are replaced by the B-spline functions (Chapter 12). The surface is then defined by:

$$\sigma(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} N_{i,n}(u)\, N_{j,m}(v)\, P_{i,j}\,. \tag{13.11}$$

Note that rational B-splines patches also exist.

13.4 Triangular patches and Bézier triangles

As mentioned in the introduction, there also exist triangular patches, and we now turn our attention to these.

Tri-parametric forms

Triangular patches are most commonly defined using an expression with three parameters, u, v and w: $\sigma(u, v, w)$. The values (u, v, w) are the barycentric coordinates. Hence, we have the relation:

$$u + v + w = 1 \qquad (u, v, w) \in \mathbb{R}^3_+ .$$

Thus, the whole triangle is described by the point P corresponding to the barycentric coordinates:

$$P = u\,P_0 + v\,P_1 + w\,P_2 .$$

From this point of view, the function σ can be seen as a deforming function which, applied to a predefined planar triangle (P_0, P_1, P_2), gives a curved triangle.

Bézier triangles

Bézier patches play an important role in surface definition. Two types of Bézier patches are commonly used. As may be imagined, the first type, Bézier triangles, were introduced before the second, namely Bézier quadrilateral patches. For the sake of simplicity, we discussed the quad case before the triangle case since a Bézier quad is a tensor product, thus allowing a "simple" discussion. Now, we consider Bézier triangles.

Given a set of control points that form, in terms of topology, a triangular network, we can define a triangular surface using the relationship:

$$\sigma(u, v, w) = \sum_{i+j+k=n} B^n_{i,j,k}(u, v, w)\, P_{i,j,k} , \tag{13.12}$$

where $B^n_{i,j,k}(u, v, w) = \dfrac{n!}{i!j!k!} u^i v^j w^k$ and u, v, w are the barycentric coordinates.

To make the meaning of the above indices clear, we give the following synthetic scheme (where $n = 3$) which corresponds to the arrangement of the control points:

$$\begin{array}{ccccccc} & & & (0,3,0) & & & \\ & & (0,2,1) & & (1,2,0) & & \\ & (0,1,2) & & (1,1,1) & & (2,1,0) & \\ (0,0,3) & & (1,0,2) & & (2,0,1) & & (3,0,0) \end{array}$$

Also, following the same index arrangement, the Bernstein polynomials, for $n = 3$ are as follows:

$$\begin{array}{ccccccc} & & & v^3 & & & \\ & & 3\,v^2\,w & & 3\,u\,v^2 & & \\ & 3\,v\,w^2 & & 6\,u\,v\,w & & 3\,u^2\,v & \\ w^3 & & 3\,u\,w^2 & & 3\,u^2\,w & & u^3 \end{array}$$

The Bernstein polynomials of a Bézier triangle are related by the recursion:

$$B^n_{i,j,k}(u,v,w) = uB^{n-1}_{i-1,j,k}(u,v,w) + vB^{n-1}_{i,j-1,k}(u,v,w) + wB^{n-1}_{i,j,k-1}(u,v,w)\,,$$

and a De Casteljau algorithm can be found, based on this recursion. First, we define:

$$D^0_{i,j,k} = P_{i,j,k}\,,$$

then, for $r \in [1,n]$ with $i+j+k = n-r$, the recursion is:

$$D^r_{i,j,k}(u,v,w) = uD^{r-1}_{i+1,j,k}(u,v,w) + vD^{r-1}_{i,j+1,k}(u,v,w) + wD^{r-1}_{i,j,k+1}(u,v,w)\,,$$

and

$$\sigma(u,v,w) = D^n_{0,0,0}(u,v,w)\,.$$

The surface passes through the three corners, its boundary consisting of the three Bézier curves corresponding to $u = 0$, $v = 0$ and $w = 0$. Another interesting property is the following. The tangent planes at the corners are based on the three triangles with such a point as vertex. For example, the tangent plane at corner $P_{n,0,0}$ is the plane of the triangle $P_{n,0,0}, P_{n-1,1,0}, P_{n-1,0,1}$. This is due to the fact that two derivatives are automatically known at the corners. For instance, at corner $P_{n,0,0}$, these two derivatives are the vectors $\overrightarrow{P_{n,0,0}P_{n-1,1,0}}$ and $\overrightarrow{P_{n,0,0}P_{n-1,0,1}}$.

In terms of derivatives, it is easy to find the derivatives of the above Bernstein polynomials. We have,

$$\frac{\partial}{\partial u}B^n_{i,j,k}(u,v,w) = n\,B^{n-1}_{i,j,k}(u,v,w)\,,$$

and similar expressions for the other partials. Now, for a Bézier triangle, the derivatives we are interested in are not the partials but some *directional derivatives.* Thus, given two points A and B (with barycentric coordinates u_A, v_A, w_A and u_B, v_B, w_B), we consider vector $AB = \overrightarrow{AB}$ whose components $u_{AB} = u_B - u_A$, etc. (which sum to zero). This vector enables us to write the directional derivative of the surface at a given point. We have:

$$\frac{d}{d_{AB}}\sigma(u,v,w) = u_{AB}\,\frac{\partial}{\partial u}\sigma(u,v,w) + v_{AB}\,\frac{\partial}{\partial v}\sigma(u,v,w) + w_{AB}\,\frac{\partial}{\partial w}\sigma(u,v,w)\,,$$

or, by means of the Bernstein polynomials:

$$n \sum_{i+j+k=n-1} B^{n-1}_{i,j,k}(u,v,w)\,(u_{AB}\,P_{i+1,j,k} + v_{AB}\,P_{i,j+1,k} + w_{AB}\,P_{i,j,k+1})\,. \tag{13.13}$$

We can then express the tangent plane anywhere (and not only at the corners) in a triangle in terms of the above definition (a derivative) as well as by means of the De Casteljau algorithm. To this end, Equation (13.13) is split into three parts:

$$\begin{aligned}\frac{d}{d_{AB}}\sigma(u,v,w) &= n\left(u_{AB}\sum_{i+j+k=n-1} B^{n-1}_{i,j,k}(u,v,w)\,P_{i+1,j,k}\right.\\ &+ v_{AB}\sum_{i+j+k=n-1} B^{n-1}_{i,j,k}(u,v,w)\,P_{i,j+1,k}\\ &+ \left. w_{AB}\sum_{i+j+k=n-1} B^{n-1}_{i,j,k}(u,v,w)\,P_{i,j,k+1}\right), \qquad (13.14)\end{aligned}$$

then, we have:

$$\frac{d}{d_{AB}}\sigma(u,v,w) = n\left(u_{AB}\,D^{n-1}_{1,0,0} + v_{AB}\,D^{n-1}_{0,1,0} + w_{AB}\,D^{n-1}_{0,0,1}\right),$$

and, as a conclusion, the tangent plane is the plane passing through the points $D^{n-1}_{1,0,0}$, $D^{n-1}_{0,1,0}$ and $D^{n-1}_{0,0,1}$ resulting from the De Casteljau algorithm.

With this material, it will be possible to find the conditions relating to the control points in order to obtain some continuity from patch to patch in the case of a composite surface.

Degree elevation. As Bézier triangles are not tensor product patches, degree elevation of such a patch does not take the same form as for a quad patch. Thus, we explicitly discuss how to elevate the degree of a Bézier triangle. We want to write the surface:

$$\sigma(u,v,w) = \sum_{i+j+k=n} B^{n}_{i,j,k}(u,v,w)\,P_{i,j,k},$$

in the following form:

$$\sigma(u,v,w) = \sum_{i+j+k=n+1} B^{n+1}_{i,j,k}(u,v,w)\,Q_{i,j,k},$$

and the problem is to find the $Q_{i,j,k}$s. The solution is then:

$$Q_{i,j,k} = \frac{1}{n+1}\left(i\,P_{i-1,j,k} + j\,P_{i,j-1,k} + k\,P_{i,j,k-1}\right).$$

Proof. Let us consider the Bernstein polynomial $B^{n+1}_{i+1,j,k}(u,v,w)$. Using the definition of such a polynomial, we have:

$$B^{n+1}_{i+1,j,k}(u,v,w) = \frac{(n+1)!}{(i+1)!j!k!}u^{i+1}v^{j}w^{k} \quad \text{then}$$

$$B^{n+1}_{i+1,j,k}(u,v,w) = u\frac{n+1}{i+1}\frac{n!}{i!j!k!}u^{i}v^{j}w^{k} = u\frac{n+1}{i+1}B^{n}_{i,j,k}(u,v,w).$$

Similarly (omitting the parameters),

$$B^{n+1}_{i,j+1,k} = v\frac{n+1}{j+1}B^n_{i,j,k} \quad \text{and} \quad B^{n+1}_{i,j,k+1} = w\frac{n+1}{k+1}B^n_{i,j,k} \,.$$

Conversely, we have:

$$u\,B^n_{i,j,k} = \frac{i+1}{n+1}B^{n+1}_{i+1,j,k}\,, \quad v\,B^n_{i,j,k} = \frac{j+1}{n+1}B^{n+1}_{i,j+1,k}\,, \quad w\,B^n_{i,j,k} = \frac{k+1}{n+1}B^{n+1}_{i,j,k+1}\,,$$

or, after summation,

$$B^n_{i,j,k} = \frac{1}{n+1}\left((i+1)\,B^{n+1}_{i+1,j,k} + (j+1)\,B^{n+1}_{i,j+1,k} + (k+1)\,B^{n+1}_{i,j,k+1}\right)\,,$$

since $u+v+w=1$. Now the initial patch definition is replaced by:

$$\sigma(.) = \sum_{i+j+k=n} \frac{1}{n+1}\left((i+1)\,B^{n+1}_{i+1,j,k} + (j+1)\,B^{n+1}_{i,j+1,k} + (k+1)\,B^{n+1}_{i,j,k+1}\right)P_{i,j,k}\,,$$

which, in turn, leads to:

$$\sigma(u,v,w) = \sum_{i+j+k=n+1} \frac{1}{n+1}B^{n+1}_{i,j,k}\left(i\,P_{i-1,j,k} + j\,P_{i,j-1,k} + k\,P_{i,j,k-1}\right)\,,$$

thus completing the proof. □

13.5 Other types of patches

Here we examine other types of patches, rational patches, rational Bézier patches and patches based on an arbitrary polyhedron.

Rational patches

The same approach as that used to define rational curves (Section 12.7) can be adopted in the case of surfaces. Thus, the construction of rational models comes down to adding an homogenous coordinate to each point of the control polyhedron. We can thus easily obtain rational Bézier quadrilaterals or triangles, NURBS-type patches, etc. We will briefly describe the case of rational Bézier patches in order to illustrate such patches.

• Rational quad Bézier patches

For a quad patch, we have:

$$\sigma(u,v) = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m}\omega_{i,j}\,B_{i,n}(u)\,B_{j,m}(v)\,P_{i,j}}{\sum_{i=0}^{n}\sum_{j=0}^{m}\omega_{i,j}\,B_{i,n}(u)\,B_{j,m}(v)}\,.$$

• Rational Bézier triangles

For a triangular patch, we find:

$$\sigma(u,v,w) = \frac{\sum_{i+j+k=n} \omega_{i,j,k}\, B^n_{i,j,k}(u,v,w)\, P_{i,j,k}}{\sum_{i+j+k=n} \omega_{i,j,k}\, B^n_{i,j,k}(u,v,w)} \,.$$

Remark 13.5 *Note that rational patches are not tensor product based patches.*

Patches based on an arbitrary polyhedron

Various types of patches exist which have not been discussed so far. We will limit ourselves here to giving a few examples of such patches.

Patches can be defined which are arbitrary polygons. The patches we have discussed so far are quadrilateral (or at least with four sides) or triangular (Bézier triangles). Some geometries are not well determined if we restrict ourselves to these patterns. Thus patches with arbitrary topology can be introduced and methods for handling these cases must be defined.

A popular patch is the so-called Clough-Tocher patch whose name is familiar to finite element people. Indeed, the Clough-Tocher finite element is a triangle with 12 degrees of freedom (Chapter 20). The node value and the two derivatives at the triangle vertices and a normal derivative at the edge midpoints. This finite element is C^1 and its construction is based on the three sub-triangles that are defined using its centroid. In terms of a surface definition, this triangle is seen as a patch with 12 degrees that allows for a C^1 continuity.

Another patch of interest is the Gregory patch . This patch is a triangle and the data consists of the vertices and the normals at these vertices. Also, a patch, proposed by Walton, [Walton, Meek-1996], is constructed with similar data. In a later section, we will return to patches which allow for a G^1 continuity.

We also frequently find patches having a particular shape. In fact, if we take the case of a four-sided patch, it is not always possible to construct a given surface using only this type of patch. Indeed, the number of patches incident to one point is in principle 4. If for any reason, this configuration is not possible, we have to aim at a different number of incident patches. Hence, three- or five-sided patches have been introduced. To reduce the number of sides of a patch, we can degenerate it (i.e., force two vertices to be coincident). On the other hand, increasing the number of sides is more tedious (notice however that the Coons patch makes it possible to envisage this case quite easily).

Finally, notice that restraining the usable domain of a patch makes it possible, while preserving a limited number of patches, to introduce arbitrary boundaries (other than the "natural" borders of the patch) and holes (which could also be obtained by subdividing the initial patch); Figure 13.3.

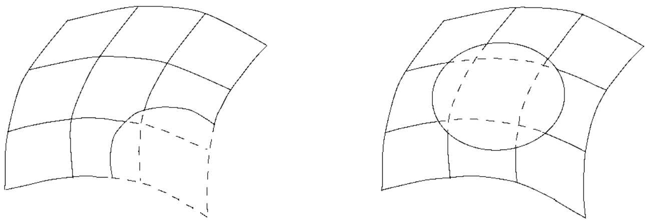


Figure 13.3: *Examples of restricted patches. Left-hand side: the boundary of interest is not the patch boundary. Right-hand side: a hole on the surface.*

13.6 Composite surfaces

As for composite curves, composite surfaces is a definition method that allows for a great flexibility in the case of arbitrary surfaces. Figure 13.4 depicts an example of a surface whose definition requires a certain number of patches[5]. Actually, the given surface is split into a series of patches which conform to one of the above patch nature.

Figure 13.4: *Example of a complex surface using a set of composite patches (view data courtesy by Dassault Systèmes, modeler CATIA).*

Given two (quad) Bézier patches, which are members of a composite surface, the aim is to define the control points in such a way as to insure some degree of continuity from one patch to the other, thus resulting in a global continuity when the entire surface is considered.

The key is then to have a continuity at the patch interfaces, i.e., at the curve level. This continuity concerns the adequate derivatives of the curves. First, we need to have the values of the derivatives of a given Bézier patch. By introducing

[5]In fact, this model certainly also includes some restricted patches and not only some natural boundaries (Chapter 15).

the notation:

$$\Delta_u P_{i,j} = P_{i+1,j} - P_{i,j} \quad \text{and} \quad \Delta_v P_{i,j} = P_{i,j+1} - P_{i,j}\,,$$

then, the formula:

$$\Delta^{rs} P_{i,j} = \Delta_u^r(\Delta_v^s P_{i,j}) = \Delta_v^s(\Delta_u^r P_{i,j})\,,$$

allows for the calculation of the derivatives of the surface $\sigma(u,v)$. We have:

$$\frac{\partial^r \partial^s}{\partial u^r \partial v^s}\sigma(u,v) = \frac{n!}{(n-r)!}\frac{m!}{(m-s)!}\sum_{i=0}^{n-r}\sum_{j=0}^{m-s} B_{i,n-r}(u)\,B_{j,m-s}(v)\,\Delta^{rs}P_{i,j}\,.$$

Now, let us consider two such patches, $\sigma_1(u_1,v_1)$ and $\sigma_2(u_2,v_2)$, with the same degrees[6] n and m and whose interface consists of:

$$\sigma_1(1,v_1) \quad \text{and} \quad \sigma_2(0,v_2)\,,$$

the curves corresponding to $u_1 = 1$ and $u_2 = 0$ respectively.

A C^0 continuity from σ_1 and σ_2 leads to having:

$$\sum_{j=0}^{m} B_{j,m}(v_1)\,P_{n,j} = \sum_{j=0}^{m} B_{j,m}(v_2)\,Q_{0,j} \quad \forall v_1 = v_2 \in [0,1]$$

where the $P_{i,j}$ are the control points of σ_1 while the control points of σ_2 are the $Q_{i,j}$s. Then, in terms of control points, this continuity is achieved if:

$$P_{n,j} = Q_{0,j} \quad \forall j\,.$$

A C^1 continuity is achieved if the first derivatives match at the interface. Using the above formula about the derivatives with respectively $r=1, s=0$ and $r=0, s=1$, we have successively:

$$\frac{\partial}{\partial u}\sigma_1(1,v_1) = n\sum_{j=0}^{m} B_{j,m}(v_1)\,(P_{n,j} - P_{n-1,j})$$

$$\frac{\partial}{\partial u}\sigma_2(0,v_2) = n\sum_{j=0}^{m} B_{j,m}(v_2)\,(Q_{1,j} - Q_{0,j})$$

and

$$\frac{\partial}{\partial v}\sigma_1(1,v_1) = m\sum_{j=0}^{m-1} B_{j,m-1}(v_1)\,(P_{n,j+1} - P_{n,j})$$

$$\frac{\partial}{\partial v}\sigma_2(0,v_2) = m\sum_{j=0}^{m-1} B_{j,m-1}(v_2)\,(Q_{0,j+1} - Q_{0,j})$$

[6] Actually, the same degree is only necessary at the interface (i.e., for m).

Then, we need to have: $\frac{\partial}{\partial u}\sigma_1(1,v_1) = \frac{\partial}{\partial u}\sigma_2(0,v_2)$ together with $\frac{\partial}{\partial v}\sigma_1(1,v_1) = \frac{\partial}{\partial v}\sigma_2(0,v_2)$ for $v_1 = v_2 \in [0,1]$. Since a C^0 continuity holds, the v-derivatives match while a condition insuring that the u-derivatives match is $P_{n,j} - P_{n-1,j} = Q_{1,j} - Q_{0,j}$ or, since $Q_{0,j} = P_{n,j}$,

$$P_{n-1,j} + Q_{1,j} = 2\,P_{n,j}\,.$$

In other words, $P_{n,j}$, $P_{n-1,j}$ and $Q_{1,j}$ are aligned[7] and, moreover $P_{n,j}$ must be the midpoint of segment $P_{n-1,j}, Q_{1,j}$.

Similarly a high order continuity can be obtained since additional conditions about the control points are enforced. For instance, a C^2 continuity also leads to having:

$$Q_{2,j} - P_{n-2,j} = 2\,(Q_{1,j} - P_{n-1,j})\,.$$

Remark 13.6 *When the parameters are not in* $[0,1]$, *a variable change must be made to return to the above discussion.*

Since a C-type continuity is very demanding, thus impeding flexibility when defining the control points, the use of a G-type continuity must be discussed. For instance, the G^1 continuity at an interface between two patches means that the two tangent planes are continuous along the interface curve.

The tangent plane for surface $\sigma_1(u_1,v_1)$, along the curve $u_1 = 1$ (we pursue the above example), is a combination such as:

$$\alpha_1 \frac{\partial}{\partial u}\sigma_1(1,v_1) + \beta_1 \frac{\partial}{\partial v}\sigma_1(1,v_1)\,,$$

that for $\sigma_2(u_2,v_2)$, along the curve $u_2 = 0$, is:

$$\alpha_2 \frac{\partial}{\partial u}\sigma_2(0,v_2) + \beta_2 \frac{\partial}{\partial v}\sigma_2(0,v_2)\,,$$

where the αs and the βs are some non-zero coefficients (indeed, some functions of the vs). The relation which is needed is then:

$$\alpha_1 \frac{\partial}{\partial u}\sigma_1(1,v_1) + \beta_1 \frac{\partial}{\partial v}\sigma_1(1,v_1) - \alpha_2 \frac{\partial}{\partial u}\sigma_2(0,v_2) - \beta_2 \frac{\partial}{\partial v}\sigma_2(0,v_2) = 0\,,$$

thus, due to the C^0 continuity, this relation reduces to:

$$\alpha \frac{\partial}{\partial u}\sigma_1(1,v_1) + \beta \frac{\partial}{\partial v}\sigma_1(1,v_1) + \gamma \frac{\partial}{\partial u}\sigma_2(0,v_2) = 0\,,$$

where $\alpha = \alpha_1, \beta = \beta_1 - \beta_2$ and $\gamma = -\alpha_2$.

The "simplest" choice is to consider constant values for these functions and to take $\beta = 0$, meaning that $\beta_1 = \beta_2$. Then, the condition for a G^1 continuity reduces to:

$$\alpha \frac{\partial}{\partial u}\sigma_1(1,v_1) + \gamma \frac{\partial}{\partial u}\sigma_2(0,v_2) = 0$$

[7]Thus, only a G^1 continuity is achieved.

and, in terms of the control points,

$$\alpha\, n \sum_{j=0}^{m} B_{j,m}(v_1)\,(P_{n,j} - P_{n-1,j}) \;+\; \gamma\, n \sum_{j=0}^{m} B_{j,m}(v_2)\,(Q_{1,j} - Q_{0,j})\,,$$

thus, since $v_1 = v_2$, we return to the above condition about the alignment of $P_{n,j}$, $P_{n-1,j}$ and $Q_{1,j}$.

Now, we consider $\beta \neq 0$ and more precisely a function such as $\beta = (1-v)\beta + v$. Then, we must have:

$$\alpha \frac{\partial}{\partial u}\sigma_1(1,v_1) \;+\; ((1-v)\beta+v)\,\frac{\partial}{\partial v}\sigma_1(1,v_1) \;+\; \gamma\,\frac{\partial}{\partial u}\sigma_2(0,v_2) \;=0\,.$$

In terms of control points, this leads to having:

$$\alpha\, n \sum_{j=0}^{m} B_{j,m}(v_1)\,(P_{n,j} - P_{n-1,j}) +$$

$$((1-v)\beta+v)\, m \sum_{j=0}^{m-1} B_{j,m-1}(v_1)\,(P_{n,j+1} - P_{n,j}) + \gamma\, n \sum_{j=0}^{m} B_{j,m}(v_2)\,(Q_{1,j} - Q_{0,j}) = 0\,,$$

or, equivalently,

$$\alpha\, n\,(P_{n,j} - P_{n-1,j}) + \beta(m-j)\,(P_{n,j+1} - P_{n,j}) + j\,(P_{n,j} - P_{n,j-1}) + \gamma\, n\,(Q_{1,j} - Q_{0,j}) = 0\,.$$

Now, let $\alpha = \gamma$, we have:

$$\alpha\, n\,(Q_{1,j} - P_{n-1,j}) \;+\; \beta(m-j)\,(P_{n,j+1} - P_{n,j}) \;+\; j\,(P_{n,j} - P_{n,j-1}) = 0\,,$$

and, for $\beta = \frac{j}{m-j}$, this reduces again to:

$$\alpha\, n\,(Q_{1,j} - P_{n-1,j}) \;+\; (P_{n,j+1} - P_{n,j-1}) = 0\,,$$

and, finally, a sufficient condition is to have the quad $Q_{1,j}, P_{n,j+1}, P_{n-1,j}, P_{n,j-1}$ planar.

As a conclusion to this discussion, we have exhibited two different conditions for G^1 continuity. Note that other conditions may also be found. Obviously, based on the degree n and m and due to the necessity of considering the continuity for several patches at the same time the construction of G^1 surface is not trivial. Also, when the degree of the interface is not the same, this work becomes more complicated (involving, for instance, the degree elevation technique).

Composite Bézier triangles

Let us consider two Bézier triangles of the same degree n, say $\sigma_1(u_1, v_1, w_1)$ and $\sigma_2(u_2, v_2, w_2)$, which share a common interface, for instance, for $w_1 = 0$ and $w_2 = 0$. The C^0 continuity holds since $\sigma_1(u_1, v_1, 0) = \sigma_2(u_2, v_2, 0)$. If $P_{i,j,k}$ (resp.

$Q_{i,j,k}$) stand for the control points of σ_1 (resp. σ_2), the above condition leads to having:

$$\sum_{i+j+k=n} B^n_{i,j,k}(u_1, v_1, 0)\, P_{i,j,0} = \sum_{i+j+k=n} B^n_{i,j,k}(u_2, v_2, 0)\, Q_{i,j,0}\,,$$

when $u_1 = u_2$ and $v_1 = v_2$. Then, as expected, a C^0 condition is:

$$P_{i,j,0} = Q_{i,j,0} \quad \text{for} \quad i + j = n\,.$$

To discuss high order continuity, we have to return to the directional derivatives of the surface. Let us recall the corresponding formula, for $w = 0$ and patch σ_1:

$$n \sum_{i+j+k=n-1} B^{n-1}_{i,j,k}(u, v, 0)\,(u_{AB}\, P_{i+1,j,0} + v_{AB}\, P_{i,j+1,0} + w_{AB}\, P_{i,j,1})\ ,$$

where AB is the direction of derivation.

Remark 13.7 *It is clear to see that the directional derivatives along the interface match automatically. Indeed, since $w_{AB} = 0$, the C^0 continuity insures this result.*

Now, for an arbitrary direction AB, if we have:

$$n \sum_{i+j+k=n-1} B^{n-1}_{i,j,k}(u, v, 0)\,(u_{AB}\, P_{i+1,j,0} + v_{AB}\, P_{i,j+1,0} + w_{AB}\, P_{i,j,1})\ ,$$

for the AB-derivative of σ_1, then, we have:

$$-n \sum_{i+j+k=n-1} B^{n-1}_{i,j,k}(u, v, 0)\,(u_{AB}\, Q_{i+1,j,0} + v_{AB}\, Q_{i,j+1,0} + w_{AB}\, Q_{i,j,1})\ ,$$

for the AB-derivative of σ_2. Thus, the C^1 continuity holds since a relationship such as:

$$\alpha\, P_{i+1,j,0} + \beta\, P_{i,j+1,0} + \gamma\, P_{i,j,1} + \gamma\, Q_{i,j,1} = 0 \quad \text{holds}\,.$$

In other words the two triangles $P_{i+1,j,0}\, P_{i,j+1,0}\, P_{i,j,1}$ and $P_{i+1,j,0}\, P_{i,j+1,0}\, Q_{i,j,1}$ are coplanar and points $P_{i,j,1}$ and $Q_{i,j,1}$ are related one with the other (symmetry).

Note that conditions insuring a G^1 take the same aspect, we have now:

$$\alpha\, P_{i+1,j,0} + \beta\, P_{i,j+1,0} + \gamma\, P_{i,j,1} + \delta\, Q_{i,j,1} = 0\,,$$

where $\delta = k\,\gamma$.

Also, as a conclusion, we return to the final observations about Bézier quads. In particular, a degree 3 patch is too rigid and patches of at least degree 4 must be used.

Other composite surfaces

Patches other than the two above can be considered to develop composite surfaces. For instance, the Gregory and the Walton patches, as briefly introduced, can serve as support for such a construction (see the next section for a more complete discussion).

13.7 Explicit construction of a composite surface

In this section, we consider a quite different approach (as in Section 12.3). The main idea is to use as data a discrete approximation of a given surface. This approximation is indeed a surface mesh composed of triangles. Triangle vertices and normals at these vertices are assumed. Then, the construction is completed triangle by triangle using the available information (singularities, normals at the triangle vertices) is such a way as to obtain a G^1 surface.

The construction is made in two main steps. The first step concerns the construction of a Bézier curve of degree 3 based on the edges of the surface mesh by taking into account the corresponding normals. The second step, using these curve definitions, concerns the construction of the patch.

Constructing the curves boundary of a patch

An edge is characterized by its two endpoints and the two corresponding (surface) normals, thus, 12 degrees of freedom. Hence, a Bézier curve of degree 3 is sought (which has the same number of degrees). Let $\gamma(t)$ be the curve associated with a given edge. We assume that:

$$\gamma(t) = \sum_{i=0}^{3} B_{i,3} P_i \,,$$

where the control points are P_0 the first endpoint of the edge, P_3 its second endpoint and:

$$P_1 = P_0 + V_0 \quad \text{and} \quad P_2 = P_3 - V_2 \,,$$

where V_0 and V_2 must be properly defined (after which V_1 will be $V_1 = P_2 - P_1$). Due to the choice of a third degree Bézier curve, we have:

$$\gamma'(t) = 3 \sum_{i=0}^{2} B_{i,2} V_i \quad \text{and} \quad \gamma''(t) = 6 \sum_{i=0}^{1} B_{i,1} (V_{i+1} - V_i) \,.$$

Then,

$$\gamma'(t) = 3\left((1-t)^2 V_0 + 2t(1-t)\, V_1 + t^2 V_2\right) \quad \text{thus} \quad \gamma'(0) = 3\, V_0 \quad \text{and} \quad \gamma'(1) = 3\, V_1 \,.$$

$$\gamma''(t) = 6\left((1-t)(V_1 - V_0) \,+\, t(V_2 - V_1)\right. \quad \text{thus}$$

$$\gamma''(0) = 6\,(V_1 - V_0) \quad \text{and} \quad \gamma''(1) = 6\,(V_2 - V_1) \,.$$

Now, following Section 11.1, the curve's normal, $\nu(t)$, at t, is parallel to:

$$\langle \gamma'(t), \gamma'(t) \rangle \gamma''(t) - \langle \gamma'(t), \gamma''(t) \rangle \gamma'(t) \,.$$

Then, for $t = 0$, we have:

$$\vec{\nu}(0) \quad // \quad \langle \vec{V_0}, \vec{V_0} \rangle (\vec{V_1} - \vec{V_0}) - \langle \vec{V_0}, (\vec{V_1} - \vec{V_0}) \rangle \vec{V_0} \quad \text{i.e.,}$$

$$\vec{\nu}(0) \quad // \quad \langle \vec{V_0}, \vec{V_0} \rangle \vec{V_1} - \langle \vec{V_0}, \vec{V_1} \rangle \vec{V_0} \;=\; (\vec{V_0} \wedge \vec{V_1}) \wedge \vec{V_0}$$

and, for $t = 1$, a similar result holds:

$$\vec{\nu}(1) \quad // \quad \langle \vec{V_2}, \vec{V_2} \rangle \vec{V_1} - \langle \vec{V_2}, \vec{V_1} \rangle \vec{V_2} = (\vec{V_2} \wedge \vec{V_1}) \wedge \vec{V_2}\,.$$

We impose the same properties not for the curve normals, $\nu(0)$ and $\nu(1)$, but for the surface normals n_0 and n_1. Indeed, we assume that $\nu(0)$ (resp. $\nu(1)$) matches n_0 (resp. n_1). In other words,

$$n_0 \quad // \quad (V_0 \wedge V_1) \wedge V_0\,,$$

$$n_1 \quad // \quad (V_2 \wedge V_1) \wedge V_2\,,$$

thus, using the first relation, n_0, V_0 and $V_0 \wedge V_1$ form a basis. Since we assume that both n_0 and n_1 are non-zero, V_0 and V_1 are linearly independent then, n_0, V_0 and V_1 form a basis. As a consequence, three coefficients exist such that:

$$n_1 = \alpha_1\, n_0 + \beta_1\, V_0 + \gamma_1\, V_1\,,$$

and, similarly,

$$n_0 = \alpha_0\, n_1 + \beta_0\, V_1 + \gamma_0\, V_2\,.$$

These can be written in terms of P_3P_0 and V_0 and V_2 only since $V_1 = P_0P_3 - V_0 - V_2$.

$$n_1 = \alpha_1\, n_0 + \beta_1\, V_0 + \gamma_1\, (P_0P_3 - V_0 - V_2)\,,$$

$$n_0 = \alpha_0\, n_1 + \beta_0\, (P_0P_3 - V_0 - V_2) + \gamma_0\, V_2\,,$$

and, a simple calculation shows that:

$$V_0 = \alpha_1 D \; + \; \beta_1 n_0 \; + \; \gamma_1 n_1\,,$$

$$V_2 = \alpha_2 D \; + \; \beta_2 n_0 \; + \; \gamma_2 n_1\,,$$

where D stands for P_0P_3 and the above (new) coefficients must be determined. We assume that $\gamma''(0)$ is parallel to n_0, then

$$\begin{aligned} V_1 - V_0 &= D - V_2 - 2\,V_0 \\ &= (1 - \alpha_2 - 2\alpha_1)\, D \; + \; (-\beta_2 - 2\beta_1)\, n_0 \; + \; (-\gamma_2 - 2\gamma_1)\, n_1 \quad // \quad n_0 \end{aligned}$$

i.e.,

$$1 - \alpha_2 - 2\alpha_1 = 0 \quad \text{and} \quad \gamma_2 + 2\gamma_1 = 0\,.$$

Now, we assume that $\gamma''(1)$ is parallel to n_1, and we have:

$$\begin{aligned} V_2 - V_1 &= 2V_2 + V_0 - D \\ &= (-1 + \alpha_1 + 2\alpha_2)\, D \; + \; (2\beta_2 + \beta_1)\, n_0 \; + \; (2\gamma_2 + \gamma_1)\, n_1 \quad // \quad n_1 \end{aligned}$$

i.e.,

$$1 - \alpha_1 - 2\alpha_2 = 0 \quad \text{and} \quad \beta_1 + 2\beta_2 = 0\,.$$

As a first result, we have $\alpha_1 = \alpha_2 = \frac{1}{3}$. Now, we express that:

$$\langle \vec{V_0}, \vec{n_0} \rangle = \langle \vec{V_1}, \vec{n_1} \rangle = 0\,,$$

then,

$$0 = \frac{1}{3}\langle \vec{D}, \vec{n}_0\rangle + \beta_1 + \gamma_1\langle \vec{n}_0, \vec{n}_1\rangle$$

$$0 = \frac{1}{3}\langle \vec{D}, \vec{n}_1\rangle + \beta_2\langle \vec{n}_0, \vec{n}_1\rangle + \gamma_2$$

and, replacing β_1 in terms of β_2 and γ_2 in terms of γ_1, these two relations lead to:

$$0 = \frac{1}{3}\langle \vec{D}, \vec{n}_0\rangle - 2\beta_2 + \gamma_1\langle \vec{n}_0, \vec{n}_1\rangle \quad \text{and}$$

$$0 = \frac{1}{3}\langle \vec{D}, \vec{n}_1\rangle + \beta_2\langle \vec{n}_0, \vec{n}_1\rangle - 2\gamma_1 \,,$$

which enable us to find:

$$\beta_2 = \frac{1}{3}\frac{2\langle \vec{D}, \vec{n}_0\rangle + \langle \vec{n}_1, \vec{n}_0\rangle\langle \vec{D}, \vec{n}_1\rangle}{4 - \langle \vec{n}_1, \vec{n}_0\rangle^2} \quad \text{and}$$

$$\gamma_1 = \frac{1}{3}\frac{2\langle \vec{D}, \vec{n}_1\rangle + \langle \vec{n}_1, \vec{n}_0\rangle\langle \vec{D}, \vec{n}_0\rangle}{4 - \langle \vec{n}_1, \vec{n}_0\rangle^2} \,.$$

Thus, all the coefficients are determined and V_0 and V_2 are known.

$$V_0 = \frac{||D||}{18}\left(6d - 2\rho n_0 + \sigma n_1\right)$$

$$V_2 = \frac{||D||}{18}\left(6d + \rho n_0 - 2\sigma n_1\right) \,,$$

where $d = \frac{D}{||D||}$ and ρ and σ are respectively

$$\rho = 6\frac{2\langle \vec{d}, \vec{n}_0\rangle + \langle \vec{n}_1, \vec{n}_0\rangle\langle \vec{d}, \vec{n}_1\rangle}{4 - \langle \vec{n}_1, \vec{n}_0\rangle^2} \,,$$

$$\sigma = 6\frac{2\langle \vec{d}, \vec{n}_1\rangle + \langle \vec{n}_1, \vec{n}_0\rangle\langle \vec{d}, \vec{n}_0\rangle}{4 - \langle \vec{n}_1, \vec{n}_0\rangle^2} \,.$$

As a conclusion, the four control points of the Bézier curve corresponding to a given edge, say AB, are determined as:

$$\left\{\begin{array}{lcl} P_0 & = & A \\ P_3 & = & B \\ P_1 & = & P_0 + \frac{||\vec{D}||}{18}\left(6\vec{d} - 2\rho\vec{n}_0 + \sigma\vec{n}_1\right) \\ P_2 & = & P_3 - \frac{||\vec{D}||}{18}\left(6\vec{d} + \rho\vec{n}_0 - 2\sigma\vec{n}_1\right) . \end{array}\right. \tag{13.15}$$

Construction of a patch

Given a triangle (S_1, S_2, S_3) and the three normals $\vec{n}_1, \vec{n}_2$ and $\vec{n}_3$, the above construction results in three Bézier curves of degree 3, one for the edge $[S_1, S_2]$, one for $[S_2, S_3]$ and the last for $[S_3, S_1]$. To create a G^1 patch, we need to use at least degree 4 functions. Thus, we first elevate the degree of the three above curves. Following Chapter 12, the corresponding control points are, for a given curve,

$$L_i = \frac{i\,P_{i-1} + (4-i)\,P_i}{4},$$

for $i = 0, 4$. Indeed, we have $L_0 = P_0$, $L_4 = P_3$ while the other L_is conform to the above formula. By using this formula for the three edges, we obtain the 12 control points that define the patch boundaries.

Hence, we have the 12 patch boundary control points ready. The issue is now how to define the three internal control points in such a way as to ensure G^1 continuity. Formally speaking, the complete control point table associated with a patch of degree 4 is as follows:

$$\begin{array}{ccccccccc}
 & & & & P_{0,4,0} & & & & \\
 & & & P_{1,3,0} & & P_{0,3,1} & & & \\
 & & P_{2,2,0} & & P_{1,2,1} & & P_{0,2,2} & & \\
 & P_{3,1,0} & & P_{2,1,1} & & P_{1,1,2} & & P_{0,1,3} & \\
P_{4,0,0} & & P_{3,0,1} & & P_{2,0,2} & & P_{1,0,3} & & P_{0,0,4}
\end{array}$$

In this control points table, three points are missing. Indeed, in terms of the triangle vertices and edges, we have (with evident notations including these triangle vertices and the $L_{k,i}$s where $L_{k,i}$ stands for the i^{th} control points of edge k)

$$\begin{array}{ccccccccc}
 & & & & S_3 & & & & \\
 & & & L_{2,1} & & L_{1,3} & & & \\
 & & L_{2,2} & & ? & & L_{1,2} & & \\
 & L_{2,3} & & ? & & ? & & L_{1,1} & \\
S_1 & & L_{3,1} & & L_{3,2} & & L_{3,3} & & S_2
\end{array}$$

In other words, we have:

$$\begin{array}{c}
P_{4,0,0} = S_1 \quad P_{0,4,0} = S_3 \quad P_{0,0,4} = S_2\,, \\
P_{0,i,4-i} = L_{1,i} \quad P_{i,4-i,0} = L_{2,i} \quad P_{4-i,0,i} = L_{3,i}\,.
\end{array} \tag{13.16}$$

And we still need to define the control points inside the patch, i.e., the points $P_{1,1,2}$, $P_{1,2,1}$ and $P_{2,1,1}$ in order to complete the definition of $\sigma(r,s,t)$ which is given by:

$$\sigma(r,s,t) = \sum_{i+j+l=4} B^4_{i,j,l}(r,s,t) P_{i,j,l}\,. \tag{13.17}$$

This construction is rather technical and it is completed in several steps. First, we consider the Bézier curves associated with the tangent vectors of the three edges[8] as already introduced:

$$\gamma_k'(t) = 3\sum_{i=0}^{2} B_{i,2} V_{k,i} \quad \text{for} \quad k = 0, 1, 2\,,$$

with $V_{k,i} = L_{k,i+1} - L_{k,i}$. Now, based on the $V_{k,i}$s, we define:

$$A_{k,0} = n_{k+1} \wedge \frac{V_{k,0}}{||V_{k,0}||} \quad \text{and} \quad A_{k,2} = n_{k+2} \wedge \frac{V_{k,2}}{||V_{k,2}||}\,,$$

as well as:

$$A_{k,1} = \frac{A_{k,0} + A_{k,2}}{||A_{k,0} + A_{k,2}||}\,.$$

Using these coefficients, we construct the quadratic Bézier $h_k(t)$ by:

$$h_k(t) = \sum_{i=0}^{2} A_{k,i} B_{i,2}(t)\,.$$

This curve allows us to define the tangent along edge k. It can then be shown that the $\gamma_k'(t)$s and the $h_k(t)$s allow us to define a suitable continuity between the patches. We construct the values $D_{k,i}$ for $k = 1, 2, 3$ and $i = 0, 1, 2, 3$.

$$D_{1,i} = P_{1,i,3-i} - \frac{P_{0,i+1,3-i} + P_{0,i,4-i}}{2}$$

$$D_{2,i} = P_{i,3-i,1} - \frac{P_{i+1,3-i,0} + P_{i,4-i,0}}{2}$$

$$D_{3,i} = P_{3-i,1,i} - \frac{P_{3-i,0,i+1} + P_{4-i,0,i}}{2}\,.$$

Using these $D_{k,i}$s and the $\gamma_k'(t)'$s, we obtain the $\lambda_{k,i}$s, for $k = 1, 2, 3$.

$$\lambda_{k,0} = \frac{\langle \vec{D}_{k,0}, \vec{V}_{k,0} \rangle}{\langle \vec{V}_{k,0}, \vec{V}_{k,0} \rangle}\,,$$

$$\lambda_{k,1} = \frac{\langle \vec{D}_{k,3}, \vec{V}_{k,2} \rangle}{\langle \vec{V}_{k,2}, \vec{V}_{k,2} \rangle}\,.$$

Using the $D_{k,i}$s and the $h_k(t)$s, we obtain the $\mu_{k,i}$s, for $k = 1, 2, 3$.

$$\mu_{k,0} = \langle \vec{D}_{k,0}, \vec{A}_{k,0} \rangle$$

$$\mu_{k,1} = \langle \vec{D}_{k,3}, \vec{A}_{k,2} \rangle\,.$$

[8]Index k, from 1 to 3, refers to edge number k. Remember that edge k is opposite vertex k, then, in what follows, as we need to write the normals n_k corresponding to edge k, we will meet, for instance, n_4 which must be identified with n_1, etc.

From the $\lambda_{k,i}$s, the $\mu_{k,i}$s, the $L_{k,i}$s, the $A_{k,i}$s and the $V_{k,i}$s, we deduce the $G_{k,i}$s, for $k = 1,2,3$ and $i = 1,2$.

$$G_{k,1} = \frac{L_{k,1} + L_{k,2}}{2} + 2\frac{\lambda_{k,0}V_{k,1}}{3} + \frac{\lambda_{k,1}V_{k,0}}{3} + 2\frac{\mu_{k,0}A_{k,1}}{3} + \frac{\mu_{k,1}A_{k,0}}{3}$$

$$G_{k,2} = \frac{L_{k,2} + L_{k,3}}{2} + \frac{\lambda_{k,0}V_{k,2}}{3} + 2\frac{\lambda_{k,1}V_{k,1}}{3} + \frac{\mu_{k,0}A_{k,2}}{3} + 2\frac{\mu_{k,1}A_{k,1}}{3}.$$

In other words, 6 virtual control points have been defined where each pair, for k, corresponds to one edge. These points are then combined and, in this way, the desired control points may be obtained. This gives successively:

$$\begin{aligned} P_{1,1,2} &= \frac{r\,G_{3,2} + s\,G_{1,1}}{r+s}, \\ P_{1,2,1} &= \frac{t\,G_{1,2} + r\,G_{2,1}}{t+r}, \\ P_{2,1,1} &= \frac{s\,G_{2,2} + t\,G_{3,1}}{s+t}. \end{aligned} \tag{13.18}$$

The patch is then well defined via Expression (13.17) and Relationships (13.16) and (13.18). As a conclusion, the resulting surface is G^1 as proved in [Piper-1987].

Remark 13.8 *Note that the three internal control points are combinations of 6 points and depend on the barycentric coordinates. In fact, we encounter a Gregory patch.*

Remark 13.9 *The case where singularities (such as corners or ridges) exist leads to a more subtle construction where these singularities are fixed.*

★
★ ★

As for the curves, numerous definitions of surfaces exist, these being of various degrees of difficulty. The context of the application and the desired geometric properties usually make it possible to decide which representation is suitable for the foreseen at hand.

The surface meshing techniques must, as with the curves, as far as possible remain independent of the representation chosen to construct these surfaces.

Moreover, the same phenomenon of perverse numerical effects as for the curves occurs with surfaces. These must be taken into account later.

Chapter 14

Curve Meshing

Curve meshing is one of the main steps in the meshing process of planar, surface or volume domains. In fact, most of the automatic mesh generation methods in $\mathbb{R}^2$ or $\mathbb{R}^3$ build the desired covering up from the data of the boundary meshes delimiting the domain considered. In two dimensions and for surfaces, the boundary is naturally formed by a set of curves. In three dimensions, the boundary is formed by a set of surfaces whose boundaries again involve a set of curves.

As we have already seen, in terms of quality, the mesh of a domain is strictly dependent on the mesh of its boundary (Chapters 5 to 7).

From a topological point of view, a curve is *a priori* a one-dimensional entity. However, when it is a component of a higher-dimensional entity (a planar region, a surface or a boundary of a solid), it must be treated in a multi-dimensional space. Hence, any control at the level of the mesh of a domain of $\mathbb{R}^2$ or $\mathbb{R}^3$, induces a similar control at the level of the curves of this domain. For instance, a size and/or directional specification concerning the mesh elements is translated into a specification onto the curves of the domain. Here, we again encounter a governed meshing problem, which is either isotropic or anisotropic. To a control based on considerations related to the envisaged application (a finite element computation) is added a control of a purely geometric nature. The desired mesh must be a good approximation, in a sense that we will specify, of the geometry.

The curve meshing problem concerns two aspects that must be combined judiciously: a geometrical aspect and an aspect related to the envisaged application. To distinguish clearly between these two cases, we will define the notion of *geometric mesh* and indicate how to construct such a mesh. Then, we will show how to generate a *computational mesh*, which is a mesh that respects the curve geometry while also satisfying specific requirements related to the application.

★
★ ★

This chapter introduces several methods to construct the mesh of a given curve. By mesh, we mean here a piecewise linear approximation of the curve, that is a discretization of the curve with straight segments (i.e., a P^1*-type mesh*). The case of meshes with other kinds of elements (for instance, P^2*-type meshes* with parabola arcs) will be covered in Chapter 22.

For reasons that will appear obvious later (at least we hope so), we first look at how to mesh a (straight) segment. In this particular case, any mesh is geometric (i.e., the geometry is perfectly approached at any point). In other words, there is no geometric-type problem. We will then discuss the case of a parametric curve and that of a curve defined by a discretization. Following this discussion, we will show how to construct a discrete representation of a curve. This being done, a given curve can be replaced by a set of straight segments and the initial meshing technique can then be used, allowing for some slight modifications, to construct the desired mesh. Finally, to conclude this chapter, we will briefly deal with the case of curves in $\mathbb{R}^3$, which are "dangling" or supported by a known surface.

Before going more deeply into the subject, let us mention, perhaps surprisingly, that the (too) rare references on this topic are not prolix, especially concerning the application related to finite elements or numerical computations[1].

14.1 Meshing a segment

In this section, we start with the, *a priori* trivial, problem of meshing a straight segment. We then look at the problem of meshing a curve and we show that several similarities exist between these two problems.

Classical segment meshing

Meshing (discretizing) a segment consists of subdividing it into a series of sub-segments of suitable lengths. In practice, these lengths depend on the objectives aimed at and the data (metric specifications) available.

Minimal mesh specifications. In this case, the information provided is relatively simple. The user indicates explicitly what he would like to obtain, for instance:

- a given number of subdivisions (assumed to be of equal size),
- a given length (i.e., a size or a step) for each mesh element,
- an element-size variation along the segment, etc.

According to the given requirements, the aim is to find, depending on the cases, the size and/or the number of sub-segments to construct.

Endpoint size specifications. Here, we assume known, on each segment, information related to the sizes h_1 and h_2 desired at its endpoints. The problem then comes down to using these data in order to deduce a reasonable series of sub-segments (their size and number) so that the first (resp. the last) sub-segment

[1]At least to our knowledge.

reflects as far as possible the requirement that it has a size close to h_1 (resp. h_2). Moreover, the intermediate sub-segments must have sizes varying smoothly (monotonically) between the sizes at the endpoints. This variation can be linear, geometric or different depending on the ratio between the two given sizes and the objective sought. If h_1 and h_2 are close, the type of variation has little influence on the discretization. However, if these two values are very different and if the (classical) length of the segment allows, noticeably different results (in terms of the number of sub-segments and their distribution) can be obtained depending on the particular size variation function chosen (one emphasizing the small sizes, another having the opposite effect).

General specification map. A more general case is that of a given field of metrics. Such a field can be continuous (analytical case) or discrete (known at certain specific points only). Moreover, the field can be isotropic (it corresponds to sizes) or anisotropic (it specifies sizes in privileged directions). Formally speaking, let us recall that an isotropic field of metrics corresponds to a field of matrices of the form:

$$\mathcal{M} = \frac{1}{\lambda} I_d \,,$$

I_d being the identity matrix and $\lambda = h^2$, where h is the desired size. In two dimensions and in the anisotropic case, the matrices are of the type:

$$\mathcal{M} = \begin{pmatrix} a & b \\ b & c \end{pmatrix},$$

which can also be written as:

$$\mathcal{M} = {}^t\mathcal{R} \begin{pmatrix} \dfrac{1}{h_1^2} & 0 \\ 0 & \dfrac{1}{h_2^2} \end{pmatrix} \mathcal{R},$$

where $\mathcal{R}$ represents the directions to follow and h_1 (resp. h_2) the sizes in these directions. If the values h, h_1, h_2 and the directions of $\mathcal{R}$ do not depend on the spatial position, the field is said to be *uniform*. If, on the other hand, these values depend on the position, they induce a *variable* field.

Isotropic governed meshing

We deliberately leave to one side the two first types of specifications (trivial) in order to focus exclusively on the general problem in the isotropic case (the anisotropic case being discussed in the next section).

Let AB be the given segment. As will be seen later, meshing AB consists essentially of computing the length of the segment and, based on this length and according to the given specifications, subdividing AB into a series of sub-segments of suitable lengths.

Length of a segment AB (general expression). Let $\mathcal{M}(t)$ be the value[2] of the metric (matrix) at point M of parameter $t \in [0,1]$ of the parameterized segment AB. Let us recall that the length of AB with respect to the metric $\mathcal{M}$ is obtained using the formula (Chapter 10):

$$l_{\mathcal{M}}(AB) = \int_0^1 \sqrt{{}^t\overrightarrow{AB}\mathcal{M}(t)\overrightarrow{AB}}dt. \tag{14.1}$$

The exact calculation of this value is usually impossible or at best very tedious. Therefore, we consider specific situations in order to use approximate formulae.

Calculation of the length of a segment AB (case 1). Suppose the values $\mathcal{M}(0)$ at A and $\mathcal{M}(1)$ at B are known. We are looking for a monotonous function $\mathcal{M}(t)$ for each $t \in [0,1]$ smoothly varying between these two values. Depending on the particular form of $\mathcal{M}(t)$, this problem is equivalent to that of finding a function $h(t)$ equal to h_A (resp. h_B) at $t = 0$ (resp. $t = 1$) and smoothly varying between these two values. Indeed, if $h(t)$ is known, Integral (14.1) becomes:

$$l_{\mathcal{M}}(AB) = \int_0^1 \sqrt{{}^t\overrightarrow{AB}\frac{1}{h^2(t)}I_d\overrightarrow{AB}}dt \tag{14.2}$$

and the calculation gives:

$$l_{\mathcal{M}}(AB) = \sqrt{{}^t\overrightarrow{AB}\overrightarrow{AB}}\int_0^1 \frac{1}{h(t)}dt = l(AB)\int_0^1 \frac{1}{h(t)}dt, \tag{14.3}$$

where $l(AB)$ denotes the classical Euclidean length of AB. The calculation of this quantity is thus based on the choice of a function $h(t)$. The simplest solution consists of choosing a linear interpolation function, by fixing:

$$h(t) = h_A + t\,(h_B - h_A).$$

Another linear solution (in $1/h$, this time) corresponds to the choice:

$$\frac{1}{h(t)} = \frac{1}{h_A} + t\left(\frac{1}{h_B} - \frac{1}{h_A}\right).$$

Finally, another example of function h consists of choosing:

$$h(t) = h_A\left(\frac{h_B}{h_A}\right)^t.$$

The choice of a particular interpolation function obviously influences the nature of the resulting distribution.

[2]In the following, we will make the confusion between $\mathcal{M}(t)$ and $\mathcal{M}(M(t))$, as we make the confusion between $h(t)$ and h_M, M being the point of parameter t.

Let us consider for instance, the last choice for h and let us compute the length of the segment AB. According to Formula (14.3), we have to compute the integral:

$$\int_0^1 \frac{1}{h(t)} dt\,,$$

that is: $\displaystyle \frac{1}{h_A}\int_0^1 \left(\frac{h_A}{h_B}\right)^t dt = \frac{1}{h_A}\left[\frac{\left(\frac{h_A}{h_B}\right)^t}{\log\left(\frac{h_A}{h_B}\right)}\right]_{t=0}^{t=1} = \frac{h_A - h_B}{h_A\, h_B \log\left(\frac{h_A}{h_B}\right)}\,.$

Thus, by multiplying by the usual length of AB, we find the length of this segment for the given field of metric:

$$l_{\mathcal{M}}(AB) = l(AB)\,\frac{h_A - h_B}{h_A\, h_B\, \log(\frac{h_A}{h_B})}\,.$$

Remark 14.1 *Notice that if $h_A = h_B$, the previous formula is undetermined. In fact, in this case, we have $h(t) = h_A$ and, thus, we find directly that $l_{\mathcal{M}}(AB) = \frac{l(AB)}{h_A}$. Notice also that if h_B is close to h_A, for instance, $h_B = h_A(1+\varepsilon)$ (for ε small), we obtain the desired length using a limited expansion of the general expression. We thus have $l_{\mathcal{M}}(AB) = \frac{2-\varepsilon}{2\,h_A}\, l(AB)$ and for $\varepsilon = 0$ we retrieve the previous value. Nevertheless, in this case, a geometric function $h(t)$ is not strictly required (it varies only a little). Therefore, a linear function is sufficient (we can easily verify that the choice in $1/h$ gives again, at order 2, the same value for $l_{\mathcal{M}(AB)}$).*

The reader can find in [Laug *et al.* 1996] the values corresponding to other choices of h and, in particular, the first two choices mentioned above.

Calculation of the length of a segment AB (case 2). Only the values $\mathcal{M}(0)$ and $\mathcal{M}(1)$ (i.e., h_A and h_B) are known (at A and at B) but, without justifying any longer the particular choice of the interpolation function, we use only these values and we consider a quadrature formula. For example, we write:

$$l_{\mathcal{M}}(AB) = \frac{l(AB)}{2}\left(\frac{1}{h_A} + \frac{1}{h_B}\right)\,.$$

If h does not vary too much between h_A and h_B, we thus obtain an approximated value relatively close to the exact value. Otherwise, the computed value is only an approximation of the result, and may be quite rough. To get a better answer, we shall consider a function $h(t)$ and apply the same quadrature formula, subdividing the segment into several pieces. The additional information required is then the value of h at each new node of the integration.

Calculation of the length of a segment AB (case 3). Here, we know a series of matrices $\mathcal{M}(t_i)$ at different points along the segment. These points M_i correspond to the parameters t_i. We then retrieve the previous discussion by posing:

$$l_{\mathcal{M}}(AB) = \sum_i l_{\mathcal{M}}(M_i M_{i+1}) \, .$$

There are now two possible solutions. Either we use the method presented in case 1 above and we apply it on each segment $M_i M_{i+1}$, or we use the method described in case 2.

Creation of the sub-segments. This stage of generating a mesh of AB is very simple. We compute the length of AB using one of the previously described possibilities. Let $l = l_{\mathcal{M}}(AB)$ be this length. We pick the closest integer value n to l and we subdivide the segment into n sub-segments of length $\frac{l}{n}$ (hence close to 1). Notice that this unit value, depending on the nature of the metric, leads to variable (Euclidean) lengths in each sub-segment (which is the desired goal).

Anisotropic meshing

We follow here the same idea as in the isotropic case. At first, we compute the length of the segment in the field of metric, then, depending on the value found, we split the segment into sub-segments of unit length. The computation of the desired length is performed, as previously, using three different solutions, depending on the input data and the fixed choices.

Calculation of the length of a segment AB (case 1). Only the metrics at the endpoints are given. In practice, we have two (non-diagonal) matrices and the calculation of the desired length consists of constructing an interpolated metric between the two known values. To this end, we use the method described in Chapter 10 that allows us to obtain, depending on the choices made, a metric at each point of the segment. A metric being known everywhere, the length can be obtained using an approximated calculation, for instance, a dichotomy.

Calculation of the length of a segment AB (case 2). This case leads to computing the desired length approximately by numerical integration of the general formula.

Calculation of the length of a segment AB (case 3). We have a series of matrices at different points along the segment. We retrieve the principle described in the isotropic case, applied here following one of the methods seen above.

Creation of sub-segments. As in the isotropic case.

Remark 14.2 *Notice the whole discussion corresponds exactly to what was performed in the internal point creation method proposed in Chapter 7.*

Examples of straight segment meshes

First, we consider here two examples of isotropic nature. In the first example, the metric is represented in Figure 14.1 where the points Q_i supporting the sizing information are shown. Table 14.1 indicates the isotropic specifications at points Q_i.

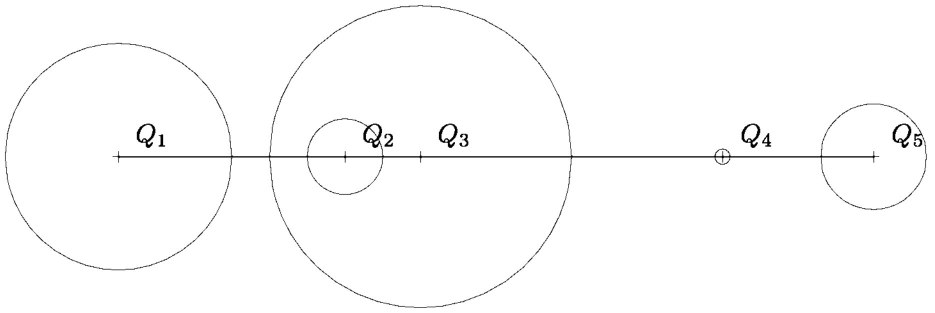


Figure 14.1: *Discrete (isotropic) size specification.*

Figure 14.2 shows the resulting mesh, for a choice corresponding to a linear propagation (function $h(t)$), for a segment when a discrete size specification is provided. The mesh has 17 elements. Figure 14.2 shows (bottom) the endpoints R_i of the elements. Above, the points Q_i defining the metric can be seen. At the top, the function $h(t)$ is represented and, by means of rectangles, its approximation on each element.

	Q_1	Q_2	Q_3	Q_4	Q_5
s_i	0.0	3.0	4.0	8.0	10.0
h_i	1.5	0.5	2.0	0.1	0.7

Table 14.1: Discrete isotropic specifications at points Q_i.

Figure 14.3 shows the resulting mesh when a geometric propagation is chosen (function $h(t)$), for the segment when the same discrete size specification is provided (Figure 14.1 and Table 14.1). The mesh now has 26 elements. Notice that the smallest sizes are privileged by this type of distribution function.

We now consider two anisotropic examples. The metric specification is also discrete. It corresponds to the data (Table 14.2) of directions and sizes at points Q_i illustrated in Figure 14.4. Two types of propagation function are represented. Figure 14.5 shows the result obtained by fixing a linear function whereas Figure 14.6 corresponds to the case of a geometric function. The figures show the mesh of the segment (bottom) and the evolution of the metrics (i.e., the interpolation between the metrics known at points Q_i) along the segment AB (top).

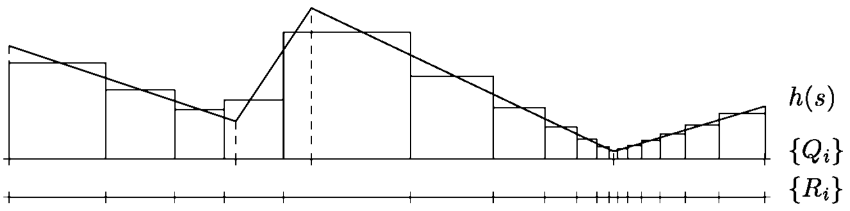


Figure 14.2: *Mesh resulting from a linear interpolation.*

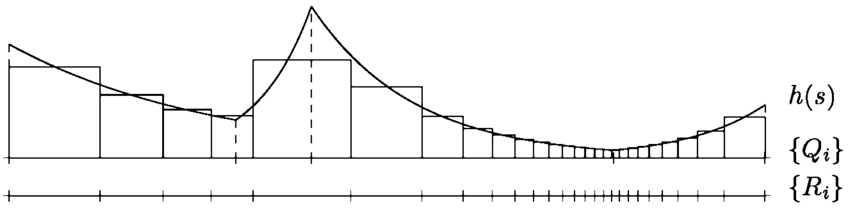


Figure 14.3: *Mesh resulting from a geometric interpolation.*

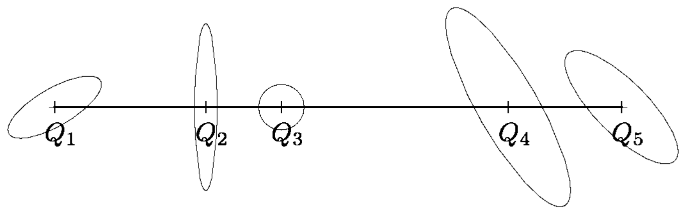


Figure 14.4: *Discrete specifications of directions and sizes.*

	Q_1	Q_2	Q_3	Q_4	Q_5
s_i	0.0	2.0	3.0	6.0	7.5
θ_i	30°	90°	0°	−60°	−45°
$h_{1,i}$	0.70	1.10	0.30	1.50	1.00
$h_{2,i}$	0.25	0.15	0.30	0.40	0.35

Table 14.2: Discrete anisotropic specifications at points Q_i.

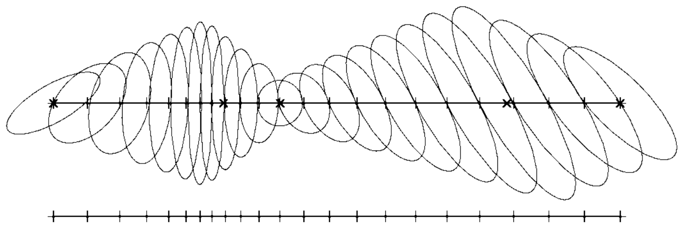


Figure 14.5: *Mesh resulting from a linear interpolation.*

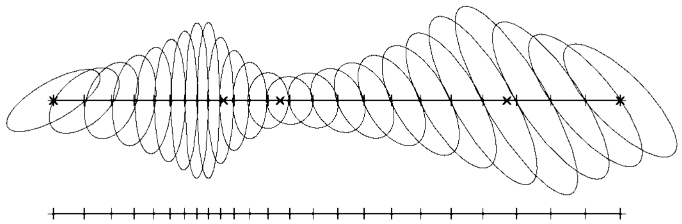


Figure 14.6: *Mesh resulting from a geometric interpolation.*

14.2 Meshing a parametric curve

We are now interested in the case of curves. The principle remains the same. First, we compute the length of the curve for the given field of metrics, then we subdivide this curve into segments, in order to obtain the corresponding unit arc length (i.e., close to 1).

However, unlike the case of a segment, here we must follow the geometry of the curve so as to make sure that the resulting mesh is close to it, in some sense. Therefore, if the given metric is not geometric by nature, it has to be corrected to take this requirement into account. To see this more clearly, we look first at a naive example.

A naive though probably wrong method. As an exercise, we analyze on a simple case of a sinusoid, two examples of meshes having uniform sized edges. Although similar in refinement, it is obvious (in Figure 14.7) that the left-hand side mesh is much better than the right-hand side mesh. The choice of a smaller stepsize would obviously lead to two almost identical meshes. This simple example emphasizes the influence of the point location, given a stepsize. The stepsize influence is very easy to see.

This leads to analyzing how to mesh a curve with respect to its geometry, first without an explicit field, then given such a field.

Geometric mesh

We wish to mesh a curve in such a way that its geometry is respected. The metric to follow is thus strictly related to the geometry (the case where a different metric is specified will be discussed later). The aim of this section is to show how to

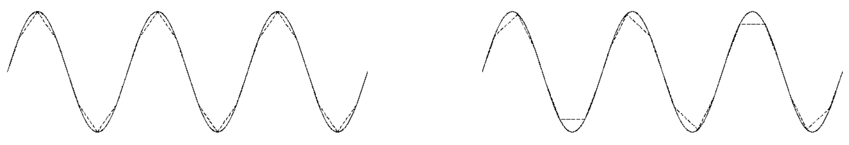


Figure 14.7: *Uniform meshes. Left-hand side: the mesh (dashed lines) "match" the geometry (full lines). Rright-hand side: the mesh has (roughly) the same stepsize but is "shifted".*

control the meshing process, that is to define the control metric so as to use it to mesh the given curve effectively.

Desired properties and related problems. The immediate question is how to qualify and quantify the parameters to ensure that the mesh follows the geometry. If h is the length of an element (a segment) and if δ measures the smallest distance between this segment and the curve, then, for a given ε, we want to have:

$$\delta \leq \varepsilon\, h\,.$$

This inequality defines a relative control (Figure 14.8) that can be interpretated in a simple way: the curve length (the arc of curve) and that of the corresponding chord (the chord sub-tending this arc) are close. If s is the length of the curve corresponding to the chord of length h, this condition can also be expressed as:

$$|h-s| \leq \varepsilon\, s \quad \text{or} \quad |h-s| \leq \varepsilon\, h\,.$$

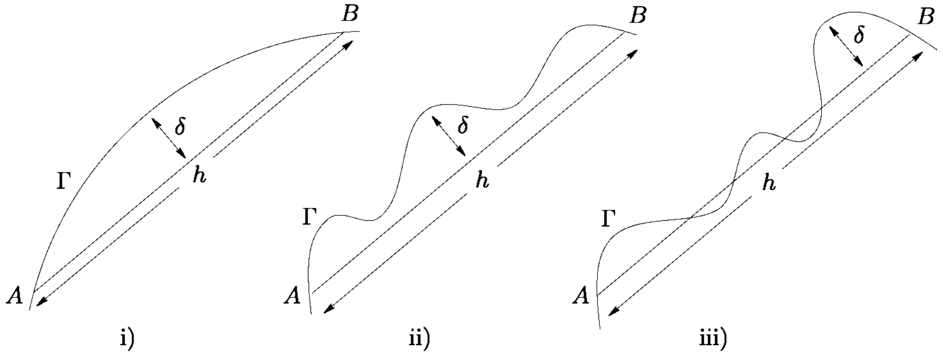


Figure 14.8: *Gap between an arc of the curve Γ and the corresponding chord, the chord AB of length h (i.e., the segment of the mesh supposed to approach this curve).*

The problem is then to find the location of the points along the curve in such a way as to satisfy this property. This problem induces two sub-problems that are obviously related to each other:

- where to locate the points and
- how, given a point, to find the next one.

To illustrate these two questions, let consider the case of a circle. It is obvious that finding a first point M_0 on the circle is easy, as any point will be suitable! Hence, M_0 being chosen, the second question consists of finding a point M_1, and step by step the series of points $M_i, i = 2, 3, ...$ such that the length of the segments M_iM_{i+1} corresponds to the given accuracy. For other types of curves and, especially curves having discontinuities, it is obvious that the points of discontinuity must necessarily be mesh points. Hence, these points being fixed, the second question consists in finding the next points that form closely spaced segments along the curve, with respect to the given accuracy. This being fixed, we assume that the particular points imposed have been identified and we discuss only the case of sufficiently regular curves (without discontinuities) that, in fact, correspond to the different pieces defined between the imposed points.

Back to the local approximation of a curve. As seen in Chapter 11, a limited expansion allows us to study the local behavior of a curve. If Γ is a supposedly sufficiently regular curve described by function γ and if s denotes the curvilinear abscissa, we can write in the neighborhood of s_0:

$$\gamma(s) = \gamma(s_0) + \Delta s \gamma'(s_0) + \frac{\Delta s^2}{2}\gamma''(s_0) + \frac{\Delta s^3}{6}\gamma'''(s_0) + \ldots,$$

with $\Delta s = s - s_0$ being sufficiently small to ensure the validity of this development (i.e., the terms corresponding to the ... are negligible). As $\gamma'(s) = \vec{\tau}(s)$ (the tangent), $\gamma''(s) = \dfrac{\vec{\nu}(s)}{\rho(s)}$ (the ratio between the normal and the radius of curvature) and as $\gamma'''(s) = \dfrac{-\rho'(s)\,\vec{\nu}(s) - \vec{\tau}(s)}{\rho(s)^2}$, the above expression can also be written as follows:

$$\gamma(s) = \gamma(s_0) + \Delta s\,\vec{\tau} + \frac{\Delta s^2}{2\,\rho(s_0)}\,\vec{\nu} - \frac{\Delta s^3}{6\,\rho(s_0)^2}(\rho'(s_0)\,\vec{\nu} + \vec{\tau}) + \ldots, \tag{14.4}$$

where $\vec{\tau} = \vec{\tau}(s_0)$ and $\vec{\nu} = \vec{\nu}(s_0)$. Notice that the point P defined by:

$$P = \gamma(s_0) + \Delta s\,\vec{\tau} + \frac{\Delta s^2}{2\,\rho(s_0)}\,\vec{\nu}$$

i.e., the point of the parabola passing through $\gamma(s_0)$, having a tangent $\vec{\tau}$ at this point and located at the distance Δs from $\gamma(s_0)$, is very close to the *osculating circle* to the curve. Let us consider the point O defined by:

$$O = \gamma(s_0) + \rho(s_0)\,\vec{\nu},$$

then:

$$\|\overrightarrow{OP}\|^2 = \left\langle \Delta s\,\vec{\tau} + \left(\frac{\Delta s^2}{2\,\rho(s_0)} - \rho(s_0)\right)\vec{\nu},\ \Delta s\,\vec{\tau} + \left(\frac{\Delta s^2}{2\,\rho(s_0)} - \rho(s_0)\right)\vec{\nu} \right\rangle,$$

thus, we have:

$$\|\overrightarrow{OP}\|^2 = \Delta s^2 + \frac{\Delta s^4}{4\rho(s_0)^2} + \rho(s_0)^2 - \Delta s^2$$

and, therefore:

$$\|\overrightarrow{OP}\|^2 = \rho(s_0)^2 + \frac{\Delta s^4}{4\rho(s_0)^2} = \rho(s_0)^2 \left(1 + \frac{\Delta s^4}{4\,\rho(s_0)^4}\right),$$

thus finally:

$$\|\overrightarrow{OP}\| = \rho(s_0)\sqrt{1 + \frac{\Delta s^4}{4\,\rho(s_0)^4}}\,. \tag{14.5}$$

To conclude, when Δs is sufficiently small, the point P above is very close to the circle of center O and of radius $\rho(s_0)$ (i.e., the osculating circle to the curve at M_0). This well-known observation has an important practical consequence. If the point P corresponding to the first three terms of Equation (14.4) is close to the curve, then the point corresponding to the osculating circle (very close to this curve) is very close to this point P. Hence, we can base the analysis on the point of the osculating circle and thus set the desired construction on the corresponding construction on the osculating circle.

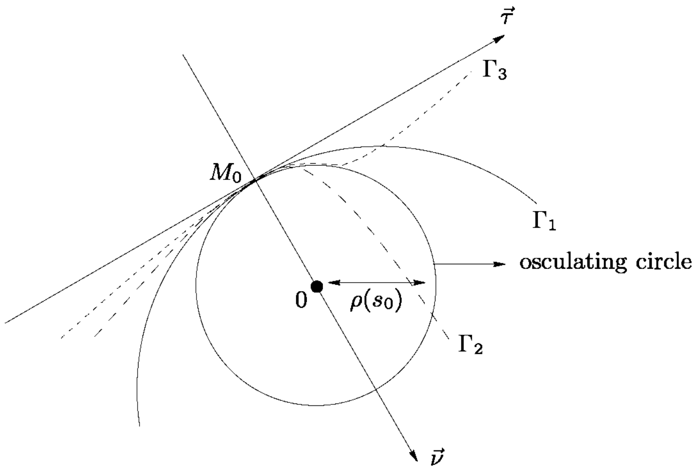


Figure 14.9: *Three types of behavior in a given neighborhood of s_0 (at M_0). We show the curves Γ_1, Γ_2 and Γ_3 passing through M_0, having the same tangent $\vec{\tau}$, the same normal $\vec{\nu}$ and the same osculating circle at M_0. It is obvious that the curve Γ_1 has a behavior such that the analysis at s_0 gives a good indication of what can be further expected, which is not the case for the two other examples.*

The validity of the reasoning is guaranteed as long as the fourth term (and the following ones) of Equation (14.4) is (are) negligible as compared to the previous ones. This leads to a restriction on the stepsize Δs possible (Figure 14.9) depending on the behavior of the curve in a neighborhood of s_0 of size this stepsize Δs.

The term we are interested in is the value:

$$\frac{\Delta s^3}{6\,\rho(s_0)^2}(\rho'(s_0)\,\vec{\nu}\,+\,\vec{\tau})\,, \tag{14.6}$$

which, in particular, involves the variation of ρ (i.e., value ρ'). This term will be negligible if:

$$\frac{\Delta s^3}{6\,\rho(s_0)^2}\|\rho'(s_0)\,\vec{\nu}+\vec{\tau}\| \ll \left\|\Delta s\,\vec{\tau}+\frac{\Delta s^2}{2\,\rho(s_0)}\vec{\nu}\right\|.$$

This condition is equivalent to assuming that:

$$\frac{\Delta s^3}{6\,\rho(s_0)^2}\|\rho'(s_0)\,\vec{\nu}+\vec{\tau}\| = \varepsilon \left\|\Delta s\,\vec{\tau}+\frac{\Delta s^2}{2\,\rho(s_0)}\vec{\nu}\right\|.$$

choosing a sufficiently small ε. We thus have:

$$\Delta s^2 = 6\,\varepsilon\,\rho(s_0)^2\,\sqrt{\frac{1+\dfrac{\Delta s^2}{4\,\rho(s_0)^2}}{1+\rho'(s_0)^2}}\,.$$

This leads to solving an equation (in Δs) of the form:

$$\Delta s^4\,(1+\rho'(s_0)^2) = 6^2\,\varepsilon^2\left(1+\frac{\Delta s^2}{4\,\rho(s_0)^2}\right)\rho(s_0)^4\,,$$

that, posing $\nabla = \Delta s^2$, can be written:

$$(1+\rho'(s_0)^2)\,\nabla^2 - 9\,\varepsilon^2\,\rho(s_0)^2\,\nabla - 36\,\varepsilon^2\,\rho(s_0)^4 = 0\,.$$

This yields:

$$\nabla = \frac{9\varepsilon^2+3\,\varepsilon\sqrt{9\varepsilon^2+16\,(1+\rho'(s_0)^2)}}{2\,(1+\rho'(s_0)^2)}\rho(s_0)^2 \quad \text{and thus} \quad \Delta s = \sqrt{\nabla}\,.$$

Hence, we have $\Delta s = \alpha\,\rho(s_0)$ and the previous relation leads to choosing as coefficient α a value such that:

$$\alpha \leq \sqrt{\frac{9\varepsilon^2+3\,\varepsilon\sqrt{9\varepsilon^2+16\,(1+\rho'(s_0)^2)}}{2\,(1+\rho'(s_0)^2)}} \approx \frac{\sqrt{6\,\varepsilon}}{\sqrt[4]{1+\rho'(s_0)^2}} \tag{14.7}$$

Such a choice ensures that the point (of the parabola):

$$\gamma(s_0)+\Delta s\,\vec{\tau}+\frac{\Delta s^2}{2\,\rho(s_0)}\,\vec{\nu} = \gamma(s_0)+\alpha\,\rho(s_0)\,\vec{\tau}+\alpha^2\frac{\rho(s_0)}{2}\,\vec{\nu}$$

is close to the curve at the order two, within the following term of Relation (14.4), that is $\alpha^3\dfrac{\rho(s_0)}{6}(\rho'(s_0)\,\vec{\nu}\,+\,\vec{\tau})$ which gives a gap of:

$$\delta = \frac{\sqrt{9\varepsilon^2+3\,\varepsilon\sqrt{9\varepsilon^2+16\,(1+\rho'(s_0)^2)}}^{\,3}}{12\,\sqrt{2}\,(1+\rho'(s_0)^2)}\,\rho(s_0) \approx \frac{\varepsilon\sqrt{6\,\varepsilon}}{\sqrt[4]{1+\rho'(s_0)^2}}\,\rho(s_0)\,. \tag{14.8}$$

As an example, Table 14.3 gives some values of α and of $\dfrac{\delta}{\rho(s_0)}$ according to the given ε and to different values $\rho'(s_0)$. Notice by the way that (and the results in the table confirm it):

$$\delta \approx \varepsilon\, \alpha\, \rho(s_0)\,.$$

$\rho'(s_0)$	α	α^*	$\delta/\rho(s_0)$
0.0	.245	.2449	.00247
0.5	.232	.231	.00234
1.0	.206	.2059	.00207
$\sqrt{2}$	.186	.186	.00187
$\sqrt{3}$	.173	.173	.00171
2.0	.164	.1638	.00164
3.0	.137	.137	.00138
5.0	.108	.108	.00108
10.0	.07729	.07726	.00077

Table 14.3: Several values of α (and the approached value α^*) determining the stepsize with respect to the gap for $\varepsilon = .01$ and different values of $\rho'(s_0)$.

From the previous discussion we deduce that the curve Γ can be analyzed in a neighborhood of $\gamma(s_0)$ of size $\alpha\,\rho(s_0)$, by looking at the parabola corresponding to the first three terms of the limited expansion of γ.

Let us consider a chord of length $\Delta s = \alpha\,\rho(s_0)$ coming from the point $\gamma(s_0)$ and look at the gap between this chord and the curve. From the previous discussion, we can estimate this gap as the gap between the chord and the parabola. The maximum corresponds to the distance between the midpoint of the parabola (the point P of abscissa $\frac{\alpha}{2}\,\rho(s_0)$) and the point M which is the point of the chord corresponding to the orthogonal projection of P. In the frame $[\gamma(s_0), \vec{\tau}, \vec{\nu}]$, the above parabola is written $y = \dfrac{x^2}{2\,\rho(s_0)}$ and the coordinates of the point P are thus:

$$\frac{\alpha}{2}\,\rho(s_0) \quad \text{and} \quad \frac{\alpha^2}{8}\,\rho(s_0)\,.$$

The normalized equation of the line passing through $\gamma(s_0)$ and $\gamma(\alpha\,\rho(s_0))$ in this frame is written:

$$\frac{2\,\rho(s_0)\,y}{\sqrt{4\,\rho(s_0)^2 + \Delta s^2}} - \frac{\Delta s\, x}{\sqrt{4\,\rho(s_0)^2 + \Delta s^2}} = 0\,,$$

or also, with respect to α:

$$\frac{2\,y}{\sqrt{4+\alpha^2}} - \frac{\alpha\, x}{\sqrt{4+\alpha^2}} = 0\,.$$

The distance from this line to the point P, i.e., the distance $\|\overrightarrow{PM}\|$, if thus of the order of:

$$\delta = \frac{\alpha^2}{4\sqrt{4+\alpha^2}}\,\rho(s_0) \approx \frac{\alpha^2}{8}\,\rho(s_0)\,.$$

Or, depending on ε:

$$\delta = \frac{3\,\varepsilon}{4\sqrt{1+\rho'(s_0)^2}}\,\rho(s_0)\,. \tag{14.9}$$

Before going further, let calculate the quantity $|h-\Delta s|$ where h is the length of the chord sub-tending the arc Δs. We have, using the second order approximation of the curve:

$$h = \sqrt{\alpha^2 + \frac{\alpha^4}{4}\rho(s_0)}\,,$$

hence:

$$|h-\Delta s| = \left|\alpha - \sqrt{\alpha^2 + \frac{\alpha^4}{4}}\right|\,\rho(s_0)\,,$$

$$|h-\Delta s| = \left|1 - \sqrt{1 + \frac{\alpha^2}{4}}\right|\,\alpha\,\rho(s_0) \approx \frac{\alpha^2}{8}\,\Delta s\,,$$

or also:

$$|h-\Delta s| \approx \frac{3\,\varepsilon}{4\sqrt{1+\rho'(s_0)^2}}\,\Delta s\,. \tag{14.10}$$

This value, measuring the relative gap between the lengths of the arc and that of the curve, will be involved later in the definition of the desired mesh that we propose.

By merging the majorations of Relations (14.8) and (14.9), we find that the gap between the chord and the curve is majorated by:

$$\delta \approx \left(\frac{\varepsilon\sqrt{6\,\varepsilon}}{{}^4\sqrt{1+\rho'(s_0)^2}} + \frac{3\,\varepsilon}{4\sqrt{1+\rho'(s_0)^2}}\right)\rho(s_0)\,.$$

The validity of these above demonstrations is ensured if the terms neglected in Relation (14.4) are in fact negligible. To this end, we are looking for an upper bound and we return to the limited expansion:

$$\gamma(s) = \gamma(s_0) + \Delta s\gamma'(s_0) + \frac{\Delta s^2}{2}\gamma''(s_0) + \frac{\Delta s^3}{6}\gamma'''(s_0) + \dots\,.$$

However, we know that a value s_j exists between s_0 and s such that:

$$\gamma_j(s) = \gamma_j(s_0) + \Delta s\gamma_j'(s_0) + \frac{\Delta s^2}{2}\gamma_j''(s_0) + \frac{\Delta s^3}{6}\gamma_j'''(s_j)$$

for each component[3] ($j = 1, d$) of γ. If m is an upper bound of $||\gamma'''||$ on the interval $[s_0, s]$ and if we note $r(s)$ the set of terms $\frac{\Delta s^3}{6}\gamma'''(s_0) + \ldots$, then:

$$||r(s)|| \leq \frac{\Delta s^3}{6}\, m\,,$$

hence, setting an accuracy ε' is equivalent to fixing:

$$\frac{\Delta s^3}{6}\, m = \varepsilon' \, \Delta s\,.$$

With the particular form of Δs, we find:

$$\varepsilon = \frac{\sqrt{1 + \rho'(s_0)^2}}{\rho(s_0)^2\, m}\, \varepsilon' \tag{14.11}$$

and, thus, ε must be less than or equal to this value.

Exercise 14.1 *Consider as a curve the quarter of circle defined by:*

$$\gamma(s) = \left(\rho\, cos(\frac{s}{\rho})\,,\, \rho\, sin(\frac{s}{\rho})\right)\,,$$

find that $m = \dfrac{1}{\rho^2}$ *and apply Relation (14.11). We then find* $\varepsilon = \varepsilon'$.

From a practical point of view, determining M is not strictly trivial. Therefore, the meshing technique must, somehow, overcome this drawback (see below).

We will then exploit the results obtained to design a meshing method. First, we introduce the notion of a *geometric mesh.*

Geometric mesh of a curve.

Definition 14.1 *A* geometric mesh *of type P^1, within a given ε, of a curve Γ is a piecewise linear discretization of this curve for which the relative gap to the curve at any point is of the order of ε. More precisely, if Δs is the length of an arc of the curve and if h is the length of the corresponding curve, we have (at the limit) when Δs tends towards* 0*:*

$$h = \Delta s$$

or also:

$$\left|\frac{h}{\Delta s} - 1\right| = 0\,,$$

or, for a given ε:

$$\left|\frac{h}{\Delta s} - 1\right| \leq \varepsilon\,.$$

[3]Correct for a scalar function, this result is not verified for a vector function. It is true only component by component and s_j depends on the index j of the component.

This last relation can be also expressed as:

$$|h - \Delta s| \leq \varepsilon \, \Delta s \,.$$

Remark 14.3 *Notice that other methods can be used to evaluate the gap between the curve and its mesh. Indeed, we can use the value of the surface enclosed between this arc and the corresponding curve rather than computing the difference of length, as above.*

Obviously, a sufficiently (infinitely) fine mesh is necessarily a geometric mesh. However, the mesh we are looking for here must be both geometric and *minimal* (that is, having as small a number of elements as possible). For instance, if we subdivide a line segment into 100 segments, we obtain a geometric mesh, whereas the mesh formed by the segment itself is already geometric and (obviously) minimal.

Remark 14.4 *Definition (14.1) is one of the possible definitions. In fact, the notion of a geometric conformity underlying to the notion of a geometric mesh is obviously determined by the application envisaged. In particular, defining the conformity to the geometry by imposing that the discretization be enclosed within a band of a given (small) length corresponds to a different definition, as coherent as the previous one, that leads naturally to a rather different meshing technique.*

If we retain the previous definition, the analysis technique of the local behavior of the curve described previously can serve to construct a geometric mesh. We then replace the parabola approaching the curve by a segment.

The local analysis indicates that, at any point of Γ, the desired length is $\alpha \, \rho(s)$ where s denotes the curvilinear abscissa at this point, α satisfies Condition (14.7) and $\rho(s)$ is the radius of curvature of the curve at s.

In terms of metrics, the Euclidean length $\alpha \, \rho(s)$, can be interpretated as the *unit* length of the metric field defined by the stepsize $\frac{1}{\alpha \, \rho(s)}$, field that can be written in a matrix form as:

$$\mathcal{M}(s) = \frac{1}{\lambda(s)} I_d \,,$$

with $\lambda(s) = \alpha^2 \, \rho(s)^2$.

Hence, in terms of metric and unit length, the meshing method consists of constructing a unit mesh for this field of metrics.

Exercise 14.2 *Look at the angular gap between the curve (its tangent) and the mesh resulting from the previous method. What is the conclusion?*

Constructing a geometric mesh of a parametric curve consists of applying this principle in its range of validity. Hence, a meshing method consists of:

- identifying the extrema (for the radii of curvature) of the curve as well as the singular points,
- setting these points as mesh vertices,

- subdividing the curve into pieces, each piece being limited by two such points,
- meshing each piece by applying the previous principle.

The meshing of a piece of curve then consists of computing its length for the field of metrics of the radii of curvature, weighted by the coefficients α that have been introduced. We then retrieve exactly the same meshing method as for a straight segment. We search the length l, we round it to the nearest integer n and we split it into segments of length $\frac{l}{n}$. Notice that the lengths computations are performed on the curve (curvilinear abscissa) and that the stepsize of the mesh is the length 1 of the curve. In other words, we identify the length of the curve and that of the chord corresponding to the stepsize.

Remark 14.5 *This meshing process is rather tedious to implement and can only be applied to configurations similar to that represented on the left-hand side of Figure 14.8. The other two situations shown on this figure must be treated in a different manner. Actually, the problem is more one of simplifying the geometry than a problem of conformity (the conformity is true within a strip of given width). In other words, a geometric mesh, according to the previous definition, cannot be reduced to a single segment in these two cases. The corollary is that such a geometric mesh may be relatively large (in terms of the number of elements) in such cases.*

Remark 14.6 *On the contrary, a geometric mesh of a set of straight segments is this set itself and contains very few elements.*

Hints regarding implementation. The implementation of the method can be envisaged in various, more or less complex, forms. The main difficulty lies in calculating the length of the (portion of the) curve with respect to the metric field constructed. In fact, during the processing of a portion of curve, we can use a very fine (uniform) sampling to evaluate the desired length.

However, this simple method can turn out to be expensive because capturing the geometry requires *a priori* a very small sampling step (although it may not be strictly required). Hence, other methods to calculate this length can be adopted. We suggest in this regard, the following method:

- we split the chord underlying the curve (we denote d the length of this chord) into n segments with n being relatively small (n depends on the presumed degree of the curve considered). We associate the curve parameter with a parameterization of this chord,
- on each interval, we *randomly* pick a point,
- for each such points, we evaluate the gap to the curve,
- the maximum of these gaps is the gap δ between the curve and the chord,
- if $\delta \leq \varepsilon\, d$, END,

- else, we retain the point of the curve corresponding to the point of the chord where the gap is maximal and we split the curve into two at this point. We replace the curve by the two corresponding chords whose lengths are respectively denoted d_1 and d_2 and we iterate the process on each segment of the subdivision.

Remark 14.7 *The previous test is equivalent to analyzing whether:*

$$\left| \frac{d}{d_1 + d_2} - 1 \right| \leq \varepsilon .$$

As a result of this algorithm, we have a set of segments P_iP_{i+1} approaching the curve. We still have to determine the length of the curve in the metric specified. For each segment constructed:

- we find the value of the metric at P_i and at P_{i+1},
- we use an interpolation between these two values to determine a metric everywhere,
- we compute the length of the segment in this interpolated metric (or using a quadrature; see the beginning of this chapter),
- if the length is bigger than a given threshold value (for instance, one-half), we use the metric of the midpoint to refine the length calculation and the process is iterated.

At completion of the procedure, we have segments whose lengths are less than or equal to the given threshold. It is then easy to deduce the desired unit mesh.

Other methods. We propose first a heuristic method:

- if P is a known mesh point, we find the next point P_{+1} that, if it were retained as such, would allow the segment PP_{+1} to be formed,
- then, considering P_{+1}, we travel the curve in the opposite direction to find the corresponding point P_{-1} and
  - if P_{-1} is "before" P, the point P_{+1} is judged correct, we form the segment PP_{+1} and we analyze the corresponding portion of curve to make sure that it stays close (for instance, by dichotomy),
  - else, we set $P_{+1} = P_{-1}$ and we form as segment PP_{+1} the segment PP_{-1} and we analyze the two corresponding portions of curve (by dichotomy).

Another method of the "dichotomy" or "divide and conquer" type (Chapter 2) can be envisaged. We identify the extrema of curvature and the singular points. We join two such consecutive points. The portion of curve limited by two such points is replaced by the corresponding segment. We analyze the gap between this segment and the curve. In the case of an intersection, we split the segment at the intersection points and we apply one of the previously described methods to each sub-segment. If the gap is satisfactory, we have the desired mesh. Otherwise, we apply one of the previous methods. The advantage of this approach is that we very rapidly obtain a situation where the portion of the underlying curve is very close to the segment and, moreover, is very regular.

Finally, another possibility consists of working on a discrete representation of the curves as will be seen below.

Remarks concerning mesh simplification. A point where the radius of curvature ρ is smaller than the given threshold ε (within a coefficient) is not retained as an extremum. This very simple intuitive idea allows us to suppress insignificant details and, hence, to construct simplified geometric meshes, within ε (i.e., such a mesh follows the geometry within ε).

As a particular case and to complete the general discussion, we look at how to mesh a circle while controlling the gap between the circle and the mesh segments.

Meshing a circle. We consider a circle of radius ρ. We construct a chord of length $h = \alpha\,\rho$, the gap δ between this chord and the circle is given by the formula:

$$\delta = \rho\left(1 - \sqrt{1 - \frac{\alpha^2}{4}}\right)$$

and thus, imposing $\frac{\delta}{\rho} < \varepsilon$ is equivalent to fixing:

$$\alpha \leq 2\sqrt{\varepsilon\,(2-\varepsilon)} \tag{14.12}$$

and, thus, an accuracy of ε (given) imposes that α is bounded in this way.

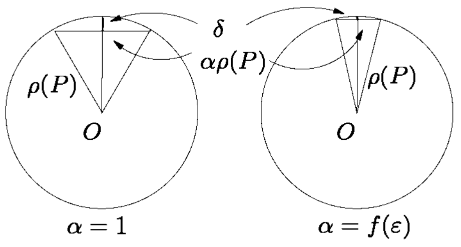


Figure 14.10: *Discretization of a circle into segments of length $\alpha\,\rho$. Left-hand side, we have $\alpha = 1$; right-hand side, α is smaller.*

If we consider segments of sizes $\alpha\,\rho$, so as to satisfy Relation (14.12), we obtain a discretization for which the gap to the circle is controlled, relative to ρ, by a threshold of value $1-\sqrt{1-\frac{\alpha^2}{4}}$, which again gives, as would be expected, according to Relation (14.5), a gap in $\frac{3\,\varepsilon}{4}\rho(s_0)$, hence of the same order (better in fact) than the given threshold ε.

As an example (see the exercise below), the choice $\varepsilon = 0.01$ leads to meshing the circle with 26 elements.

Exercise 14.3 *Verify that this number corresponds to a limited expansion of order 2 with the value* $\varepsilon = 0.01$ *(hint: start from Relation (14.6), assume it is equal to* $\varepsilon\,\Delta s$*, set* $\Delta s = \alpha\,\rho(s_0)$*, deduce* α *and find the number of segments of length* $\alpha\,\rho(s_0)$ *required to cover the perimeter of the circle).*

Exercise 14.4 *Calculate the number of points required if a limited expansion of order 1 is used (Hint: suppose that* $\frac{\Delta s^2}{2\,\rho(s_0)} = \varepsilon\,\Delta s$*, deduce, if* $\Delta s = \alpha\,\rho(s_0)$*, a value of* α *then deduce the number of points required to cover the circle). Deduce that this analysis is not sufficient as too many elements are in fact required to obtain the desired accuracy. Restart by controlling the term in* Δs^3 *with respect to the term in* Δs^2*. What is the conclusion? Notice that in fact the method deduced from these two approaches is unusable in practice. At the same time, notice that we find a justification of the previous method.*

Meshing algorithm without a metric map

As mentioned above, the geometric mesh of the curve is not necessarily suitable for a given calculation. For instance, a portion of curve reduced to a line segment has a geometric mesh identical to the segment, irrespective of its length. Hence, this element size can be very different from the other sizes of the neighboring elements and at least, a control of the size variation from element to element must be performed (see again in Chapter 10 the metric correction procedure).

Meshing algorithm with a metric map

The data is a curve and a field of isotropic or anisotropic metrics. This field, derived from an arbitrary calculation, does not necessarily follow the geometry of the curve. Hence, meshing the curve so as to conform to this field does not usually lead to a geometric approximation of the curve.

The idea then is to combine the given field with the intrinsic geometric field of the curve (i.e., the field of the radii of curvature). The metric to consider is defined as the intersection of two given metrics (Chapter 10).

14.3 Curve meshing using a discrete definition

In this approach, the curve Γ is not known explicitly during the meshing process. It is actually supposed to be known only via a discrete definition (i.e., a mesh reputed sufficiently small to reflect the true geometry). The construction of this mesh, the so-called *geometric mesh*, is the responsibility of the user's favorite CAD system.

Each segment of the geometric mesh is replaced by a curve C^1 or G^1 by using the available information (neighbors, singular points, tangents, normals, etc.). The union of these curves forms the *geometric support* that now serves as a geometric definition of the curve Γ. The mesh is then constructed based on this support so as to follow a given metric map if such a map is specified. We then encounter three cases:

- without map specification, we construct a geometric mesh that can be made suitable for numerical computations afterwards via a smoothing of the possible size shocks between two successive elements,
- is a map has been specified, we construct a mesh that follows this map, or
- in the same case, we construct a mesh that both follows this map and which is geometric.

The underlying idea is to get rid of the mathematical form of γ (the function representing Γ) and thus, to propose a method disconnected from the CAD system that generated the curve. Actually, the sole data required is a mesh, possibly a very fine one, which any CAD system is able to provide.

Construction of a definition from discrete data

We follow here the principle given in Chapter 12 where we chose to construct a curve of degree 3 based on each of the segments of the discrete data. If $\gamma(t)$ is the parameterization of this curve, we have:

$$\gamma(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 \,, \tag{14.13}$$

and the question is to find the coefficients a_i as well as the interval of variation of the parameter t. The answer is given in Chapter 12 to which the reader is referred. The values of the a_i's and the degree of the curve obtained are given with respect to the known information.

The choice of a different type of curve is, needless to say, another solution.

Curve approximation by a polygonal segment

This construction consists of replacing the arcs of the curve by polygonal segments, by controlling the gap between each of these segments and the corresponding arc.

A simple solution consists of distributing regularly (i.e., uniformly) an arbitrary large number of points along the curve. This solution, which is relatively easy to implement, usually leads to a polygonal support that is much too fine (the number of elements being too great). Another solution consists of finding the points strictly required so that the above gap is bounded. The advantage is then to minimize the size of the resulting discretization while following the geometry accurately, especially in high curvature regions.

Construction of the polygonal segment: method 1. The careful reader has most probably noticed that the desired polygonal segment is nothing other than a geometric mesh according to the previous definition. Hence, any of the previously introduced methods may serve to provide the desired result.

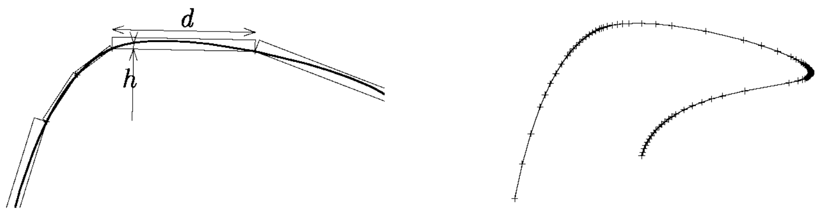


Figure 14.11: *Geometric approximation using a discrete definition formed by a polygonal segment respecting a given tolerance* ε. *Left-hand side;* $\varepsilon = 0.08$, *right-hand side,* $\varepsilon = 0.01$.

Construction of the polygonal segment: method 2 (bandwidth method). We suggest, however, a rather different method (which we could have presented at the same time as the other methods).

We consider an arc of the curve and the underlying segment. We denote in Figure 14.11 (left-hand side):

- h the distance from one point of the segment to the arc,
- d the length of this segment and

we give ε a tolerance threshold. The aim is to ensure $h < \varepsilon\, d$, at any point along the segment. To this end:

- we find the extremum or the extrema in h and we denote such a point E_i,
- if the relation $h < \varepsilon\, d$ is satisfied at this (these) point(s), then the segment is judged correct and is retained in the mesh,
- else, we subdivide this segment at these points and we iterate.

Simple in its principle, this method is nonetheless quite technical (in particular, to find the extrema). Figure 14.11 shows two results, depending on the value of ε considered.

Curve meshing from a discrete definition

The general scheme is always identical. We compute the length of the curve, then we construct the discretization based on this length. The length of the curve is calculated based on the polygonal segment previously constructed. This length is evaluated with respect to the given metric.

Mesh without field specification. Here again, we encounter the previously mentioned problem. The geometric mesh is not *a priori* necessarily correct in view of a numerical computation. The size variation between two adjacent elements may be large and, if this is the case, a control of this variation may be required.

Mesh with respect to a field specification. Once more, we reach the same conclusion as above. The metric to follow does not necessarily conform to the geometry. Depending on whether we have to respect the latter or not, we must perform the intersection between the specified metric and the geometric metric. Notice, in principle, that we do not know explicitly the latter metric as we have only the polygonal segment serving as a geometric support. Therefore, a way of determining the points of minimal radii of curvature and the singular points (discontinuities) is, for instance, to store this information during the construction of the polygonal segment. This being done, we find the usual situation. With a field specification, we mesh the curve by portions, each portion being delimited by two such consecutive points.

Examples of planar curve meshes

We propose, in Figures 14.13, two examples of meshes of a same curve obtained for the specification of the discrete metric field described in Figure 14.12. On the left-hand side, the metric specified has been interpolated linearly while on the right-hand side, the interpolation is geometric. In these figures, we give the interpolated metrics at the various vertices of the resulting mesh. This allows us to verify the coherence of the result with the type of interpolation used.

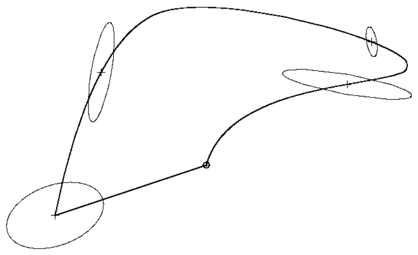


Figure 14.12: *Discrete anisotropic size specification. The information about directions and sizes is known only at specific points.*

14.4 Re-meshing algorithm

Curve remeshing is a rather different problem although it is similar to the previously discussed problem. The applications are manifold. Let us mention, in particular, the possibility of simplifying the mesh, optimizing (for a given criterion) or also adapting the mesh, among the possible applications.

The data is then a mesh and a goal to achieve. The aim is thus to modify the mesh so as to match (or to satisfy) this requirement.

The available mesh modification tools are very simple (see Chapter 18, for instance). We can in fact:

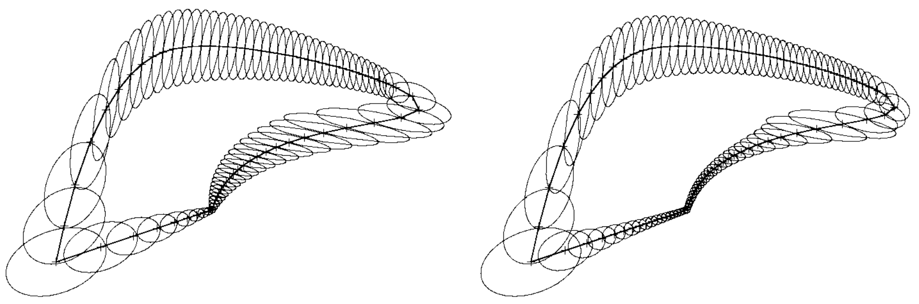


Figure 14.13: *Mesh resulting from a linear interpolation (left-hand side) and a geometric interpolation (right-hand side).*

- add points,
- delete points,
- move points.

The idea is then to use these tools in order to remesh the underlying curve to conform to the fixed objective while preserving, in a certain way, the geometry of this curve. The goal can be expressed by assuming known a metric $\mathcal{M}$ which is continuous or discrete (mesh optimization or adaptation) or a given accuracy (mesh simplification). The immediate question is related to the way in which the geometry of the underlying curve is defined. In fact, the aim is to retrieve (or even to invent) this geometry from the sole data available (the initial mesh).

Discovering the geometry can be achieved in various ways. For instance:

- we identify the presumed singular points (for example, by looking, with respect to a threshold, at the angle variations from element to element),
- we split the initial mesh into pieces where each piece is limited by two such (consecutive) singular points,
- we construct a *geometric support*, for instance, a cubic on each element (of each portion). To this end, we can follow the previously described method,
- this geometric support allows us to directly or indirectly (via a piecewise linear approximation with a polygonal segment as already seen) find the geometric features of the curve (radii of curvature, tangents, normals) or, at least, approached values of these quantities.

We then have a definition of the geometry that can be used to drive the operations required to achieve the fixed goal. We then find a situation similar to that related to the meshing of a curve when a metric specification is given or to an objective formulated in a different way.

Remark 14.8 *Numerous and various difficulties can be expected. Let us mention in particular the problem related to the definition, for a given portion of curve, of a reasonable normal (a tangent) at singular points, assuming the latter have been correctly identified.*

14.5 Curves in $\mathbb{R}^3$

The construction of a mesh for a curve of $\mathbb{R}^3$ may have various forms, depending on whether the curve considered is part of a known surface or not.

Dangling curves

We consider here the curves of $\mathbb{R}^3$ that are not traced on a known surface, which can be used to extract the required information.

Remark 14.9 *Initially, notice that, if we restrict ourselves to two dimensions, the following gives the results previously established.*

We reuse the same reasoning as for the planar curves. The local analysis of the curve corresponds, in the neighborhood of s_0, to the limited expansion:

$$\gamma(s) = \gamma(s_0) + \Delta s\gamma'(s_0) + \frac{\Delta s^2}{2}\gamma''(s_0) + \frac{\Delta s^3}{6}\gamma'''(s_0) + \dots,$$

with $\Delta s = s - s_0$ being sufficiently small to ensure the validity of the development. We still have $\gamma'(s) = \vec{\tau}(s)$ and $\gamma''(s) = \dfrac{\vec{\nu}(s)}{\rho(s)}$ while now:

$$\gamma'''(s) = -\frac{1}{\rho(s)^2}\left(\rho'(s)\,\vec{\nu}(s) \,+\, \vec{\tau}(s) \,+\, \frac{\rho(s)}{R_T(s)}\,\vec{b}(s)\right),$$

where $\vec{b}(s)$ is the binormal vector at s (Chapter 11) and $R_T(s)$ is the radius of torsion of the curve at s. Hence, we have:

$$\gamma(s) = \gamma(s_0) + \Delta s\,\vec{\tau} + \frac{\Delta s^2}{2\,\rho(s_0)}\,\vec{\nu} - \frac{\Delta s^3}{6\,\rho(s_0)^2}(\rho'(s_0)\,\vec{\nu} \,+\, \vec{\tau} + \frac{\rho(s_0)}{R_T(s_0)}\,\vec{b}) + \dots, \qquad (14.14)$$

where $\vec{\tau} = \vec{\tau}(s_0)$, $\vec{\nu} = \vec{\nu}(s_0)$ and $\vec{b} = \vec{b}(s_0)$. The behavior of the curve if that of the parabola (the first three terms of the development) if:

$$\frac{\Delta s^3}{6\,\rho(s_0)^2}\left(\rho'(s_0)\,\vec{\nu} \,+\, \vec{\tau} + \frac{\rho(s_0)}{R_T(s_0)}\,\vec{b}\right), \qquad (14.15)$$

which involves the variation of ρ (i.e., the value ρ') and the radius of torsion R_T, is small before the previous terms. If the torsion is zero (the radius of torsion is infinite) the problem is then a planar problem and we return to the discussion regarding planar curves. On the other hand, this implies, for a given ε, a condition similar to Condition (14.7), which can be written as:

$$\alpha \approx \frac{\sqrt{6\,\varepsilon}}{\sqrt[4]{1+\rho'(s_0)^2+\left(\dfrac{\rho(s_0)}{R_T(s_0)}\right)^2}}\,. \tag{14.16}$$

The gap to the parabola is then of the following form:

$$\delta \approx \frac{\varepsilon\sqrt{6\,\varepsilon}}{\sqrt[4]{1+\rho'(s_0)^2+\left(\dfrac{\rho(s_0)}{T(s_0)}\right)^2}}\,\rho(s_0)\,. \tag{14.17}$$

If the torsion is sufficiently small (the radius of torsion T is large), we can neglect the gap according to $\vec{b}$ to the plane $[\vec{\tau},\vec{\nu}]$ and, locally, use the meshing method in the plane. In the opposite case, the parabola of the plane $[\vec{\tau},\vec{\nu}]$ is not a good approximation of the curve and the analysis must be extended to the next term so as to account for the torsion. Thus, we approach the curve by a cubic of $\mathbb{R}^3$ and we control the gap between this cubic and the curve via the following term in the development. It is obvious that the calculations required are very technical and, for that reason, we will go no further into this topic.

Curves of a parametric surface

We distinguish here several types of curves:

- the curves interface between patches,
- the curves traced on a patch,
- the curves images of a segment in parametric space which, approached by a segment, form the edges of the surface mesh elements.

All these points will be dealt with in the next chapter.

Chapter 15

Surface Meshing and Re-meshing

Mesh generation for surfaces representing the boundaries of a three-dimensional domain is, undoubtedly, a crucial point, especially for the computational schemes in finite element methods. As well as this field of application, the construction of surface meshes is also an important feature in applications such as visualization, virtual reality, animation, etc.

In this chapter, we discuss some methods that make it possible to construct the mesh of a segment or a curve traced on a surface, as well as the mesh of a surface itself. As in the previous chapter, there are two possible approaches to dealing with this meshing problem, depending on the definition of the geometric model. In practical terms, the problem is of a different nature, depending on whether a CAD type definition is known or the sole data available corresponds to a discretization (i.e., a polyhedral approximation) of the given surface. The first case leads rather to envisaging the development of appropriate mesh generation methods, while in the second case, mesh modification methods will be preferred.

★
★ ★

First, we briefly discuss curve meshing (already mentioned in Chapter 14), in the particular case where they are traced onto a surface (or along a side of a patch).

Then, by means of examples, we illustrate the different types of surface meshes that can be constructed, *a priori*. These examples allow us to specify the nature of the given problem and to indicate the various possible approaches. We will see that there exists a direct approach (where the surface itself is meshed) and an indirect approach (where a mesh is constructed in a parametric space before being mapped onto the surface).

We have decided to exclude here the case of implicit surfaces (discussed in Chapter 16) and explicit surfaces[1], and focus on the case of a surface defined by one or more patches (i.e., a parametric surface). To this end, we examine first the case of a single patch, before considering composite surfaces. In the latter situation, we discuss a possible alternative, depending on whether the resulting mesh is *patch-dependent* or not. Specific cases where the parametric space is defined in terms of a constructive method are discussed via the example of molecular surfaces. Then we add some comments about reconstruction methods where the data is a cloud of points. Finally, to end the chapter, we change data types to focus on surfaces defined in a discrete way, i.e., from a mesh. We study an approach that tends to provide an adequate mesh of the given surface, based on a series of modifications of the initial mesh, which forms the sole data at hand.

All that has been mentioned, except the last case, assumes that the patches defining the surface are correct (in particular, that global conformity between patches is ensured, and that there are no overlaps, no holes, no folds or any other undesirable artifact). We also deal with what can be termed a *true problem.* In practice, the surfaces to be meshed are usually not suitably defined with regard to the desired objective. Geometric modeling systems (CAD) allow us to define composite surfaces from a (bottom-up or top-down) analysis for which the constraints (imposed by the manufacturing or visualization process) are rather different from those required by the meshing techniques. Among the divergences, let us mention the problem related to the numerical accuracy of the models.

15.1 Curve meshing (curve member of a surface)

The curves that are of interest here are essentially the curves interface between two patches and/or the curves traced on a surface.

The different methods for curve meshing have been reviewed in Chapter 14 and the reader is referred to this chapter. In this section, we will simply make some practical remarks about the specificities of this problem, i.e., when we consider that the curves are part of a surface.

We should immediately point out that a classical meshing technique for curves in $\mathbb{R}^3$ (such as described in Chapter 14) is still valid here, provided that care is taken with:

- the local curvatures of the surface (in the vicinity of the curve) and/or,
- potential specifications and, obviously,
- the envisaged application.

Indeed, from the strictly geometric point of view, the mesh of a straight segment requires only one element. However, this straight segment is a (potential) edge of a surface mesh triangle that forms a polyhedral approximation of the surface. Hence,

[1]Such a surface can always be expressed in a parametric form (as indicated in Chapter 11), thus making it possible, in principle, to apply the techniques described in this chapter.

the edge lengths are proportional to the principal radii of curvature of the surface. In the isotropic case, the construction of an equilateral element then needs to bound the mesh element size. This operation can be performed by considering, not only the intrinsic properties of the curve, but also those of its close neighborhood. In the anisotropic case, the reasoning is quite similar, except that this time the classical equilateral (Euclidean) element is not longer the aim.

When the curve considered is an interface curve between two patches (a boundary curve), the meshing problem is rather similar. The desired edge size of the discretization also depends on the local curvatures of the curve and of its two (or more) adjacent surfaces.

15.2 First steps in surface meshing

In this section, we try to give a global view of surface mesh generation and we make some remarks about this interesting problem. To this end, we start by introducing several examples of surface meshes obtained using different mesh generation techniques[2]. This will then allow us to state the expected properties of such meshes. Finally, we introduce various possible solutions (the main techniques employed being described in detail later).

Various approaches to surface meshing

Before going further in the discussion of the principle, we will illustrate, on a given geometry[3], several examples that are representative of different mesh related aspects (Figures 15.1 and 15.2).

Surface mesh by example. Let us focus first on structured mesh generation. The mesh in Figure 15.1 (left-hand side) was obtained using a multi-block type method, the different patches having been meshed using a transfinite interpolation on a quadrilateral (Chapter 4). In such an approach, the geometry of the surface is simply governed by the geometry of the four sides of each patch. Therefore, the surface mesh is defined only by the discretization of the boundary edges (composed here by 10 segments).

The mesh in Figure 15.1 (right-hand side) was generated by mapping a regular grid (a 10×10 structured grid) onto the surface. Hence, in contrast to the previous case, all resulting mesh vertices belong to this surface.

Let us now turn to unstructured meshing techniques based on the use of parametric spaces associated with the patches. The mesh in Figure 15.2(i) was constructed by mapping onto the surface uniform isotropic meshes in the parametric spaces.

[2]The aim here is not to examine the different approaches used in detail, but rather to point out the key problems in surface mesh generation.

[3]The perspicacious reader may have recognized the spout of the well-known *Utah teapot*, composed of four parametric patches.

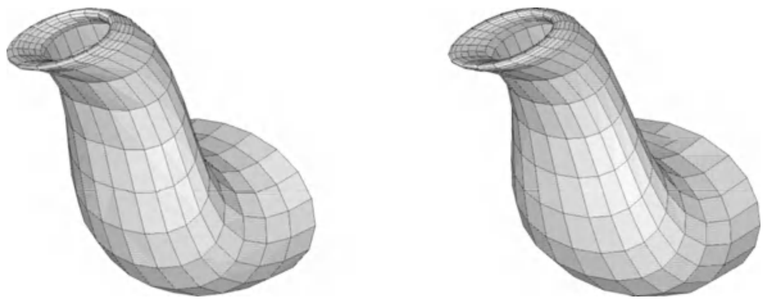


Figure 15.1: *Multiblock-method based on a local transfinite interpolation (left-hand side). Naive parametric mesh, uniform grid, and projection onto the surface using the surface definition itself (right-hand side).*

The mesh in Figure 15.2(ii) was obtained by mapping onto the surface anisotropic meshes in the parametric spaces. These meshes were constructed so as to ensure that the resulting mesh, after mapping, will be uniform.

The mesh in Figure 15.2(iii) was obtained by mapping onto the surface the anisotropic meshes of the parametric spaces, controlled by the minimal radius of curvature (so as to bound the angles between the mesh edges and the tangent planes to the surface).

The mesh in Figure 15.2(iv) was obtained by mapping the anisotropic meshes of the parametric spaces, controlled by the two principal radii of curvature.

The mesh in Figure 15.2(v) was obtained by mapping the anisotropic meshes in the parametric spaces, controlled this time by bounding the ratio between the two principal radii of curvature.

Finally, the mesh in Figure 15.2(vi) was obtained in a similar fashion, by more subtly controlling the use of the two radii of curvature.

It is thus easy to realize that numerous possibilities exist to obtain a surface mesh, some of which may be more relevant than others. Therefore, the question arises: *"What is really needed and what has to be done to get it?"*

We will now try to answer this tedious and haunting question.

Desired properties

As can be easily imagined, the desired surface mesh must verify certain properties. Obviously, or rather *a priori* (as will be seen later), the vertices of the mesh elements must belong to the surface (that is, to the geometry, defined in one way or another).

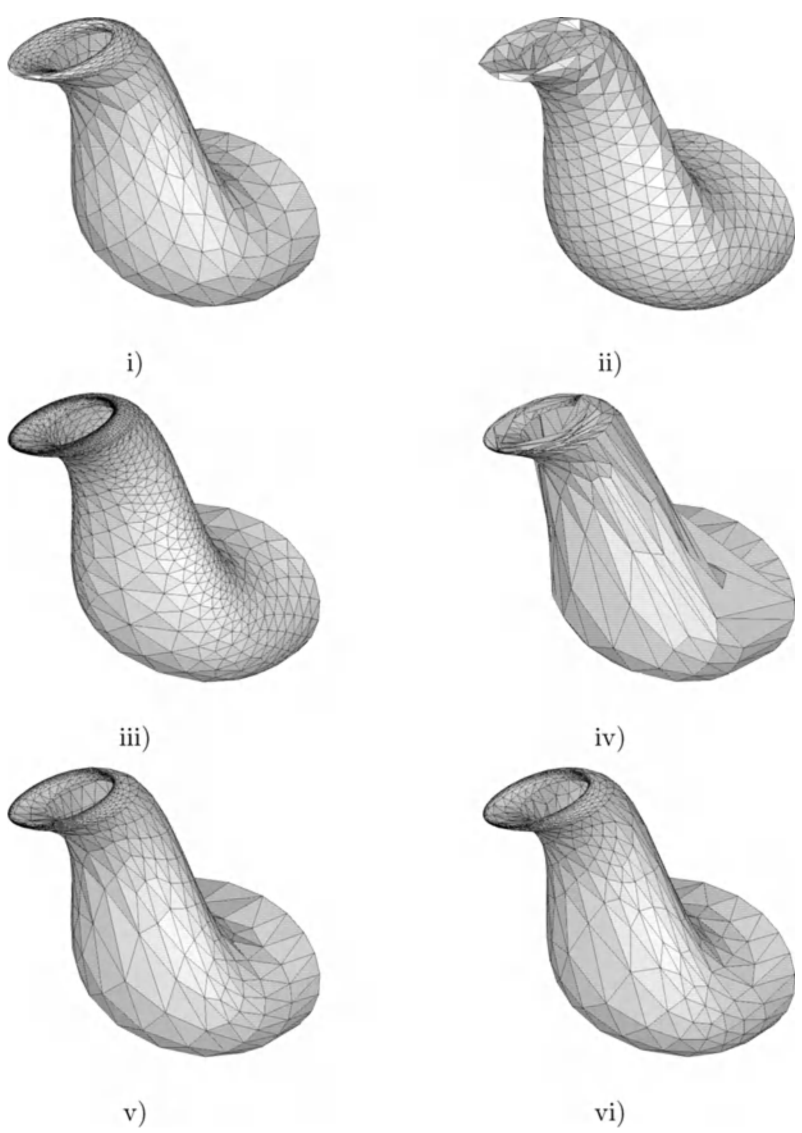


Figure 15.2: *Unstructured meshes: i) uniform unstructured mesh in the parametric space and mapping, ii) unstructured mesh projected onto the surface to obtain a uniform mesh at this level, iii) isotropic mesh based on the minimum of the radii of curvature, iv) anisotropic mesh without a stretching limit, v) anisotropic mesh with a (user-specified) control on the ratio between the two principal radii of curvature, vi) anisotropic mesh with an automatic adjustment of the maximal ratio between the two principal radii of curvature.*

Let Σ be the given surface and let P, Q, R, ..., be the vertices of mesh $\mathcal{T}$. Thus, we want to have:

$$\forall P \in \mathcal{T} \implies P \in \Sigma\,, \tag{15.1}$$

As certain methods do not guarantee this feature (for all vertices in $\mathcal{T}$), we can simply require that the minimal distance between any point P and Σ is as small as possible. In this case, we introduce an accuracy threshold ε that controls this gap.

Nevertheless, this feature is only a usually desired necessary condition (cf. Chapter 14) that does not, however, guarantee that the resulting mesh will be satisfactory. Hence, a stronger constraint can be expressed. For instance, we can require all mesh edges to be as close to the true surface as possible. In this case, for each edge PQ of $\mathcal{T}$, we want to have:

$$d(PQ, \Sigma) \leq \varepsilon\,, \tag{15.2}$$

where ε represents the desired tolerance value and $d(PQ, \Sigma)$ is the maximal distance (the gap) between the edge PQ and the surface Σ.

Pursuing the analysis, it appears that Property (15.2) is a necessary although not a sufficient condition. The following property comes to reinforce this condition:

$$d(PQR, \Sigma) \leq \varepsilon\,, \tag{15.3}$$

for each triple of points (P, Q, R), vertices of a triangle of $\mathcal{T}$ (simplicial mesh). This relation indicates that the distance (the gap) between any triangle PQR and the surface Σ must be bounded. For quadrilateral (or hybrid) meshes, we have a similar property.

Finally, the surface mesh must have the same aspect as the surface (i.e., it must reflect the geometry). Hence, if the surface is locally of order C^1 or G^1, the mesh must represent this feature as closely as possible (possible oscillations must be avoided in this case). Similarly, discontinuities (singular points, ridges or crest lines) must be present in the resulting mesh. These requirements can be formulated more concisely as follows:

$$\texttt{Smooth surface mesh.} \tag{15.4}$$

Exercise 15.1 *Find some examples where Property (15.1) holds while Property (15.2) is not satisfied. Similarly, discuss the relation between the two other properties.*

Remark 15.1 *Note that the above properties concern only the geometrical point of view of the surface. In the case where element sizes (densities, directions, etc.) are specified, this constraint must be coupled, in some sense, with the previous ones.*

To conclude, the problem here is to express the desired properties in terms of requirements that are usable by the mesh generation method considered. Thus, it is easy to imagine that a suitable method is one that can be controlled up to a certain level (via a metric field).

Direct surface meshing

A *direct* mesh generation method is a method acting directly at the level of the surface considered (without using, for example, a parametric space). Among the most commonly used techniques, we find some of the methods already discussed in the general chapters.

Method related to a specific definition. When the surface is defined in a particular way, for example as a *sweeping surface*, it is possible to take advantage of this feature to construct the surface mesh. Thus, if the surface is constructed by sweeping a curve Γ_1 along a curve Γ_2, a mesh can be easily obtained by sweeping a discretization of Γ_1 along a discretization of Γ_2 (or vice versa).

This type of method is obviously limited in its range of application and, moreover, does not allow the quality of the geometric approximation (the mesh) of the true surface to be effectively controlled. Moreover, special care must be taken regarding potential degeneracies that can appear locally on the resulting surface [Rogers, Adams-1989].

Extension of a two-dimensional method. An algebraic method (when the surface defines a quadrilateral-shaped domain) or an advancing-front type method (Chapter 6) can be extended to process surfaces. This extension feature is less obvious for a Delaunay type method (Chapter 7), which is more adapted to meshing a surface by meshing the associated parametric spaces.

For an algebraic method (Chapters 4 and 13), the resulting surface is strongly related to the generation method. In fact, the geometry of the surface is defined by the method, usually from the mesh of the domain boundaries only. Notice that no explicit control of the gaps (at the level of the edges or faces) between the discretization and the surface can be obtained. Therefore, this type of method is of interest for particular geometries (although the low cost may be considered an advantage). From this point of view, parts of surfaces are frequently well processed by such an approach (a more sophisticated method then being unnecessary, or even too costly, to implement).

The advancing-front approach, or more precisely its extension to surface processing, is more general[4]. This type of approach is also widely used in industrial meshing technologies. The main principle of this method follows, broadly speaking, the classical scheme of an advancing-front method in two dimensions ([Löhner, Parikh-1988], [Nakahashi, Sharov-1995], etc.):

- identify the key features of the boundary of the patch considered. These points are those making it possible to define a set of almost regular lines along the side of the patch. We then naturally retrieve the singular points of the surface among those points,

[4]Notice by the way that it is common to find, when dealing with the extension of a known method, the assertion that this operation is both natural and easy. In practice, this judgment must be strongly qualified, the difficulties encountered not being of the same order.

- define a mesh vertex for each of these points and identify the set of lines joining two such vertices,
- mesh these lines (refer to the different curve meshing techniques discussed previously),
- connect the meshed lines together so as to define closed curves[5],
- use in each of the parts thus-defined a classical advancing-front approach. From a front edge (the initial front being formed by the discretized lines of the contour), construct (or choose) a candidate point on the surface to form an equilateral triangle. Several ways exist to construct (choose) a candidate point. The simplest idea is to consider, for a given front edge, an average tangent plane (Π) in the middle of this edge (for example, by considering the tangent planes at its endpoints). An ideal point is defined in this plane at a distance that makes it possible to create an equilateral triangle, and then projected onto the surface, the corresponding triangle then being formed. The front is updated,
- repeat the previous stage, as long as the front is not empty.

Figure 15.3 illustrates the main steps of this method: i) characteristic points are identified, ii) the contour of the patch is constructed and then meshed, iii) a unique contour is defined and iv) an optimal triangle is formed.

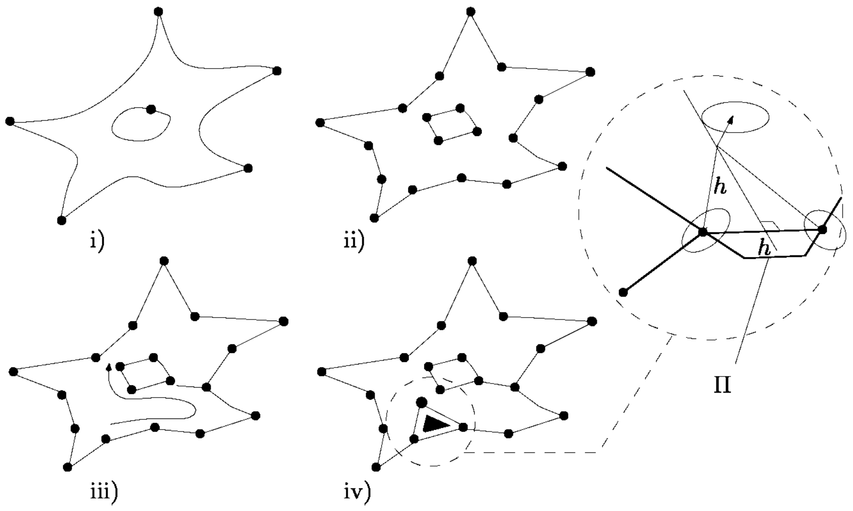


Figure 15.3: *Main steps of an advancing-front type strategy.*

The proposed scheme, while being conceptually simple, hides however certain difficulties. Such difficulties exist in two dimensions, up to a certain point, but

[5]This point, which is unnecessary in two dimensions, is a source of substantial simplification here.

they are relatively easy to overcome. An immediate example where such a problem arises is the case where the candidate point is located outside the domain. In two dimensions, this means that there must be an intersection between the edges incident to that point (forming a new triangle) and one or more edges of the current mesh. As such, a simple intersection test allows us to detect and to avoid this configuration.

It is more or less obvious that, for a surface, the same situation does not necessarily lead to an intersection. Therefore, the detection of this situation requires the use of more complex data structures (a neighborhood space, in three dimensions).

The case where the projection of the point considered (supposed optimal) on the true surface does not exist, meaning that this point is probably located straight down a hole, outside the surface or even leads to a topological ambiguity (for example, two neighboring surfaces exist and it is not possible to decide which one is the surface on which the point must lie).

In practice, provided the validity of the point creation process is ensured, as well as its valid connection with an edge, it is usual to conduct this creation stage iteratively. An initial optimal point is defined *a priori* and the corresponding solution is analyzed to possibly optimize the location. In other words, the process consists of using the additional information now available, about the point considered.

The topological and geometrical difficulties supposedly being resolved, notice that the proposed method must, however, incorporate a control on the accuracy of the approximation of the surface obtained. This control[6] can be performed by looking at the gap between a mesh edge and the surface (control of the interpolation error between a segment and a curve) and, more precisely, by controlling the gap between a mesh triangle and the surface (control of the interpolation error between a triangle and a surface; this control cannot be restricted to the sole control of the triangles edges). We propose here re-examining the ideas developed in Chapter 10, which make it possible, using the second derivatives (the Hessian), to obtain such a control.

Remark 15.2 *The coupling between a direct approach and a parametric space offer additional information that makes it possible to overcome some of the difficulties mentioned previously [Frykestig-1994].*

Octree-type method. We have seen (Chapter 5) that an approach based on a spatial decomposition of the domain of interest makes it possible to generate a mesh of this domain. This type of approach also allows us to construct a mesh of the domain boundary if the discretization of the surface is not a data of the problem. We will now examine this last feature.

Given a bounding box of the domain, it is possible to generate directly a mesh of the surface, without using a parametric space (cf. [Grice *et al.* 1988] and [Shephard, Georges-1991]). The construction of the decomposition tree follows the general classical scheme already described, with only a few slight modifications (in

[6]And this will be true for the whole range of methods discussed in this chapter.

fact numerous variants exist). The basic idea consists of inserting in the tree the entities of the geometric model[7] in the increasing order of their dimensions. More precisely, the construction scheme includes the following stages:

- identifying corners and points of discontinuity of the model,
- inserting these points in the current tree, the stopping criterion for subdivision being unchanged (i.e., at most one point per cell),
- then, for each terminal cell, using the local properties of the surface in order to decide on the possible refinement of the cell. Recall that the element size is related to the cell size, the latter also being related to the local curvature of the surface (which can be estimated at a finite number of points in the cell by sampling),
- and, when all cells have a size compatible with the given size map (*intrinsic* or specified), the critical point consists of determining the interactions between the tree cells and the geometric model, which can lead to refining the cells.

The analysis of the interactions between a cell and the geometric model involves calculating intersections. In fact, it is necessary to determine exactly (i.e., to the required accuracy) the intersections of corners, edges and faces of the octants with the surface [Kela-1989]. The intersection points are then used to create edges (connecting to such points) and loops (joining several edges and forming a closed contour) in each terminal cell. These entities will provide the vertices, edges and faces (possibly subdivided into triangles) of the final mesh. When the result of these tests is not known (if the intersection points are not returned by the modeler) or if this result is ambiguous, more costly operations may be used.

Before discussing the meshing stage itself, a balancing procedure is applied on the tree structure, using the [2:1] rule (limiting the difference between neighboring cells to 2 levels).

The creation of the mesh elements is performed as in the classical case. Here, however, only the boundary cells are of interest. There is no *a priori* limit on the geometric complexity of the entities contained in a tree cell. In practice, several criteria allow a cell to be refined when the complexity becomes too great to handle. Notice that meshing the portion of surface enclosed within a cell is an operation which is approximatively of the same order of complexity as a classical automatic meshing technique. More details on creating mesh elements in boundary cells can be found in [Shephard *et al.* 1988b] and [Shephard, Georges-1991], among others.

The main difficulties associated with the octree approach are related to:

- the creation of badly shaped elements (the intersections between the octants and the surface are not well handled),
- the topological ambiguity problems mentioned above that can compromise the meshing process of boundary octants.

[7]Instead of the entities of the boundary discretization.

Figure 15.4 shows an example of a surface mesh obtained using a direct octree-based method. Notice that the resulting mesh has not been optimized, the alignment of vertices clearly denotes the octants surface interactions.

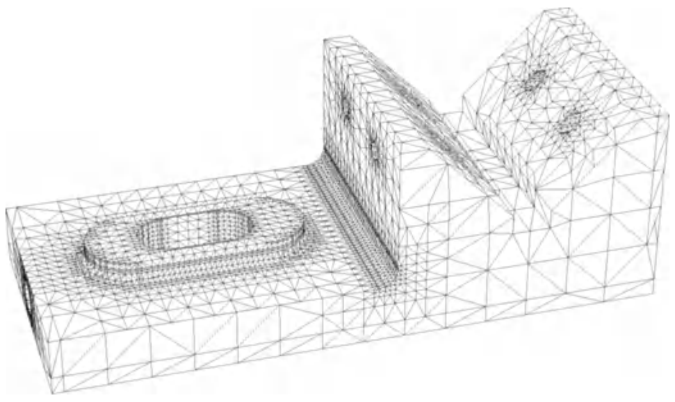


Figure 15.4: *Example of a surface mesh generated using a direct octree-based approach, without optimization (data courtesy of MacNeal-Schwendler Corp.).*

Other methods. Among the other possible methods, we find those methods based on grid mapping and all methods based on iso-surfaces (implicit surfaces, *Marching-cubes* type algorithms, voxels, etc., Chapter 16) and the methods for which the input data is a cloud of points located on the surface (for example, see below or [Hoppe *et al.* 1991]). In this section, we only give some idea of the approaches based on the mapping of a predefined grid (which are indeed close to certain methods discussed in Chapter 8).

- Grid-mapping methods.

The principle underlying these methods is very simple. We define a grid (a regular mesh) in a plane located above the surface, in such a way that the projection of this grid, in a suitable direction $\vec{d}$, entirely encloses the surface. Supplied with this assumption, we analyze the projection of the grid cells on the surface in the direction $\vec{d}$. To this end, we examine (Figure 15.5) the projection of the grid nodes:

- if the direction line $\vec{d}$ coming from a node intersects the surface, the intersection point is kept as a mesh vertex,
- otherwise, we find mainly two configurations: either the projection goes through a hole (a loop of vertices exist in some neighborhood of the projected point that belongs to the previous class) or the projection falls outside the surface (an external side of the surface exists in some neighborhood).

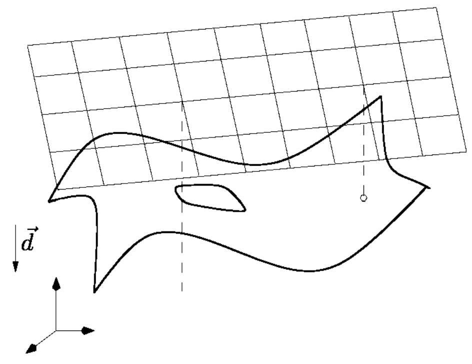


Figure 15.5: *Projection of a grid onto the surface. Two cases are visible: the projection is found (right-hand side) or the projection does not exist (left-hand side) because of a hole. The third case, where the projection is outside the surface, is not described here.*

Notice that this approach makes the mesh independent of the patches describing the surface. However, it is obvious that the main drawback of the approach is related to projection problems (independently of the fact that the approximation of the surface by the mesh is not controlled and that, in addition, a uniform grid is always used).

In such an approach, several problems must be taken into account. Among the pertinent questions that arise, let us mention the following ones:

- How should the plane support of the grid (with respect to the surface) be chosen?
- Is such a plane always defined (bijectivity of the projection)?
- What are the potential perverse effects following a bad choice of the plane?
- What is the (possible) dependency between this choice and the presumed result?

Assuming these questions to be settled, several points need to be discussed.

The first obvious observation is that if the four nodes of a cell have a projection on the surface, then the resulting quadrilateral so formed is a candidate mesh element. On the other hand, any different situation requires a more subtle discussion, as will be seen later.

If the four projections of a cell vertices exist, it is possible to form a quadrilateral. This element needs to be validated as a mesh element. This consists essentially of finding if this quadrilateral does not mask a hole (whose size is necessarily smaller than the grid stepsize) or does not overlap a "gap" between two portions of the surface (connected component problem or topology violation). In other words, we must verify the topological compatibility of the mesh, i.e., preserve the surface topology.

The case where one or more vertices do not project onto the surface corresponds to the existence of a portion of boundary (internal, a hole, or external, an external side) in a neighborhood whose size can be determined by looking at the projections of the adjacent nodes. An analysis of these cases is needed in order to find the corresponding configuration (or, at least, a plausible configuration from the point of view of topological compatibility):

- the existence of a projection defect for a point, although there exists a loop of its neighbors that projects without difficulty, presumes the existence of a hole whose size is related to this loop. Provided this configuration is identified, it is possible to find a mesh that also includes a hole (in fact, a potentially rough approximation of the true hole) that therefore maintains the topology of the surface. To this end, we first detect an approximate side of this hole and we modify the neighboring elements so as to account for;

- the same phenomenon without a suitable loop of neighbors representative of the existence of a surface border. As above, it is possible (although tedious) to find an approximation of this border preserving the initial topology.

These remarks indicate the complexity of the problems to be taken into account and suggest that, in order to be applicable, this type of method must be coupled with rather complex techniques (adaptation, i.e., local refinement, quick search for intersection, verification of the topological consistency of the surface, etc.).

Hence, to conclude, this type of method is only really applicable for relatively simple surfaces (low curvature, no hole and no topological ambiguity).

Construction in two dimensions and mapping

Unlike the methods mentioned above, here the mesh generation method is applied in an adequate $\mathbb{R}^2$ domain, then, via a projection function, the mesh constructed in this domain is projected onto the $\mathbb{R}^3$ surface. Three types of method may be considered, which is the so-called "classical" methods (*quadtree*, advancing-front and Delaunay). In this context, the problem to be solved consists of governing the mesh generation in the one or the several parametric space(s) in order to guarantee the resulting mesh to be adequate after being projected onto the surface. In the following, we discuss more specifically the Delaunay approach while observing that the other classical methods can also be used (at least, if the mesh in the parametric space is isotropic, a classical limit for the *quadtree* type methods).

15.3 A single patch

We focus first on a surface composed of a single patch. We will deal later with surfaces composed of several patches, for which two situations need to be considered, the *patch-dependent* case where a discretization of the patch boundaries is necessarily present in the final mesh, and the *patch-independent* case where the inter-patch boundaries are not necessarily present in the resulting mesh.

•Typical patches

Before dealing with the single patch case, let us recall the various types of patches used in CAD systems. The simplest patches and ones which are widely used are polygons with a small number of sides (thus, we usually find triangles and quadrilaterals). However, this type of patch does not offer the flexibility necessary and desirable to define certain surfaces. This is why more complex patches have also been developed, for example, those where only part of the patch is used (hole(s)) and those where the useful area does not correspond to the whole patch (the border limiting the useful area is not the "natural" side of the patch).

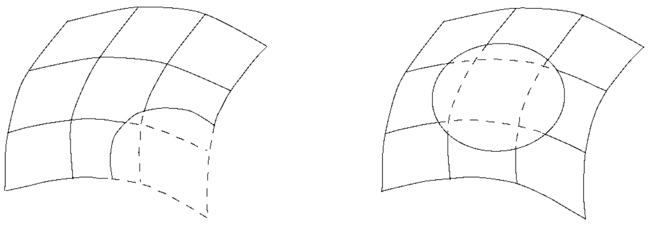


Figure 15.6: *Two examples of complex patches: left-hand side, only a part of the patch is used; right-hand side, a hole exists in the surface.*

For the sake of simplicity, we can, at first, leave to one side all these special patches and deal with complete patches. Then, it is relatively easy to see what may happen with the other types of patches.

•Desired features in the parametric case

Let Σ denote the surface, σ its parameterization and Ω its parametric space ($\Omega \subset \mathbb{R}^2$). Let us denote by A, B, ... the points of Ω and by P, Q, ... their images via σ. As stated previously, the surface mesh must satisfy certain properties (Relations (15.1) *et seq.*). Relation (15.1) simply means that:

$$A = (u, v) \in \Omega \quad \Longrightarrow \quad P = \sigma(A) = \sigma(u, v) \in \Sigma\,, \tag{15.5}$$

which is ensured if σ is the exact definition of Σ. Otherwise, the distance from P to Σ must be smaller than a suitable threshold ε (relative or absolute). Relation (15.2) indicates that the edges are close to the surface:

$$AB \in \Omega \quad \Longrightarrow \quad PQ = \sigma(A)\sigma(B) \text{ is such that } d(PQ, \Sigma) \leq \varepsilon\,, \tag{15.6}$$

where ε is a given threshold of accuracy.

Remark 15.3 *Saying that $PQ = \sigma(A)\sigma(B)$ is only an approximation; indeed, $\sigma(AB)$ is* a priori *a curve traced on Σ and, in particular, $\sigma(AB) \neq \sigma(A)\sigma(B)$.*

Similarly, Relation (15.3) becomes:

$$ABC \in \Omega \Longrightarrow PQR = \sigma(A)\sigma(B)\sigma(C) \text{ is such that } d(PQR, \Sigma) \leq \varepsilon\,, \tag{15.7}$$

hence, the distance from triangle PQR, the image[8] of the triangle ABC, to the surface Σ is bounded.

Finally, Relation (15.4) must be verified, in particular to preserve, via its mesh, the possible regularity of the underlying surface.

The problem is then, if possible, to express these requirements in mesh requirements in the parametric space. To this end, we return to the basic definitions of parametric patches (Chapters 11 and 13), to the notion of metric (Chapter 10) and to that of control space (Chapter 1).

Briefly, and before going into further detail, let us recall that the edge length in $\mathbb{R}^3$ can be expressed as a function of the corresponding edge AB in $\mathbb{R}^2$ and of the first fundamental form of Σ. Similarly, the control of the distance between an edge and Σ can be done by using the second fundamental form of the surface. On the other hand, the control of the distance from a triangle to Σ requires a more subtle analysis.

•Meshing a patch

As already mentioned, any meshing technique in the plane is *a priori* a conceivable method. However, we will restrict ourselves to a Delaunay-type method (Chapter 7). We have seen in fact that this type of method could rather easily follow a given field of constraints (of specifications). The idea is to govern the mesh in the parametric space in such a way that the desired properties are (automatically) satisfied, after mapping on the surface.

Geometric aspects. Thus, we want to control the surface mesh by controlling the mesh of the patch in the parametric space. At first, notice that the control in the parametric space is essentially performed by controlling the lengths of the mesh edges. We must then evaluate the length of a mesh edge. To find this value, we go back to the definition of the length of a curve, then we will improve this by considering that this curve is an edge.

The context is illustrated in Figure 15.7. The surface is denoted Σ, its parametric space (u, v) is Ω. Moreover, we consider an interval $[a, b]$ in $\mathbb{R}$. The curve PQ of the surface Σ is the image by a function γ of the interval $[a, b]$ in $\mathbb{R}$. This curve is also the image by the function σ of a curve AB of Ω and this curve AB is the image by a function ω of the interval $[a, b]$. We then have:

$$\sigma : \Omega \longrightarrow \mathbb{R}^3,\ (u,v) \longmapsto \sigma(u,v)\,,$$

$$\gamma : [a,b] \longrightarrow \mathbb{R}^3,\ t \longmapsto \gamma(t)\,,$$

$$\omega : [a,b] \longrightarrow \Omega,\ t \longmapsto \omega(t)\,,$$

with the relation linking γ, ω and σ: $\gamma = \sigma \circ \omega$.

[8]In fact, the triangle PQR is the triangle whose vertices are the images of the corresponding vertices in the parametric space (see the remark about the edges).

The length of the curve PQ, Chapter 11, can be written as:

$$l(PQ) = \int_a^b \|\gamma'(t)\|\, dt = \int_a^b \sqrt{\langle \gamma'(t), \gamma'(t) \rangle}\, dt\,.$$

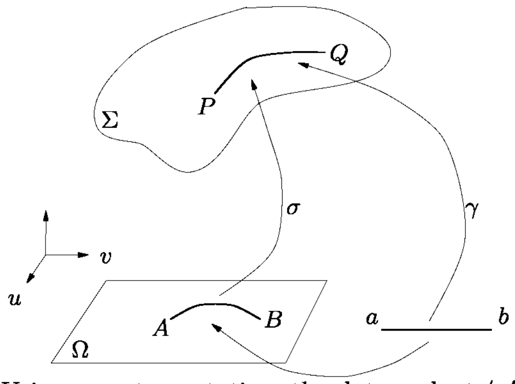


Figure 15.7: *The segment $[a,b]$ in $\mathbb{R}$. The curve AB of Ω, image of $[a,b]$ by ω. The curve PQ of Σ, image of ab by γ is also the image of AB by σ.*

Using a vector notation, the dot product $\langle \gamma'(t), \gamma'(t) \rangle$, is expressed by:

$$\langle \gamma'(t), \gamma'(t) \rangle = {}^t\gamma'(t)\gamma'(t) = {}^t\omega'(t){}^t\sigma'(\omega(t))\sigma'(\omega(t))\omega'(t)\,,$$

as $\gamma = \sigma \circ \omega$. That is, by denoting $\vec{\tau_1} = \sigma'_u$ and $\vec{\tau_2} = \sigma'_v$,

$$\langle \gamma'(t), \gamma'(t) \rangle = {}^t\omega'(t) \begin{pmatrix} \vec{\tau_1} \\ \vec{\tau_2} \end{pmatrix} (\vec{\tau_1}\, \vec{\tau_2})\, \omega'(t)\,.$$

If we now assume that $\omega(t)$, for t between $a = 0$ and $b = 1$, is an edge, the edge AB, we have $\omega(t) = A + t\overrightarrow{AB}$ and $\omega'(t) = \overrightarrow{AB}$ and, hence, we find:

$$\langle \gamma'(t), \gamma'(t) \rangle = {}^t\overrightarrow{AB} \begin{pmatrix} \vec{\tau_1} \\ \vec{\tau_2} \end{pmatrix} (\vec{\tau_1}\, \vec{\tau_2})\, \overrightarrow{AB}$$

and, for PQ, the length is:

$$l(PQ) = \int_0^1 \sqrt{{}^t\overrightarrow{AB} \begin{pmatrix} \vec{\tau_1} \\ \vec{\tau_2} \end{pmatrix} (\vec{\tau_1}\, \vec{\tau_2})\, \overrightarrow{AB} dt}\,,$$

which is also the classical formula: $l(PQ) = \int_0^1 \sqrt{{}^t\overrightarrow{AB}\mathcal{M}_1(A + t\overrightarrow{AB})\overrightarrow{AB}}\, dt\,.$

where $\mathcal{M}_1$ is the matrix expression of the first fundamental form of the surface Σ. We have thus found the link between the length of an edge AB in the parametric space and the length of the curve of Σ, image by σ of this edge.

In our case, during the meshing stage, the image of an edge of Ω will be an edge of Σ. In fact, this is equivalent to approaching the *curve* PQ by the *edge* PQ, that is to have:

$$\sigma(AB) \approx \sigma(A)\sigma(B)\,,$$

which can be also written differently by saying $\sigma(A+t\overrightarrow{AB}) \approx P+t\overrightarrow{PQ}$, thus saying that:

$$||\overrightarrow{PQ}|| \approx l(PQ) = \int_0^1 \sqrt{{}^t\overrightarrow{AB}\mathcal{M}_1(A+t\overrightarrow{AB})\overrightarrow{AB}}\,dt\,,$$

where PQ is an edge and, thus, we have found a link between the length of an edge AB in the parametric space and the expected approximate value of the length of an edge PQ of Σ. This link allows us to control the mesh on Σ by controlling the mesh of Ω.

We now deal with the case where a metric is specified on the surface. Let $\mathcal{M}_3$ be the associated current matrix. This matrix is a 3×3 matrix (hence the index $_3$). The problem is then to find the relation between a length on Σ and the corresponding length in Ω, which is equivalent to exhibiting the matrix $\mathcal{M}_2$, a 2×2 matrix, relative to $\mathcal{M}_3$.

In principle, it is sufficient to follow the same reasoning as previously, simply changing the definition of the dot product (Chapter 10). We now have:

$$\langle \gamma'(t), \gamma'(t)\rangle = {}^t\gamma'(t)\mathcal{M}_3\gamma'(t)\,,$$

or:

$$\langle \gamma'(t), \gamma'(t)\rangle = {}^t\omega'(t) \left\lfloor \begin{pmatrix} \vec{\tau}_1 \\ \vec{\tau}_2 \\ \vec{\nu} \end{pmatrix} \mathcal{M}_3\,(\vec{\tau}_1\,\vec{\tau}_2\,\vec{\nu}) \right\rfloor_2 \omega'(t)\,,$$

with $\vec{\nu}$ the unit normal and the notation $\lfloor\ \rfloor_2$ that indicates that only the first two lines and two columns of the matrix are considered. By denoting Π the transition matrix from the canonical basis of $\mathbb{R}^3$ to the local basis at the current point $M = P + t\overrightarrow{PQ}$, this relation can be written:

$$< \gamma'(t), \gamma'(t) >= {}^t\omega'(t)\,\lfloor {}^t\Pi\,\mathcal{M}_3\,\Pi \rfloor_2\,\omega'(t)\,.$$

Posing: $\mathcal{M}_2 = \lfloor {}^t\Pi\mathcal{M}_3\Pi \rfloor_2$, which is, as mentioned, the matrix formed by the first two lines and two columns of the matrix ${}^t\Pi\mathcal{M}_3\Pi$. Thus, we have found the matrix $\mathcal{M}_2$ corresponding to the given matrix $\mathcal{M}_3$. The length of the curve PQ for the metric $\mathcal{M}_3$ is then:

$$||\overrightarrow{PQ}|| \approx l_{\mathcal{M}_3}(PQ) = \int_0^1 \sqrt{{}^t\overrightarrow{AB}\mathcal{M}_2(A+t\overrightarrow{AB})\overrightarrow{AB}}dt\,.$$

To construct a mesh, we approach the length of the edge PQ by the length of the curve PQ and we say that $PQ = P + t\overrightarrow{PQ}$, then, we want to have as above:

$$l_{\mathcal{M}_3}(PQ) = \int_0^1 \sqrt{{}^t\overrightarrow{AB}\mathcal{M}_2(A+t\overrightarrow{AB})\overrightarrow{AB}}dt\,.$$

Remark 15.4 *By taking $\mathcal{M}_3 = I_d$, we find $\mathcal{M}_2 = \lfloor {}^t\Pi\, I_d\, \Pi \rfloor_2 = \lfloor {}^t\Pi\, \Pi \rfloor_2 = \mathcal{M}_1$, the first fundamental form of Σ. We then find the classical case as a particular case of the general situation.*

This leads to the following remark.

Remark 15.5 *A Euclidean length of 1 in Σ corresponds to the choice $\mathcal{M}_2 = \mathcal{M}_1$ for Ω. In other words, the unit circle of Σ corresponds to the ellipse $\mathcal{M}_1$ of Ω.*

The control of the gap between an edge and the surface is performed by using a metric $\mathcal{M}_3$ that still needs to be defined. To govern the surface mesh, the mesh of Σ when a metric $\mathcal{M}_3$ is specified, we must then govern the mesh of Ω for a metric $\mathcal{M}_2$ (2×2 matrix) constructed as indicated above. This control will allow us, depending on the choice of $\mathcal{M}_3$, to the edge to satisfy a particular property. For example, as will be seen later, an adequate choice of $\mathcal{M}_3$ will mean that this edge will not be further from the surface than a given threshold value.

From the metric point of view, we consider $\mathcal{M}_3$ and we want to have

$$l(PQ) = 1 \text{ for } \mathcal{M}_3\,,$$

that is:

$$1 = \int_0^1 \sqrt{{}^t\overrightarrow{PQ}\mathcal{M}_3(P + t\overrightarrow{PQ})\overrightarrow{PQ}}dt\,.$$

To this unit value for $\mathcal{M}_3$ on Σ corresponds a unit value for $\mathcal{M}_2$ in Ω:

$$1 = \int_0^1 \sqrt{{}^t\overrightarrow{AB}\mathcal{M}_2(A + t\overrightarrow{AB})\overrightarrow{AB}}dt\,.$$

The aim is then to construct a mesh in Ω for which the edges are of unit length so as to ensure the same property for the corresponding edges on Σ.

Exercise 15.2 *Verify that:*

$$\mathcal{M}_3 = \begin{pmatrix} \frac{1}{h^2} & 0 & 0 \\ 0 & \frac{1}{h^2} & 0 \\ 0 & 0 & \frac{1}{h^2} \end{pmatrix}$$

implies that the edges on Σ are of uniform length of size h. On a simple example, to be defined, show the control matrix $\mathcal{M}_2$.

Choice of the control metrics. A matrix of the form:

$$\mathcal{M}_3 = \begin{pmatrix} \frac{1}{h^2} & 0 & 0 \\ 0 & \frac{1}{h^2} & 0 \\ 0 & 0 & \frac{1}{h^2} \end{pmatrix}. \tag{15.8}$$

specifies, as already seen, a field of uniform sizes h on the surface. A matrix of the form:

$$\mathcal{M}_3(P) = \begin{pmatrix} \dfrac{1}{h^2(P)} & 0 & 0 \\ 0 & \dfrac{1}{h^2(P)} & 0 \\ 0 & 0 & \dfrac{1}{h^2(P)} \end{pmatrix}, \tag{15.9}$$

where now, h depends on the position, specifies a variable field of sizes on the surface. If we pose $h(P) = \alpha\rho(P)$ where $\rho(P)$ is the smallest of the radii of curvature ρ_1 and ρ_2 at point P of Σ and α is a adequate coefficient, we obtain an *isotropic* control of the sizes related to the radii of curvature (thus to the geometry of Σ). A matrix of the form:

$$\mathcal{M}_3(P) = {}^t\mathcal{D}(P) \begin{pmatrix} \dfrac{1}{\alpha^2\rho_1^2(P)} & 0 & 0 \\ 0 & \dfrac{1}{\beta^2\rho_2^2(P)} & 0 \\ 0 & 0 & \lambda \end{pmatrix} \mathcal{D}(P) \tag{15.10}$$

with $\mathcal{D}(P)$ the principal directions at P, α and β adequate coefficients and λ an arbitrary scalar value gives an *anisotropic* control over the geometry that takes the two principal radii of curvature into account. Also, a matrix of the general form:

$$\mathcal{M}_3(P) = {}^t\mathcal{R}(P) \begin{pmatrix} \dfrac{1}{h_1^2(P)} & 0 & 0 \\ 0 & \dfrac{1}{h_2^2(P)} & 0 \\ 0 & 0 & \dfrac{1}{h_3^2(P)} \end{pmatrix} \mathcal{R}(P) \tag{15.11}$$

with $\mathcal{R}(P)$ directions and h_i three sizes, gives a control on the three directions of $\mathcal{R}(P)$ and specifies the length expected in these specific directions.

Control obtained depending on the choice fixed: the edges. We have *a priori* four types of control matrices (Relations (15.8) to (15.11)).

Let consider a matrix of the form (15.9) and the choice $h(P) = \alpha\,\rho(P)$. The question is to fix α so as to obtain a certain control within a given ε of the gap between the mesh and the surface. Recall that ρ is the smallest of the two principal radii of curvature ρ_1 and ρ_2 at P.

By definition (Chapter 11) the circle of radius ρ centered at point $O = P + \rho\vec{\nu}$ is an approximation at the order two of the curve intersection of the surface with any plane supported by $\vec{\nu}$ with ρ the radius of curvature in this particular plane. Hence, approximating the surface by an edge while controlling the accuracy is equivalent to controlling, for this particular case, the gap between a discretization of the osculating circle (the circle above) of the plane and the latter.

We can see immediately that fixing $\alpha = 1$ is equivalent to discretizing this circle with 6 edges. The length of the circle for the metric (15.9) is indeed $2\,\pi \approx 6$. Then, Figure 15.8 (left-hand side), taking an edge of length ρ, we obtain $\delta = \rho\,(1 - \sqrt{\frac{3}{4}})$ and thus $\frac{\delta}{\rho} = 1 - \sqrt{\frac{3}{4}} \approx 0.15$. The relative gap to the circle is 15%.

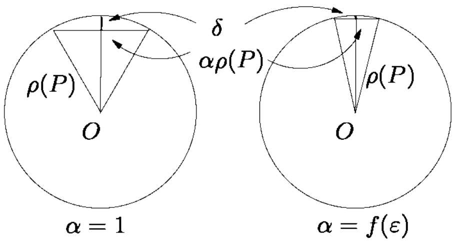


Figure 15.8: *Discretization of a circle with a stepsize $\rho(P)$ (left-hand side), or with a stepsize $\alpha\,\rho(P)$ (right-hand side).*

On the other hand, let us assume that the discretized edge is of size $\alpha\,\rho$, then we found:

$$\delta = \rho\,(1 - \sqrt{1 - \frac{\alpha^2}{4}})$$

and thus, setting $\frac{\delta}{\rho} < \varepsilon$ comes down to fixing:

$$\alpha \leq 2\,\sqrt{\varepsilon\,(2-\varepsilon)} \tag{15.12}$$

and, hence, an accuracy within a given ε imposes that α is bounded as previously.

Remark 15.6 *We define thus a* relative *control, within ε, of the distance between an edge and the surface.*

Let us now consider the anisotropic case: the control matrix is that of Relation (15.10) in which the coefficients α and β are involved. For a fixed ε, the simplest choice consists of following Relation (15.12) for α and setting $\beta = \alpha$.

For ρ_1 and the corresponding principal direction, we find a gap δ_1, for ρ_2 and its direction, we have δ_2. According to the choice of α and β, we have $\delta_1 = \varepsilon\,\rho_1$ while $\delta_2 = \varepsilon\,\rho_2$. Hence, if ρ_2 and ρ_1 are different, we have $\delta_2 > \delta_1$, the *absolute* gap is not the same in the two principal directions and thus this value varies according to the direction. Setting a constant gap comes down to setting $\delta_1 = \delta_2$ and leads to fixing:

$$\alpha \leq 2\,\sqrt{\varepsilon\,(2-\varepsilon)}$$

as above and to defining:

$$\beta \leq 2\sqrt{\varepsilon\frac{\rho_1}{\rho_2}\left(2 - \varepsilon\frac{\rho_1}{\rho_2}\right)}\,. \tag{15.13}$$

Remark 15.7 *Thus we have a* local absolute *control, within ε, of the distance between an edge and the surface, depending on the directions.*

The case of other matrices (i.e., non-geometric *a priori*) prescribes sizes based on physical criteria (related to the behavior of the solution to the problem considered). There is no specific reason why this specification must be consistent with the surface geometry. Hence, we must deduce from the given metrics a metric that reflects the desired physical aspect as well as taking the geometrical constraints into account.

Control obtained: faces. The previous discussion gives hints about how to control the gap between the mesh edges and the surface. However, such a control gives no guarantee that the triangles created with these edges will be close to the surface or that the resulting mesh has the desired regularity (the presence of folds, for example). Hence, a specific process must be performed to ensure these properties. The basic idea is to modify the field $\mathcal{M}_3$ or to involve mesh adaptation principles:

- Regarding the discussion in Chapter 10, we know that the gap between a triangle and the surface needs to account for a upper bound of the Hessian of the surface on this triangle. This bound allows us to find, within a given tolerance, a bound on the edge size. This bound then serves to define the matrix $\mathcal{M}_3$ and, more precisely, its coefficients.

- A rather different idea consists of using the mesh adaptation principle (Chapter 21). An initial mesh is constructed, analyzed and, depending on the result, the process is iterated with the field of the metrics deduced from the analysis.

A meshing method. To clarify the ideas, let us consider a Delaunay-type method (Chapter 7). Meshing a patch comes down to:

- choosing the type of control (the metric) allowing us to obtain the desired mesh (i.e., to fix the field of matrices $\mathcal{M}_3$),

- deducing the field of corresponding matrices $\mathcal{M}_2$,

- meshing the parametric domain boundaries based on the field of $\mathcal{M}_2$ (Chapter 14),

- meshing the parametric domain, from the boundary discretization, using an anisotropic Delaunay-type method (Chapter 7),

- mapping this mesh on the surface Σ via the function σ.

The mesh in Ω following the approach described in Chapter 7 consists of constructing an initial mesh based on the sole boundary vertices and in adding points along the edges. Then, these points are inserted and the process is iterated on the resulting mesh. The key is thus to properly calculate the edge lengths. This calculation naturally requires the use of the surface, hence the meshing technique differs from a purely two dimensional meshing technique, in that it must have access to the surface (in a discrete way, for instance).

We now provide an example. Figures 15.9 to 15.12 [Borouchaki *et al.* 1999], show different meshes of the patch defined by:

- Ω, the parametric space is the circle of radius 10,
- the function σ defining the surface is

$$\begin{cases} u^3 + 10\,u \\ v^3 + 10\,v \\ 100 \sin u \cos v \end{cases}$$

The surface meshes in Figures 15.9 to 15.12 (right-hand side) have been created according to this principle. Figures 15.9 to 15.12 (left-hand side) show the four two-dimensional meshes of the corresponding parametric spaces.

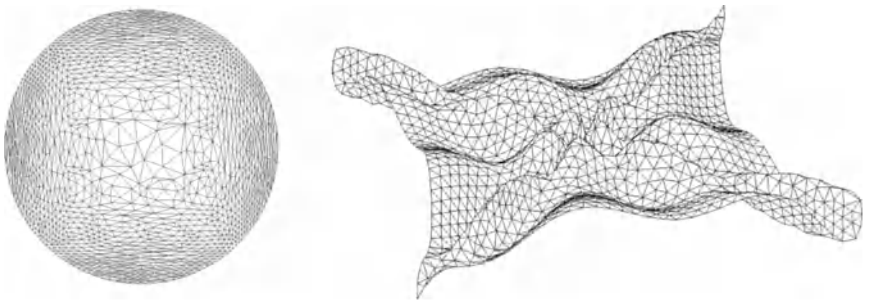


Figure 15.9: *Uniform mesh on* Σ. *Left-hand side: mesh of the parametric space; right-hand side: uniform mesh with a stepsize* $h = 50$.

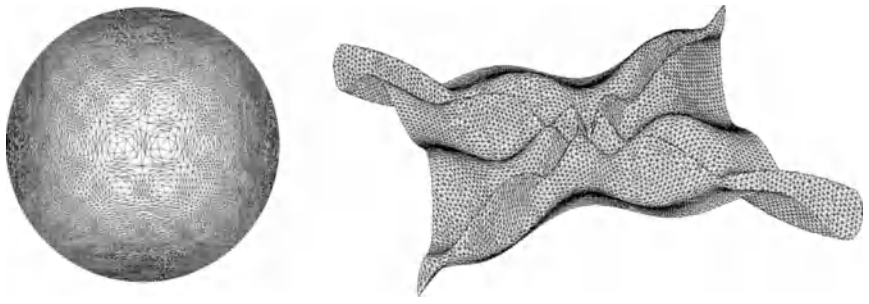


Figure 15.10: *Uniform mesh on* Σ. *Left-hand side: mesh of the parametric space; right-hand side: uniform mesh with a stepsize* $h = 20$.

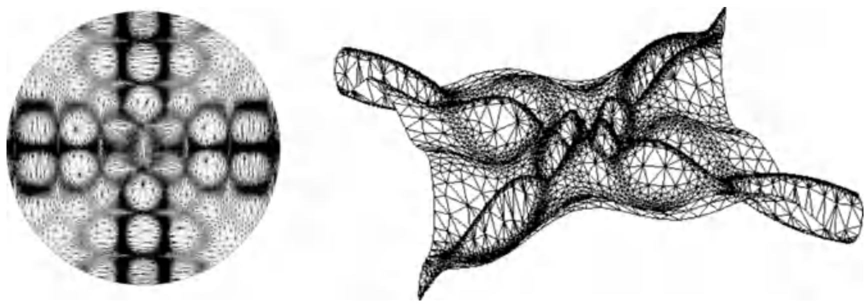


Figure 15.11: *Left-hand side: mesh of the parametric space; right-hand side: isotropic mesh controlled by* ρ.

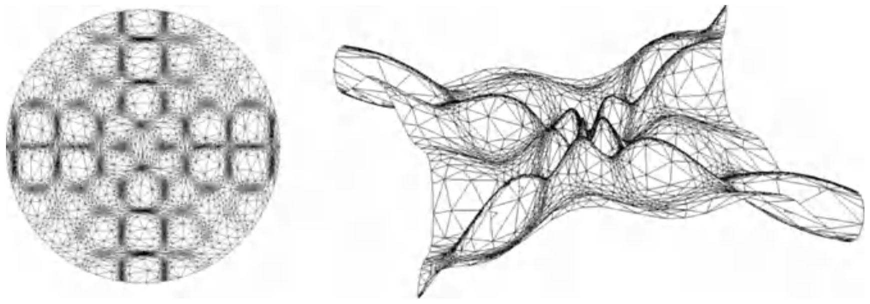


Figure 15.12: *Left-hand side: mesh of the parametric space; right-hand side: anisotropic mesh controlled by* ρ_1 *and* ρ_2.

15.4 Multi-patches surface (patch-dependent)

When the surface is defined by several patches (each of them being parameterized), the previous algorithm (or a similar one) can be used, provided that the interfaces between the different patches are meshed first, in a unique way, so as to obtain a conforming join from one patch to another. The patch-dependent approach consists of processing the patches one by one and thus in preserving the interfaces between patches.

Synthetic scheme. The meshing scheme uses the idea of the multibloc methods (Chapter 4). It involves two successive stages:

- meshing the interface curves between patches,
- meshing each patch using the discretization of its boundaries as defined in the previous stage.

Meshing the interfaces between patches

The first stage of meshing a composite surface consists of meshing the interface curves. This operation makes it possible to guarantee that the curves shared by several surfaces are meshed in a consistent (conforming) way.

To this end, we apply the technique for meshing the curves of $\mathbb{R}^3$ already described in Chapter 14 and reviewed at the beginning of this chapter. We will not spend more time on this issue here.

At completion of this stage, the interface curves between patches are meshed in a geometric way or to take a specified field of sizes into account.

Meshing the patches

Using the discretized patch boundaries, we will construct the meshes of patches. The union of all meshes of the patches will lead to a correct mesh of the surface, the interfaces having been correctly defined during the previous stage.

The principle of the meshing technique for a multi-patches surface consists of meshing each patch separately via its associated parametric space. The final mesh is the union of the meshes of the different patches. The conformity of the resulting mesh is ensured by the conformity of the interfaces between patches (see above).

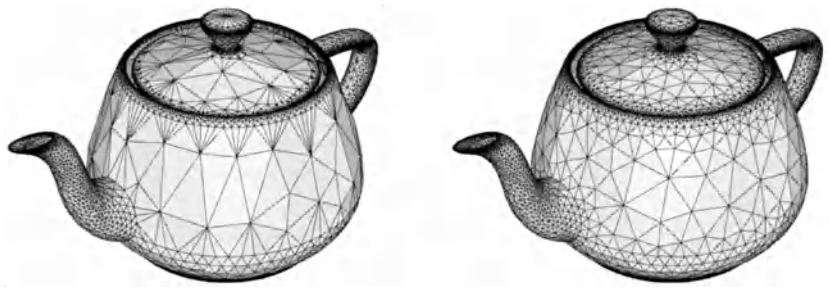


Figure 15.13: *Isotropic geometric mesh of the minimal radius of curvature of a multi-patches surface, the Utah teapot (left-hand side) and isotropic geometric mesh incorporating a size map correction (right-hand side).*

Notice that such an approach may lead to some rather large disparities between the element sizes from one patch to another. This is notably the case when the mesh is a geometric mesh, based on the model curvatures (Figure 15.13, left-hand side).

To obtain a mesh in which the size gradation is controlled, the idea is to use the resulting mesh to construct a control space. More precisely, with each mesh vertex is associated information on the local size and, possibly, the stretching direction of the elements. This discrete field of sizes (metrics) being created, we can apply a smoothing procedure on it to bound the size variations from one point to another (Chapter 10). Once this operation has been performed, it is then possible to use this information to govern the creation of a new surface mesh in the parametric spaces (Figure 15.13, right-hand side).

15.5 Multi-patches surface (patch-independent)

For various reasons, we now want the surface mesh not to follow the patches in some regions. Consider first very small patches, which are probably not useful from the numerical point of view and which, if meshed, would lead to very large meshes. Consider also very thin patches having a small edge as compared with the neighboring ones. This case necessarily induces the creation of very stretched elements. Figure 15.14 shows an example of such situations. In this figure, we show the entire geometry (i) and the enlargement of an area composed of stretched patches (ii). Any mesh respecting these patches (for example, the mesh in iii) contains flat elements that are *a priori* inappropriate to the calculations.

Thus, it seems necessary to get rid of the boundaries of some patches. To this end, two approaches can be envisaged. We can follow an *indirect* approach that consists of constructing a mesh respecting the patch boundaries and then in modifying this mesh to get rid of this constraint in the regions where it leads to undesirable effects. We can also consider a *direct* approach that, prior to any meshing, get rid of this type of situation.

Indirect approach

Each patch is processed via the previously described method (patch-dependent), then we get rid of (some) patch boundaries. The operators involved in this process are the classical operators for surface mesh modification and surface mesh optimization (as described more precisely in Chapter 19). We distinguish essentially a node relocation operator (that preserves the connections between the vertices) and operators that, with fixed point positions, modify their connections. Among these operators, we find the edge swaps (that change the common edge between two faces) and refinement or derefinement operators that make it possible to:

- subdivide a given edge and to modify the faces sharing the edge accordingly,
- suppress one or several edges either by merging nodes or by temporarily creating a hole (Figure 15.15), and then remeshing it.

Whichever the operator is used, it is necessary to preserve the geometric and topological coherence of the result. Firstly, the result must stay close to the surface. Then, the resulting topology must be identical to the initial one. Notice that it is important to classify the entities of the mesh (points, edges and faces) and to propagate this information from the current mesh and the modified mesh (in order, in particular, to be capable later of identifying the entities subjected to boundary conditions for the problem considered).

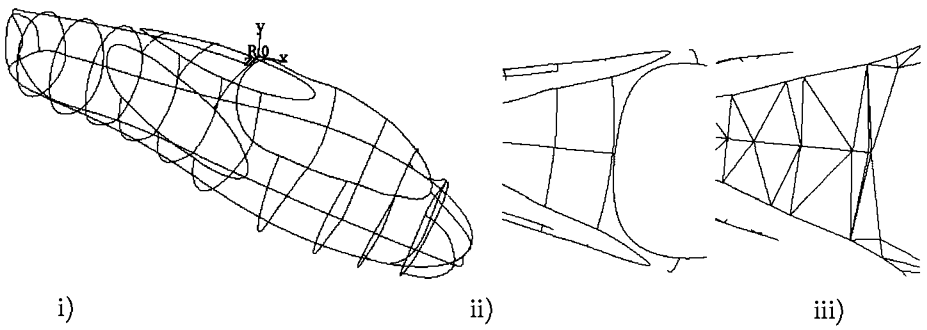


Figure 15.14: *An example of mesh construction preserving the patches defining the geometry and the perverse effects of this constraint on the resulting mesh. In i), we show the limits of several patches modeling the geometry, in ii) and iii), we show respectively the detail of a region of this description and the resulting mesh in which two flat triangles are constructed.*

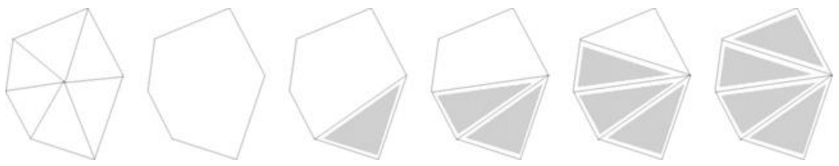


Figure 15.15: *Left-hand side, the initial mesh. From left to right, the sequence of remeshing of a hole formed by removing the ball of the point deleted.*

Direct approach

A direct approach is based on the fact that meshing techniques are available to process a patch. The idea is then to represent a set of patches by a single patch. Hence, the difficulty is to construct this single patch. This is relatively easy to perform when the geometry is defined by a polyhedron (i.e., when the set of patches of this definition forms such a polyhedron). Moreover, this method can be extended to an arbitrary surface. Figure 15.16 illustrates an application of this method[9]. We see in i), the initial surface, ii), a planar mapping of the complete surface where each triangle corresponds in a one to one way to a piece of the surface. Therefore, an injective function σ can be constructed depending on the parameters u and v to project any point of the parametric space onto the surface. This being done, an anisotropic mesh of the parametric space must be created, its mapping onto the surface leading to the desired mesh. The example of the mesh in the figure is simpler, a regular (uniform) grid over the parametric space is developed, in iii) and, in iv), we show the mechanical mapping of this regular mesh on the surface. The main ideas of the method are the following:

- identification of the boundaries (principal and secondary) and of the sides of

[9]Work in progress at the 3S laboratory, Grenoble, by F. Noël.

the holes,

- subdivision into triangles of the parametric space associated with each patch,
- topological merging of the triangles,
- identification of the edges of the main boundary and of the secondary boundaries and sides of the holes,
- topological meshing of the secondary contours and holes (edges and faces are associated with these contours),
- node distribution for the main contour along the boundary of a convex domain,
- determination of the positions of the other nodes using a barycentrage technique from their neighbors. The resulting mesh allows us to define the parametric space associated with the whole surface. A bijection exists between each triangle of this two-dimensional mesh and a triangular-shaped portion of the surface.

The convex planar domain (corresponding to the parametric space) is meshed so as to take the constraints into account, for example, regarding the boundaries of the secondary contours or the holes. A governed meshing technique can thus be used (for example an anisotropic Delaunay-type method) to mesh the parametric space (see above).

The resulting mesh in the parametric space is then mapped onto the surface using the one to one function linking the mesh triangles to the corresponding triangular cells on the surface.

Notice that this technique of planar mapping needs still to be improved and that the example provided reflects only preliminary experiments on this very promising topic.

15.6 Ill-defined multi-patches surface

As pointed out, complex surfaces can be defined with a rather important number of patches. In what precedes, we have implicitly assumed that the set of patches defining the surface considered formed, by itself, a conforming mesh (actually, rather a covering-up) of this surface (Chapter 1). In practice, this assumption is most likely not verified. Actually we find several situations preventing the desired conformity. This is related to the fact that the geometric definitions of the patches are not, *a priori*, directly motivated by this criterion but are rather of a visual nature or related to a possible manufacturing process. Moreover, the numerical accuracy used in a CAD system may be variable from place to place in a given model and, is definitely not conceived in a meshing view.

In practice, the awkward situations encountered correspond to:

- overlappings between two or more patches,

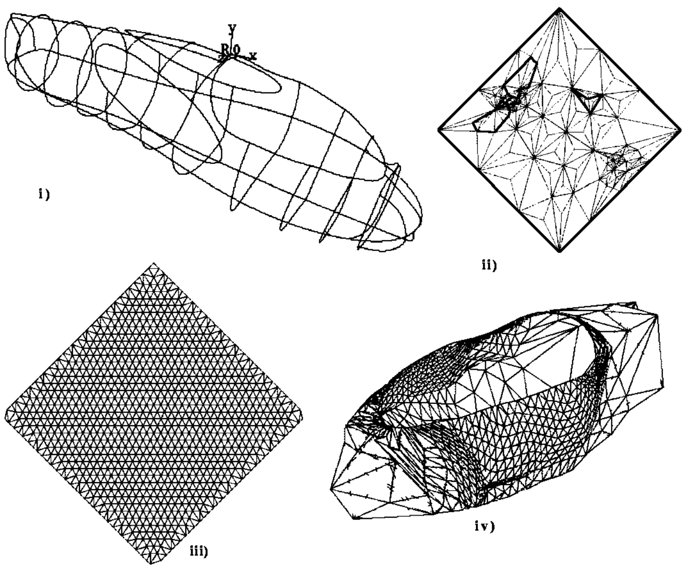


Figure 15.16: *An example of mesh construction based on a single patch definition. We see, i), the definition with patches of the entire surface. In ii), we show the planar mapping of these patches. In iii), we define a regular grid in the parametric space and, in iv), we provide the mapping of this regular mesh onto the surface.*

- gaps (holes) between patches assumed to be joined,
- non-conforming joins between patches regarding corners assumed to be but not identical (within a tolerance) and supposedly common borders. For example, the boundary of a patch matches only part of the boundary of another patch instead of the entire patch, etc.

All these defects, detected from a mesh generator but not visible in general by the user and with no effect in a manufacturing process, essentially concern defects related to points (absence or duplication of points), lines (bad join between successive or common lines, absence or duplication) or surfaces (holes, partial duplication). Moreover, as already mentioned, disparate edge sizes (patch boundaries) lead to ill-suited patches with respect to a possible mesh. Attempting to mesh, for example patch by patch, a surface composed of patches having these pathologies would certainly lead to a failure[10].

Thus, before attempting to mesh, it is necessary to fix, with respect to the objective, the surface components. This task is especially important, as pointed out, but is rather tedious to carry out. To date, there is no fully automatic,

[10]Notice that it is the best desirable result. In fact, if the mesh generation algorithm fails to detect these defects, there may be no other easy way of detecting that the resulting mesh is wrong. Hence, any calculation performed on such a mesh would give surprising results.

quick and of universal range method allowing such a surface to be *repaired.* Some mostly interactive (i.e., that require the user intervention) software products exist, which may ease this type of process. The idea is to offer a maximum of features regarding the visualization and the detection of the defects as well as to provide all types of local modification tools (node relocation, edge swapping, creation or deletion of entities, etc.) in order to correct the surface while ensuring that the result preserves the initial topology in a plausible way. It might well be imagined therefore that repairing a surface is, at present, a hot topic.

We must however mention that work regarding the automation of such processes, [Barequet *et al.* 1998], is being carried out.

15.7 Molecular surfaces

The so-called computational chemistry by means of PDE's models and FEM or similar solution methods is a relatively new field of activities.

A typical example is concerned with solvation problems where the molecule behavior is investigated using various models such as VWS (Van der Waals Surface), SAS (Solvent Accessible Surface) or SES (Solvent Excluded Surface or Connolly surface) where the surface, the spatial support of the computation, is defined in one way or another, [Tomasi *et al.* 2003], leading to various mesh generation problems.

Whatever the case, in terms of the mesh generation method, this situation is basically of the parametric (thus indirect) type where a two-dimensional parametric space is meshed and the resulting mesh is mapped on the real surface in order to obtain the desired three-dimensional surface mesh.

One of the major differences between this case and the classical situation is in the manner in which the parametric space is defined. Instead of being defined by means of a CAD approach (using one or several patches and the corresponding σ functions), the parametric space is defined in a constructive way related to the specificity of the geometry in hand, [Laug, Borouchaki-2002].

A molecule is made up of atoms following some arrangement. A given atom is seen as a ball or its bounding sphere, S, known by its center, C, and its radius, r. Therefore, a molecule is seen as the outer hull of a series of intersecting spheres S_i given through the corresponding C_i and r_i.

With each atom is associated a patch by means of a stereographic projection (Figure 15.17). In the academic case of only one atom, the full sphere is split into two parts by introducing an artificial cut and two patches are defined. In the case of intersecting spheres, intersections are computed leading to restrict the domain to be projected and obtain trimmed (arbitrary) patches. The projection of a point $P = (x, y, z)$ is defined by considering the North pole of the sphere, $N = (0, 0, 2r)$ in the figure, and obtained as the intersection of line PN with plane $z = 0$ which gives the function $\sigma(u, v)$ of the related patch.

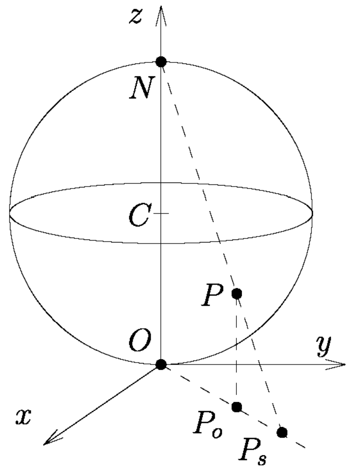


Figure 15.17: *Definition of the parametric space by means of a projection. P_o is the orthogonal projection of point P while P_s is its stereographic projection.*

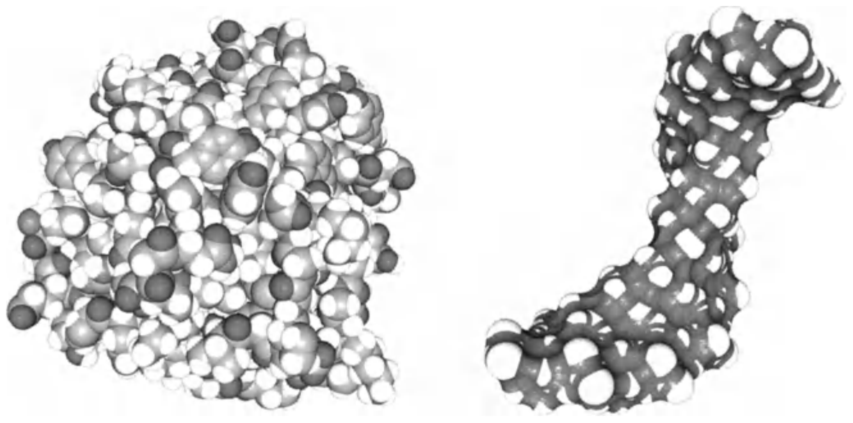


Figure 15.18: *Molecular surfaces. Left, SAS model of the papaine molecule. Right, SES model of the dna molecule.*

The parametric domains being defined, we return to the classical meshing process as discussed in the previous sections. Typical examples are depicted in Figure 15.18 demonstrating two different molecules and the related surface meshes (thanks to [Laug, Borouchaki-1999]). The surface for the SAS example involves a sphere of radius r_p, the *probe*, whose center rolls on the VWS (the surface of the envelope of the union of the given atoms) and defines a new envelope, the SAS. The other example is a SES case where a probe is used to smooth the reentrant part of the initial surface.

15.8 Surface reconstruction

Before going further, note that marching-cube methods and implicit surfaces are discussed in Chapter 16.

We are given a set of points in R^3 which have been captured on a surface by means of a particular device (such as a scanner) and the goal is to construct a mesh which is supposed to accurately represent this surface.

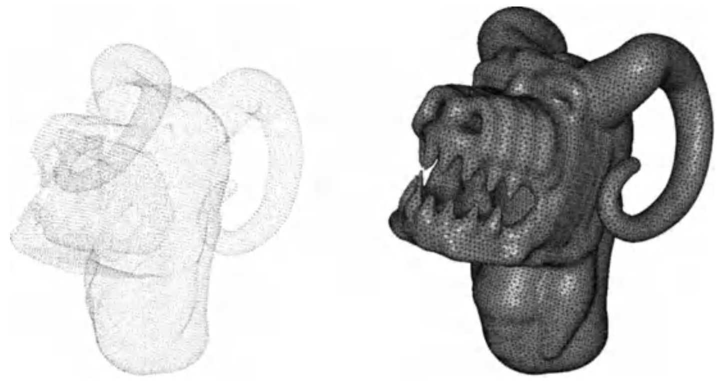


Figure 15.19: *Smooth surface. Left, the given cloud of points; right, the surface mesh after reconstruction (22,677 vertices (7 missing), 45,340 triangles, .60 sec. for the reconstruction process).*

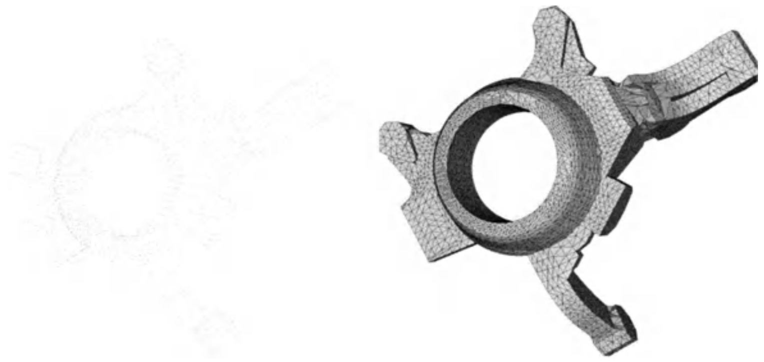


Figure 15.20: *Realistic surface. Left, the given cloud of points; right, the surface mesh after reconstruction (4,756 vertices (27 missing), 9,458 triangles, .30 sec. for the reconstruction process).*

Various methods exist to deal with this problem, in specific, for a given nature of the sample, e.g. unorganized or with a peculiar organisation (layers, etc.). Nevertheless, most of the proposed methods proves to be very poor in view of further calculations. This is due to a number of reasons. On the one hand, a series of method have been devised for graphical purposes only leading to nice pictures (e.g. the goal). On the other hand, some methods are mainly of academic interest where the goal is to prove some propreties and to have some guarantees about the reconstructed surface. To this end a number of prerequisites is assumed about the sample itself (density, uniformity, etc.), the expected smoothness, the geometry (local thickness, etc.). With such a type of formulation of the problem, guarantees can be obtained but realistic cases generally violate a particular *a priori* necessary properties and thus the reconstructed surface is what it is and not more.

The main drawbacks consist of undesirable holes, non-manifold edges, default of orientation, missing points, ambiguities (a solution has been constructed but other reasonable solutions seem also appropriate).

Recent advances in this problem consider using the Delaunay triangulation of the given set of points and then extract the triangles defining the seeked surface among the facets of the tetrahedra of this triangulation. Criteria (see below) to decide whether a facet must be retained as a surface triangle are multiple and somehow antagonist, hence the difficulty of the problem.

Indeed, we are interested in a smooth surface whether it is smooth or not (e.g. corners or ridges exist such as in mechanical parts).

Mid-surface simulation can be used both to select a number of triangles and to delete some other. Nevertheless, the resulting surface is in general unsatisfactory. Indeed, some selected triangles are obviously not in the solution while some triangles not retained must be in the solution. In other words, the resulting surface includes a number of (relatively) small holes. Fixing these holes relies in finding one or several facets which recover them.

To do this, criteria are used which must produce a "smooth" solution. Criteria include proximity concerns (such as distance from point to point) or emptyness of Gabriel's balls (the diametral ball for an edge, the equatorial ball for a triangle), angles between adjacent facet normals (to prevent folds), control of the normal at a given vertex (which is evaluated using the cells of the dual), the three angles of a facet itself and angle defects.

For a given point P, the last criterium sums the angles in P of all the retained triangles including P as a vertex (note that this value is part of the discrete approximation of the Gauss curvature of a surface). The targeted value is then 360 degrees for a planar closed surface, it is also 360 degrees along a straight ridge while it must be adapted for open or curved surfaces. Most of the vertex presents a value close to the targeted value. A value largely bigger or smaller than 360 is quite possible (for instance, at a corner) but evenly might be an indicator of bad reconstruction (for instance, a wide excess appears when a fold exists).

None of these criteria is meaningless by itself but none is sufficient to decide for a solution. In specific, a given criterion is likely to be verified almost everywhere (as demonstrated by an *a posteriori* statistic about the corresponding quantity) while being strongly violated in some part of the solution. Therefore, the guarantee of correctness could be only of a statistical nature meaning validating the method using a large series of different cases.

Figure 15.19 shows a rather simple case illustrative of a smooth surface and a relatively good sample (while sharp regions around the teeth may lead to ambiguity). On the other side, Figure 15.20 shows a typical case illustrative of a mechanical surface with various "bad" features such as thin local thickness, ridges, etc., despite the apparent good quality of the sample.

15.9 Discrete surface (re-meshing process)

We consider here a situation in which the data of the problem is of a discrete nature. Actually, we have a meshed surface (a triangulation) with which additional information may be associated. These specifications, when given, allow us to know the particularities and specificities of the underlying geometric model (corners, ridges and/or constraints, tangents, etc.). Without such information, the task of the remeshing process will be more tedious [Löhner-1995], [Löhner-1996a].

We will first specify how the underlying surface can be defined from the given triangulation. Then, we will focus on the remeshing problem, treated in practice via geometric and topological modifications applied to the initial mesh. We will indicate in particular the main features of a remeshing algorithm.

Definition of a discrete surface

We will see in Chapter 19 that the intrinsic properties of a surface (normals, principal radii of curvature, etc.) can be extracted[11] from an initial mesh of the surface, so as to construct the metric of the tangent plane. Recall that this metric (defined in the tangent planes associated with the given mesh vertices) makes it possible to control (bound) the gap between the mesh edges and the underlying surface.

Identification of the singularities. If the singularities of the model are not explicitly specified, a pre-processing stage makes it possible to identify these features. This stage, which is almost fully automatic, must be user-supervised. We have already mentioned that a ridge can be identified using a threshold on the adjacent face angle and that a corner is, notably, a vertex sharing three ridges.

Other entities (constraints) can be provided (by the modeling system) which must be preserved in the final mesh.

Construction of a geometric support. This operation is central to the remeshing approach. Indeed, it is aimed at defining, internally, a geometry (that is, an analytical representation of the underlying surface, the initial mesh being supposed to be an approximation of this surface). In practice, the geometric support is a composed surface of class G^1 (i.e., ensuring the identity of the tangent planes between adjacent patches).

Among the various approaches possible, one of particular interest is that suggested by Walton and Meek [Walton, Meek-1996], based on the Gregory patches

[11]That is, evaluated approximatively.

[Gregory-1974] (Chapter 13). This support, suitably defined, can be used to know the exact position of a vertex on a surface (usually the closest location), given a current mesh vertex and a direction.

Re-meshing a discrete surface

The remeshing of a surface defined by a polyhedral approximation (a mesh) involves topological modifications (edge swapping, vertex merging, edge splitting, etc.) and geometric modifications (node relocation) which will be considered in detail in Chapter 19.

Optimal mesh. The objective of the remeshing process is to obtain a mesh in which the elements have a size (and possibly an orientation) conforming to the geometric size field. In other words, the aim is to obtain an *optimal mesh* with respect to the given geometric specifications. To this end, the element edges of the current mesh are analyzed and possibly optimized in size, so that any edge AB of the final mesh has a length l_{AB} (in the metric specified) such that:

$$\frac{1}{\sqrt{2}} \le l_{AB} \le \sqrt{2}, \quad \forall AB \in \mathcal{T}. \tag{15.14}$$

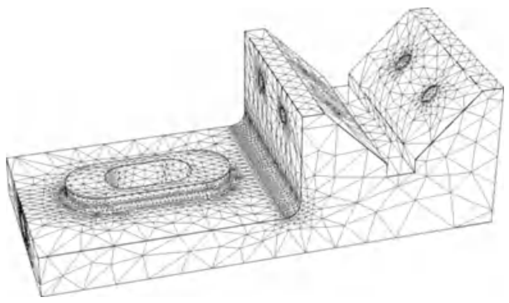


Figure 15.21: *Geometric remeshing of a discrete surface.*

Remeshing algorithm. The general scheme of the surface remeshing algorithm can be written as follows:

- initializations: identification of the singularities (corners, ridges, etc.);
- evaluation of the intrinsic properties of the surface (curvatures, normals, etc.);
- construction of a geometric support of class G^1;
- remeshing:
  - for each edge AB of the current mesh,

- if $l_{AB} > \sqrt{2}$, subdivide the edge into unit length segments,
- else if $l_{AB} < \frac{\sqrt{2}}{2}$, merge the two endpoints of the edges,
- if the mesh has been modified, apply edge swappings on the newly created faces;

- optimization of the resulting mesh using node relocation as well as edge swapping.

Figure 15.21 illustrates an example of a discrete surface remeshing. The surface is defined from the mesh represented in Figure 15.4.

Another example is depicted in Figure 15.22 while other examples can be found in Chapter 21.

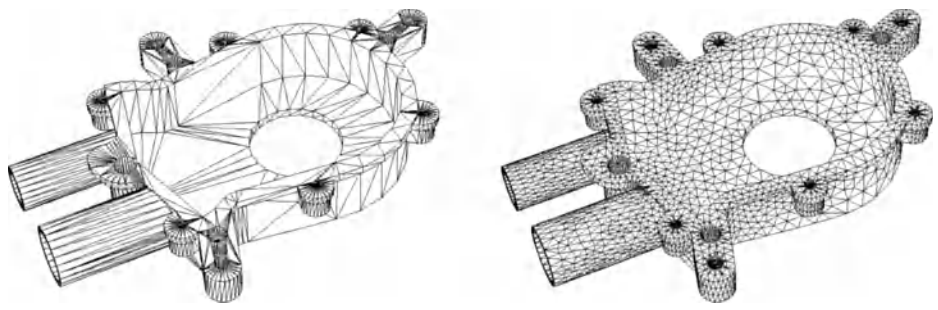


Figure 15.22: *Geometric remeshing of a discrete surface (left) with a size control.*

Chapter 16

Meshing Implicit Curves and Surfaces

In practical terms and notably in applications related to geometric modeling and graphical visualization, the usual representation of curves and surfaces is the *parametric* one (cf. Chapters 12 and 13). Nevertheless, other representations exist and are employed to some extent depending on the applications envisaged. An *explicit* representation is based on functions of the form $z = f(x, y)$. This approach is quite limited in practice, as the surfaces defined in this way are usually rather "rudimentary" or do not correspond to concrete cases. A third approach consists of defining a surface as the set of points (x, y, z) in $\mathbb{R}^3$, which are solutions of an equation of the type $f(x, y, z) = 0$. The study of such surfaces, called *implicit surfaces*, is the subject of this chapter.

Interest in implicit curves and surfaces has increased over the last few years, notably due to the emergence of discrete (sampled) data for modeling computational domains. *Discrete geometry* attempts to transpose the results of classical (affine and differential) geometry to the discrete field. However, as pointed out by [Hoffmann-1993], the application field of implicit functions seems to remain largely underestimated.

★
★ ★

Given an implicit curve or surface representing the boundary of a computational domain, we focus here on the problem of meshing this boundary. In the first section, we recall some basic definitions and properties of implicit functions. Then, we deal with the mesh generation of implicitly defined planar curves. In the third section, we indicate how the meshing techniques for curves can be extended to the meshing of implicit surfaces. Application examples are shown at the end of this section to illustrate the various meshing techniques for these surfaces. The last section briefly presents the basic principles of constructive geometry for implicit domains before dealing with mesh generation of implicit domains (whose boundaries are implicit curves or surfaces), presented here as a natural extension of meshing techniques for curves and surfaces.

16.1 Review of implicit functions

In this section, we recall the main definitions and properties related to implicit planar curves and surfaces. In particular, we see how the main results of differential geometry introduced in Chapter 11 are involved.

A preliminary remark

As meshing techniques suitable for parametric curves and surfaces have been extensively developed, as seen before, one might ask if it were not possible to convert an implicit function into one (or more) equivalent parametric representation(s), so as to find a known problem (Chapter 14).

Notice first that any parametric rational curve or surface assumes an implicit form, the way of obtaining it being widely known (see [Sederberg-1983], among others). However, obtaining the parametric form corresponding to a given implicit function is not trivial, for many reasons that we need not concern ourselves with here [Hoffmann-1993]. In practical terms, this old problem[1] turns out to be extremely difficult to solve in the general case, which justifies the development of direct meshing methods for implicit curves and surfaces.

Implicit planar curves

Let $f : \Omega \longrightarrow E$ be a function of class C^k ($k \geq 1$) on an open set Ω of the affine plane E (here $E = \mathbb{R}^2$). The set Γ of points $M(x, y) \in \Omega$ such that $f(M) = f(x, y) = 0$ is the so-called *implicit curve* defined by $f = 0$:

$$\Gamma = \{M(x,y) \in \Omega\,,\ f(M) = f((x,y) = 0\}\,.$$

Definition 16.1 *A point $M(x, y)$ of the curve Γ is said to be an* ordinary *(or* regular*) point if it is such that:*

$$\nabla f(M) = {}^t\left(\frac{\partial f(M)}{\partial x}, \frac{\partial f(M)}{\partial y}\right) \neq 0\,,$$

where the operator ∇ denotes the gradient. Thus, the curve Γ is said to be regular *if $\nabla f(M) \neq 0$ for each point M. The point M is said to be* singular *if $\nabla f(M) = 0$.*

Notice that the inverse image of a value k of $\mathbb{R}$ is the solution to the equation $f(M) = k$, at a point M of $\mathbb{R}^2$. Thus, the curve Γ, defined by $f = 0$, is $f^{-1}(0)$, the inverse image of 0. More generally, a curve Γ_k defined by $f(x, y) = k$ is called an *iso-value curve* (or *level curve*), k being the *level* of f.

The geometric interpretation of the theory of implicit functions makes it possible to deduce the following result.

Theorem 16.1 (implicit functions) *If M_0 is an ordinary point of the curve Γ, there exists a neighborhood $V(M_0)$ such that $\Gamma \cap V$ is the support of a parameterized arc Γ_0 of class C^k.*

[1]In the last century, [Salmon, 1885] already proposed a way of getting rid of the parameters of parametric equations.

Hence, it is important to bear in mind from this result that the implicit curve Γ admits local parameterizations at each of its regular points[2] Theorem (16.1) makes it possible to write in explicit form $y = y(x)$. Figure 16.1 illustrates a parameterization of Γ.

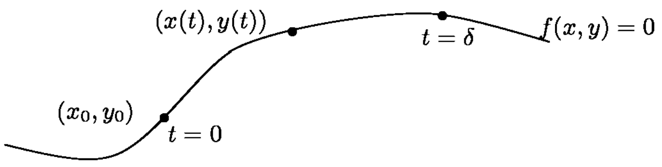


Figure 16.1: *A parameterization $x(t), y(t)$ of the implicit function $f(x, y) = 0$ passing through the ordinary point (x_0, y_0).*

Study of critical points. In the case where the implicit function theorem holds, the study of the extrema of f requires a preliminary study of its critical points.

Definition 16.2 *We say that a function f has a* local maximum *(resp.* local minimum*) at a point P, if there exists a neighborhood $\mathcal{V}$ of P such that:*

$$\forall M \in \mathcal{V}, \quad f(M) \leq f(P) \qquad (\textit{resp. } f(M) \geq f(P)\,).$$

An extremum *is a maximum or a minimum.*

The curve Γ has a *critical point*, say $M(x, y)$, if:

$$\nabla f(M) = 0 \quad \Longleftrightarrow \quad \frac{\partial f}{\partial x} = \frac{\partial f}{\partial y} = 0\,.$$

By extension, a *critical value* represents the value of the function at a critical point. The study of the Hessian H_f of f makes it possible to specify the nature of this critical point. More precisely, if $H_f(M) > 0$ we have a *minimum*, if $H_f(M) < 0$ we have a *maximum* and if $H_f(M) = 0$ we cannot conclude (we must then study the "sign" of the derivatives at order 3).

For an ordinary point $M(x, y)$ of an implicit curve Γ to be a *point of inflection*, its coordinates must satisfy the equation $H(x, y) = 0$, where the function H is defined by:

$$H = f_y'^2 f_{xx}'' - 2 f_x' f_y' f_{xy}'' + f_x'^2 f_{yy}''\,,$$

which can also be written as follows:

$$H = \begin{vmatrix} 0 & f_x' & f_y' \\ f_x' & f_{xx}'' & f_{xy}'' \\ f_y' & f_{xy}'' & f_{yy}'' \end{vmatrix}\,.$$

[2]From the numerical point of view, it is important to know that the value close to the regular values are regular as well.

Tangent vector. Using the implicit function theorem, we deduce that the tangent at an ordinary point $M_0(x_0, y_0)$ to the parameterized arc Γ_0 is the line defined by the equation:

$$\langle \nabla f(M_0), \overrightarrow{MM_0} \rangle = 0\,.$$

In fact, there exist two intervals I and J and an application φ of class C^k verifying $\varphi(x_0) = y_0$, such that the relations:

$$(x, y) \in I \times J \quad \text{and} \quad f(x, y) = 0$$

are equivalent to $x \in I$ and $y = \varphi(x)$. Thus, the set of corresponding points M is Γ_0, the arc of Cartesian equation $y = \varphi(x)$.

The tangent at M_0 to Γ_0 is the line of equation $y - y_0 = \varphi'(x_0)(x - x_0)$ and thus we have:

$$\varphi'(x_0) = -\frac{f'_x(x_o, y_0)}{f'_y(x_o, y_0)}\,.$$

Hence, the tangent is defined by:

$$(x - x_0)f'_x(x_0, y_0) + (y - y_0)f'_y(x_0, y_0) = 0 \tag{16.1}$$

Hence, we note $\vec{\tau}$ the unit tangent vector at an ordinary point M_0 to Γ.

Principal normal. The normal $\vec{\nu}$ to the curve Γ at a regular point $M_0(x_o, y_0)$ is parallel to the vector of coordinates $f'_x(x_0, y_0), f'_y(x_0, y_0)$, that is, to the gradient vector ∇f. The vector

$$\vec{\nu}(M_0) = \frac{\nabla f(M_0)}{\|\nabla f(M_0)\|}$$

is called the *principal normal* to the curve at M_0 and verifies $\|\vec{\nu}(M_0)\| = 1$. Notice that, as expected, $\langle \vec{\tau}, \vec{\nu} \rangle = 0$, the two vectors being orthogonal.

Curvature. The calculation of the *local curvature* at a regular point $M_0(x_0, y_0)$ is based on the relation $\langle \vec{\tau}, \vec{\nu} \rangle = 0$. From a practical point of view, we again use the implicit functions theorem and we introduce the curvilinear abscissa s (considering this time the arc Γ_0):

$$\frac{d(\langle \nabla f, \vec{\tau} \rangle)}{ds} = \left\langle \frac{d\nabla f}{ds}, \vec{\tau} \right\rangle + \left\langle \nabla f, \frac{d\vec{\tau}}{ds} \right\rangle = 0\,,$$

which will allow us to retrieve one of Frénet's formulae linking the local curvature at M_0 to the derivative of the tangent (Chapter 11):

$$\frac{d\vec{\tau}}{ds} = C.\vec{\nu} = \frac{\nabla f}{\|\nabla f\|}\,. \tag{16.2}$$

using a similar approach, the other Frénet's formula would be written:

$$\frac{d\vec{\nu}}{ds} = -C.\vec{\tau}\,. \tag{16.3}$$

Distance from a point to a curve. In Chapter 14, we discussed the tedious problem of geometric mesh generation for (parametric) curves. In order for the mesh to follow the arc geometry, the maximal distance δ between the arc and the segment discretizing it must be bounded (i.e., the length of the curve and the length of the chord are close to each other). Thus, if h is the length of the segment, it is desirable that, for a given value ε:

$$\delta < \varepsilon\, h\,.$$

One of the problems encountered consists of evaluating this distance δ. We will focus on calculating the Euclidean distance of a given point P to a planar curve Γ implicitly defined by $f = 0$.

If the function f is a polynomial, numerical techniques can be used to evaluate this distance [Kriegman, Ponce-1990]. However, in most cases, it is rather tricky to obtain an accurate answer to this question and usually a first order approximation of this distance is sufficient to decide whether or not to pursue the algorithm.

The distance d can be defined as the minimum of the distances from P to any other point Q of Γ:

$$d(P,\Gamma) = \min_{Q\in\Gamma} \|Q - P\|\,, \tag{16.4}$$

which leads to a minimization problem.

If P is a regular point, a Taylor series of f in the neighborhood of P reads:

$$f(Q) = f(P) + \langle h, \nabla f(P)\rangle + \frac{1}{2}({}^t h H_f(P) h) + \ldots\,,$$

where $h = \overrightarrow{PQ}$ is sufficiently small to ensure the validity of this expansion. By truncating this expansion at the order 1, we have:

$$f(Q) = f(P) + \langle h, \nabla f(P)\rangle + \mathcal{O}(\|h\|^2)\,.$$

In practical terms, we simply consider the approximation:

$$f(Q) \approx f(P) + \langle h, \nabla f(P)\rangle\,.$$

The triangular inequality then makes it possible to write:

$$|f(P) + \langle h, \nabla f(P)\rangle| \geq |f(P)| - |\langle h, \nabla f(P)\rangle|$$

then, using Cauchy-Schwartz's inequality:

$$|f(Q)| \geq |f(P)| - \|h\|\,\|\nabla f(P)\|\,. \tag{16.5}$$

Finally, the distance $d(P,\Gamma)$, defined as the value of $\|h\|$ such that the right-hand term of the previous expression vanishes [Taubin-1992], is approached using the formula:

$$d(P,\Gamma) \approx \frac{|f(P)|}{\|\nabla f(P)\|}\,. \tag{16.6}$$

Practical aspects. In applications, to reduce the number of distance calculations, we can use the properties of the *Lipschitz* functions. For such a function f, we have $|f(P) - f(Q)| \leq \lambda \|P - Q\|$, for all P, Q, λ being a positive parameter characterizing f. In fact, Lipschitz's constant λ (the smallest λ satisfying the equation) defines the lower bound of the module of the derivative function. If f is a continuous function, then Lipschitz's constant is the maximal slope of the function, which is reached at one of the zeros of the second derivative of the function (i.e., at a global minimum of f').

Hence, for a given point P, let us denote $Q \in f^{-1}(0)$ the point such that:

$$\|P - Q\| = d(P, f^{-1}(0)) .$$

Then, we can write that:

$$|f(P)| < \lambda\, d(P, f^{-1}(0)) .$$

Hence, $\lambda^{-1} f(P)$ represents a distance bound for f (the sign is meaningful).

Remark 16.1 *This result makes it possible in practice to use a Newton's method (that locally converges at the order two) to find the coordinates of the root in only a few iterations using the formula:*

$$P^{k+1} = P^k - \frac{f(P^k)\nabla f(P^k)}{\|\nabla f(P^k)\|^2}$$

P^0 being a given starting point. We can thus search the first intersection between a given line and the function f.

We will go into more detail on how to use this result in the section related to the mesh generation of implicit planar curves.

Extension to implicit surfaces

We will now focus on implicit surfaces defined by a function $f(M) = f(x, y, z) = 0$. Most of the results on planar curves can be extended to the case of surfaces.

Normal, tangent. For a regular surface Σ, the vector ∇f defines a vector that is orthogonal to Σ. In fact, a regular surface is *orientable.* The unit normal vector to the surface at a regular point P is given by:

$$\vec{n}(P) = \frac{\nabla f(P)}{\|\nabla f(P)\|} . \tag{16.7}$$

Critical points. As for planar curves, a necessary condition to have a critical point is the following:

$$\nabla f = 0 \Longleftrightarrow \frac{\partial f}{\partial x} = \frac{\partial f}{\partial y} = \frac{\partial f}{\partial z} = 0 .$$

We introduce then the Hessian matrix:

$$H_f = \begin{bmatrix} f''_{xx} & f''_{xy} & f''_{xz} \\ f''_{yx} & f''_{yy} & f''_{yz} \\ f''_{zx} & f''_{zy} & f''_{zz} \end{bmatrix}, \tag{16.8}$$

which is a symmetric matrix if the function is of class C^2. The study of the determinant $Det(H_f)$ of the Hessian matrix makes it possible to conclude: if $Det(H_f) < 0$ we have a *saddle point*, if $Det(H_f) > 0$ we have an *extremum* (if $H_f > 0$ (resp. $H_f < 0$) it is a *minimum* (resp. *maximum*)) and if $Det(H_f) = 0$, we cannot conclude, we must then study the partial derivatives at the order 3.

Principal curvatures. To find the principal curvatures and the principal directions at a regular point $M_0(x_0, y_0)$, we again use the fact that $\langle \vec{\nu}, \vec{\tau} \rangle = 0$. Let $\vec{u}$ and $\vec{v}$ be two vectors forming an orthonormal basis of the tangent plane $\Pi(M_0)$ and let $\vec{\tau}$ be a unit tangent vector at M_0. We can write:

$$\vec{\tau} = \cos\theta \vec{u} + \sin\theta \vec{v}\,.$$

From the formula giving the curvature κ_τ in the direction $\vec{\tau}$:

$$\kappa_\tau = \frac{{}^t\vec{\tau}\, H_f\, \vec{\tau}}{\|\nabla f\|},$$

we can deduce the formulae of the principal curvatures κ_1 and κ_2 and the principal directions τ_1 and τ_2 (expressed in the basis $[\vec{u}, \vec{v}]$ [Monga, Benayoun-1995]):

$$\kappa_1 = \frac{{}^t\vec{\tau}_1\, H_f\, \vec{\tau}_1}{\|\nabla f\|}, \qquad \kappa_2 = \frac{{}^t\vec{\tau}_2\, H_f\, \vec{\tau}_2}{\|\nabla f\|}. \tag{16.9}$$

Supplied with this theoretical background, we will now discuss the problem of meshing a domain defined by an implicit function. However, prior to focusing more closely on meshing techniques for implicit curves and surfaces, we will first introduce a general formulation of this problem which then makes it possible to find a general meshing scheme, in two and three dimensions.

16.2 Implicit function and meshing

We will first attempt to formulate the problem in a practical way, that is, in view of the envisaged application, the geometric meshing of an implicit domain for a numerical simulation using finite element methods. For the sake of simplicity, we will limit ourselves to the two-dimensional case only.

Problem statement

Let Γ be an implicitly defined curve, i.e., the solution to an equation of the form $f(x, y) = 0$, f being a differentiable function admitting 0 as a regular value.

The problem of discretizing Γ can be seen as the search for a polygonal curve $\widetilde{\Gamma}$ of same topology as Γ that forms a sufficiently accurate approximation of Γ. This problem can be formally described using the following conditions (see [Velho *et al.* 1997]):

- $\widetilde{\Gamma}$ is a piecewise linear approximation of Γ,
- there exists a homeomorphism[3] $h : \Gamma \longrightarrow \widetilde{\Gamma}$ such that:

$$\forall P \in \Gamma\,, \qquad d(P, h(P)) < \varepsilon\,, \tag{16.10}$$

where $\varepsilon > 0$ is a user-specified tolerance and d is the usual Euclidean distance in $\mathbb{R}^2$.

Remark 16.2 *The tolerance value ε corresponds to the quality of the geometric approximation of the curve geometry.*

In practice, we have seen that with a piecewise linear approximation, the lengths of the edges of the discretization being locally proportional to the radii of curvature (Chapter 14).

Geometric mesh

The problem we are interested in is to mesh an implicit curve in order to correctly approach its geometry. In other words, the metric to follow is of a purely geometric nature.

Desired properties and related problems. The first property desired is to assume a relative control over the curve, such that the length of the curve (the arc) and that of the chord locally approaching it (the underlying chord) are close. To this end, if h denotes the length of a segment and if δ measures the largest distance between this segment and the curve, then, for a given ε, we want to have the following condition:

$$\delta \leq \varepsilon\, h\,.$$

Notice that with parametric curves, the length of the arc can be easily obtained using the curvilinear abscissa s. With implicit curves, such a feature does not exist (the implicit function theorem allowing only a local parameterization). One way of evaluating the distance between the segment and the curve consists of sampling the segment and in calculating the largest distance between the sampled points and Γ:

$$\delta = \max_{P_i \in I} d(P_i, \Gamma)\,,$$

where I represents a small interval along the segment considered.

The second problem is related to the location of the points along the curve. From a given point M_0, it is necessary to find the series of points $M_i\,, i = 2, 3, ...$

[3]Thus ensuring that Γ and $\widetilde{\Gamma}$ have the same topology.

such that the length of the segments $M_i M_{i+1}$ is compatible with the given accuracy ε. It is obvious that the points of discontinuity (the singular points) must be part of the mesh (Chapter 14). We still have to find the number and the location of the other points. If we consider a neighborhood that is sufficiently small around the point M_0, we can apply the local study introduced in Chapter 14. This leads to locating the point M_1 at a distance $\alpha\,\rho(M_0)$ from the point M_0, where the coefficient α is given by Relation (14.7) and $\rho(M_0)$ is the radius of curvature of Γ at M_0. The point M_1 is then defined as the intersection of the circle of radius $\alpha\,\rho(M_0)$ centered at M_0 and the curve Γ.

General scheme. A geometric mesh of an implicit curve can be obtained in the following manner:

- identify the extrema (of the radii of curvature) of the curve and the singular points,
- initialize the mesh with these points,
- approach the curve using sub-curves corresponding to the curves joining two such consecutive points,
- mesh each piece using the previous principle.

The main difficulty of this approach is related to the calculation of the lengths of the portions of curve. An easy way consists of using a sufficiently small (uniform) sampling to evaluate the desired length numerically. Obviously, this method can be costly. A "binary-searching" type method or a "divide-and-conquer" type method (Chapter 2) can also be envisaged. We will study different possible approaches below.

Before going in more detail about the meshing of implicit curves, notice that for a given accuracy threshold ε, it is possible to obtain different meshes of the same implicit curve, depending on the nature of the meshing algorithm employed. Hence, the notion of optimal result must be considered.

Optimal mesh. In Chapter 1, we already mentioned the notion of an optimal mesh and indicated that this notion is related to the application envisaged (i.e., a mesh is optimal with respect to a certain criterion and not necessarily optimal for another criterion). Here, we will consider an optimal mesh to be that which corresponds to a good geometric approximation (see below) and which contains a minimal number of vertices. In fact, we consider the following.

Definition 16.3 *The* optimal *piecewise linear approximation is that which, among all possible solutions having the same quality of geometric approximation, minimizes the number of elements.*

However, this simple geometric criterion is not sufficient in practice to estimate the optimality of a solution (see also Chapter 18). Thus, for example, for a finite element computation, the number of elements of a mesh is an important factor (as it conditions the size of the matrices). Moreover, the (shape and size) element qualities are also important as they relate to the numerical accuracy of the results and the convergence of some computational schemes [Ciarlet-1978]. It is thus important to avoid the creation of (poor quality) badly-shaped elements.

In practical terms, it is convenient to consider as optimal a mesh that achieves an acceptable compromise between the different criteria considered. Thus, for isotropic triangles in two dimensions, a quality close to 1 defines well-shaped elements. Such a quality indicates that the lengths of the edges are close to 1 (possibly in a given metric). We bring the notion of optimal mesh down to the notion of unit mesh.

Definition 16.4 *In two dimensions, a* unit mesh *is a mesh whose edge lengths are close to 1.*

Such a mesh is considered to be optimal.

General principle

Implicit curves and surfaces offer less flexibility than parametric curves and surfaces. In particular, it is rather tedious to determine the intrinsic properties of these curves and surfaces. In addition, tracking an implicit curve is a difficult problem (we will see that, from a given point, numerical techniques make it possible to locate a neighboring point on the curve). This is why, meshing techniques are largely inspired by heuristics. In particular, the classical approaches (see for instance, [Allgower, Schmidt-1985] and [Allgower, Gnutzmann-1987]) consist of:

- sampling the domain of definition (using a covering-up of the domain),
- searching the roots of the function in the cells of the covering-up,
- constructing a topology (i.e., a mesh) whose vertices are the root of the function.

In other words, a spatial partitioning (a set of disjoint and congruent cells enclosing the domain) of the domain is created and the implicit function is locally approached in each cell of this covering-up. These approaches usually involve numerical techniques to find the points along the curve (surface) in a given cell.

Remark 16.3 *Notice that creating a sampling makes the problem a discrete one, the value of the implicit function being known at the vertices of the sample. In fact, most of the envisaged methods are capable of dealing directly with discrete data (for instance, those obtained using a scanning device).*

Remark 16.4 *Notice that this problem is slightly different from that aiming at reconstructing a topology from a set of points all belonging to the boundary of the domain (see [Hoppe-1994] for example).*

In the next section, we examine different approaches to constructing a geometric mesh of an implicit curve in two dimensions (the meshing problem for implicit surfaces will be dealt with in the following section).

16.3 Implicit curve meshing

As we have already mentioned, meshing an (implicit) curve consists of discretizing it into a finite number of segments of suitable lengths. It is obvious that these lengths depend on the envisaged application (i.e., on the constraints related to the application) and on the given metric information.

Construction of a spatial covering-up

Consider a given implicit curve Γ. The sampling stage consists of finding a set of points $P_i \in \Gamma$ sufficiently dense for the geometry of Γ to be approached within a tolerance of ε. The several techniques proposed can be classified into different classes, based on whether they proceed:

- by (exhaustive) enumeration,
- by continuation,
- by (adaptive) subdivision, etc.

To see this more clearly, we will first consider a naive approach to finding the points of Γ.

A "naive" approach (ray-tracing). To find the points on the curve, we can intersect Γ with a family $\mathcal{D}$ of lines (the domain is somehow sampled by a beam of lines).

Thus formulated, the problem consists of solving a set of one-variable equations of the type $f(x_i, y) = 0$ for a sample of uniformly distributed points $x_i \in \mathbb{R}$ (Figure 16.2).

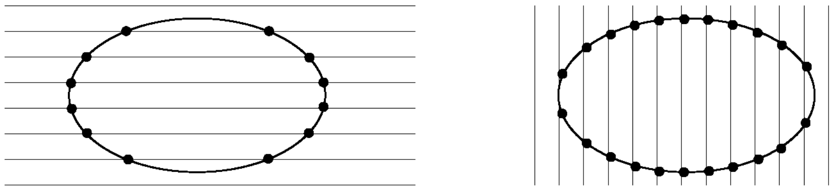


Figure 16.2: *Naïve method for an implicit curve. Notice that the same curve Γ leads to different samplings depending on whether the family of lines is horizontal (left-hand side) or vertical (right-hand side).*

This example simply illustrates the influence of the lines. Clearly, there does not exist any suitable criterion (i.e., working in all cases) to fix the size and the density of the rays *a priori*. In fact, some lines do not contribute to the sampling while others can "miss" some intersections.

Close to the naive approach, we then find an approach of the exhaustive enumeration type.

Exhaustive enumeration approaches. The existence of discrete data, supplied by scanning devices for instance, leads to a problem for which the spatial covering-up is given. Usually, this covering-up is a regular grid (uniform and axis-aligned) or a Coxeter-Freudenthal triangulation (each cell of a regular grid is split into two triangles, [Freudenthal-1942], [Coxeter-1963]), the values of the implicit function being known at its vertices.

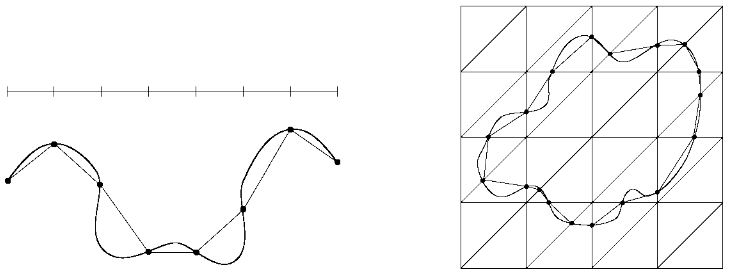


Figure 16.3: *Uniform approximation of a parametric curve (left-hand side) and of an implicit curve using a method of the exhaustive enumeration type (right-hand side).*

We can make here an analogy with the mesh of parametric curves: the segment representing the domain of the parameters is split into equally-sized sub-segments and the function f is used to find the vertices and to construct the polygonal approximation of the curve (Figure 16.3).

The principle of the *exhaustive enumeration* method consists of examining all cells of the partition and to process only those that are intersected by the curve. At the vertices of such a cell, the signs of the function are not constant. The curve Γ intersects a given cell side if the values of f at the two endpoints of this side are of opposite signs (Figure 16.4). The intersection points of Γ with a cell sides are either determined by a linear interpolation based on the values at the two edge endpoints, or using a procedure to find the roots (see below). We will see later that this technique allows us to approach the curve using a piecewise linear approximation $\tilde{\Gamma}$.

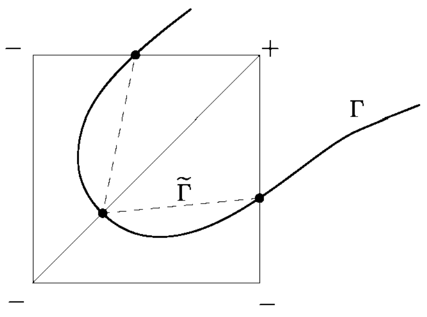


Figure 16.4: *Exhaustive enumeration method : intersection of the implicit curve* Γ *with two cells of the Freudenthal triangulation.*

As for the naive approach, the main difficulty of this type of approach (especially well-suited to discrete data) is to fix the resolution (i.e., the cell size) so as to capture the local behavior of the curve as well as possible. Again, too coarse a sampling can lead to intersections being missed or to reconstructing a topology that is different from that of the curve[4].

This leads us to envisage methods that allow the curve to be *tracked* (in a sense that we will specify later).

Methods by continuation. The techniques by continuation aim at finding, from a given point $M_i \in \Gamma$, a point M_{i+1} close to it on the curve. Depending on the manner of predicting the position of M_{i+1}, the methods are of the *progression* type or of the *prediction-correction* type.

- Methods by progression.

This concerns incremental methods. From a basic cell (i.e., containing a portion of the curve), the curve is approached by a set of cells, the partitioning being constructed "on the fly". Adding a cell can be performed by adjacency, the adjacency direction being determined by studying of the sign of the function at the vertices of the current cell (Figure 16.5). The neighboring cell is thus the cell sharing the edge of the current cell having its endpoints of opposite signs (the values of the implicit functions being evaluated at the cell vertices). The process stops as soon as the curve is fully enclosed in the cells. A coloring scheme can be used to avoid going back to cells that have already been analyzed. A stack (Chapter 2) is used to store the cells to be processed.

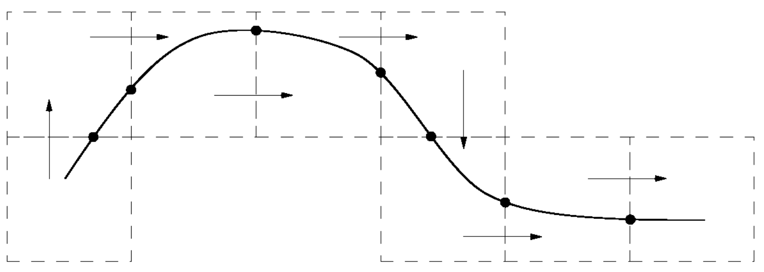


Figure 16.5: *Method by progression, the partitioning is constructed "on the fly" from a root cell.*

Such an approach is sensitive to the cell sizes which, if too coarse, do not allow the local behavior of the implicit curve to be captured (see the ambiguity problems below) and, if too fine, lead to a too great number of samples (thus penalizing further processing). Notice that, as for the naive approach, no criterion exists to fix the cell size *a priori*.

[4]However, too small a size is penalizing as it slightly increases the number of intersections checks and requires more accurate algorithms.

- Methods by "prediction-correction".

These methods allow us to calculate the position of a point on the curve from a known point $M_i \in \Gamma$ and a small displacement. The point M_i is "moved" along the tangent $\vec{\tau}(M_i)$ to the curve at M_i, to get a point M'_{i+1}. The position of this point (which does not belong to Γ) is then (iteratively) corrected, for example using a Newton-Raphson method (Chapter 11), to find a point $M_{i+1} \in \Gamma$ (Figure 16.6).

In this approach, the covering-up is virtual. One difficulty consists of fixing the stepsize of the displacement to avoid too fine a sampling or, on the other hand, to "miss" the curve (the correction method is not converging, especially in highly curved regions). Another problem consists of finding a starting point M_0 in each connected component of Γ.

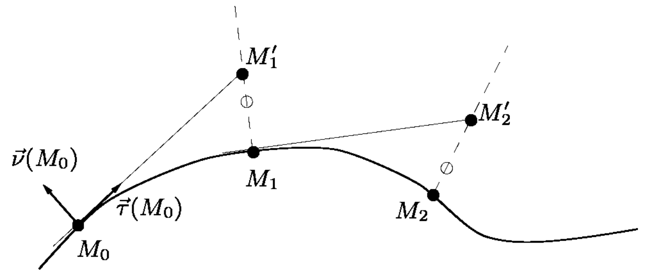


Figure 16.6: *Method by continuation of the prediction-correction type. From a position M_0, we determine a series of points M_1,..,M_n along the curve Γ.*

This type of method is based on the property of orthogonality (in two dimensions) of the gradient vector ∇f and the vector $\vec{H} = (-\partial f/\partial y, \partial f/\partial x)$. The latter is thus tangent to the level curves[5] of f and, in particular, to $f(x,y) = 0$. Given a point $M_0 = (x_0, y_0)$, the points of the connected component containing M_0 are solutions to a system of the form:

$$\begin{cases} \dfrac{dx}{dt} = -\dfrac{\partial f}{\partial y} & \text{and } x(0) = x_0\,, \\ \dfrac{dy}{dt} = \dfrac{\partial f}{\partial x} & \text{and } y(0) = y_0\,. \end{cases} \tag{16.11}$$

The position of the new point M_1 is estimated as $M'_1 = M_0 + \partial\vec{H}(M_0)$. The point M'_1 does not belong in principle to Γ. A correction scheme of the Newton-Raphson type aims at moving M'_1 back to a point M_1 of Γ.

Method by recursive subdivision. From the study of the previous approaches, we can observe that controlling the geometric approximation of Γ is rather delicate. We have seen that there does not exist a general criterion to fix, *a priori*, the size of the cells of the partition (supposed to be uniform so far). In particular,

[5]The level curves of f correspond to the curves defined by $f(x,y) = k$, $k \in \mathbb{R}$.

the curves with discontinuities of the order G^1 are difficult to approach with such methods.

On the contrary, it seems reasonable to suppose that if the cell size is variable and, for example, related to the local curvature, the approximation of the curve would be better (even without talking about a reduced number of elements). That is the basic idea of adaptive subdivision methods.

According to the principle of spatial decomposition methods (Chapter 5), from a bounding box of the domain, this type of approach constructs the partitioning of the domain in a recursive way. The bounding box is subdivided into four equally sized cells that can be organized hierarchically (using a *quadtree*, for example). The cell refinement is linked to a criterion related to the local curvature of Γ.

Among the useful criteria commonly used, we should mention:

- the *planarity* (i.e., the angle between the normals at the intersection points),
- the variation of the radii of curvature at the intersection points,
- the number of intersection points, etc.

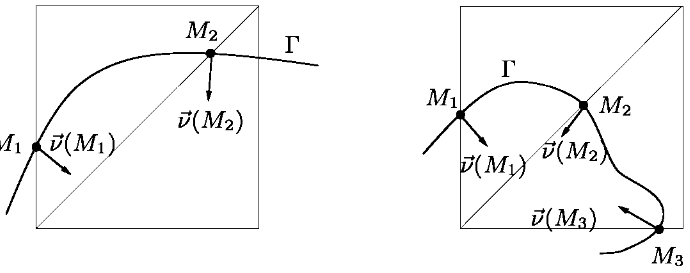


Figure 16.7: *Adaptive subdivision based on the evaluation of the local curvature. The normals $\vec{\nu}(M_i)$ at the intersection points M_i of Γ with the edges of the covering up are calculated and, from their variation (gap), we deduce a possible refinement of the cells (right-hand side).*

This type of approach again raises the problems already discussed in Chapter 5 for the meshing of the domain boundaries. Notice here, however, that the tree structure is not necessarily balanced (using the 2:1 rule, for example). Actually, the aim of the decomposition is to provide a sample of points belonging to Γ and which is sufficiently representative of the behavior of the curve.

Once the sampling has been done (and independently of the approach chosen), we must connect the points of Γ together in order to obtain a mesh of the curve.

Remark 16.5 *Notice that if the objective is to visualize the curve Γ, it is sufficient to create a cloud of points that is dense enough to capture the variations of the curve and to provide its visual aspect.*

Meshing an implicit curve from a point cloud

From the cloud of points defined at the previous stage, we now try to extract a mesh of the given curve. Notice immediately that this problem is very similar to that of the search for a curve passing through a given set of points (see [Hoppe *et al.* 1991], for example). However, unlike the latter[6], we have here some additional information provided by the spatial covering up.

Construction of a geometric mesh. Depending on the sampling technique adopted, the mesh creation can be more or less trivial. Thus, with a continuation method, the mesh is obtained by simply connecting the intersection points (the M_i's) two-by-two in the order of their creation. With the enumeration type methods, the task is more tedious. In this case, the discretization is obtained by analyzing each cell intersected by the curve and by connecting the intersection points belonging to the cell sides. It should be pointed out that some configurations can lead to topological ambiguities (Figure 16.8).

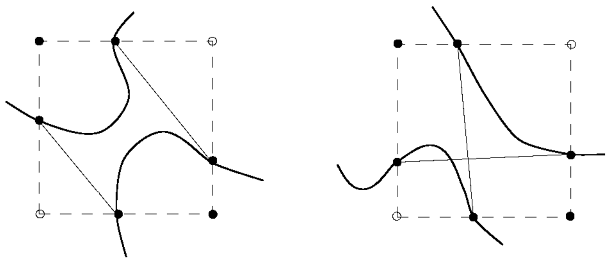


Figure 16.8: *Example of topological ambiguities: we illustrate here two ways of connecting the intersection points that violate the topology of the implicit curve.*

Exercise 16.1 *Identify the topological ambiguities possible with a enumeration type method and propose a simple way of getting rid of these ambiguities (hint: examine the sign of the function at the vertices of the partition and use the intrinsic properties of the curve).*

From a practical point of view, the mesh edges are created by using predefined patterns (*templates*) based on the sign of the implicit function at the cell vertices. For a uniform decomposition or a quadtree type decomposition, only the terminal cells intersected by the curve need to be considered. We thus identify $2^4 = 16$ distinct patterns.

At completion of this stage, the implicit curve Γ is thus approached by a polygonal segment. However, the geometric mesh obtained is not necessarily correct with a view to numerical computation (the size variations between two edges incident at a vertex might be great). A control on the edge size variation can be imposed. Here, we come up against a well-known problem.

[6]Which can sometimes be NP-hard, for example, when the goal is to find the closed polygon of minimal perimeter having the given points as vertices (traveling salesman problem).

Mesh of an implicit curve defined by a discretization. We find here a case where the metric to follow is not necessarily of a geometric nature. We must then intersect this metric (Chapter 10) with the geometric metric. The problem can be identified with the rather difficult curve remeshing detailed in Chapter 14. We can thus adopt a technique which consists of reconstructing a geometric support (a parametric arc this time) and then remeshing this support using classical optimization tools (point insertion, vertex removal and node relocation). However, we can also use the information extracted from this covering up (singular points, normals, tangents, etc.) to control the remeshing procedures. Notice that point insertion can be performed via the spatial covering up (to facilitate root searching).

In this section, we have mentioned root searching several times (i.e., the intersections between the curve and a given edge). Before dealing with the surface meshing, we will go back to several practical aspects of curve meshing.

Computational aspects of implicit curve meshing

In this short section, we briefly mention several practical aspects of the implicit function meshing and especially the search for intersection points and the approximate calculation of the gradient of the function at a given point.

Root finding. The methods previously discussed all rely on the identification of the intersection points between an edge of the covering up and the curve Γ. When the implicit function is continuous and monotonic, the intermediate value theorem ensures the existence of (at least) one solution along the edge AB, if the values $f(x_A, y_A)$ and $f(x_B, y_B)$ of the function at these points are of opposite signs. If the derivative of the function is known exactly, a Newton type method can be used to find the root, that is, the point $P(x, y)$ such that $f(x, y) = 0$. However, this method can be quite impredictable and may not converge for some functions.

In all cases, a binary searching method allows us to quickly find the root within a given tolerance ε. The tolerance is usually based on a fraction of the edge length or on a maximum number of iterations [Bloomenthal-1988]. In some special cases, the algorithm can be modified so as to improve the localization of the intersection point, notably when the function f vanishes on a sub-segment of AB (Figure 16.9 and see [Blinn-1982], [Frey, Borouchaki-1996], for example).

When the curve Γ is known in a discrete way (i.e., the corresponding implicit function is not known explicitly), the intersection points may be approached using a linear interpolation based on the given values of the function. Given an edge AB for which the values $f(x_A, y_A)$ and $f(x_B, y_B)$ are of opposite signs, a linear interpolation along AB makes it possible to find the value of f at $P(x, y)$ (corresponding to the parameter t) using the formula:

$$f(x, y) = (1 - t)\, f(x_A, y_A) + t\, f(x_B, y_B)\,, \tag{16.12}$$

where the function f varies linearly between $f(x_A, y_A)$ for $t = 0$ and $f(x_B, y_B)$ for $t = 1$.

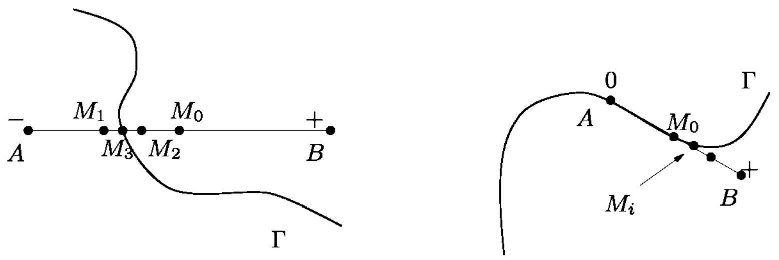


Figure 16.9: *Search for the intersection point using a binary searching method along the edge AB. Left-hand side; the intersection point is identified as the point M_3. Right-hand side; the vertex A is a root, but the modified algorithm makes it possible to find a point M_i as the intersection point.*

Remark 16.6 *The detection and the identification of singular and critical points can be performed during the binary searching of the roots by modifying the corresponding algorithm in order to take into account information regarding the gradient of the function at each of the evaluation points [Attili-1997], [Schöberl-1997].*

Estimation of the gradient. At a regular point $P = (x, y)$, the normal $\vec{\nu}(P)$ to the curve Γ can be estimated as the unit gradient vector at this point. If the partial derivatives are not known analytically, the gradient can be approached numerically, using a finite difference scheme:

$$\langle \nabla f(P), \delta(P) \rangle \approx f(P + \delta(P)) - f(P)\,, \qquad (16.13)$$

where $\delta(P)$ represents a small size (generally chosen as a fraction of the cell size of the covering up) vector. A centered difference scheme, which is also possible, then gives:

$$\langle \nabla f(P),\, 2\,\delta(P) \rangle \approx f(P + \delta(P)) - f(P - \delta(P))\,. \qquad (16.14)$$

Posing $\delta(P)_i = h e_i$ where e_i is the unit vector related to the i-axis and h is a small value, we obtain, for the first approximation:

$$\nabla f(P) \approx \frac{1}{h} \begin{pmatrix} f(P_1) - f(P) \\ f(P_2) - f(P) \end{pmatrix},$$

where $P_i = P + h\, e_i$ and this approximation relates the error to h while the second approximation relates the error to h^2. Depending on the value of h, either of these approximations can be chosen.

Remark 16.7 *At singular points, where the gradient is not defined, the normal can be obtained as a mean value of the normals at the neighboring vertices.*

We will now show that mesh generation for implicit surfaces involves similar approaches to those used for meshing implicit curves.

16.4 Implicit surface meshing

The techniques used to mesh an implicit surface derive from the techniques developed for meshing implicit curves. However, several problems that are particular to surfaces will be dealt with in this section.

For the same reasons as in two dimensions, the meshing of implicit surfaces relies on a sampling stage and a mesh construction stage given a point cloud. Constructing a spatial covering up makes it possible to localize the processes. In other words, the goal is to capture locally the behavior of the surface in a small volume element. Hence, each element of the covering up intersected by the surface is analyzed and the roots of the function (i.e., the points of Σ) are identified along the edges of this element.

Construction of a spatial covering-up

The objective is here to find a covering up of the domain of study that allows us to extract a set of points $P_i \in \Sigma$ which is sufficiently dense for the geometry of the surface Σ to be approached within a given tolerance ε. As in two dimensions, we have the following classification of spatial covering methods:

- methods by exhaustive enumeration,
- methods by continuation,
- methods by adaptive subdivision.

Methods by exhaustive enumeration. In this type of approach, the covering up $\mathcal{R}$ is usually an input of the problem and this covering up is then a regular grid or a triangulation (Coxeter-Freudenthal type, for example). This is notably the case when the implicit surface is an iso-surface defined in a discrete way using scanning devices (scanners). The implicit function is sampled and its value is known at the nodes of a lattice of (usually cubical) cells, that is both structured (for each point, the number of adjacent points is constant) and uniform (the distance being two points is constant). The principle of an *exhaustive enumeration* method consists of:

- identifying the elements of $\mathcal{R}$ intersected by the surface and
- locating the intersection points (of Σ with the edges of the elements of $\mathcal{R}$).

The surface Σ will then be discretized using a piecewise linear approximation $\widetilde{\Sigma}$ in each element of $\mathcal{R}$.

As in two dimensions, this type of approach is sensitive to the grid resolution (this being given or constructed). Figure 16.10 shows the influence of the element size on the accuracy of the geometric approximation.

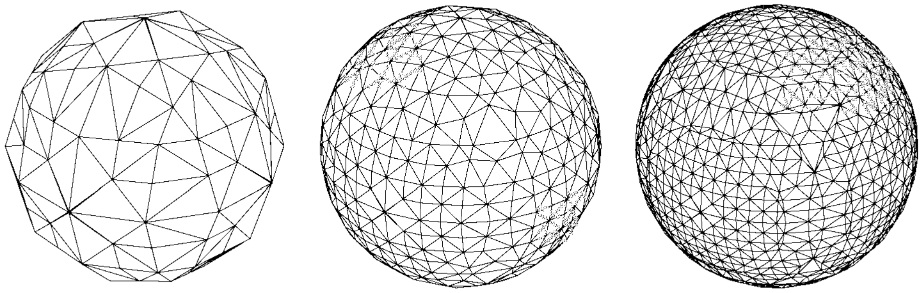


Figure 16.10: *Discretizations of a sphere by successive refinements. At each stage, the covering up is uniform (i.e., formed by equally-sized cells).*

Continuation methods. Continuation methods for surfaces are on all points similar to continuation methods for curves in two dimensions. We thus find methods by *progression* and methods by *prediction-correction.*

Recall just that a progression method consists of starting from an element of $\mathcal{R}$, the *seed* and progressing by adjacency towards the neighboring elements. The direction of the progression is defined by the sign of the values of the function at the vertices of the current element. A stack is used to store the elements to be processed. In principle, this method makes it possible to process a single connected component for a given seed.

As for implicit curves, prediction-correction type methods can be applied successfully to "track" an implicit surface. From an initial point M_0, we look for a point M_1' obtained by a small displacement of M_0 in the tangent plane $\Pi(M_0)$ associated with M_0. The point M_1' *predicted* is then iteratively relocated onto the surface to a point M_1, via a *correction* stage based on a Newton-Raphson procedure.

Remark 16.8 *The tedious part corresponds to the determination* a priori *of a seed for each connected component. Moreover, the set of vertices obtained is not intrinsically ordered which may result later (during the structuration stage) in overlapping elements.*

Adaptive subdivision methods. For the same reasons as in two dimensions, it is desirable to construct a covering up $\mathcal{R}$ that is representative of the intrinsic properties of the surface.

An adaptive subdivision method consists of including the domain of study into a bounding box which is recursively subdivided into identical elements[7]. We thus obtain a structure that is naturally organized hierarchically, an *octree* for example in the case where the basic element is a cube (Chapter 5).

[7]The requisite of getting identically shaped elements is justified by the need to avoid the degeneracy of volumes resulting from the subdivision.

Remark 16.9 *A covering up based on tetrahedral cells is also possible. Hence, the* Kuhn tetrahedron, *based on the points* $(0,0,0)$, $(1,0,0)$, $(1,1,0)$ *and* $(1,1,1)$, *can be subdivided into similarly shaped tetrahedra. The subdivision procedure leads to eight sub-tetrahedra, four of which share a vertex of the initial tetrahedron and the four others being internal (Figure 16.11).*

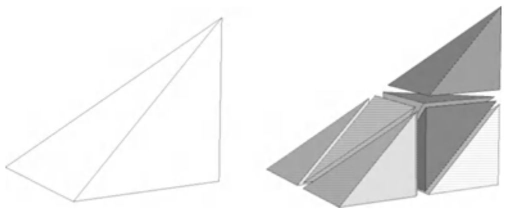


Figure 16.11: *Decomposition of Kuhn's tetrahedron (left-hand side) into eight tetrahedra (right-hand side).*

Exercise 16.2 *Enumerate the tetrahedra of Kuhn's decomposition. Show that the same decomposition applied on the tetrahedron based on the points* $(0,0,0)$, $(1,0,0)$, $(0,1,0)$ *and* $(0,0,1)$ *does not lead to similarly shaped elements.*

An element of $\mathcal{R}$ is subdivided whenever a specific criterion is not satisfied for this element. The criteria considered for refining an element of $\mathcal{R}$ are mainly based on [Schmidt-1993]:

- the variation between the normals at the vertices (or the faces) of the current element (*planarity* criterion):

$$\max_{P_i} \langle \vec{v}_i, \vec{\nu}(P_i) \rangle < \cos(\varepsilon_1),$$

  where $\vec{v}_i$ represents the unit vector supported by the segment P_iP_{i+1}, $\vec{\nu}(P_i)$ is the unit normal at P_i and ε_1 is a given tolerance,

- the *divergence* of the normals at the vertices P_i with respect to the normal at the center of the element M, measured by the quantity:

$$1 - \min_{P_i}(\langle \vec{\nu}(P_i), \vec{\nu}(M) \rangle < \cos(\varepsilon_2),$$

- the changes between the signs of the values of the function at the vertices,

- the proximity of a singularity, etc.

We will see in Chapter 19 that some criteria can also be used to optimize the surface meshes.

With this type of approach, the subdivision leads to a partitioning for which the element density (and thus that of the sampling points) is proportional to the local curvature (for a geometric triangulation, that is, for which the geometric approximation to the surface is controlled by a given tolerance value ε). However, the discontinuities of order C^0 of the surfaces are more tedious to capture exactly and require more sophisticated algorithms. Figure 16.12 illustrates the problem of searching for the intersection points along the edges of the elements of $\mathcal{R}$.

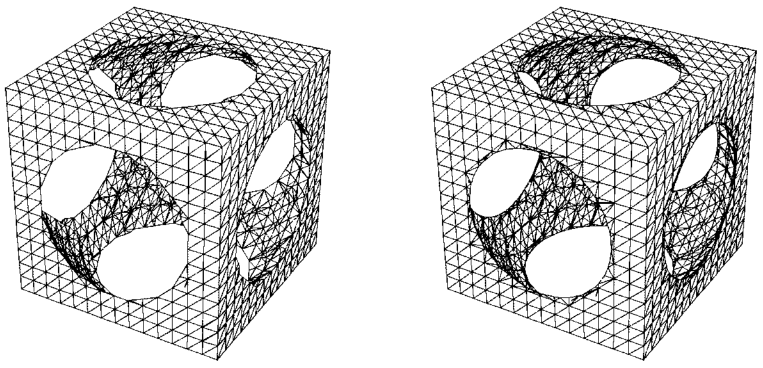


Figure 16.12: *Example of root finding,* "wiffle cube". *Left-hand side: a linear interpolation is used to find a root along an edge of the decomposition. Right-hand side: a binary searching algorithm is used to find a change of sign of the function along the edge, thus improving the approximation of the sphere.*

Remark 16.10 *Another approach consists of starting from a coarse partition of the space into tetrahedra. The latter are analyzed to decide on a possible refinement, according to the criteria mentioned. The difference lies in the fact that when a tetrahedron is subdivided, its barycenter (or any other suitable point) is created and inserted in the current triangulation using the* Delaunay kernel *procedure (Chapter 7) [Frey, Borouchaki-1996]. This approach constructs, incrementally, a triangulation of the domain conforming to the Delaunay criterion.*

We now deal with the construction of a surface mesh from the point cloud obtained at completion of the sampling stage.

Mesh of an implicit surface defined by a point cloud

At this stage, we have a cloud of points located on the surface, as dense as the partitioning was fine. We now have to connect these points in order to obtain a geometric mesh of the surface. It seems obvious that the tedious aspect of this operation consists of making sure that the topology of the discretization conforms to that of the implicit surface it represents.

For the sake of convenience, we will detail here the construction of a mesh where the cloud of points comes from a enumeration type method[8] or an adaptive subdivision method. We will leave it up to the reader to see how a mesh may be obtained from a continuation method [Bloomenthal-1988], [Wyvill *et al.* 1986].

Construction of a geometric mesh. When the covering up $\mathcal{R}$ is a uniform grid, the connectivity of $\mathcal{R}$ allows us to use a very simple (and nowadays very popular) algorithm to construct a surface mesh. This algorithm has also proved to be especially well-suited for processing discrete data [Lorensen, Cline-1987].

[8]This choice is also justified by the fact that this type of technique is widely used in practice.

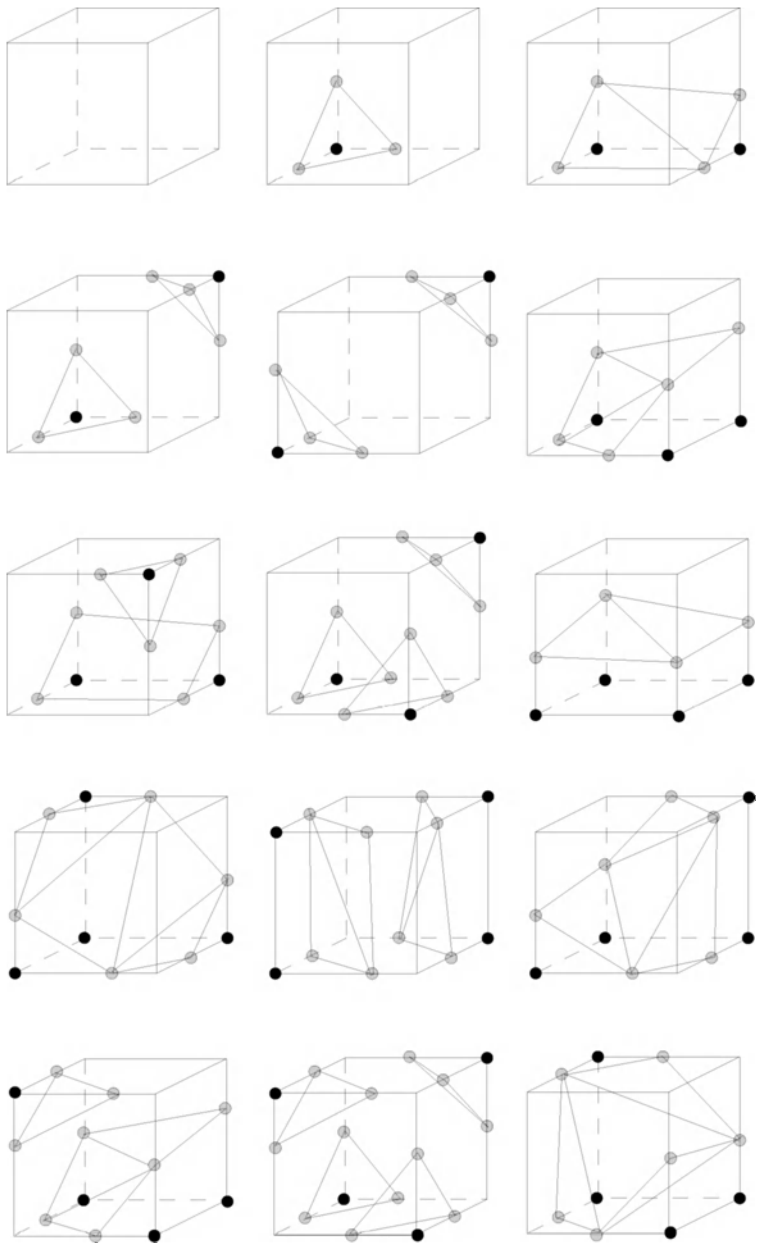


Figure 16.13: "Marching Cubes" *algorithm: surface-cell intersections. The* $2^8 =$ 256 *possible patterns can be reduced to a set of 15 configurations using different properties of symmetry and rotation preserving the topology of the triangulated surface (notice here that the polygons having more than three vertices have been subdivided into triangles).*

- "*Marching Cubes*" algorithm

Based on a *divide and conquer* approach, the method consists of analyzing any element of $\mathcal{R}$ intersected by Σ (the identification of an element K being based on the study of the signs of the function at the vertices of K). Each vertex can be either positive or negative (the case where the value is zero is a peculiar case that can be resolved by *dilating* the surface locally), the eight vertices of a given element K allowing us to construct an index with a value in $]0, 256[$, (as $256 = 2^8$).

In practice, each value of the index is associated with a list of polygons used to locally approach the surface. The 255 potential lists can be reduced to 15 representative cases, using symmetric and rotational properties. These 15 cases lead to 15 predefined patterns (*templates*) that serve to define a piecewise linear approximation of the surface in each element.

Exercise 16.3 *Retrieve the cases representative in Figure 16.13 from the 256 possible cases.*

Such an algorithm constructs polygons that have 3 to 6 vertices (Figure 16.14). To get a mesh composed only of triangles, we have to subdivide the polygons of a higher degree than 3 into triangles. This involves a combinatorial procedure (to find all the possible topologies) as well as a geometrical procedure (to choose from all the possible ones, the one that leads to the best geometric approximation).

Theoretically, *Catalan's number* of order n

$$Cat(n) = \frac{(2n-2)!}{n!(n-1)!},$$

is a formula giving the number N_n of different triangulations of a polygon with n vertices: $N_n = Cat(n-1)$. Thus, when $n = 3, 4, 5$ or 6, we find respectively $N_n = 1, 2, 5, 14$ (Chapter 18).

The drawback of the method is related to the fact that topological ambiguities and/or non-closed surfaces can be created. This is the case when an element of $\mathcal{R}$ contains a face in which vertices of opposite signs are diagonally opposed two-by-two (Figure 16.15) [Dürst-1988], [van Gelder, Wilhelms-1992], [Montani *et al.* 1994].

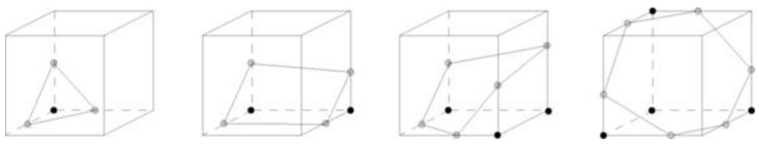


Figure 16.14: *Different types of polygons constructed using predefined patterns ("templates") by a* "Marching-Cubes" *algorithm, if the numbers of vertices are 3,4,5,6, from left to right, respectively.*

Several techniques enable us to remedy this problem, for example using:

- the value of the function at the center of the face [Wyvill *et al.* 1986] (although this solution may fail [Matveyev-1994]),

- a bilinear representation of the function, a hyperbolic curve describing the intersection of Σ with an edge, the value of the function at the intersection points of the asymptotes of the hyperbola making it possible to predict the topology [Nielson, Hamann-1991].

Notice, however, that these approaches cannot in practice be used for discrete data.

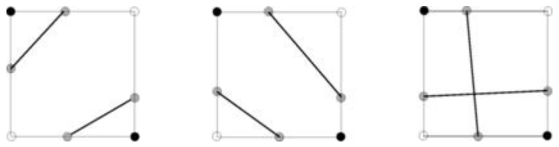


Figure 16.15: *Faces presenting a topological ambiguity. Several connections are possible between the vertices.*

Adaptive subdivision methods. In this type of approach, the elements of the spatial partitioning $\mathcal{R}$ are all of the same type (cubes or tetrahedra) although their shapes and sizes can vary locally, based on the local properties of the surface.

If the partitioning is represented by an octree, the construction of the geometric mesh is close to that used in the "*Marching Cubes*" algorithm (see, for instance, [Wilhelms, van Gelder-1990], [Frey-1993]).

When the partitioning is simplicial, vertices of opposite signs can be separated by a single plane, thus leading to only $2^4 = 16$ possible configurations, the only polygons formed being triangles and quadrilaterals (Figure 16.16). The quadrilaterals obtained can then be subdivided into triangles according to geometric criteria (Chapter 19). The degenerate cases (a tetrahedron vertex belongs to the surface) can be avoided by locally *expanding* the surface [Frey, Borouchaki-1996].

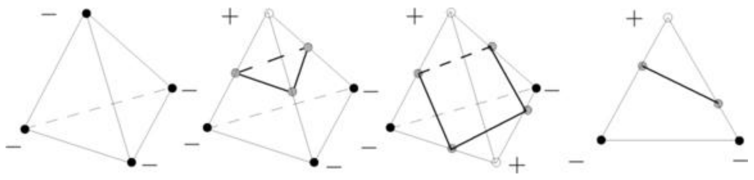


Figure 16.16: *Different triangulations possible, depending on the sign at the vertices, of a cell for a simplicial covering up.*

Meshes obtained by such approaches are geometric meshes. However, the size variations between neighboring elements are not controlled, thus this type of mesh may not be suitable for a finite element type calculation. Here we are confronted again with the problem of surface mesh optimization.

Surface mesh optimization

The purpose here is not to specify the optimization operations used for surface meshes which will be described in Chapter 19, but to present the context of such an optimization.

Shape quality optimization. The objective is to optimize a pertinent quality criterion with respect to the envisaged application (here a finite element type calculation). A shape quality measure for a triangle K is given by the formula:

$$Q_K = \alpha \frac{h_{max}}{\rho_K}, \tag{16.15}$$

where h_{max} represents the diameter of the element and ρ_K the radius of the inscribed circle (α is a normalization coefficient so that $Q_K = 1$ for an equilateral triangle). The goal is to obtain a quality value close to 1 for all mesh elements. To this end, topological modification operators are used (which preserve the point locations but modify their connections) as well as metric modifications (which modify the points locations while preserving their connections).

Mesh simplification. The number of elements in a geometric mesh is related to the gap between the element and the underlying surface. However, depending on the application envisaged, too great a number of elements can be penalizing. *Simplification* (or decimation) methods reduce the number of elements of a mesh while preserving the quality of the geometric approximation (Chapter 19). Such methods involve the same modification operators as optimization procedures.

Computational aspects of implicit surface meshing

In some applications (such as computer graphic visualization or when the surface mesh is the input of a volumetric meshing technique), the consistent orientation of the triangulation can be a very important requirement (Chapter 6).

Orientation of the triangles. The orientation of the triangles in the surface mesh can be performed either *a priori*, or *a posteriori*. In the first approach, the polygons of the patterns used to mesh are oriented counterclockwise (for example) based on a canonical orientation at the cell level (Figure 16.17 for a tetrahedron). In the second approach, the polygons are oriented by adjacency in a consistent way, in each connected component (see Chapters 1 and 2 for data structures and algorithms appropriate to this type of process).

Memory resources and data structures. Memory resources correspond to the data structures necessary to store the information related to the mesh (nodes, triangles, etc.) as well as the structures related to the spatial partitioning. The main internal data structures contain:

- an array of mesh vertices (coordinates),

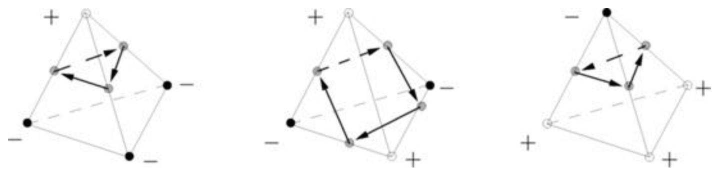


Figure 16.17: *Consistent orientation of polygons in a tetrahedron.*

- an array of triangles (list of vertices),
- an array for the neighboring elements (edge adjacency),
- an array for the vertices of the covering up,
- an array for the elements of the covering up,
- additional resources (for example, to store the value of the function at the vertices of the covering up), etc.

Examples of surface meshes

To conclude this section, we provide several examples of implicit surface meshes created by some of the methods described above. To help the understanding of these meshes, Table 16.1 indicates some characteristic values, np, ne denote respectively the number of vertices and elements of the triangulation and Q_M, Q_{pire} indicate the mesh quality and the quality of the worst element before optimization.

-	np	ne	Q_M	Q_{pire}
case 1	6,656	12,816	1.88	22.07
case 2	36,208	72,432	3.57	190.
case 3	122,940	244,431	4.02	925.

Table 16.1: Statistics related to several examples of implicit surface meshes.

The evaluation of (implicit) surface meshes is based on geometric criteria related to the quality of the geometric approximation (see [Frey, Borouchaki-1998] and Chapter 19). On the other hand, depending on the envisaged application, (i.e., a finite element calculation), it is important to guarantee a good element shape quality.

The example in Figure 16.18 corresponds to the reconstruction of an iso-surface using an exhaustive enumeration method based on a regular grid.

The second example (Figure 16.19) presents different meshes of an implicit surface of degree six corresponding to the equation [Schmidt-1993]:

$$f(x,y,z) = (x^2+y^2-4)(x^2+z^2-4)(y^2+z^2-4) - 4.0078 = 0\,.$$

The initial mesh (left-hand side) has been created using an adaptive partition in tetrahedra. Notice that the geometric approximation of the surface is correct,

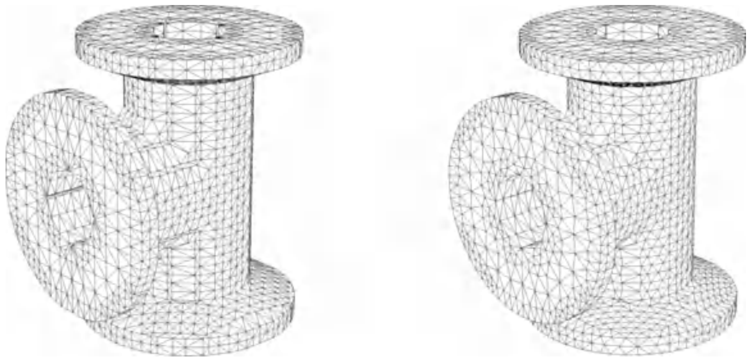


Figure 16.18: *Exhaustive enumeration method applied to the reconstruction of a surface from a regular grid. Initial mesh (left-hand side) and optimized mesh (right-hand side).*

although the shape quality of the mesh is not acceptable (at least for a finite element calculation). Hence, this mesh has been optimized (middle) and then simplified (right-hand side).

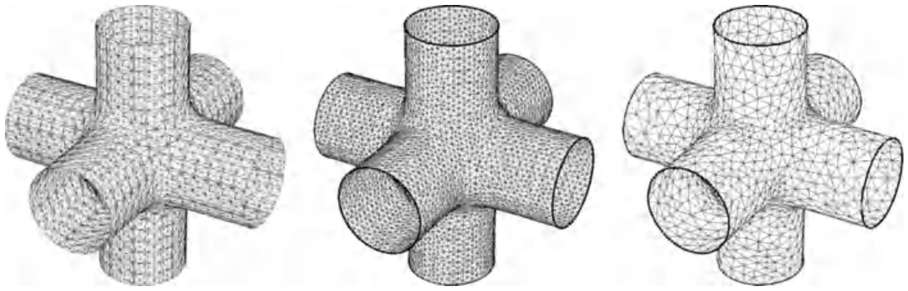


Figure 16.19: *Different meshes of an implicit surface: initial geometric mesh obtained by an adaptive method (left-hand side), mesh optimized with respect to the triangle shape quality (middle) and simplified geometric mesh (right-hand side).*

Finally, the last two examples (Figures 16.20 and 16.21) show iso-surface meshes for biomedical applications[9] created from discrete data.

In the example in Figure 16.20 it seems obvious that the original mesh (left-hand side) contains too many elements to be numerically exploitable (67, 106 vertices and 134, 212 triangles). Therefore, a simplification procedure is applied to obtain a geometric mesh where the density of the elements is related to the local curvature of the surface [Frey, Borouchaki-1998].

[9]Here, the iso-surfaces are associated with anatomic structures.

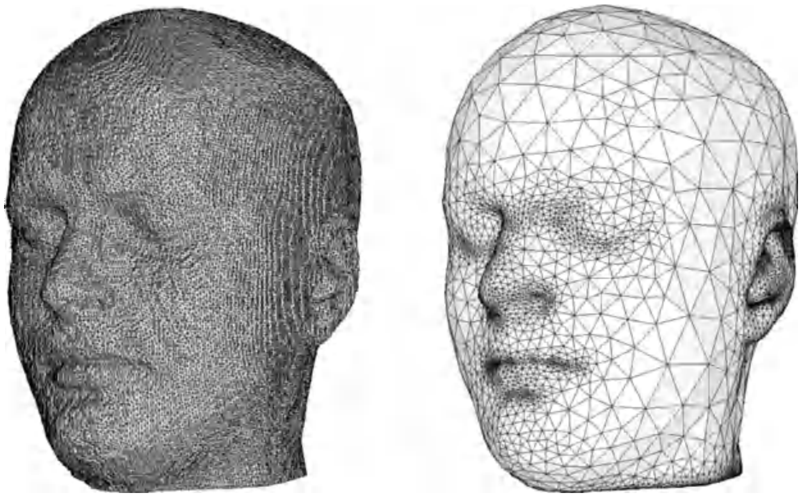


Figure 16.20: *Mesh of an iso-surface of a head from volumetric data (left-hand side, data courtesy of Mika Seppa, Low Temp. Lab., University of Technology, Helsinki, Finland) and simplified and optimized mesh (right-hand side).*

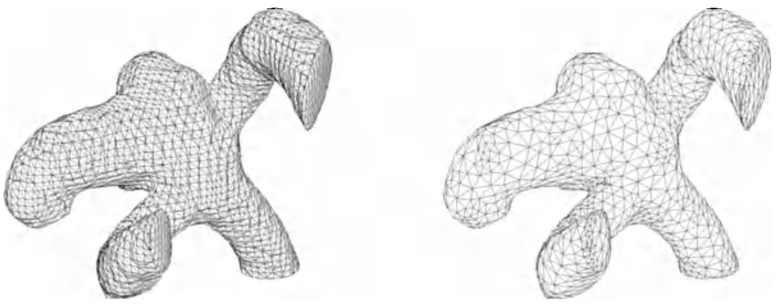


Figure 16.21: *Biomedical iso-surface meshes (an anevrism of the central cerebral artery) from discrete data. Left-hand side: initial mesh obtained using a* "Marching Lines" *algorithm [Thirion-1993]. Right-hand side: optimized and simplified mesh.*

16.5 Extensions

To conclude this chapter, we will briefly evoke the possibility of using implicit functions for modeling purposes. We will then mention the construction of meshes for such domains.

Modeling based on implicit functions

Traditionally, we can distinguish three geometric modeling systems, depending on whether they:

- are based on a boundary representation of the solid (a *B-Rep*), the objects being represented by the union of their boundaries,
- use a spatial decomposition, the domain being approached by the union of the internal cells or the cells intersected by the domain boundary,
- involve constructive geometry (CSG), the domain being represented by a tree structure in which the leaves are the primitives and the nodes correspond to Boolean operations.

Constructive solid geometry. Constructive solid geometry (CSG) stems from the work of [Rvachev-1963] related to R-functions, in the context of the numerical resolution of problems in complex domains. In this type of representation, the domain is defined via a unique function, which is at least C^0-continuous, with real values $f : \mathbb{R}^d \longrightarrow \mathbb{R}^+$, called the *representation function* (*F-Rep*). This function can be defined either algorithmically (that is encoded with an algorithm that returns the value of the function at a given point), or in a discrete manner, for instance, at the nodes of a regular grid [Pasko *et al.* 1995]. With this approach, the topology of the domain is preserved both implicitly (within the tree structure) and explicitly (by the primitives).

A functional representation. If we consider a closed domain defined by such a function f, we assume that:

$$\left\{\begin{array}{rcl} P \in \overset{\circ}{\Omega} & \Longleftrightarrow & f(P) > 0 \\ P \in \partial\Omega & \Longleftrightarrow & f(P) = 0 \\ P \in \mathbb{R}^d \setminus \Omega & \Longleftrightarrow & f(P) < 0 \end{array}\right. \tag{16.16}$$

which introduces a *classification* of the points of $\mathbb{R}^d$. Hence, the points P internal to the domain are those verifying $f(P) > 0$, whereas the boundary points are such that $f(P) = 0$. We can see how this approach relates with implicit curves and surfaces introduced in the previous sections.

Primitives. In CSG, numerous geometric forms can be considered as primitives, from the simplest ones to the more "exotic" ones, from planes and quadrics (spheres, cones, cylinders, ellipsoids, etc.), to superquadrics (Figure 16.22).

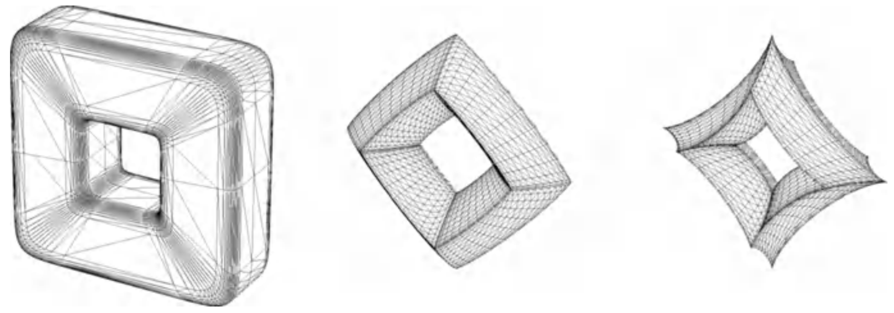


Figure 16.22: *Examples of superquadric surfaces (supertorus) that can be used as primitives in CSG.*

More generally, each *primitive* can be expressed as a special instance (occurrence) of a function chosen from a finite set of possible types. Hence, for instance, the equation:

$$f(x,y,z) = r^2 - \left((x-c_x)^2 + (y-c_y)^2 + (z-c_z)^2\right) = 0$$

defines, in three dimensions, the primitive corresponding to the representation of a sphere of radius r and centered at the point $C = (c_x, c_y, c_z)$.

Boolean operations. As we have already mentioned, constructive geometry is based on the combination of primitives using boolean unary and binary operations. In this context, the set operations is replaced by operations related to the corresponding functions.

For example, the *union* and the *intersection* of two primitives, defined by the function f_1 and f_2, can be written as follows[10]:

$$f_1 \cup f_2 = f_1 + f_2 + \sqrt{f_1^2 + f_2^2} \quad \text{and} \quad f_1 \cap f_2 = f_1 + f_2 - \sqrt{f_1^2 + f_2^2}\,. \qquad (16.17)$$

Notice that these functions C^k-continuous ($k \geq 1$) present C^1 discontinuities when $f_1 = f_2 = 0$.

See [Pasko *et al.* 1995] for the definition of several other operations.

Implicit domain meshing

We have seen that an implicit domain is one whose boundary is an implicit curve (surface). Given this remark, two approaches are possible to mesh such a domain:

- the *hierarchical* approach which consists of meshing the boundaries of the domain using one of the meshing techniques described here, prior to providing the resulting mesh to a "classical" meshing technique (Chapters 5 to 7),

[10]Using the symbols relative to the underlying sets.

- the *global* approach which considers the mesh generation for implicit domains as an extension of the meshing techniques for implicit curves and surfaces.

We will now briefly explain the global approach.

Global approach. Enumeration or adaptive subdivision methods are more adapted to this problem than continuation methods, mainly because the latter "track" the boundary and "forget" the elements not intersected by the boundary. The creation of the point cloud and then of the boundary mesh (curve or surface) is thus considered to be resolved.

The internal point creation and thus of the elements is related to the type of the spatial partitioning used. Hence, in three dimensions, when the covering up is of composed of *voxels* (uniform regular grid), or of the *octree* type, predefined patterns can be used [Frey *et al.* 1994]. For a simplicial covering up, the *texel* approach proposed by [Frey *et al.* 1996], consists of introducing a point in each simplex (this point is chosen so as to guarantee a suitable mesh gradation) and in using the *Delaunay kernel* as the element creation procedure (cf. Chapter 7).

Finally, an optimization procedure based on the element shape quality is applied, first on the boundary elements, then on the interior of the domain (see Chapters 18 and 19 for more details about the modifications used).

Chapter 17

Mesh Modifications

The aim of this chapter is to review some methods designed for mesh modification and manipulation (except for those concerning mesh optimization which will be discussed in the next two chapters). Provided with one or several meshes we are concerned with the methods that manipulate this (these) mesh(es) in various ways. Among the points of interest here, we first review mesh transformations with a geometric nature, the way in which two meshes can be merged into a single one, refinement techniques and various operations related to the attributes associated with the mesh in question.

★
★ ★

Geometric mesh transformations are briefly reviewed together with transformations resulting in a global or a local mesh refinement and methods for geometric type change. In addition we discuss how to merge two meshes (sharing a common boundary part). Then, we give some information about node construction and node numbering for elements other than P^1 elements (namely where the nodes can be different from the vertices). Various representative finite elements are listed to illustrate how to construct (to number) the nodes. Then we focus on how to optimize, in some sense, both the vertex (node) and the element numbering in a mesh. To conclude the chapter, various other aspects are discussed (including how to manage the mesh physical attributes properly, how to use two different meshes, etc.).

17.1 Mesh (geometric) modifications

Depending on the geometry of the domain we want to mesh and the capabilities of the mesh generation method that is used, it is often tedious to construct a mesh that satisfies certain repetitive properties (such as symmetry) enjoyed by the domain itself.

Thus, one way to achieve such repetitive features is to design the mesh generation process by defining explicitly what is expected. In other words, a domain with a symmetry (if we consider this example) is split into two parts. One of these is effectively meshed (using a mesh generation method), then a symmetry is applied to the resulting mesh and both meshes are merged in order to complete a mesh of the whole domain that obviously includes this symmetry.

Popular geometric transformations

For the sake of simplicity, we only consider P^1 meshes (the element nodes are the element vertices, the element edges are straight and the element faces (in three dimensions) are planar; see Definitions (17.1) and (17.2) below.

Given a mesh, the goal is to create a new mesh resulting from a classical geometric transformation (symmetry, translation, rotation, isotropic or anisotropic dilation or a more general transformation explicitly defined by a transformation matrix).

If the transformation corresponds to a *positive isometry*, it only affects the position of the mesh points; thus both the connectivity and the numbering of the elements remain unchanged.

Otherwise, a transformation corresponding to a *negative isometry*, acts on the position of the mesh points and, in our situation, on the numbering (or connectivity) of the element vertices, in such a way that the resulting elements maintain positive surface areas (in two dimensions), or positive volumes (in three dimensions). To achieve this, a permutation of the list of the element vertices must be carried out (for example, a triangle with vertices $(1,2,3)$ must be transformed into the triangle with vertices $(1,3,2)$). The different transformations of interest are:

- symmetries with respect to a line or a plane,
- translations or shifts of a given vector,
- isotropic or anisotropic dilations about a given center, whose dilation coefficients are given,
- rotations of a given angle around a point or an axis (in three dimensions),
- or general transformations (given through their explicit matrices).

In addition, any combination of these operators defines a new transformation.

In practice, a transformation can be defined using a matrix $\mathcal{T}_{ra}$. Thus, if P is a vertex in the initial mesh, the corresponding vertex, P', is obtained as:

$$P' = \mathcal{T}_{ra}(P). \tag{17.1}$$

This simply means that we have, in $\mathbb{R}^2$:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = [\mathcal{T}_{ra}] \begin{pmatrix} x \\ y \end{pmatrix},$$

with $\mathcal{T}_{ra}$ a 2×2 matrix (or more generally, a $d \times d$ matrix, in d dimensions).

Nevertheless, in order to give an explicit and easy description of the transformations considered, we use a coordinate system where these coordinates are homogenous (i.e., vertex with coordinates (x, y) is seen as the triple $(x, y, 1)$). The matrix is then a $(d+1) \times (d+1)$ matrix (regarding these matrices, see [Rogers, Adams-1989]). Hence, in two dimensions:

$$\mathcal{T}_{ra} = \begin{bmatrix} 1 + A^2F & ABF & ACF \\ ABF & 1 + B^2F & BCF \\ 0 & 0 & 1 \end{bmatrix} \text{ with } F = \frac{-2}{A^2 + B^2}$$

corresponds to a symmetry with respect to the line $Ax + By + C = 0$, while

$$\mathcal{T}_{ra} = \begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix}$$

is a translation of vector (T_x, T_y). A dilation of coefficients (α_x, α_y) whose center is (C_x, C_y) is characterized by

$$\mathcal{T}_{ra} = \begin{bmatrix} \alpha_x & 0 & C_x(1 - \alpha_x) \\ 0 & \alpha_y & C_y(1 - \alpha_y) \\ 0 & 0 & 1 \end{bmatrix}$$

and a rotation of angle α about a point $P = (P_x, P_y)$ is defined by

$$\mathcal{T}_{ra} = \begin{bmatrix} \cos\alpha & -\sin\alpha & P_x \\ \sin\alpha & \cos\alpha & P_y \\ 0 & 0 & 1 \end{bmatrix}.$$

In three dimensions, we have respectively (with similar notations)

$$\mathcal{T}_{ra} = \begin{bmatrix} 1 + A^2F & ABF & ACF & ADF \\ ABF & 1 + B^2F & BCF & BDF \\ ACF & BCF & 1 + C^2F & CDF \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ with } F = \frac{-2}{A^2 + B^2 + C^2}$$

for a symmetry.

$$\mathcal{T}_{ra} = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

for a translation and

$$\mathcal{T}_{ra} = \begin{bmatrix} \alpha_x & 0 & 0 & C_x(1 - \alpha_x) \\ 0 & \alpha_y & 0 & C_y(1 - \alpha_y) \\ 0 & 0 & \alpha_z & C_z(1 - \alpha_z) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

for a dilation. For a rotation about the x-axis, $\mathcal{T}_{ra}$ reads:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ or } \begin{bmatrix} 1 & 0 & 0 & P_x \\ 0 & \cos\alpha & -\sin\alpha & P_y \\ 0 & \sin\alpha & \cos\alpha & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -P_x \\ 0 & 1 & 0 & -P_y \\ 0 & 0 & 1 & -P_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

depending on whether the rotation is made around the origin or an arbitrary point P. For the rotations about the other axis, we find similar expressions, i.e., (with P at origin):

$$\mathcal{T}_{ra} = \begin{bmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ and } \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

about the y and the z-axis respectively. Matrix $\mathcal{T}_{ra}$ defined by:

$$\begin{bmatrix} a^2 - Sd + CS_2g & ab - Se + CS_2h & ac - Sf + CS_2i & 0 \\ SC_2a + Cd + SS_2g & SC_2b + Ce + SS_2h & SC_2c + Cf + SS_2i & 0 \\ -S_2a + C_2g & -S_2b + C_2h & -S_2c + C_2i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

corresponds to a rotation of angle α, of axis $A = (A_x, A_y, A_z)$ around point $P = (0,0,0)$. In this matrix (cf. Figure 17.1), we have:

- if $A_x \neq 0$, $\phi = \arctan\left(\frac{A_y}{A_x}\right)$, $\theta = \arctan\left(\frac{A_z}{\sqrt{A_x^2 + A_y^2}}\right)$,
  $C = \cos\phi$ and $S = \sin\phi$, $C_2 = \cos\theta$ and $S_2 = \sin\theta$,
- otherwise, if $A_y \neq 0$, $\theta = \arctan\left(\frac{A_z}{\sqrt{A_x^2 + A_y^2}}\right)$,
  $C = 0$, $S = 1$ and $C_2 = \cos\theta$, $S_2 = \sin\theta$,
- otherwise $C = 0$, $S = 1$ and $C_2 = 0$, $S_2 = 1$,

where

$$a = C_2C\,, \qquad b = C_2S\,, \qquad c = -S_2 \quad \text{and} \quad d = -\cos\alpha S - \sin\alpha S_2 C\,,$$

$$e = \cos\alpha C - \sin\alpha S S_2 \quad \text{and} \quad f = -\sin\alpha C_2 \qquad \text{and finally}$$

$$g = -\sin\alpha S + \cos\alpha S_2 C\,, \qquad h = \sin\alpha C + \cos\alpha S S_2 \quad \text{and} \quad i = \cos\alpha C_2\,.$$

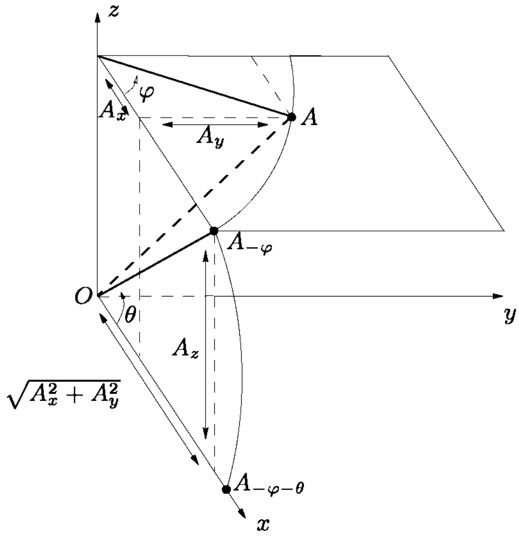


Figure 17.1: *Rotation about axis $A = (A_x, A_y, A_z)$. The angles ϕ and θ enable us to return to a rotation in terms of α about the x-axis.*

As an exercise, we now check that the above matrix is nothing other than the combination of the matrices

$$\mathcal{T}_{ra}(\phi, \vec{Z}) \circ \mathcal{T}_{ra}(\theta, \vec{Y}) \circ \mathcal{T}_{ra}(\alpha, \vec{X}) \circ \mathcal{T}_{ra}(-\theta, \vec{Y}) \circ \mathcal{T}_{ra}(-\phi, \vec{Z})$$

where $\mathcal{T}_{ra}(angle, vecteur)$ stands for the matrix related to the rotation of angle *angle* and axis *vecteur*.

We consider the case where $A_x \neq 0$. Then, writing only the sub-matrices corresponding to the rotation, the above expression leads to computing the product of the five following matrices:

$$\begin{bmatrix} C & -S & 0 \\ S & C & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} C_2 & 0 & S_2 \\ 0 & 1 & 0 \\ -S_2 & 0 & C_2 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & c\,\alpha & -s\,\alpha \\ 0 & s\,\alpha & c\,\alpha \end{bmatrix} \begin{bmatrix} C_2 & 0 & -S_2 \\ 0 & 1 & 0 \\ S_2 & 0 & C_2 \end{bmatrix} \begin{bmatrix} C & S & 0 \\ -S & C & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

using the above notations (where $c\,\alpha$ is short for $cos\,\alpha$ and $s\,\alpha$ is short for $sin\,\alpha$). It is then easy to write this operation as follows:

$$T_1\,T_2\,T_3\,T_4\,T_5\,.$$

Now, we compute the product $T_1\,T_2$, thus:

$$\begin{bmatrix} CC_2 & -S & CS_2 \\ SC_2 & C & SS_2 \\ -S_2 & 0 & C_2 \end{bmatrix}.$$

We note the product $T_3\,T_4\,T_5$ by:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}.$$

We then return to the above matrix T_{ra} after completing the product of these two matrices. To end, we express $T_3\,T_4\,T_5$ as a function of T_3 and the product $T_4\,T_5$, thus leading to the coefficients a, b, ..., i. □

Remark 17.1 *The center of rotation is supposed to be at the origin; to return to a general situation, this corresponds to adding the translation which takes into account the adequate point $P = (P_x, P_y, P_z)$ This comes down to applying first a translation of vector ${}^t(-P_x, -P_y, -P_z)$ and, after applying this to the result of above matrix T_{ra}, to applying the opposite translation.*

Remark 17.2 *In the case of a rotation, the definition of angles ϕ and θ by an arctangent gives a determination at modulo π. When programming such an operator, this ambiguity must be removed using the sine and cosine of the angles in such a way that they are well determined. Thus, we compute the coefficients C, S, C_2 and S_2 directly according to A_x, A_y and A_z to avoid this indetermination.*

Local and global refinement

We briefly discuss methods resulting in a global or a local refinement in a mesh.

Global refinement method

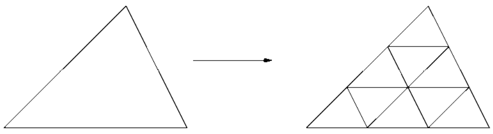


Figure 17.2: *Global partitioning of a triangle for $n = 3$.*

Provided with a global subdivision parameter, n, a global refinement method involves a "uniform" partition of all elements in the mesh. Each element is then split into n^d (d being the spatial dimension) elements with the same geometric nature. It could be noted that splitting triangles, quads, as well as hexahedra or pentahedra results in *similar* elements. On the other hand, the same method results in *non-similar* sub-meshes in the case of tetrahedral elements.

A global refinement method can be used both for mesh construction and for analyzing the quality of a solution. In the first case, a coarse mesh is constructed which is then subdivided in order to achieve a fine enough mesh. In the second case, repeated global refinements is one way to check some convergence issues about the solutions computed at the different levels. Note that such a method rapidly leads to large meshes (in terms of the number of elements). In this respect, the different refined meshes can be used to compare a mesh adaptation method with variable and local stepsizes with a "reference" solution computed on a uniform fine mesh.

Remark 17.3 *A particular process must be carried out when the element under partition has a boundary edge (face). This must be subdivided by taking into account the geometry of the boundary.*

Local refinement method. Unlike the previous global method, a local refinement method allows the creation of meshes with variable densities (in terms of element or vertex spacing). Thus, some extent of flexibility is obtained. Such a method can serve as a way to achieve some adaptive features and can therefore be used to deal with adaptive processes (as will be seen in Chapter 21 for example).

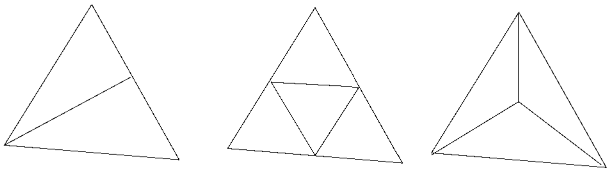


Figure 17.3: *Some local possibilities for splitting a triangle.*

In practice, a list of vertices is given, around which a refinement is expected. The way in which this list is defined depends on the criteria used in the computational process. For instance, edges in the current mesh which are judged to be too long can be defined and their endpoints can be put into the above list of vertices. Then, the elements are split according to the local situation. Figure 17.3 shows an example where a point is created along one edge (left-hand side), three points are defined as edge midpoints (middle) and a point is inserted inside the initial triangle (right-hand side).

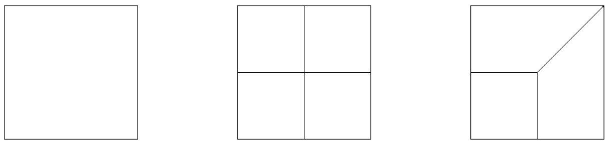


Figure 17.4: *Several local partitionings of a quad.*

Figure 17.4 considers the case of a quad. The initial quad (left-hand side) is split into four sub-quads by introducing the edge midpoints (middle) or into three sub-elements by introducing two points along two consecutive edges (right-hand side). Figure 17.5 illustrates a tet example. One point is created along one edge (left-hand side), one point is created in a face (middle) while a point is created inside the initial tet (right-hand side).

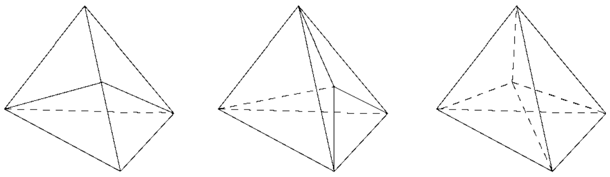


Figure 17.5: *Several local partitionings of a tet.*

Remark 17.4 *The important point is to ensure a good quality of the resulting elements, which means seeing whether the shape of the initial elements is preserved or altered by the local process.*

Remark 17.5 *Another topic to be addressed is that of maintaining a conforming mesh. To this end, elements in some neighborhood of the elements which are refined must be considered as candidates for some extent of refinement.*

Numerous theoretical works can be found on how angles, shapes, etc., are preserved or not when applying local splitting (see, for instance, [Rivara-1984b], [Rivara-1990], [Rivara-1997]).

The reader is also referred to Chapter 21 for more details about local mesh modification tools and to Chapter 8 where some other possibilities for local splitting are given.

Type conversion

Geometric type change of the elements in a mesh proves to be useful in various contexts. A quad element can be split by means of triangles using three different patterns (Figure 17.6). A triangle may be split into three quads (same figure) but the resulting quads may be poorly-shaped (see Chapter 18 regarding quality measures for both triangles and quads).

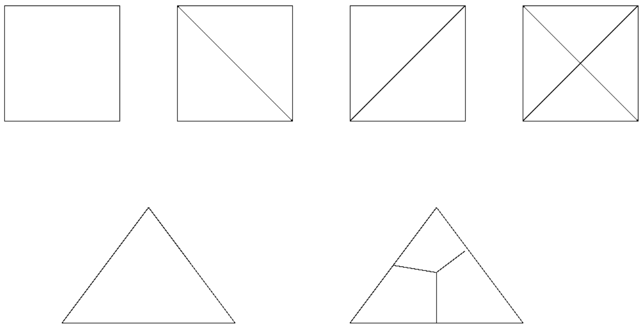


Figure 17.6: *A triangle leading to three quads (bottom) and a quad leading to two or four triangles (top).*

Similar change type operators can be formally defined for three-dimensional elements. The above transformation of a triangle into quads extends to three dimensions which allows a tet to be split by means of hexahedra. Nevertheless, as will be seen in Chapter 18, the degree of the vertices in the resulting mesh is very poor. Pentahedra can be split using tets (actually three tets are required). Hexahedra can be split into tets using five or six tets (Figures 18.3 and 18.4).

Remark 17.6 *There are* 74 *possibilities for remeshing a given hex into tets, with two solutions for a five-element splitting and* 72 *for a six-element pattern.*

Exercise 17.1 *Find this number of splitting (hint: find all the types of tet that can be constructed based on the hex vertices (58 cases are encountered), then, examine all the possible valid pairs. Suppress the pairs of tets separated by a face plane, look at the case where a face plane is common to two tets and show the 72 cases where the face is really a common face).*

17.2 Merging two meshes

Various other mesh modification or mesh post-processing are also of interest. Among these is a process for merging two given meshes into a single one when these two initial meshes share a boundary region.

Merging two meshes into a single one is frequently necessary. Several points must be taken into account including how to find the common entities (vertices, edges and faces), in spite of the round-off errors, and how to complete this task quickly. Another question is to find a new numbering (labeling) for the vertices (resp. elements) in the resulting mesh.

Before going further, let us give a more precise formulation of the merging problem. Let Ω_1 and Ω_2 be two domains in two or three dimensions and let $\mathcal{T}_1$ and $\mathcal{T}_2$ be the two corresponding meshes. We assume that these two domains share a common boundary region. We also assume that these meshes have been produced by any of the suitable mesh generation methods in an independent manner. Nevertheless (at least irrespective of the round-off errors) the meshes of the interface regions are assumed to be strictly identical. This mesh is in $d-1$ dimensions and thus consists of segments ($d=2$) or of faces ($d=3$).

If $\mathcal{T}_1 \cap \mathcal{T}_2$ stands for the mesh of the above interface, the merging problem consists of combining the vertices and the elements in $\mathcal{T}_1$ and in $\mathcal{T}_2$ in such a way as to obtain a single mesh $\mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2$. This question can be subdivided into three sub-problems:

- a problem of a metric (geometric) nature which consists of identifying the entities (vertices, edges and faces) common to the two meshes,
- a problem of vertex numbering that leads to finding an index (a label) for all the vertices in the mesh resulting from the merging process. This involves assigning, in a consistent manner, a numbering system to the three types of vertices that are met in the resulting mesh, say the former vertices in $\mathcal{T}_1$ not common with $\mathcal{T}_2$, those in $\mathcal{T}_2$ not common in $\mathcal{T}_1$ and, finally, those in $\mathcal{T}_1 \cap \mathcal{T}_2$, the interface,
- and a problem of element numbering that consists of finding a global numbering system in the resulting mesh.

As assumed, the points in the interface are supposed identical (i.e., this interface is meshed in a similar way once the two initial meshes have been constructed). Nevertheless, in practice, and due to the potential round-off problems, these points which are assumed to be identical are in fact identical for a given tolerance threshold. Thus, a searching technique with an accuracy of ε must be used. The common entities being identified, it remains to find the numbering of both the vertices and the elements. Let $num(P)_{\mathcal{T}_i}$ be the index (the label or the number) of vertex P in $\mathcal{T}_i$ for $i = 1, 2$. The problem becomes one of finding these values for the entire set of vertices whereas the element labeling problem is obvious.

All these points will be discussed below, but first we discuss how to find the common entities and then we turn to the numbering issues.

Finding the common vertices (geometric point of view). Here, given a vertex in $\mathcal{T}_1$ also included in $\mathcal{T}_1 \cap \mathcal{T}_2$, we discuss how to find the corresponding vertex in $\mathcal{T}_2$ (and conversely). Using a hashing technique (Chapter 2) has proved very efficient for this task. This operation consists of several stages:

- Constructing a hashing table based on mesh $\mathcal{T}_1$, the first mesh.

The two initial meshes are enclosed[1] in a quadrilateral "box" (hexahedral box in three dimensions). The box dimensions Δ_x, Δ_y and, in three dimensions, Δ_z are computed as well as $\Delta = \max(\Delta_x, \Delta_y, \Delta_z)$, see Figure 17.7. Given ε, a threshold tolerance, we define a sampling stepzize δ as:

$$\delta = 2\,\varepsilon\,\Delta\,,$$

which is the *resolution power* of the method.

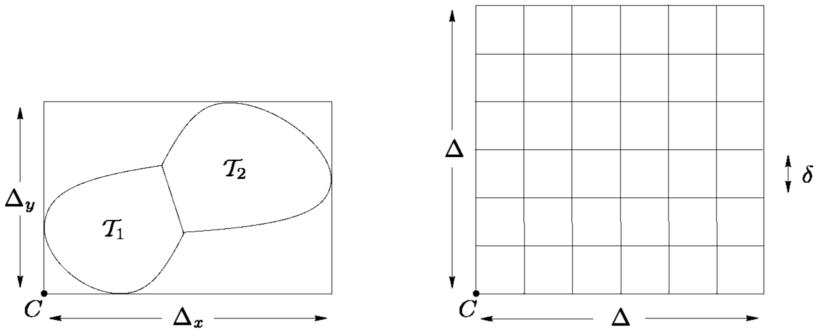


Figure 17.7: *The box enclosing the two initial meshes and the virtual grid with the parameter values associated with the merging algorithm.*

A grid with boxes (cells) of size δ is defined[2] which formally discretizes the space. The vertices in $\mathcal{T}_1$, the first mesh, are classified with respect to the grid boxes. To this end, if x, y and z are the coordinates of a given point P, we associate with this triple a triple of integers i, j and k defined as $i = \dfrac{x - x_{min}}{\delta}$, $j = \dfrac{y - y_{min}}{\delta}$

[1]Note that it is also relevant to define the enclosing box as the box based on only one mesh (for instance, the smaller mesh, in terms of the number of vertices), this box being slightly dilated.

[2]In practice, this grid is not actually constructed.

and $k = \dfrac{z - z_{min}}{\delta}$, where x_{min}, x_{min} and z_{min} are the coordinates of the bottom left corner of the introduced box.

Then, a hashing function is used which associates a hashing value with the triple i, j, k (and thus with point P). For example, $Hfunc(i, j, k) = i + j + k \;\; modulo(mod)$, where mod is a given integer, is a possible hashing function (see also Chapter 2).

This key serves as an entry point in a linked list containing the points of mesh $\mathcal{T}_1$. For example, if $Hfunc$ and $Link$ are the two above "tables", for a given point P with coordinates x, y, z, we compute $key_{hach} = Hfunc(P) = Hfunc(i, j, k)$, then, if $Link(key_{hach}) \neq 0$, $Q_1 = Link(key_{hach})$ is the index of the first point related to P while the other points (Q_l, if any) related to P are obtained via the sequence:

```
$l \leftarrow 1$, $key_{hach} \leftarrow Q_1$
REPEAT $key_{hach} = Link(key_{hach})$
  IF $key_{hach} \neq 0$,
    $l \leftarrow l + 1$,
    $Q_l \leftarrow key_{hach}$
  OTHERWISE, END.
```

Then, this material[3] is used to find the vertices in the second mesh which are common to the vertices in the first mesh.

Remark 17.7 *In some geometric situations, the virtual enclosing grid can be defined as the intersection of the two enclosing boxes associated with the two initial meshes in which the interface necessarily falls. Depending on the relative positions of the initial meshes, the virtual grid is significantly reduced and thus the corresponding parameters are better adapted.*

Now, we can proceed to the next stage of the process.

- Analysis of mesh $\mathcal{T}_2$ with regard to the above hashing table.

Once all the points in $\mathcal{T}_1$ have been processed, we consider the vertices in $\mathcal{T}_2$. Given point P in $\mathcal{T}_2$, we compute[4] $key_{hach} = Hfunc(P)$, the value associated with P, i.e., the equivalent of the box in which P falls. If x, y are the coordinates of P (for the sake of simplicity, in what follows, we assume a two-dimensional problem), we also compute the keys associated with the virtual eight points whose coordinates are $(x, y \pm shift)$, $(x \pm shift, y)$, $(x - shift, y \pm shift)$ and $(x + shift, y \pm shift)$ where $shift = \frac{\delta}{2} = \varepsilon\,\Delta$ (in fact, a value smaller than this theoretical value must be used, for instance, instead of $shift = 0.5\,\delta$, it is advisable to take $shift = 0.499\,\delta$).

[3]Another use of this hashing is to find the points in $\mathcal{T}_1$ which are "close" to a given point.

[4]If the grid is based on the first mesh only, it is necessary to check that P falls within the grid before computing its key, otherwise, point P is not considered.

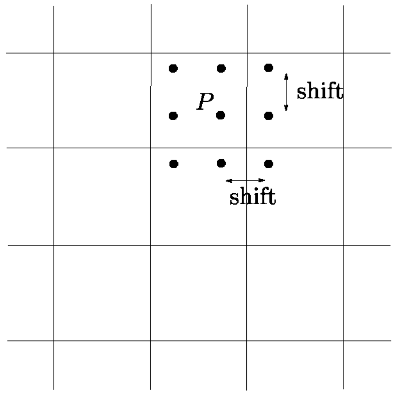


Figure 17.8: *The eight virtual points result in four different hashing keys.*

Due to the value of $shift$, only four different keys can be found (Figure 17.8). Then, let key_{hach} be one of these entry points, we just have to visit table $Hfunc$ in parallel with table *Link* to find the vertices of $\mathcal{M}_1$ which are "close" to point P under examination.

Let Q be such a point, then if $\max(|x_P - x_Q|, |y_P - y_Q|) \leq \delta$, the pair (P, Q) is a candidate pair of common points, otherwise, table *Link* is visited to find any other possible candidates.

Remark 17.8 *In this algorithm, we can stop once the first candidate pair has been obtained or we can continue in order to check if there is more than one point in $\mathcal{T}_2$ which forms a candidate pair. In such a case, a natural idea is to pick the closest point as the solution.*

Remark 17.9 *Following the previous remark, due to round-off errors or due to the more or less strict coincidence of the "common" boundaries, it is possible to find several points in $\mathcal{T}_2$ which are identified with only one point in $\mathcal{T}_1$; see Figure 17.9. To prevent such a problem, we consider the pairs of "identical" points resulting from the merging of $\mathcal{T}_1$ with $\mathcal{T}_2$ along with those resulting from the merging of $\mathcal{T}_2$ with $\mathcal{T}_1$. Then, the pairs which are in both lists are solutions.*

Finding the common entities. Common geometric vertices are obtained using the previous algorithm. Nevertheless, merging two meshes requires finding not only the common vertices but the common edges and faces as well. In addition, the merge can be based on a more general definition of the common entities involving not only the geometric point of view.

- Pure geometric merging process

In this case, the common vertices are determined using the above method. Then, common edges are those whose endpoints are common while common faces are those whose vertices are common. Exhibiting both the common edges and faces

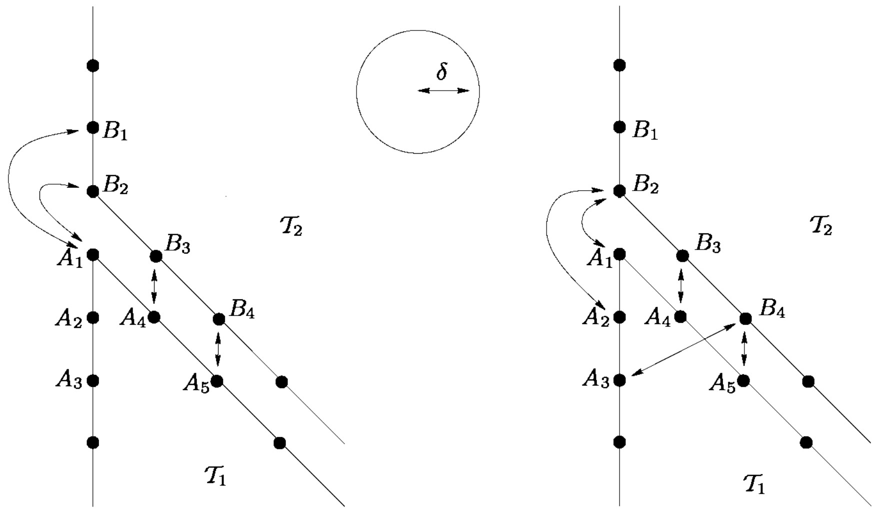


Figure 17.9: *Merging $\mathcal{T}_1$ and $\mathcal{T}_2$ results in the pairs $(B_1, A_1), (B_2, A_1), (B_3, A_4), (B_4, A_5), \ldots$ (left-hand side) while merging $\mathcal{T}_2$ and $\mathcal{T}_1$ produces the pairs $(A_1, B_2), (A_2, B_2), (A_3, B_4), (A_4, B_3), (A_5, B_4), \ldots$ (right-hand side). Then the points which are judged coincident are A_1 and B_2, A_4 and B_3, A_5 and B_5, etc. Note that, for clarity, the two meshes have been slightly shrunk, in fact the common boundaries are assumed to be coincident or, at least, very close.*

and not only the common vertices allows for an easy check of the Euler formula (Chapter 1), thus providing a way to guarantee the correctness of the resulting mesh.

- More sophisticated merging process

In this case, we consider both the geometric aspect and the physics of the problem. Including the physics in a mesh (see Chapter 1) involves associating some physical attributes (integer values, for instance) with the mesh entities (vertices, edges and faces). Then two vertices are said to be common if they are geometrically common and, in addition, if their two physical attributes are identical. Similarly, common edges and faces are defined by taking into account the physics. For instance, introducing physical considerations is a simple way to define cracks (in fact, two edges located at the same location exist at a given time and, due to the nature of the problem under consideration, are then clearly different: a crack appears.)

Numbering issues. This point may concern two aspects, element numbering and vertex numbering. Actually, depending on how the mesh data structure is defined, the mesh elements have an explicit numbering (or index) or are simply sequentially enumerated (meaning that the first element is that located at the beginning of the "table" which stores them). Whatever the case, the elements in mesh $\mathcal{T}$ are formed by the elements in mesh $\mathcal{T}_1$ sequentially followed by those of mesh $\mathcal{T}_2$. On the other hand, vertex numbering must receive some explicit attention.

If n_1 (resp. n_2) is the number of vertices in $\mathcal{T}_1$ (resp. $\mathcal{T}_2$) and if $num(P)_{\mathcal{T}_1}$ is the index[5] of vertex P in mesh $\mathcal{T}_1$, then the aim is to define $num(P)_{\mathcal{T}}$ for any point P in $\mathcal{T}$. The points in $\mathcal{T}$ are classified into three categories: those which are points of the former mesh $\mathcal{T}_1$, those which are common to $\mathcal{T}_1$ and $\mathcal{T}_2$, the interface, and, finally, those which are points of the former mesh $\mathcal{T}_2$ not common to the interface.

Once the common entities have been established, the common points are known. In other words, we know a correspondence table like

$$P \in \mathcal{T}_2 \iff Q \in \mathcal{T}_1 \,,$$

for all the points in the interface. Then, the following algorithm can be used to find the vertex numbering:

```
$num(P)_{\mathcal{T}} = 0$ for all $P \in \mathcal{T}_2$
$n \leftarrow n_1$,
IF $P \in \mathcal{T}_1$
   $num(P)_{\mathcal{T}} \leftarrow num(P)_{\mathcal{T}_1}$
IF $P \in \mathcal{T}_2$
   IF $P \in \mathcal{T}_1 \cap \mathcal{T}_2$
      $num(P)_{\mathcal{T}} \leftarrow num(Q)_{\mathcal{T}_1}$ where $P \in \mathcal{T}_2 \iff Q \in \mathcal{T}_1$
   OTHERWISE IF $num(P)_{\mathcal{T}} = 0$
      $n \leftarrow n + 1$
      $num(P)_{\mathcal{T}} \leftarrow n$.
   END IF
END IF
```

In fact, initializing this table from $n_1 + 1$ to $n_1 + n_2$ ensures an adequate initialization (since $n_1 + n_2$ is larger than the number of vertices in the final mesh). Using the initialization stage of this algorithm allows us to number the vertices by visiting the elements in the global mesh. Indeed, $num(P)_{\mathcal{T}} = 0$ means that the index of vertex P must be defined while $num(P)_{\mathcal{T}} \neq 0$ means that the index of P has been already established (P is a member of the (former) first mesh or P is in the interface or, finally, P has already been labeled when processing an element prior to the current element).

Remark 17.10 *The vertex numbering is* a priori *not optimal (in any sense), especially, merging $\mathcal{T}_1$ and $\mathcal{T}_2$ results in a different numbering from merging $\mathcal{T}_2$ and $\mathcal{T}_1$. Thus, a renumbering algorithm can be necessary to optimize the final mesh.*

Remark 17.11 *The merging of two meshes, a very natural and frequent operation in mesh generation packages, can be seen in a very different way. In fact, based on the methodology used to define the mesh of the entire domain by partitioning this domain into regions where a mesh generator is used, the regions common to two (local) meshes are formed by a set of two lines or two surfaces that have been*

[5]Indeed, in general, $num(P)_{\mathcal{T}_1} = P$.

explicitly defined beforehand. Therefore, the interfaces are a priori known, i.e., not to be sought in one way or another. In fact, the storage of this information in a mesh data structure is not trivial, and consequently we must regenerate them by computation. This apparent weakness is compensated for, on the one hand, by the efficiency of the merging algorithm and, on the other hand, by the resulting modularity in the sense that merging formally becomes independent of the design of the local meshes.

To conclude, it should be noted that, in terms of CPU, the hashing technique used to find the coincident vertices has proved to be fairly inexpensive. Thus, the computational effort necessary to merge two meshes is mainly due to the I/O steps of the process.

17.3 Node creation and node labeling

Both the creation and the adequate numbering of finite element nodes is a crucial task when finite elements other than P^1 are needed (a section, at the end of the chapter, will come back to this question, which is also discussed in Chapter 20). In this section we discuss how to assign a number (an index) to the nodes of a given mesh. We briefly consider this numbering issue at the mesh generation step and we turn to the case of meshes whose nodes do not reduce to their vertices. Renumbering issues (for optimization purposes) will be discussed in the next section.

Vertex and element labeling

As discussed in many chapters, numerous mesh generation methods exist which complete a so-called P^1 mesh (see Definition (17.2), below). Apart from some specific methods which result in a specific vertex (element) numbering, there is no particular reason that the vertex (element) numbering obtained should be optimal. The creation of a mesh with nodes other than the mesh vertices leads to the same conclusion since the sole aim of the node numbering (discussed below) is to assign a number to the mesh nodes without any concern about the optimality of this numbering.

Node creation

To introduce the problem, we discuss how to define P^2 finite elements. As previously seen, mesh generation algorithms implicitly construct P^1 meshes. To make this notion precise, we first review Definition (1.15) in Chapter 1:

Definition 17.1 *A node is a point supporting one or several unknowns or degrees of freedom (dof).*

A formal definition of a P^1 type mesh is as follows.

Definition 17.2 *A P^1 mesh if a mesh whose geometry is P^1, meaning that the element edges are straight and, in three dimensions, the element faces are planar, and whose nodes are reduced to its vertices.*

Every mesh vertex is then considered as a node and the geometric elements[6] are actually the finite elements (once the basis functions are defined; see Chapter 20).

In this section, we assume that a mesh implicitly of a P^1-type is given and we wish to define a P^2-type mesh. We assume in addition that the given mesh is *compatible* in a sense that will made clear hereafter, with respect to a P^2 interpolation. The problem is then to define the "midpoint" nodes along the boundary edges (for a problem in two dimensions; see Figure 17.10), as the extra nodes required for the interior edges are trivial to obtain. Boundary nodes must be properly located on the boundary and thus we need to know which edge is a boundary edge and, for a boundary edge, we also need to have a suitable representation of the geometry, so as to enable the proper location of the new node.

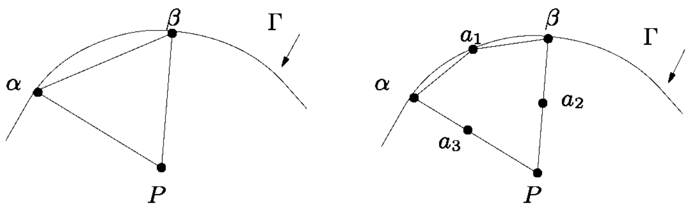


Figure 17.10: *Construction of a P^2 finite element. An element resulting from the mesh generation method previously used is shown on the left where Γ denotes the boundary domain. The corresponding P^2 element is depicted on the right-hand side. This six node triangle includes as nodes the three vertices P, α, β together with the above three edge "midpoints" a_i. Actually, for a straight six node triangle, the "midpoints" are the edge midpoints while for a curved (isoparametric) six node triangle, these a_i's must be properly located on Γ (for a boundary edge).*

Hence, two problems must be addressed. First, how to find a proper location for the nodes (other than element vertices). Second, how to find a global numbering of the nodes.

Node construction and p–compatibility

The most used finite elements (see below) involve either edge midpoints as nodes, several points along every element edge and/or some points inside the element. In three dimensions, nodes may be defined along element edges, element faces or inside the elements. Note also that element vertices may or may not be nodes.

Definition 17.3 *A given mesh, assumed to be P^1 is termed p-compatible if finite elements of degree p can be constructed resulting in a valid mesh.*

[6]I.e., the elements constructed by any mesh generation method.

Actually a mesh is p-compatible if all of its elements are p-compatible. The element depicted in Figure 17.10 is 2-compatible.

At this stage, the p-compatibility is assumed and remains purely intuitive. Let us simply consider that edges close enough to a curved boundary Γ result in a p-compatible mesh. This will be discussed with more detail in Chapters 20 and 22 where conditions ensuring this property will be developed. In fact, we have introduced this notion to make the node numbering problem consistent (we assume that creating these nodes poses no problem).

Node labeling

When the nodes are created, it is necessary to find a suitable numbering system for them both at the element level (*local numbering*) and at the mesh level (*global numbering*). At the element level, it is necessary to find a number (an index) for a given node. At the mesh level, it is necessary to associate a global index with the above local indices. Furthermore, it could be of great interest to optimize the resulting node numbering (see below).

Nodes are created along the mesh edges, in the mesh faces or inside the mesh elements based on the type of the finite element which must be defined. In the case where a constant number of nodes is defined for a given entity, the index of a node can be easily related to the index of the underlying entity. On the other hand, when the number of nodes varies for a given entity, finding a node index requires more subtle attention.

- Local numbering.

The definition of a finite element includes the definition of its nodes (see Σ_K hereafter). In other words the local numbering of the nodes is explicitly known. For example, the P^2 triangle consists of six nodes. The first three nodes are the three vertices while the mid-edge nodes form the nodes $4, 5$ and 6 of the element. Usually, the first nodes are the vertices and they follow their local numbering while the others, if any, are sequentially numbered based on their supporting entities (the edges, then the faces and, finally, (the interior of) the element).

- Global numbering.

Given a finite element, i.e., its node list, the aim is to find a global node numbering over the entire mesh. In practice, at the computational step, the elements are considered one at a time and the correspondence (let us consider again the six node triangle)

$$[1,2,3,4,5,6] \Longrightarrow [n_1,n_2,n_3,...,n_6]\,,$$

is used to perform the necessary calculation. Indeed, the labels n_1, n_2 and n_3 are known (as a result of the mesh generation step which completes a P^1 mesh) and the numbering problem reduces to finding the three remaining labels. To this end, we can construct the list of the mesh edges, assign a number to any edge while maintaining the relationship between this edge number and the element numbers. Then, fix $num = np$, where np is the number of mesh vertices and we use the following algorithm:

```
FOR $i = 1, ne$, $ne$ being the number of elements
   Visit the edges of element $i$ in the mesh.
   If the examined edge has not been previously visited
      (while visiting an element with index $i$ less than the current index),
      Assign the index of the known node to the edge node
      Proceed to the next edge.
   OTHERWISE
      Set $num = num + 1$
      Associate $num$ with the edge and
      Take $num$ as the index of the node in the examined edge.
   END IF
END FOR
```

For a finite element with more than one node on a given edge, the previous scheme must be slightly modified. Let k_n be the (constant) number of nodes per edge, then the following algorithm must be used:

```
FOR $i = 1, ne$
Visit the edges in the mesh.
If the examined edge has not been previously visited
   Use the index of the edge to find the various indices
   for the nodes related to this unique index
   Assign these indices to the nodes of the edge when considering
   it as indicated above based on the indices of its endpoints.
   Proceed to the next edge.
OTHERWISE, IF $j$ is the edge index,
   then the values $num_1 = np + k_n(j-1) + 1$, ...,
   $num_{k_n} = np + k_n(j-1) + k_n = np + k_n j$ are the node indices
   for this edge.
   Consider the endpoint indices, $i_1$ and $i_2$ and
   IF $i_1 < i_2$
      Assign $num_1$, $num_2$, ..., $num_{k_n}$ to the edge nodes
      while seeing it from $i_1$ to $i_2$
   IF $i_1 > i_2$
      Assign $num_1$, $num_2$, ..., $num_{k_n}$ to the edge nodes
      now seen from $i_2$ to $i_1$,
END FOR
```

For a finite element having some nodes on their faces (other than the nodes located on the face edges), the previous algorithm allows the face node numbering. An index is assigned to the face and one has just to take care how the corresponding node labels are assigned so as to insure a consistent numbering for a face shared by two elements. Again, comparing the labels of the face vertices is a solution to complete a consistent node numbering. Nodes inside an element are numbered according to the same principle.

Remark 17.12 *In this discussion we have assumed a constant number of nodes for each entity. If this number varies, the global index is not obtained by a simple formula as above but by adding one to the previous index used (note that the relationship between the entity index and the (first) node index is also required).*

Some popular finite elements

Due to node definition and numbering problems, we recall here some of the finite elements that are, for the most part, extensively used in the finite element computational processes commonly included in available software packages.

A finite element is fully characterized by a triple (see Chapter 20):

$$[K, P_K, \Sigma_K],$$

where K is a mesh element, Σ_K is the set of degrees of freedom defined over K and P_K denotes the basis polynomials (the shape functions) on K.

In terms of geometry, we are mostly interested in what K should be. In terms of node definition, we have to focus on Σ_K. The other point, the basis polynomials, does not fall within the scoop of this book (see the ad-hoc literature).

The geometry. Let us consider a triangle. The geometry of the three node triangle (namely the Lagrange P^1 finite element) is fully determined by the three vertex elements. Similarly, the geometry of the straight six node triangle (Lagrange P^2 element) is characterized by the three element vertices while the geometry of the curved (isoparametric) six node triangle is defined using the three element vertices along with the three edge "midpoints". Thus, in the defining triple, K is the triangle resulting from the mesh generator or this triangle enriched by three more points.

The node definition. The nodes are the support of the degrees of freedom. A given node may support one or several degrees. There are various types of degrees. The most simple finite elements, namely *Lagrange type* elements, involve as degrees the value(s) of the unknown(s) which is (are) computed. For instance, depending on the physical problem, the pressure, the temperature, the displacements, the stresses, the velocity, the Mach number, etc., could be the unknown functions. Then, a temperature leads to one degree, its value, while a velocity is associated with two (three) degrees (one degree for one direction). More sophisticated finite elements, such as *Hermite type* elements, involve as degree the value of the unknown function together with some of its derivatives. Also *beam*, *plate* and *shell* elements use various derivatives as degrees of freedom.

Now, given an element, the nodes can be located at various places. "Poor" elements have only their vertices as nodes. Other elements, in addition, use some nodes located along their edges, on their faces or inside them. Note also that there exist elements whose vertices are not nodes. In addition, finite elements exist where several types of nodes are encountered for which the number of degrees of freedom varies. A few examples of finite elements can be found in Chapter 20.

17.4 Renumbering issues

As a consequence of vertex, node or element numbering problems, another issue of interest concerns the vertex (node) or element renumbering methods that allow the minimization, in some sense, of the resulting matrices (i.e., based on the created mesh) used at the solution step of a computational process.

Renumbering concerns the solution methods that will be used subsequently. The aim is both to minimize the memory resources needed to store the matrix associated with the current mesh and to allow an efficient solution step for some solution methods where a matrix system must be considered.

Vertex renumbering

Various methods exist for mesh vertex (element) renumbering purposes. In what follows, we discuss in some detail one of the possible methods, namely the so-called Gibbs method ([Gibbs *et al.* 1976a]), and a variation of it.

For the sake of simplicity, we consider P^1-type meshes and the case of arbitrary meshes is discussed afterwards. The basic scheme of this method can be summarized by the three following steps:

- the search for a good *starting point* to initialize the algorithm (following [A.George, Liu-1979]),
- the optimization of the numbering *descent*,
- the numbering itself based on the previous material using the Cuthill-MacKee algorithm ([Cuthill, McKee-1969]).

In order to describe the method, we need to introduce some notations and definitions.

The mesh under consideration is $\mathcal{T}$, P_i is a mesh vertex and *np* and *ne* are the number of vertices and the number of elements, K_j is a mesh element. In terms of graphs, the vertices are also referred to as the *nodes*. Two such nodes are said to be neighbors if there exists an element in $\mathcal{T}$ with these two nodes as vertices (or nodes, in terms of finite element nodes). The *degree* of a node P, $deg(P)$, is the number of its neighboring nodes. The *graph*, Gr, associated with $\mathcal{T}$ indicates the connections between the nodes (Figure 17.11). Note that a graph may contain one or more connected components, the connected component number k denoted as C_k. The neighbors of node P constitute $\mathcal{N}_1(P)$ level one of its *descent*. The neighbors of the nodes in $\mathcal{N}_1(P)$, not yet referenced, form $\mathcal{N}_2(P)$, level two of the descent of P and so on. The *descent* of node P, $\mathcal{D}(P)$ is the collection of the levels $\mathcal{N}_k(P)$. The *depth* of P, $d(P)$, is the number of levels in $\mathcal{D}(P)$.

First, we establish the neighboring relationships from element to element. At the same time, the degrees of the mesh nodes are found. Then, several steps are performed:

- Step 1: an initialization step corresponding to the following algorithm:

```
Pick in $\mathcal{T}$ the node $P$ of minimal degree $deg(P)$
(1) define $\mathcal{D}(P)$, the descent of $P$,
consider the last level in this set, i.e., $\mathcal{N}_k(P)$,
select the node $Q$ of minimal degree.
form $\mathcal{D}(Q)$ and compare $\mathcal{D}(P)$ with $\mathcal{D}(Q)$,
```

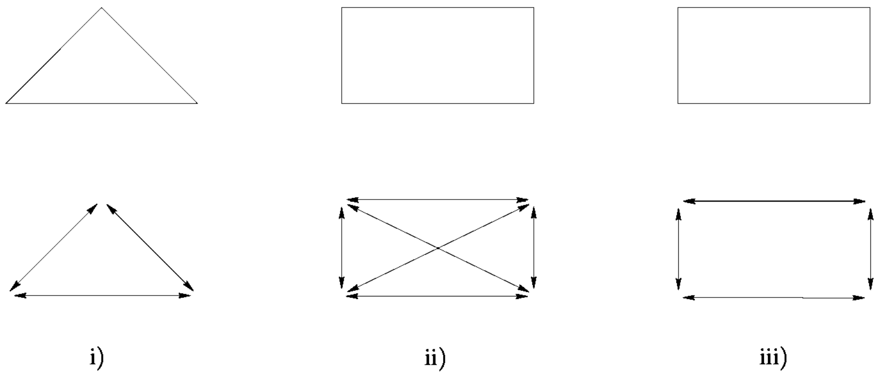


Figure 17.11: *Connections from nodes to nodes for a triangle i) and a quadrilateral ii). All the nodes of the element interact. For a quadrilateral iii), a reduced graph can be constructed by omitting the connections corresponding to the diagonals.*

```
IF $d(Q) > d(P)$
   replace $P$ by $Q$
   RETURN to (1)
OTHERWISE
   the initialization is completed.
END IF.
```

This step merits some comments. Actually, the initial guess P is the point with minimal degree. Indeed, P must be a boundary point but this way of processing avoids explicitly establishing the list of the boundary points while leading to the desired result after very few iterations.

On completion of the initialization step, we have found PQ the *pseudo-diameter* of the renumbering graph which now serves to pursue the method.

- Step 2:

Given the pseudo-diameter PQ together with $\mathcal{D}(P) = \{\mathcal{N}_1(P), \mathcal{N}_2(P), ..., \mathcal{N}_p(P)\}$ and $\mathcal{D}(Q) = \{\mathcal{N}_1(Q), \mathcal{N}_2(Q), ..., \mathcal{N}_p(Q)\}$ where p is the depth of both the descent of P and that of Q, we construct the mesh descent $\mathcal{D}$ and more precisely its level $\mathcal{N}_i$ which will enable us to find the desired renumbering of the nodes.

To this end, three stages are necessary. First, we construct the pairs (i,j) as follows:

```
for $M$ a given point in $\mathcal{T}$
find the level of $\mathcal{D}(P)$ and that of $\mathcal{D}(Q)$ where is $M$.
Let $\mathcal{N}_i(P)$ and $\mathcal{N}_k(Q)$ be these two levels,
form the pair $(i,j)$ such that $j = p + 1 - k$.
```

Using this information, we can start the construction of the global levels:

```
If the pair $(i,i)$ exists for point $M$
```

```
    put $M$ in the level $\mathcal{N}_i$
    suppress $M$ from the list of nodes to be examined, i.e., from graph $Gr$.
```

If Gr is empty, we go to Step 3 of the global algorithm, otherwise, the remaining nodes must be classified in the different levels.

The current graph Gr can now consist of a set of one or more disjoint connected components. These are found and ordered following the decreasing number of nodes they have. Let C_k be the component of index k and let *ncomp* be the number of connected components, then we analyze the *ncomp* components:

```
FOR $k = 1, ncomp$
   FOR $m = 1, p$
      compute $n_m$ the number of nodes in level $\mathcal{N}_m$,
      compute $h_m = n_m +$ (the number of nodes) $\in C_k \cap \mathcal{N}_m(P)$,
      compute $l_m = n_m +$ (the number of nodes) $\in C_k \cap \mathcal{N}_m(Q)$,
   END FOR
   compute $h_0 = \max_{m=1,p}(h_m)$ where $h_m - n_m > 0$,
   compute $l_0 = \max_{m=1,p}(l_m)$ where $l_m - n_m > 0$,
      IF $h_0 < l_0$
         put all the nodes in $C_k$ in level $\mathcal{N}_i$
         where $i$ is the first index in the pair $(i, j)$ associated with
         the node under examination,
      OTHERWISE IF $h_0 > l_0$
         put all the nodes in $C_k$ in level $\mathcal{N}_j$
         where $j$ is the second index in the pair $(i, j)$ associated
         with the node,
      OTHERWISE IF $h_0 = l_0$
         put all the nodes in $C_k$ in the level with the
         smallest dimension,
      END IF
END FOR
```

In this way, each vertex of Gr has been assigned a level in the descent. Moreover, a nice balancing is obtained in the different levels, then the numbering, following Cuthill-Mackee, can be processed. This is the purpose of the following step.

- Step 3:

First, if $deg(Q) < deg(P)$, we reverse P and Q and the levels obtained at the previous step to ensure that the numbering will start from the endpoint of the pseudo-diameter of lower degree. Reversing the levels leads to fixing $\mathcal{N}_i = \mathcal{N}_{p-i+1}$, for $i = 1, p$. Then we want to obtain $newnum(M)$, for $M \in \mathcal{M}$, the desired number of the vertex M. To this end:

```
Initialization, one starts from $P$, thus $M = P$
   $n = 1$
   $newnum(M) = n$
   $\mathcal{N}_0 = \{P\}$
FOR $k = 1$
```

```
    If $\exists M \in \mathcal{N}_k$ with adjacent vertices not yet renumbered
       construct the list $C(M) = adj(M) \cap \mathcal{N}_k$ where $adj(M)$ notes
       the vertices neighbor of $M$
    OTHERWISE IF $\exists S \in C(M)$ not yet renumbered (such points being
    considered following the increasing order of their degrees)
       $n = n + 1$
       $newnum(M) = n$.
    END IF
END FOR

FOR $k = 2, p - 1$
    WHILE $\exists M \in \mathcal{N}_k$ with adjacent vertices not yet renumbered
       construct the list $C(M) = adj(M) \cap \mathcal{N}_{k+1}$ where $adj(M)$ notes
       the vertices neighbor of $M$
    S OTHERWISE IF $\exists S \in C(M)$ not yet renumbered (such points being
    considered following the increasing order of their degrees)
          $n = n + 1$
          $newnum(M) = n$.
END FOR
```

If some cases (in fact, if P and Q were reversed and if j was selected in the previous step, i.e., $h_0 > l_0$ was found when component C_1 was analyzed or P and Q were not reversed but i was selected in the previous step) the previous renumbering is reversed:

```
FOR $i = 1, np$
    $i \leftarrow np - i + 1$
END FOR
```

This vertex renumbering method has proved very efficient, and is, moreover, fully automatic. It results in a significant minimization of both the *bandwidth* and the *profile* of the sparse matrix that will be associated with the mesh being processed.

Arbitrary meshes. Given an arbitrary mesh (in terms of the node definition), the previous material can be applied for node renumbering. Let us consider a simple example, a P^2 mesh where the nodes are the mesh vertices together with the edge midpoints. A possible node renumbering method could be as follows:

- apply the previous method to the P^1 mesh associated with the current mesh. This comes down to only considering the vertices of the P^2 mesh as nodes,

- use the resulting node numbering to complete the node numbering of the P^2 mesh. To this end, initialize $i = 1$ and $num = 1$, then

  - pick the node i in the P^1 mesh, find the *ball* of node i, assign numbers $num, num + 1, ...$ to the P^2 nodes of the elements in this ball that have not yet been renumbered, update num and repeat this process (with $i = i + 1$).

Remark 17.13 *This rather simple renumbering method is probably not optimal but appears to be reasonable on average. To obtain better results, the classical method could be performed by considering the full node graph of the mesh, which therefore increases the effort required.*

Remark 17.14 *The case of adapted meshes, in specific anisotropic meshes, is not really a suitable situation for the above numbering method.*

Element renumbering

A compact numbering of the elements may be a source of benefit at the solution step as this could minimize some cache default or some memory access default.

A simple idea to achieve this feature is to make the element numbering more compact around the mesh vertices. Thus, it could be of interest to renumber the mesh vertices first before processing the element renumbering step. A synthetic algorithm is then:

```
Initializations
   $num = 0$
   $newnum(1 : ne) = 0$
FOR $i = 1, np$
   FOR $j = 1, ne$
      IF $P_i$ is a vertex in $K_j$ and $newnum(j) = 0$
         $num = num + 1$
         $newnum(j) = num$
      END IF
   END FOR $j$
END FOR $i$
```

with the same notations as above (K_j denotes an element and P_i denotes a vertex). At a glance, the complexity of this algorithm is something like $\mathcal{O}(n^2)$ where n stands for ne (or np), thus, this algorithm is not really suitable in terms of computer implementation.

Application examples

To illustrate the efficiency of the above method (for vertex renumbering), we give some selected examples related to various meshes.

Table 17.1 presents various characteristics of the meshes before and after being renumbered. The first two meshes are in two-dimensional space, the two others correspond to three-dimensional cases. The values of interest are the *total profile*, the *mean profile*, the *bandwidth* and v the number of nodes dealt with per second as a function of the size of the expected matrix, assumed to be an n by n symmetric matrix (with $n = np$, the number of nodes of the mesh). The bandwidth, β, is defined by

$$\beta = \max_i |i - j_i| ,$$

where $i = 1, n$ and j_i is the index of the first row where an *a priori* non-zero coefficient may exist. Indeed the value $|i - j_i|$, denoted as δ_i, measures the distance between the first " non-zero" row and the diagonal of the resulting matrix (i.e., in terms of vertex indices, the distance, given vertex index i, between the vertex, adjacent to vertex i, which is as far as possible from vertex i). The profile and the mean profile are then

$$pr = \sum_{i=1}^{n} \delta_i \quad \text{and} \quad pr_{mean} = \frac{pr}{n} \quad \text{with} \quad n = np$$

.

Mesh	1	2	3	4
np	36,624	59,313	43,434	92,051
ne	72,783	116,839	237,489	463,201
β (initial)	35,396	53,580	42,516	89,530
pr	364,616,352	827,498,368	483,132,896	2,194,965,760
pr_{mean}	9,955	13,951	11,123	23,845
β (final)	811	458	1,368	5,186
pr	11,115,036	9,166,162	39,944,464	172,534,912
pr_{mean}	303	154	919	1,874
$\frac{pr}{pr_{mean}}$	33	90	12	13
v	29,000	27,000	11,000	10,600

Table 17.1: Gibbs: bandwidth and profile before and after renumbering. Examples one and two concern two-dimensional triangle meshes while the two other examples are three-dimensional tetrahedral meshes.

The Gibbs method proves to be robust and efficient. In addition, it is fully automatic and does not require any directive. Comments on these statistics can be seen below (to allow the comparison with other renumbering methods).

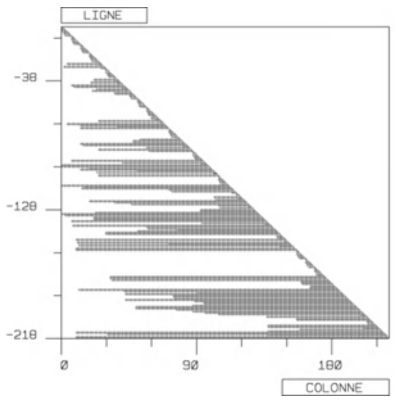
Remark 17.15 *Any conclusion on the efficiency of a Gibbs based renumbering method must be made more carefully in the case of adapted meshes where the density of elements varies greatly from place to place or also when anisotropic elements made up the mesh. In such cases, other renumbering methods may be advocated (see below).*

Other renumbering methods

Among the various renumbering methods, the so-called *frontal method*[7] is now briefly discussed (to make some comparisons with the Gibbs method possible and to take into account the above remark).

In essence, this method consists of radiating from a starting front. Let $\mathcal{F}_0$ be a set reduced to one node or a collection of adjacent nodes (generally chosen on the domain boundary), then the algorithm can formally be written as follows:

[7]Also referred to as *greedy*.



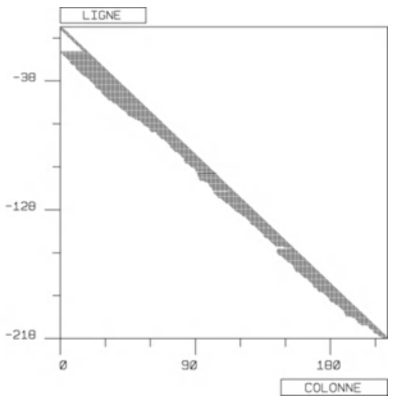


Figure 17.12: *Matrix occupation associated with a given mesh before (left-hand side) and after node renumbering (right-hand side). For the sake of clarity, this example concerns a "small" mesh different from those described in Table 17.1.*

```
$k \leftarrow 0$
(A) FOR $i = 1, card(\mathcal{F}_k)$ (the number of nodes in $\mathcal{F}_k$)
   FOR all the elements sharing node $i$ in $\mathcal{F}_k$
      construct $\mathcal{V}_i$, the set of indices of the nodes neighboring
      node $i$, not yet renumbered
      FOR all the nodes in $\mathcal{V}_i$ (let $j$ be the index of such a node)
         compute $nbneigh(j)$, the number of neighbors of node $j$
      END FOR
      renumber the nodes in $\mathcal{V}_i$ as a function of $nbneigh(j)$,
      the number of neighbors (in increasing or decreasing order
      of $nbneigh$)
   END FOR
```

Thus, set $\mathcal{V}_i$ forms the new renumbering front, then:

```
$k \leftarrow k + 1$
$\mathcal{F}_k = \mathcal{V}_i$
IF $card(\mathcal{F}_k) > 0$
   return to (A)
OTHERWISE, END.
   The process is completed (the numbering may be reversed).
```

Numerous variations exist. For instance, this method of node renumbering depends greatly on the choice of the initial front and could therefore require the user's directive. It also depends on the way in which the nodes of the $\mathcal{V}_i$'s are selected.

Finding the initial front can be the user's responsibility or can be obtained using the above technique for the construction of the *pseudo-diameter* of the mesh.

Choosing how to proceed the $\mathcal{V}_i$s can be done:

- by selecting the node in $\mathcal{V}_i$ with minimum degree,
- by selecting the node in $\mathcal{V}_i$ with maximum degree,
- by selecting the node in $\mathcal{V}_i$ with minimum index,
- and so on.

Tables 17.2 and 17.3, similar to the previous table, give the statistics related to two different variations of a frontal method. Table 17.2 considers a method whose initial front is the point with minimal degree (presumably a boundary point) while Table 17.3 shows a method where the pseudo-diameter is used to initialize the front.

Mesh	1	2	3	4
np	36,624	59,313	43,434	92,051
ne	72,783	116,839	237,489	463,201
β (initial)	35,396	53,580	42,516	89,530
pr	364,616,352	827,498,368	483,132,896	2,194,965,760
pr_{mean}	9,955	13,951	11,123	23,845
β (final)	552	303	1,390	4,527
pr	16,145,040	12,647,528	39,873,528	254,601,584
pr_{mean}	440	213	918	2,765
$\frac{pr}{pr_{mean}}$	23	65	12	9
v	55,000	50,000	15,000	15,000

Table 17.2: Naive frontal method: bandwidth and profile before and after renumbering (the examples are the same as in Table 17.1).

Mesh	1	2	3	4
np	36,624	59,313	43,434	92,051
ne	72,783	116,839	237,489	463,201
β (initial)	35,396	53,580	42,516	89,530
pr	364,616,352	827,498,368	483,132,896	2,194,965,760
pr_{mean}	9,955	13,951	11,123	23,845
β (final)	630	357	1,425	5,172
pr	12,296,281	10,045,739	41,060,840	235,254,128
pr_{mean}	335	169	945	2,555
$\frac{pr}{pr_{mean}}$	30	82	12	9
v	39,000	37,000	13,000	12,300

Table 17.3: Frontal method starting from the pseudo-diameter: bandwidth and profile before and after renumbering.

Comparing the three tables leads to some comments. In terms of bandwidth, in our (limited) experience the two frontal methods give as the worst value one smaller (i.e., better) than the Gibbs method. Nevertheless, the Gibbs method results in a better profile (and thus the ratio $\frac{pr}{pr_{mean}}$ is larger for any of the examples we have tried). In terms of CPU, the naive method is less demanding than the other frontal method and the Gibbs method requires more effort. As a final remark, it could be observed that the ratio of improvement due to a renumbering method depends on the geometry of the domain. In this respect, thin regions connected to large regions, loops (holes), etc., interact on the result.

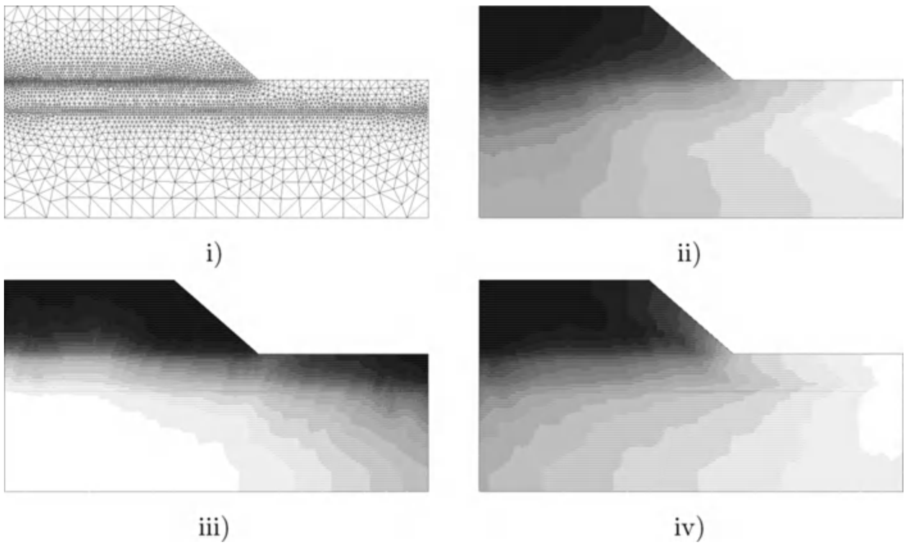


Figure 17.13: *Gibbs versus frontal methods. The original mesh i); Gibbs ii); naive frontal iii); and frontal with pseudo-diameter iv). Colors are associated with vertex indices (from white (vertex #1) to black (vertex #np). The color variation indicates the way the methods proceed.*

Element coloring. A method can be constructed to color the elements of a given mesh. The goal is to obtain several disjoint packets of elements, each of which corresponds to one color while two elements in two different neighboring packets have two different colors. Using ideas similar to the four colors theorem, this method consists of renumbering the elements of a mesh by creating packets of elements such that any neighbor of an element in a given packet is not located in the same packet. When only two packets are created, this method clearly looks like the "Red-Black" method [Melhem-1987] which allows separation of nodes into two disjoint sets. Put briefly, the algorithm is based on searching for the neighbors of an element with the aim of separating these elements. The main application of this method is the numerical solution to a problem on a vector computer, but the idea may be applied to various areas.

Other numbering (coloring) issues. Various methods can be developed based on particular numbering or coloring techniques.

17.5 Miscellaneous

In this section, we briefly mention some topics about mesh modification that have not already been covered.

Mesh manipulation and physical attributes. As mentioned earlier, a mesh devoted to a given numerical simulation does not include only a geometric aspect and must contain information about the physics of the problem. In this respect, the elements in the mesh are not fully described when the vertex coordinates and the vertex connection are known. Actually, physical attributes are also part of the mesh description. At the element level, it has proved to be useful to have some physical attributes associated with the element itself as well as with the element entities (vertex, edge, face). Therefore, any mesh modification includes two parts. The part concerned with the geometric aspect of the mesh manipulation has been already described while the way to manage the physical attributes of the given mesh and to derive the corresponding information for the resulting mesh must be now discussed.

In practice, the physical aspects of the resulting mesh inherit, in some way, the physical aspects included in the initial mesh. The issue is to make the desired correspondences precise, based on the type of mesh manipulation used. Figure 17.14 shows two basic examples and gives some idea of what must be done.

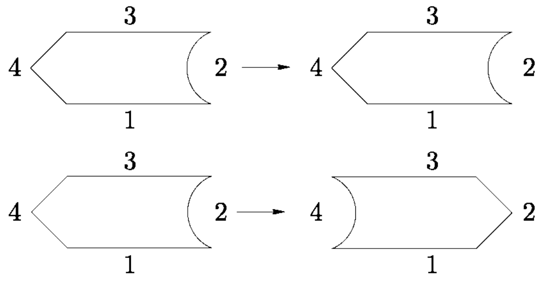


Figure 17.14: *According to the transformation (translation (top), symmetry (bottom)), attributes* 1, 2, 3 *and* 4 *remain unchanged or must be permuted as* 1, 4, 3 *and* 2.

Non-obtuse mesh. See Chapter 18.

Crack or widthless region construction. Defining one or several cracks or regions with little thickness makes it possible to deal with some types of problems in solid or fluid mechanics.

A crack (a thin region) is a line (a surface) which has *a priori* two meshes. In the case of a crack, these two meshes are coincident, in some sense. After processing, the crack may open and become longer.

Constructing a thin zone makes it possible, for example, to introduce finite elements like line elements between the two initial elements sharing the common edge (in two dimensions).

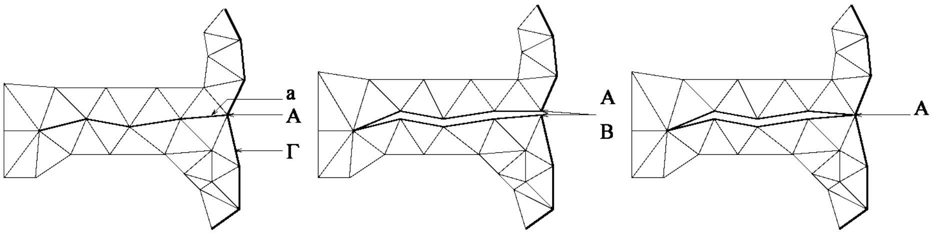


Figure 17.15: *Left-hand side: a part of the domain, a set of edges that must be duplicated (a), the boundary* Γ *and the initial endpoint of the crack (A). Middle: the crack appears and the initial point A is duplicated (in point A and another point B which is distinct). Right-hand side: initial point A is not duplicated.*

Creating a crack of a zone with no thickness is not an easy task for a mesh generation method. The fact that there is at least one pair of strictly coincident points at the beginning implies that the classical methods (*quadtree-octree*, advancing-front or Delaunay based) are not suitable, at least in their classical versions (Chapters 5, 6, 7). Such a situation is indeed seen as a cause of failure. Therefore, defining such a zone is not possible without using specific post-processing[8].

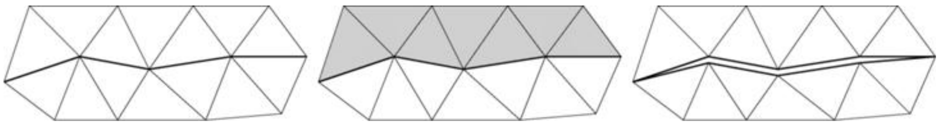


Figure 17.16: *Elements located on one of the side of the crack are identified, then the crack is defined by duplicating the edges that have been detected in this way.*

The first method consists of a suitable construction of two meshes which are then merged while avoiding the points and the edges previously identified (see above) to be merged. Thus, a merging operator is used to complete the desired result. A second method (Figure 17.16) consists of finding all the elements that share the edges to be duplicated, identifying the adjacencies and then suppressing these adjacencies by duplicating the entities concerned.

Periodic mesh. Some problems show periodic properties at the boundary condition level in some part of the domain boundary which, in turn, has the same periodic aspect (think about a translation or a rotation).

[8]Or, at least, some degree of tinkering inside the mesh generation method!

Some computational software then make it possible to reduce the analysis to only one portion of the domain provided the appropriate periodic conditions are defined in such a way as to relate some parts one with another. It is then possible to deduce the entire domain together with the solution in this entire domain whereas the computation has only been done in one portion of this domain (for example, an analysis of a single turbine blade allows us to obtain the result for the whole turbine).

In terms of how to construct a periodic mesh, it is sufficient to be sure that the two zones that must be in correspondence have been meshed in the same way. Thus consistency will be ensured when the pair of nodes to be connected are identified.

Remark 17.16 *If the region to be processed is composed of quads (in three dimensions) and if the elements (hexs or pentahedra) are split into tets, it is necessary to constrain the element split in order to obtain a compatible mesh on both sides (if mesh conformity is required).*

★
★ ★

In conclusion, mesh modification methods (with a view of mesh optimization) have not been discussed in this chapter. This point is discussed in Chapter 18 for planar and volumic meshes and Chapter 19 deals with surface meshes.

Chapter 18

Mesh Optimization

Optimizing a mesh with respect to a given criterion is an operation that is frequently used with various goals in mind for a wide range of applications. First, optimization in itself is useful because the quality (the convergence of the computational schemes and the accuracy of the results) of the numerical solutions computed at the mesh nodes clearly depends on the quality of the mesh. In this respect, mesh generation methods usually include an optimization stage that takes place at the end of the entire mesh generation process. An optimization process may serve some more specific purposes such as the mesh adaptation, for instance, included in an adaptive computational procedure. Moreover, the tools involved in optimization methods can be also used in some particular applications (mesh simplification being a significant example).

★
★ ★

The aim of this chapter is to introduce some methods designed for mesh optimization purposes. First, some information is given on how to compute element surface areas and volumes. Applications based on surface area and volume values are discussed, including localization and intersection problems, then we turn to the definition of mesh quality. Afterwards, we introduce some local tools for mesh optimization.

Having introduced these tools, and with regard to the given objectives, mesh optimization methods are discussed both in terms of strategies and computational aspects. Actually, mesh optimization can be considered as a step of a mesh generation method (in general, the last step of the method). It also can be seen as a stand-alone process.

This chapter only considers planar and volumic meshes. Moreover, only the geometrical aspect is considered meaning that P^1 meshes (and more generally meshes whose element nodes are identical to element vertices) are discussed. Surface meshes, which are slightly different, will be discussed in the next chapter.

18.1 About element measurement

In this section, we discuss how to compute the surface areas or volumes of the different types of mesh elements. We then give some indications about how such values can be used for various purposes. Note that the elements we are interested in are defined with an orientation (see Chapter 1).

Element surface area

The only element for which the surface area can be obtained directly is the triangle (due to its simplicial aspect). Thus, if K is a triangle whose vertex coordinates are denoted by x_i, y_i, $(i = 1, 3)$, then the surface area of K is:

$$S_K = \frac{1}{2} \begin{vmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{vmatrix}, \tag{18.1}$$

or, in an equivalent form:

$$S_K = \frac{1}{2} \begin{vmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{vmatrix}. \tag{18.2}$$

In the case of a quad, the surface area can be computed as the sum of the surface areas of the two triangles formed by considering one of its diagonals (either diagonal being suitable; see Figure 18.1).

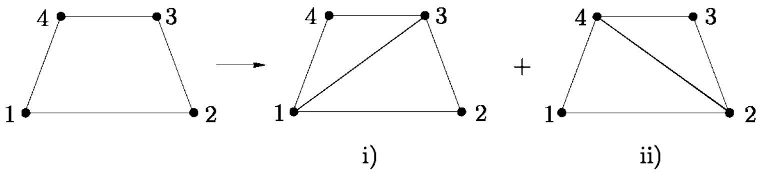


Figure 18.1: *Analysis of a quad by means of the four corresponding triangles. Two triangles allow for the surface area calculation while four triangles are necessary to check the convexity of the quad (see below).*

Element volume

In this case, only the volume of a tetrahedron is easy to compute. If K is a tet whose vertex coordinates are denoted by x_i, y_i, z_i, $(i = 1, 4)$, then the volume of K is:

$$V_K = \frac{1}{6} \begin{vmatrix} x_2 - x_1 & x_3 - x_1 & x_4 - x_1 \\ y_2 - y_1 & y_3 - y_1 & y_4 - y_1 \\ z_2 - z_1 & z_3 - z_1 & z_4 - z_1 \end{vmatrix}, \tag{18.3}$$

or, similarly:

$$V_K = \frac{1}{6}\begin{vmatrix} 1 & x_1 & y_1 & z_1 \\ 1 & x_2 & y_2 & z_2 \\ 1 & x_3 & y_3 & z_3 \\ 1 & x_4 & y_4 & z_4 \end{vmatrix} . \tag{18.4}$$

To compute the volume of a pentahedron, it must be subdivided into three tets (Figure 18.2).

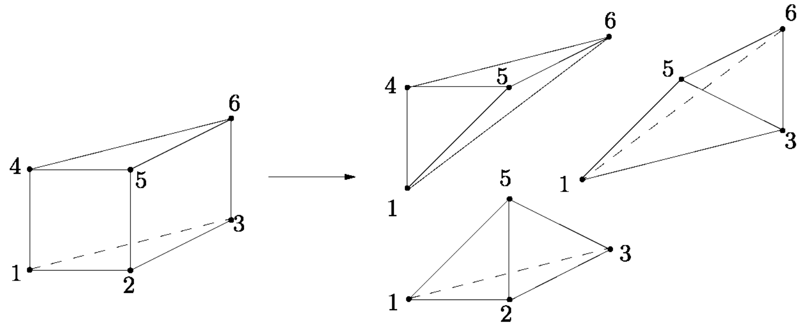


Figure 18.2: *Analysis of a pentahedron by means of three tets. Note that only one of the various possible partitions is given.*

For a hexahedral element, six or five tets are necessary. The partition based on six tets involves first splitting the hex into two pentahedral elements to which the above partition into three tets is applied (Figure 18.3).

A pattern with five tets is also possible (Figure 18.4) where the tet inside the volume is considerably different to the four others (unlike the pattern with 6 elements) for a regular initial hex.

Other measurements

Surface areas or volumes can provide other information which is useful in this context.

Convexity of a quad. A quad element in two dimensions can be analyzed to decide whether it is convex by using four triangle surface areas. Actually we define the four triangles that can be constructed using one or the other diagonal of the quad (Figure 18.1). We then compute the four surface areas, S_i, of these triangles. Hence, if the four S_is are positive, the quad is convex. Otherwise, if one of the S_is is negative, the quad is non-convex. If one of the S_is is zero, the quad is said to be degenerated (i.e., two consecutive sides are aligned). If two of the S_is are negative, each being one part of the two possible decompositions, then the quad is self-intersecting while if two negative S_is correspond to the same decomposition, the quad is negative (Figure 18.5).

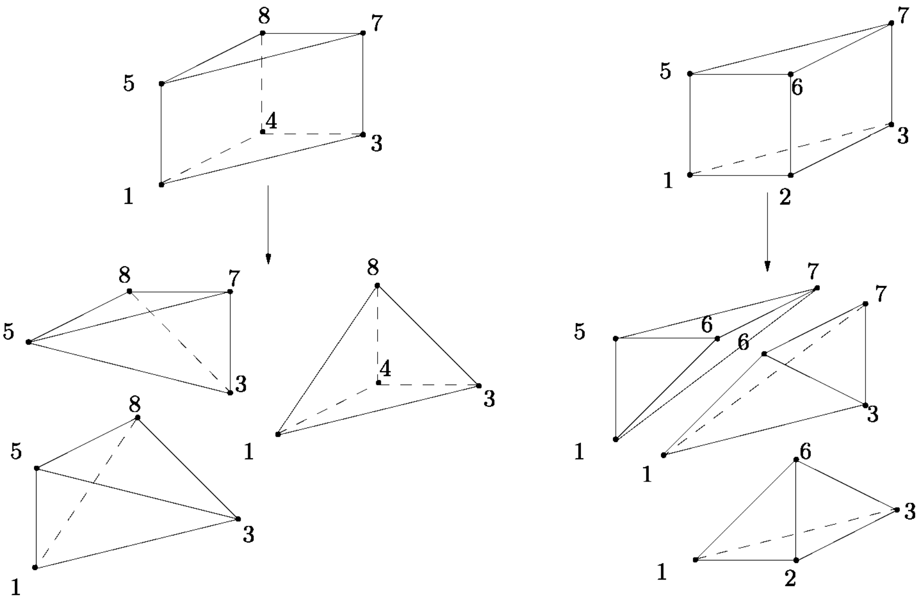


Figure 18.3: *Analysis of a hex by means of six tets. The initial hex* 12345678 *is primarily split into two pentahedra (*123567 *and* 134578*). In this example, the resulting partition is not a conforming mesh of the initial hex. Thus, such a partition is suitable for volume computation while a different one must be considered for a different purpose (for instance, if we want to convert a hex mesh into a tet mesh).*

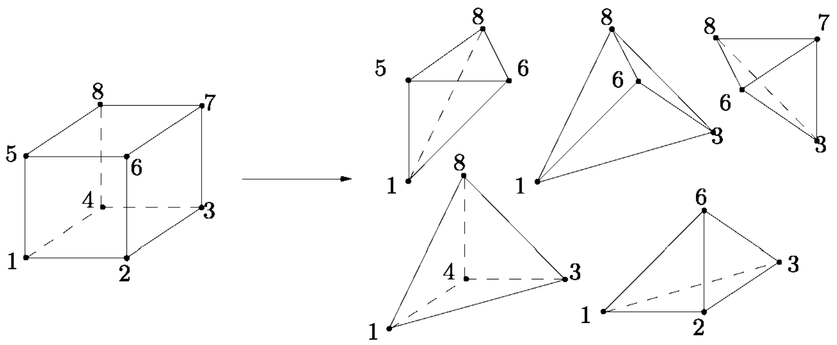


Figure 18.4: *Analysis of a hex by means of five tets.*

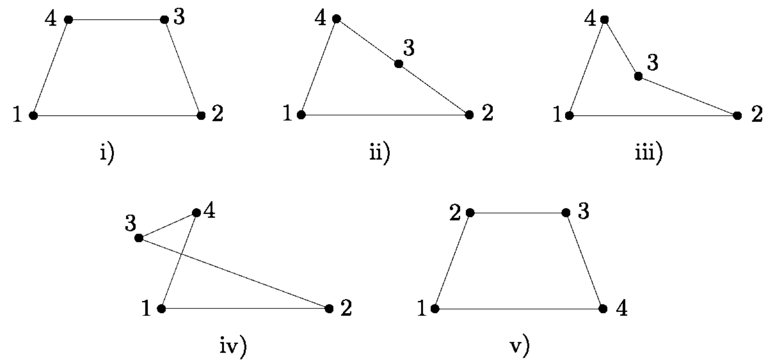


Figure 18.5: *The various configurations for a quad. A positive quad i); a degenerated quad ii); a non-convex quad iii); a self-intersecting quad iv); and a negative quad v).*

Localization processes. The localization of a given point in a mesh is a frequent issue that arises in various situations. In Chapter 7, we saw that it enables us to find, in a current mesh, the element within which a given point falls, for instance, the point we want to insert in this mesh. Also in Chapter 2, this localization problem was mentioned in the examples on data structures and basis algorithms.

Here, we consider this problem again while observing that the localization of a given point in a given mesh can be completed with the help of surface area or volume evaluations. For the sake of simplicity we first restrict ourselves to simplicial meshes and we consider a convex domain.

Let P be a point and $\mathcal{T}$ be a simplicial mesh whose elements are denoted by K. The method relies on the observation that, given an element K in $\mathcal{T}$, we can compute the $d+1$ (d being the spatial dimension) surface areas (volumes) of the virtual-simplices defined by joining P to the $d+1$ edges (faces) of K, and if the $d+1$ quantities are positive then P falls in K, otherwise, if there exist one or several negative (null) quantities, then P can be classified as a member of a particular precise half-plane (half-space).

As a consequence, if K_0 is an arbitrary element in $\mathcal{T}$, we compute the $d+1$ surface areas (volumes) associated with K and we decide to pass to the neighboring element of K_0 through the edge (face) related to the negative quantity (we assume that the neighboring relationships from element to element are known). Then the element thus-defined is used as the *basis* of the localization process and we repeat the same procedure.

It should be noted that using a control space (see Chapter 1) enables us to pick as the element K_0 an element not too far from P, resulting in a low cost algorithm.

Remark 18.1 *For meshes other than simplicial ones, the same method applies while, in some cases, replacing the elements with simplices. Thus, the problem* *could be significantly more expensive in terms of CPU time.*

Remark 18.2 *Non-convex domains lead to a tedious solution of the localization problem. Indeed, it could be necessary to pass through a boundary in order to reach the solution. Thus, the latter problem must be addressed, which is not so easy.*

Intersection processes. Similarly, surface area or volume computations may serve to solve some intersection problems. Indeed, we define some adequate virtual-elements whose surface area or volume signs allow the decision.

18.2 Mesh quality (classical case)

Relatively easy to define for simplicial meshes, the notion of quality must be carefully addressed for other types of meshes. In every case, several criteria may be used to evaluate mesh quality.

Shape or aspect ratio

This notion is basically associated with simplicial elements (triangles[1] or tetrahedra). The aspect ratio of a given simplex K is defined by:

$$\mathcal{Q}_K = \alpha_2 \frac{h_{max}}{\rho_K}\,, \tag{18.5}$$

where h_{max} is the element *diameter*, i.e., its longest edge while ρ_K is the inradius of element K. Notice that this value varies between 1 to ∞ and that, moreover, the closer $\mathcal{Q}_K$ is to 1, the better triangle K is. Indeed, $\mathcal{Q}_K$ measures the so-called *degradation* of element K. In two dimensions, we have:

$$\mathcal{Q}_K = \alpha_2 \frac{h_{max} p_K}{S_K} \tag{18.6}$$

where p_K is the half-perimeter of K and S_K is the surface area of K. Similarly, in three dimensions, we have:

$$\mathcal{Q}_K = \alpha_3 \frac{h_{max} S_K}{V_K} \tag{18.7}$$

where, now, S_K is the sum of the face surface areas and V_K is the volume of K. In these expressions, α is a normalization factor which results in the value one for the equilateral (regular, in three dimensions) simplex.

Exercise 18.1 *Find the value of the normalization coefficients α of Relationships (18.6) and (18.7).*

Remark 18.3 *In terms of computation, it is advisable to compute the inverse of the above relationships. In this way a zero surface (volume) element does not lead to a numerical problem.*

In Figure 18.6, we give a graphic impression of the way in which the quality of a triangle, Relation (18.6), varies according to the location of the three vertices.

[1] See the next chapter for surface triangles.

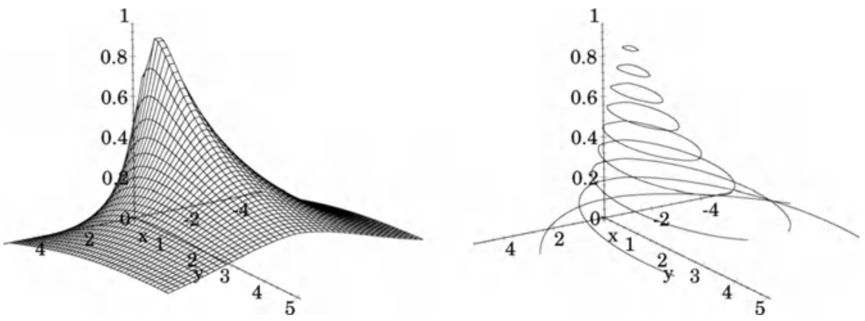


Figure 18.6: *Variation of the aspect ratio according to vertex C of triangle ABC, given AB. Edge AB spans the interval $[-0.5, 0.5]$ along the x-axis (rear) and C is in the half-plane bounded by this axis. Left: quality function. Right: iso-contour values of the quality function. For the sake of clarity, this figure displays the inverse of the quality defined in Relationships (18.5) and (18.6) (note that the maximum value 1 is obtained when C is at location $(0, \frac{\sqrt{3}}{2})$).*

Other criteria

Numerous quality functions can be used as an alternative way to determine the quality of a given simplex. The simplest one is:

$$\mathcal{Q}_K = \beta_2 \frac{h_s^2}{S_K} \tag{18.8}$$

in two dimensions, with β a normalization factor and $h_s = \sqrt{\sum_{i=1}^{3} L_i^2}$ where L_i is the length of edge i of triangle K. Similarly, in three dimensions, we have:

$$\mathcal{Q}_K = \beta_3 \frac{h_s^3}{V_K} \tag{18.9}$$

where $h_s = \sqrt{\sum_{i=1}^{6} L_i^2}$, L_i also being the length of edge i of tetrahedron K.

Exercise 18.2 *Find the value of the normalization coefficients β for the above cases.*

Remark 18.4 *See the previous remark regarding the case where zero elements may exist.*

Apart from the two above functions to appreciate the quality of a simplex, we encounter numerous other ways. For more details about this, see, among others, [Cavendish *et al.* 1985], [Baker-1989b] or [deCougny *et al.* 1990] or again [Dannelongue, Tanguy-1991] and a survey by [Parthasarathy *et al.* 1993].

Before enumerating some of the possible quality measures (in three dimensions), we introduce a few notations. For a given element K, with volume V_K, the inradius is denoted by ρ_K and the circumradius is r_K. The length of edge i of K is L_i, the surface area of face i is S_i. We now introduce $S_K = \sum S_i$, the sum of the surface areas of the faces of K, h_{max} the diameter of K, i.e., $h_{max} = \max_i L_i$, and $h_{min} = \min_i L_i$ and, lastly, L_{mean} the average of the L_is. Then, the quality measures are as follows:

- $\dfrac{r_K}{\rho_K}$, the ratio between the radii of the two relevant spheres,
- $\dfrac{r_K}{h_{max}}$ (or $\dfrac{r_K}{h_{min}}$), the ratio between the circumradius and the element diameter (the shortest edge),
- $\dfrac{h_{max}}{h_{min}}$ or $\dfrac{h_{min}}{h_{max}}$, the ratio between the edges with extremal lengths,
- $\dfrac{V_K^4}{(\sum\limits_{i=1}^{4} S_i^2)^3}$, the ratio between the volume and the surface areas of the faces,
- $\dfrac{h_{mean}^3}{V_K}$, the ratio between the average edge length and the volume,
- δ_{max}, the maximal dihedral angle between two faces,
- θ_{min}, the minimal solid angle associated with the vertices of K.

with, for the two last measures, the following definitions.

Definition 18.1 *The dihedral angle between two faces is the value*

$$\pi \pm \arccos\langle \vec{n}_1, \vec{n}_2 \rangle$$

depending on the configuration of the two faces with respect to $\vec{n}_i$, the normals of the faces under interest.

Definition 18.2 *The solid angle at a vertex is the surface area of the portion of a unit sphere centered at this vertex bounded by the three faces sharing this point.*

Exercise 18.3 *Find the equivalent definitions in two dimensions (for those cases where it makes sense).*

Exercise 18.4 *Give the normalization factors resulting in a unit value for the above quality measures (when it makes sense).*

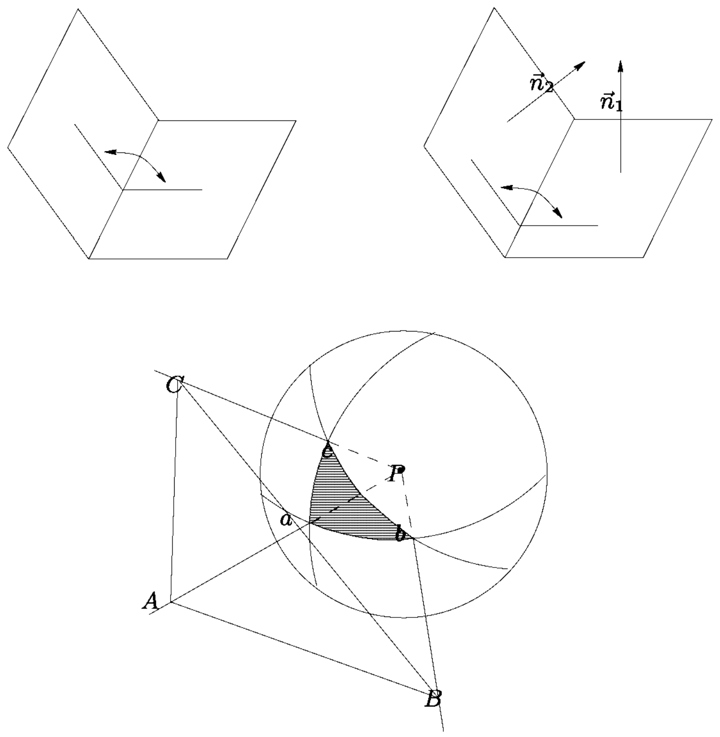


Figure 18.7: *Dihedral angle (top) and solid angle (bottom).*

Simplicial element classification

The above quality criteria enable us to decide whether a simplex is good or not in terms of quality. It could be interesting to classify the bad-quality elements so as to discard one or other causes from which this bad quality results. This task is covered in the exercises (see also [George, Borouchaki-1997] where a detailed discussion is given).

Exercise 18.5 *Show that, in terms of the geometrical aspect, there exist three types of triangles (hint: there are only two types of ill-shaped triangles). Show the criterion leading to this classification without ambiguity.*

Exercise 18.6 *Similarly, show the eight types of tetrahedral elements (hint: consider both the volume and the type of the element faces following the triangle classification).*

Exercise 18.7 *Discuss how the above measures allow (or are not suitable in all cases) for the element classification.*

Non-simplicial elements

In this section, we discuss the quality of quad, hex and pentahedral elements.

Quad. In two dimensions[2], quads are usually appreciated through several quantities including the so-called *aspect ratio*, *skew parameter* along with two *taper coefficients* (one taper being related to one axis); see [Robinson-1987] and [Robinson-1988]. Although quite natural for some peculiar geometries, the above notions are not so well defined in a general context. Thus, we can look for other types of measurements. One idea could be to find in the above series of measures those which apply in this case. For instance, the ratio $\frac{h_{max}}{h_{min}}$ where h_{min} and h_{max} denote the smallest and the largest edges can be considered. Nevertheless, this measure must be coupled with information about angles between two edges.

We would like to propose a new[3] formula which offers several advantages. First, it looks like Relationship (18.7). Second, it appears to be an efficient way to discriminate between the elements (in terms of quality). Then, negative or non-convex quads are detected on the fly. Finally, only one measure is involved. Thus, we propose the following:

$$Q_K = \alpha \frac{h_{max}\, h_s}{S_{min}}, \tag{18.10}$$

where α is a normalization factor ($\alpha = \frac{\sqrt{2}}{8}$), S_{min} is the minimum of the four surface areas that can be associated with K. Indeed, these surface areas are those of the four triangles that can be defined (review the way to decide whether a quad is convex or not, Figure 18.1), $h_s = \sqrt{\sum_{i=1}^{4} L_i^2}$ with L_i the length of edge i of K and h_{max} is the longest length among the four edges and the two diagonals. In practice, we compute S_{min} and if this value is correct, we pursue the test, otherwise, the quad being invalid, it is useless to continue.

Remark 18.5 *Another way to judge a quad is to consider that its quality is that of the worst triangle that can be constructed based on three of the quad vertices.*

Remark 18.6 *Following Remark 18.3, it is advisable to compute the inverse of the above quality function (to avoid a possible overflow when one of the surface areas involved is zero).*

Figure 18.8 gives an impression of how the measure of Relation (18.10) varies according to the geometry of the analyzed quad. In this respect, the classical aspect ratio, skew parameter and taper coefficients as well as the global measure are displayed for an arbitrary variation.

Preliminary remarks about hex and pentahedral elements. Examining the quality of the element faces only gives a rough idea of the element quality. Indeed, the case of the *sliver* tet where the four faces are well-shaped while the volume is (almost) zero clearly indicates that the face qualities alone are not sufficient to qualify an element. In addition, quad faces are not necessarily planar,

[2]See the next chapter for a surface quad.
[3]As far as we know.

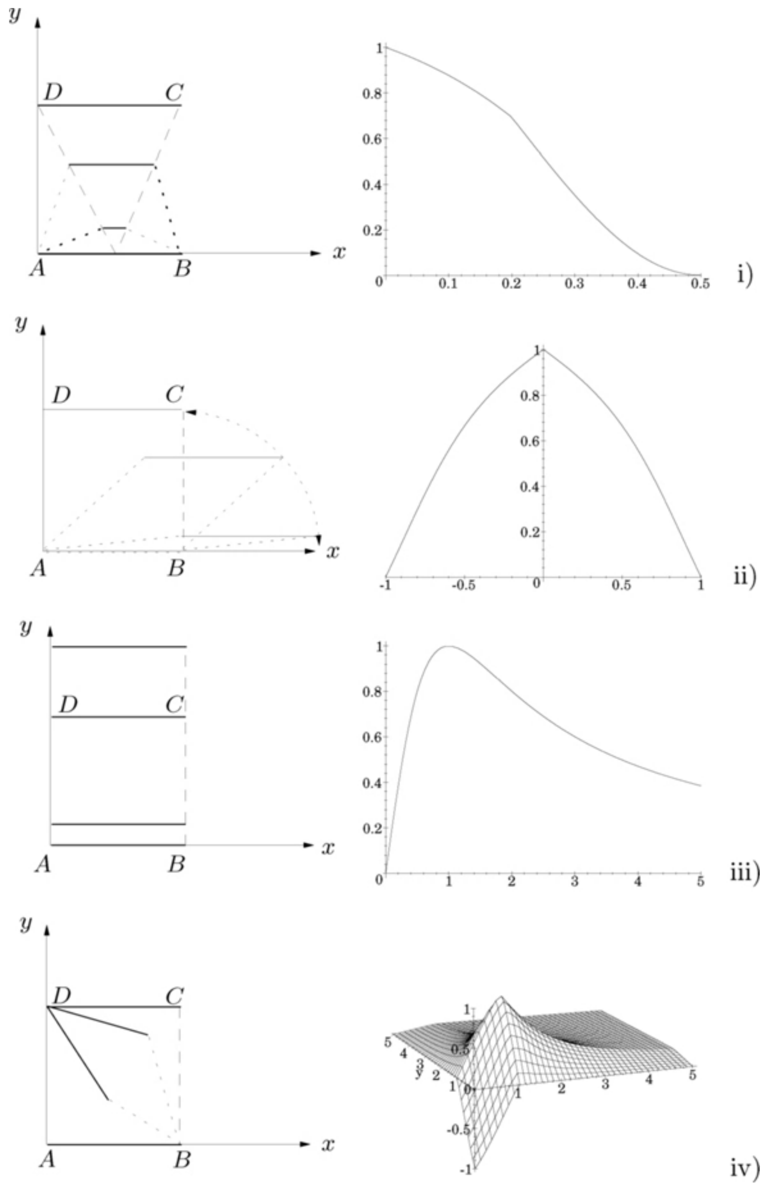


Figure 18.8: *Variation of the quality function in a quadrilateral (edge AB is fixed). Optimal quadrilateral is denoted by $ABCD$: i) taper measure: points C and D vary along the dashed line; ii) skewness measure: the point C varies along the circle of radius $1 = \|AB\|$ (edge CD is parallel to AB); iii) aspect ratio measure: the edge CD (along the y-axis) varies in the interval $[0, 5]$; iv) global measure: points A, B and D are fixed, point C varies in the interval $[0, 5] \times [0, 5]$.*

thus introducing some extent of torsion that must be taken into account to qualify an element with such a face. Thus, other measurements are needed which are not purely two-dimensional.

Quad face. In advance (see Chapter 19), we introduce the *roughness* (or smoothness) of a quad face. We consider the two diagonals of the quad $ABCD$, *i.e,* AC and BD and we consider the two dihedral angles that are defined in this way. Then, the smoothness of $ABCD$ is:

$$\mathcal{S}_{ABCD} = \min(\mathcal{P}_{AC}, \mathcal{P}_{BD}), \tag{18.11}$$

where

$$\mathcal{P}_{AC} = \frac{1 + \langle \vec{n}_{ABC}, \vec{n}_{ACD} \rangle}{2} \quad \text{and} \quad \mathcal{P}_{BD} = \frac{1 + \langle \vec{n}_{ABD}, \vec{n}_{BCD} \rangle}{2},$$

are the two *edge planarities* involved in the construction. In these relations, $\vec{n}_{ABC}$, for instance, denotes the unit normal related to triangle ABC.

In practice, the face smoothness measure will be used to quantify the *torsion* of a three-dimensional element.

Hex. As there is not a unique formula as for a quad or different criteria for analysis, it is possible to appreciate the quality of an element by observing the quality of the various partitions of it into tets. The worst tet then gives the desired answer which, combined with the element face qualities, enables us to conclude.

Pentahedral element. The previous idea can also be retained, the element analysis is based on a partition by means of tets.

18.3 Mesh quality (isotropic and anisotropic case)

The previous discussion allows mesh appreciation when the only concern is the shape aspect of the elements in the mesh, i.e., without other considerations (such as metric specifications). We now turn to a different view-point. In what follows, a metric map is supplied and the mesh is evaluated to decide if it conforms to this specification or not. In other words, the problem concerns the appreciation of a given mesh with regard to some specified size (isotropic case) or directional and size prescriptions (anisotropic case).

Efficiency index

Let l_i be the length of the edge i with respect to the given metric. The *efficiency index* of a mesh is defined as the average value of the squares of the differences to 1 of all mesh edge lengths (let na be the number of mesh edges), hence:

$$\tau = 1 - \frac{1}{na}.\sum_{i=1}^{na}(1 - e_i)^2, \tag{18.12}$$

with $e_i = l_i$ if $l_i \leq 1$, $e_i = 1/l_i$ if $l_i > 1$.

This coefficient seems adequate for a quick estimation of the mesh quality with respect to a given metric map. Table 18.1 presents the sensitivity of this measure in the case of an isotropic map, l being constant for all mesh edges (which is highly unlikely although it shows the effect of the size variation on τ) and indicates that the edge lengths are l times too long or too short (a value $l = 5$ or $l = 0.2$ means that all edges are 5 times too long or 5 times too short). The optimal value is $l = 1$ and in fact, any value of τ greater than 0.91 ensures a reasonable mesh quality with respect to the metric map.

l	100	20	10	5	3	2	$\sqrt{2}$	1.3	1.2	1.1	1
τ	0.019	0.097	0.19	0.36	0.51	0.75	0.91	.9467	.9722	.9917	1.

Table 18.1: Sensitivity of the efficiency index.

The reader could consult this table in order to interpret the numerical results obtained in a given governed problem. More recently, [Dobrzynski-2005] proposed another more sensitive formula:

$$\tau = exp\left(\frac{1}{na}.\sum_{i=1}^{na} e_i\right), \tag{18.13}$$

with $e_i = l_i - 1$ if $l_i \leq 1$, $e_i = 1/l_i - 1$ if $l_i > 1$.

Element quality

In this section, we turn to a general point of view.

Simplicial elements. In this case element quality reduces to the above aspect ratio. When the metric map the mesh must conform to is isotropic, we return to Relation (18.6) or equivalent relations. In the case of an anisotropic metric map, the notion of aspect ratio is more delicate. In fact, an approximate expression can be used. For instance, in two dimensions, $\mathcal{Q}_K$, the quality of triangle K, can be defined as:

$$\mathcal{Q}_K = \max_{1\leq i\leq 3} Q^i_K, \tag{18.14}$$

where Q^i_K is the triangle quality in the Euclidean space related to the metric specified at any vertex P_i of K. To evaluate the quality Q^i_K of K, it is merely necessary to transform the Euclidean space associated with the metric specified at any vertex P_i in the classical Euclidean space and to consider the quality of the triangle K_i which is the triangle image of K. This means that:

$$Q^i_K = \mathcal{Q}_{K_i}. \tag{18.15}$$

Let $(\mathcal{M}_i)_{1\leq i\leq 3}$ be the metrics specified at the vertices $(P_i)_{1\leq i\leq 3}$ of K. We have:

$$\mathcal{M}_i = \mathcal{P}_i \Lambda_i \mathcal{P}_i^{-1} \qquad 1 \leq i \leq 3, \tag{18.16}$$

where $\mathcal{P}_i$ is the matrix enabling us to transform the canonical basis into the basis associated with the eigenvectors of $\mathcal{M}_i$ and $\Lambda_i = \begin{pmatrix} \lambda_{i,1} & 0 \\ 0 & \lambda_{i,2} \end{pmatrix}$ is the diagonal matrix formed by the eigenvalues of $\mathcal{M}_i$. Let $(h_{i,j})_{1 \leq i \leq j \leq 2}$ be the quantities defined by $h_{i,j} = 1/\sqrt{\lambda_{i,j}}$; these values represent the unit length in the direction of the eigenvector related to the eigenvalue $\lambda_{i,j}$ of $\mathcal{M}_i$. The matrix $\mathcal{T}_i$ transforming the Euclidean space associated with $\mathcal{M}_i$ in the usual Euclidean space is defined by:

$$\mathcal{T}_i = \mathcal{H}_i \mathcal{P}_i \,, \tag{18.17}$$

where $\mathcal{H}_i$ is the diagonal matrix $\begin{pmatrix} 1/h_{i,1} & 0 \\ 0 & 1/h_{i,2} \end{pmatrix}$. As a result, the vertices of K_i are $\mathcal{H}_i\mathcal{P}_iP_1, \mathcal{H}_i\mathcal{P}_iP_2$ and $\mathcal{H}_i\mathcal{P}_iP_3$, respectively, and we have:

$$Q_K^i = \alpha \frac{h'_{max} \sum\limits_{1 \leq j < k \leq 3} \|\mathcal{H}_i\mathcal{P}_i\overrightarrow{P_jP_k}\|}{Det(\mathcal{H}_i\mathcal{P}_i\overrightarrow{P_1P_2}, \mathcal{H}_i\mathcal{P}_i\overrightarrow{P_1P_3})} \tag{18.18}$$

with α the same normalization factor as in the classical case, and

$$h'_{max} = \max_{1 \leq j < k \leq 3} \|\mathcal{H}_i\mathcal{P}_i\overrightarrow{P_jP_k}\| .$$

However, as

$$Det(\mathcal{P}_i) = 1 \,,$$

$$Det(\mathcal{H}_i) = \sqrt{\lambda_{i,1}\lambda_{i,2}} = \sqrt{Det(\mathcal{M}_i)}$$

and

$$\|\mathcal{H}_i\mathcal{P}_i\overrightarrow{P_jP_k}\| = \sqrt{{}^t\overrightarrow{P_jP_k}\mathcal{M}_i\overrightarrow{P_jP_k}} \,;$$

we have

$$Q_K^i = \alpha \frac{\max\limits_{1 \leq j < k \leq 3} \sqrt{{}^t\overrightarrow{P_jP_k}\mathcal{M}_i\overrightarrow{P_jP_k}} \sum\limits_{1 \leq j < k \leq 3} \sqrt{{}^t\overrightarrow{P_jP_k}\mathcal{M}_i\overrightarrow{P_jP_k}}}{\sqrt{Det(\mathcal{M}_i)}Det(\overrightarrow{P_1P_2}, \overrightarrow{P_1P_3})} \,. \tag{18.19}$$

Exercise 18.8 *Show that the above relation reduces to the classical aspect ratio formula when we consider the case of isotropic metric maps (i.e., the corresponding matrices are identity matrices with a given ratio).*

Elements other than simplices. A quad with unit length edges and $\sqrt{2}$ length diagonals are the targeted values. In practice, these values are sufficient to appreciate the element (and no angle consideration is needed). Similar notions extend to hex and pentahedral elements.

Optimal mesh

A formal notion of an optimal mesh is relatively tedious to define. This question was already raised in Chapter 1, while noticing that optimality must be considered with regard to the reason for which the mesh has been constructed and therefore with regard to its further use.

Given $\mathcal{Q}_K$, a quality measure for the element K in mesh $\mathcal{T}$, we have previously defined the mesh quality as:

$$\mathcal{Q}_{\mathcal{T}} = \max_{K \in \mathcal{T}} \mathcal{Q}_K \,.$$

In a later section we will see that other characteristic values may be suitably used to analyze a mesh, such as the mean of the element qualities, the distribution of the elements based on their quality, etc. Whatever the choice, it may be noticed that the notion of optimality lies in theory in a family of meshes. For instance, given several meshes, it can be said that the optimal mesh is that for which the chosen measure is optimal (minimal if we return to the usual definition about the shape of the elements in an isotropic simplicial mesh). In practice, the size (the number of vertices or elements) must also be used as one of the parameters in the analysis (so as to minimize the computational cost in a numerical simulation using this mesh, for example). Therefore, it could be stated that an optimal mesh is that for which the chosen quality function is optimal while, at the same time, its number of vertices (elements) is minimal.

In practical terms, it may be concluded that the optimal mesh is that which gives a suitable compromise between various criteria. The problem then reduces to only one quality measure. In fact, in the isotropic case in two dimensions and for a mesh composed only of triangles, a quality value of 1 (i.e., close to 1) implies that:

- the elements in the mesh have a quality value close to 1,
- the number of elements is minimal.

A quality value close to 1, for a triangle, implies that its edges have a length close to 1 (or a constant value h). For a given size map, we again see that this means that the edge lengths are close to 1 (with respect to this size map). Therefore, the definition we now propose is rather natural (and intuitive).

Definition 18.3 *A unit mesh is a mesh with unit length edges.*

In two dimensions, a unit triangle (with unit edges) is optimal (it is equilateral) while this is not the case for a tet.

Remark 18.7 *A triangle with unit length edges is necessarily a good element whose surface area is $\frac{\sqrt{3}}{4}$. In contrast, a tet with unit length edges*[4] *may have a volume as small as we want thus corresponding to a ill-shaped element (the infamous* sliver*).*

[4]Consider a tet where four edge lengths are close to one and where the two other edge lengths are close to $\sqrt{2}$. The edges therefore have a length "close" to 1 and, nevertheless, the tet volume is zero!

Following this remark, the notion of optimality is made more precise. It includes a length aspect combined with a surface (volume) aspect.

Definition 18.4 *An optimal simplicial mesh is a unit mesh in which the element surface areas are $\frac{\sqrt{3}}{4}$ in two dimensions or in which the element volumes are $\frac{\sqrt{2}}{12}$ in three dimensions (or, more generally, $\frac{\sqrt{d+1}}{d!\sqrt{2^d}}$, in d dimensions).*

Remark 18.8 *Notice (again) that optimality is here related to a quality measure with regard to a size map and not directly related to the number of elements (see the above remark about this aspect).*

After the two above definitions together with the previous remarks and in practical terms, the efficiency index is a consistent way to judge a mesh in two dimensions. In three dimensions, the same analysis must be based on edge length appreciation and on the aspect ratio of the elements (in order to see whether the element volumes are consistent or not).

Remark 18.9 *For elements other than simplices, unit edge length could be a reasonable requirement as coupled with other considerations in some cases (for instance, $\sqrt{2}$ length diagonals for a quad as previously mentioned).*

Remarks about optimality

Discussing optimality may raise to some interesting issues. Various questions may be discussed. For a given problem (based on what data are known):

- is there a mesh with unit quality?
- is there a mesh with minimal size?

For an *a priori* given number of elements:

- is there a mesh having this number of elements and, if so, what is its quality?

Note, in practice, and mostly in three dimensions, that the objective is to have good quality meshes (i.e., with a quality value close to the theoretical value) and that a mesh not too far from this abstract target is generally considered to be satisfactory.

18.4 Tools for mesh optimization

Various local tools can be used for optimization purposes. The most popular include the following:

- node relocation,
- edge collapsing (to remove a vertex),
- edge swapping,

- face swapping,
- vertex degree relaxation (which, as will be seen, can be achieved by a judicious combination of the three above tools),
- edge splitting (to add a vertex), etc.

Actually, mesh optimization tools can be classified into two categories. Those maintaining mesh connectivity (i.e., acting on the vertex positions) and those acting on mesh connectivity (while the vertex locations are preserved).

Optimization maintaining the connectivities

In this category of local tools, we basically encounter those methods which result in moving the element vertices. All methods that can be developed in this sense can be seen as a variant of the well known Laplacian smoothing, [Field-1988], [Frey, Field-1991].

Basically, a process acting on node relocation concerns the so-called balls. Let us recall the following definition.

Definition 18.5 *Let P be a vertex in mesh $\mathcal{T}$, the ball associated with P is the set of elements in $\mathcal{T}$ having P as a vertex.*

A ball could be a closed ball (the vertex is an internal vertex) or an open ball (the vertex is a boundary vertex). In the following, we only consider closed balls.

The simplest node relocation method can be written as:

$$P' = \frac{1}{n}\sum_{j=1}^{n} P_j\,, \tag{18.20}$$

where the P_js are the vertices of the ball other than P. A first variation consists of adding some relative weights; thus, we obtain:

$$P' = \frac{\sum\limits_{j=1}^{n} \alpha_j P_j}{\sum\limits_{j=1}^{n} \alpha_j} \tag{18.21}$$

where α_j is an appropriate weight associated with point P_j.

Nevertheless, before going further, we propose replacing the above scheme by a relaxation method. In fact, efficiency reasons can be involved along with the two following observations.

Remark 18.10 *Above point P' (one or the other) could fall outside the ball in the case of a non-convex ball.*

and, as a consequence

Remark 18.11 *Moving a given point to its "optimal" location may result in an invalid mesh, thus cancelling the operation. A non-optimal relocation, however, may improve the mesh quality to some degree.*

Hence, an auxiliary point, P^*, is introduced, for instance, such as in Relationship (18.20):

$$P^* = \frac{1}{n} \sum_{j=1}^{n} P_j \,, \tag{18.22}$$

and a relaxation scheme is defined as:

$$P = (1 - \omega) P + \omega P^* \,, \tag{18.23}$$

where ω is the relaxation parameter[5]. Following this method, we now introduce various vertex relocation methods.

Laplacian smoothing. The relaxed variant of this well-known method uses auxiliary point defined in Relationship (18.22). Following the previous remark, an explicit check of the positiveness of the surface areas (volumes) is required.

Remark 18.12 *Classical optimization strategies can be used for smoothing purposes. A cost function is defined which is assumed to be sufficiently smooth. Descent directions are then exhibited and the process is governed as in a classical optimization process, [Freitag, Gooch-1997]. It should be noted that most of the quality functions described above are non-differentiable thus leading to a tedious optimization process.*

Weighted smoothing. In this case, a weight is associated with each point making it possible to define the auxiliary point by

$$P^* = \frac{\sum\limits_{j=1}^{n} \alpha_j P_j}{\sum\limits_{j=1}^{n} \alpha_j} \,, \tag{18.24}$$

where an appropriate choice of the α_js must be made (see below).

Smoothing based on element quality. Provided with a simplicial mesh, we consider the ball of a given point P. Let f_j be the external edges (faces in three dimensions) of this ball. Then the elements in the ball are nothing more than the combinations (P, f_j) (where $j = 1, n$, n being the number of elements in the ball). Thus an ideal point I_j is associated with each f_j in such a way as to ensure an optimal quality (aspect ratio) for the virtual element (I_j, f_j). Using these points,

[5]It seems advisable to set ω close to one in two dimensions and smaller in three dimensions.

we define the smoothing process as:

$$P^* = \frac{\sum\limits_{j=1}^{n} \alpha_j I_j}{\sum\limits_{j=1}^{n} \alpha_j}, \tag{18.25}$$

where the α_j can be defined as follows:

- $\alpha_j = 1$, the weights are constant and we return to the classical method,
- $\alpha_j = 0$, for every element of the ball except for the worst one (in terms of quality) for which we take $\alpha_j = 1$,
- $\alpha_j = Q_{K_j}$, the weights are related to the quality of the elements,
- $\alpha_j = Q^2_{K_j}$, the weights are related to the square of the element qualities,
- or, more generally, $\alpha_j = g(Q_{K_j})$, meaning that the weights are related to a certain function g of the quality of the elements in the ball.

This method can also be applied in an anisotropic case by using the relevant definition of the quality, which leads to a different positioning of the points I_j.

Smoothing based on edge lengths. In this case, the key-idea is to define the I_js so as to obtain as far as possible a unit length for the edges emanating from these points. The unit length notion is based on the metric map which is assumed. Thus, a relation like:

$$I_j = P_j + \frac{\overrightarrow{P_jP}}{\|\overrightarrow{P_jP}\|}\, \overline{h}_j\,, \tag{18.26}$$

where the P_js are the vertices of the external faces in the ball and $\overline{h}_j$ is the average size related to the edge P_jP approaches the desired result. Indeed, the above relation is nothing more than:

$$I_j = P_j + \frac{\overrightarrow{P_jP}}{l_{\mathcal{M}}(P_jP)}\,, \tag{18.27}$$

where $l_{\mathcal{M}}(P_jP)$ is the length of the edge P_jP evaluated in the metric $\mathcal{M}$ associated with the edge.

Non-simplicial elements. To some degree, the above methods extend to this case. In particular, for a quadrilateral element, an edge length based smoothing operator must lead to unit edge lengths and a $\sqrt{2}$ length for the two diagonals of the elements (as previously indicated).

Global smoothing. The above discussion concerns a local smoothing procedure where the points are considered one at a time. A global smoothing procedure can be developed leading to the solution of a global problem. It is, however, uncertain whether such a method is really efficient.

Topologically driven node relocation. The objective is now to relocate a vertex and, in addition, to

- move it away from a given edge,
- move it away from a given plane,
- move it along a given edge,
- etc.

Any method of the above type can be used by adding the constraint during the analysis of the criterion that must be optimized. In some cases, the goal is not to optimize the mesh but just to maintain some degree of quality, meaning that the main concern is to remove an undesired topological pattern rather than to effectively optimize the mesh.

Exercise 18.9 *Revisit the smoothing techniques in the case where a ball may be an open ball (for a boundary vertex for instance).*

Optimization maintaining the vertex positions

We turn to various local tools that leave the vertex location unchanged.

Edge swapping in two dimensions. Edge swapping[6], or simply swap, is a rather simple topological operation involving swapping the edge shared by two elements. In the case of triangular elements, the swap is possible since the quad formed by the two adjacent triangles is a convex polygon. Swap can also be performed in quad meshes or mixed meshes (whose elements include both triangles and quadrilaterals). In general, a swap procedure must first be validated to ensure that the resulting mesh is still valid and, second, be evaluated with regard to the optimization criterion that must be enhanced.

Edge swap can be seen as an optimization procedure in itself or it can be used as one ingredient in some more sophisticated processes (node removal, degree relaxation, etc.).

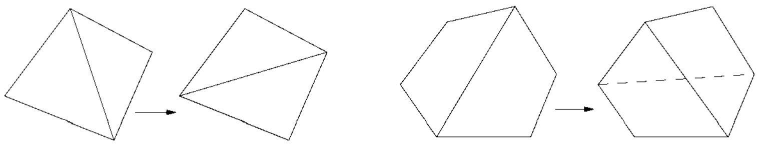


Figure 18.9: *Edge swapping for a pair of adjacent triangles (left-hand side) and in the case of adjacent quads (right-hand side) where two solutions are possible* a priori.

[6]Also referred to as diagonal swapping or diagonal flipping.

Generalized edge swapping (the three-dimensional case). In what follows, only simplicial meshes are discussed. Within this context, we first consider the extension to three dimensions of two-dimensional edge swapping. This leads to a *face swapping* where the face common to two tetrahedra is removed, an edge is created and the two-element initial polyhedron is replaced by a three tetrahedron polyhedron (note that only convex polyhedra can be successfully dealt with).

The inverse local transformation can be defined leading to replacing three tetrahedra sharing an edge by two tetrahedra by suppressing the edge considered. Actually, a more general transformation corresponds to this operation. It acts on a so-called *shell.* Recall the following definition.

Definition 18.6 *Let $\alpha\beta$ be an edge in mesh $\mathcal{T}$, the shell associated with $\alpha\beta$ is the set of elements sharing this edge.*

As for balls, a shell could be opened or closed. In what follows we only discuss the case of closed shells where the edge of the shell is an internal edge.

Then the key idea is to consider such shells. Formally speaking, the generalized edge swapping operator leads to considering all the possible triangulations of a *pseudo-polygon* associated with the edge. The vertices of this polygon are defined by the shell vertices other than α and β, the two endpoints of the edge defining the shell. Figure 18.10 shows these possible remeshings in the case of a five-element shell, every triangulation being formed by joining all the triangles of the polygon with both α and β, in order to define the pair of tetrahedra which are part of the desired mesh.

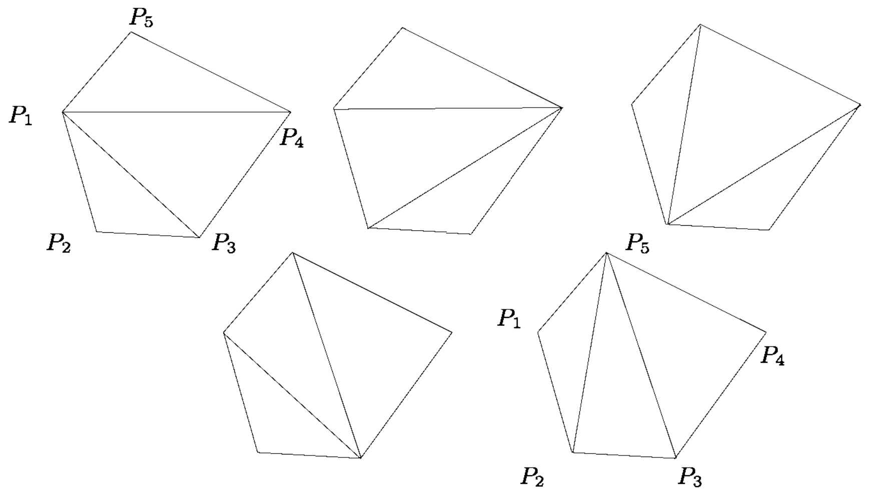


Figure 18.10: *The five triangulations related to a five-point polygon.*

The Catalan number of order n

$$Cat(n) = \frac{(2n-2)!}{n!(n-1)!}$$

gives the maximal number of *topologically* possible triangulations, N_n, of a shell[7] of n elements. Indeed, we have

$$N_n = Cat(n-1)\,. \tag{18.28}$$

Exercise 18.10 *Establish the previous relation.*

n	3	4	5	6	7	8	9	10	11	12	13
N_n	1	2	5	14	42	132	429	1,430	4,862	16,796	58,786
Tr_n	1	4	10	20	35	56	84	120	165	220	..

Table 18.2: Number of topologically different triangulations that are valid as a function of the number of vertices in the polygon related to one edge.

Table 18.2 gives N_n, the number of possible triangulations as a function of n. It also indicates Tr_n the number of different triangles in each possible triangulation. In this enumeration, the validity of the triangulations is not considered (only the topological aspect is taken into account).

Remark 18.13 *As previously mentioned, the swap procedure requires that the polyhedron under treatment is convex for a two or three element pattern while this requirement is not strictly needed for patterns involving more than three elements. Indeed, the swap must be validated by checking explicitly the positiveness of the element volumes that are concerned.*

Remark 18.14 *The generalized swap is a tedious problem for elements other than simplices.*

Edge collapsing. Provided with an edge, $\alpha\beta$, we replace this edge by only one point A. Formally speaking, this leads to positioning α on β, or vice versa or again finding a point location between α and β. Figure 18.11 shows the three possible solutions on an example. From a practical point of view, it is sufficient to check if the ball of point A, resulting from the reduction, is valid. To this end, we examine the shell $\alpha\beta$ and we check the validity of the balls of α and β when these two vertices are replaced by point A.

This operator may be classified among the geometric operators as it maintains the connectivities, if the new connections to A are seen as the "union" of the former connections to α and β.

Degree relaxation. First, we give the definition of the degree of a mesh vertex.

Definition 18.7 *The degree or the valence of a mesh vertex is the number of edges*[8] *emanating from this point.*

[7]The topologically possible solutions in three dimensions are constructed by enumerating all the two-dimensional *geometric* valid remeshings of a convex (planar) polygon with n vertices.

[8]It is also, in two dimensions, the number of elements sharing the point.

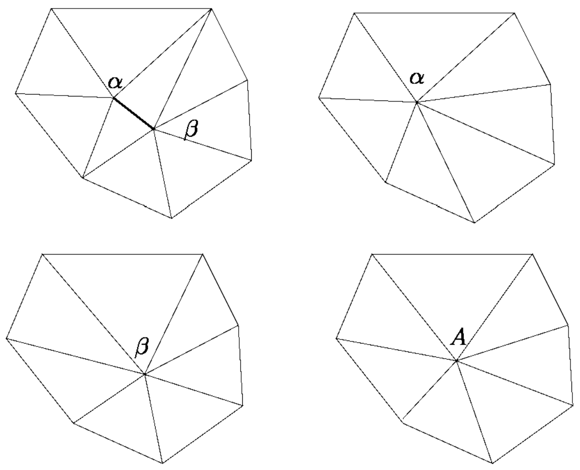


Figure 18.11: *Edge collapsing. The initial pattern can be replaced by three different configurations, vertex β is collapsed with vertex α, vertex α is collapsed with vertex β or these two vertices are collapsed using vertex A, for example, the midpoint of the initial edge.*

We consider the edges and not the elements because in the matrices used in finite element calculus and for P^1 simplicial meshes, the edges determine the matrix bandwidth.

Now the question is deciding what an optimal degree is. In two dimensions, a value of six is desired for simplicial meshes while a value of four is optimal for quad meshes. In three dimensions, a value of twelve allows for tetrahedral meshes[9] while six is the optimal degree of a vertex in a hexahedral mesh.

A point with a degree less than the targeted value is said to be *under-connected*, while a point with a degree larger than this value is said to be *over-connected*. Note that the same notion applies to an edge.

Relaxing the degree of a mesh consists of modifying the vertex (or edge) degrees, by means of topological operators, so as to tend on average to the targeted value; see [Frey, Field-1991].

Remark 18.15 *Optimizing a mesh with bad vertex degrees may result in poor results. This means that the optimization tools are penalized when dealing with such situations. On the other hand, optimization tools may lead to good results when a degree relaxation has been carried out beforehand.*

Ill-constrained entities. This notion applies to vertices (as previously seen) as well as to edges, faces or elements, which will now be discussed.

Definition 18.8 *A triangle is said to be over-constrained if two of its edges are*

[9]This value corresponds to a ball with 20 optimal elements, i.e., the triangulation by a regular icosahedron of the space centered at the point defining the ball.

members of the domain boundary. Similarly, a tet with three boundary faces is over-constrained.

Thus, in terms of edges or faces (extending the previous definition to elements other than simplices), we have the following.

Definition 18.9 *An internal edge is said to be over-constrained if its two endpoints are members of the domain boundary. Similarly, an internal face whose extremities are in the boundary is over-constrained.*

For most problems, such ill-constrained entities must be avoided. Thus, optimization tools can serve to remove this kind of pathologies.

Exercise 18.11 *Consider again the modification operators when the shells are opened (i.e., those associated with a boundary edge).*

Non-obtuse mesh

This point concerns simplicial meshes in two dimensions. Non-obtuse meshes are required for some particular applications. For instance, a problem solved by means of a finite volume method takes advantage of non-obtuse meshes. To specify the notion of a non-obtuse mesh, we firstly give the formal definition of such a mesh.

Definition 18.10 *A two-dimensional simplicial mesh is said to be non-obtuse if it does not include any obtuse angles, such angles being defined by the pairs of edges sharing a vertex.*

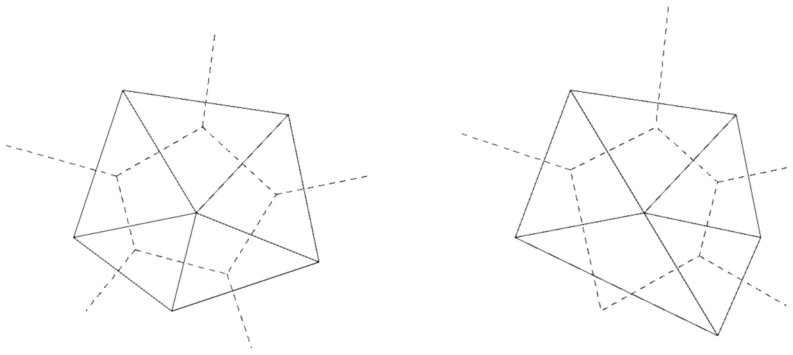


Figure 18.12: *Obtuse and non-obtuse meshes. The cells associated with the triangles sharing a given vertex are displayed. As the mesh is non-obtuse, the cell around the point is fully included in the ball of this point (left-hand side) while, on the other hand, it has a part outside this ball (right-hand side) when the mesh is obtuse.*

As mentioned earlier, non-obtuse meshes are of special interest in some applications. This is due to the fact that the cells around the element vertices are fully included in the corresponding balls. Note that the cells are constructed from the perpendicular bisector related to the element edges (Figure 18.12). The fundamental property is the orthogonality of these cells and the current mesh (more precisely the mesh edges and the cell edges are orthogonal). Hence, this nice feature coupled with the internal aspect of the cells can be a benefit for some problems.

Remark 18.16 *We return here to the Voronoï cells the duals of the triangulation (see Chapter 7)* **only** *in the case of a Delaunay mesh. In this respect, one could note that even a Delaunay mesh is not necessarily a non-obtuse mesh. See, for instance, Figure 18.13, where a point outside the circle ensures the Delaunay property but could be outside the two lines depicted in the figure, thus resulting in an obtuse angle.*

Clearly, a non-obtuse mesh allows the construction of cells enjoying the above properties (i.e., an orthogonality property while entirely inside the domain[10]). Thus, given a mesh resulting from such or such a method, it could be interesting to develop an algorithm that allows us to remove the obtuse angles (if any).

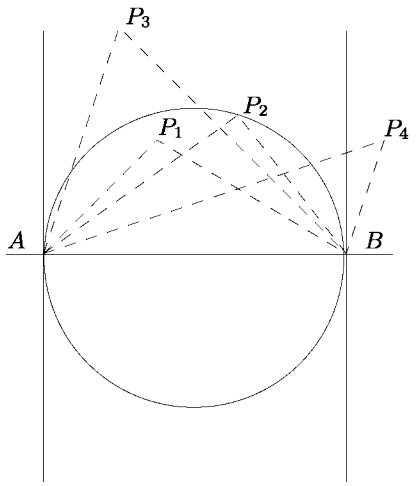


Figure 18.13: *Given an edge AB, we define the circle whose diameter is AB, then we construct the two lines orthogonal to AB passing through A and B. This results in three regions. The interior of the circle, the exterior of this circle exterior to the two above lines and the exterior of this circle inside the two lines. Clearly a point like P_1 (or P_4) leads to an obtuse triangle and a point like P_3 leads to a non-obtuse triangle.*

A precise analysis of Figure 18.13 can be fruitful to define an algorithm based on local modifications that makes it possible to suppress the obtuse angles.

A rough idea of the method could be to process all balls in the mesh (see Definition 18.5). Let P be the vertex defining a ball, we consider the edges external to this ball (thus, these edges act as the edge AB in Figure 18.13). We define the circles and the two orthogonal lines associated with these edges to exhibit a region intersection of the suitable area where P can be located. If this region is

[10]Note that defining the cells around the vertices by means of the median lines results in cells fully inside the domain but the orthogonality feature is lost.

not empty, then P is relocated inside and all the elements in the ball are non-obtuse[11]. Otherwise, a more subtle process must be defined. For instance, we can try to move A and/or B along AB so as to obtain a smaller edge, thus resulting in a smaller circle. Note also that vertices opposite a boundary edge have some degree of rigidity. To overcome this fact, points can be required to further subdivide the boundary edges.

Remark 18.17 *In the above method, essentially based on heuristics, no proof of convergence is given. Nevertheless, careful use of the classical optimization tools governed by the previous scheme, results in the desired solution in most cases.*

To conclude, it should be noted the analogy with the condition of Delaunay admissibility for an edge as described in Chapter 9.

Delaunay triangulations and non-obtuse meshes. As pointed out, a Delaunay triangulation in two dimensions is not necessarily a non-obtuse triangulation. In fact we have a property of maximization about the minimum angle included in a pair of adjacent triangles and not the opposite property. Nevertheless, a method for point placement can be found in [Chew-1989b] which results in a bound for the angle of the mesh element in the case of a Delaunay strategy for vertex connection. Given a domain, i.e., a polygonal discretization of its boundary, a Delaunay algorithm based on the boundary vertices and using as internal points the circumcenters of the elements allows a mesh where the angles are bounded. Note that when a circumcenter falls outside the domain, the corresponding boundary edge is subdivided by introducing its midpoint.

Following this remark where an upper bound on the angles exists, we could observe that a Delaunay triangulation is not necessarily non-obtuse. Conversely, a non-obtuse triangulation is necessarily a Delaunay triangulation.

Now, we look at what could be the extension to three dimensions of the notion of a non-obtuse triangulation. Before introducing a reasonable characterization, we return to the two-dimensional case. Given a triangulation, if we consider the circles of minimum radius (the smallest circles) that enclose the triangles, then for a non-obtuse triangulation, we have the following properties:

- the smallest circle enclosing a non-obtuse triangle is its circumcircle,
- a non-obtuse triangle is *self-centered*, meaning that its circumcenter falls inside the triangle.

Thus, based on the above property, a triangulation in three dimensions is termed non-obtuse if all of its tets are self-centered.

[11] For the sake of simplicity, we use the term non-obtuse triangle to describe an element whose angles are non-obtuse.

18.5 Strategies for mesh optimization

First we compute the initial quality of the point, the edge, the element or the set of such entities included in the initial configuration. We then compute the same quality for the entity or all the entities related to the solution or the different possible solutions based on a simulation. Finally, we decide to effectively apply the optimization process, as a function of the quality evolution. Several strategies can be chosen to govern the decision. The process is applied:

- if the resulting configuration is strictly improved,
- if the resulting configuration is improved to some extent,
- in the case of multiple possible solutions, by selecting the first valid solution occurring in the simulation or by choosing the best solution among all,
- and so on.

Another issue consists of defining the way in which the operator is used. We can decide to process

- all the mesh entities starting from the first and going to the last,
- only some entities selected *ad-hoc* (using a heap based on a relevant criterion, edge length if edges are to be processed, or using a quality threshold, etc.),
- all the entities, or only some of them, randomly picked,
- and so on.

The question is now to design an automatic and global optimization method by deciding on a strategy for both the choice of the local operators and the order in which to use them (see below), assuming that the strategy related to a given local operator is fixed.

Several observations can be helpful in defining such a strategy. Assuming that a local operator sequence is given, a stopping criterion must first be defined. In fact, several classes of criteria are possible, as indicated below. The process is repeated as long as:

- the mesh is affected by one operation,
- the mesh is affected by one or several operations,
- a given threshold (in terms of quality) has not been achieved,
- and so on.

Once this has been decided, a strategy must be defined. There is some flexibility regarding the possible choices. Indeed, we can

- apply every local operator to all the entities concerned by its application, and then turn to a different operator,

- consider a given mesh entity and apply all the possible local operators before turning to a different entity,
- combine the two above approaches.

It is also possible to classify the pathologies following the degree of optimization that could be expected and to deal with the mesh entities accordingly. In other words, the worst entities are dealt with first.

Remark 18.18 *An immediate question about an optimization process is to know if the optimum has been reached. In practice, the purpose is to improve the mesh and achieving the optimum or not remains a purely theoretical, non-trivial issue. For example, the presence of wells, the fact that the function in optimization is differentiable or not, etc., are parameters that act on the conclusion. From a practical point of view, using some degree of* randomization *in the possible choices may, in most cases, avoid the cases where a well is found. On the other hand, looking for a strict optimum may turn out to be costly and, ultimately, for relatively little gain in efficiency.*

18.6 Computational issues

In this section, we discuss some computational aspects related to the above tools. First, we consider how to construct the balls or shells which, as previously seen, are the local supports of the procedure. We then give some indications about how to develop optimization tools.

Ball construction. For a simplicial mesh, we refer the reader to Chapter 2 where some solutions resulting in a ball construction are described. For other types of meshes, similar methods can be defined.

Shell construction. Again, Chapter 2 presents a method for shell construction in simplicial meshes that can be extended to other mesh types.

Choice of a criterion. As previously indicated, there may be several criteria for quality evaluation. In such a case, it is necessary to decide which criterion to choose. In practice, if we take two different criteria which vary in the same way (i.e., both measure the quality in any case), optimizing one of these automatically leads to optimizing the other.

Computing a criterion. When the simulation of a given optimization tool includes a large number of possible solutions (as is the case when considering the generalized swap for a shell), CPU cost considerations impose the optimization of the "simulation-effective application" pair. The operations used in the simulation are of a purely geometric nature (surface or volume computations, element quality evaluations, before and after the process in question has been applied), while the effective application of the operator leads to defining the new elements and the new neighborhood relationships (if these must be maintained) that can be affected in the process. A careful computer implementation of these two phases enables us to minimize the global CPU cost of the whole process.

In order to reduce the cost it could be noted that some quantities involved in a given optimization process, while part of the global evaluation of the configuration, remain constant during the process (for instance, the external faces of the ball of a given point P are not affected by any relocation of point P. In this example the neighboring relationships are also preserved).

General scheme for a local optimization procedure. Using an optimization operator is quite simple. First, its result is simulated regarding both the validity and the quality evolution of the elements concerned. If an improvement is observed in the simulation phase, the simulated output is retained. When several solutions are possible, the best one is selected (or the first possible solution which has been observed). Thus, instead of computing the full criterion, we could look first at the surface (volume) and if it is negative, there is no point in pursuing the computation. As a consequence, when numerous criteria of this type must be evaluated, we could first compute all the surfaces and, if one of these quantities is wrong, stop the process.

Exercise 18.12 *Return to Table 18.2 and examine how the CPU cost could be minimized.*

18.7 Application examples

In this section, we first indicate how to judge a mesh and then we give some particular examples of mesh optimization.

Mesh appreciation

Mesh analysis is a crucial and difficult point when investigating meshes with a large number of elements, especially in the three-dimensional case. Two complementary approaches can be followed: a graphic visualization and a purely numerical analysis.

Both methods have advantages and drawbacks. First of all, in some cases, using graphic software could be helpful to give some idea of the appearance of the mesh aspect or its quality. Nevertheless, despite the powerful graphic software available, this method of mesh appreciation could be unsuitable when meshes with a large number of elements are considered (the screen is too black) or, simply, for three-dimensional meshes. In addition, the CPU time necessary to display a large mesh could be rather long.

On the other hand, numerical analysis of a mesh must be defined carefully. The aim here is to find one or several pertinent criteria that are easily readable and unambiguously reflect the aspect of the mesh we wish to examine.

Visualization. For efficiency, graphic visualization must offer numerous tools which allow the easy examination of the mesh under consideration. In addition, these tools must be incorporated in a system which must be as user-friendly as possible.

In terms of mesh correctness, a *shrink* applied to the mesh elements is a fast way to detect any overlapping or defaults of connectivity. Nevertheless, it is not so easy to detect a negative surface (volume) element. One possible way to make this check possible is to associate a color with the element, this color being related to an oriented normal.

In terms of quality functions, using colors and cuts (in three dimensions) allows us to display some isocontours of these functions.

Numerical appreciation. The numerical verification of meshes is based on the computation of quantities associated with them. For example, in terms of mesh correctness, it is of interest to be sure that both the surfaces or volumes of elements are all positive and that mesh connectivity is right. In terms of mesh quality, element quality extrema, average element quality and histograms showing the distribution of elements according to their quality give a quick understanding of the mesh under consideration.

Positiveness of element surface areas or volumes is obvious to check by simply computing these quantities. To check whether or not a mesh is correct in terms of connectivity, it is necessary to construct the adjacency graph associated with the mesh (i.e., to establish for every element the list of its neighboring elements) and to verify that this graph is closed[12].

Mesh quality is easy to obtain by computing the quality of all the entities concerned based on the quality function we are interested in. Then various numerical values (extrema, mean value, adequate norms, etc.) as well as histograms of distribution of the analyzed entities according to their quality can be used.

A few examples

We now give an example in two dimensions (for the sake of clarity) that concerns the optimization of a triangular mesh. Figure 18.14 shows the mesh in its initial configuration (left-hand side) and after being optimized (right-hand side). In this case (in two dimensions), a simple view makes it possible to see the efficiency of the optimization process since the mesh includes a reasonable number of elements.

To demonstrate the effect of shape optimization in three dimensions, we observe the distribution of the mesh elements with regard to their quality value. As a visualization has no real interest, Table 18.3 presents various parameters characterizing the mesh before and after optimization. The example given (after [George, Borouchaki-1997]) corresponds to a tet mesh.

Table 18.3 gives, in the first line, the distribution, as a percentage of the total number of elements, of the elements according to their quality and the ranks from

[12]For a non-manifold mesh, a more subtle method must be considered. For instance, an edge on a surface mesh could be shared by more than two elements.

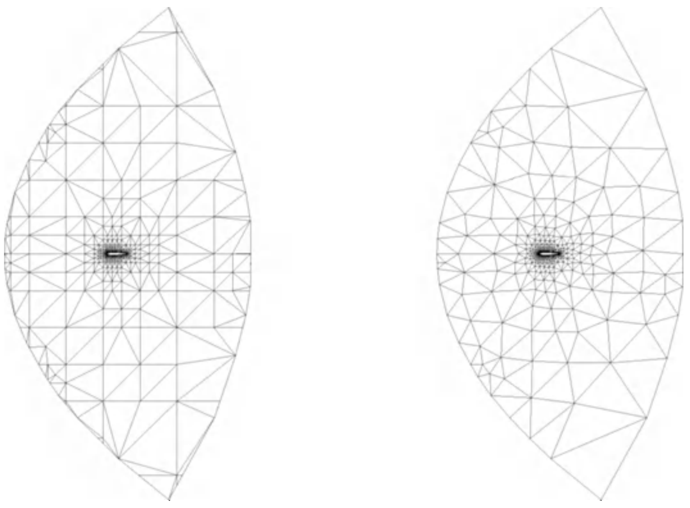


Figure 18.14: *Optimization of a mesh in two dimensions. The brute mesh (quadtree) of the domain (left-hand side) and the optimized mesh (right-hand side).*

1 to 2, from 2 to 3, from 3 to 10 and larger than 10. In the second line, we give the number of corresponding elements. The value denoted as *target* is the quality value of the best possible tet that can be created based on the worst face in the domain boundary. $\mathcal{Q}_{\mathcal{T}}$ is the global quality value of the mesh, i.e., the value of the worst element, *ne* and *np* respectively note the number of elements and the number of vertices in the mesh. The last two lines show the same quantities for the mesh after optimization. A rapid examination of these figures gives an immediate impression of the efficiency of the optimization process.

Q	$1-2$	$2-3$	$3-10$	>10	*target*	$\mathcal{Q}_{\mathcal{T}}$	*ne*	*np*
before	63	21	12	3	8.30	755	3,917	1,004
.	2,475	824	486	132				
after	72	21	6	0	8.30	11.44	3,608	1,015
.	2,620	759	224	5				

Table 18.3: Shape quality of the tet mesh before and after optimization.

Specifically, it can be seen that $\mathcal{Q}_{\mathcal{T}}$ is close to *target* (i.e., in the same range) and that the percentages of elements in the various quality ranges have been changed as desired. Nevertheless, there are still some elements with a relatively poor quality. This fact is generally due to two reasons. The worst quality (the value *target*) is not necessarily attainable if the domain, for instance, is such that moving the point related to the worst face in the boundary is not possible (or the required swaps at some vicinity of this face are not permitted). Moreover, the optimization process, due to its computer implementation in terms of strategy, does not always lead to the optimum (as previously indicated). Note also that the optimization procedures have been applied only on the internal vertices and edges in the mesh, the boundary mesh entities remaining unchanged.

For other examples, when the optimization criterion is no longer the element shape but includes some other aspects (via a metric, for example), we suggest that the reader refers to some other chapters in the book which deal more specifically with this problem.

Chapter 19

Surface Mesh Optimization

In Chapter 18, we described several methods to optimize planar or volumic meshes based on criteria notably related to the shape and the size of the elements. We mentioned that these methods cannot generally be applied directly to surfaces. This is why we now deal with the optimization of surface meshes which, while using the same general principles as the methods described in Chapter 18, nevertheless presents numerous specificities.

Surface meshes play an important role in various numerical applications. Hence, for finite element methods, it is well established that the quality of the geometric approximation may affect the accuracy of the numerical results as well as the convergence of the computational scheme [Ciarlet-1991]. In this type of application, a surface mesh is conceived, in principle, as the description of the boundary of a computational domain in three dimensions (cf. Chapters 5 to 7). Actually, to be useful, these meshes must conform to certain criteria, related to the geometry of the surfaces they represent (we expect an element size variation based on the local curvature) or to the physical behavior of the problems studied (element density greater in regions where the gradient of the solution varies). However, in the last case, following a physical criterion does not mean excluding conformity to the geometric properties of the surface.

Moreover, it is frequent that a given surface mesh is not satisfactory, either because it corresponds to too coarse an approximation of the surface, or because it contains too many elements to be usable. We focus here on the optimization of such a mesh with respect to the geometry it represents, so as to obtain a mesh of a geometric nature (for which the gap between the discretization and the geometry of the surface is bounded by a given tolerance value) and such that the quality (in shape and/or in size) of the triangles and/or the quadrilaterals is acceptable for finite element calculations.

Surface mesh modification and optimization operators need to access certain information about the surface and its properties. This information serves, in particular, to position a point on the surface or to locate a point on the surface, given a point and a direction. If such a surface is known exactly (given an analytical definition, for example) or indirectly (using queries to a geometric modeling system, for example), this information is easily accessible. However, if the input is already a mesh, the surface approached by the polyhedral representation is not known. The given discretization will then be used to construct a geometric support (i.e., a mathematical representation) having adequate regularity and continuity properties. This support can then be queried to obtain the information about the surface required by the optimization process.

Once the intrinsic properties of the surface are known (or at least estimated), it is possible to construct the metric of the tangent plane (cf. Chapter 15). This metric will be used to govern the mesh modification and mesh modification procedures. In particular, the edge lengths are calculated in this metric[1].

★
★ ★

In this chapter, we specify the criteria and the methods used to optimize surface meshes. The first section introduces the shape and size quality measures adapted to surface meshes. In the second section, we indicate how to retrieve, from the given discretization, the intrinsic properties of the surface (radii of curvature, normals, etc.) and thus to construct the metric associated with the tangent planes that will be used to govern optimization algorithms. The third section deals with the problem of constructing a geometric support when the input is a mesh. Mesh optimization operators and algorithms are introduced respectively in the fourth and fifth sections. Finally, several examples of optimized surface meshes are given in the sixth section. Examples of simplified meshes are also proposed to illustrate a particular application of the optimization, the surface mesh simplification.

19.1 Quality measures

In this section, we deal with how to evaluate the quality of surface meshes. This information will be useful during a global surface mesh optimization procedure. We mainly examine the case of meshes composed exclusively of triangles, giving some indications nonetheless about quadrilateral meshes.

Surface mesh quality (classical case)

Recall that (cf. Chapter 18) the shape ratio of a triangle K is defined by the relation:

$$Q_K = \alpha \frac{h_{max}}{\rho_K} = \alpha \frac{h_{max} p_K}{S_K} , \tag{19.1}$$

where h_{max} is the diameter of K, ρ_K is the in-radius, p_K is the half-perimeter and S_K is the area of K. The coefficient α is chosen in such a way that the quality of an equilateral triangle is equal to one.

[1]Which may also be combined with a "physical" metric which is, for instance, representative of the behavior of a solution during a numerical computation.

This formula indicates the degradation of an element K as compared with the equilateral triangle. In practice, we use the inverse value of $\mathcal{Q}_K$, to avoid numerical problems.

Remark 19.1 *Notice that this quality measure makes it possible to appreciate a triangle with no other metric consideration but the geometry.*

This element quality measure allows us to define a more global quality measure, for a whole set of triangles. Hence, the quality of the ball $\mathcal{B}(P)$ of a point P is given by:

$$\mathcal{Q}_{\mathcal{B}(P)} = \max_{K \in \mathcal{B}(P)} \mathcal{Q}_K \,. \tag{19.2}$$

By extension, the quality of a surface mesh $\mathcal{T}$ is defined by the relation:

$$\mathcal{Q}_{\mathcal{T}} = \max_{K \in \mathcal{T}} \mathcal{Q}_K \,. \tag{19.3}$$

We can, similarly, define an average quality value for a mesh, using the relation:

$$\overline{\mathcal{Q}_{\mathcal{T}}} = \frac{1}{ne} \sum_{i=1}^{ne} \mathcal{Q}_{K_i} \,, \tag{19.4}$$

where ne denotes the number of mesh elements.

Surface mesh quality (general case)

If size specifications are given or if a metric map is provided, the previous approach is slightly modified. We now have to decide whether or not the current mesh conforms to these specifications.

Efficiency index. Let l_{AB} be the length of an edge AB in the metric specified. The *efficiency index* allows us to estimate the average deviation of the lengths as compared to 1 (the reference value). More precisely, this index is defined as:

$$\tau = 1 - \frac{1}{na}.\sum_{i=1}^{na}(1 - e_i)^2 \,, \tag{19.5}$$

where na denotes the number of mesh edges and $e_i = l_i$ if $l_i \leq 1$, $e_i = 1/l_i$ if $l_i > 1$.

This measure enables a rapid estimation of the conformity of a mesh with respect to a given (isotropic or anisotropic) size map. In practice, a value $\tau \geq 0.91$ indicates that the mesh respects the specification well. The reader can refer to Table 18.1 (Chapter 18) to appreciate the sensitivity of the index in the isotropic case.

Quality of a triangle (in two dimensions). If the metric map is isotropic, we *naturally* retrieve Relation (19.1). When the metric map is anisotropic, the quality of a triangle, in two dimensions, is then defined by the relation:

$$\mathcal{Q}_K = \max_{1 \leq i \leq 3} \mathcal{Q}_K^i , \tag{19.6}$$

where $\mathcal{Q}_K^i$ represents the quality of triangle K measured in the Euclidean space related to vertex P_i of K. If $(\mathcal{M}_i)_{1 \leq i \leq 3}$ is the metric specified at the vertices P_i of K, we can also write (see the proof in Chapter 18):

$$\mathcal{Q}_K^i = \alpha \frac{\max\limits_{1 \leq j < k \leq 3} \sqrt{{}^t\overrightarrow{P_jP_k}\mathcal{M}_i\overrightarrow{P_jP_k}} \sum\limits_{1 \leq j < k \leq 3} \sqrt{{}^t\overrightarrow{P_jP_k}\mathcal{M}_i\overrightarrow{P_jP_k}}}{|\sqrt{Det(\mathcal{M}_i)}|Det(\overrightarrow{P_1P_2}, \overrightarrow{P_1P_3})} , \tag{19.7}$$

where α is a coefficient of normalization.

This quality measure, defined in the plane (i.e., in two dimensions), can be extended to surface triangles. The metric $\mathcal{M}_i$ at a vertex P_i is then defined in the tangent plane $\Pi(P_i)$ associated with this vertex and the expression of $\mathcal{Q}_K^i$ is modified accordingly [Frey, Borouchaki-1999].

Quality of a surface triangle. Let us consider the vertex P of triangle K. Let $\vec{n}_K$ be the unit normal to the plane of triangle K and let $\vec{n}(P)$ be the unit normal to the surface (at the tangent plane $\Pi(P)$) at P. We note by θ the angle between the vectors $\vec{n}_K$ and $\vec{n}(P)$ and we denote $\tilde{K}$ the image of triangle K by a rotation of angle θ around the axis defined by the vector $\vec{n}_K \wedge \vec{n}(P)$ (cf. Figure 19.1).

By construction, the triangle $\tilde{K}$ belongs to the tangent plane $\Pi(P)$. Its quality $\mathcal{Q}_K$ can then be measured with respect to the metric $\mathcal{M}(P)$ defined in the tangent plane $\Pi(P)$ at vertex P, using Relation (19.7). This comes down to defining the quality $\mathcal{Q}_K$ of a surface triangle as:

$$\mathcal{Q}_K = \max_{P_i \in K} \mathcal{Q}_{\tilde{K}_i}^i . \tag{19.8}$$

By extension, the quality $\mathcal{Q}_{\mathcal{T}}$ of a surface mesh $\mathcal{T}$ and its average quality $\overline{\mathcal{Q}_{\mathcal{T}}}$ are defined, as in the classical case, by the following relations:

$$\mathcal{Q}_{\mathcal{T}} = \max_{K \in \mathcal{T}} \mathcal{Q}_K , \qquad \overline{\mathcal{Q}_{\mathcal{T}}} = \frac{1}{ne} \sum_{i=1}^{ne} \mathcal{Q}_{K_i} . \tag{19.9}$$

Quality of a surface quadrilateral. We have seen (Chapter 18) that in two dimensions, the shape quality $\mathcal{Q}_K$ of a quadrilateral K can be evaluated, in the isotropic case, using Formula (18.10):

$$\mathcal{Q}_K = \alpha \frac{h_{max} \, h_s}{S_{min}} , \tag{19.10}$$

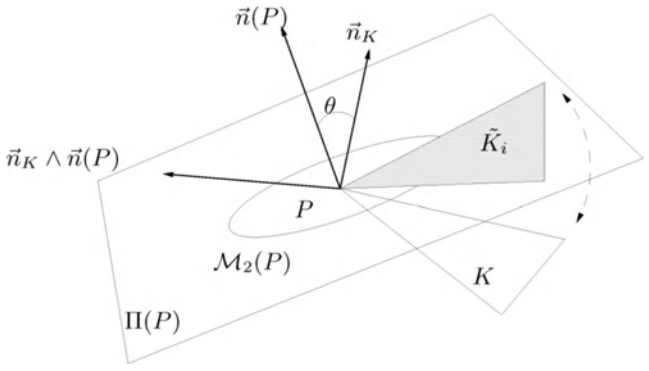


Figure 19.1: *Evaluation of the quality of a surface triangle in the tangent plane* $\Pi(P)$ *associated with the vertex* P. *The metric* $\mathcal{M}_2(P)$ *represents the metric of the tangent plane at* P.

where α is a normalization factor($\alpha = \frac{\sqrt{2}}{8}$), S_{min} is the minimum among the four surfaces that can be associated with K, $h_s = \sqrt{\sum_{i=1}^{4} h_i^2}$ with h_i the length of edge i of K and h_{max} the largest length among the four edges and the two diagonals.

In general (when a metric map has been supplied), an optimal quadrilateral is a quadrilateral having unit length edges and diagonals of lengths $\sqrt{2}$.

We can also use the notion of *roughness* of a quadrilateral. To this end, we consider the two diagonals (AC and BD) of the quadrilateral $ABCD$ and we measure the two dihedral angles so defined. Then, the regularity of $ABCD$ is defined as:

$$\mathcal{P}_{ABCD} = \min_{A,B,C,D}(\mathcal{P}_{AC}, \mathcal{P}_{BD}), \tag{19.11}$$

where

$$\mathcal{P}_{AC} = \frac{1 + \langle \vec{n}_{ABC}, \vec{n}_{ACD}\rangle}{2} \quad \text{and} \quad \mathcal{P}_{BD} = \frac{1 + \langle \vec{n}_{ABD}, \vec{n}_{BCD}\rangle}{2},$$

represent the values of *planarity* of the edges present in the construction (see below the geometric criteria). In these relations, $\vec{n}_K$ denotes the unit normal to the considered triangle K.

In practice, the measure of the regularity of a quadrilateral face will be used to quantify the *torsion* of a tridimensional element.

Quality of the geometric approximation

The quality measures introduced in the previous paragraphs translate numerically the deformation of a triangle and/or its conformity with respect to the size (metric) map specified. However, these measures do not allow[2] us to evaluate the quality

[2]Or only in a indirect way.

of the geometric approximation (of the discretization), which is the way the mesh reflects the geometry of the surface. In particular, it is important to make sure that the gap between the elements and the surface is locally controlled. Indeed, recall the following definition (Chapter 15).

Definition 19.1 *A* geometric mesh *(of type P^1) within a given ε of a surface surface Σ is a piecewise linear discretization of this surface for which the relative gap to Γ is at any point of the order of ε.*

In other words, the problem is then to make sure that a given surface mesh is a geometric mesh for a fixed relative gap ε. To this end, we introduce several measures to evaluate the quality of the geometric approximation of a surface.

Geometric criteria. Several rather simple (almost intuitive) geometric criteria can be used to characterize a surface mesh. These measures are normalized (i.e., range between 0 and 1) according to the usual principle that a value close to 1 indicates that the element (and by extension, the mesh) under consideration is satisfactory with respect to this criterion. These geometric criteria thus behave like quality measures rather than like degradation (deformation) measures of the mesh elements.

We will now specify several criteria such as *planarity*, *deviation* and *roughness* of a surface. These criteria are local measures of the behavior of the surface in the vicinity of a mesh vertex [Frey, Borouchaki-1998].

- Planarity

The geometric discontinuities of a surface are generally expressed by a rapid variation of the directions of the normals to the surface in the neighborhood of a point (or between two adjacent triangles). The ridges and singularities are characteristic examples of C^0 continuity. However, when the surface is supposed to be G^1 continuous in the neighborhood of a point, a rapid variation of the normal to the surface in this neighborhood is most likely an indication that the mesh density in this area is not able to capture the local variations of the surface. To evaluate this lack of density, we introduce the following definition.

Definition 19.2 *The* planarity $\mathcal{P}_P$ *at point P is defined as the maximal angle between the normal $\vec{n}_P$ to the surface at P and the normals $\vec{n}_{P_i}$ at the vertices P_i of the ball of P, other than P:*

$$\mathcal{P}_P = \frac{1}{2}\big(1 + \min_{P_i}\langle \vec{n}_P, \vec{n}_{P_i}\rangle\big)\,. \tag{19.12}$$

According to this principle, we can define the planarity $\mathcal{P}_{AB}$ of an edge AB as follows:

$$\mathcal{P}_{AB} = \frac{1}{2}(1 + \langle \vec{n}_{K_1}, \vec{n}_{K_2}\rangle)\,,$$

where K_1 and K_2 are two triangles sharing the edge AB. Hence, the planarity of an edge is the measure of the dihedral angle between two triangles characterizing the geometric continuity of the surface along AB.

Remark 19.2 *When the planarity value $\mathcal{P}_P$ at P and the minimum of the planarity values $\mathcal{P}_{PP_i}$ at the edges PP_i incident to P are slightly different, the point P is a* singular point.

- Roughness

Considering the set of edges PP_i incident to a point P of the surface, we can define, from the previous measure, the local *roughness* $\mathcal{S}_P$ of the surface at point P as:

$$\mathcal{S}_P = \min_{P_i} \mathcal{P}_{PP_i} \,. \tag{19.13}$$

The degree of roughness[3] of the surface in the neighborhood of point P thus represents the minimal value of the planarity values over the set of edges incident to P.

- Deviation

A variant of the planarity measure at point P consists of evaluating the deviation of the mesh edges with respect to the geometry (i.e., the surface). In other words, we attempt here to evaluate the maximal gap between the edges and the tangent plane $\Pi(P)$ at P. Thus, we suggest the following definition:

Definition 19.3 *The* deviation $\mathcal{D}_P$ *at point P corresponds to the maximal angle between the edges PP_i incident to P and the tangent plane* $\Pi(P)$, *calculated as follows:*

$$\mathcal{D}_P = 1 - \min_i |\langle \vec{n}_P, \vec{u}_i \rangle| \,, \tag{19.14}$$

where $\vec{u}_i$ represents the unit vector supported by the line PP_i and $\vec{n}_P$ is the unit normal vector to the surface at P.

Remark 19.3 *From a practical point of view, the deviation criterion is less accurate than the planarity. Indeed, consider a point P for which the three following relations are satisfied:*

$$\langle \vec{n}_P, \vec{u}_i \rangle = 0 \,, \qquad \langle \vec{n}_{P_i}, \vec{u}_i \rangle = 0 \,, \qquad \langle \vec{n}_P, \vec{n}_{P_i} \rangle = -1 \,.$$

At point P, the deviation of the edges with respect to the tangent plane is judged to be good, while the planarity criterion indicates that the surface is locally badly discretized in the neighborhood of P (Figure 19.2). Such a point P is then considered as a singular point (for which the normal to the surface is not defined).

[3]Notice that the notion of roughness of a piecewise linear interpolation surface (or a Cartesian surface) has been defined as the L^2 norm squared of the gradient of the function f defining the surface, integrated over the triangulation $\mathcal{T}$ [Rippa-1990]:

$$|f|^2_{\mathcal{T}} = \sum_{i=1}^{n} |f|^2_{K_i} \,, \qquad \text{with} \quad |f|^2_{K_i} = \int_{K_i} \left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2 dxdy \,,$$

where K_i denotes a triangle of $\mathcal{T}$ and n is the number of elements of $\mathcal{T}$.

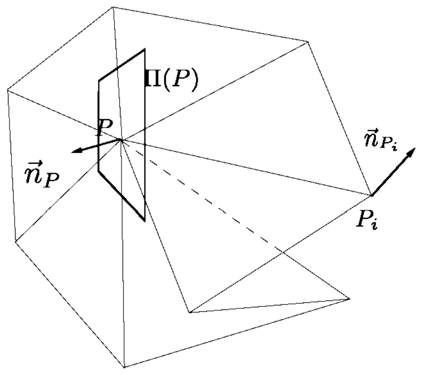


Figure 19.2: *Example for which the variation of the unit normals between two neighboring vertices is better captured by the planarity measure than by the measure of the local deviation at P.*

Notice that these criteria do not involve the map of the geometric metrics which may possibly be supplied. In other words, we consider here isotropic criteria. In the anisotropic case, the surface mesh is geometric if it complies with a geometric metric map, for which the edge lengths are locally proportional to the principal radii of curvature. To verify that a mesh complies with such a map, we introduce the following criterion.

Size quality. Suppose given a (geometric) metric map $\mathcal{M}$, the size quality $\mathcal{L}_{AB}$ of a mesh edge AB is defined by the relation:

$$\mathcal{L}_{AB} = \begin{cases} l_{AB} & \text{if} \quad l_{AB} \leq 1 \\ \dfrac{1}{l_{AB}} & \text{else} \end{cases} , \tag{19.15}$$

where l_{AB} represents the length of edge AB in the metric $\mathcal{M}$ specified (Chapter 10). In other words, if edge AB is parameterized by $t \in [0,1]$:

$$l_{AB} = \int_0^1 \sqrt{{}^t\overrightarrow{AB}\, \mathcal{M}(A + t\overrightarrow{AB})\, \overrightarrow{AB}}dt\,, \tag{19.16}$$

and, when the metric is position independent: $l_{AB} = \sqrt{{}^t\overrightarrow{AB}\, \mathcal{M}\, \overrightarrow{AB}}$.

By extension, the size quality can be defined at a vertex P, depending on the edge PP_i incident to P, in the following manner:

$$\mathcal{L}_P = \min_{P_i} \mathcal{L}_{PP_i}\,. \tag{19.17}$$

Combined criterion. The previous criteria can be combined into a single weighted criterion $\mathcal{C}_P$ defined at a vertex P, for example, in the following way:

$$\mathcal{C}_P = \mathcal{P}_P^{\alpha_1}\, \mathcal{D}_P^{\alpha_2}\, \mathcal{S}_P^{\alpha_3}\, \mathcal{L}_P^{\alpha_4}\,, \tag{19.18}$$

where the coefficients α_i are such that:

$$\sum_{\alpha_i=1}^{4} \alpha_i = 1\,.$$

Remark 19.4 *The context of the application makes it possible to specify the values of these coefficients, depending on their desired relative weights.*

A global measure and an average measure at the mesh level can also be defined as:

$$\mathcal{C}_\mathcal{T} = \min_{P\in\mathcal{T}} \mathcal{C}_P \qquad \text{et} \qquad \overline{\mathcal{C}}_\mathcal{T} = \frac{1}{n}p \sum_{P\in\mathcal{T}} \mathcal{C}_P\,, \tag{19.19}$$

where np is the number of vertices in the mesh $\mathcal{T}$.

We now have a set of criteria allowing us to analyze whether a given mesh is a geometric mesh or not. We will see later that these criteria can also be used to control the mesh modification and mesh optimization operators.

Optimal surface mesh

We have already mentioned that, given $\mathcal{Q}_K$ a quality measure for an element K of a mesh $\mathcal{T}$, the global mesh quality is defined as:

$$\mathcal{Q}_\mathcal{T} = \max_{K\in\mathcal{T}} \mathcal{Q}_K\,.$$

The notion of an optimal mesh theoretically refers to a set of meshes. As for planar or volumic meshes, we could say that the *optimal surface mesh* is that for which the measure considered is optimal. Formally speaking, the optimal surface mesh is that which, simultaneously:

- optimizes the geometric criterion (or the quality function) considered,
- minimizes the number of elements (vertices).

In practice, we are trying to comply as well as possible with the various geometric criteria, the quality measures and the need to minimize the number of elements. As in two dimensions, a surface mesh composed exclusively of triangles must have edge lengths close to 1. Here we again encounter the notion of unit mesh (Chapter 18).

Definition 19.4 *A* unit mesh *is a mesh in which the elements edges are of unit length.*

Given this definition, we can state the following.

Definition 19.5 *A surface mesh composed exclusively of triangles is optimal if it is a unit mesh with respect to the geometric metric map (i.e., proportional to the principal radii of curvature).*

A good way of evaluating the optimality of a surface mesh consists of using the efficiency index previously introduced.

Remark 19.5 *For quadrilaterals, we try to have unit edge lengths and diagonals of length $\sqrt{2}$.*

19.2 Discrete evaluation of surface properties

In the previous section, we have seen that the geometric criteria and the other quality measures involve quantities such as face normals or normals to the surface at mesh vertices. Moreover, we know that a geometric mesh is unit mesh according to the geometric metric map (i.e., the map proportional to the principal radii of curvature). If the mesh is the unique data of the problem, the intrinsic properties of the surface (normals, tangent planes, radii of curvature, etc.) are not known explicitly.

In this section, we first show how to find these properties in an approximate (discrete) way. Then, we indicate how to establish the metric of the tangent plane. Finally, several practical aspects (related to the data structure) are mentioned.

Intrinsic properties

Normal and tangent plane. The tangent plane $\Pi(P)$ at a regular point P of the surface is defined from the unit normal at P. If the unit normal vector $\vec{n}(P)$ is not known, it can be approached as the average (possibly weighted) values of the normals $\vec{n}_{K_i}$ at the triangles incident to P (i.e., the triangles of the ball of P):

$$\vec{n}(P) = \frac{\sum\limits_{K_i} \omega_i \, \vec{n}_{K_i}}{\| \sum\limits_{K_i} \omega_i \, \vec{n}_{K_i} \|} . \tag{19.20}$$

In practice, the weights ω_i associated with the normals at the triangles can be related to various (representative) quantities of the mesh, for example:

- the surfaces $S(K_i)$ of the triangles,
- the inverse of the surfaces $S(K_i)$ of the triangles (to emphasize the smallest elements),
- the angle $\alpha_i(P)$ at P of each triangle K_i,
- etc.

The unit normal vector $\vec{n}(P)$ at P serves to define the tangent plane. In fact, the point P belongs to $\Pi(P)$ and the vector $\vec{n}(P)$ is a vector orthogonal to $\Pi(P)$ (Chapter 11 and Figure 19.3).

Remark 19.6 *When the point P is a singular point, the tangent plane is not defined. If P is along a ridge, we can define a normal on each side of the edge. To this end, we consider the two open balls at P (limited by the ridge) and we calculate a unit normal vector using Formula (19.20).*

Remark 19.7 *Recall that for parametric surfaces (Chapter 11), the tangent plane is directed by the two tangent vectors $\vec{\tau}_1(P)$ and $\vec{\tau}_2(P)$:*

$$\vec{n}(P) = \frac{\vec{\tau}_1(P) \wedge \vec{\tau}_2(P)}{\|\vec{\tau}_1(P) \wedge \vec{\tau}_2(P)\|} .$$

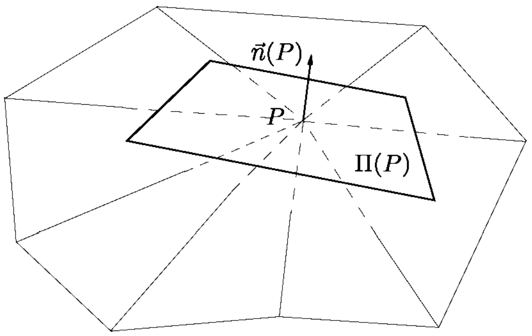


Figure 19.3: *Unit normal vector at P and associated tangent plane.*

We will now examine how to determine the curvatures and the principal directions of curvature at the vertices of a given mesh.

Principal curvatures (summary). At a point P, which is assumed to be regular, of the mesh, the normal $\vec{n}(P)$ is calculated using Formula (19.20). Let us consider the ball $\mathcal{B}(P)$ of point P. The edges PP_i, for each $P_i \in \mathcal{B}(P)$, are assumed to be traced on the surface.

Given a tangent vector $\vec{\tau}(P) \in \Pi(P)$ to the surface at P, there exists a curve Γ traced on the surface admitting $\vec{\tau}(P)$ as tangent. If $C(P)$ is the curvature of Γ at P, we have seen that the normal curvature[4] $\kappa_n(\vec{\tau}(P))$ of Γ at P is defined as (Chapter 11):

$$\kappa_n(\vec{\tau}(P)) = C(P)\,\cos\alpha\,, \tag{19.21}$$

where α is the angle between $\vec{n}(P)$ and $\vec{\nu}(P)$: $\cos\alpha = \langle \vec{\nu}(P), \vec{n}(P) \rangle$, $\vec{\nu}(P)$ being the unit normal vector to Γ at P (Figure 19.4).

To find the principal curvatures, we use Meusnier's theorem. We know that all curves traced on the surface and having the same tangent vector at P have the same curvature $\kappa(P) = |\kappa(\vec{\tau}(P))|$ at P. We then consider a particular curve, the normal section (corresponding to the intersection of a plane defined by the vectors $\vec{\tau}(P)$ and $\vec{n}(P)$). For such a curve, the vectors $\vec{n}(P)$ and $\vec{\nu}(P)$ are collinear. Indeed, recall that if Γ is the intersection of the surface Σ by a normal section, the *Meusnier's circle* of diameter κ_n is the geometric locus of the points P_i endpoints of the segments PP_i such that $\|\overrightarrow{PP_i}\| = C(P)$, the curvature of a curve whose normal $\vec{\nu}$ forms an angle α with the unit normal $\vec{n}$ to Σ at P.

By definition, an infinity of normal sections exist around P. Among these, let us consider the two sections (orthogonal together) whose normal curvatures are respectively minimal and maximal. These two curvatures are the *principal curvatures* at P, denoted $\kappa_1(P)$ and $\kappa_2(P)$, and the associated directions are the *principal directions* of unit vectors $\vec{\tau}_1(P)$ and $\vec{\tau}_2(P)$.

[4]Notice that the sign of $C(P)$ changes with the orientation of $\vec{n}(P)$.

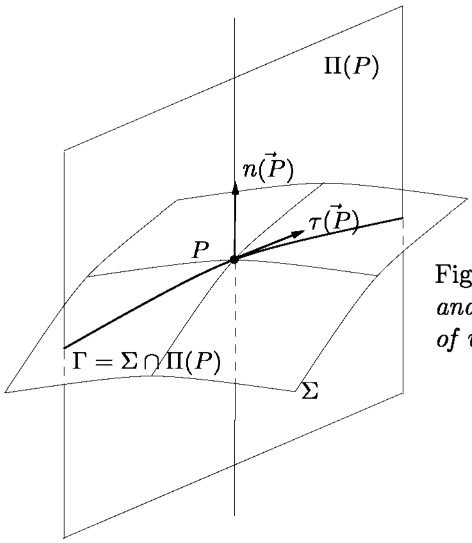


Figure 19.4: *A normal section at P and the curve Γ traced on the surface of vector director $\vec{\tau}(P)$.*

Thus, we propose a way of evaluating the mean curvature and the principal curvatures at any vertex of a given mesh. More precisely, we will calculate the minimal radius of curvature and the principal radii of curvature that will serve later to define the geometric metric at any mesh vertex P.

Calculation of the minimal radius of curvature (isotropic case). Let $\mathcal{T}$ be a surface mesh representing locally the geometry of the surface. Let us then consider $\mathcal{B}(P)$ the ball of a vertex P (supposed regular). It seems natural to consider an edge PP_i of $\mathcal{B}(P)$ as the discretization of a curve Γ_i traced on the surface and belonging to a normal section.

Let Γ_i be such a curve, of normal parameterization $\gamma_i(s)$. Using a Taylor expansion at order 2 of $\gamma_i(s)$ at $P = \gamma_i(s_0)$, we can write:

$$\gamma_i(s_0 + \Delta s) = \gamma_i(s_0) + \Delta s \vec{\tau}(s_0) + \frac{\Delta s^2}{2\,\rho_i(s_0)} \vec{\nu}(s_0) + \mathcal{O}(\Delta s^3)\,, \tag{19.22}$$

where Δs represents a small increment of s. The *osculating circle* $C_i(P)$ to γ_i at P is the circle of radius $\rho_i(s_0)$ and its center O_i is given by:

$$O_i = P + \rho_i(s_0)\,\vec{\nu}(s_0)\,.$$

Consider then the point P_i defined by:

$$P_i = P + \Delta s \vec{\tau}(s_0) + \frac{\Delta s^2}{2\,\rho_i(s_0)} \vec{\nu}(s_0)\,,$$

that is the point of the parabola whose tangent is $\vec{\tau}(s_0)$ and which is at a distance Δs from $\gamma(s0)$. This point is thus very close to the osculating circle of the curve

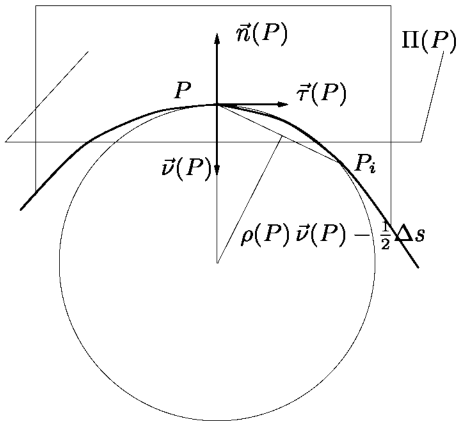
(Chapter 14). This allows us to write (Figure 19.5, left-hand side):

$$\left\langle \overrightarrow{PP_i}\,,\, \frac{1}{2}\overrightarrow{PP_i} - \rho_i^* \vec{\nu}(s_0) \right\rangle = 0\,,$$

considering a circle of radius ρ_i^* passing through P and P_i.

According to the previous remark (relative to the proximity of P_i to the osculating circle), we can then approach $\rho_i(s_0)$ by ρ_i^* (for a sufficiently small value of Δs, refer to the discussion in Chapter 11). The circle of radius ρ_i^* is called the *approximate osculating circle* and we deduce the value of the radius of curvature:

$$\rho_i^*(s_0) = \frac{\langle \overrightarrow{PP_i}, \overrightarrow{PP_i} \rangle}{2\,\langle \vec{\nu}(s_0), \overrightarrow{PP_i} \rangle}\,. \tag{19.23}$$



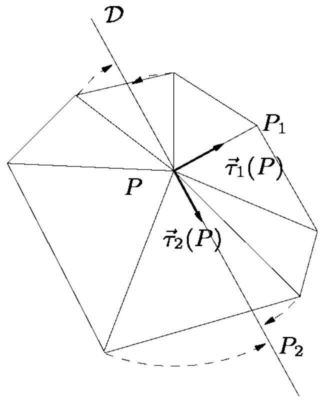


Figure 19.5: *Approximation of the osculating circle at point P (left-hand side). Trivial determination of the principal radii of curvature and the principal directions of curvature (right-hand side).*

In other words, when the point of abscissis $(s_0 + \Delta s)$ is approached by the point P_i, the approximate radius of curvature ρ_i^* is a good (accurate) approximation of the radius of curvature at P.

This result now makes it possible to define the minimal radius of curvature $\rho(P)$ at P by the relation:

$$\rho(P) = \min_{PP_i} \rho_i^*\,. \tag{19.24}$$

This value will allow us to define the isotropic metric (i.e., the size here) at any vertex P of the mesh (see below).

Calculation of the principal radii of curvature ("naive" approach). The calculation of the minimal radius of curvature at a regular mesh vertex P using the previous approach can be slightly modified, in order to find the principal radii of curvature at P.

In fact, reconsider the previous reasoning about the edges PP_i incident to P. It is easy to find the edge PP_1 corresponding to the minimal radius of curvature, $\rho_1(P)$. Hence, the unit vector $\vec{\tau}_1(P)$ supported by the edge PP_j gives the direction of maximal curvature. To determine the maximal radius $\rho_2(P)$, we proceed as follows (Figure 19.5, right-hand side):

- calculate the line $\mathcal{D}$ orthogonal to $\vec{\tau}_1(P)$,
- then position (in distance) the points P_i of $\mathcal{B}(P)$ (the ball of P) on $\mathcal{D}$,
- identify the point P_2 (on $\mathcal{D}$) such that $\overrightarrow{PP_j}$ is maximal,
- we obtain $\vec{\tau}_2(P) = \frac{\overrightarrow{PP_2}}{||\overrightarrow{PP_2}||}$, the unit vector indicating the direction of minimal curvature.

Remark 19.8 *An alternative consists of considering the maximal radius of curvature first and then of applying the above procedure to find the minimal radius of curvature.*

Remark 19.9 *This calculation, although very simple in principle, leads in practice to some problems. The fact of fixing one of the directions to find the other makes this procedure strongly dependent on the given mesh. If the given mesh is, for example, a triangulation obtained by refinement of a regular grid, the directions of curvature can be "shifted" as compared to the true directions (Figure 19.6).*

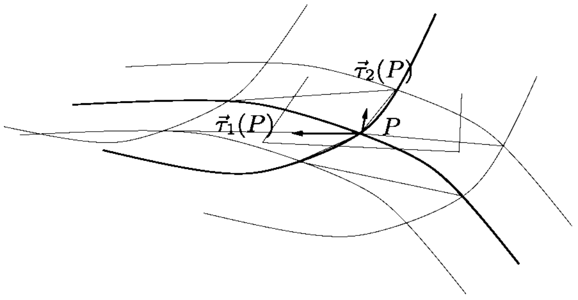


Figure 19.6: *Particular case where the calculation of the principal directions of curvature is distorted by the discretization.*

For this reason, we now indicate a more accurate approach to calculating the principal directions of curvature.

Calculation of the principal radii of curvature (approach 2). Various approaches have been proposed. Let us mention, in particular, one based on a least square approximation formula of *Dupin's indicatrix* suggested, among others, by [Todd, McLeod-1986] and [Chen, Schmitt-1992]. However, to be valid, this calculation supposes that the vertices around the vertex P considered can be matched together two by two.

Dupin's indicatrix of the surface at P, which allows us to locally approximate the surface by a paraboloid, corresponding to a Taylor expansion at the order 2 is defined by the equation:

$$\kappa_1 x^2 + \kappa_2 y^2 = 1\,, \tag{19.25}$$

where $x = \sqrt{\rho}\cos\theta$ and $y = \sqrt{\rho}\sin\theta$ for a point $M(x,y)$ of the tangent plane at P (in polar coordinates, function of ρ and θ). This equation is that of a conic section, each point M in the direction θ is located at a distance $\sqrt{\rho}$ from P. If the point P is an elliptical point, the Dupin's indicatrix is an ellipse, if P is hyperbolic, the indicatrix is formed by two branches of hyperbolas (Chapter 11).

Another possible approach consists of using a least square approximation of the directions of the tangents and the normal curvatures to calculate the principal directions and curvatures. To this end, we calculate for each edge PP_i incident to P a normal curvature, based on the technique previously described. We thus obtain on $\mathcal{B}(P)$ a set of directions (tangents) and of normal curvatures. The normal curvature $\kappa_n(P)$ can be expressed in a quadratic form (Chapter 11). Let $[P, \vec{\tau}_1, \vec{\tau}_2]$ be an orthonormal basis of the tangent plane $\Pi(P)$, any vector $\vec{\tau}$ can be written as a linear combination $\vec{\tau} = \lambda\vec{\tau}_1 + \mu\vec{\tau}_2$. Then, the normal curvature in the direction $\vec{\tau}$ can be written as:

$$\kappa_n(\vec{\tau}) = (\lambda\ \ \mu)\begin{pmatrix} \kappa_n(\vec{\tau}_1) & \kappa_n^{12} \\ \kappa_n^{21} & \kappa_n(\vec{\tau}_2) \end{pmatrix}\begin{pmatrix} \lambda \\ \mu \end{pmatrix}, \tag{19.26}$$

where $\kappa_n^{12} = \kappa_n^{21}$. As the matrix is symmetric, the vectors $\vec{\tau}_1$ and $\vec{\tau}_2$ can be chosen so as to diagonalize the matrix: $\kappa_n^{12} = \kappa_n^{21} = 0$.

Let us write the normal curvature as follows:

$$\kappa_n(\vec{\tau}) = (\tau_x\ \ \tau_y)\ {}^t\mathcal{D}\begin{pmatrix} \kappa_1 & 0 \\ 0 & \kappa_2 \end{pmatrix}\mathcal{D}\begin{pmatrix} \tau_x \\ \tau_y \end{pmatrix},$$

where τ_x and τ_y denote the components of $\vec{\tau}(P)$ in the basis and $\mathcal{D}$ corresponds to the principal directions at P. We can express $\mathcal{D}$ according to only one of the principal directions (the two vectors being orthogonal):

$$\mathcal{D} = \begin{pmatrix} \tau_{1,x} & \tau_{1,y} \\ -\tau_{1,y} & \tau_{1,x} \end{pmatrix}.$$

Repeating the same process for the m edges PP_i incident to P, we finally obtain a system of m linear equations [Moreton-1992]:

$$\begin{pmatrix} u_{1,x}^2 & u_{1,x}u_{1,y} & u_{1,y}^2 \\ .. & .. & .. \\ u_{m,x}^2 & u_{m,x}u_{m,y} & u_{m,y}^2 \end{pmatrix}\begin{pmatrix} \tau_{1,x}^2\kappa_1 + \tau_{1,y}^2\kappa_2 \\ 2\tau_{1,x}\tau_{1,y}(\kappa_1 - \kappa_2) \\ \tau_{1,x}^2\kappa_2 + \tau_{1,y}^2\kappa_1 \end{pmatrix} = \begin{pmatrix} \kappa_{n,0} \\ .. \\ \kappa_{n,m} \end{pmatrix}, \tag{19.27}$$

where the unknowns are obviously the expressions depending on the principal curvatures and directions. This system is of the form $AX = B$.

Remark 19.10 *Notice, however, that this system depends on the mesh (i.e., of the number of edges incident to P). Hence, some of the equations may be redundant (think in particular of the pairs of opposite edges). It is thus important to identify and remove such equations.*

Remark 19.11 *Another particular case corresponds to the configuration where only three edges are incident to P. The system is then fully determined (3 equations and 3 unknowns).*

To resolve such a system (usually over-determined), we proceed as follows. Satisfying all the equations simultaneously is (in principle) not possible, hence we look for the best possible compromise, for example in the least squares sense (i.e., the sum of the squares of the distances between the left and the right members is minimal). The general formula to approximate the least squares solution consists of obtaining a well determined system (n equations and n unknowns) by multiplying by ${}^tA$ and by introducing $\bar{X}$ the approximate solution of X:

$$({}^tA\,A)\,\bar{X} = {}^tA\,B\,.$$

Once this system has been solved[5] (i.e., the values of $\bar{X}$ are known) and according to the relation $\tau_{1,x}^2 + \tau_{1,y}^2 = 1$, we then have the system of four equations with four unknowns to solve:

$$\left\{\begin{array}{lcl} \tau_{1,x}^2\kappa_1 + \tau_{1,y}^2\kappa_2 & = & \bar{x}_0 \\ 2\tau_{1,x}\tau_{1,y}(\kappa_1 - \kappa_2) & = & \bar{x}_1 \\ \tau_{1,x}^2\kappa_2 + \tau_{1,y}^2\kappa_1 & = & \bar{x}_2 \\ \tau_{1,x}^2 + \tau_{1,y}^2 & = & 1 \end{array}\right. . \tag{19.28}$$

The expressions of the principal curvatures are:

$$\kappa_1 = \frac{\bar{x}_1 + \bar{x}_3 - \sqrt{(\bar{x}_3 - \bar{x}_1)^2 + \bar{x}_2^2)}}{2} \text{ and } \kappa_2 = \frac{\bar{x}_1 + \bar{x}_3 + \sqrt{(\bar{x}_3 - \bar{x}_1)^2 + \bar{x}_2^2}}{2} \tag{19.29}$$

and the components of the vector $\vec{\tau_1}$ corresponding to the first principal direction are:

$$\tau_{1,x} = \sqrt{\frac{1}{2}\left(\frac{\bar{x}_1 - \bar{x}_3}{\kappa_1 - \kappa_2} + 1\right)} \text{ and } \tau_{1,y} = \sqrt{\frac{1}{2}\left(1 - \frac{\bar{x}_1 - \bar{x}_3}{\kappa_1 - \kappa_2}\right)}, \tag{19.30}$$

the unit vector $\vec{\tau_2}$ being orthogonal to $\vec{\tau_1}$.

Exercise 19.1 *Retrieve the two expressions of the principal curvatures and directions from Relations (19.28).*

An extension of this approach consists of constraining each pair of opposite edges incident to a vertex P to join with a continuity of order G^2. The pairs of edges are G^1 continuous when they share the same tangent vectors. G^2 continuity means that the curves defined by the pairs of opposite edges share the same binormal vector [Shirman, Séquin-1991], [Moreton-1992].

[5]Using a classical method of linear algebra (see for instance [Golub, VanLoan-1983]).

Calculation of the principal curvatures (approach 3). Finally, to conclude this discussion about the calculation of the principal curvatures and directions, we present here a third approach (suggested by [Hamann-1993], among others).

The basic idea is to go back to a problem of parametric surface, for which the calculation of the principal curvatures can be performed analytically (using the fundamental forms; see Chapter 11). To this end, we start by searching for an interpolation surface passing through P and approaching at best the endpoints of the edges PP_i incident to P. More precisely, this scheme consists of:

- finding the ball $\mathcal{B}(P)$ of P,
- calculating the projections of the $P_i \in \mathcal{B}(P)\,, p_i \neq P$ on the tangent plane $\Pi(P)$ at P,
- considering the projections as abscissas and the distances $||\overrightarrow{PP_i}||$ as ordinates in a local frame in $\Pi(P)$,
- constructing a polynomial of approximation (a quadric, for example) for these points in the local frame,
- calculating the principal curvatures according to this polynomial.

Remark 19.12 *Notice that with this approach, we also find a (over-determined) system of linear equations (depending on the number of incident edges).*

Let $\sigma(u, v)$ be the polynomial of approximation, the surface of approximation is then defined as the set of points $(u, v, \sigma(u, v)) \in \mathbb{R}^3$. The principal curvatures κ_1 and κ_2 of the polynomial:

$$\sigma(u,v) = \frac{1}{2}(c_{2,0}u^2 + 2c_{1,1}uv + c_{0,2}v^2)\,,$$

at point $(0, 0, \sigma(0, 0))$ are given by the roots of the equation:

$$\kappa^2 - (c_{2,0} + c_{0,2})\kappa + c_{2,0}c_{0,2} - c_{1,1}^2 = 0\,. \tag{19.31}$$

Remark 19.13 *Variants of this approach consist of using cubical splines as approximation functions [Wang, Liang-1989] or Beta-splines [Tanaka* et al. *1990], for example, or even more simply of considering the circles circumscribed to the triangles (PP_iP_j) [Chen, Schmitt-1992].*

Metric of the tangent plane

Once the principal curvatures and directions have been calculated using one of the methods described previously, the metric $\mathcal{M}_3$ of the tangent plane can be constructed at each vertex P of the given mesh. We have already seen that the control of the gap between an edge and the surface for a given value ε fixed can be obtained via the metric $\mathcal{M}_3$, the so-called geometric metric (Chapter 15).

As we consider edges as curves traced on the surface, this leads to the matrix $\mathcal{M}_2(P)$, which is the trace of the matrix $\mathcal{M}_3(P)$ on the tangent plane $\Pi(P)$ at P:

$$\mathcal{M}_2(P) = \mathcal{M}_3(P) \cap \Pi(P) .$$

In other words, at any point P, the metric $\mathcal{M}_2(P)$ is the metric *induced* by $\mathcal{M}_3(P)$ on the tangent plane $\Pi(P)$ to the surface at P (Chapter 10).

Remark 19.14 *The metric $\mathcal{M}_2$ is only defined in the tangent planes associated with the vertices.*

It is obvious that the choice of the metric induces the nature of the mesh to obtain. In principle, we are looking for a metric having the general form:

$$\mathcal{M}_3(P) = \begin{pmatrix} a_P & b_P & c_P \\ b_P & d_P & e_P \\ c_P & e_P & f_P \end{pmatrix} . \tag{19.32}$$

Hence, we consider the following *geometric* metrics (i.e., in the tangent plane):

- (anisotropic) metric of the principal curvatures (or metric of the principal radii of curvature):

$$\mathcal{G}_2(P)_{\kappa_1,\kappa_2} = {}^t\mathcal{D}(P) \begin{pmatrix} \dfrac{\kappa_1^2(P)}{\alpha_\varepsilon^2} & 0 \\ 0 & \dfrac{\kappa_2^2(P)}{\beta_\varepsilon^2} \end{pmatrix} \mathcal{D}(P) , \tag{19.33}$$

  where $\mathcal{D}(P)$ is a matrix corresponding to the principal directions $\vec{\tau}_1$ and $\vec{\tau}_2$ at P, where α_ε (resp. β_ε) is a coefficient allowing an anisotropic (relative) control of the gap ε to the geometry (Chapter 13), which takes into account the principal directions and curvatures;

- (isotropic) metric of the minimal radius of curvature:

$$\mathcal{G}_2(P)_\rho = \begin{pmatrix} \dfrac{1}{h^2(P)} & 0 \\ 0 & \dfrac{1}{h^2(P)} \end{pmatrix} , \tag{19.34}$$

  where ρ is the minimal radius of curvature and the variable $h(P) = \alpha\rho(P)$ depends on the position and α is an adequate coefficient, related to the geometric approximation (i.e., to the gap between the edges of the discretization and the surface);

- anisotropic (resp. isotropic) *physico-geometric* metric. Here we consider the metric $\mathcal{G}_2(P)_{\kappa_1,\kappa_2}$ (resp. $\mathcal{G}_2(P)_\rho$) intersected by an arbitrary field of metrics (for example, supplied after a calculation).

Exercise 19.2 *Retrieve and justify the value of the coefficients α and β used in Relation (19.33).*

In practice, we could also consider the isotropic map $\mathcal{M}_3(P)$ of the intrinsic sizes.

Definition 19.6 *The* intrinsic size *at any vertex $P \in \mathcal{T}$ corresponds to an average value (possibly weighted) of the Euclidean lengths of the edges PP_i incident to P.*

The average value corresponds to the solution of a minimization problem (between the desired size and the edge length). Actually, the problem is to find $h(P)$ for any vertex P so as to minimize the function:

$$H(P) = \sum_{P_i} (h(P) - \|\overrightarrow{PP_i}\|)^2 ,$$

where the P_is denote the points of the ball of P but P.

Remark 19.15 *When the size $h(P)$ is, at any point P, smaller than or equal to $\rho(P)$, the metric $\mathcal{M}_3(P)$ thus defined is a geometric metric.*

Once these metrics have been defined, the problem arises of evaluating the edge lengths with respect to the geometric metric.

Calculation of the edge lengths in $\mathcal{M}_2$. At a given point P, the length of an edge PP_i can be approximated using Formula (19.16) by considering the point $P^* = PP_i^* \cap \mathcal{M}_2(P)$ where P_i^* is the projection of point P_i in the tangent plane $\Pi(P)$ (Figure 19.7).

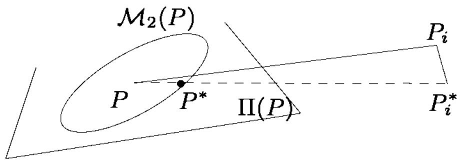


Figure 19.7: *Calculation of the length of edge PP_i in the metric $\mathcal{M}_2(P)$ associated with the tangent plane $\Pi(P)$ at P.*

From these results, we now have a theoretical framework to evaluate the conformity of the mesh with respect to a given metric $\mathcal{M}_3$.

Mesh conformity. We aim to decide whether a given mesh conforms to a geometric metric specification $\mathcal{G}_3$ defined at any mesh vertex.

Definition 19.7 *A mesh $\mathcal{T}$ conforms to a given metric map $\mathcal{M}_3$ (or more precisely to its restriction $\mathcal{M}_2$ in the tangent planes) if and only if:*

$$\frac{1}{\sqrt{2}} \leq l_{AB} \leq \sqrt{2}, \quad \forall AB \in \mathcal{T}. \tag{19.35}$$

When the metric map $\mathcal{M}_2$ considered is the map $\mathcal{G}_{\kappa_1,\kappa_2}$ (resp. $\mathcal{G}_\rho$), the mesh is an anisotropic (resp. isotropic) geometric mesh. Such a mesh is called a $\mathcal{G}$-mesh.

Remark 19.16 *When the metric map $\mathcal{G}_2$ is isotropic, the following assertions are equivalent:*

$$\mathcal{T} \text{ conforms to } \mathcal{G}_2 \iff \mathcal{T} \text{ conforms to } \mathcal{G}_3 .$$

Combination of metrics. When a metric map other than that corresponding to the geometric metrics (intrinsic map) is supplied, it is interesting to combine these two maps. To this end, we consider the metric map corresponding to the intersection of metrics $\mathcal{H}_2(P) = \mathcal{G}_2(P) \cap \mathcal{M}_2(P)$ in the tangent planes associated with the mesh vertices (Chapter 10).

Remark 19.17 *By definition, the map $\mathcal{H}_2$ is a geometric map.*

Computational aspects

We will close this section by noticing briefly several practical aspects related to the evaluation of the intrinsic properties of surface meshes.

Data structure. The approaches described above involve the ball of a mesh vertex P. It is thus important to use an efficient internal data structure that allows us to retrieve the elements of the ball of a vertex easily.

For a triangular surface mesh, it is convenient to use a topological data structure such as those introduced in Chapter 2. The triangles are considered as oriented triples[6] to which are associated (at most) three edge neighbors. We could also use a richer representation, using a connection matrix for each triangle.

The case of mixed meshes (composed of triangles and/or quadrilaterals) requires more care in the definition and the management of the data structure.

Identification of singular points. For meshes representing real surfaces (i.e., the boundaries of a real domain), we need to carefully identify the singularities (corners, ridges, etc.) of the model. If the points of C^1 or G^1 discontinuity are not explicitly provided (for instance, supplied by a geometric modeling system), we must carry out a pre-processing stage on the data to extract this information.

Hence, for example, a *ridge*[7] could be identified as a mesh edge such that the dihedral angle between the adjacent faces along the edge is larger than a given threshold. A *corner* is a mesh vertex where three (or more) ridges are incident or a vertex such that two incident ridges form a very acute angle.

In addition to the singular points, a certain number of entities (points, edges, faces) can be imposed, the *required* or *constrained* entities. These entities must be present in the resulting mesh (see [George, Borouchaki-1997] for a description of a data structure allowing this type of entity to be specified).

[6]This is not a restriction, the surfaces concerned being supposed orientable.
[7]Also called *crest line*.

19.3 Constructing a geometric support

When the sole data available is a surface mesh, the construction of a geometric support makes it possible to define, internally to the procedure, a geometry. The latter is actually a mathematical representation of a surface, the given discretization then being supposed to be an approximation (at the second order) of it. This support must, at least, interpolate the vertices and the normals at the mesh vertices[8], in order to satisfy the conditions of continuity of order G^1, except at the points of discontinuity (which are supposed explicitly known).

The problem is then to construct (to invent, so to speak) a surface composed of order G^1, from the given surface mesh, each triangle serving to define a patch. To this end, two adjacent patches must necessarily have the same tangent plane along their common edge, if the latter is not a singularity of the surface (i.e., is not a ridge).

Classical approaches

Several approaches have been proposed to construct a geometric support globally of class G^1 from a piecewise triangular representation of a surface. In principle, in all these approaches, each triangle serves as support of a patch (cf. Chapter 13).

The methods suggested by [Farin-1986] and [Piper-1987] seem well-suited to dealing with the problem of defining a geometric support. They consist of subdividing each triangle into three triangles and in defining on each newly created triangle a polynomial patch of degree 4 (quartic), such that the continuity of the transverse tangent planes along each boundary edge is ensured, on the one hand between the new triangles resulting from the subdivision of the original triangle and, on the other hand, between the couples of new triangles issued from the subdivision of two adjacent triangles.

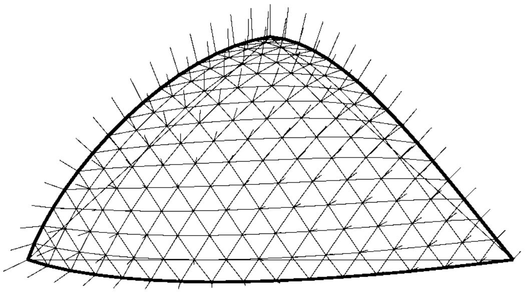


Figure 19.8: *Walton's patch of order G^1 associated with a surface triangle.*

This technique, however, requires rather large memory resources. Actually, after subdivision, 28 control points (9 of them common to the adjacent triangles)

[8] *A priori*, as the initial mesh is a sufficiently accurate geometric approximation of the surface, this geometric support reasonably "emulates" the role of the geometric modeling system.

are associated with each initial mesh triangle. Moreover, each triangle leads to the definition of three patches. The definition of these patches greatly depends on the shape quality of the triangle support (thus, if the latter is badly shaped, for example too stretched, the current method presents some instability in the patch definition).

Modified approach and Walton's patch

More recently, the method proposed by Walton and Meek [Walton, Meek-1996], presents the advantage, on the one hand, of explicitly taking into account the geometric specifications (notably the normals to the surface at the vertices) and, on the other hand, of being relatively simple in its formulation and less memory consuming than the previous approaches.

Basically, this method consists of defining a network of boundary curves for the patches, as well as the related transverse tangent planes, independently of one another, using an interpolation of the normals to the surface at the vertices. Each patch is then defined independently, from its boundary, by taking the specifications related to its boundary (the tangent planes) into account using Gregory's approach [Gregory-1974] (Chapter 13).

The specificity of this method lies in the definition of the network of curves from the sole data of the normals at the vertices (each curve and the related transverse tangent plane is completely defined from the normals at its endpoints). Each boundary curve represents a cubic polynomial whose principal normals (to the curve) are coincident with the normals to the surface at the endpoints. The transverse tangent planes (to the boundary curves) are generated from the tangent vectors at any point along these curves and from vectors resulting from a quadric interpolation of the binormals at the endpoints. Hence, the sole specification of the normals at the two endpoints of an edge is sufficient to define a boundary curve of a patch (based on the endpoints) as well as the transverse tangent plane to the surface along the curve. Finally, from the transverse tangent planes of any triangle, a Gregory patch is generated using a classical approach. Each triangle is thus associated with a patch defined analytically using a rational function in the interior and a quartic along its boundary. This method requires, for each triangle, only 9 control points common to the adjacent triangles.

Remark 19.18 *The scheme suggested by Walton-Meek can be extended to the case of C^0 discontinuities present in "realistic" geometries (ridges, corners, etc.).*

Constructing the geometric support

Here we briefly recall the construction principle for a surface composed of patches globally of class G^1 from an initial (geometric) surface mesh.

The construction scheme comprises the following stages:

- associate with each mesh edge AB a curved segment of degree 3 passing through A and B, while making sure that the principal normals in A and B are collinear to the unit normals to the surface in A and B,

- define the tangent plane generated by the vectors tangent to the curve and to the binormal and associated with the boundary curve,
- raise the degree of the boundary curves so as to construct a polynomial surface of degree 4 (Gregory) over each triangle.

The surface thus defined is of order G^1. This is related to the unique (unambiguous) definition of the tangents (from the normals to the edges), so as to ensure the desired transition between the patches.

Remark 19.19 *For domains presenting C^0 discontinuities, the geometric support is constructed as previously, except at the entities of discontinuity where the tangents are defined in order to obtain a C^0 continuous surface.*

Using the geometric support

As mentioned previously and to conclude on this topic, let us recall that the geometric support is used to answer the following queries:

- given a point and a direction, find the closest surface point,
- return the normal and the minimal radius of curvature at a surface point.

This support thus represents an analytical definition of the surface comparable to that generated by a modeling system (CAD). This support enables us to know the position of the closest surface point from a given point and a specified direction. A second essential requisite concerns the geometric specifications of the surface in the vicinity of these points, a fundamental requirement for any local topological mesh modification (see below). This query provides two types of information, depending on the surface characteristics in the neighborhood of the point considered.

If the point presents (at least) a continuity of order C^2 (in practice, a continuity of order G^1, a tangent plane continuity, is sufficient), the normal to the surface and the minimal of the principal radii of curvature at this point is the sole information required. If the point presents a discontinuity of the tangent planes (the case of points located on a ridge), the two normals to the surface as well as the two minimals of the principal radii of curvature on both sides of the ridge and the tangent to this edge at this point are supplied. This information is returned by the modeling system together with the location of the point considered.

19.4 Optimization operators

As we have seen in Chapter 18, numerous specific tools have been developed for mesh optimization. Similarly, for surface meshes, these operators can be classified into two categories, either of topological or geometric nature. The first ones preserve the mesh connectivity but modify the positions of the mesh vertices and are thus, in some way, related to the geometry of the mesh (a point should be located on the surface). The main operation is the node relation. The second type affect the mesh topology, that is, the mesh connectivity (i.e., the connections between vertices, cf. Chapter 1) and are thus related to the geometry in a different way (for instance, an edge flip must preserve the quality of the geometric approximation). Among these operators, we can mention the edge swap and the entity (vertex or edge) deletion, for example.

In the following sections, we will describe these operators in detail, in a particular case of surface meshes.

Geometric optimization

The basic principle of a geometric optimization operator consists of moving the vertices (by modifying their coordinates) while making sure that these vertices remain (to a certain extent that will be specified later) on the surface.

Node relocation algorithms are usually based on the local notion of vertex ball (the union of all elements sharing a vertex P). The ball of a vertex can be *closed*, for an internal vertex, or *open*, for a boundary vertex. For the sake of convenience, we will only study here the case of closed balls.

Node relocation. A trivial node relocation process consists of moving the vertex P under consideration to the barycenter of the positions of the n vertices P_i of the ball of P. This can be formally written (Chapter 18):

$$P = \frac{1}{n}\sum_{i=1}^{n} P_i \,. \tag{19.36}$$

Notice however that after this modification, the new position of the vertex P is no longer on the surface. Hence, the geometric support must be used to move the point back to the surface, that is to modify its coordinates again . A variation consists of weighting the previous barycentrage, Relation (18.21).

As for two and three dimensional meshes, a more efficient technique consists of introducing a relaxation. The role of the relaxation is to avoid moving the point too far from its original position, which is considered correct (geometrically speaking) during the iterations. Hence, we consider the point P^* defined as the P in Relation (19.36):

$$P^* = \frac{1}{n}\sum_{i=1}^{n} P_i \,,$$

and the relaxation, of parameter ω, then consists of calculating (as in Chapter 18) the new position of the point P as:

$$P = (1-\omega)P + \omega P^* \,. \tag{19.37}$$

In practice, we can choose a sub-relaxation, for example, by taking a value $\omega = 0.5$.

Smoothing techniques. As for two and three-dimensional meshes, smoothing techniques can be applied to surface meshes. We indicate here two particular techniques, based on element quality or edge lengths.

- Smoothing based on quality

In this approach, we consider the ball of a given vertex P. With each external edge f_i of the ball (i.e., each edge not connected to P) is associated an ideal point P_i^* such that the element formed by f_i and P_i^* has an optimal shape quality. The smoothing process is then defined as:

$$P^* = \frac{\sum\limits_{i=1}^{n} \alpha_i P_i^*}{\sum\limits_{i=1}^{n} \alpha_i}, \tag{19.38}$$

where the coefficients α_i can be chosen in several ways (cf. Chapter 18).

Remark 19.20 *This method is also applicable in the case of anisotropic meshes, provided the corresponding notion of quality is used.*

- Smoothing based on edge lengths

The idea consists this time of defining the position of the optimal points P_i^* of the ball of P, so as to define internal edges of unit length. The unit length is related to the size (or metric) map specified. Hence, we obtain a relation like:

$$P_i^* = P_i + \frac{\overrightarrow{P_iP}}{l_{\mathcal{M}}(P_i, P)}, \tag{19.39}$$

where $l_{\mathcal{M}}(P_i, P)$ represents the length of edge P_iP in the metric $\mathcal{M}$ associated with P_iP.

Notice that in practice, regardless of the approach taken, we want to move the point P in the associated tangent plane $\Pi(P)$. To this end, we consider the projections f_i' of the external edges f_i of the ball in the tangent plane $\Pi(P)$ and we look for the position of the associated optimal point using any of the above approaches.

Constrained node relocation. Smoothing and node relocation techniques described previously make it possible to find the position of an optimal point (in the tangent plane associated with the point P to be moved). However, it is necessary to validate this position before accepting the move. In particular, the point P^* is accepted if:

- the geometric approximation associated with the ball of the new point is better than that related to the initial configuration. This can be expressed as:

$$\min\langle \vec{n}_{K_j}, \vec{\nu}(P_i)\rangle \leq \min\langle \vec{n}_{K_j'}, \vec{\nu}(P_i^*)\rangle \,,$$

- the shape quality of the elements of the ball of P^*, $\mathcal{B}(P^*)$, is better than that of $\mathcal{B}(P)$,
- the lengths of the internal edges of $\mathcal{B}(P^*)$ conform to the specified metric:

$$\frac{\sqrt{2}}{2} \le l_{P^*P_i} \le \sqrt{2}\,,$$

where $\vec{n}_{K_j}$ and $\vec{\nu}(P_i)$ denote respectively the normals to the triangles K_j and to the vertices P_j of the ball of P.

For a given vertex P, the node relocation consists of redefining the elements incident to this vertex. To this end, from each edge on the boundary of this configuration, we define an optimal element, thus a point, that belongs to the plane supporting the element of the configuration containing the edge. Then, we determine the barycenter of all these points to get an ideal point. The node relocation then consists of moving step by step the vertex toward the ideal point (considering the projections on the surface) if the previous criteria are satisfied.

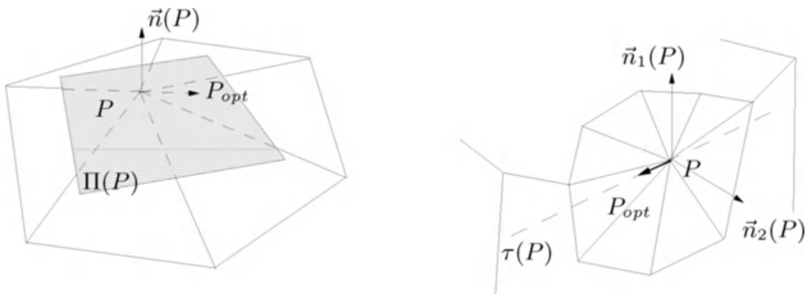


Figure 19.9: *Node relocation for a surface mesh. Left-hand side: node relocation for a G^1 vertex. Right-hand side: node relocation when the vertex is located in a discontinuity of the surface (ridge).*

Remark 19.21 *If the point P considered is located on a ridge (C^0 discontinuity), the relocation procedure is more tedious. Actually, we must consider open balls around P. In fact, the geometric constraints are slightly changed and the checks must be performed in each open ball (Figure 19.9, right-hand side).*

Edge splitting. The subdivision of an edge PQ is an operation that consists of inserting points along the edge, so as to split the edge into unit length segments in the metric specified. We retrieve here the problem of splitting a segment, already discussed in Chapter 14.

In practice, we replace this general procedure by a simpler one consisting in inserting the midpoint M of the edge PQ in the current mesh. More precisely, the midpoint of the edge is created (on the straight segment) and then mapped on the surface using the geometric support. This process involves replacing the two

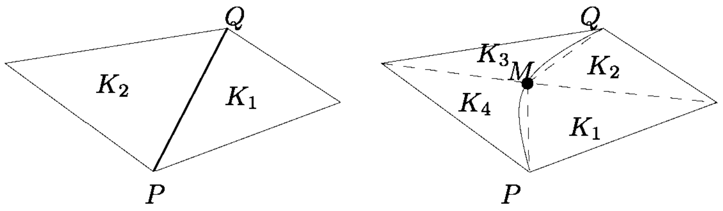


Figure 19.10: *Edge splitting by adding a point on the surface. The two triangles sharing the initial edge (left-hand side) are replaced by four triangles sharing the two newly created sub-segments (right-hand side).*

triangles sharing the initial edge by four new triangles sharing two by two the two sub-segments created after the insertion of the midpoint (Figure 19.10).

Specific geometric conditions must be satisfied for the subdivision to be applied:

- the geometric approximation of the new configuration must be better than (or equal to) that of the initial configuration,
- the lengths of the edges AB of the newly created triangles must conform (or at least be close to unit length) to the metric specified:

$$\frac{\sqrt{2}}{2} \leq l_{AB} \leq \sqrt{2},$$

- the shape quality of the final configuration must be improved (as compared with that of the initial configuration).

Remark 19.22 *Notice that the edge splitting operation is usually combined with edge swapping operations (see below), to form a more complex operator. In this case, the last geometric constraint can be slightly relaxed.*

Topological optimization

We now examine the different operators that make it possible to modify a mesh while preserving the vertex positions.

Edge swapping. Edge swapping is a simple operation consisting in replacing the edge common to two adjacent triangles by the *alternate edge* (i.e., the diagonal linking the vertices not connected by the initial edge). In principle, this operation is similar to edge swapping in two dimensions. However, we will see that additional constraints are applied in the case of surface meshes.

In two dimensions, edge swapping is possible as soon as the polygon formed by the union of the two triangles is convex. For surfaces, we will examine if the configuration of the pair of alternative triangles (after swapping) is *optimal* (in a sense that will be specified) with respect to the geometric approximation and to the element shape quality compared with the original configuration.

Remark 19.23 *Notice than an edge can be swapped if and only if it is not constrained (it is not a ridge, for example).*

For surfaces, verifying that the initial configuration is convex is only meaningful if the two triangles are exactly in the same plane (which is a rather particular case). However, according to the shape quality measures introduced previously, we can express several conditions that must be checked and satisfied in order to apply the edge swapping.

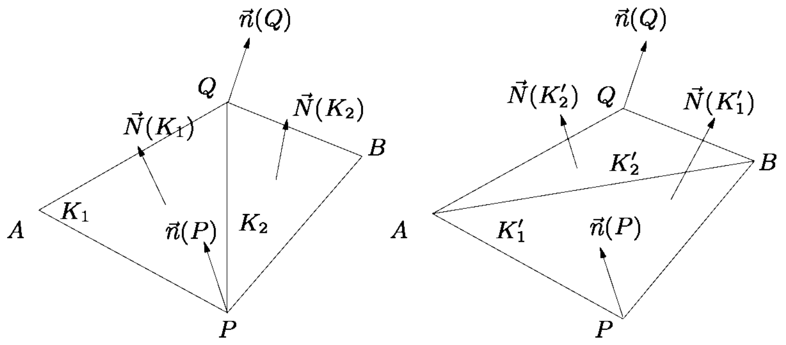


Figure 19.11: *Edge swapping on the surface, initial configuration (left-hand side) and alternative configuration (right-hand side).*

Let $K_1 = APQ$ and $K_2 = BQP$ be two adjacent triangles sharing the edge PQ (Figure 19.11) and let $K_1' = APB$ and $K_2' = BQA$ be the triangles of the alternate configuration. We denote by $\vec{N}(K_i)$ the normal to the triangle K_i and by $\vec{n}(P_i)$ the normal to the vertex P_i. The edge PQ can be changed into the edge AB if and only if:

- the (dihedral) angle between faces K_j' of the alternative configuration does not exceed the angle limit of a ridge (C^0 discontinuity). In other words, the surface must not be "*folded*" locally,
- the geometric approximation (of the underlying surface) of the alternative configuration is better than (or identical to) that of the initial configuration. This can be expressed by the relation:

$$\min\langle \vec{N}(K_j), \vec{n}(P_i)\rangle \leq \min\langle \vec{N}(K_j'), \vec{n}(P_i)\rangle\,, \quad 1 \leq i \leq 3, 1 \leq j \leq 2\,,$$

- the shape quality of the triangles of the alternative configuration is better than that of the original configuration:

$$\min Q_{K_j'} < \min Q_{K_j}\,, \quad 1 \leq j \leq 2\,,$$

- the length l_{AB} of the edge AB conforms to the specified metric:

$$\frac{\sqrt{2}}{2} \leq l_{AB} \leq \sqrt{2}\,,$$

- the edge AB does not already exist in the mesh[9]. We can use a hash table (Chapter 2) to avoid this problem (Figure 19.12, left-hand side). A similar case (Figure 19.12, right-hand side) can be detected using the geometric criteria, the new face after swapping be almost coplanar with two (or more) existing faces.

Here, the order of the checks is arbitrary. In practice, the user must choose an order that favors, as much as possible, a quick reject (a condition not satisfied).

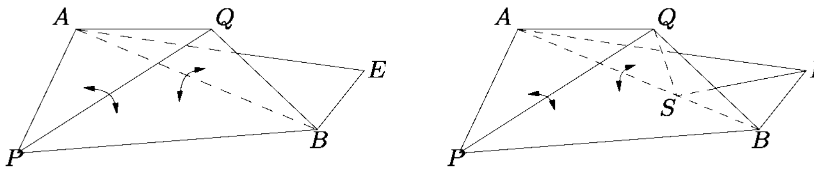


Figure 19.12: *Particular cases where the edge swapping PQ is not possible: the alternative edge AB already exists (left-hand side), the face ABQ and the faces ASQ, BSQ are almost coplanar (right-hand side).*

Edge deletion (vertex removal). This operator consists of retriangulating the ball of the vertex P to be deleted. In two dimensions, this operation comes down to triangulating a star-shaped polygon (with respect to the vertex P), without an internal point. To this end, several methods have been proposed. We suggest here a method that presents the advantage of being applicable to surface meshes. We recall first the principle of the method in two dimensions.

The basic principle of the operation aims at reducing the degree (i.e., the number of incident edges; see Chapter 18) of the vertex to be deleted to the value 3 or 4 (in the degenerate case). The cavity can then be retriangulated trivially (Figure 19.13).

The process involves applying to each edge incident to P an edge swapping, if possible (if the orientation of the resulting triangles is preserved). Schematically, the reduction algorithm can be written as follows:

- apply edge swappings to each edge incident to P,
- iterate the process, if a modification has been carried out.

This process converges in principle toward a configuration of 3 or 4 edges incident to the vertex to be deleted (the second configuration representing a quadrilateral having two orthogonal diagonals).

The application of this algorithm to geometric surface meshes requires a minimum of attention. In two dimensions, the edge swapping is applied if the triangles of the alternative configuration are valid. For surfaces, the alternative configuration must be validated by the conditions described in the previous paragraph. We can, however, relax a little this constraint by requiring that each newly created

[9]A situation which is impossible with planar or volumic meshes.

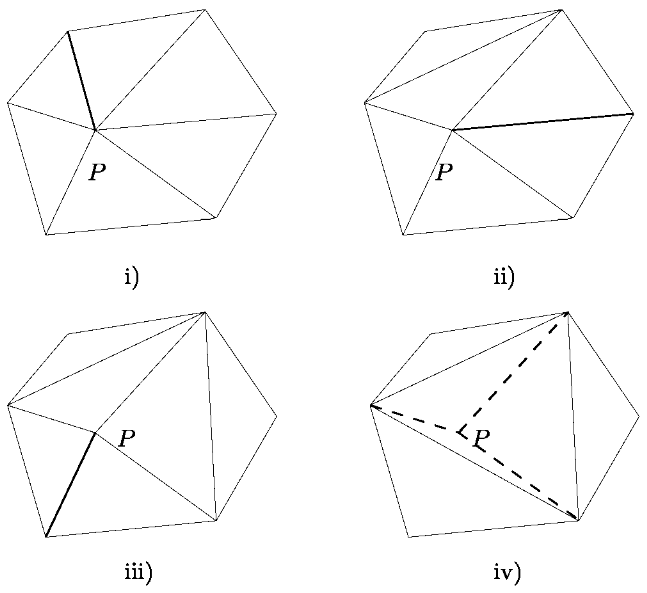


Figure 19.13: *Edge deletion by vertex removal. Edge swappings are applied iteratively (steps i) to iv)) to reduce the degree of the vertex P to 3. The three triangles sharing this vertex can then be removed.*

triangle of the cavity not enclosing the vertex to be removed (hence a triangle that will be part of the final configuration) must be close to the surface. This is equivalent to imposing that the gap (the angle) between the normal to the triangle and the normals to the surface at the three vertices is bounded by a given tolerance value. We can also impose a condition regarding the shape quality of the triangles formed.

The deletion operator being defined, the convergence of the procedure is no longer ensured. In practice, this means that if an intermediate configuration is blocked, the deletion cannot be carried out and the initial situation must be restored. This leads to extra (useless) work.

For these reasons, we favor another edge deletion operator consisting in merging the two edge endpoints into a single point.

Identification (merging) of the edge endpoints. Given an edge PQ, we try to replace this edge by a single point. To this end, we consider two operators that reduce the edge either into one of its endpoints (cf. Figure 19.14), or into a new point (cf. Figure 19.15).

- Identification on one of the endpoints.

This operation can be seen as a specific re-writing (remeshing) of the ball of the vertex merged. The final configuration being known, it can thus be validated *a priori*, based on the geometric constraints. This is indeed a specific remeshing as all possible configurations are not analyzed systematically to retriangulate the vertex ball (see Chapter 18).

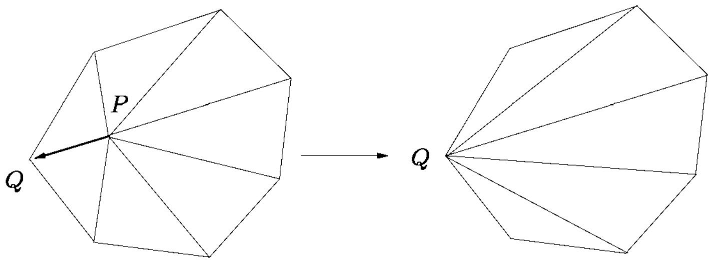


Figure 19.14: *Edge deletion by identifying its two endpoints (the vertex P is merged with the vertex Q).*

In practice, it is sufficient to substitute the vertex Q to the vertex P in all triangles of the ball (without the two triangles sharing the edge PQ) and then to verify if the following conditions are satisfied:

- the shape quality of the new triangles is acceptable,
- the length l_{QP_i} of any edge of the triangles in the final configuration conforms to the specified metric,
- the geometric approximation of the final configuration is controlled:

$$\min\langle \vec{N}(K_j), \vec{n}(P_i)\rangle < \tau \,,$$

for each triangle K_j of vertices $P_i, i = 1,3$. For an isotropic mesh, for example, the value τ represents the maximum gap allowed, $\tau = \cos\theta$, where θ is such that $\alpha = 2\sin\theta$, $\alpha\rho(P)$ being the size imposed locally by the metric map at each point P (depending on the minimal radius of curvature $\rho(P)$), cf. Chapter 15).

Remark 19.24 *We are not trying here to strictly improve the shape quality of the initial ball. In practice, if $\mathcal{Q}_P$ denotes the quality of the initial ball and $\mathcal{Q}$ the quality of the new mesh, we want to have $\mathcal{Q} \geq 0.6\mathcal{Q}_P$ (the coefficient 0.6 is chosen so that a configuration of six equilateral triangles can be retriangulated into four triangles).*

- Edge reduction.

The reduction of an edge into a new vertex is an operation similar to the previous one, although it is more costly. Actually, in this case, two balls must be retriangulated (the ball of P and that of Q). Moreover, the position of the new vertex M must be supplied by the geometric support associated with the mesh. The geometric and shape quality requirements are identical to those used in the identification of the edge endpoints.

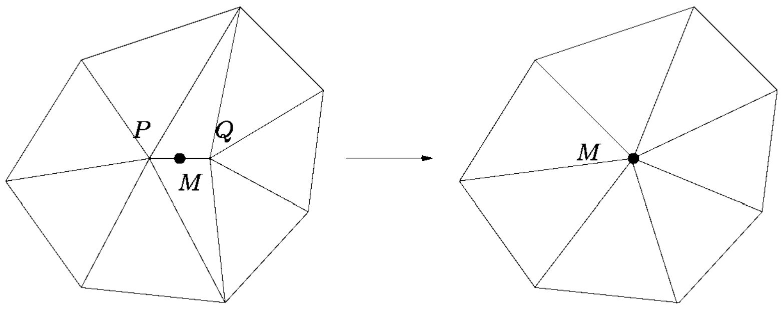


Figure 19.15: *Edge deletion by reduction. The vertices P and Q are merged into a new vertex M.*

Relaxation of the degree of a point. Recall that the degree of a vertex is the number of edges incident to this vertex. We have seen in Chapter 18 that in two dimensions, 6 is the optimal value for the degree of a mesh vertex. This value is also optimal for surface meshes.

The aim of the procedures for relaxing the degree of a point is to get rid of the *under-connected* and *over-connected* vertices (vertices for which the degree is respectively lesser than or greater than the target value). To this end, the process consists of iteratively applying edge swappings in order to reach the target value.

In the following section, we describe several techniques used for surface mesh optimization.

19.5 Optimization methods

The strategies carried out to optimize surface meshes are very close to those used for classical meshes in two and three dimensions.

The initial values of the quality measures relative to the mesh or to a particular set E of triangles (for example, a vertex ball or a pair of adjacent triangles) of the initial mesh are calculated and serve to initialize the optimization procedure. The same quantities are evaluated for each new configuration E' corresponding to a topological or geometric operation. The operator is applied if the optimization criteria are satisfied. In particular, the modification is carried out if the resulting configuration strictly improves the measure chosen ($\mathcal{Q}_{E'} > \mathcal{Q}_E$), or if the resulting configuration improves the measures by a certain coefficient ($\mathcal{Q}_{E'} > \alpha \mathcal{Q}_E$).

Among the various possible configurations, the one leading to the best (resp. first) optimization is picked. All these choices can also depend on the order in which the mesh entities are considered. Hence, it is possible to process the elements:

- in a natural order (i.e., based on the increasing or decreasing indices),
- according to a specific relation of order (for example, sorted according to an increasing order of the edge lengths),

- of a certain type exclusively (for example, all triangles of a connected component, all boundary points, etc.),
- according to any other criterion, including a random order.

Shape optimization (classical case)

In the classical case, the surface mesh is optimized exclusively with respect to the element shape quality. From the initial mesh, a geometric support G^1 continuous is constructed to represent the underlying surface. This support is used notably to find the position of a point on the surface.

From the algorithmic point of view, the current mesh elements are iteratively analyzed and those whose quality exceeds a certain tolerance are removed. The deletion of badly-shaped elements is performed using the topological and geometric mesh modification tools.

In principle, a heuristic procedure is used which consists of simulating the application of an operator on the badly-shaped configuration and, based on the simulated results, in carrying out an optimization procedure on the mesh. Notice that it may sometimes be useful to degrade locally the mesh quality. Hence, for example, it may be interesting to merge two closely spaced points (event if the quality of the resulting configuration is worse than that of the initial one) before applying edge swapping to improve the shape quality of the new configuration.

Size optimization (isotropic and anisotropic case)

When one (or several) metric map(s) is (are) supplied, the optimization problem is a little more tedious. We first try to get back to a one size map problem, the geometric metric $\mathcal{G}_3$. Hence, the required metric intersections are carried out (Chapter 10).

Remark 19.25 *If the metric map supplied is not geometric, we calculate the metric map $\mathcal{G}_3$ using one of the approaches described in this chapter.*

The problem boils down to evaluating and then modifying a given mesh $\mathcal{T}$ according to a metric map $\mathcal{M}_3$. In practice, we use the induced metric $\mathcal{M}_2$, in other words, we consider the trace $\mathcal{M}_2(P)$ of $\mathcal{M}_3(P)$ in the tangent plane associated with each mesh vertex P.

The aim of size optimization for a surface mesh is to obtain a unit mesh with respect to the metric map $\mathcal{M}_2$. This approach is (obviously) based on the edge lengths analysis. Hence, mesh optimization involves applying the following process iteratively on the current mesh edges:

```
REPEAT
   FOR ALL edges $AB$ in $\mathcal{T}$
      calculation of $l_{AB}$
      IF $l_{AB} < \frac{1}{\sqrt{2}}$
         deletion of $AB$
      ELSE IF $l_{AB} > \sqrt{2}$
```

```
          subdivide $AB$ into $n$ sub-segments ($n$ depending of $l_{AB}$)
       END IF
    END FOR
    FOR ALL mesh vertices $P$
       node smoothing, coupled with
       local edge swappings
    END FOR
WHILE the current mesh is modified.
```

Remark 19.26 *The edges are, generally, processed in a random order. Thus, the edge splitting or vertex deletion operators are carried out alternatively.*

Remark 19.27 *A patch of the geometric support is associated with each newly created point so as to improve the further searching and localization procedures.*

Remark 19.28 *Finally, notice that it is more interesting (for practical considerations) to subdivide an edge using its midpoint, rather than inserting n points along this edge.*

Optimal mesh

Let us simply recall here that we aim at getting an optimal surface mesh, that is a mesh $\mathcal{T}$ such that:

$$\frac{1}{\sqrt{2}} \leq l_{AB} \leq \sqrt{2}\,, \quad \forall AB \in \mathcal{T}\,. \tag{19.40}$$

The above procedure (size optimization) aims at creating edges that satisfy this criterion.

19.6 Application examples

To conclude, we now provide several examples of optimized isotropic surface meshes.

Surface mesh optimization examples

Table 19.1 presents some information related to the surface meshes. For the sake of clarity, we denote respectively by $\mathcal{T}_i^0$ the initial mesh, by $\mathcal{T}_i^g$ a geometric mesh (with respect to the map of the minimal radius of curvature) and by $\mathcal{T}_i^s$ a simplified geometric mesh. In this table, *np* and *nt* denote respectively the number of vertices and triangles in the mesh, ε is the relative distance to the surface allowed (in degrees; see Chapter 15) and the quantities $\overline{\mathcal{P}_T}$, $\overline{\mathcal{D}_T}$, $\overline{\mathcal{R}_T}$ and $\overline{\mathcal{L}_T}$ correspond respectively to the average values of the geometric criteria of planarity, deviation, roughness and edge sizes in the meshes. Finally, the value $\overline{\mathcal{Q}_T}$ represents the shape quality of the mesh (i.e., the quality of the worst element).

Figure 19.16 shows an example of initial surface mesh, i). From this mesh, the edges separating the faces of C^0 continuity and the singular points have been

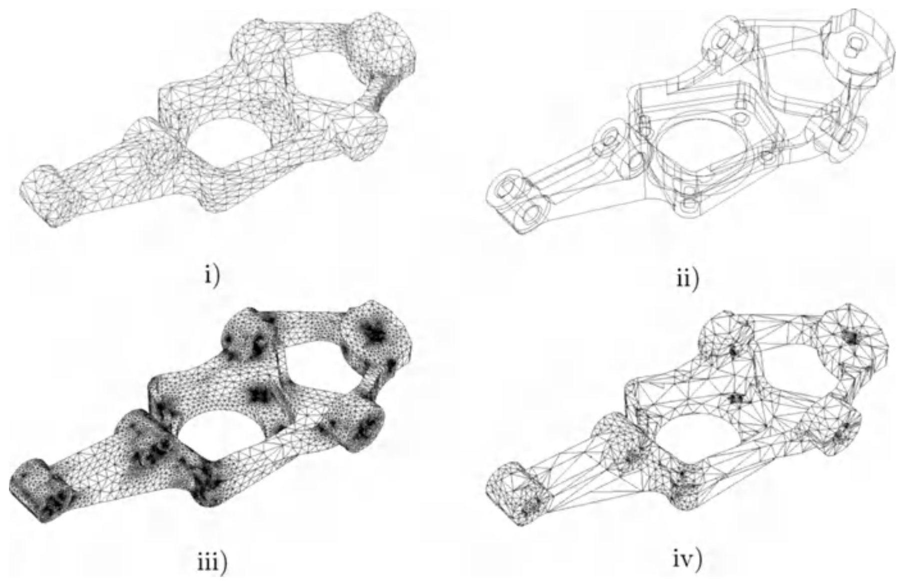


Figure 19.16: *Optimization of a surface mesh. i) initial surface mesh (data courtesy of MacNeal-Schwendler Corp.) ii) identification of the ridges, G^1 discontinuities and constrained entities, iii) enriched geometric mesh (metric correction $\varepsilon = 1.5$, surface deviation 5 degrees) iv) simplified geometric mesh (surface deviation 14 degrees, without metric correction).*

identified, ii). Two meshes can then be generated from these specifications: an enriched and optimized mesh, iii) and a simplified mesh, iv). From the initial mesh, a surface composed of patches globally of class G^1 has been constructed thus defining the geometric support. This has been used to find the position of the vertices in the enriched mesh, obtained by subdividing the initial mesh edges. A simplified mesh has been extracted from the enriched mesh (see below).

Figure 19.17 depictes another example where the intial surface mesh comes from a reconstruction method (see Chapter 15) and is optimized with regard to geometric properties (isotropy, gradation and element shape while preserving the geometry).

Mesh simplification

A particular and interesting application of optimization concerns the geometric simplification of surface meshes. When the surface meshes have a number of elements that is too large to be used (for example, in numerical simulations or for visualization purposes; see below), it is desirable to reduce the number of elements, while preserving (to some extent) the geometric approximation of the surface. This is the aim of mesh *simplification*[10] algorithms.

[10]Also called *decimation*, *coarsening* or *reduction*.

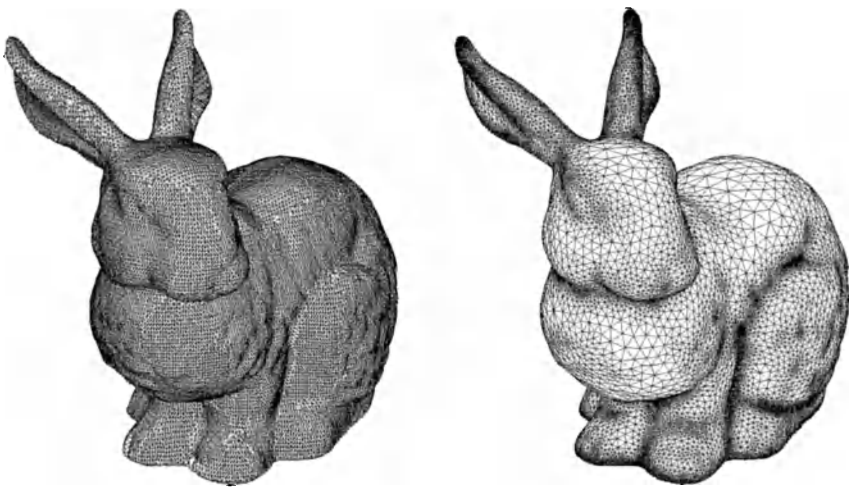


Figure 19.17: *Original mesh reconstructed from scanned data (Computer Science Department, Stanford University) and corresponding isotropic geometric mesh.*

meshes	np	nt	ε	$\overline{\mathcal{P}_T}$	$\overline{\mathcal{D}_T}$	$\overline{\mathcal{R}_T}$	$\overline{\mathcal{L}_T}$	$\overline{\mathcal{Q}_T}$
$\mathcal{T}_1^0$	1,685	3,370	-	0.96	0.88	0.97	0.71	0.70
$\mathcal{T}_1^g$	36,329	72,658	0.01	0.99	0.96	0.99	0.95	0.86
$\mathcal{T}_1^s$	3,823	7,646	0.03	0.98	0.87	0.98	0.92	0.64
$\mathcal{T}_2^0$	67,106	134,212	-	0.80	0.57	0.75	0.72	0.46
$\mathcal{T}_2^g$	64,048	128,096	0.01	1.00	0.94	1.00	0.94	0.83
$\mathcal{T}_2^s$	11,814	23,628	0.02	0.98	0.88	0.99	0.96	0.84
$\mathcal{T}_3^0$	43,575	87,150	-	0.98	0.92	0.98	0.56	0.25
$\mathcal{T}_3^g$	77,291	154,582	0.02	0.98	0.87	0.99	0.95	0.84
$\mathcal{T}_3^s$	17,240	34,480	0.05	0.97	0.85	0.98	0.95	0.83

Table 19.1: Statistics related to various surface meshes.

Basic principle. In principle, mesh simplification is an operation of the same nature as optimization, in that it involves the same operators. In practice, given a (geometric) surface mesh that is assumed to comply with a given size map $\mathcal{G}_2$, the simplification aims at creating an optimal surface mesh with respect to a modified size map, corresponding to a greater surface approximation.

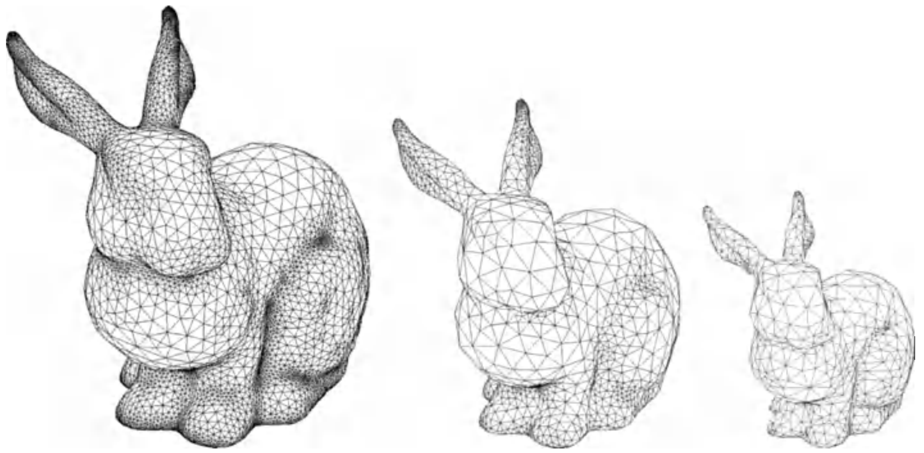


Figure 19.18: *Geometric simplifications of a surface mesh, corresponding to deviations to the surface* $\varepsilon = 0.03$, $\varepsilon = 0.1$ *and* $\varepsilon = 0.29$. *Notice that as we move further from the object (as simulated here), the visual aspect remains very close to the original one.*

The first stage consists of calculating a geometric metric map $\widetilde{\mathcal{G}_2}$ from the map $\mathcal{G}_2$ for a greater tolerance ε than the original one. Then, a classical optimization procedure described previously is applied to obtain a simplified surface mesh.

The key to the method lies in respecting the following rules (see, for example, [Frey, Borouchaki-1998]):

- the control of the distance to the geometry, *the simplified map is a geometric map*,
- the resulting mesh is optimal with respect to this modified map,
- the shape quality of the final mesh elements is controlled.

Remark 19.29 *Notice however that a vertex where the minimal radius of curvature* ρ *is smaller than the threshold* ε *(within a coefficient) can be simplified. This idea makes it possible to remove details that are judged useless (at the given resolution) and, therefore, to construct simplified geometric meshes, within* ε.

Actually, given the previous remark, it is easy to note that mesh simplification consists of working in a "strip" of a given width (obviously related to the specified tolerance value). The modifications are applied when the edges concerned by the operation remain in this strip.

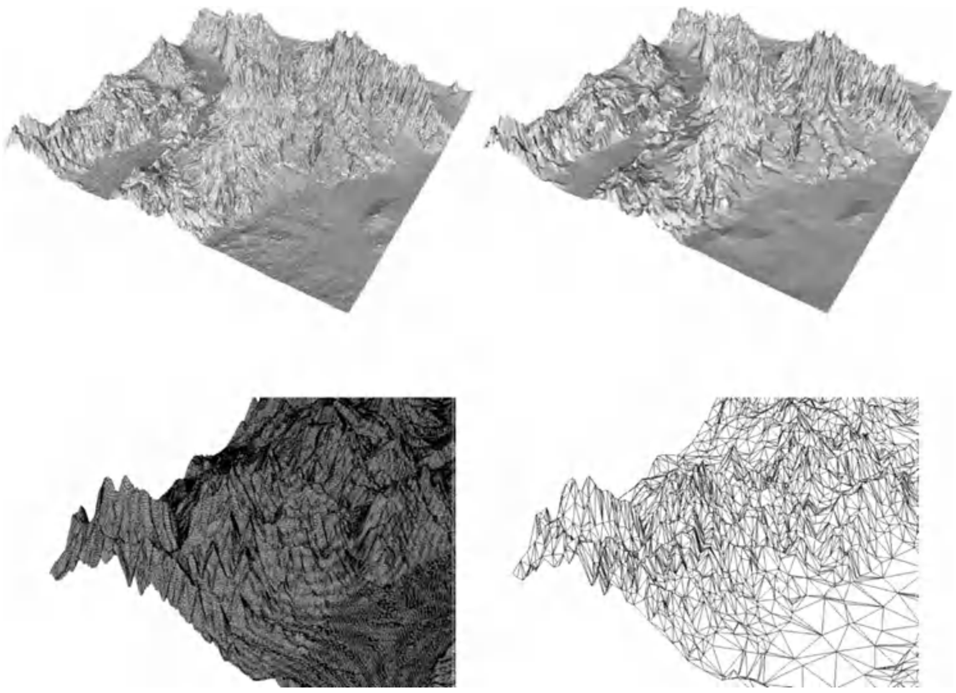


Figure 19.19: *Application to terrain simplification: the original map is made up of* 1,024 × 1,024 *vertices, thus* 1,046,529 *quadrangles; the simplified map includes only* 57,714 *vertices and* 45,421 *triangles and the gap is* $\delta = 5$.

Alternative method. Instead of using optimization tools, the Delaunay kernel (Chapter 7) after being reversed can be used to simplify a given mesh. The idea is to consider the classical insertion method of Relation 7.2

$$\mathcal{T}_{i+1} = \mathcal{T}_i - \mathcal{C}_P + \mathcal{B}_P \,, \tag{19.41}$$

where point $P = P_{i+1}$ is inserted in $\mathcal{T}_i$ and to reverse it before considering

$$\mathcal{T}_{i-1} = \mathcal{T}_i - \mathcal{B}_P + \mathcal{C}_P \,, \tag{19.42}$$

where vertex $P = P_i$ is removed from $\mathcal{T}_i$ while giving sense to the corresponding entities with regard to the surface in hand. Actually the ball $\mathcal{B}_P$ is nothing more than the triangles sharing point P and $\mathcal{C}_P$ is a set of triangles covering the polygon bounded by the external edges of the ball. While this makes sense only for a planar case, such a method can be used in the case of a surface and, in both case, not only one solution exists. Therefore, the problem simply turns to control the surface before and after a vertex deletion by observing the same criteria as in the previous approach. It is clear that deciding if a vertex must be removed must be made using a strategy to give some judicious priority in the deletion process.

Remark 19.30 *We have a clear parallel with the opposite process, e.g. surface enrichment by adding points to achieve some objectives. In this case, starting from* *a coarse surface mesh and given the set of points that can be added, the strategy relies in adding the point whose effect in the geometry approximation is maximal.*

Application to visualization. A "natural" application of mesh simplification is related to the representation of scenes (graphic visualization). Based on the distance and the point of view selected, it may be interesting to have various surface meshes at variable resolutions. Figures 19.18 and 19.19 illustrate this idea of mesh simplification adapted to graphic visualization.

Remark 19.31 *Moreover, mesh simplification enables the* compression *of surface meshes, which may be extremely useful, particularly for transmitting data [Taubin, Rossignac-1998].*

Chapter 20

A Touch of Finite Elements

Up to now we have discussed many aspects regarding mesh generation and mesh modification and while our final objective is to provide meshes required for finite element simulations, we have not yet dealt with such methods[1] directly.

However, in some parts of the book, we have faced some mesh generation problems where more advanced knowledge about finite elements was necessary to understand what needed to be done (see, for instance, the numbering issues related to finite element nodes in Chapter 17). Also, in later chapters, we will discuss h-methods, p-methods and hp-methods (in terms of meshing technologies) and clearly, at that time, it will be necessary to know more about the corresponding finite element requirements. Thus, covergae of the finite element method appears to be necessary before moving on to further investigations.

Nonetheless, library bookshelves are straining under the weight of literature on finite element theory as well as practical manuals for finite element methods, and it is clearly impossible for this book to compete with such a wealth of specialized literature. Thus, the point of view adopted here will be clearly motivated by the following question: "what must be known about finite elements in order to successfully deal with a meshing problem?".

In what follows, the theoretical point of view is largely based on [Ciarlet-1991] to which the reader is referred for a more advanced view of the problem together with a comprehensive list of relevant references about finite element theory and practice.

★
★ ★

Given this objective, the chapter is organized as follows. We consider a simple problem to which the finite element method is applied in order to find an approximate solution. The main aspects of the method are introduced. We start from the continuous PDE problem which models the problem under investigation. We replace this problem by a discrete problem whose solution is approached by means

[1]Only some brief allusions to finite element theory were given in Chapter 1.

of a finite element method. The notion of a finite element is discussed, defined as a geometrical element considered along with a set of degrees of freedom and a set of basic functions. Several types of finite elements are given including curved elements. Error estimate issues are briefly covered and related to the notions introduced in Chapter 10.

In terms of practice, curved finite elements are analyzed. Then, explicit use of a finite element approximation is discussed where we show how to compute a stiffness matrix, a right hand side and the resulting discrete system. This will present an opportunity to see how a finite element approximation can be implemented on a computer. Finally, some examples of popular finite elements are given to illustrate what types of interpolation, nodes and degrees of freedom can be encountered.

20.1 Introduction to a finite element style computation

To introduce the terminology, notations, etc., which will be used, we consider a very simple PDE problem and we briefly mention the main steps of its finite element approximation. Furthermore, P^1 and P^2 examples of finite elements will be illustrated. In specific, we will show how to compute the stiffness matrix and the right-hand side of such elements for the PDEs used as an example.

Let us consider a simple academic example. Let Ω be a domain and Γ be its boundary. We assume that Ω is composed of two sub-domains, denoted as Ω_i, and that Γ includes three parts, denoted as Γ_i. We want to solve the following problem: find u, the solution to

$$\left\{\begin{array}{rcll} -div(k_i \nabla u) & = & F_i & \text{in } \Omega_i \quad (i=1,2) \\ k_i \dfrac{\partial u}{\partial n} + g_i u & = & f_i & \text{on } \Gamma_i \quad (i=1,2) \\ u & = & 0 & \text{on } \Gamma_3 \end{array}\right. \tag{20.1}$$

where ∇ represents the gradient and $\dfrac{\partial}{\partial n}$ is the derivative with respect to the normal along Γ_i.

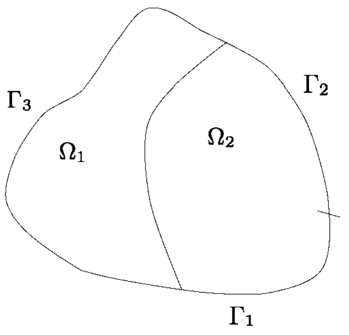


Figure 20.1: *The domain Ω with its two sub-domains and the domain boundary Γ with its three components.*

This system of partial differential equations models a heat transfer problem[2]. The physical conditions and the material coefficients to which the problem is subjected include the conductivity coefficient k_i of domain Ω_i, the source term F_i for the sub-domain Ω_i, the flux term f_i for the boundary Γ_i, the transfer coefficient g_i for the boundary Γ_i and, finally, the Dirichlet condition $u = 0$ which is prescribed on Γ_3.

In what follows, we will describe the method, without going into too much detail. The initial PDE problem is then analyzed with a view to a finite element approximation. This analysis includes several steps. A weak formulation is established, then approximate solutions are constructed.

First, a better suited formulation, called the *variational formulation*, or *weak formulation*, is derived from the above formulation by means *of partial differential equations* (PDE) (after a Green formula). This weak formulation can be written as follows:

$$\left\{\begin{array}{l} \text{Find} \quad u \in V \quad \text{such that} \\ a(u,v) = f(v)\,,\, \forall v \in V\,, \end{array}\right. \tag{20.2}$$

where V is the space of admissible values. In this way, we meet the operator a and the form f which are associated with the operators of Relationship (20.1).

In fact, using a Green formula, it can be seen that if u conforms to (20.1) and if v is an appropriate differentiable function, we have successively:

$$\sum_{i=1}^{2} \int_{\Omega_i} (-\,div(k_i\,\nabla u) - F_i)\, v \,=\, 0\,,$$

$$\sum_{i=1}^{2} \left(\int_{\Omega_i} k_i \nabla u \,\nabla v \,-\, \int_{\Omega_i} F_i\, v \,-\, \int_{\Gamma_i} k_i \frac{\partial u}{\partial n}\, v \right) = 0\,,$$

$$\sum_{i=1}^{2} \left(\int_{\Omega_i} k_i \nabla u \,\nabla v \,+\, \int_{\Gamma_i} g_i\, u\, v \,-\, \int_{\Omega_i} F_i\, v \,-\, \int_{\Gamma_i} f_i\, v \right) = 0\,.$$

This leads to a formulation equivalent to (20.1), this variational formulation then serves as a starting point for the finite element approximation. We return to the formulation given in (20.2) by introducing the following definitions:

$$a(u,v) = \sum_{i=1}^{2} \left(\int_{\Omega_i} k_i \nabla u \,\nabla v \,+\, \int_{\Gamma_i} g_i\, u\, v \right)$$

$$f(v) = \sum_{i=1}^{2} \left(\int_{\Omega_i} F_i\, v \,+\, \int_{\Gamma_i} f_i\, v \right)\,.$$

[2]I.e., the initial problem is a heat transfer problem and physicists have proved that such a PDE system is an adequate model for it.

This *continuous problem* is then replaced by an *approximate problem.* In practice, the explicit solution of the continuous problem is not generally possible. This has led to the investigation of approximate solutions using, in our context, the finite element method. Basically, this method consists of constructing a *finite dimensional* sub-space V_h of space V and defining u_h, an approximate solution of u, the solution to the following problem:

$$\left\{ \begin{array}{l} \text{Find} \quad u_h \in V_h \quad \text{such that} \\ a(u_h, v_h) = f(v_h)\,, \forall v_h \in V_h\,. \end{array} \right. \tag{20.3}$$

Under appropriate assumptions (see below), it can be shown that this problem has a unique solution, u_h, and that the *convergence* of u_h to the solution, u, is directly related to the manner in which the functions v_h of space V_h approach the functions v of space V and therefore the manner in which the space V_h is defined.

Thus, the finite element method consists of constructing a finite dimensional space V_h such that, on the one hand, a suitable approximation is obtained and, on the other hand, the actual computer implementation is not too difficult. This construction is based on the three following basic ideas:

1. the creation of a mesh, denoted by $\mathcal{T}_h$[3] or simply $\mathcal{T}$, of domain Ω so that the domain can be written as a finite union of elements K, the members of $\mathcal{T}$;
2. the definition of V_h as the set of functions v_h, whose restriction to each element K in $\mathcal{T}$ is, in general, a polynomial;
3. the existence of a basis for the space V_h whose functions have a *small support.*

In other words, constructing a mesh $\mathcal{T}_h$ is an essential prerequisite when defining space V_h. The functions in V_h are usually polynomials over every element in the mesh which are continuous over the neighboring elements (i.e., across the element interfaces). The simplest case for triangular meshes (in two dimensions) is to choose piecewise linear functions which, after interpolation on the mesh node values, make it possible to define the desired solution everywhere.

Remark 20.1 *Index h in V_h (and in $\mathcal{T}_h$) is related to the fact that V_h depends on the mesh. This value h must be seen as a size parameter (the element diameter) which, in fact, defines a family of meshes and, therefore, a family of approximations and which, to demonstrate the theoretical convergence and accuracy (the approximation order), tends towards* 0.

The elements K in mesh $\mathcal{T}_h$ have a certain number of properties which are characteristics in the finite element method (Chapter 1).

[3]By convention, $\mathcal{T}_h$ denotes a triangulation of Ω such that:

$$h = \max_{K \in \mathcal{T}_h} h_K$$

where h_K is the diameter of polyhedron K.

The choice of the basis function in V_h consists of taking functions whose restriction at every element K is written as:

$$u_h(x) = \sum_{i=1}^{N} \phi_i(u)\, p_i(x)\,, \tag{20.4}$$

where p_i is the basis function i of the polynomial space previously defined (while $p_i(x)$ is the value of this basis function at a location x in K) and $\phi_i(u)$ is the value of the degree of freedom i associated with u_h while N denotes the *number of degrees of freedom* and the number of p_is as well, i.e., the dimension of the space engendered by these p_is. As will be seen in greater detail below, a finite element is then characterized by a suitable choice of the following triple:

- K, a geometrical element;
- P_K, a finite dimensional space of functions defined over K;
- Σ_K, a set of degrees of freedom associated with the functions defined over K.

Hence, for all unknowns in the problem (here, the unique unknown, u, is a "temperature") we have $dim P_K = card(\Sigma_K) = N$ and a section below discusses this triple in more detail, the basic support for defining the finite elements.

Provided with these choices, Problem (20.3) is then replaced by a matrix problem. In fact, if locally, in each K, we have:

$$u_h(x) = \sum_{i=1}^{N} \phi_i(u)\, p_i(x)\,,$$

globally we have a similar expression, i.e.,:

$$u_h(x) = \sum_{j=1}^{M} \phi_j^*(u)\, p_j^*(x)\,,$$

where M is now the total number of degrees of freedom when the mesh is considered globally and when we have removed[4] the degrees of freedom of the nodes in Γ_3, while $\phi_j^*(u)$ and $p_j^*(x)$, at index j, are indeed the values $\phi_i(u)$ and $p_i(x)$ for the indices i of the corresponding elements K. Therefore, for (20.3), we have:

$$a(u_h, p_j^*(x)) = f(p_j^*(x)) \text{ for } j = 1, M$$

and the problem is now:

$$\left\{ \begin{array}{l} \text{Find} \quad U_k \; k = 1, M \quad \text{such that} \\ a(\sum_k U_k p_k^*(x), p_j^*(x)) = f(p_j^*(x))\,, \end{array} \right. \tag{20.5}$$

[4]This is a formal point of view, in practice, some techniques are used that allow for the Dirichlet boundary conditions with no specific numbering of the nodes as implicitly assumed here.

which can be written in a matrix form as follows:

$$\mathcal{A} U = \mathcal{B}$$

where $\mathcal{A}$ is the matrix with coefficient $a_{i,j} = a(p_i^*(x), p_j^*(x))$ and $\mathcal{B}$ is the *right-hand side*, whose component b_j is $b_j = f(p_j^*(x))$.

Before discussing the element level, note that the basis function j, in the linear case, is 1 at node j and 0 otherwise (the *hat* function). This property leads to sparse matrices, in fact, we have $a_{i,j} = 0$ for most pairs (i, j). For example, again in the simplest case, the triangle of degree 1, we have $a_{i,j} = 0$, unless the nodes i and j belong to the same triangle K.

20.2 Definition and first examples of finite elements

Here, we return to the formal definition of a finite element, using the above triple. We then give some examples of such triples.

Definition of a finite element

Now, we specify what the above triple means. As previously mentioned and again after [Ciarlet-1991], a finite element is defined by the triple[5]:

$$[K, P_K, \Sigma_K] .$$

In this triple K is the "geometric" element, i.e., a mesh element. P_K denotes the space functions of the finite element. The basis functions in this space are referred to as the *shape functions* of the finite element. Finally, Σ_K is the set of degrees of freedom associated with K.

From a geometric point of view, and according to the space dimension, K may be a triangle, a quadrilateral, a tet, a pentahedron, a hex, etc.

Space P_K is generally made of polynomials. This space with the finite dimension N is built from N basis functions. Thus, if p_i is a basis function, any function p in P_K can be written (cf. Relation (20.4)) as:

$$p = \sum_{i=1}^{N} \alpha_i \, p_i \, ,$$

where the α_is are some coefficients (in fact, the degrees of freedom).

The set of degrees of freedom Σ_K, the ϕ_is in Relation (20.4) or, similarly, the above α_is may be some values of the function p, where p is a function in P_K, at

[5]In fact, K itself is often referred to as a finite element. This clearly corresponds to a simplification in the notations since, indeed, this is K combined with P_K and Σ_K which is a finite element.

a node[6] in the element K. Such a node is denoted by a in the following. A finite element with this type of node is said to be a *Lagrange* type element.

When the $\phi_i(p)$s include at least one value of a derivative of p, for example, with evident notations, Dp, $\frac{\partial p}{\partial x}$, $\frac{\partial p}{\partial y}$, $\frac{\partial p}{\partial n}$, D^2p, $\frac{\partial^2 p}{\partial x^2}$, etc., we face a *Hermite* type finite element.

Following these definitions, we note that P_K can be seen as a space of functions (the ps) or directly as a list of basis functions (the p_is), which is clearly two ways of writing the same thing. Similarly, Σ_K can be defined as a set of degrees of freedom, the $\phi_i(p)$s, and we reach a global node definition, or a set of nodes, the as, with which one or several degree(s) of freedom is(are) associated. Based on the context, one or the other of these definitions will be used.

The important fact is that the triple $[K, P_K, \Sigma_K]$ properly defines a finite element if the set Σ_K is P_K*-unisolvent.* This means that there exist N functions p_i's in space P_K which are linearly independent. In particular, we have:

$$\phi_j(p_i) = \delta_{ij}$$

with δ_{ij} the Kronecker delta ($\delta_{ij} = 1$ if $i = j$ and $\delta_{ij} = 0$ otherwise).

A more general definition of this concept is to make use of one *reference element*, $\hat{K}$, together with a mapping function F_K so as to define the above triple from a reference triple denoted by:

$$[\hat{K}, \hat{P}, \hat{\Sigma}] .$$

In other words, a unique reference triple $[\hat{K}, \hat{P}, \hat{\Sigma}]$ may serve to characterize all finite elements in the mesh which have the same geometric type provided the corresponding functions F_K are given. Thus, starting from one triple $[\hat{K}, \hat{P}, \hat{\Sigma}]$ and one function F_K, it is possible to obtain the triple $[K, P_K, \Sigma_K]$ of every element K. If $\hat{p}$ (resp. $\hat{a}$) denotes the functions (resp. the nodes) in $\hat{P}$ (resp. in $\hat{K}$), we have:

$$\begin{array}{rcl} K &=& F_K(\hat{K}) \\ P_K &=& \{p = \hat{p} \circ F_K^{-1} \, , \hat{p} \in \hat{P}\} \\ \Sigma_K &=& \{p(a) \, , a = F_K(\hat{a}) \, , \, \hat{a} \in \hat{K}\} . \end{array} \tag{20.6}$$

where, for the sake of simplicity, we assume only one degree of freedom of the form $p(.)$ and where $\hat{a}$ stands for the nodes in $\hat{K}$. As previously mentioned, P_K can be also written as:

$$P_K \quad = \quad \{p = \sum \alpha_i p_i \quad \text{with} \quad p_i = \hat{p}_i \circ F_K^{-1} \, , \hat{p}_i \in \hat{P} \, , i = 1, N\} \tag{20.7}$$

for convenience, while in turn and for the same reason, Σ_K can be expressed in two ways:

$$\Sigma_K \quad = \quad \{\phi_i(p) \, , i = 1, N\} . \tag{20.8}$$

$$\Sigma_K \quad = \quad \{a_j \, , j = 1, M \, ; \phi_i(p(a_j^i)) \, , i = 1, N\} . \tag{20.9}$$

[6] A node supports one (or several) degrees of freedom. According to their type, the nodes in a finite element may be its vertices, its mid-edge points, some points in its facets or some particular points inside K.

In other words, either using the entire list of degrees of freedom, or using the list of nodes and, for each of them, the list of the corresponding degrees of freedom.

To make sense, the above definition implies that F_K is *invertible*. This being satisfied, the two triples are equivalent (and, if in addition, F_K is affine as it is in the P^1 case, the two triples are affine equivalent). In this way, we have defined a family of finite elements which may be represented using a unique element, the reference element. Notice that making use of a reference element is not strictly required in the case of simplicial element, but this is a source of simplification due to its great ease of application.

Remark 20.2 *This notion of equivalence allows us to reduce the definition of a finite element to that of the corresponding reference element. As will be seen, this property is met again at the computational step, a part of the computation for a given element concerns the reference element and therefore does not change when we consider one element or another of the mesh.*

Remark 20.3 *In the case where the function is affine, we have affine elements. In contrast, other types of elements can be defined, for example, curved (or isoparametric) elements.*

First examples of finite elements

In this section we give some examples of finite elements which are rather simple (and widely used). At the end of this chapter, a certain number of other finite elements will be given.

The P^1 triangle in two dimensions. The first example is the well known P^1 Lagrange triangle in two dimensions (also referred to as Courant's triangle or as the $T3$ element). It is defined, Figure 20.2 (top), by means of the following reference definitions:

- $\hat{K}$, the straight triangle of "unit" side whose vertices are $\hat{v}_1 = (0,0)$, $\hat{v}_2 = (1,0)$ and $\hat{v}_3 = (0,1)$.
- $\hat{P} = P^1 = \{1 - \hat{x} - \hat{y} \ , \hat{x} \ , \hat{y}\}$, i.e., $dim P_K = 3$.
- $\Sigma = \{\hat{a}_i = \hat{v}_i \, , i = 1,3 \ ; \phi_i(p) = p(a_i), a_i = F_K(\hat{a}_i), \ i = 1,3\}$
- $F_K \in (P^1)^2$.

Remark 20.4 *Note that F_K can be written as $F_K = \{F_K^i\}_{i=1,d}$ where d is the spatial dimension and we have $F_K^i \in P^1$.*

The Q^1 quadrilateral in two dimensions. The Q^1 Lagrange quad in two dimensions (also referred to as the $Q4$ element) is defined as shown in Figure 20.2 (bottom):

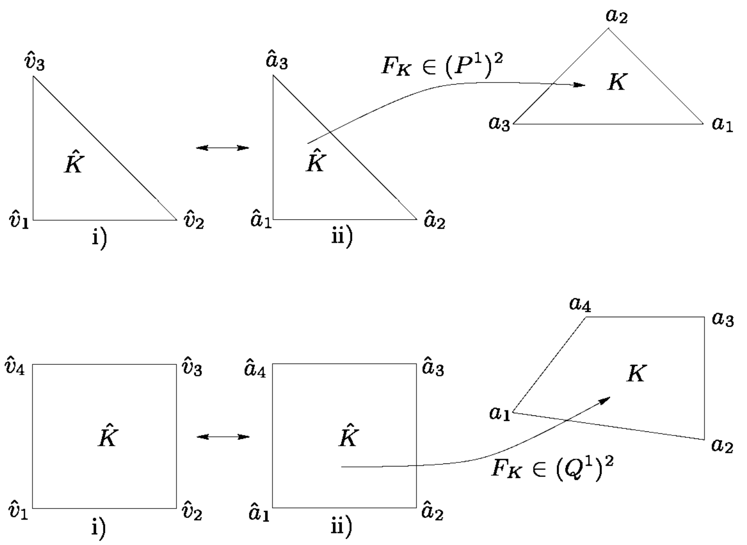


Figure 20.2: *Lagrange type finite elements of degree 1, the $T3$ and the $Q4$ finite elements. Left-hand side: the vertex labeling in the reference element (indeed $\hat{v}_1$ could simply be the vertex of label* 1*, etc.). Middle: the reference node labeling. Right-hand side: the current node labeling.*

- $\hat{K}$, the unit square whose first vertex $\hat{v}_1$ is at the origin[7].
- $\hat{P} = Q^1 = \{(1-\hat{x})(1-\hat{y})\ ,\hat{x}(1-\hat{y})\ ,\hat{x}\hat{y}\ ,(1-\hat{x})\hat{y}\,\}$. $dim P_K = 4$.
- $\Sigma = \{\hat{a}_i = \hat{v}_i\,, i = 1,4\ ; \phi_i(p) = p(a_i), a_i = F_K(\hat{a}_i),\ i = 1,4\}$.
- $F_K \in (Q^1)^2$.

The two P^2 triangles in two dimensions. Figure 20.3 (left-hand side), the *affine* P^2 Lagrange triangle in two dimensions (also referred to as the $T6$ element) is defined as:

- $\hat{K}$, the "unit" triangle as for the $T3$ element.
- $\hat{P} = P^2 = \{(1-\hat{x}-\hat{y})(1-2\hat{x}-2\hat{y})\ ,\hat{x}(2\hat{x}-1)\ ,\hat{y}(2\hat{y}-1)\ ,4\hat{x}(1-\hat{x}-\hat{y})\ ,4\hat{x}\hat{y}\ ,4\hat{y}(1-\hat{x}-\hat{y})\ \}$. $dim P_K = 6$.
- $\Sigma = \{\hat{a}_i = \hat{v}_i\,, \hat{a}_{i+3} = \frac{\hat{v}_i+\hat{v}_{i+1}}{2}\ , i = 1,3\ ; \phi_i(p) = p(a_i), a_i = F_K(\hat{a}_i),\ i = 1,6\}$
- $F_K \in (P^1)^2$.

The *isoparametric* P^2 Lagrange triangle in two dimensions is similarly defined but, in this case, we have $F_K \in (P^2)^2$.

[7]Obviously, it is also possible to center the square, to give it a side 2 and to define if from -1 to $+1$, for example.

The two Q^2 quadrilaterals in two dimensions. The Q^2 Lagrange quad in two dimensions (also referred to as the $Q8$ element), (see Figure 20.3 (right-hand side)), is defined as:

- $\hat{K}$, the unit square.
- $\hat{P} = Q^2 = \{(1-\hat{x})(1-\hat{y})(1-2\hat{x}-2\hat{y})\ ,\hat{x}(1-\hat{y})(2\hat{x}-2\hat{y}-1)\ ,-\hat{x}\hat{y}(3-2\hat{x}-2\hat{y})\ ,(1-\hat{x})\hat{y}(-2\hat{x}+2\hat{y}-1)\ ,4\hat{x}(1-\hat{x})(1-\hat{y})\ ,4\hat{x}\hat{y}(1-\hat{y})\ ,4\hat{x}(1-\hat{x})\hat{y}\ ,4(1-\hat{x})\hat{y}(1-\hat{y})\}$. $dimP_K = 8$.
- $\Sigma = \{\hat{a}_i = \hat{v}_i\,,\hat{a}_{i+4} = \frac{\hat{v}_i+\hat{v}_{i+1}}{2}\ ,i = 1,4\ ;\phi_i(p) = p(a_i), a_i = F_K(\hat{a}_i),\ i = 1,8\}$.
- $F_K \in (Q^1)^2$ (affine case) or $F_K \in (Q^2)^2$ (isoparametric case).

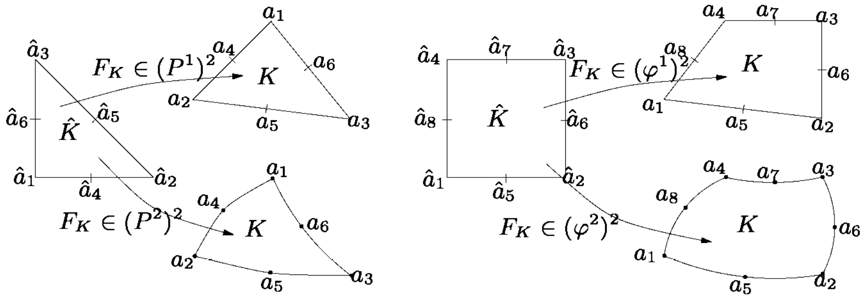


Figure 20.3: *Lagrange type finite elements of degree 2, affine (top) and isoparametric (bottom) $T6$ and $Q8$ finite elements together with the corresponding reference element.*

Remark 20.5 *In the case of an affine element, the mid-edge node is the midpoint of the edge and, for instance, for the $T6$ element, $a_4 = F_K(\hat{a}_4) = F_K(\frac{\hat{a}_1+\hat{a}_2}{2})$. In this case, we have $F_K(\frac{\hat{a}_1+\hat{a}_2}{2}) = \frac{F_K(\hat{a}_1)+F_K(\hat{a}_2)}{2}$ and a_4 can be obtained as $\frac{a_1+a_2}{2}$. However, for the curved $T6$ element $F_K(\frac{\hat{a}_1+\hat{a}_2}{2}) \neq \frac{F_K(\hat{a}_1)+F_K(\hat{a}_2)}{2}$ and then $a_4 = F_K(\hat{a}_4) = F_K(\frac{\hat{a}_1+\hat{a}_2}{2})$ is not, in general, $\frac{a_1+a_2}{2}$.*

In a subsequent section, we will give various examples of common finite elements other than those presented here.

20.3 Error estimation and convergence

The initial problem, Problem (20.2), concerns a solution in space V of $a(u,v) = f(v)$. The approached problem, Problem (20.3), is to find a solution to $a(u_h, v_h) = f(v_h)$ in space V_h. If this problem is solved using one or several numerical schemes or quadratures, this leads to the solution $a_h(u_h, v_h) = f_h(v_h)$ in which a_h (resp. f_h) are approximations of a (resp. of f). The question is then to ensure that the approached solution u_h tends towards u with h in a pertinent way thanks to a proper choice of the various parameters and ingredients used in the various steps included in the approximation.

Local (element) quantity and global quantities

Passing from the initial continuous problem (formulated in domain Ω) to its numerical approximation and to the calculus corresponding to the mesh elements approaching Ω is thus made by replacing the global quantities by some local discrete quantities. The various relationships between these two types of quantities are as follows:

- Ω is replaced by $\mathcal{T} = \bigcup_{K \in \mathcal{T}} K$,
- Γ is approached by the "sides" ∂K of the appropriate element K,
- a function v_h in V_h is replaced by its restriction in K, v_h/K, in space $P = P_K$,
- a global node b_j corresponds to a set of local nodes a_i, several elements K sharing this node b_j. A numbering problem is found here leading to identifying the global index of a local node,
- similarly, the basis functions and the degrees of freedom are seen at a global and a local level,
- the restriction in K of a $v_h = \Pi_h v$ is $\Pi_K v$.

These notions (and notations) being done, we recall, using a simple example (an elliptic problem) the abstract basis of the convergence and error estimate issues.

Error estimation and convergence issues

Error estimate as well as convergence of the approximation is analyzed by observing if:

$$\|u - u_h\| \longrightarrow 0$$

when $h \longrightarrow 0$ in some sense, i.e., for an appropriate norm $\| \, . \, \|$.

From a mathematical point of view, the notion of a convergence is linked not to a (discrete) approximation but to a family of (discrete) approximations defined by a parameter h which tends towards zero. With each value of h is associated a space V_h and with each V_h is associated one approximation u_h of Problem (20.3):

$$a(u_h, v_h) = f(v_h)\,.$$

Such a family is said to be convergent if:

$$\lim_{h \longrightarrow 0} \|u - u_h\| = 0\,.$$

After the previous discussion, the convergence of u towards u_h, i.e., $||u - u_h|| \longrightarrow 0$, is based on Cea's lemma which proves that a constant C exists, independent of h, such that:

$$||u - u_h|| \leq C \inf_{v_h \in V_h} ||u - v_h||$$

and, therefore, a sufficient condition for convergence is that there exists a family of V_h such that:

$$\lim_{h \longrightarrow 0} \inf_{v_h \in V_h} ||u - v_h|| = 0 .$$

Then, we need to show that:

$$||u - u_h|| \leq C(u)\, h^{\beta} ,$$

which implies that:

$$||u - u_h|| = \mathcal{O}(h^{\beta}) ,$$

meaning that the convergence is of order β for the norm $||\,.\,||$, while it is obvious that different norms can be found according to the regularity of the problem and the nature of the approximation.

The distance between u and u_h, *i.e*, $\inf_{vh \in V_h} ||u - v_h||$, is estimated by looking at the distance between u and $\Pi_h u$ the interpolate of u. This leads to a result in:

$$||u - u_h|| \leq C\, ||u - \Pi_h u|| .$$

If we assume Ω to be a polygonal (resp. polyhedral) domain, then we have strictly:

$$\Omega = \sum_{K \in \mathcal{T}} K$$

where K is a polygonal (resp. polyhedron) and, in this case, straight finite elements (the above triple) can be chosen in ordert To construct the sub-spaces V_h (which are therefore included in V). Then, in such a situation, we have a *conforming* finite element method.

Hence,

$$(\Pi_h\, u)/K = \Pi_K u$$

for all K in $\mathcal{T}$ and, we can write:

$$||u - \Pi_h\, u|| = \{\sum_{K \in \mathcal{T}} ||u - \Pi_K\, u||^2\}^{\frac{1}{2}} .$$

As a consequence, estimating the error between u and u_h comes down to seeing what the local errors of interpolation are:

$$||u - \Pi_K\, u|| .$$

We then show (after some assumptions about the interpolation) that there exists a constant C, independent of K, such that:

$$|v - \Pi_K v|_{m,K} \leq C\, \frac{h_K^{k+1}}{\rho_K^m} |v|_{k+1,K}$$

for $0 \leq m \leq k+1$ and for all v in $H^{k+1}(K)$. This relation is related to the degree k of the polynomials in P_K and two quantities already mentioned many times:

- the diameter of element K, say h_K and
- the radius of the circle (the sphere) inscribed in K, i.e., ρ_K.

From a practical point of view and for the envisaged error estimate, we use one reference element, $\hat{K}$, on which the error is estimated. We then evaluate this error on element K (assumed to be affine equivalent). The full proof for this estimation is slightly delicate and makes use of various issues in functional analysis. This is beyond our scope here. It is enough to retain that, based on the problem considered and according to the assumed regularity, we obtain some majorations for some appropriate norms which enable us to prove the convergence of these approximations together with the order of this convergence.

Remark 20.6 *Unsurprisingly, some quality criteria seen in Chapter 18 involve the ratio between h and ρ.*

Quadrature influence

As already seen, the actual computation often makes use of integration schemes. In this way, the problem to be solved is no longer $a(u_h, v_h) = f(v_h)$ but rather $a_h(u_h, v_h) = f_h(v_h)$. If ϕ stands for a function that must be integrated in K, we then compute:

$$\int_K \phi(x,y)dK = \sum_l \omega_l \phi(x_l, y_l)$$

where the quadrature is defined by means of l points (x_l, y_l) and l weights ω_l.

Bear in mind that the choice of the quadrature formula must be made in such a way as to retrieve the corresponding error estimate (the order of convergence) when no quadrature is used. In other words, the quality of the approximation is preserved.

For instance, this is the case when, for an approximation based on polynomials of degree k, we use a formula which exactly integrates the polynomials of degree $2k-2$.

Curved elements

Using curved or isoparametric elements is one way to deal with non-polygonal (polyhedral) domains for which an approximation by means of straight finite elements leads to a poor geometric approximation.

For such domains whose boundary or a portion of the boundary is curved, we generally use two types of finite elements. Inside the domain[8], straight finite elements are employed. Near the domain boundary, we use finite elements with at least one curved edge (face), in order to have a more precise approximation of the underlying geometry. These elements are said to be "curved" or isoparametric.

[8]In principle, as it is not necessarily true everywhere.

In this short section, we give some indications about this type of element and, specifically, we show how they can be constructed.

For such elements, the error analysis is made as it was for straight (affine) elements. In particular, provided the deformation between the curved elements and the straight elements is not too great, the same discussion holds. Similarly, the role of a quadrature formula is as above.

The concern is more specifically how to define an isoparametric element. This leads to a problem of element node definition which comes down to finding a (local) index and, more importantly, a suitable location for these nodes.

Let us again take the case of a curved $T6$ triangle. As previously seen, this element has as its nodes its three vertices and its three mid-edge points. In fact, these last three nodes are effectively the midpoints of the edges when these edges are straight segments. The case of a curved edge is slightly different. Here the theory and the practice are somewhat different. In theory, specifically for a convergence study, it is convenient to take as a mid-edge node for a given edge the point projection on the boundary of the mid-edge point; we assume in addition that the distance between these two points is small. This is clearly the case when h tends towards zero, and therefore the convergence issue may be based on such an assumption. Nevertheless in practice, the curved edges are represented by a number of *arcs of parabola* and, *unfortunately*, the best approximation possible for a curve by such arcs does not mean that the projection of the midpoint of the edge is the best possible candidate to construct an arc of parabola close enough to the geometry. Moreover, since h does not tend towards zero, such a fact is not necessarily without effect. Thus, as will be seen in detail in Chapter 22, the mid-edge nodes are not necessarily the projections of the mid-edge points concerned, they are also related to the local curvature of the boundary near the region under examination.

To summarize, it is necessary to:

- mesh the curve related to the boundary edge by means of an arc of parabola as close as possible to the geometry (Chapter 22),

- check that the Jacobian of the transformation defined from the reference element to the current element is strictly positive in all points, so as to ensure the property of invertibility and thus results in elements with a positive surface area.

Nevertheless, this scheme merits some remarks. First, it is not always true, in practice, that the edge is close enough to the boundary curve. This condition would hold in regions with a high curvature (i.e., where the minimal radius of curvature is small) only for a sufficiently small edge. Then, the P^2 element (if we consider this case) thus constructed would probably be adequate but certainly too small, therefore leading to a mesh with too many elements. This refutes the fact that we know in advance that, for an equivalent or a better geometric approximation, the size h of a curved element may be larger than the size of a straight element constructed at the same place. An intuitive idea is to consider that a P^1 mesh of size at least $2\,h$ allows the construction of a P^2 mesh with the same size which is equivalent or better than a P^1 mesh with the half size (i.e., h). Note that in this way, for the same number of degrees of freedom, the number of elements is less. We refer the reader to Chapter 22, where there are some indications about the construction of P^2 meshes (the discussion is based on a P^1 mesh completed by one of the automatic mesh generation methods previously described).

Remark 20.7 *In this discussion, we have essentially seen as a curved element example the case of simplicial elements (in two dimensions) of degree 2. However, it is clear that there exist other isoparametric elements, with a degree other than 2, a dimension other than 2 and a geometric nature other than simplicial.*

Remark 20.8 *To conclude, notice that the case of elements in three dimensions is far from being trivial. Indeed, this case leads to surface meshing problems.*

20.4 Stiffness matrix and right-hand side

In this section, we firstly give the general expressions of the stiffness matrix and the right-hand side associated with the problem presented here. We give the corresponding element quantities (at the level of one element) and we show the case of an affine $T3$ triangle and for an affine $T6$ triangle together with an isoparametric $T6$ triangle. Then we discuss some specific questions regarding the information that must be stored or found in the mesh so as to make it possible to obtain these quantities. We also look at the quadrature formula and the way in which the global quantities are obtained by *assembling* the element quantities thus obtained.

General expression of a stiffness matrix

The stiffness matrix results from the contribution of $a(u_h, v_h)$ of Equation (20.1) through the two terms $-div(k_i \nabla u)$ and $g_i u$. After a Green formula, we have

$$a_h(u_h, v_h) = \int_{\Omega_i} k_i \nabla u_h \nabla v_h + \int_{\Gamma_i} g_i u_h v_h$$

and thus (omitting index i)

$$a_h(u_h, v_h) = \sum_{K \in T} \int_K k \nabla u_h \nabla v_h + \sum_{K \in T} \int_{K \cap \Gamma} g u_h v_h \,.$$

Apart from the coefficients, the quantities we are interested in are the restrictions over the elements of the various functions of the above relationship. For convenience we denote $[\mathcal{K}]$ the $d \times d$ matrix with coefficients k. We now consider $(u_h)/K$, the restriction over K of u_h. Following the approximation we have $(u_h)/K = \sum\limits_i \phi_i(u)\, p_i$ where $\phi_i(u)$ is the degree of freedom i associated with function u_h and p_i is the basis polynomial i of space P_K (note that v_h is similarly considered).

With $[\mathcal{P}]$ the row $[p_1, p_2, ..., p_N]$ and $\{u_{i,K}\}$ the vector of the degrees of freedom in K, the above expression is conveniently written as

$$\{(u_h)/K\} = [\mathcal{P}]\,\{u_{i,K}\}\,.$$

With this concise notation, we have

$$\nabla(u_h)/K = {}^t\{\frac{\partial u_h/K}{\partial x}, \frac{\partial u_h/K}{\partial y}, \frac{\partial u_h/K}{\partial z}\}$$

which is simply expressed by

$$\{\nabla(u_h)/K\} = [\mathcal{DP}]\,\{u_{i,K}\}\,,$$

where

$$[\mathcal{DP}] = \begin{bmatrix} \frac{\partial p_1}{\partial x} & \frac{\partial p_2}{\partial x} & \cdots & \frac{\partial p_N}{\partial x} \\ \frac{\partial p_1}{\partial y} & \frac{\partial p_2}{\partial y} & \cdots & \frac{\partial p_N}{\partial y} \\ \frac{\partial p_1}{\partial z} & \frac{\partial p_2}{\partial z} & \cdots & \frac{\partial p_N}{\partial z} \end{bmatrix}$$

Thus, using these quantities, we have

$$\int_K k\nabla u_h \nabla v_h = {}^t\{v_{i,K}\} \int_K {}^t[\mathcal{DP}][\mathcal{K}][\mathcal{DP}]dK\ \{u_{i,K}\}$$

$$\text{and} \int_{\partial K} g u_h v_h = {}^t\{v_{i,K}\} \int_{\partial K} {}^t[\mathcal{P}]g[\mathcal{P}]d\partial K\ \{u_{i,K}\}\,,$$

where $\int_K {}^t[\mathcal{DP}][\mathcal{K}][\mathcal{DP}]dK + \int_{\partial K} {}^t[\mathcal{P}]g[\mathcal{P}]d\partial K$ forms $[\mathcal{A}_K]$ the so-called *elementary stiffness matrix* of element K.

As previously indicated, this matrix is actually computed by means of the corresponding reference element $\hat{K}$. We have, following (20.7), $p_i = \hat{p}_i \circ F_K^{-1}$, thus

$$[\mathcal{P}] = [\hat{\mathcal{P}}] \text{ and } [\mathcal{DP}] = [\mathcal{DF}^{-1}][\mathcal{D}\hat{\mathcal{P}}]$$

hold where $[\mathcal{DF}]$ denotes the derivatives of F_K. Using this yields:

$$[\mathcal{A}_K] = \int_{\hat{K}} {}^t[\mathcal{D}\hat{\mathcal{P}}]\,{}^t[\mathcal{DF}^{-1}][\mathcal{K}][\mathcal{DF}^{-1}][\mathcal{D}\hat{\mathcal{P}}]\mathcal{J}d\hat{K} + \int_{\partial\hat{K}} {}^t[\hat{\mathcal{P}}]g[\hat{\mathcal{P}}]\mathcal{J}_\Gamma d\partial\hat{K}\,,$$

with $\mathcal{J}$ the Jacobian of F_K, i.e., $\mathcal{J} = det(\mathcal{DF})$ and $\mathcal{J}_\Gamma$ the corresponding term on Γ, i.e., we have $d\partial K = \mathcal{J}_\Gamma d\partial\hat{K}$. In terms of computation, the above integrals can be obtained directly (exact quadrature) or after a numerical quadrature based on some quadrature nodes (and coefficients). In this case, an integral is replaced by a summation and we need to evaluate the quantities of interest at the nodes of the quadrature formula.

Having the elementary matrices ready, we can obtained the global stiffness matrix $[\mathcal{A}]$. As can be easily established we have

$$[\mathcal{A}] = \sum_{K \in T} {}^t[\mathcal{B}_K][\mathcal{A}_K][\mathcal{B}_K]\,,$$

where $[\mathcal{B}_K]$ ensures the correspondence between the local node numbering (i.e., when element K is considered) and the global node numbering (when all the mesh is considered). Thus, $[\mathcal{B}_K]$ gives the correspondence between the local and the global numbering and enables us to assemble[9] the global matrix by merging in it the contributions due to the local matrices (see below).

General expression of a right-hand side

The right-hand side results from the contribution of $f(v_h)$ of Equation (20.1) through the term over the domain and that due to the boundary. We have,

$$f(v_h) = \int_{\Omega_i} F_i v_h + \int_{\Gamma_i} f_i v_h$$

and thus (omitting index i),

$$f(v_h) = \sum_{K \in \mathcal{T}} \int_K F v_h + \sum_{K \in \mathcal{T}} \int_{K \cap \Gamma} f v_h \,,$$

or again

$$f(v_h) = \sum_{K \in \mathcal{T}} {}^t\{v_{i,K}\}\{ \int_K {}^t[\mathcal{P}] F dK + \int_{K \cap \Gamma} {}^t[\mathcal{P}] f d\partial K \} \,.$$

The elementary contribution, the so-called *elementary right-hand side*, is composed of the two terms $\int_K {}^t[\mathcal{P}]FdK + \int_{K\cap\Gamma} {}^t[\mathcal{P}]fd\partial K$. It is denoted by $[\mathcal{RHS}_K]$. The calculation is performed over $\hat{K}$:

$$[\mathcal{RHS}_K] = \int_{\hat{K}} {}^t[\mathcal{P}] F \mathcal{J} d\hat{K} + + \int_{\partial \hat{K}} {}^t[\mathcal{P}] f \mathcal{J}_\Gamma d\partial \hat{K} \,.$$

As before, using the matrix $[\mathcal{B}_K]$ allows us to obtain the global right-hand side:

$$[\mathcal{RHS}] = \sum_{K \in \mathcal{T}} {}^t[\mathcal{B}_K][\mathcal{RHS}_K] \,.$$

Remark 20.9 *Elementary (global) quantities other than the stiffness matrix or the right-hand side, namely, the mass matrix (in an evolution problem), etc., can be obtained in the same way.*

About the $T3$ triangle

To demonstrate the above discussion, we look at the $T3$ triangle in two dimensions. In this example, we consider $d = 2$ and $N = 3$ while the basis polynomials are $[\hat{\mathcal{P}}] = \{1 - \hat{x} - \hat{y} \,, \hat{x} \,, \hat{y}\}$. Then, we have:

$$[\mathcal{D}\hat{\mathcal{P}}] = \begin{bmatrix} -1 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix}$$

[9]Notice that some solution methods do not require the explicit construction of the global matrix.

Also $F_K = \{F_K^i\}_{i=1,2} \in (P^1)^2$. Then, if a_i stands for node i of K and (x_i, y_i) denotes its two coordinates, we define $a_{ij} = a_i - a_j$ and similar expressions for x_{ij} and y_{ij} and since

$$F_K(\hat{x}, \hat{y}) = (1 - \hat{x} - \hat{y})\{a_1\} + \hat{x}\{a_2\} + \hat{y}\{a_3\},$$

we have

$$F_K(\hat{x}, \hat{y}) = \{a_1\} + \hat{x}\,\{a_{21}\} + \hat{y}\,\{a_{31}\}.$$

Thus,

$$[\mathcal{DF}] = \begin{bmatrix} x_{21} & y_{21} \\ x_{31} & y_{31} \end{bmatrix}$$

and

$$\mathcal{J} = det(\mathcal{DF}) = x_{21}y_{31} - x_{31}y_{21}$$

$$[\mathcal{DF}]^{-1} = \frac{1}{\mathcal{J}} \begin{bmatrix} y_{31} & -y_{21} \\ -x_{31} & x_{21} \end{bmatrix}.$$

Now, we have all the ingredients ready for the stiffness matrix calculation. Since the latter is symmetric in our case, we just write its lower part. The integral over $\hat{K}$ leads to the term[10]:

$$[\mathcal{A}_K] = \frac{k}{\mathcal{J}} \int_{\hat{K}} \begin{bmatrix} y_{23}y_{23} + x_{23}x_{23} & \dots & \dots \\ y_{31}y_{23} + x_{31}x_{23} & y_{31}y_{31} + x_{31}x_{31} & \dots \\ -y_{21}y_{23} - x_{21}x_{23} & -y_{21}y_{31} - x_{21}x_{31} & y_{21}y_{21} + x_{21}x_{21} \end{bmatrix} d\hat{K},$$

which, since $\int_{\hat{K}} d\hat{K} = \frac{1}{2}$, reduces to

$$[\mathcal{A}_K] = \frac{k}{2\mathcal{J}} \begin{bmatrix} y_{23}y_{23} + x_{23}x_{23} & \dots & \dots \\ y_{31}y_{23} + x_{31}x_{23} & y_{31}y_{31} + x_{31}x_{31} & \dots \\ -y_{21}y_{23} - x_{21}x_{23} & -y_{21}y_{31} - x_{21}x_{31} & y_{21}y_{21} + x_{21}x_{21} \end{bmatrix}.$$

The integral over $\partial\hat{K}$ leads to boundary contributions if $\partial\hat{K}$ is a boundary edge where g acts. Let us assume that edge a_1a_2 is such a boundary edge, then the corresponding contribution is (g being assumed to be constant per element edge)

$$[\mathcal{A}_K] = g\,\mathcal{J}_\Gamma \int_0^1 \begin{bmatrix} (1-\hat{x})^2 & \dots & \dots \\ \hat{x}(1-\hat{x}) & \hat{x}^2 & \dots \\ 0 & 0 & 0 \end{bmatrix} d\hat{x}.$$

The element of integration is $\mathcal{J}_\Gamma d\hat{x}$. Indeed, for this edge, we have $a = a_1 + \hat{x}a_{21}$ where a is a point on edge a_1a_2. Then we have $x = x_1 + \hat{x}x_{21}$ and $y = y_1 + \hat{x}y_{21}$. Then

$$d\partial K = \sqrt{dx^2 + dy^2} = \sqrt{x_{21}^2 d\hat{x}^2 + y_{21}^2 d\hat{x}^2} = \sqrt{x_{21}^2 + y_{21}^2}\, d\hat{x},$$

[10]The conductivity is assumed to be isotropic (and constant per element), then $[\mathcal{K}]$ acts through the single coefficient k in all matrix coefficients. The general case where $[\mathcal{K}]$ is not isotropic results in a similar calculation provided several different values of k are employed in the appropriate terms.

and, finally, if $\mathcal{J}_\Gamma = \sqrt{x_{21}^2 + y_{21}^2}$ we have $d\partial K = \mathcal{J}_\Gamma d\hat{x}$. Using an exact quadrature leads to

$$[\mathcal{A}_K] = g\,\mathcal{J}_\Gamma \begin{bmatrix} \frac{1}{3} & \dots & \dots \\ \frac{1}{6} & \frac{1}{3} & \dots \\ 0 & 0 & 0 \end{bmatrix}.$$

Similarly, provided they are in the adequate part of the boundary, the two other edges contribute in the same way. We have $\mathcal{J}_\Gamma = \sqrt{x_{32}^2 + y_{32}^2}$ (edge a_2a_3) or $\mathcal{J}_\Gamma = \sqrt{x_{31}^2 + y_{31}^2}$ (edge a_3a_1) and the corresponding contributions are successively

$$[\mathcal{A}_K] = g\,\mathcal{J}_\Gamma \begin{bmatrix} 0 & \dots & \dots \\ 0 & \frac{1}{3} & \dots \\ 0 & \frac{1}{6} & \frac{1}{3} \end{bmatrix}$$

and

$$[\mathcal{A}_K] = g\,\mathcal{J}_\Gamma \begin{bmatrix} \frac{1}{3} & \dots & \dots \\ 0 & 0 & \dots \\ \frac{1}{6} & 0 & \frac{1}{3} \end{bmatrix}.$$

Now the complete elementary stiffness matrix is the sum of the above contributions.

We turn now to the right-hand side calculation. For the term over K, we consider a quadrature formula based on the three element vertices, then, using $\hat{K}$, we have

$$\int_{\hat{K}} {}^t[\mathcal{P}]F\mathcal{J}d\hat{K} = \frac{\mathcal{J}F}{6}\sum_{l=1}^{3} {}^t[\mathcal{P}(\hat{x}_l, \hat{y}_l)],$$

which leads to the term:

$$\frac{\mathcal{J}F}{6}\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \text{ or } \frac{\mathcal{J}}{6}\begin{bmatrix} F_1 \\ F_2 \\ F_3 \end{bmatrix},$$

depending on whether F is assumed to be constant per element or used via its values at the element vertices. The boundary term is simply:

$$f\mathcal{J}_\Gamma \int_0^1 \begin{bmatrix} (1-\hat{x}) \\ \hat{x} \\ 0 \end{bmatrix} d\hat{x} = f\mathcal{J}_\Gamma \begin{bmatrix} \frac{1}{2} \\ \frac{1}{2} \\ 0 \end{bmatrix} \text{ or } \mathcal{J}_\Gamma \begin{bmatrix} \frac{f_1}{2} \\ \frac{f_2}{2} \\ 0 \end{bmatrix},$$

for edge a_1a_2 with $\mathcal{J}_\Gamma = \sqrt{x_{21}^2 + y_{21}^2}$. In the first solution we have assumed f to be constant per edge and an exact integration while in the second solution we have used a quadrature (based on the two edge endpoints) and f_1 (resp. f_2) are the corresponding values of f. For the two other edges, in the case where they contribute, we obtain a similar contribution, for instance,

$$\mathcal{J}_\Gamma \begin{bmatrix} 0 \\ \frac{f_2}{2} \\ \frac{f_3}{2} \end{bmatrix},$$

for edge a_2a_3 with now: $\mathcal{J}_\Gamma = \sqrt{x_{32}^2 + y_{32}^2}$.

The linear $T6$ triangle

This section deals only with the affine triangle, and the isoparametric case will be discussed in the next section. In this example, we have $d = 2$ and we consider the so-called $T6$ triangle as a finite element. For both triangles (affine or curved), we have $N = 6$ and:

$$[\hat{P}] = \{(1-\hat{x}-\hat{y})(1-2\hat{x}-2\hat{y})\,,\hat{x}(2\hat{x}-1)\,,\hat{y}(2\hat{y}-1)\,,4\hat{x}(1-\hat{x}-\hat{y})\,,4\hat{x}\hat{y}\,,4\hat{y}(1-\hat{x}-\hat{y})\,\}\,.$$

Then $[\hat{\mathcal{DP}}]$ is now:

$$[\hat{\mathcal{DP}}] = \begin{bmatrix} -3+4\hat{x}+4\hat{y} & 4\hat{x}-1 & 0 & 4-8\hat{x}-4\hat{y} & 4\hat{y} & -4\hat{y} \\ -3+4\hat{x}+4\hat{y} & 0 & 4\hat{y}-1 & -4\hat{x} & 4\hat{x} & 4-4\hat{x}-8\hat{y} \end{bmatrix},$$

hence, $[\hat{\mathcal{DP}}]$ is now a function of $\hat{x}$ and $\hat{y}$. For convenience, we write this fact as:

$$[\hat{\mathcal{DP}}] = [\hat{\mathcal{DP}}(\hat{x},\hat{y})]\,.$$

For this affine case, the mapping function is the same as for the $T3$ element, i.e.,

$$F_K(\hat{x},\hat{y}) = \{a_1\} + \hat{x}\,\{a_{21}\} + \hat{y}\,\{a_{31}\}\,.$$

Thus, $[\mathcal{DF}]$, $\mathcal{J}$ and $[\mathcal{DF}]^{-1}$ are as in the $T3$ element. The general expression of the stiffness matrix, for element K, is:

$$[\mathcal{A}_K] = \int_{\hat{K}} {}^t[\hat{\mathcal{DP}}(\hat{x},\hat{y})]\,{}^t[\mathcal{DF}^{-1}][\mathcal{K}][\mathcal{DF}^{-1}][\hat{\mathcal{DP}}(\hat{x},\hat{y})]\mathcal{J}d\hat{K} + \int_{\partial\hat{K}} {}^t[\hat{P}]g[\hat{P}]\mathcal{J}_\Gamma d\partial K\,.$$

In the first term, ${}^t[\mathcal{DF}^{-1}][\mathcal{K}][\mathcal{DF}^{-1}]$ is constant. Indeed, if $[\mathcal{FKF}]$ stands for this 2×2 matrix, we have:

$$[\mathcal{FKF}] = \frac{k}{\mathcal{J}^2}\begin{bmatrix} y_{31}y_{31}+x_{31}x_{31} & -y_{21}y_{31}-x_{21}x_{31} \\ -y_{21}y_{31}-x_{21}x_{31} & y_{21}y_{21}+x_{21}x_{21} \end{bmatrix}.$$

Thus, the first term of the stiffness matrix is:

$$\int_{\hat{K}} {}^t[\hat{\mathcal{DP}}(\hat{x},\hat{y})][\mathcal{FKF}][\hat{\mathcal{DP}}(\hat{x},\hat{y})]\mathcal{J}d\hat{K}\,,$$

and, although it is tedious, an exact integration can be used. The terms we have to compute are of the form:

$$\int_{\hat{K}} \frac{\partial p_i}{\partial x}\frac{\partial p_j}{\partial x}d\hat{K}\,,\ \int_{\hat{K}} \frac{\partial p_i}{\partial x}\frac{\partial p_j}{\partial y}d\hat{K} \text{ and } \int_{\hat{K}} \frac{\partial p_i}{\partial y}\frac{\partial p_j}{\partial y}d\hat{K}\,,$$

for $i = 1,6$ and $j = 1,6$ while we have:

$$\int_{\hat{K}} \dots\ d\hat{K} = \int_{\hat{x}=0}^{1}\int_{\hat{y}=0}^{1-\hat{x}} \dots\ d\hat{x}d\hat{y}\,.$$

Thus we need to compute 108 integrals of the above type. For example, we have:

$$\int_{\hat{K}} \frac{\partial p_1}{\partial x}\frac{\partial p_1}{\partial x} d\hat{K} = \frac{1}{2} \;, \int_{\hat{K}} \frac{\partial p_1}{\partial x}\frac{\partial p_2}{\partial x} d\hat{K} = \frac{1}{6} \;, \ldots$$

On completion, we easily find the contribution of K to the stiffness matrix.

To complete this matrix, we need to compute the second part of the general expression. Let us examine the case where edge a_1a_2 contributes in this boundary contribution. First, the element of integration is as it was for the $T3$ triangle. Indeed, we have $d\partial K = \mathcal{J}_\Gamma d\partial\hat{K}$ where $\mathcal{J}_\Gamma$ depends on the edge under consideration. Then, we obtain (omitting symbol $\hat{}$)

$$g\,\mathcal{J}_\Gamma \int_0^1 \begin{bmatrix} (1-x)^2(1-2x)^2 & . & . & . & . & . \\ x(x-1)(1-2x)^2 & x^2(2x-1)^2 & . & . & . & . \\ 0 & 0 & 0 & . & . & . \\ 4x(1-x)^2(1-2x) & 4x^2(1-x)(2x-1) & 0 & 16x^2(1-x)^2 & . & . \\ 0 & 0 & 0 & 0 & 0 & . \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} dx\,,$$

which, with $\mathcal{J}_\Gamma = \sqrt{x_{21}x_{21} + y_{21}y_{21}}$ and since an exact quadrature can be used, reduces to:

$$g\,\mathcal{J}_\Gamma \begin{bmatrix} \frac{2}{15} & \ldots & \ldots & \ldots & \ldots & \ldots \\ -\frac{1}{30} & \frac{2}{15} & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & 0 & \ldots & \ldots & \ldots \\ \frac{1}{15} & \frac{1}{15} & 0 & \frac{8}{15} & \ldots & \ldots \\ 0 & 0 & 0 & 0 & 0 & \ldots \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Now the complete elementary stiffness matrix is the sum of the two above contributions. Note that boundary contribution of edges other than that used here are dealt with in the same way as for the $T3$ element. For edge a_2a_3, we have: $\mathcal{J}_\Gamma = \sqrt{x_{32}x_{32} + y_{32}y_{32}}$ and the possible contribution is:

$$g\,\mathcal{J}_\Gamma \begin{bmatrix} 0 & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & \frac{2}{15} & \ldots & \ldots & \ldots & \ldots \\ 0 & -\frac{1}{30} & \frac{2}{15} & \ldots & \ldots & \ldots \\ 0 & 0 & 0 & 0 & \ldots & \ldots \\ 0 & \frac{1}{15} & \frac{1}{15} & 0 & \frac{8}{15} & \ldots \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

while edge a_1a_3 possibly contributes through the term:

$$g\,\mathcal{J}_\Gamma \begin{bmatrix} \frac{2}{15} & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & \ldots & \ldots & \ldots & \ldots \\ -\frac{1}{30} & 0 & \frac{2}{15} & \ldots & \ldots & \ldots \\ 0 & 0 & 0 & 0 & \ldots & \ldots \\ 0 & 0 & 0 & 0 & 0 & \ldots. \\ \frac{1}{15} & 0 & \frac{1}{15} & 0 & 0 & \frac{8}{15} \end{bmatrix},$$

with $\mathcal{J}_\Gamma = \sqrt{x_{31}x_{31} + y_{31}y_{31}}$ as can be easily seen.

The right-hand side comprises two possible terms. The term over K is:

$$\frac{\mathcal{J}F}{6}\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{bmatrix} \text{ or } \frac{\mathcal{J}}{6}\begin{bmatrix} 0 \\ 0 \\ 0 \\ F_1 \\ F_2 \\ F_3 \end{bmatrix},$$

if the quadrature used is based on the three mid-edge vertices and F is assumed to be constant per element or used via its three relevant values. The boundary term (for instance, for edge a_1a_2) is computed using a quadrature based on the edge endpoints and its midpoint, thus we have (with evident notation for $f_.$):

$$\frac{\mathcal{J}_\Gamma}{6}\begin{bmatrix} f_1 \\ f_2 \\ 0 \\ 4\,f_{12} \\ 0 \\ 0 \end{bmatrix},$$

with $\mathcal{J}_\Gamma = \sqrt{x_{21}x_{21} + y_{21}y_{21}}$ as above.

The isoparametric $T6$ triangle

We now turn to an isoparametric $T6$ element. Basically the same approach can be retained. In specific $[\mathcal{P}]$, the basis polynomials, are those of the affine case. However, since $F_K(\hat{x},\hat{y}) = \sum_{i=1}^{6} p_i(\hat{x},\hat{y})\,\{a_i\}$, the matrix $[\mathcal{DF}]$ is no longer constant over K. Thus both $\mathcal{J}$ and $[\mathcal{DF}^{-1}]$ are functions of $\hat{x}$ and $\hat{y}$. Consequently a numerical quadrature must be employed and we obtain, as a K contribution:

$$\sum_{l=1}^{q} \omega_l\, {}^t[\hat{\mathcal{DP}}(\hat{x}_l,\hat{y}_l)][\mathcal{FKF}(\hat{x}_l,\hat{y}_l)][\hat{\mathcal{DP}}(\hat{x}_l,\hat{y}_l)]\mathcal{J}(\hat{x}_l,\hat{y}_l)\,.$$

Using the three mid-edges as quadrature nodes we have to compute $[\mathcal{FKF}]$, $[\hat{\mathcal{DP}}]$ and $\mathcal{J}$ at these nodes. Following:

$$F_K(\hat{x},\hat{y}) = \sum_{i=1}^{6} p_i(\hat{x},\hat{y})\,\{a_i\}\,,$$

we find:

$$[\mathcal{DF}] = \begin{bmatrix} -3+4x+4y & 4x-1 & 0 & 4-8x-4y & 4y & -4y \\ -3+4x+4y & 0 & 4y-1 & -4x & 4x & 4-4x-8y \end{bmatrix}\{a_i\}\,,$$

then, for the node $(\frac{1}{2},0)$ we find:

$$[\mathcal{DF}(\frac{1}{2},0)] = \begin{bmatrix} -1 & 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & -1 & -2 & 2 & 2 \end{bmatrix} \{a_i\},$$

which reduces to:

$$[\mathcal{DF}(\frac{1}{2},0)] = \begin{bmatrix} -x_1 + x_2 & -y_1 + y_2 \\ -x_1 - x_3 - 2x_4 + 2x_5 + 2x_6 & -y_1 - y_3 - 2y_4 + 2y_5 + 2y_6 \end{bmatrix}$$

$$[\mathcal{DF}(\frac{1}{2},0)] = \begin{bmatrix} x_{21} & y_{21} \\ 2x_{54} + x_{61} + x_{63} & 2y_{54} + y_{61} + y_{63} \end{bmatrix}$$

and

$$[\mathcal{DF}(\frac{1}{2},0)^{-1}] = \begin{bmatrix} 2y_{54} + y_{61} + y_{63} & -y_{21} \\ -(2x_{54} + x_{61} + x_{63}) & x_{21} \end{bmatrix},$$

whose determinant is $\mathcal{J}(\frac{1}{2},0)$. Similarly, and left as an exercise, we can obtain $[\mathcal{DF}(\frac{1}{2},\frac{1}{2})^{-1}]$ and $\mathcal{J}(\frac{1}{2},\frac{1}{2})$ as well as $[\mathcal{DF}(0,\frac{1}{2})^{-1}]$ and $\mathcal{J}(0,\frac{1}{2})$. From these quantities, we have $[\mathcal{FKF}]$ ready at the three quadrature nodes. Similarly, we compute the $[\hat{\mathcal{DP}}]$s involved in the formula. We find successively:

$$[\hat{\mathcal{DP}}(\frac{1}{2},0)] = \begin{bmatrix} -1 & 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & -1 & -2 & 2 & 2 \end{bmatrix},$$

$$[\hat{\mathcal{DP}}(\frac{1}{2},\frac{1}{2})] = \begin{bmatrix} 1 & 1 & 0 & -2 & 2 & -2 \\ 1 & 0 & 1 & -2 & 2 & -2 \end{bmatrix},$$

$$[\hat{\mathcal{DP}}(0,\frac{1}{2})] = \begin{bmatrix} -1 & -1 & 0 & 2 & 2 & -2 \\ -1 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}.$$

Now using the above $[\hat{\mathcal{DP}}(\hat{x}_l,\hat{y}_l)]$s, we have all the ingredients required to compute the contribution of element K to the stiffness matrix.

Let us now consider the contribution of an edge part of the boundary which contributes to the stiffness matrix. We want to use a quadrature whose nodes are the edge endpoints along with the edge midpoints. Let a_1a_2 be this edge. If a stands for a point on this edge, we have

$$a = F_K(\hat{x},\hat{y}) = (1-\hat{x})(1-2\hat{x})a_1 + \hat{x}(2\hat{x}-1)a_2 + 4\hat{x}(1-\hat{x})a_4,$$

and, for instance,

$$x = (1-\hat{x})(1-2\hat{x})x_1 + \hat{x}(2\hat{x}-1)x_2 + 4\hat{x}(1-\hat{x})x_4,$$

thus,

$$dx = \{(4\hat{x}-3)x_1 + (4\hat{x}-1)x_2 + \hat{x}(4-8\hat{x})x_4\}d\hat{x},$$

and similar expressions hold for y and dy. For $\hat{x}=0$, this reduces to:

$$dx = (-3x_1 - x_2 + 4x_4)d\hat{x} = (3x_{41} + x_{42})d\hat{x}$$

and we have, in this first quadrature node,

$$\mathcal{J}_\Gamma = \mathcal{J}_\Gamma(0) = \sqrt{(3x_{41} + x_{42})^2 + (3y_{41} + y_{42})^2}\,.$$

The same calculation for the midpoint $\hat{x} = \frac{1}{2}$ and for the other endpoint, $\hat{x} = 1$, leads to a similar expression for the corresponding $\mathcal{J}_\Gamma$. We obtain successively:

$$\mathcal{J}_\Gamma(\frac{1}{2}) = \sqrt{x_{21}^2 + y_{21}^2} \text{ and}$$

$$\mathcal{J}_\Gamma(1) = \sqrt{(x_{14} + 3x_{24})^2 + (y_{14} + 3y_{24})^2}\,.$$

Now, the boundary contribution is obtained based on these three quadrature nodes. We have:

$$\int_{\partial\hat{K}} {}^t[\mathcal{P}]g[\mathcal{P}]\mathcal{J}_\Gamma d\hat{x} = \sum_{l=1}^{3} \omega_l\, {}^t[\mathcal{P}(\hat{x}_l, 0)]g(\hat{x}_l, 0)[\mathcal{P}(\hat{x}_l, 0)]\mathcal{J}_\Gamma(\hat{x}_l, 0)\,,$$

with $\omega_l = (\frac{1}{6}, \frac{4}{6}, \frac{1}{6})$, this leads to:

$$[\mathcal{A}_K] = \frac{1}{6}\begin{bmatrix} g_1\mathcal{J}_\Gamma(0) & \dots & \dots & \dots & \dots & \dots \\ 0 & g_2\mathcal{J}_\Gamma(1) & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & \dots & \dots \\ 0 & 0 & 0 & 4g_{12}\mathcal{J}_\Gamma(\frac{1}{2}) & \dots & \dots \\ 0 & 0 & 0 & 0 & 0 & \dots \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

The contribution of the two other edges, if relevant, is obtained in a similar way. We find successively:

$$[\mathcal{A}_K] = \frac{1}{6}\begin{bmatrix} 0 & \dots & \dots & \dots & \dots & \dots \\ 0 & g_2\mathcal{J}_\Gamma(0) & \dots & \dots & \dots & \dots \\ 0 & 0 & g_3\mathcal{J}_\Gamma(1) & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & 0 & \dots \\ 0 & 0 & 0 & 0 & 4g_{23}\mathcal{J}_\Gamma(\frac{1}{2}) & \dots \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

where

$$\mathcal{J}_\Gamma(0) = \sqrt{(3x_{52} + x_{53})^2 + (3y_{52} + y_{53})^2}\,.$$

$$\mathcal{J}_\Gamma(\frac{1}{2}) = \sqrt{x_{32}^2 + y_{32}^2} \text{ and}$$

$$\mathcal{J}_\Gamma(1) = \sqrt{(x_{25} + 3x_{35})^2 + (y_{25} + 3y_{35})^2}\,,$$

and, for the last edge:

$$[\mathcal{A}_K] = \frac{1}{6}\begin{bmatrix} g_1\mathcal{J}_\Gamma(0) & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & \dots & \dots & \dots \\ 0 & 0 & g_3\mathcal{J}_\Gamma(1) & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & 0 & \dots \\ 0 & 0 & 0 & 0 & 0 & 4g_{13}\mathcal{J}_\Gamma(\frac{1}{2}) \end{bmatrix},$$

where now,

$$\mathcal{J}_\Gamma(0) = \sqrt{(3x_{61} + x_{63})^2 + (3y_{61} + y_{63})^2}\,.$$

$$\mathcal{J}_\Gamma(\frac{1}{2}) = \sqrt{x_{31}^2 + y_{31}^2} \text{ and}$$

$$\mathcal{J}_\Gamma(1) = \sqrt{(x_{16} + 3x_{36})^2 + (y_{16} + 3y_{36})^2}\,.$$

To complete this example, we have to compute the right-hand side. The term over K is computed based on a quadrature formula using as nodes the three vertices and the three edge midpoints. Thus, we have:

$$[\mathcal{RHS}_K] = \int_{\hat{K}} {}^t[\mathcal{P}]F\mathcal{J}d\hat{K} = \frac{1}{6}\{\sum_{l=1}^{q} {}^t[\mathcal{P}(\hat{x}_l, \hat{y}_l)]F(\hat{x}_l, \hat{y}_l)\mathcal{J}(\hat{x}_l, \hat{y}_l)\}\,,$$

which, with evident notations, reduces to:

$$[\mathcal{RHS}_K] = \frac{1}{6}\begin{bmatrix} F_1\mathcal{J}(0,0) \\ F_2\mathcal{J}(1,0) \\ F_3\mathcal{J}(0,1) \\ F_{12}\mathcal{J}(\frac{1}{2},0) \\ F_{23}\mathcal{J}(\frac{1}{2},\frac{1}{2}) \\ F_{13}\mathcal{J}(0,\frac{1}{2}) \end{bmatrix}.$$

Nevertheless, the above formula is too poor and a more precise quadrature must be used. It is also based on the six nodes in K and, after [Glowinski-1973], it results in the following contribution:

$$[\mathcal{RHS}_K] = \frac{1}{360}\begin{bmatrix} 6F_1\mathcal{J}(0,0) - (F_2\mathcal{J}(1,0) + F_3\mathcal{J}(0,1)) - 4F_{23}\mathcal{J}(\frac{1}{2},\frac{1}{2}) \\ 6F_2\mathcal{J}(1,0) - (F_1\mathcal{J}(0,0) + F_3\mathcal{J}(0,1)) - 4F_{13}\mathcal{J}(0,\frac{1}{2}) \\ 6F_3\mathcal{J}(0,1) - (F_1\mathcal{J}(0,0) + F_2\mathcal{J}(1,0)) - 4F_{12}\mathcal{J}(\frac{1}{2},0) \\ 32F_{12}\mathcal{J}(\frac{1}{2},0) + 16\,(F_{23}\mathcal{J}(\frac{1}{2},\frac{1}{2}) + F_{13}\mathcal{J}(0,\frac{1}{2})) - 4F_3\mathcal{J}(0,1) \\ 32F_{23}\mathcal{J}(\frac{1}{2},\frac{1}{2}) + 16\,(F_{13}\mathcal{J}(0,\frac{1}{2}) + F_{12}\mathcal{J}(\frac{1}{2},0)) - 4F_1\mathcal{J}(0,0) \\ 32F_{13}\mathcal{J}(0,\frac{1}{2}) + 16\,(F_{12}\mathcal{J}(\frac{1}{2},0) + F_{23}\mathcal{J}(\frac{1}{2},\frac{1}{2})) - 4F_2\mathcal{J}(1,0) \end{bmatrix}.$$

The RHS boundary contributions, if any, are computed using a quadrature formula based on the two edge endpoints and the edge midpoint. For edge a_1a_2, we have:

$$[\mathcal{RHS}_K] = \int_{\partial\hat{K}} {}^t[\mathcal{P}]f\mathcal{J}_\Gamma d\partial\hat{K}$$

$$= \frac{1}{6}\{{}^t[\mathcal{P}(0,0)]f_1\mathcal{J}_\Gamma(0) + 4\,{}^t[\mathcal{P}(\frac{1}{2},0)]f_{12}\mathcal{J}_\Gamma(\frac{1}{2}) + {}^t[\mathcal{P}(1,0)]f_2\mathcal{J}_\Gamma(1)\}\,,$$

which reduces to:

$$[\mathcal{RHS}_K] = \frac{1}{6}\begin{bmatrix} f_1\mathcal{J}_\Gamma(0) \\ f_2\mathcal{J}_\Gamma(1) \\ 0 \\ 4\,f_{12}\mathcal{J}_\Gamma(\frac{1}{2}) \\ 0 \\ 0 \end{bmatrix},$$

where the $\mathcal{J}_\Gamma$s are identical to those used for the boundary contribution of the same edge to the stiffness matrix. Other edge contributions are obtained in the same way.

Exercise 20.1 *Based on the previous examples, compute the stiffness matrix and the right-hand side for a PDE modeling a two-dimensional elasticity problem (provided a small deformation problem with planar stress). Hint: we face a problem with two degrees of freedom per node where can be easily expressed using the previous quantities (polynomials, mapping, Jacobian, etc.; see [Ciarlet-1986], [Ciarlet-1988]).*

Element quantities and data structure

In the previous finite element examples we have seen how to compute a stiffness matrix and a right-hand side. Based on these calculations, some practical details about what a mesh data structure could be have been implicitly introduced. Indeed, a mesh data structure must store any information which is required to make the desired calculation possible. In this respect, we have used various information, including:

- the vertex (node) numbering,
- the vertex (node) coordinates,
- the classification of the mesh elements in terms of sub-domains (or materials),
- the classification of the mesh boundaries (mesh edges in two dimensions) in terms of the part of the boundary in which they are located,
- the classification of the mesh vertices (nodes) with the same objective (in our example, to easily find the nodes subjected to a Dirichlet condition).

Numbering and coordinates are natural entries. The problem of membership is a mesh entity classification problem.

Provided with this information, both relevant calculations are possible while using adequate physical coefficients. In what follows, we give some details about these issues (restricting ourselves to a two-dimensional problem).

Vertex (node) coordinates. The node coordinates are obviously used when computing such or such elementary quantities. Note that provided with the coordinates of the three vertices of the affine $T6$ triangle, it is easy to retrieve the coordinates of the mid-edge nodes. Thus, it is not strictly necessary to store these values in the mesh data structure. On the other hand, the 6 pairs of coordinates must be stored for an isoparametric $T6$ triangle if we want to avoid calculating on the fly which would also require a geometric description of the domain boundaries.

Vertex (node) numbering. The (global) node numbering allows us to complete the global matrix and the global right-hand side of the discrete system (see below).

Classification of the mesh entities. The point is to know to which material belongs a given element. In this way, it is possible to assign the right material coefficient to this element. Similarly, we have to know which part of the boundary an element edge belongs to (provided it is not internal). Based on this knowledge, the contribution to the stiffness matrix or to the right-hand side is possible while assigning appropriate coefficients.

It is then desirable to associate an integer value[11] with the elements in the mesh (serving as material number or material pointer) and, similarly, an integer value (pointer) with all faces, edges and nodes for each element [Shephard-1988], [Beall, Shephard-1997].

Note that this data about elements, faces, edges and vertices makes it possible to easily find the information at the node level. Indeed, if a node is also a vertex, there is no problem and if a node is not a vertex it is only necessary to know where it is located. If it belongs to an edge, then its attribute for classification (the above integer) is that of this edge, etc.

Curved boundaries. Following the previous paragraph and based on the same classification, it is easy to see whether or not an edge belongs to a curved boundary. In such a case, the relevant boundary is identified and the node(s), if any, that must be constructed are created on this entity.

Mesh data structure. After the above discussion, the mesh data structure must be designed with these objectives in mind. In this respect, return to Chapter 1 and see the section about the external structure.

Integrals, quadratures and global system

As seen above, it is sometimes possible, given some assumptions about the nature of the data used (about what are the polynomials, the complexity of the operators involved in the problem, the case where the data values are constant per element, for example), to compute exactly some of the integrals involved in the quantity sought. This is, nevertheless, not the case in general. Thus, making use of numerical scheme is generally required.

As seen and based on the degree of the functions in space P_K and on what needs to be computed, a particular formula must be retained. The objective is to make sure that the error due to the quadrature is "consistent" with regard to the interpolation error of the finite element itself.

Usually, the numerical scheme for quadrature is chosen in such a way as to exactly integrate a polynomial of degree n, where n is defined in accordance with the finite element, of the order of the expression resulting from the development

[11] Or pointer or any other equivalent manner.

of the operator and the nature of the data (for example, if the latter are assumed to be constant per entity (element, face, edge, etc.)).

The global system As previously seen, assembling the contributions of the mesh elements takes the following form:

$$[\mathcal{A}] = \sum_{K \in \mathcal{T}} {}^t[\mathcal{B}_K][\mathcal{A}_K][\mathcal{B}_K]\,,$$

to complete the global matrix and:

$$[\mathcal{RHS}] = \sum_{K \in \mathcal{T}} {}^t[\mathcal{B}_K][\mathcal{RHS}_K]\,,$$

to obtain the global right-hand side of the system. In these expressions, matrix $[\mathcal{B}_K]$ related to element K is a boolean matrix (whose coefficients are 0 and 1), which makes it possible to merge the local contribution at the right place in the global quantity.

To specify what matrix $[\mathcal{B}_K]$ is, let us return to the above example with the simplest element, the $T3$ triangle. Thus, $[\mathcal{B}_K]$ is a $N \times M$ matrix where $N = 3$ is the number of degrees of freedom in the $T3$ triangle and M is the total number of degrees of freedom in the mesh[12]. All the coefficients of the matrix are zero except three. Indeed we have $[\mathcal{B}_K]_{1,i_1} = 1\,, \quad [\mathcal{B}_K]_{2,i_2} = 1$ and $[\mathcal{B}_K]_{3,i_3} = 1,$, if i_1, i_2 and i_3 are the global indices of the three degrees in K (i.e., the degree with indices $1, 2$ and 3 in this element).

From a practical point of view, the different matrices $[\mathcal{B}_K]$ do not really exist and a careful writing can be used which luckily avoids their construction (and their storage) while reflecting the mechanism they represent.

20.5 A few examples of popular finite elements

In the following list, we do not seek to be exhaustive, we just want to give several examples of finite elements representative of the various types of element that may be encountered. In this respect, examples of planar, surface and volume elements are given illustrated by different interpolation methods (Lagrange, Hermite, etc.) and different geometrical natures (straight element, curved element).

[12]Including now the degrees corresponding to a Dirichlet condition.

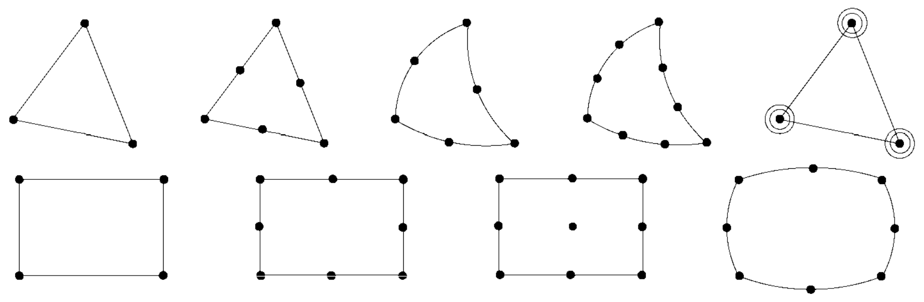


Figure 20.4: *Examples of planar finite elements. $T3$ Lagrange, the three node triangle, $T6$ Lagrange, the straight six node triangle and the isoparametric six node triangle, Lagrange triangle with 10 nodes and Hermite triangle with 10 nodes. Affine $Q4$ and $Q8$ quads and isoparametric $Q8$ and $Q9$ Lagrange element, with respectively four, eight and nine nodes.*

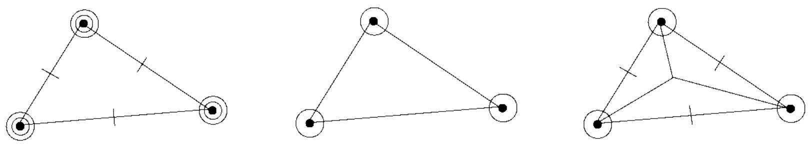


Figure 20.5: *Examples of triangular finite elements for surfaces, plates or shells.*

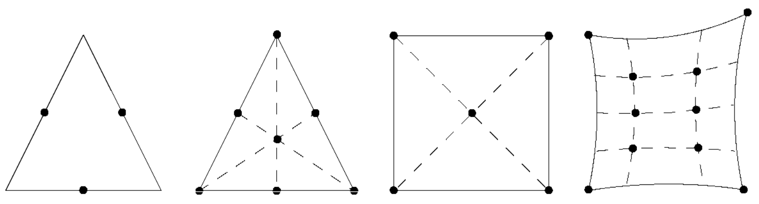


Figure 20.6: *Examples of hybrid finite elements where the nodes are less usual. From left to right, the nodes are the mid-edge points (the vertices not being nodes), the nodes are the vertices and the intersection points of the two diagonals, the nodes are the vertices and the Gauss points.*

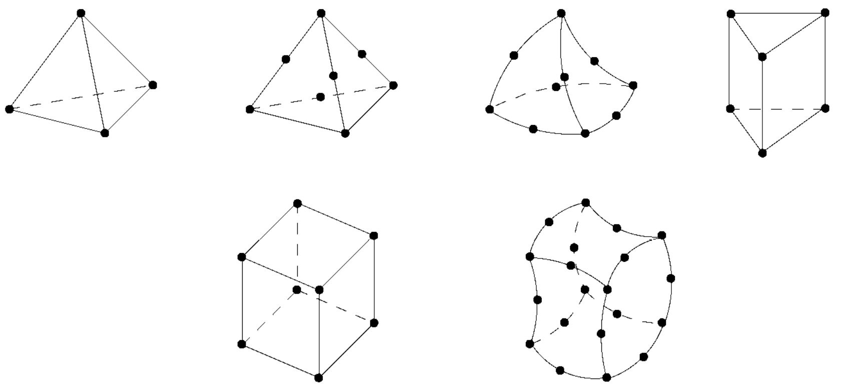


Figure 20.7: *Examples of volume finite elements. The $T4$ tet and the affine and isoparametric $T10$ tets. A pentahedral element with 10 nodes and an isoparametric hex with 20 nodes.*

Chapter 21

Mesh Adaptation and H-methods

In this chapter, we consider mesh construction from the perspective of a finite element style computation. The objective is to introduce some mesh generation methods or mesh modification methods resulting in a mesh that conforms to some pre-specified requirements in terms of sizes (isotropic problem) or in terms of sizes and directional properties (anisotropic problem). The basic principle for adapted mesh construction is to collect information about the sizes, the directions and the related sizes using an adequate data structure, a *control space* (or a background mesh), and then to use this information to construct a mesh conforming (as far as possible) to these specifications.

As pointed out in Chapter 20 for finite element convergence and accuracy, theoretical error estimates involve the parameter h, the size of the mesh elements. Thus, at least for isotropic situations, the necessity of adapting the hs in the mesh is rather natural[1]. On the other hand, anisotropic cases are not so trivial. Actually, numerical experiments (mainly in two dimensions) resulting in nice solutions could lead us to think that h-adaptation is well suited to the problem, and yet theoretical proofs are not fully available at this time (except for simple problems).

Hence, we discuss a mesh adaptation method related to the element size (direction), in other words an *h-method.* This type of method is, as will be seen, the basic ingredient that makes it possible to compute the solution by means of adaptive solution methods. In brief, such a method is an iterative process where, at each iteration step (or after a small number of steps), an adapted mesh is constructed which is used to calculate the solution. Then this solution is analyzed in order to decide whether the iterations should continue or not.

Several categories of h-adaptive methods may be considered. One class of methods is based on the local modification of the current mesh so as to fit the sizing (directional) requirements. Another class of methods involves reconstructing the whole mesh of the domain from its boundary discretization while conforming to

[1]Intuitively, adapting the element sizes, according to the problem, comes down to constructing elements which are smaller or larger in some regions, the size h then being changed from one region to another (problems involving mechanics or fluids, etc.) or to adjust this h while keeping it as constant as possible (for wave propagation problems, for example).

the sizing (directional) requirements. On the other hand, we can find adaptive methods that belong to neither of these above classes. For instance, hierarchic methods, multigrid type methods and non-conforming or overlapping methods are examples of such approaches (and are not covered in this book). Finally, adaptation methods by means of degree adaptation also exist, the so-called *p-methods*, which are discussed in the next chapter.

In what follows we discuss the first two types of methods with special emphasis on the global approach where the mesh is entirely recreated. Indeed, issues discussed throughout the previous chapters are now reviewed with the present objective in mind. This means that most of the aspects covered in this chapter have been already discussed to some extent in various parts of the book[2]. However, to keep the chapter as self-contained as possible, some notions will be recalled within the context of a h-adaptive mesh generation method.

★

★ ★

To this end, this chapter first recalls what a control space is and how it can be used to give the information that is required to control the mesh generation process. In this respect, localization as well as interpolation problems are discussed. Then we turn to h-adaptation by means of the local modification of a given mesh. Afterwards, h-adaptation based on the whole construction of a mesh is presented. Finally, computational schemes suitable for adaptive methods are given, together with details about the various steps involved in these schemes. The metric aspect (i.e., the error estimate problem) is assumed to be known and thus will only be touched upon briefly.

21.1 Control space (background mesh)

A *control space* (Chapter 1) is a structure whose purpose is to govern a mesh modification or a mesh generation process by providing the required information about the adaptation. Such information may be specified in various ways. Among these, and based on the adaptive strategy, we can find element refinement (coarsening) demands, desired sizes (or densities) or desired directions and related directional sizes.

A demand at the element level mainly corresponds to an isotropic adaptation problem addressed by means of local mesh modifications. On the other hand, isotropic or anisotropic specifications expressed in terms of sizes, densities, directions, etc., are mostly related to an adaptation problem based on complete reconstruction of the mesh.

In the first case, a list of elements (vertices) is specified which must be processed locally. As an example, if a refinement is demanded (at the element level) then the elements in question are subdivided into smaller elements or, if the demand

[2]Therefore, in principle, it is merely necessary to dip into those parts of the book where the necessary information may be found.

concerns vertices, the elements sharing such a vertex are processed similarly. Then, if h stands for the element size, the refined elements will have a size smaller than their initial h (for instance, $\frac{h}{2}$ if we consider the edge lengths. This leads to a ratio of 4 (or 8) in terms of surface (volume) variation). This means that a subdivision is made if h is too long but irrespective of the initial range of h. In this way, we go from a value h to a value $\frac{h}{2}$ (for example) while this new value, closer to the desired value, may be still too large.

On the other hand, if a size map is specified (generally at the vertices of a mesh serving as a background mesh), then the affected elements must conform to this map, meaning that the hs of the adapted elements must be as close as possible to the specifications. In this way, the adaptation method allows the creation not only of smaller elements (in the case of a refinement as above) but also of elements with a prespecified h.

As seen in Chapter 1, a general control space comprises two kinds of data including a spatial description of the domain and metric information provided in some way based on this spatial description. Thus, a control space is a pair *(background mesh-metric specification)* and, for the sake of simplicity, we may use the term background mesh with the same meaning.

Definition of a background mesh

A background mesh serves as a control space for the mesh modification or generation method.

Background mesh: spatial aspect. As already mentioned, a background mesh, in terms of its spatial aspect, can follow various types.

Let us consider that the current mesh (under modification or construction) is a simplicial mesh[3] (triangular or tetrahedral mesh according to the spatial dimension). Then, following Chapter 1, the background mesh could be:

- a simple (uniform or not) grid (*type 1*),
- a tree-type structure (such as a quadtree or an octree) (*type 2*),
- an arbitrary mesh (*type 3*), for instance, a simplicial mesh in our case.

The point is that the background mesh encloses the domain where the adapted mesh will be established. As will be seen, this property, which is a source of simplicity, does not always hold (see the discussion about localization problems below).

Isotropic background mesh: metric aspect. Based on the type of background mesh and related to the adaptation method, the metric aspect could be defined in various ways. The most popular methods consist of the following:

[3]We mainly consider the case of simplicial meshes as they offer great flexibility in terms of adaptation. Nevertheless, obviously, non-simplicial meshes can also be used for adaptive purposes.

- a refinement (derefinement) demand at the element level,
- a refinement (derefinement) demand at the vertex level,
- a adaptation demand expressed in terms of sizes at the vertex[4] level.

The first two cases are mostly those found in adaptation methods based on the local modification of a given mesh, while the third case is generally that used when the adaptation method consists of completely reconstructing the mesh.

For a local modification based method, the background mesh is, in general, the current mesh. The mesh elements contain the adaptation requests and are processed accordingly (see below).

For a global adaptive method, the background mesh could follow one of the three above types. For a *type 1* background mesh, the cells of the grid contain the adaptation requests. These could be related to the cells themselves (for instance, their sizes or a *scalar* value associated with the cells which indicates the desired h in the corresponding regions or again a *scalar* value defined at the cell vertices which is furthermore used to define the desired h in the related regions). For a *type 2* background mesh, we return to the same discussion. Either the cells (quadrants or octants) have been constructed so as to have a size conforming to the desired sizing specification and these sizes govern the adaptation process or a point representative of the cell (its centroid for instance) or the cell corners support the metric information (a *scalar* value). In the case of a *type 3* background mesh, the most popular method consists of associating a *scalar* value with the element vertices. Then, in this case and for an adaptive loop of computation consisting of several iteration steps, the background mesh at iteration step i could be the current mesh of iteration $i-1$ enriched with the sizing (directional) specifications, these values resulting from an *a posteriori* analysis of the solution in the current mesh.

Anisotropic background mesh: metric aspect. Unlike the isotropic situation, a background mesh for anisotropic mesh control contains information with both directional and sizing aspects. In this respect, a *type 3* background mesh appears to be a suitable solution to define the adaptation directives. Each element vertex of the background mesh is associated with a *tensor* value representing both the desired directions and the related edge sizes in the neighborhood of the vertices.

Such a specification consists of $d \times d$ matrices provided at some points. Such a matrix is denoted, given a point P, as

$$\mathcal{M}(P) = \begin{pmatrix} a_P & b_P \\ b_P & c_P \end{pmatrix}$$

[4]Actually, a demand for element refinement or coarsening can be expressed at the element level (for instance, it can be demanded to refine twice a given element). Nevertheless, it appears to be simpler to have the refinement demand expressed at the element vertex level, i.e., it can be demanded to refine twice around a given vertex. However, these two kinds of formulation are closely related.

if we consider a two dimensional problem (i.e., d the spatial dimension is 2). Above point P could be a mesh vertex when the background mesh is itself a mesh or a cell corner (or again a point representative of a cell) when the background mesh is a grid or a tree structure. From the metric point of view, the reader is referred to Chapter 10 for the meaning of the coefficients involved in the above matrix.

Remark 21.1 *The isotropic case is a peculiar occurrence of the anisotropic situation where a matrix $\mathcal{M}(P)$ reduces to $\frac{\mathcal{I}}{h_P^2}$ where $\mathcal{I}$ stands for the identity matrix and h_P is the size which is desired around point P.*

Remark 21.2 *In essence, the sizing (directional) information contained in a background mesh is of a discrete nature*[5] *which means that attention must be paid on how to obtain consistent continuous information (see below).*

Alternative to a background mesh. Analytical definition of the metric aspect by means of an analytical function is a solution where there is no need for a background mesh to specify the desired metrics. However, in terms of computer implementation, the mesh generation method must be coupled with the corresponding procedure which changes for every new problem.

Defining simple boxes encoding the desired metrics can be (and was) used in some governed mesh adaptation (generation or optimization) methods.

Finally and more flexible is the use of various types of source where a source is a topological entity (point, line, surface) to which is associated a metric information. Then, the metric for an arbitrary location is defined as a (linear or more complex) function of the distance from it to the closest source. Combination of several sources can be envisaged in some location close to several such sources.

Needless to say that, while proved in various papers, these methods which, in some sense, emulate what a background mesh is, are less flexible or general than those making use of a "true" background mesh.

Use of a background mesh

Whatever the nature of the background mesh, the actual mesh modification or (re)construction is governed by this structure to which a series of queries is made so as to collect the required information.

Various problems must be carefully addressed including mainly localization problems and, the localization problems being completed, interpolation problems must also be addressed so as to extract the metric information related to the region under consideration.

Localization problem. We face a localization (or searching or point-location) problem since a frequently demanded operation is to find which element (cell) of the background mesh contains a given point of the current mesh.

[5]Except for rapid testing purposes where analytical sizing (directional) functions can be used for the sake of simplicity and thus where an "ideal" control space is used.

An adaptive method based on the local modification of the current mesh does not lead to difficulties since there is no localization problem (indeed, the localization is implicit). In contrast to this method, a global adaptive method strictly requires the localization of points in the background mesh. As mentioned earlier, the background mesh could be a grid, a quadtree-octree type mesh or an arbitrary simplicial[6] mesh.

The localization of a point in a cell of a grid or a quadtree-octree type background mesh has been discussed in Chapters 2 and 5 and does not lead to serious difficulties.

On the other hand, a simplicial background mesh must be considered in a more subtle way. We can face cases where the current mesh, as well as the background mesh, are not convex, include holes and are not even strictly coincident while they approach the same domain (i.e., the (approximate) domain covered by the background mesh is not exactly that covered by the current mesh due to the fact that the boundary of the current mesh is not meshed in the same way). Thus regions of the current mesh may fall outside the background mesh and vice versa.

Therefore, depending on the (global) mesh generation method, localization or searching problems may be of greater or lesser difficulty. As a first remark, it could be observed that a (uniform or not) grid (or a quadtree-octree structure, see for instance [Krause, Rank-1996]) can be constructed to help the searching operation in a given (arbitrary) mesh. Using such a structure, it is easy and fast to find the cell within which a given point falls.

Then, the point is to find the element(s) of the background mesh corresponding to this box which is an easy task. Actually, each background vertex is associated with a background element. Then, a background point in the box within which the current point falls is found and thus an element is found as well which can serve as initial element for the searching procedure. Given this background element, the searching problem, in general, becomes a very local problem that can be successfully solved using a classical searching method (see Chapter 2). Nevertheless, for non-convex domains, an element in a given box (enclosing the point under consideration) is not necessarily close (topologically speaking) to this point, meaning that a searching procedure initialized with this element may lead to meeting the boundary of the domain. Figure 21.1 illustrates three possible situations that can be encountered in such a localization process.

A natural idea to overcome a difficulty due to a non-convex local situation is to visit the boxes neighboring the initial box in order to reach the right part of the domain, then a classical searching method easily finds the solution. Note that this solution could demand more effort in terms of CPU time.

Remark 21.3 *For a Delaunay-type method, a different solution can be envisaged as it is possible to take advantage of the current mesh which is necessarily convex (return to Chapter 7 where a method based on an enclosing box is discussed for which a convex situation always holds). Thus, the background mesh is convex or*

[6]A non-simplicial mesh can be used as well which results in the same analysis but requires a more precise process. Indeed, splitting the non-simplicial elements in terms of simplices allows us to return to the simplicial case (see Chapter 18) for the searching problem.

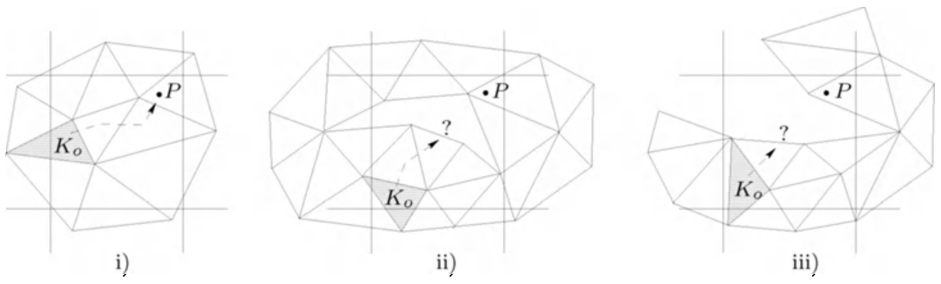


Figure 21.1: *Localization in a two-dimensional domain. We can see one box of the grid associated with the mesh and the mesh elements included or near this box. Three local patterns are visible: i) the local context is convex, ii) a hole exists while point P and element K_0, from which the localization search starts, are separated by this hole, iii) it is demanded to pass through the boundary to reach the element containing point P.*

not but the mesh under construction is convex and a subtle writing of the searching procedure easily results in the desired solution.

Localization problem in a surface mesh. Searching on a surface could be a more tedious problem. Actually, two cases can be considered.

First the surface is the boundary of a domain and a mesh of the latter can be helpful to find the location of a point. In this case, in principle, we return to the previous paragraph where a d-dimensional entity (a point in $\mathbb{R}^3$) must be located in a d-dimensional entity (the background mesh).

In contrast to the previous case, if only a surface mesh is given, we face a localization problem where a $d+1$-dimensional entity (point P) must be located on a d-dimensional domain entity (the surface mesh being seen as a topological entity). It is then easy to construct a grid (or an octree) which encloses the surface mesh. Then, given a point, the cell within which it falls is found[7], after which an element in this cell is selected and a classical searching procedure is used. In this way we return to the same difficulties as in the previous paragraph and the same solutions apply.

Interpolation problem. Once a point of the current mesh has been properly localized in the background mesh, we have to collect the sizing (directional) information related to this point (say, related to a "small" region surrounding the point in question). Since the sizing is known in a discrete way, we have to define an interpolation scheme about discrete values so as to extract consistent information at the considered point (region).

Recall that the sizing information is assumed to be defined at the background mesh vertices, then several interpolation problems are encountered based on the

[7]For the sake of simplicity, we consider that point P lies on a triangle in the surface. If not, the projection of P onto the plane of the triangles must be used in this search.

type of background mesh and the entity within which the point under investigation falls (background vertex, background edge, background face or element or background cell).

First, let us consider an isotropic problem. We assume that the background mesh is a simplicial mesh and that point P has been located in a given element (a triangle in the example figure), then using the available metric data (the hs at the element vertices) and the way in which the h function varies, we want to find h_P the corresponding h at P. Then the first case is obvious, point P is coincident with a background point. The second case can be successfully dealt with using the material discussed in Chapter 10 where various interpolation schemes can be found that interpolate h along an edge from the hs at its two endpoints. Indeed, given the hs at the edge endpoints, the desired h at the given point is obtained based on one of these interpolation functions. The two other cases are more tedious. Using a classical interpolation scheme (between two points) leads to splitting the problem into two parts as can be seen in Figure 21.2 (parts ii) and iii)). First, point P_{12} is found in edge P_1P_2 and the h of this point is obtained by interpolation from those at P_1 and P_2. Then, using segment P_3P_{12}, a solution can be found based on the same type of interpolation. Hence, the scheme is not necessarily commutative and thus the resulting h may depend on the way the known hs are used. To overcome this, see part i) of the same figure, a classical interpolation scheme can be used (for instance, a P^1-type interpolation) and the solution depends on what type of variation is desired. We can use a scheme like:

$$h_P = \omega_1(P)h_1 + \omega_2(P)h_2 + \omega_3(P)h_3 \,,$$

with evident notations or something like:

$$h_P = \left(\frac{\omega_1(P)}{h_1} + \frac{\omega_2(P)}{h_2} + \frac{\omega_3(P)}{h_3} \right)^{-1} .$$

In the case of quad elements, tet elements or other classical elements, the same approach can be retained provided the corresponding interpolation function is used.

Remark 21.4 *Note that a grid or tree-type background mesh can be considered in a similar way. Point P is located on this structure and the hs associated with the box (the cell) including P are interpolated as above (for instance, using a Q^1-type interpolant in the case of a quadrilateral cell; see Chapter 5).*

Obviously, the same problem when an anisotropic context is desired leads to the same kind of discussion. Now, we need to interpolate both the sizing values (the hs) and the corresponding directions. The material in Chapter 10 allows solution when the interpolation is based on two points (as when P belongs to an edge). However, when the interpolation scheme must use three (or more) points (point P falls within a triangle or within a cell), the above scheme is no longer valid.

Heuristics can be used to complete the solution. Let us consider the case where point P falls within a triangle for which the metric information is known at the

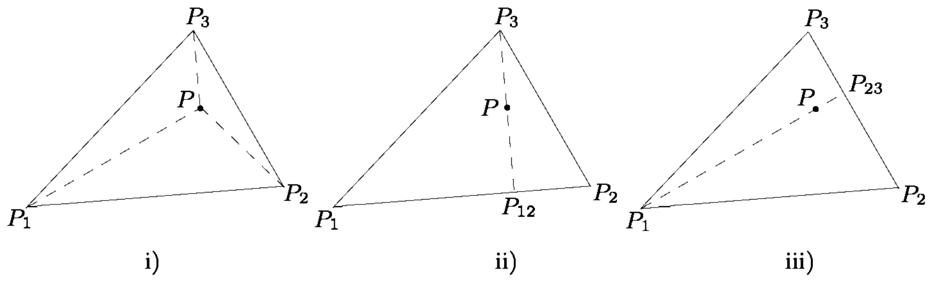


Figure 21.2: *Interpolation in a two-dimensional domain. The three interpolations are equivalent in the case of a linear function while cases ii) and iii) result in a different solution at point P depending on whether P_{12}, constructed using P_1 and P_2, or P_{23}, related to P_2 and P_3, is used in the interpolation scheme.*

three vertices. If $\mathcal{M}_i$, $i = 1, 3$, are the three corresponding metric matrices, then a possible interpolation scheme could be as follows:

- compute $\mathcal{M} = \sum_{i=1}^{3} \omega_i(P)\, \mathcal{M}_i$ where $\omega_i(P)$ is a weight function (that of a P^1-interpolation in this case),
- find the eigenvectors of matrix $\mathcal{M}$, i.e., $e_1, ..., e_d$ (d being the spatial dimension, i.e., $d = 2$ in this example)
- compute $h_i^j = \left(\sqrt{{}^t e_j\, \mathcal{M}_i\, e_j}\right)^{-1}$, for $i = 1, 3$ and $j = 1, d$,
- finally, compute $h^j = \sum_i \omega_i(P)\, h_i^j$, for $j = 1, d$,

then the matrix at P is the matrix whose eigenvectors are the above e_js while the size h^j, related to direction e_j, is that obtained by the above interpolation scheme.

Remark 21.5 *Other interpolation schemes can be developed leading to a different variation in both the directions and the sizes. In this way, emphasis can be placed on a particular entity depending on what is expected.*

21.2 Adaptation by local modifications

In this adaptive strategy, the current mesh is locally modified in order to construct an adapted mesh. Thus, the background mesh is naturally the current mesh and the directives for refinement or coarsening are known at the vertices of this mesh.

Local modification of the elements, the basic ingredient of this adaptive strategy, is mainly based on the general tools discussed in Chapters 17 and 18. Two categories of local modifications can be demanded. One corresponds to element refinement so as to increase the density of the mesh in a given region, while the other corresponds to element coarsening leading to a coarser mesh in a specified region. In the following discussion, we recall some tools that ensure the level of refinement (coarsening) which is demanded while maintaining a conforming mesh.

Due to the local tools involved in this type of adaptive approach, it could be noted that local modification based methods are mainly suitable for *isotropic* adaptation.

• Element refinement or coarsening

Element refinement

Element refinement demands can be expressed either at the element level or at the element vertex level.

Refinement around vertices. In this section, we assume that the refinement demand is expressed at the element vertex level (and not at the element level). In this way, it will be easy to maintain a conforming mesh as will be seen below.

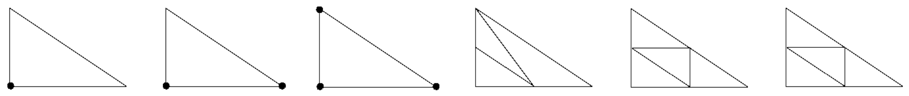


Figure 21.3: *The element must be refined around one, two or three of its vertices. Possible partitions are depicted where only triangles exist.*

Let P be a vertex in the mesh, if a refinement is prescribed around P, then all the edges emanating from P are subdivided into two edges using the edge midpoint. Based on this subdivision, the element is partitioned into sub-elements. The case of a triangular element is shown in Figure 21.3 where, on the left-hand side, the refinement demand concerns one, two or three vertices. On the right-hand side of the figure, the corresponding partitions of the initial triangle can be seen.

Obviously all the elements sharing a point where a refinement is required are subdivided in such a way that a conforming refined mesh is automatically obtained.

Remark 21.6 *When a boundary edge must be subdivided, it is necessary to return to the boundary geometry so as to properly locate the midpoint involved in the process.*

The same procedure for a quad element leads to five cases (Figure 21.4) depending on both the number of affected vertices and their relative locations. Note that this local procedure leads to dividing an affected edge twice, the process is a *2-level* process. A variation, proposed by [Schneiders-1996b], considers different templates as depicted in Figure 21.5. In this case, three subdivisions can be employed based on the number of vertices where a refinement is demanded. Thus, the procedure is a *3-level* procedure. As before, conformity is automatically maintained[8], as can be easily observed in the figure (just consider the cases of adjacent

[8]Given an adequate definition of the templates used to refine a triangle in the case of a mesh comprising triangular and quad elements.

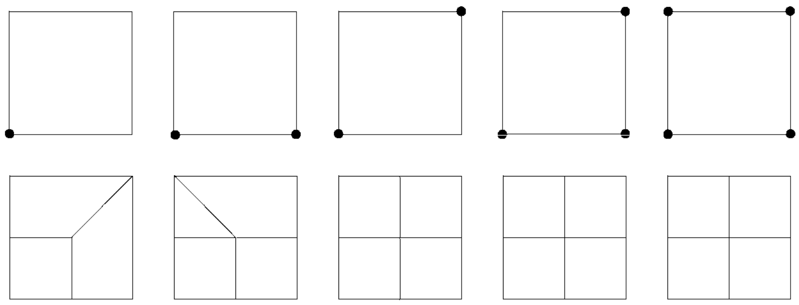


Figure 21.4: *The element must be refined around one, two, three or four of its vertices leading to the partitions depicted. The refinement procedure is a* 2-level *procedure. Note that a triangle is formed in case two (when this is undesirable, the pattern defined for three or four vertices can be used).*

elements and apply the corresponding templates to them).

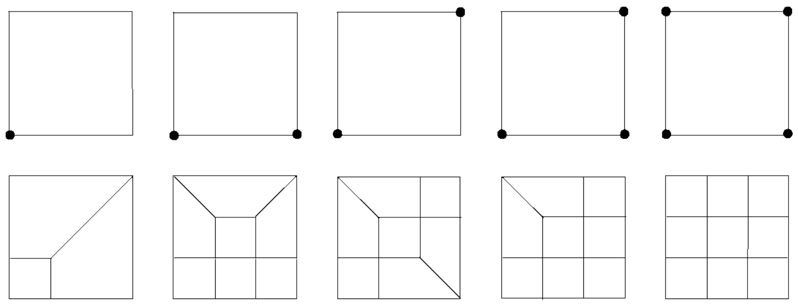


Figure 21.5: *The refinement procedure is now a* 3-level *procedure. Note that no triangles are formed, whatever the case.*

Three-dimensional elements can be dealt with in the same vein. Nevertheless, subdivision procedures are not so obvious. Some examples of possible partitions are given in Figures 21.6 and 21.7.

The templates of Figure 21.6 are now fully described. Let i and j be two vertex indices, we note ij the index of the midpoint of points i and j. With this concise notation, for a given element $[1, 2, 3, 4]$, the four series of templates depicted in the figure correspond to the following vertex enumeration:

- $[1,2,3,4]$ gives $[12,2,23,24]$, $[1,12,3,4]$, $[12,23,3,4]$ and $[12,23,24,4]$ when we refine around vertex 2. Indeed, we define a "small" tet around vertex 2 and, removing this tet, the remaining polyhedron is a pentahedron which, in turn, is split into 3 tets.

- $[1,2,3,4]$ gives $[1,2,3,24]$ and $[1,3,24,4]$ when the edge $[2,4]$ is subdivided.

- $[1,2,3,4]$ gives $[13,1,12,14]$, $[12,2,23,24]$, $[23,3,13,34]$, three "small" tets, along with $[12,23,13,4]$, $[12,13,14,4]$, $[12,24,23,4]$ and $[23,34,13,4]$ when we refine around vertices 1, 2 and 3 meaning that the 6 mid-edges are introduced in the partition.

- [1, 2, 3, 4] gives, introducing again the 6 mid-edges, [13, 1, 12, 14], [12, 2, 23, 24], [23, 3, 13, 34], [34, 4, 14, 24], four "small" tets at the corners and [13, 23, 14, 12], [12, 14, 24, 23], [23, 14, 34, 13] and finally [34, 23, 24, 14].

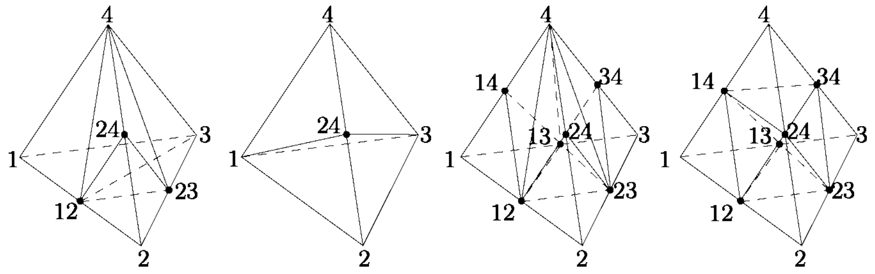


Figure 21.6: *Selected examples of three-dimensional partitions for tetrahedral elements.*

Exercise 21.1 *Show that the first refinement scheme admits an alternate a priori valid subdivision. Hint: examine the remaining prism (on the fly, retrieve and discard the Schönhart case which is no longer valid).*

Exercise 21.2 *Show that the last refinement scheme is only one of the three possible subdivisions of a given tet. Hint: examine the polyhedron formed when the four corner based tets are removed. Look at the six possible diagonals based on the six remaining vertices.*

Exercise 21.3 *Examine how the tet subdivision patterns interact with mesh conformity.*

The templates, as depicted in Figure 21.7, concern the case of a 2-level subdivision. More complex and flexible templates can be found when a 3-level procedure is used. In this case, we return to Figure 21.5 for the patterns corresponding to the face subdivision. Using these templates, no conformity problems may be expected.

Local refinements can be coupled with classical mesh optimization tools (such as node balancing, (generalized) edge swapping, etc.) in order to increase (or to preserve) the mesh quality.

Remark 21.7 *In many of the above examples, it could be noted that if the edge lengths of the initial element are about 1 then the edge lengths of the refined elements range from 1 to $\frac{1}{2}$ (or from 1 to a $\frac{1}{3}$). Thus, it is not so easy to achieve a size variation following a different variation for two adjacent elements.*

Remark 21.8 *Multiple refinements can be easily obtained by repeating the above element partitioning. Nevertheless, a repeated series of refinements may alter the angles in the elements if some care is not taken thus resulting in ill-shaped elements.*

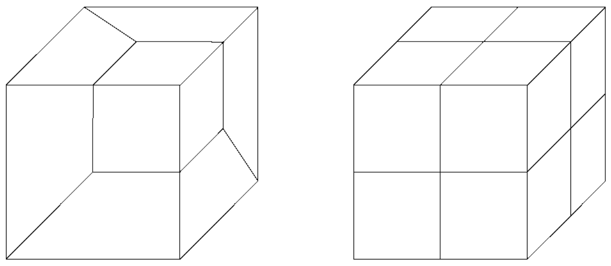


Figure 21.7: *Two examples of three-dimensional partitions for hexahedral elements.*

With regard to what an optimal mesh is (see Chapter 18) and following the two above remarks, it appears that local adaptation methods necessarily induce some extent of rigidity.

Element refinement. In this section, we assume that the refinement demand is expressed at the element level. Thus, maintaining mesh conformity must be made explicitly. Indeed, when an element is refined, it is necessary to propagate the refinement to some neighboring elements if the interfaces (edges or faces) between the refined element and some of its neighbors are affected by the refinement procedure.

Basically, we return to the above templates. However, simplicial elements offer alternative solutions. In this respect, a triangular element can be subdivided by the so-called *longest-side subdivision* method or one of its variations. Advocated in numerous papers including, in a recent issue, [Rivara-1997], the basic idea of this method is to split a triangle by introducing the midpoint of its longest edge and then to propagate (for conformity reasons) the subdivision to some of its neighbors. Nice theoretical issues indicate that the path of propagation is finite and thus mesh conformity is easily obtained and the number of elements still remains reasonable. In addition, this strategy results in a refined mesh where the angles are bounded.

Exercise 21.4 *In two dimensions, show that if a triangle is split by introducing the midpoint of its longest edge, then the two resulting triangles are such that the minimum angle is greater than or equal to the minimum angle in the initial configuration. Therefore, the angles do not alter even when repeating this splitting procedure.*

Also, many variations have been investigated including coupling with the Delaunay criterion. The three-dimensional extension of this idea to tet elements can be found in [Rivara, Levin-1992] and, more generally, there are numerous papers by the same author(s) discussing simplicial mesh refinement.

Remark 21.9 *It could be of interest to store the history of a refinement procedure. This can be achieved by storing the item genealogy. Such information could be useful for the inverse process, i.e., a derefinement procedure, as will be seen below.*

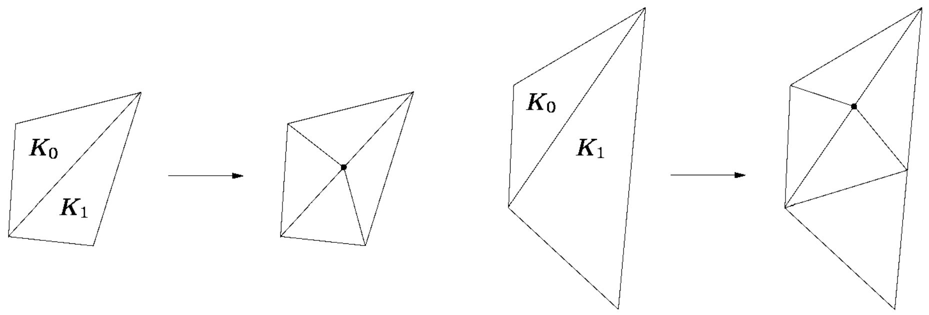


Figure 21.8: *Some occurrences for the longest-side subdivision refinement method in two dimensions. Left: triangle K_0 is candidate for refinement and the edge common to K_0 and its neighbor K_1 is the longest edge of both of these triangles, then the refinement is confined in this pair of elements. Right: triangles K_0 and K_1 are candidates for refinement and the refinement path also propagates to some other (not depicted) elements.*

Element coarsening

Constructing a coarser mesh, based on local modifications of a given mesh, is more tedious. Actually, two approaches can be envisaged that address the derefinement problem in a rather different way. First, the coarsening procedure is seen as an inverse algorithm based on a previously applied refinement algorithm. Second, the coarsening of a mesh is seen as a autonomous procedure that applies for arbitrary meshes however they may have been created.

Derefinement as the inverse of refinement. As it is based on the inverse of a refinement procedure, such an approach is only suitable in regions of the mesh where a refinement has been applied in a previous stage of the computational process. Given a list of candidate nodes for the derefinement procedure, we have to check whether or not they can be removed from the mesh. This point concerns both maintaining mesh conformity and the fact that the genealogy of the investigated node allows the procedure. The genealogy mainly depicts relationships like *children-parent* for the mesh entities (vertices, edges, faces and elements). Based on this history, derefinement consists of rolling backward the refinement procedure which led to the current mesh.

This inverse process either has an immediate effect or otherwise is more subtle to be processed as it must consider not only a transformation (to be processed backward) but also a series of transformations whose inverse implies several operations.

Arbitrary derefinement method. Such a method considers an arbitrary mesh and ignores the way in which it was created. Thus, this derefinement method is based on classical optimization tools (see Chapter 18) that lead to vertex removal (and, consequently, edge, face and element removal). Apart from some peculiar local pathologies (for instance, a vertex in a triangular mesh only shared by three triangles) where an obvious solution can be obtained, the basic tool for mesh coarsening is the edge collapsing operator. Due to this, simplicial meshes are more flexible than other types of meshes as they are more readily candidates for successful edge collapsing.

• R-method

A *r-method* is a method which adapts the sizes (the hs) while maintaining the connections between the mesh vertices, thus this connectivity remains unchanged. The principle lies in moving the mesh vertices, in such a way as to increase or decrease the vertex density in certain regions in the domain in accordance with the behavior of the physical phenomenon in question. This can be seen as a mesh deformation.

While well suited to some classes of problems, this method suffers from a certain degree of rigidity. From a numerical point of view, it must be carefully checked that the deformed elements (once their vertices are moved) remain valid and have an acceptable quality. The deformation process (by attracting the vertices in a given region or by repulsing them) is in general an iterative process, the vertex shifts being processed step by step.

Remark 21.10 *It is often very efficient to combine an r-method with some of the previous methods for mesh enrichment or coarsening.*

Note that this method is a local processing (based on vertex moving) but has a global effect on the mesh.

Comments about local methods

Local modification-based adaptive methods are one solution for completing adapted meshes. Nevertheless, a precise examination of the different points discussed in this section leads to some comments. Indeed, this adaptive approach offers both a series of positive aspects and some weaknesses.

Let us mention first some apparent weaknesses of a local method. Nevertheless, note that a tricky implementation could be, in some cases, one way to overcome some of these weaknesses:

- In essence, local methods appear to be a solution for isotropic adaptive problems and seem unlikely to be suitable when anisotropic features are expected.
- The mesh conformity is more or less automatically insured.
- Mesh derefinement is not so obvious.
- Continuous variation in size is not easy to obtain (indeed, we start with an edge length h and obtain a length $\frac{h}{2}$ and thus obtaining another type of gradation is not so simple. In other words, such an approach is far from optimal when the desired size is not in a ratio 2. or 0.5).

On the other hand, positive features of a local method include the following:

- Localization problems are not an issue as the background mesh is, in general, the mesh itself.
- The use of predefined templates allows an easy implementation.
- Also, templates lead, in principle, to a reduced effort in terms of CPU.

Note that other computer issues, such as boundary management, are common to both the local approach and the global method discussed in the next section and, thus, do not influence the appreciation of the local approach.

Thus, based on these observations, the user must decide whether or not a local method is a suitable solution to a particular problem.

21.3 Global isotropic adaptation method

Automatic mesh generation methods, as discussed in the previous chapters, are natural candidates for completing adapted meshes. Thus, we now discuss how to use or to modify these methods to fulfill this objective.

In Chapters 5, 6 and 7, we have described *quadtree-octree*, advancing-front and Delaunay-type methods. In what follows, we successively return to each of these methods to see if they can be suitably used for the construction of meshes conforming to a metric map specified in advance.

Quadtree-octree based methods can be used to obtain graded (isotropic) meshes as the quadtree-octree structure can be constructed in such a way as to conform to some sizing properties. In this case, the control space could be the quadtree-octree structure itself provided a suitable criterion has been used to complete this structure.

Advancing-front type methods can result in the same kind of meshes by locating the created point properly (for example, at a location such that the resulting edges are of the desired size).

A Delaunay-type method as previously described is also a possible solution that can be used to obtain governed meshes.

In the following sections, we give some details about these three classes of automatic mesh generation methods when an isotropic adaptive problem is considered (the case of an anisotropic situation will be discussed in subsequent sections).

Remark 21.11 *To make the discussion of each method independent, some materials, common to several methods, may, to some extent, be repeated in what follows. Note also that a good knowledge of the methods in their classical version is assumed.*

•H-method and quadtree-octree approach

A quadtree-octree type mesh generation method (Chapter 5) allows for some flexibility in the mesh creation process leading to the creation of meshes conforming to pre-specified isotropic requests.

A quadtree-octree type mesh generation method can be decomposed into several steps where the first concerns the construction of the underlying quadtree-octree structure. Then, based on this spatial structure, field points are inserted and elements are constructed prior to some degree of optimization.

Underlying tree construction. The tree construction is the basis of the mesh construction. In the classical meshing method, the tree is constructed based on the domain boundary discretization provided as data. Recall that this tree construction is a recursive process in which several criteria are used to decide whether or not the current tree is fine enough or must be refined again.

In the context of a h-method, the same idea can be retained but now the tree is constructed based on the boundary discretization of the domain and the sizing function which is desired. Let $h(P)$ be the size at point P where P is a vertex of the background mesh serving as control space, then the tree must take into account this information. Thus, a reasonable method resulting in an adequate tree structure could be as follows:

- construct the tree structure based on the domain boundary discretization (which is assumed to conform to the size map),
- refine, if necessary, the above tree, by checking if it conforms to the hs included in the background mesh which means that the size of a terminal cell corresponds to the desired element size. To this end, for each cell of the tree, we can find the vertices of the background mesh which fall within this cell. Then, examine if the cell size conforms to the sizing values. If not, pursue the tree decomposition until a satisfactory matching has been reached, while applying classical rules (such as the 2:1 rule) to balance the tree. Figure 21.9 depicts how the h's data interact with the tree construction.

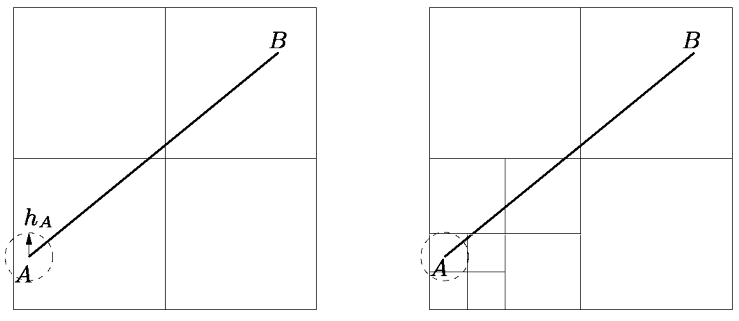


Figure 21.9: *Tree decomposition using a prescribed size. The size at point A forces the initial tree box containing A (left-hand side) constructed without explicit sizing prescription to be decomposed two levels deeper (right-hand side) to adjust to the size h_A represented by the dashed circle.*

Remark 21.12 *The tree level l is then related to the size of a mesh edge, h, as $l = \log_2(b/h)$ where b is the length of a side of the root cell.*

Remark 21.13 *Note that, in this method, the distance between two connected mesh vertices (i.e., an edge length) is controlled by the cell sizes. Hence, the edge size control is not obtained explicitly but results from the tree decomposition.*

Remark 21.14 *Also, using the [2:1] rule results in a ratio of one, one half or two between two adjacent cells (in terms of size).*

Element construction. Once the tree has been completed, the element creation follows the same aspect as in a classical quadtree-octree type method (see Chapter 5).

Mesh optimization. Mesh optimization, taking place at the end of the meshing process, is slightly different than in the classical meshing problem. Unlike this case where only the element shapes were considered, we now have to take into account two (possibly antagonistic) aspects. Actually, what is needed are well-shaped elements conforming to a specified size. Thus, these two aspects must be the objective of the optimization stage. We refer the reader to a further section on this point as optimization purposes in this quadtree-octree based method are similar to those of the two other meshing techniques described below.

In conclusion, adaptation using a *quadtree-octree* method, such as that proposed above, is essentially based on a particular construction of the tree structure so that the cells in this tree reflect through their sizes the desired metric specifications. Therefore, the control is achieved by means of the tree.

•H-method and advancing-front approach

An advancing-front type mesh generation method (see Chapter 6) allows for some flexibility in the mesh creation process leading to the creation of meshes conforming to pre-specified isotropic requests.

An advancing-front type mesh generation method includes the construction of the field points and their connection with the current front and a final step that corresponds to some extent of optimization.

Point creation and point connection. One way to control the mesh creation with regard to a given sizing map is to analyze the length of the mesh edges. Let us assume a three-dimensional case. Then, given a front face, say ABC, we examine the context to decide whether an existing point or a new point must be created that can be combined with face ABC in order to form an element. In the case where this candidate element is indeed formed, we will have introduced one, two or three new edges depending on the situation. As a consequence, a control on this (these) new edge(s) allows us in principle to complete what is needed.

Thus, let us examine a (potentially new) edge of this candidate element, say AP (we assume that P is known, see below regarding a way to find such a point). If $h(t)$ stands for a sizing function defined for this edge (with $h(0) = h(A)$ and

$h(1) = h(P)$), then the length of edge AP with respect to $h(t)$ is:

$$l_{AP} = d_{AP} \int_0^1 \frac{1}{h(t)}\, dt \tag{21.1}$$

where d_{AP} is the usual (Euclidean) distance.

Actually, $h(t)$ is only known in a discrete way (thanks to the background mesh). So, if only $h(A)$ and $h(P)$ are provided, we have (using an approximation based on a rather simple quadrature formula):

$$l_{AP} = d_{AP} \frac{\dfrac{1}{h(A)} + \dfrac{1}{h(P)}}{2} \tag{21.2}$$

while if edge AP intersects a series of elements in the background mesh and if M_i stands for the i^{th} intersection point, then we can define the length of AP as:

$$l_{AP} = \sum_{i=0}^{i=n} l_{M_i M_{i+1}} \tag{21.3}$$

where (using the same quadrature formula)

$$l_{M_i M_{i+1}} = d_{M_i M_{i+1}} \frac{\dfrac{1}{h(M_i)} + \dfrac{1}{h(M_{i+1})}}{2} . \tag{21.4}$$

In this relationship, $h(M_i)$ is obtaine by interpolation on the background mesh. Using these definitions, the (potentially new) edges of the analyzed element are examined. This means (see Figure 21.10) that we may be interested in computing the lengths of edges AP, BP and CP, given the face ABC. Based on these values, as compared with the unity, point P is moved towards or away from the face (see Chapter 6) and the same analysis is repeated until (a satisfactory) convergence is obtained.

In theory, we need to find a point, P, the solution of:

$$l_{AP'} = l_{BP'} = l_{CP'} = 1 ,$$

which is, in practice, unrealistic. Therefore, the above approximate method (based on iterations) is a reasonable solution.

The above discussion assumed that P was found in some neighborhood of face ABC meaning that an existing point can be used or, on the other hand, that P is constructed somewhere. This being done, point P is easily adjusted using the previous material. The point is then to find an initial (and not too bad) candidate point. One method could be, based on face ABC, to define point P_{opt} such that element K is well-shaped. Then, among the three edges $P_{opt}A$, $P_{opt}B$ and $P_{opt}C$, the one or those which will be added in the mesh in the case where element K

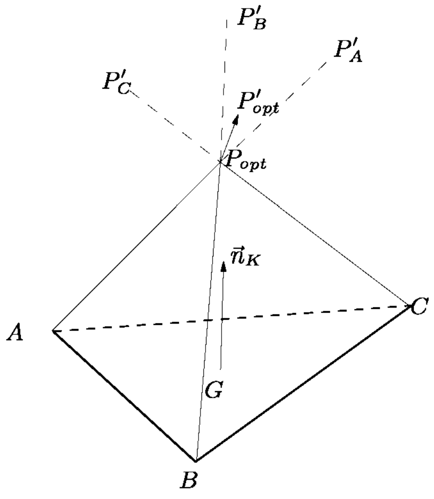


Figure 21.10: *Optimal point P creation, given a front face ABC the resulting tetrahedron K is considered optimal as it has edge lengths conforming to the desired sizing map. P_{opt} is first constructed, then P'_A, P'_B and P'_C are defined and an adequate combination of these (virtual) points is used to define P'_{opt}. Finally, the new location of P is prescribed to be this P'_{opt}.*

is retained, are examined based on the metric map. If P_{opt} is a free-point, its location is adjusted to meet as closely as possible a unit length for the edges of element K. Then P is assigned to be the resulting adjusted point. Note that all these operations are coupled with the classical validity checks as performed in a classical advancing front type method.

Remark 21.15 *Note that the radii used for the searching operations for the candidate points must take into account the desired element sizes.*

Remark 21.16 *In [Rassineux-1995], the internal points are constructed using an* octree, *in three dimensions. Thus, provided this tree structure is developed in accordance with the size map (cf. the previous approach), we obtain a priori a mesh which is reasonably satisfactory with regard to this size map.*

Mesh optimization. As above, we refer the reader to a later section on the optimization step of the method.

In conclusion, adaptation using an advancing-front method, such as that proposed here, basically relies on an appropriate location of the points with regard to the front faces. The global control, each point being well located for its face, is due to the fact that the candidate points are selected in a neighborhood of the faces in question and thus these points are globally well located.

• H-method and Delaunay-type method

A Delaunay-type mesh generation method also appears to be flexible enough to carry out the creation of meshes conforming to pre-specified isotropic requests.

As we saw in Chapter 7, a Delaunay-type mesh generation method can be decomposed into several steps that concern the insertion of the boundary vertices, the boundary enforcement, the creation of a suitable set of field points prior to their insertion and some extent of optimization.

Delaunay kernel. Recall that, under appropriate assumptions, the Delaunay kernel is the basic ingredient that makes it possible to insert a point in a given triangulation. This Delaunay kernel is simply expressed by the relationship

$$\mathcal{T}_{i+1} = \mathcal{T}_i - \mathcal{C}_P + \mathcal{B}_P \,, \tag{21.5}$$

where the cavity and the ball of P are involved. Then, whatever the size map, this basic algorithm remains valid.

Boundary enforcement. Obviously, the *a posteriori* enforcement of the boundary entities when the boundary vertices have been inserted, remains unchanged in this adaptive context.

Internal point creation. Following the classical point creation method (Chapter 7) using as a support the edges of the current mesh, we have to examine how this method could be used in the present adaptive context.

Note initially that the proposed method is fairly similar to that used in the above advancing-front approach since it is based on a control related to the edge lengths.

Thus, let us examine an edge, say AB. If $h(t)$ stands for a sizing function defined for this edge (where $h(0) = h(A)$ and $h(1) = h(B)$ hold), then the length of edge AB with respect to $h(t)$ is

$$l_{AB} = d_{AB} \int_0^1 \frac{1}{h(t)}\, dt \tag{21.6}$$

where d_{AB} is the usual (Euclidean) distance.

Actually, $h(t)$ is only known in a discrete way (thanks to the background mesh). So, if only $h(A)$ and $h(B)$ are provided, we have (using an approximation based on a rather simple quadrature formula):

$$l_{AB} = d_{AB}\, \frac{\dfrac{1}{h(A)} + \dfrac{1}{h(B)}}{2} \,, \tag{21.7}$$

while if edge AB intersects a series of elements in the background mesh and if M_i stands for the i^{th} intersection point, then we can define the length of AB as:

$$l_{AB} = \sum_{i=0}^{i=n} l_{M_i M_{i+1}} \tag{21.8}$$

where $l_{M_i M_{i+1}}$ is approached using Relationship (21.4). Using these definitions, the edges of the current mesh are analyzed to create some points along them.

This can be done using the very simple algorithm that follows (where n is the number of the M_is):

```
$l = 0$. (where $l$ stands for a length)
DO FOR $i = 0, n$
   $l = l + l_{M_i M_{i+1}}$
   (A) - IF $l > 1$
      create one point between $M_i$ and $M_{i+1}$ at a unit distance from the
      previously created point ($M_0 = A$ at the first step),
      $l = l - 1$ and return to (A).
END FOR $i$.
```

This process is repeated for all the edges of the current mesh leading to the creation of a series of points. These are inserted using the Delaunay kernel, resulting in a new mesh. The process is then repeated until all the edges are satisfactory (in terms of length). Actually, the unit value is replaced by an appropriate value close to one (as it is for all of these length based methods).

Mesh optimization. As above, we refer the reader to the next section about the optimization step of the method.

In conclusion, adaptation for a Delaunay-type method mostly relies (in the proposed approach) on the proper position of the points in the mesh edges. The global control, every point being well located on its supporting edge, is due to the fact that the points are filtered before insertion, this leading to a global consistence.

• Mesh optimization tools (isotropic case)

The optimization stage met in all automatic mesh generation methods (for simplicial meshes) is, in practice, the same for all of them (*quadtree-octree*, advancing-front or Delaunay). As already seen, this stage is the last in the mesh construction process. The optimization is governed by a quality objective (at the element level) with includes two criteria, a *shape* criterion and, at the same time, a *size* criterion. The point is what mesh quality is expected and, based on this objective, what tools can be used and how to combine them (i.e., how to define a mesh optimization strategy) so as to improve the mesh resulting from the first steps of the mesh generation (modification) method.

Mesh evaluation. Provided with a metric map, the mesh quality analysis must naturally take into account this map to examine whether or not the mesh elements conform to it. In addition, as we are considering an isotropic situation, the element shapes (aspect ratios) must be as good as possible. Indeed, we have discussed all these aspects in Chapter 18 where aspect ratio measures as well as length efficiency index have been introduced and which form the basic ingredients we need to evaluate a mesh quality.

However, the above appreciation is not directly related to the main reason for constructing the presumably adapted mesh and thus, this appreciation is only formal. In fact, in the context of an adaptive loop of finite element computation (our purpose), the right tool to analyze the mesh quality is the error estimate which analyzes the quality of the solution of the PDE problem under consideration. Since the error estimate is not known in the meshing process, we assume that the above formal appreciation is valid (and consistent). Thus, at the meshing stage of the entire process, we still use this kind of appreciation.

As detailed in Chapter 18, various local tools can be used to optimize a given mesh. In our context, the same tools are used in the optimization step included in the mesh generation methods.

Shape optimization. Classical optimization tools (Chapter 18) can be used such as node relocation, edge swapping, generalized swapping, etc.

Size optimization. Edge collapsing or edge splitting can be performed when a given edge is too short (in terms of l measured according to the discrete metric map) or too long. Prior to actually apply such an operator, it is necessary to check whether or not it results in a better solution. For instance, an edge that is too long could be retained if splitting it into two shorter edges results in two violations of the sizing criterion (instead of only one violation in the initial configuration). Apart from these two processes, node relocation proves to be useful. A free vertex, namely P, can be moved as follows:

$$\overline{P}_j = P_j + \frac{\overrightarrow{P_jP}}{\|\overrightarrow{P_jP}\|}\,\overline{h}_j \tag{21.9}$$

where $\overline{h}_j$ is such that $l_{P_jP} = 1$ in the metric. Then, the new position of point P can be the centroid of the $\overline{P}_j$s.

Optimization strategy. First, it could be observed that respecting a size map while maintaining well-shaped elements could be two conflicting objectives, based on what the size map is. Thus, the optimization strategy must take into account this possible conflict. Following our experience, we suggest firstly optimizing the length criterion and then, this objective being satisfied, optimizing the element shapes (although this may alter the length criterion to some degree).

Comments about global methods (isotropic)

As for the local approach, we give some preliminary observations about global adaptive methods. The aim is not to pass judgment on the various mesh generation methods that may be employed but to give a general impression of a global approach.

Indeed, this adaptive approach offers both a series of positive aspects and some weaknesses. Let us mention first some apparent weaknesses of a (isotropic) global method:

- Localization problems could be an issue and could be time consuming.
- CPU requirements could be large, particularly when the difference between the initial mesh and the adapted mesh concerns only a small part of the computational domain.

Now, positive features of a global method are briefly indicated.

- Essentially, global methods appear to be a good solution for isotropic adaptive problems.
- Mesh conformity is not an issue.
- Mesh derefinement is obvious since it is no different from a refinement problem.
- Continuous variation in size is easy to obtain.

Note that other computer issues, such as boundary management, are common to the local method discussed in a previous section and, thus, do not count in the appreciation of the approach.

Thus, based on these observations, we have to decide whether or not a global method is a suitable solution to carry out a given problem.

21.4 Global anisotropic adaptation method

Anisotropic meshes or meshes with anisotropic meshed regions are of great interest for some numerical simulations (we can consider the construction of boundary layers or the case where the problem induces some shocks).

As for the previous meshing problem, anisotropic mesh generation can *a priori* be based on any of the classical mesh generation methods. Nevertheless, each of them must be precisely examined in order to see if it can be easily extended to this particular meshing problem. In what follows, we review the *quadtree-octree*, the advancing-front and the Delaunay-type methods with this precise objective.

•H-method and quadtree-octree approach

Basically, the mesh elements resulting from a quadtree-octree type method, are strongly related, in terms of size as well as in terms of shape, to the underlying tree structure. More precisely, the mesh elements are created based on the cells of the tree structure. Thus, since the cells are in essence isotropic, it is not so easy to find a method resulting in the construction of anisotropic elements. Up to now and as far as we know, there is no available literature about this point and no attempts to construct an anisotropic quadtree-octree structure.

Remark 21.17 *Nevertheless, it could be observed that a global anisotropy (i.e., where the anisotropy is globally defined and aligned with the usual coordinate axis and also does not vary from regions to regions) can be obtained by constructing the tree structure accordingly.*

Despite the above observations, it could be of interest to examine how it is possible to construct a quadtree (octree) structure using a repeated subdivision of the root cell not aligned with the sides of this cell. This being completed, we could examine whether or not such a spatial partition can be exploited for anisotropic mesh construction.

• H-method and advancing-front approach

An advancing-front type mesh generation method is more flexible than the previous type of method since point connection as well as point placement can be guided by anisotropic criteria.

In an anisotropic context, directional features and sizes varying in these directions are expected at the mesh element level. This leads to introducing what the length of an edge is, with respect to a given metric map.

Length of an edge. In Chapter 10, we have seen that computing a length consists of using the dot product related to a quadratic form. Here we are concerned with the curve Γ joining two points A and B. If $\gamma(t)$ is a parameterization of Γ of class C^k $(k \geq 1)$ such that $\gamma(0) = A$ and $\gamma(1) = B$, the *length* $L(\gamma)$ of the arc Γ in the metric $\mathcal{M}_\gamma$ is defined by the expression:

$$L(\gamma) = \int_0^1 \|\gamma'(t)\| dt = \int_0^1 \sqrt{{}^t\gamma'(t)\, \mathcal{M}_\gamma\, \gamma'(t)}\, dt\,. \qquad (21.10)$$

Therefore, the restriction of the parameterized arc Γ to the vector $\overrightarrow{AB}$ (i.e., the edge under examination) with the parameterization $\gamma(t) = A + t\overrightarrow{AB}$, $t \in [0,1]$ and $\gamma(0) = A, \gamma(1) = B$ enables us to write the length $L(\gamma)$ of the line segment AB as:

$$L(\gamma) = \int_0^1 \sqrt{{}^t\overrightarrow{AB}\, \mathcal{M}_\gamma\, \overrightarrow{AB}} dt\,, \qquad (21.11)$$

where M_γ is the metric specification in γ. Once the metric does not vary with the position $(\mathcal{M}_\gamma = \mathcal{M})$, we obtain:

$$L(\gamma) = \sqrt{{}^t\overrightarrow{AB}\, \mathcal{M}\, \overrightarrow{AB}}\,.$$

Having this definition, it becomes possible to compute the lengths of the mesh edges is the case where an anisotropic metric is given.

Internal point creation. As for the isotropic case above, creating an adapted mesh is based on the edge length control with respect to the metric map. Given a front face ABC, we want to know whether a point P in the current mesh exists which is suitable for the construction of elements whose edge lengths are compatible with the sizing and directional specifications. If such a point is found, it is used to create a mesh element, otherwise, the above method is used to find the location of an *optimal* point. A point P_{opt} is first constructed in such a way that the element $ABCP$ has a nice shape quality. The position of P_{opt} is then adjusted in order to meet unit edge lengths in the metric.

The element constructed based on ABC and P_{opt} is an element whose size conforms to the size specification and whose stretching and orientation follow the directional features included in the anisotropic metric map.

Metric interpolation. In practice, the metric at a vertex is known in a discrete manner (in fact at the background mesh vertices). Therefore, it is necessary to use a metric interpolation procedure to obtain the metric value(s) at a vertex P in the current mesh. To this end, the element K in the background mesh in which point P falls is identified (this point is that resulting in an optimal element). Using the metric information at the vertices in K, the metric at P is obtained by interpolation and then serves to adjust the position of P, in order to obtain unit length for the edges AP, BP and CP.

Remark 21.18 *The advancing-front approach also makes it possible to construct some boundary layers in the vicinity of a given boundary. This boundary forms a front which is then "pushed" in order to define the first layer, this then forms a new front and the same method is repeated [Kallinderis* et al. *1995].*

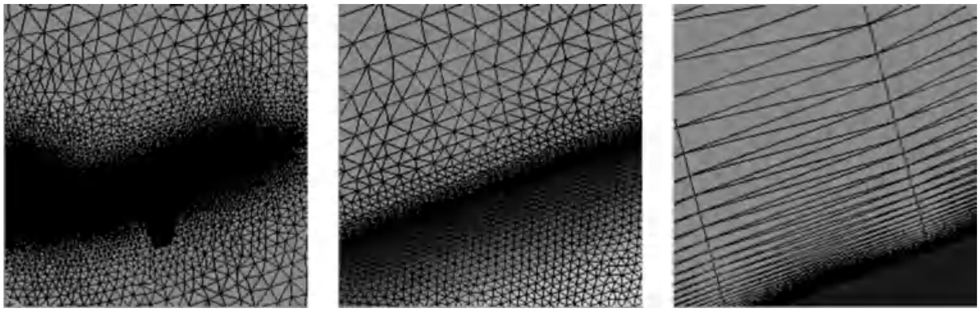


Figure 21.11: *One can guess an aircraft and two close views where boundary layers can be seen. This example, courtesy of Dassault-Aviation, is a case where and advancing-front method is used to obtain a number of boundary layers after which a Delaunay-type method is used to fill the remaining domain.*

Remark 21.19 *A combination with a Delaunay-type method is also a solution for completing anisotropic elements. The optimal points are created using an advancing-front strategy and are then connected using a Delaunay method (see [Mavriplis-1992], for instance).*

• H-method and Delaunay-type method

Given the final theoretical remark of Chapter 7, the Delaunay-based method as described in this chapter is probably also a possible answer for anisotropic mesh generation. As previously seen, a Delaunay-type method is mainly based on the notion of a distance between two points. Indeed, the Delaunay kernel, the field point creation phase and the optimization phase all make extensive use of this kind of information[9].

[9]While the other phase (boundary enforcement), part of the whole method, is, in principle, not affected by the present meshing context.

Edge length. Before going further, we recall how the necessary lengths are defined. Given a curve Γ, joining two points A and B, we consider $\gamma(t)$ a parameterization of Γ, at least of class C^1 (t ranging from 0 to 1) such that $\gamma(0) = A$ and $\gamma(1) = B$. Then, $l_{\mathcal{M}}(A, B)$, the distance, following the metric map characterized by the matrices $\mathcal{M}$s, between A and B is $l(\Gamma)$, the length of Γ defined as follows:

$$L(\gamma) = l_{\mathcal{M}}(A, B) = \int_0^1 \sqrt{{}^t\overrightarrow{AB}\, \mathcal{M}_\gamma\, \overrightarrow{AB}}dt\,, \tag{21.12}$$

or, if metric $\mathcal{M}$ does not depend on the position:

$$L(\gamma) = l_{\mathcal{M}}(A, B) = \sqrt{{}^t\overrightarrow{AB}\, \mathcal{M}\, \overrightarrow{AB}}\,.$$

Now, using this length definition, we can return to the scheme of the isotropic case and compute the length of AB using metric information, the $\mathcal{M}$ matrices, collected on the background mesh.

Point insertion method (Delaunay kernel). Actually, the classical Delaunay point insertion algorithm is no longer suitable. Indeed, following the method proposed in the isotropic case while modifying the way in which the points are located (i.e., by computing the unit lengths using Relation (21.12) instead of Relation (21.6)) results in *a priori* well located internal points. Inserting these points via the classical Delaunay kernel is possible, but the underlying proximity notion (based on a Euclidean distance) does not match the way in which the distances from point to point have been evaluated. Thus, the local procedure (already described elsewhere):

$$\mathcal{T}_{i+1} = \mathcal{T}_i - \mathcal{C}_P + \mathcal{B}_P\,,$$

enabling us to insert point P in triangulation $\mathcal{T}_i$ so as to complete $\mathcal{T}_{i+1}$ must be extended to the present context.

Note that $\mathcal{B}_P$ is the set of elements formed by joining P with the external edges (faces) of $\mathcal{C}_P$, these being the series of elements in $\mathcal{T}_i$ whose circumcircle (circumsphere) encloses P. Thus, now, this notion of a circle (sphere) is defined according to the metric associated with the problem.

The key-issue is to construct a proper *cavity*, i.e., the set $\mathcal{C}_P$. To this end, we consider a two-dimensional meshing problem and, at first, we return to the classical case. Let K be an element of $\mathcal{T}_i$, let O_K be its circumcenter and let r_K be its circumradius. Then, element K will be a member of $\mathcal{C}_P$ if

$$\alpha(P, K) = \frac{d(P, O_K)}{r_K} < 1 \tag{21.13}$$

where d is the usual distance. To take into account a metric map, this classical characterization is replaced by:

$$\alpha_{\mathcal{M}}(P, K) = \frac{l_{\mathcal{M}}(P, O_K)}{r_K} < 1 \tag{21.14}$$

where, now,

- O_K stands for the point equidistant to the vertices of K. This means that this point is the center of the circumcircle of K according to the metric defined by $\mathcal{M}$ (in general, this circle is an ellipse in the usual Euclidean space) and,

- r_K is the radius of this circle according to the Euclidean space defined by $\mathcal{M}$, i.e., r_K is the length between the point equidistant to the three vertices of K, the above O_K, and one of these points.

Computing O_K as well as r_K is not, in general, possible, as $\mathcal{M}$ varies from one point to another. Thus, *approximate* solutions must be involved leading back to a Euclidean problem. The simplest one consists of approaching $\mathcal{M}$ by the value of $\mathcal{M}$ at point P. This approximation results in a possible construction of $\mathcal{C}_P$ (after a correction step) and the point insertion method applies in an anisotropic context (see Chapter 7).

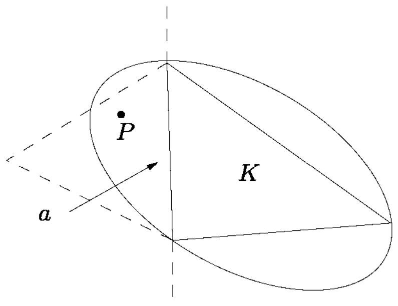


Figure 21.12: *The circumcircle associated with element K once the metric is fixed. The construction of the cavity of P by means of adjacencies relies, starting from one triangle in this set (dotted line), in examining its neighboring triangles, here that share the edge denoted by a, i.e., element K.*

It has been proved (thanks to the correction step) that the resulting cavity has the desired properties and thus that replacing it by the ball of P results in what is expected: an anisotropic point insertion method.

Remark about the boundary enforcement. In practice, as the elements in manipulation can be strongly anisotropic, numerical problems[10] may arise.

Internal point creation. This step is similar to the isotropic case previously discussed while the lengths are evaluated using the above (anisotropic) formula. We return to the isotropic scheme where queries about the background mesh together with interpolation of the collected information are used to compute the length of a given edge. As for the Delaunay kernel, an approximation is used in this length computation.

[10]While in principle, this stage is similar to how it was in a classical case.

• Mesh optimization tools (anisotropic case)

Common to all automatic methods resulting in simplicial elements, an optimization phase forms part of the meshing process. The optimization here is based on a shape quality criterion coupled with a size (directional) quality criterion. The delicate issue is, as in the isotropic case, to define a optimization strategy that produces what is desired.

Mesh evaluation. The analysis of the resulting mesh must take into account the specified metric map. The aspects about optimization itself have been discussed in Chapter 18. It should just be borne in mind that the efficiency index allows for a global appreciation of the lengths of the mesh edges. Moreover, using an error estimate may provide information (in the form of a metric map) that makes the mesh analysis possible. The tools for optimization as described in Chapter 18 apply here with no restriction.

Shape optimization. Among the classical optimization tools for element shape improvement, we have edge swaps, point moves (with a unit length), etc.

Length optimization. The edge lengths having been computed in the given metric, the collapsing (resp. splitting) operators deal with the edges that are too small (resp. too long). The point relocation procedure consists of trying to obtain unit length edges (in the metric at the considered vertex P).

Optimization strategy. As in the isotropic governed case, an efficient strategy leads to optimizing the mesh by firstly considering the size criterion. A quality (shape) criterion is then considered.

Comments about global methods (anisotropic)

In this section, we give some remarks about anisotropic global mesh adaptation methods. As in the isotropic case, a series of (relative) weaknesses can be found:

- localization problems may be tedious,
- the time required for a remeshing may be relatively great, specifically when the adaptation is rather local,
- numerical problems may arise due to the stretching of the elements (when computing surface areas or volumes, for instance).

Nevertheless, a certain number of positive features can be mentioned:

- the global remeshing approach is a likely solution for anisotropic adaptive problems,
- a mesh gradation is, in general, relatively easy to achieve (or to maintain).

As in the isotropic case, mesh derefinement (coarsening) is a trivial task since it does not differ from a refinement method. Note that it is always the user's responsibility to decide whether the problem at hand requires the global anisotropic approach.

21.5 Adaptation

Adaptation is a key issue for automatic simulations where the purpose is to insure a given accuracy of the solution. Given a tolerance threshold, the problem is to compute, for a given PDE problem, a solution whose accuracy conforms, in some way, to this threshold.

There are several approaches suitable for adaptation purposes. As discussed up to now in this chapter, these include the h-method where the mesh, the support for the computational step, is adapted in terms of element size (density) or sizes and directions. In an isotropic meshing problem, the desired mesh must be coarser or finer in a particular region as specified by an error estimate that analyzes the quality of the solution computed using the current mesh as a spatial support. In an anisotropic context, the elements must be aligned in the directions specified by such an error estimate which is assumed to have a directional aspect and, at the same time, these elements (indeed their edges) must have the required lengths.

Following the previous discussion, the mesh generation aspect can be envisaged in two ways in order to design a h-method. The first is based on local modifications of the current mesh while the other involves a entire mesh (re)construction at each step of the process (or after a few iteration steps).

The ingredients needed in this context include local mesh modification tools (local approach) or fully automatic mesh generation processes (global approach) resulting in adapted meshes along with solution methods coupled with *a posteriori* error estimates able to provide mesh specifications used, in turn, to repeat the full process until the tolerance threshold has been achieved.

In what follows, we indicate what a global computational scheme could be for both a local and a global meshing approach in the case of an adaptive loop of FEM style computation. The example concerns a three-dimensional case from which it is easy to infer what a two-dimensional case could be.

21.5.1 General framework of a local adaptation method

The general framework of a h-method adaptive loop of computation when the adaptation is based on local mesh modifications at each step is illustrated in Figure 21.13 (in three dimensions).

The general scheme includes two parts. Left, we find a scheme similar to that in a classical mesh generation method. Right, we see the part directly related to the adaptation phase.

More precisely, for the classical part, we start from a CAD system (box "CAD" in the figure), to define a first surface mesh ($j = 0$), then a 3D mesh generator

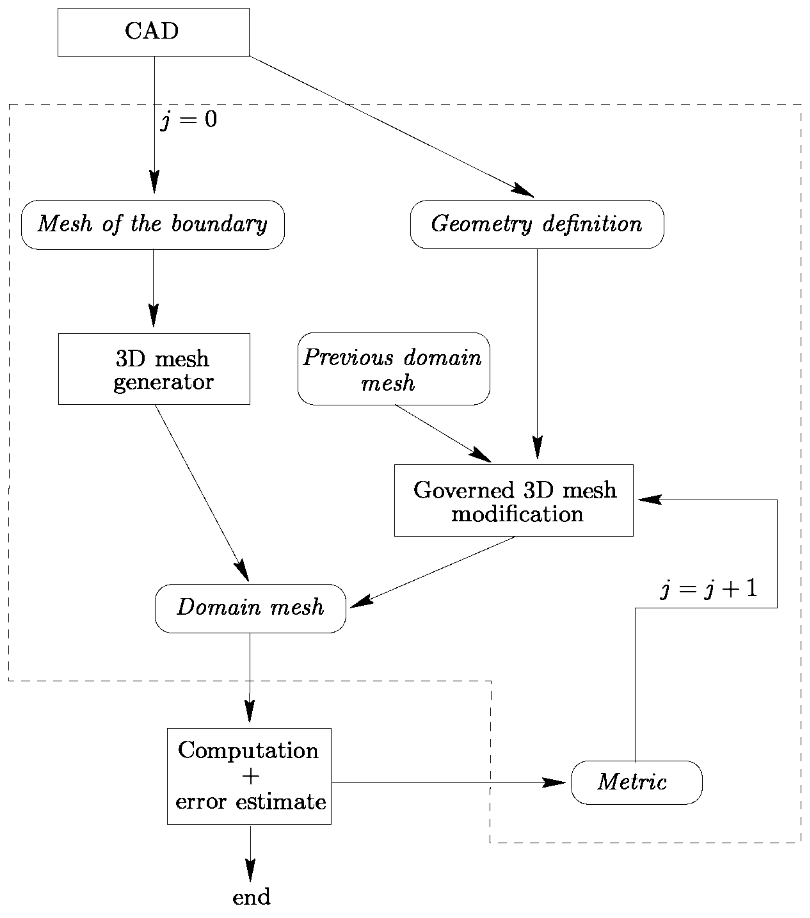


Figure 21.13: *General framework for a local adaptation method.*

creates the mesh of the domain and this part (left-hand side of the figure) is nothing other than a *classical mesh generation problem.* Then the solution of the PDE problem is computed and analyzed using an *ad-hoc* error estimate. The latter provides a metric map (box "Metric"). Based on this metric map, the process is iterated ($j = j + 1$), thus consisting in the second part of the figure (right-hand side) and corresponding to the *governed mesh modification problem.* Note that the geometry of the surface (box "Geometry definition") is now strictly necessary as will be discussed below. The local mesh modification step consists of modifying the mesh so as to complete a mesh conforming to the data included in the metric map.

The metric map is indeed a simple request associated with each element of the type *element to be subdivided* or, conversely, *element to be coarsened.* In practice, the same type of requests may be given at the vertices of the current mesh (for example, refine (once or several times) around a given vertex).

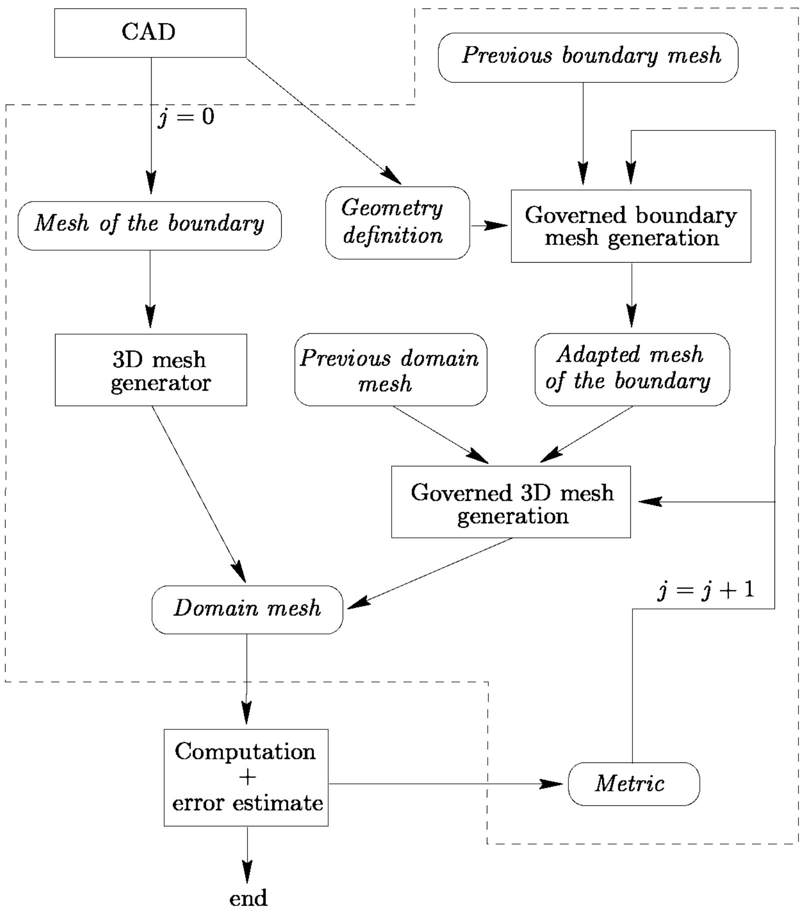


Figure 21.14: *General framework for a global adaptive method.*

21.5.2 General framework of a global adaptation method

The general framework of a h-method adaptive loop of computation when the adaptation is based on the entire mesh (re)construction at each step is depicted in Figure 21.14.

As for the previous approach, we can see two parts as shown in the previous figure. Indeed, the first part (left-hand side, the *classical mesh generation problem*) remains unchanged while the second part (right-hand side, the *governed mesh creation problem*) is rather different.

The classical part in the scheme is similar to that in the previous case and completes an initial mesh of the domain using an initial mesh of its surface as data. In contrast, the adaptive part in the scheme differs slightly from that included in the previous scheme. Now this part comprises two steps. One concerns the surface mesh processing while the other, based on an automatic mesh generation method, considers as input the above surface mesh (assumed to be conform to the metric specifications) and completes the domain mesh accordingly.

Remarks about an adaptive scheme

The two above general frameworks include a series of processes which, in turn, involve various types of input and data flows. In what follows, we give some ideas and comments about these aspects.

Geometry definition. The precise definition (in an analytical way, for instance) of the geometry of the domain surface to be meshed is not strictly required in a classical meshing process[11]. In fact, the domain mesh is generally obtained using a discretization of its boundary as input data. Therefore, this boundary mesh is assumed to be known and is sufficient for the definition of the domain in question.

In contrast, in an adaptive meshing process, the discretization (the mesh) of the domain surface may vary during the iterations, depending on the metric specifications. It is then necessary to have access to a definition of the geometry (the boundaries) of the domain. In practice, there are two ways to obtain the information related to the geometry:

- using a direct access to a geometric modeler (a CAD system) to which queries are made by the mesher in order to know the geometrical or topological information which is needed (corners, ridges, curve definitions, surface definitions, normals, tangents, curvatures, etc.);
- using an indirect access, which means using a mesh, the so-called *geometric mesh* which serves as the geometric definition of the domain (Chapter 19).

When a geometric mesh serves as a support for the geometry definition, the properties of the boundary curve (resp. surface) are obtained using this discretization. In principle, such a mesh is constructed by the CAD system and the geometric approximation it forms is assumed to accurately reflect the geometry of the curve (resp. surface) it models. Therefore, a mesh whose element density indicates the curvature well is a suitable candidate. On the other hand, it must be noted that such a mesh is not generally a suitable mesh for a finite element computation (in particular, the mesh gradation is not necessarily controlled).

Given such a geometric support, we return to the previous case in which queries are addressed to the geometry in order to obtain the required information.

The boundary mesh construction. A mesh of the domain boundary is the natural data input[12] of a mesh construction or a mesh modification procedure. In practice, there are two types of problem regarding surface meshes.

[11]This is strictly the case for an advancing-front or a Delaunay-type method but this is not necessarily the case for a classical *octree* method where the surface mesh and the volume mesh can be constructed at the same time.

[12]See the previous note.

The classical part of the adaptive mesh construction scheme consists of constructing an initial mesh of the domain. This is done with no special knowledge, i.e., of a metric nature related to the physical behavior of the problem. The sole properties used are those related to the geometry. The surface mesh results from a geometric modeler or by using an appropriate surface mesh method. However, this mesh must be reasonable (meaning that it is representative of the underlying geometry, see Chapter 19), but it is probably not ideal with regard to the physics included in the PDE problem in hand, which therefore justifies the use of an adaptive approach.

In a local approach, the surface is remeshed so as to take into account the given metrics and the geometric metric. The basic idea is always the same, and aims at constructing unit length edges. The remeshing process, discussed in Chapter 19, makes use of geometrical operations (point relocations) or topological operations (edge swaps, edge collapsing, point insertion, etc.). In brief, remeshing a surface is seen as an optimization procedure. The information which is then pertinent concerns:

- the proper location of a point on the surface,
- the access to the properties of the surface (normals, tangents, principal radii of curvature, etc.).

Note that the local surface remeshing can be made at the same time that the local remeshing of the volume is made.

In a global approach, the surface mesh is constructed in a stand-alone step with no connections with the volume mesh. See Chapters 14 and 15 for a detailed discussion on mesh generation methods for curves and surfaces.

The mesh construction. Using an initial surface mesh as data, the volume mesher constructs an initial mesh in the domain. It is clear (following the mesh generation methods described in this book) that the quality of this three-dimensional mesh is strongly related to the quality of this surface mesh. If a metric map (obtained using an error estimate after the solution analysis) is available, it is possible to numerically evaluate whether the current mesh is satisfactory or not. If it is not judged to be good enough, the mesh generation process is iterated, thus leading to the construction of a new mesh, taking into account the geometric metric map together with the physical metric map. The current mesh then becomes the background mesh for the next iteration step.

As for the surface meshing case, two types of methods can be envisaged to obtain adapted volume meshes. A local approach makes use of the optimization operators, as we have already seen. In a global approach, the new mesh is entirely recreated using the metric map defined at the vertices in the background mesh. The mesh generation is then governed, meaning that we aim to construct a mesh with unit length edges.

The solution step. The solution step comprises a solution method (the method used to solve the resulting matrix system) and an error estimate to access the quality of the solution that is computed. This estimate also serves to translate this analysis in terms of directives that are directly usable by the mesh generation method.

In a global adaptation scheme, it is advisable to use an iterative solution method in which the solution which is sought is initialized by the solution obtained at the previous iteration step. Therefore, interpolation methods must be used to interpolate the solutions from mesh to mesh. This raises the problem of how to *transport the solution* from the background mesh to the current mesh. These methods involve finding the position of a vertex in the current mesh in the background mesh and are thus based on localization procedures (Chapter 18).

The metric map. As previously indicated, the metric map is a discrete set of values (tensors) which are usually known at the vertices in the background mesh. When several metrics are specified at these vertices, an intersection procedure (Chapter 10) is used to find a unique metric. A continuous map is then obtained by (linear) interpolation using the vertex values.

21.6 Application examples

In this section, we show some examples of adapted meshes corresponding to various iteration steps in an adaptive process. In all these examples, the approach is a global one for the domain meshing (planar or volume domain) and a local approach (by remeshing) for the boundary (curves or surfaces) meshing.

The first series of examples (Figure 21.15) corresponds to a mechanical device in two dimensions. The mesh generation method is a *quadtree* type method. In this example, the metric map is isotropic, the sizing function is defined in an analytic manner using two real valued functions:

$$h1(x,y) = 4|r_1 - ||\overrightarrow{C_1P}||| + 0.05 \quad \text{and} \quad h2(x,y) = 2|r_2 - ||\overrightarrow{C_2P}||| + .02\,, \quad (21.15)$$

where $C_1 = (-3,5)$ and $C_2 = (10,-5)$ are the center-points of two discs whose radii are respectively $r_1 = 7$ and $r_2 = 13$. Ten iteration steps for mesh construction were necessary to capture the analytical map in a satisfactory manner.

mesh - iter	np	ne	Q_T	$\overline{Q}$	l_{min}	l_{max}	l_{avg}	τ
part - 0	2,204	3,794	2.9	1.29	0.24	8.33	0.75	0.88
part - 1	2,808	4,975	2.9	1.28	0.21	21.13	0.77	0.89
part - 3	5,090	9,502	3.64	1.27	0.24	29.9	0.75	0.90
part - 10	7,772	14,847	2.9	1.27	0.24	1.78	0.74	0.92

Table 21.1: Statistics about the numerical evaluation of adapted meshes by means of a quadtree type method.

Table 21.1 gives the results of the adaptation for this example. The values np and ne are the number of vertices and the number of elements in the mesh, while Q_T and $\overline{Q}$ note the worst and the mean qualities of the triangles. The minimum, maximum and mean edge lengths are respectively denoted by l_{min}, l_{max} and l_{avg}, while τ is the efficiency index.

Remark 21.20 *Note that this type of method (based on a tree) favors the creation of small edges (in the metric) and, more specifically, in this analytic example in which the mesh gradation (by means of the [2:1] rule) is not really compatible with the given metric map. Nevertheless, the spatial tree decomposition has captured the required map, as can be seen by observing the value of index τ.*

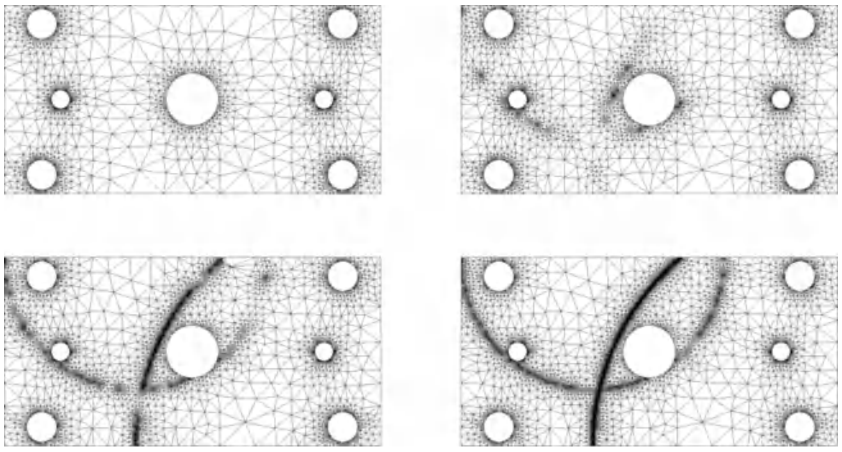


Figure 21.15: *Adapted meshes of a mechanical device using a quadtree-type method (iteration steps 0, 1, 3 and 10) for an analytical isotropic metric map.*

The second example concerns a CFD. case, in two dimensions. The problem deals with a viscous calculus around a NACA012 type profile at Mach 0.95 with a Reynolds $5,000$. A characteristic configuration of the *fish tail* is sought with an instationarity due to shocks wakes interactions. The mesh adaptations allow the shock regions, the boundary layers and the wake to be captured. Figure 21.16) shows the adapted meshes and the corresponding density iso-contours at iterations 0, 3 and 6 (six adaptation steps were necessary to capture the physics). A type method (with a point insertion scheme using the edges of the current mesh; see Chapter 7) was used for the global remeshing at each iteration step and a Navier-Stokes solution method was used.

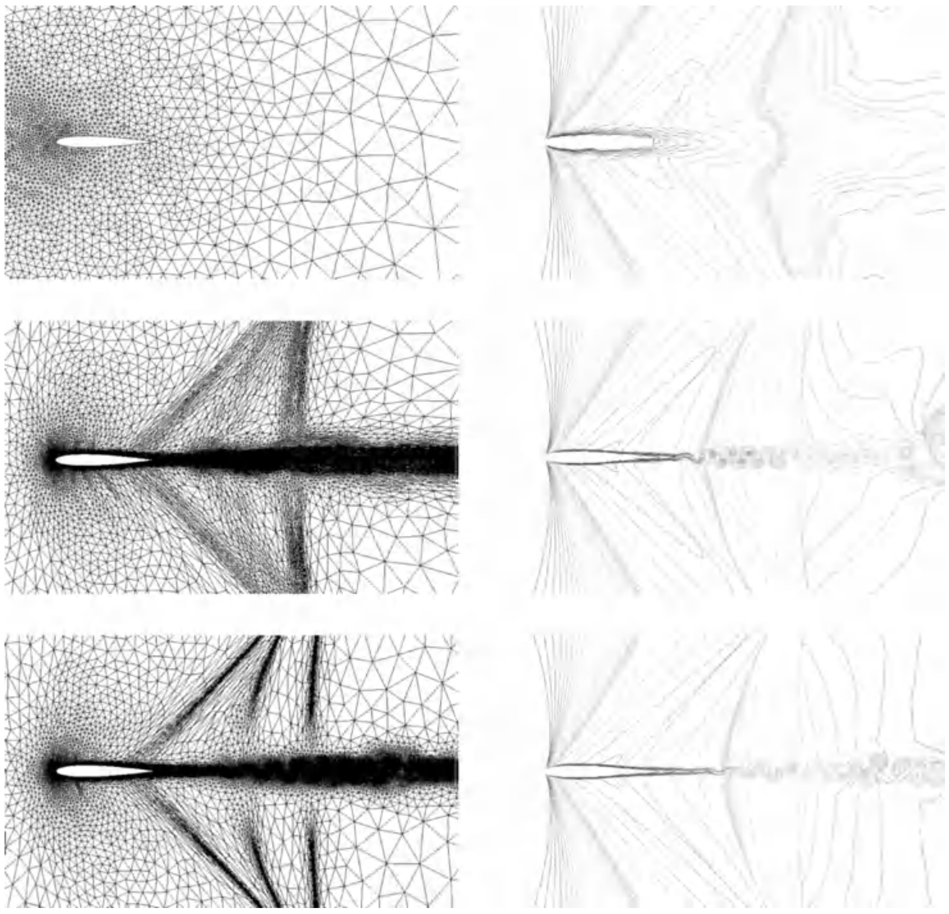


Figure 21.16: *Original mesh (top left-hand side) and adapted meshes using a (anisotropic) Delaunay-type method around a wing profile (NACA012) at iteration steps 0, 3 and 6 for a Navier-Stokes computation in CFD with the corresponding density iso-contours.*

To conclude, we give examples of adapted meshes in three dimensions. In the first example, the shape of the domain is deliberately simple[13], a sphere with a unit radius centered at the origin and an analytic isotropic metric map[14] is used. Figure 21.17 shows the surface meshes at steps 0, 1 and 7 in the adaptation. Figure 21.18 shows the volume meshes (and some cuts by various planes) at the corresponding steps.

An isotropic mesh optimization procedure was used for the surface remeshing [Frey, Borouchaki-1998]. A Delaunay method was employed to adaptively remesh the volume [George, Borouchaki-1998].

Table 21.2 gives the main figures for the different iteration steps. The values

[13]Note that the visualization of map in three dimensions is delicate or even not really possible

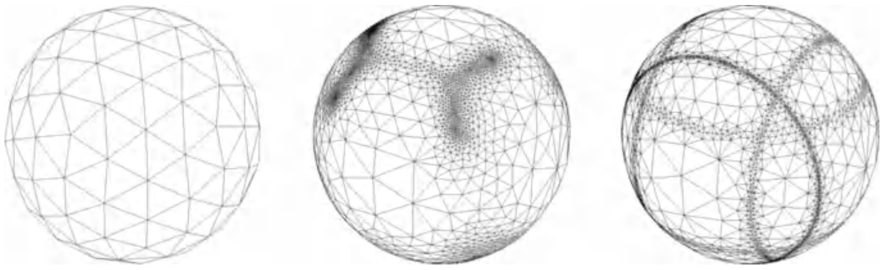


Figure 21.17: *Adaptation examples in three dimensions. Isotropic adapted surface meshes at iteration steps 0, 1 and 7.*

-	np	ne	τ	$l_{\%}$	Q_T	$1-2$	$2-3$
Initial mesh	277	1,200	0.515	7	1.8	100	-
Iteration 1	23,023	124,362	0.814	37	47.	81	11
Iteration 3	115,215	647,119	0.944	78	12.	78	20
Iteration 7	253,068	1,416,617	0.961	86	8.	74	24

Table 21.2: Statistics about the different iteration steps.

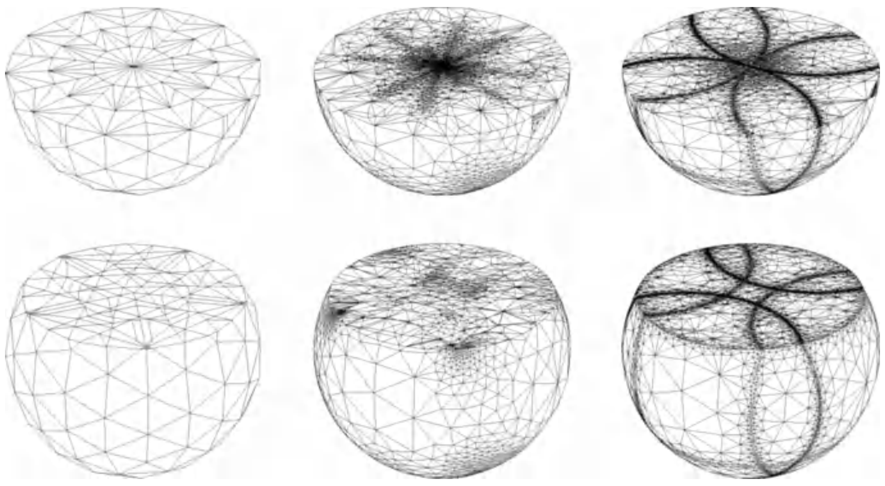


Figure 21.18: *Adaptation examples in three dimensions. Isotropic adapted volume meshes at iteration steps 0, 1 and 7, cuts by the plane $z = 0$ (top) and by the plane $z = 0.5$ (bottom).*

in the case of arbitrary complex geometries.

[14]While used as a discrete data in order to emulate what a concrete case is.

$l_{\%}$ denote the percentage of edges whose lengths are compatible with the size map (i.e., such that $\sqrt{2}/2 \leq l \leq \sqrt{2}$), Q_T, $1-2$ and $2-3$ denote the worst quality in the mesh and the number of elements with a quality between 1 and 2 and between 2 and 3, respectively).

The next examples, [Frey, Alauzet-2005] and [Dobrzynski *et al.* 2005], show concrete cases of isotropic or anisotropic mesh adaptation.

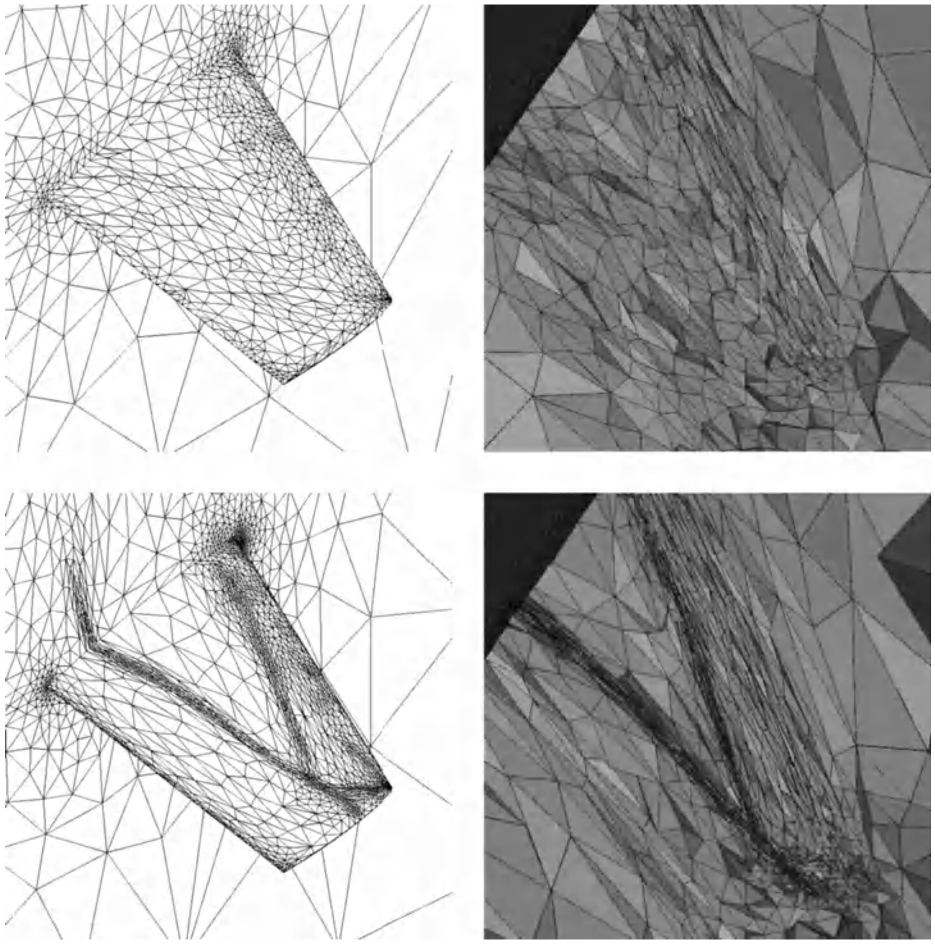


Figure 21.19: *Anisotropic adapted surface and volume meshes (iteration 0 and 9) for a transonic Euler case (data courtesy of F. Alauzet).*

The first concrete example concerns a classic numerical simulation of transonic air flow around the ONERA M6 wing. A Euler solution is computed for a Mach number equal to 0.8395 with an angle of attack of 3.06 degrees. This transonic simulation case gives raise to a well- known lambda-shock. The initial mesh is a relatively coarse mesh containing 7,815 vertices, 5,848 boundary triangles and 37,922 tetrahedra. The variable used to adapt the mesh is the Mach number.

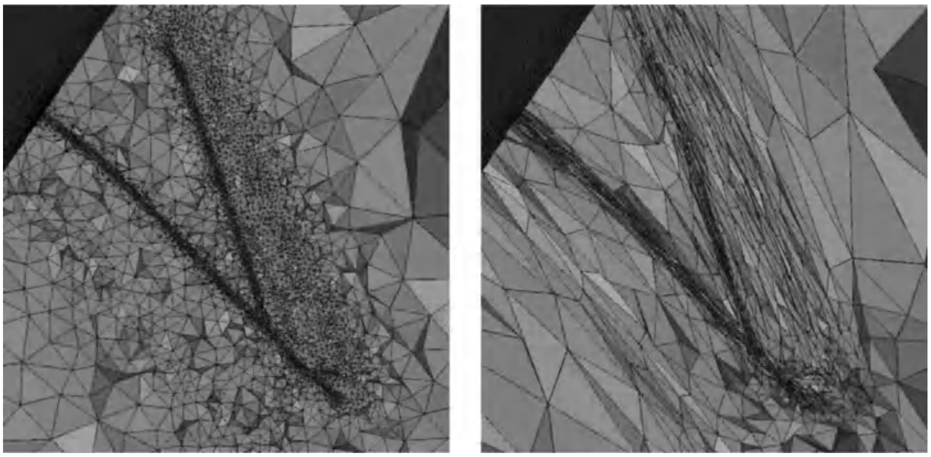


Figure 21.20: *Isotropic versus anisotropic adapted volume meshes for a transonic Euler case (data courtesy of F. Alauzet).*

The mesh has been adapted 9 times, every 250 time steps. Figure 21.19 shows the adaptation in the anisotropic case. In this example, the maximal aspect ratio achieved for the anisotropic elements is about 10. Nevertheless, the anisotropic metric leads to a drastic reduction of the number of degrees of freedom, roughly one order less than in the isotropic case (for the same error level).

Figure 21.20 compares the final isotropic mesh (iteration 9) containing $231,113$ vertices and $1,316,631$ tetrahedra and the above final anisotropic mesh containing $23,516$ vertices and $132,676$ tetrahedra.

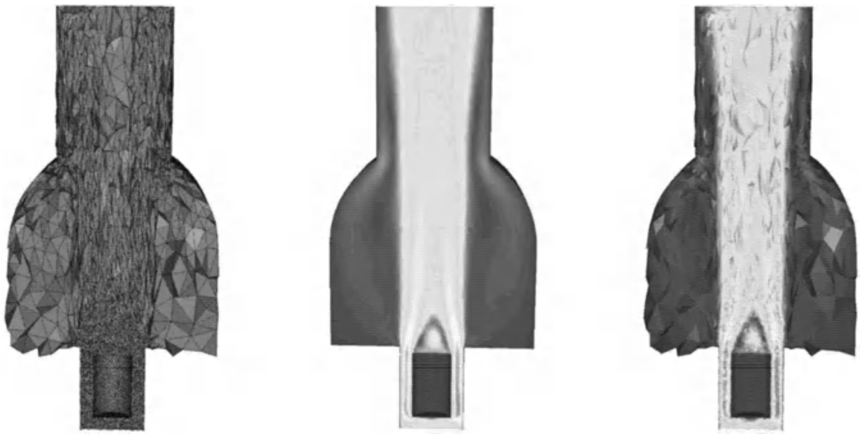


Figure 21.21: *Anisotropic adapted mesh and density values for the air cooling simulation (data courtesy of CEA, Cadarache, and C. Dobrzynski).*

Figure 21.21 demonstrates the use of local anisotropic adaptation for air-cooling problem for the design of a nuclear spent fuel. This simulation involves a weak coupling between the Navier-Stokes and an advection-diffusion equations. The inflow velocity, under the canister is $2\,m.s^{-1}$. There is free flow at the top of the chimney and all walls are considered viscous walls. The Reynolds number is 18 000. The flow is displayed at time $t = 52,5\,sec.$ The cylinder which represents the nuclear waste canister has been considered as a heat source (the temperature of the waste is about 500 degrees) and the input air represented a cold source (the temperature of this source is the external temperature that is about 20 degrees). Neumann conditions have been prescribed on the remaining part of the domain.

Chapter 22

Mesh Adaptation and P or Hp-methods

P-methods and hp-methods represent solutions to mesh adaptation which provide, in some way, an alternative to h-methods. In brief, a p-method, in contrast to a h-method (Chapter 21), consists of varying the degree p in the approximation while keeping the mesh sizes h unchanged. In this way, the quality (the richness) of the approximation is adapted to the way in which the solution varies.

Well established for some model problems, particularly when the geometry of the domain is relatively simple, p-methods are somewhat delicate to implement when the geometry is really complex. The fact that the size is constant while the degree of the approximation is the sole parameter results in a strong constraint which is not simple to overcome. Therefore, hp-methods have been introduced which combine a p-adaptation with a h-adaptation and thus offer the advantages of both methods while avoiding the unflexibility mentioned above.

★
★ ★

This chapter is organized in two distinct non-symmetric parts. The first part concerns the construction of finite elements other than straight (linear) elements (P^1 type elements) which are naturally obtained by using automatic mesh generation methods. The second part discusses p and hp-methods, mainly with regard to an adaptive process.

This last aspect will only be dealt with briefly. In contrast, we discuss in greater detail some possible approaches for the construction of finite elements other than P^1, which are used in p-methods (where p varies) and also in all calculus with finite elements with a (constant) degree p other than 1. This type of construction, mostly seen in the case of P^2 triangles, leads us firstly to consider how to mesh a curve (assumed to be planar for the sake of simplicity) by means of parabola arcs. We then turn to the creation of P^2 elements using this curve mesh as input data. In this context, we present several approaches which essentially involve adequate post-processing a P^1 type mesh. As will be seen, various technical difficulties can be expected. Some examples of such problems are given in three dimensions.

Let us recall that we are mostly concerned with the impact of the solution methods on mesh generation techniques. Therefore, we mainly focus on this particular aspect and less on using p-methods for solution control from an adaptive point of view.

22.1 P^2 mesh

We consider here a mesh where the elements are P^2. For the sake of simplicity, we limit ourselves to a problem in two dimensions. First, we discuss how to mesh a curve (a domain boundary in this case). In other words, our concern is to show how to obtain a polygonal approximation where the elements (the segments) are parabola arcs. We then show how to construct P^2 elements.

Meshing a curve (review)

Let us consider a parametric curve Γ whose equation is $\gamma(s)$, where s is the curvilinear abscissa. In addition, we assume that the curve has the required regularity when needed. In Chapter 14, we saw how to discretize such a curve by means of line segments (a P^1 mesh), in such a way as to control the gap between the curve and its discretization. Now, we return to the same problem in the case of a P^2 discretization, say one composed of parabola arcs.

Local behavior of a curve. We give again, with the same notations, the expansion of Relation (14.4), for example, in some vicinity of $\gamma(s_0)$ and for small enough Δs:

$$\gamma(s) = \gamma(s_0) + \Delta s\,\vec{\tau} + \frac{\Delta s^2}{2\,\rho(s_0)}\,\vec{\nu} - \frac{\Delta s^3}{6\,\rho(s_0)^2}(\rho'(s_0)\,\vec{\nu} + \vec{\tau}) + \ldots .$$

The curve can be approximated using the first three terms in its expansion at ε, if we take $\Delta s = \alpha\,\rho(s_0)$ such that (Relationship (14.7)):

$$\alpha \approx \frac{\sqrt{6\,\varepsilon}}{{}^4\sqrt{1+\rho'(s_0)^2}} .$$

Also in Chapter 14, we saw that replacing the parabola for approximation (of the expansion) by means of a line segment discretization leads to a gap between this discretization and the curve in the order of:

$$\delta \approx \left(\frac{\varepsilon\sqrt{6\,\varepsilon}}{{}^4\sqrt{1+\rho'(s_0)^2}} + \frac{3\,\varepsilon}{4\,\sqrt{1+\rho'(s_0)^2}} \right) \rho(s_0) .$$

In this estimate, the first term controls the gap related to the approximation by the expansion while the second term controls the discretization gap.

An approximation of type P^2 consists of discretizing the parabola[1] by means of of parabola arcs, thus by itself. The mesh element in the discretization is then the portion of parabola passing through the three points $\gamma(s_0)$, $\gamma(s_0 + \frac{\Delta s}{2})$ and $\gamma(s_0 + \Delta s)$. The above relation is then:

$$\delta \approx \left(\frac{\varepsilon\sqrt{6\,\varepsilon}}{4\sqrt{1+\rho'(s_0)^2}} + 0 \right) \rho(s_0)$$

and therefore the total gap has the order of the first terms (as the gap in discretization is now zero). This gap is then:

$$\frac{\varepsilon\sqrt{6\,\varepsilon}}{4\sqrt{1+\rho'(s_0)^2}}\,\rho(s_0)\,.$$

Thus, for a given relative tolerance of ε, the result obtained in this way is much more precise than in the case of a P^1 approximation. As a consequence, the analysis may be based on a coarser approximation of the expansion. Indeed, we can look for a value of ε' such that:

$$\frac{\varepsilon'\sqrt{6\,\varepsilon'}}{4\sqrt{1+\rho'(s_0)^2}} \approx \frac{\varepsilon}{4\sqrt{1+\rho'(s_0)^2}}\,,$$

which means:

$$\varepsilon'\sqrt{6\,\varepsilon'} \approx \varepsilon\,.$$

An easy computation gives:

$$\varepsilon' \approx \sqrt[3]{\frac{\varepsilon^2}{6}}\,. \tag{22.1}$$

To give an idea, a tolerance value $\varepsilon = 0.01$ leads to $\varepsilon' = 0.025$ while $\varepsilon = 0.001$ leads to $\varepsilon' = 0.0055$. Therefore, we can take a larger tolerance value while obtaining a nice control. In the case of a circle, for $\varepsilon = 0.025$ we find $\alpha = 0.391$, this shows that about 16 mesh entities are necessary to obtain a control in 0.01 (in fact, somewhat better), while in comparison, such a control in P^1 requires using 26 mesh elements (Chapter 14).

Exercise 22.1 *Let us consider the circle of radius ρ (centered at the origin, for simplicity). Discretize it by means of* 16 P^2 *mesh elements such that each of these parabola arcs has its endpoints A and B and its midpoint M in the circle. Then, compute:*

- *the gap between the chord AB and the circle,*
- *the gap between the chord AM and the circle,*

[1] A method could be conceived, where the expansion is pushed one order higher, in order to construct as a discretization parabola arcs lying on the curve defined by the first four terms in the expansion. In this case, it would be necessary to return to the entire discussion, as in Chapter 14.

- *the gap between the parabola arc and the circle (thus, this value at the mid-point of the mid-arc, the arc related to AM).*

Conclusions? (Hints: find the value α associated with the mesh element AB, deduce the value of ε giving the gap as a ratio of ρ, repeat this for the mesh entity AM and perform the explicit calculus for the arc). Observe what is obtained by changing the number of mesh entities, for example, for a number like 8 *or* 32.

Remark 22.1 *In terms of continuity, the P^2 discretization thus-obtained is rather good.*

Remark 22.2 *The above reasoning analyzes the behavior of the curve using an expansion between s_0 and $s_0 + \Delta s$. It is clear that a similar study can be made between s_0 and $s_0 - \Delta s$, thus leading to the same conclusion about the resulting gap. As a consequence, for the same gap, it is possible to create a mesh entity twice as large $(2\,\alpha\,\rho(s_0))$ by centering it at the point $\gamma(s_0)$. As previously seen, the accuracy remains in ε. Nevertheless, the continuity from mesh entity to mesh entity is slightly modified.*

Geometric mesh of a curve. By analogy with Definition (14.1), a P^2 geometric mesh can be defined as follows.

Definition 22.1 *A P^2 type* geometric mesh *at a given ε of a curve Γ is a parabolic piecewise discretization of this curve whose relative gap is in the order of ε at any point. If Δs is the length of an arc of the curve and if h is the length of the corresponding parabola arc, then, h tends towards Δs with Δs.*

Hence, we return to the same definition as for P^1 meshes and thus the same characterization of this property.

As seen for Definition (14.1), notice that this is a reasonable definition, i.e., related to the difference in lengths or, similarly, related to the relative gap to the curve. This definition is only one possible formulation of the notion of a P^2 geometric mesh. Indeed, another definition may be, for example, to observe the value if the surface area comprised between the arc of curve the the arc of parabola which approaches this curve.

After the previous discussion and with regard to the proposed definition, constructing a geometric mesh of a smooth curve is made by computing the length of the curve using the metric of the main radii of curvature weighted by adequate coefficients α. As a corollary, meshing a curve leads to:

- finding the inflection points in this curve together with the singular points (corners),
- imposing these points as mesh vertices,
- splitting the curve into parts, each of which is bounded by to such consecutive points,

- meshing each part by using the previous principle.

Remark 22.3 *Identifying the points with a maximal curvature and prescribing them as arc midpoints enables us to have a nice regularity in these points. Nevertheless, a reasonable result is also obtained when the points of inflection are not explicitly considered.*

As a conclusion, we have briefly described a method that allows the analysis of P^2 meshes for curve meshing (in $\mathbb{R}^2$). This analysis can be followed to give idea about how to conceive a mesh generation method. Here is only one of the possible approaches and clearly other methods may be envisaged. Moreover, the actual implementation of the above principles is not necessarily obvious. Nevertheless, the remarks given in Chapter 14 may serve as a source of inspiration.

Meshing a curve. For this problem, the mesh construction of a curve with or without a metric map (not necessarily of a geometric nature), we refer the reader to Chapter 14. When no metric specifications are known, a smooth variation in size for two neighboring mesh elements is a natural target. On the other hand, when a specification of a physical nature (i.e., related to the physics of the problem in question) is given, it is important to make sure that the sizes specified by the physics remain compatible with those related to the geometry.

P^2 finite elements

For the sake of simplicity, we consider a planar case (in two dimensions) and we consider the construction of P^2 type triangles. We assume the domain to be meshed to have a curve (a series or a number of curve parts) as a boundary. The question is then to define, specifically for the triangles having at least one boundary edge, the nodes of these triangles together with their edges (other than the boundary edge(s)). In terms of mesh construction method and *a priori*, there are two ways of addressing this type of problem:

- a type P^1 mesh with step h_1 (see the remark below) is available which is then transformed in a P^2 type mesh with step h_2 where $h_2 = h_1$;
- a type P^2 mesh is constructed directly. This approach initially requires meshing the domain boundaries by means of P^2 mesh elements, as above, and then creating the P^2 triangles which form the final mesh of the domain.

The second approach implies the construction method to be P^2. The problem is no longer, for a given edge, to find the point suitable for the construction of a P^1 triangle (thus with a purely geometric nature) but to deduce from a P^2 edge, the three points and the two other edges required to define a P^2 triangle.

Keeping in mind the discussions about the main mesh generation methods, it is then clear that the corresponding algorithms (*quadtree-octree*, advancing-front or Delaunay, for example) are in essence purely of a geometric nature. Developing methods directly resulting in P^2 type elements is then probably possible but presumably not so obvious.

Following these remarks, the first approach is more reasonable in practice. Thus, we ask the mesh generation method to produce a geometric mesh (thus, a P^1 mesh) and then this mesh is modified into a P^2 type mesh by means of post-processing.

This way of processing merits some comments, the first regarding the sizes of the mesh elements that are to be taken into account.

Remark 22.4 *Let h_1 be the size of a P^1 mesh element and let h_2 be that of a P^2 mesh entity, then fixing $h_2 = h_1$ implies that the number of P^2 nodes (the element vertices and the edge midpoints) is the same as the number of P^1 nodes (the element vertices) in the P^1 mesh obtained by subdividing into four the P^2 triangles. Following what we saw about the geometric approximation of a curved boundary, it is then possible to take, for a similar accuracy, a step h_2 twice as large as step h_1. Therefore, in this approach, it is possible to take, when constructing the P^1 mesh, a relatively large step and, in particular, start from a mesh which is a relatively coarse approximation of the boundary geometries.*

Doing so, a P^1 mesh is available whose steps may be large. It is easy to see, for a boundary edge, that if the difference with the geometry is relatively small, there is no major difficulty in finding the mid-node required to transform this segment into a parabola arc. After which, in such a case, the difficulty when constructing the other edges is related to the local configurations.

A second remark must be made about this approach based on the modification of a P^1 mesh so as to complete a P^2 mesh. The local aspect of the approach implies that it is not strictly identical to a curve mesh construction method.

Remark 22.5 *Processing by means of transformation considers the boundary edges element by element while the curve meshing problem considers the entire curve. Therefore, the processing by element approach does not* a priori *see what happens in some vicinity of the element in treatment. This may result in a possible lack of smoothness of the curve thus-obtained. To overcome this trouble, it is necessary to have access to some global information such as tangents, normals, etc. Moreover, the two endpoints of the edge under examination are prescribed while these points are not necessarily optimal with regard to the meshing problem applied to the entire boundary curve.*

Despite this, up to now, it remains realistic to follow the local approach. Indeed, if the useful geometric information is known at each point (by means of queries to a geometric modeler or to a mesh assumed to give a suitable geometric definition, the *geometric support*, to some extent, it is possible to give a global aspect to the local process. The only negative feature nevertheless remains the constraint regarding the two edge endpoints that are imposed at the extremities of the arc of the curve to be constructed.

Nevertheless, in the following, we follow this principle, i.e., the construction of P^2 elements by local transformation of P^1 elements. However, before going further, we make some comments on the second approach, i.e., what a P^2 type mesh generation method could be.

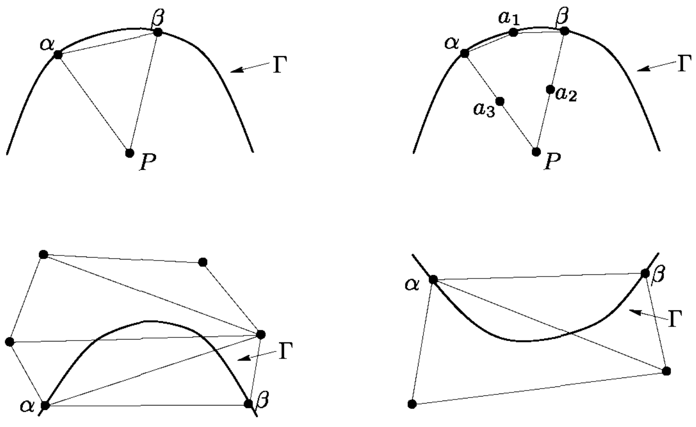


Figure 22.1: *Construction of a P^2 finite element. Top: the P^1 element as constructed by using a mesher is such that the construction of the corresponding P^2 element is "immediate". Note that the gap in P^1 is relatively poor while the gap in P^2 is rather better. Bottom: the constructed elements must be processed by a more sophisticated process if we wish to transform them into P^2 elements.*

Remarks about direct construction of a P^2 mesh. For simplicity, we focus on a triangular mesh and we assume that the boundaries of the domain to be meshed have been previously discretized using the curve meshing method described above.

Therefore, the vertices of the boundary meshing entities are assumed to be adequately located. In other words, the domain boundaries are suitably approximated.

The problem in question is then the direct construction of P^2 triangles using this discrete data input as information. The immediate question is the following:

"What is the pertinent quality criterion that must be used to govern the P^2 mesh construction?"

For "classical" mesh generation methods (from Chapter 4 to Chapter 8), while limiting ourselves to a isotropic situation, we know that the optimality criterion is the fact that the triangles are equilateral. What becomes this criterion in the present case? Before giving some ideas about this, let us mention that the literature is rather weak on this point. Since abstract results (about convergence and error estimates) make use of the fact that parameter h, the element size, tends towards zero, they fail to give particular indications about realistic cases where this value is not necessarily small.

It seems evident (and intuitive) that if the gap between the boundary edge (linear approximation of the boundary curve) and the boundary itself is small enough, then the reasoning used in P^1 can be followed and the optimality criterion remains unchanged. The case of interest is when this gap, for a linear approximation, is relatively large and when this gap, in P^2, is judged to be good enough (the geometry is suitably approximated). This implies that the optimal point related to the

P^1 edge which allows the creation of an equilateral P^1 triangle is not necessarily optimal when a P^2 triangle must be defined. This depends on the curvature of the boundary in the region of interest. In other words, given that a curved edge suitably meshed is not sufficient, some other conditions must be considered.

By analogy with the equilaterality criterion held in P^1, we can say that a P^2 triangle is satisfactory if the angle bounded by the two tangents in its three vertices is close to 60 degrees. If the edge is a curved edge, the tangent in question is the tangent to the curve, if the edge is a line segment, the tangent is the edge itself.

It is easy to see that this criterion results in a restriction on the length of the curved edge and makes it possible to find a reasonable position for the third vertex whatever the local concavity may be.

This being satisfied, the vertex we are seeking can be located at the intersection of the two line segments which form the above angle with the tangents to the curve. Then, the two other edges are defined using these line segments. Either the edge in construction belongs to the adequate line segment or it must be curved and this line segment is only its tangent. In such a case, it is necessary to construct a portion of a parabola using as the input data one of its endpoints, the tangent at this point and the length.

The example in Figure 22.2 shows, in comparison with a classical P^1 mesh (left-hand side), the P^2 mesh obtained by a direct construction method (see the curved boundaries).

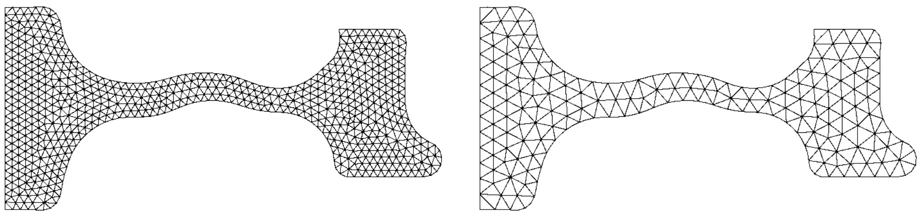


Figure 22.2: *P^1 and P^2 planar meshes. On the left, a P^1 triangular mesh with a uniform element size and, on the right, the P^2 mesh resulting from a direct construction with twice the element size.*

22.2 P-compatibility

The above discussion and the simple examples in Figure 22.1 allow us to make the notion of p-compatibility more precise. In the following and for the sake of simplicity, we only consider the case where $p = 2$.

Definition 22.2 *A geometric element (a P^1 element) is said to be* 2-compatible *if the distance between its boundary edge(s) and the boundary curve is such that there exists a point in the boundary for each of this(these) edge(s) that makes it* *possible to construct a valid P^2 element by a simple post-processing of the given P^1 element.*

This is a naive definition (somehow a tautology) which indicates that the simplest construction (find a point on the curve and construct a portion of parabola joining the two edge endpoints and passing through this point) results in a valid element. This means that there is no interference in the various edges (curved or straight sided). This implicitly implies that the new points (the "midpoints") remain close to the initial straight-sided segment and thus do not create any problem for the elements neighboring the element under consideration.

Note that the theory about error estimates and convergence issues (Chapter 20), deliberately considers a situation of this type. From a practical point of view, a 2-compatible (straight sided) element is easy to transform into an element of degree 2. Nevertheless, this does not mean that an element not in this case cannot be transformed into an element of degree 2 (as will be seen below).

P-compatibility *a priori*

Such a case is ideal. The mesh generation method completes P^1 elements which are 2-compatible. Then, it is sufficient to find the edge "midpoint" and to define the corresponding parabola arc. If the straight edge-boundary edge distance is small and if the curve is close to a parabola (or enjoys a certain symmetry with respect to the perpendicular bisector of the straight segment), the point we are seeking can be simply chosen as the projection of the edge midpoint onto the boundary.

In the case where the boundary curve does not satisfy this property regarding regularity and symmetry, the sought point *is not* the projection of the edge midpoint (return to the section about how to mesh a curve by means of parabola arcs while minimizing the gap).

Keeping this in mind, how can we ensure that a P^1 mesh satisfies *a priori* all the underlying requisites? The simplest idea is to construct a mesh whose elements having a boundary edge(s) are such that this(these) edge(s) is(are) close to the boundary and, moreover, are such that the third vertex leads to an element which is almost equilateral. This is exactly the objective of classical mesh generation methods. Most of the elements must be equilateral. The immediate question that occurs is to know if any geometry can be approached with such triangles? The answer is probably yes when a fine enough mesh is used, and probably no if not.

Remark 22.6 *Imposing a rather small size in order to obtain such a mesh is, as previously seen, not strictly necessary. The resulting P^2 mesh is, in general, too fine and thus dealing with such a mesh requires an unnecessarily high cost.*

Following these remarks, it is possible to replace the criterion about equilaterality by a similar criterion related to the angles formed by the tangents to the edges. Thus, given a boundary edge (a straight sided segment as we are in P^1), its angle with the next edge in the triangle is measured by taking into account the underlying boundary curve. Note that this criterion is not one of the criteria usually considered in classical mesh generation methods.

In conclusion, it is far from certain that this approach is the most suitable in all cases. In the following section, we propose another way to address this problem.

P-compatibility *a posteriori*

We consider here that we have an arbitrary P^1 mesh and we want to transform it into a P^2 mesh. The simplest idea consists of returning to the previous case. All edges (all situations) leading to a difficulty in the desired construction are modified so to suppress this difficulty. The possible modifications (here in two dimensions) that can be used are those described in Chapter 18. Thus they are based on an adequate combination of tools for mesh optimization such as point relocations and edge swaps.

The local configuration (an element and its neighboring elements with regard to the boundary edge under examination) to be dealt with is analyzed. Several situations may be found:

- the point in the curved edge to be added remains close enough to the corresponding straight sided edge and falls within the triangle having this edge or, again, falls outside this triangle (based on the local concavity) but has no influence with any of the neighboring elements,
- this point falls very close to one of the other edges in the triangle dealt with or again falls within a triangle other than the current triangle (one of its neighbors or, more tedious, an element that is not directly a neighbor of the triangle under examination).

In the first case, constructing a P^2 element is obvious. In the second case, we propose modifying the local neighborhood so as to return to a standard situation while observing that the present situation is related to the fact that a P^1 mesh too coarsely "violates" the geometry. Figure 22.3 depicts three examples of such situations.

Regarding how the local context must be modified so as to remove the difficulty during the construction, first, notice that in case i) and, to some extent, in case ii) an operation like an *edge swap* is not useful. In the first case, the resulting situation is similar to the initial configuration. In the second case, the new situation leads to having the node to be created very close to one of the other edges in the triangle (which results in a bad quality P^2 element or even a negative Jacobian for a very close configuration). A small change in the node position (by means of a moving point process) is then one possible way to move the new node away (by increasing the distance between this node and the edge). For the configuration shown in case iii), it is also possible to find a local strategy that suppresses the problem. First, it must be observed that using a local modification (point relocation or edge swap) does not allow us to remove the parasite point located outside the P^2 domain while being inside the P^1 domain. Thus, we propose (Figure 22.4), after [Dey-1997]:

- constructing the desired node M,

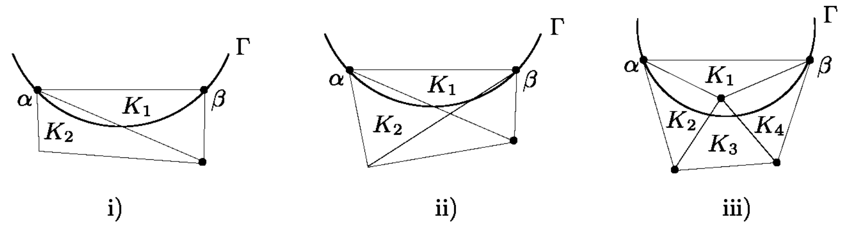


Figure 22.3: *Three local configurations where the transformation of a P^1 mesh into a P^2 mesh is not immediate because the geometrical approximation is too poor. In i) and ii), the point to be created falls outside the initial triangle, in iii) the situation is far more dramatic as the point to be defined is outside the triangle and, moreover, there are several "parasite" triangles (and even a point) which certainly impede the process.*

- constraining the two edges αM and βM in the P^1 mesh,
- classifying the resulting triangles according to their positions with respect to boundary Γ and classifying the mesh entities (points and edges) in internal entities and boundary entities. This classification is made using the inheritance based on the classification of the entities in the initial mesh while preserving a coherent result,
- deleting the exterior triangles (in this way, the parasite point disappears),
- analyzing the new context again.

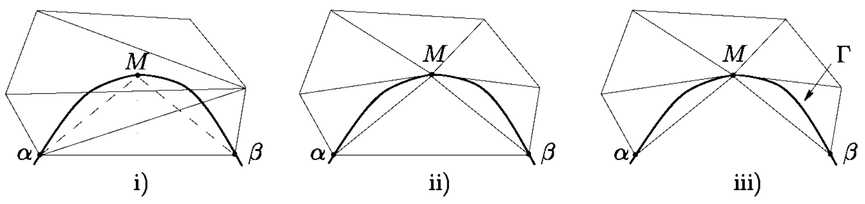


Figure 22.4: *The initial configuration and the predicted point M. This point is located in an element other than the triangle supposed to be a boundary element. The two boundary edges formed after inserting M are enforced. Consequently, the mesh is modified and the resulting elements are classified so as to find the new triangles that have a boundary edge.*

Notice that we encounter here an edge enforcement problem (as described in Chapter 7) and that the proposed method leads to what is expected while the resulting situation is not optimal. Indeed, if iterating this process, it could be necessary to subdivide the presumed boundary edges several times and, as a consequence, obtain an unnecessarily fine P^2 mesh. Note also that applying this heuristic is not trivial in three dimensions (as also seen in Chapter 7).

An example of a temporary pathology. When there are thick regions or when several boundary portions are close to one another (Figure 22.5), which are frequent situations in realistic cases, a difficulty may arise, which may be only temporary (this is not known *a priori*).

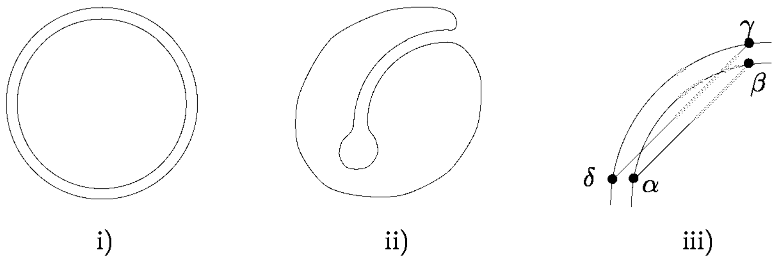


Figure 22.5: *Two local configurations where there is a thick region. The P^1 mesh is correct but, when constructing the P^2 mesh, auto-intersections at the boundary level are created, which may only be temporary. In fact, dealing with edge $\alpha\beta$ leads to an intersection with edge $\delta\gamma$, whereas if this edge is processed first, dealing with the other edge becomes possible.*

As the boundary is assumed to be correct (not self-intersecting), there is necessarily a suitable geometric mesh of this boundary. The problem is then to find the value of the threshold ε which ensures an accurate approximation and avoids self-intersections. The difficulty in finding the proper value of this threshold results from the fact that the points or the elements which are geometrically close to each other are not necessarily close in terms of the topology. One way to detect such a situation is to make use of a global structure like a *neighborhood space* (see Chapter 1) previously constructed in a proper way. Indeed, in this structure, it is demanded to store the mesh entities so as to, for a given entity, retrieve at a low cost the set of mesh entities falling in some neighborhood.

Given such a tool, the data structure is queried when constructing an element and the possible conflicts are detected. The strategy that may be used is then of a purely heuristic nature. Indeed, it is possible:

- to accept a temporary collapsing while maintaining the list of the entities concerned, prior to re-processing (i.e., subdividing) these entities. As the solution exists, it will be found (at least, we hope so);
- not to construct such a situation (which means we temporarily ignore the entity concerned) in the hope that the pathology will disappear when dealing with some other entities.

As a conclusion, obtaining p-compatibility *a priori* or *a posteriori* makes sense only in cases which are already relatively close to the solutions and, in practice, the interesting cases are precisely those which are pathologic and, due to this, are bad candidates for such processing. Also, we have seen that looking for proper compatibility may lead (after some heuristics) to a valid result but one which is probably not optimal (in terms of the number of mesh entities). Therefore, other methods must be investigated (see below).

22.3 Construction of P^2 elements

We first restrict ourselves to a planar situation.

Element 2-compatible

Following the definition of the notion of compatibility, the transformation of a P^1 mesh into a P^2 mesh reduces to locally modifying the initial mesh, i.e., element by element.

Other elements

In such a case, a purely local processing, since the processing only of the boundary edge is not sufficient, does not lead to a solution. In particular, it is necessary, for "curving" the boundary edge under treatment, to curve one or several other presumably internal edges. Intuitively, this consists of "defining" the empty region necessary to obtain a suitable construction. In practice, this approach can be seen as an optimization method in which a certain propagation (a deformation between the neighboring entities) from entity to entity is necessary so as to achieve the desired result.

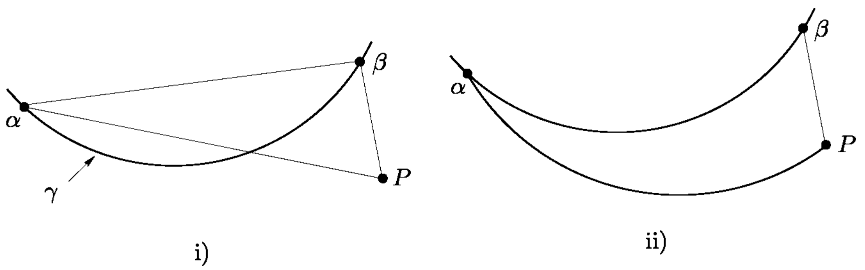


Figure 22.6: *A local configuration where it is necessary to curve an* a priori *straight-sided edge so as to have enough space for the boundary edge which is to be processed.*

This meshing problem is a problem with constraints (Figure 22.6). Edge $\alpha\beta$ (parabola arc) having been constructed, it is necessary to construct a curved edge,αP, as far as possible (in accordance with the local configuration) so as to obtain a P^2 element of reasonable quality. It should be noted that this mesh deformation may lead to processing some other edges in the neighboring elements. Here, we can also use the tangents to the relevant edges.

Dealing with a surface element

We turn now to the same problem for a surface mesh. The targeted application is clearly the construction of surface meshes but also the construction of the "curved" faces in three-dimensional meshes of type P^2.

Following an observation already given, meshing a surface or transforming a (planar) triangular face into a curved triangular face (belonging to the same surface) are not, *stricto sensu*, identical problems. In the first case, the surface is globally considered, in the second case, the process is a local one.

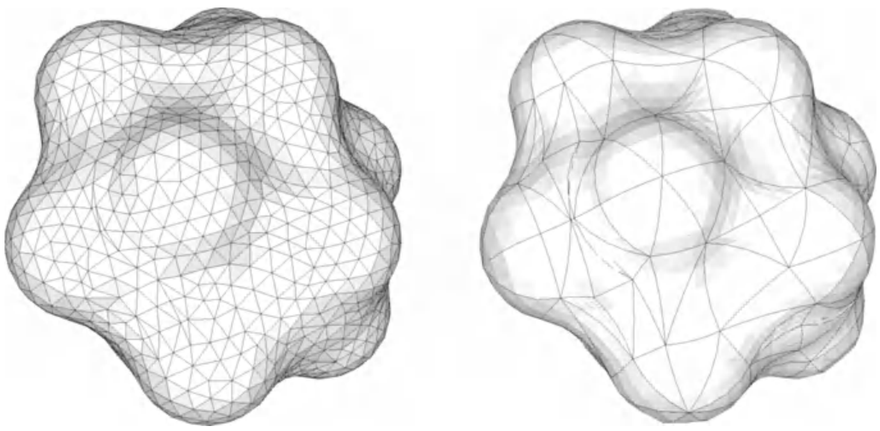


Figure 22.7: *Anisotropic meshes of a molecule of* cyclohexane. *Left-hand side, mesh with P^1 triangles controlled by an angular tolerance of 8°; right-hand side, a P^2 mesh for a tolerance of 32°.*

Meshing a surface by means of P^2 triangles. In this case, the principles of mesh construction described in Chapter 15 may be used. The method involves finding the (curved) edges of the triangles that ensure that the surface mesh such-created is a suitable approximation of the real surface. We have seen that this problem does not reduce to ensuring that the curved edges are adequately close to the surface. Indeed, controlling the gap between such an edge and the surface does not automatically imply that a triangle with these edges as sides is adequately close to the surface, for the same given threshold value. In addition, the interface between two adjacent triangles must have, in some cases, a certain extent of smoothness thus implying, at least, that the edge discretization must not only take into account the gaps (in terms of distance) to the surface but must also consider a control regarding the gaps related to the variation of the tangent planes or even consider some other geometric properties (variation in the curvature, radii of curvature, etc.).

As for a planar P^2 triangle, it is vital to control the size and the direction of the edges using the tangent plane at each vertex (the effect of the tangent in a planar case is now replaced by these various tangent planes).

An example of a P^2 surface mesh is depicted in Figure 22.7, right-hand side (BLSURF software).

Transforming a P^1 surface mesh into a P^2 mesh. In such a case, the geometry is assumed to be known (via a modeler, a mathematical description or a representative mesh) and, in addition, that a P^1 mesh is available. The question is then to transform this mesh into a P^2 mesh.

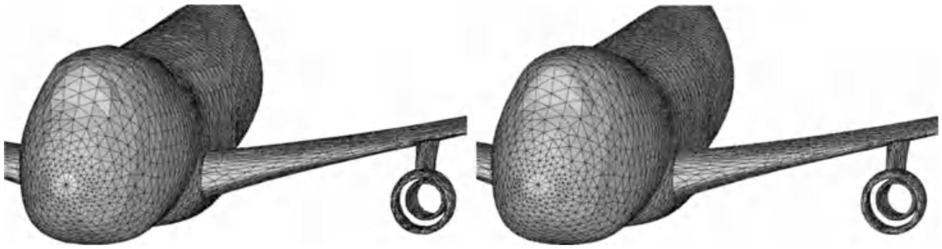


Figure 22.8: *P^1 and P^2 meshes of a Boeing 747. The mesh on the left results from a mesh simplification at 85% of a very fine surface mesh (original data provided by Boeing). The mesh on the right is the transformation of the P^1 triangles into P^2 triangles.*

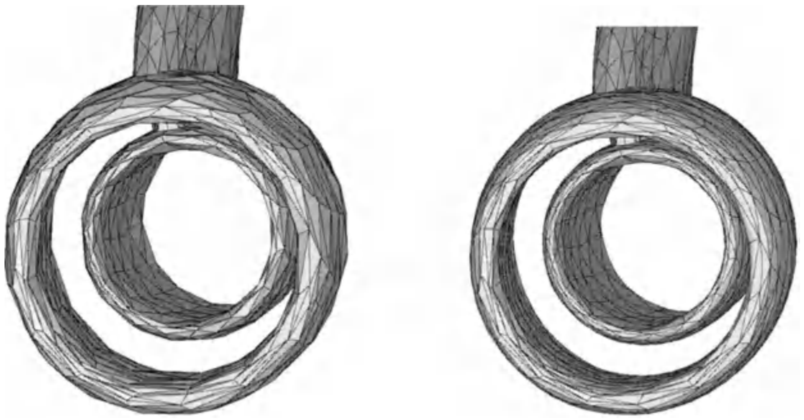


Figure 22.9: *Enlargement of part of the P^1 and P^2 meshes of the previous figure. It is easy to see the quality of the geometric approximation of the surface in the P^2 mesh as compared to the linear approximation seen in the P^1 mesh.*

We return to the above discussion. If the P^1 mesh follows the right properties (control by the tangent planes) then the transformation is of the same nature.

The construction is indeed relatively easy to do. The examples in Figures 22.8 and 22.9 are courtesy of H. Borouchaki. Otherwise, it may be necessary to refine the P^1 mesh based on the local curvatures in order to obtain a mesh that is easy to transform.

Volumic case, example of the P^2 tetrahedron

We now give some indications on the construction of tetrahedra of degree 2. We only focus on elements having one or several entities (edge or face) lying on the boundary. Obviously, the most interesting case is when this portion of the boundary is a curved region.

Element 2-compatible. As in two dimensions, a 2-compatible tetrahedron is relatively easy to transform into a curved tetrahedron with 10 nodes (see Chapter 20). The boundary face is curved (let us assume that there is only one such face) and, for the position of the three nodes to be defined, we return to a curve meshing problem, now in three dimensions. Note that meshing these curves, edges shared by several elements, must be such that the curved face and its neighboring faces form as regular as possible a discretization of the real surface (when this latter is regular). In other words, meshing a curve in this context requires considering the underlying surface.

It must be noticed that obtaining a 2-compatible mesh for a straight sided tet mesh forming a not necessarily close approximation of the real geometry is a non trivial problem and, in particular, may lead to unnecessarily small elements. On the other hand, the technique based on edge or face enforcement, local modification of the resulting mesh and classification of the new mesh entities (vertices, edges and faces) with respect to the boundary is in this case rather delicate (Chapter 7). The alternative approach, based on the control via the various tangent planes to the faces, gives better results (specifically, it is not necessary to refine too much the initial mesh and thus the number of elements remains reduced while the quality of the approximation is still within the desired range).

Other elements. An element (resulting from an automatic mesh generation method) which is a coarse approximation of the real geometry may, as in two dimensions, not be suitable for such an immediate construction. In particular, this may lead to curving some face *a priori* not in the boundary in order to obtain a region which is large enough to allow the construction of an element of adequate quality.

22.4 Elements of higher degree

The p-methods basically lead to modifying the degree p of the underlying polynomials until some values rather larger than 2 are obtained. In this respect, we find some examples where the value of p could be 5, 6, or more. Therefore, for example if $p = 3, 4, ...$, it is of interest to discuss, for the curves, how to construct arcs of cubics, quartics, etc. and, for the faces, how to construct some surfaces with the corresponding degree.

Let us just mention a few remarks about the case of a P^3 mesh entity, an arc of cubic, for a curve meshing case in the plane. We start again from a limited expansion until the term of degree 3. Then, assuming adequate hypotheses about Δs, the error due to stopping this expansion at this order can be appreciated by looking at the following term (at the order 4). This being done, we can follow the method described in Chapter 14 and we construct an arc of cubic in order to approximate the curve defined by the three first terms in the expansion, namely:

$$\Delta s\,\vec{\tau} + \frac{\Delta s^2}{2\,\rho(s_0)}\,\vec{\nu} - \frac{\Delta s^3}{6\,\rho(s_0)^2}(\rho'(s_0)\,\vec{\nu} + \vec{\tau})\,.$$

For a P^3 triangle, we find a construction similar to that used for a P^2 triangle while now controlling the tangents at the endpoints of the edges of the elements. In particular, one condition that must be verified is to ensure that the arc of cubic, the P^3 edge which approximates the curve, retains the same concavity between these two endpoints (thus implying a restriction on the length in the case where this concavity changes).

As a conclusion about these ideas of methods able to construct elements of an order higher than 1, it is clear that this topic is still a topic of interest which remains to be really well addressed (apart from the case where the curves or the surfaces remain smooth enough).

22.5 P-methods and hp-methods

In this short section, we briefly return to mesh adaptation methods based on a p or an hp approach.

P and hp-methods

Essentially, a p-method allows us, according to a criterion (resulting from an adequate error estimate), to predict the degree of the approximation which is appropriate with regard to the way in which the solution varies. Note that this degree changes during the computational process and also may vary from one element to another in the mesh. The literature on p-methods is extensive and, in particular, see [Babuska, Guo-1988], [Babuska *et al.* 1989], [Babuska, Suri-1990] and [Dey *et al.* 1997], (among many others).

Constructing a finite element of order p is not done as in the classical case. In other words, when p varies, we do not consider the set of finite elements (their basis polynomials) with the corresponding order p, but a finite element which is hierarchically defined. This being assumed, changing the degree involves adding the contributions of some shape functions whose degree is less than or equal to the chosen value and ignoring the contributions related to higher degrees.

An adaptation scheme

A possible (reasonable) scheme for an adaptation loop in terms of h or p comprises several steps, as for an adaptation in h only (as seen in the previous chapter). First, an initial mesh is completed using a classical method, while trying to obtain a mesh that is reasonably suitable for further use while, at this time, no precise information can be used regarding the order or the size of the elements (an error estimate *a priori* or some knowledge about the physical behavior may nevertheless be helpful in this construction). An initial solution is then computed using this mesh as a spatial support and an error estimate *a posteriori* analyzes its quality so as to estimate the sizes (h) or the degrees (p) adequate for the elements in the mesh of the next iteration step. The adaptation process then consists of constructing this mesh by taking these requests into account. Note, as for an h-method, that this adaptive process may be a local or a global process (nevertheless, the available examples are mostly local).

The new spatial support then serves to compute a new solution and a new error estimation is made and, until a given stopping criterion has been achieved, the iteration steps are continued.

The actual implementation of such a process includes two aspects which are slightly different. First, if the size parameter is affected, we return to the discussion in the chapter devoted to h-methods which describes various approaches that allow the construction (or the modification) of a mesh while following a given size map. Then, if the size remains unchanged, changing the order in the solution method results in two new problems:

- the construction of the "curved" entities in the mesh that are affected by the current step,
- the construction of the finite elements with the desired order p.

The latter aspect is out of the scope of this book, thus we advise the reader to return to the brief comment previously given and to the *ad hoc* literature (including the above references), while the first aspect leads us back to the preliminary discussion given in this chapter. Note that only the elements having a boundary entity located in a portion of the boundary which is curved are delicate to handle and that this may lead to processing some other *a priori* internal elements, in some neighborhood.

Chapter 23

Moving or Deformable Meshing Techniques

Nowadays, adaptive methods are widely used to solve partial differential equations which involve large solution variations like shock waves, boundary layers, etc. These techniques can now be considered as fully mature. Indeed, they have successfully demonstrated their ability in significantly reducing the computational cost of computer simulations while simultaneously preserving or even improving the accuracy of the numerical solutions. The main idea is to adapt the mesh points locally in order to concentrate them in the regions where the gradient of the solution rapidly varies (see Chapters 21 and 22). However, if the regions of interest are moving or evolving in time, such as fronts or shocks in hyperbolic equations, these techniques need to be modified to adapt the mesh with respect to time [Alauzet *et al.* 2007].

Regarding the numerical simulation of time-dependent PDE problems, mesh adaptation methods can be split into two categories, *static* or *dynamic*. The first class, corresponding basically to h-refinement techniques, consists of adapting the mesh by possibly adding nodes and updating node positions and interpolating variables from the old mesh to the new mesh, all these operations being performed at discrete time levels. Obviously, the principal attraction of these methods lies in their simplicity and, to some extent, their robustness in dealing with an arbitrary number of phenomena concurrently. However, a well known artifact is often noticed when dealing for instance with hyperbolic PDEs, the numerical dispersion. In addition, in order to preserve the time stepping accuracy, small steps are usually required. This class of methods has been thoroughly described in Chapter 21.

For dynamic methods, often called *moving mesh* methods or r-methods, the nodes are moved according to the solution of an equation involving velocities of nodes in order to preserve the concentration of nodes in regions of high variation of the gradient of the numerical solution. Typically, in arbitrary-Lagrangian-Eulerian (ALE in short) flow simulations, a boundary (or a part of the boundary) of a mesh follows a deformed feature. As the number of nodes is kept fixed during the simulation, problems may arise due to the excessive distorsion of the mesh, a phenomenon usually referred to as mesh tangling. Therefore, a mesh optimization stage is often required or a regularization term must be added in the movement equations to help, prevent or fix this situation.

★
★ ★

In this chapter, we will focus on meshing problems for physical or artificial phenomena for which time evolution and (topological and geometrical) shape modification are intrinsic features. For instance, when external domain boundaries remain fixed in shape while relative motion arises for internal objects, such as that undergone around pistons, pumps or jet engines. Examples in which the domain boundaries are also transformed often occur in crash simulation, metal forging, aeroelasticity as well as in many biomedical simulations. In such applications, the computational domain changes dramatically in shape, geometry and topology.

First, we deal with the motion of rigid bodies embedded in a computational domain, in two and three dimensions. In particular, we mention a practical solution to the problem of computing an admissible field of displacements at mesh vertices. Next, we briefly describe two methods to regularize a distorded or invalid (tangled) mesh in the context of ALE methods. Then, we turn to the problem of handling large mesh deformations of the boundary of a domain during a simulation and finally, we address the tedious problem of interface tracking in unsteady (flow) simulations. Examples of mesh adaptation are provided to show that the same set of local mesh modification techniques can be used to deal successfully with different types of applications.

23.1 Rigid body motion

Many phenomena in physical simulations can be represented as unsteady problems for which time evolution is the key feature. In some of these problems, the evolution does not concern the shape (this term stands here for both the geometry and the topology) of the different parts of the domain that remains unchanged during the simulation, but affects their relative position and motion. Classical examples arise in combustion simulations around multibody aircraft engines or pistons (Figure 23.1).

This area of study has received much attention in finite element simulations and in computer graphics. In these two fields, the common approximation of the computational domain is produced using tetrahedral meshes, or triangular meshes for surface boundaries. Classically, the motion of bodies assumed to remain rigid can be described by a combination of translations and rotations, from a reference (initial) position. Typically, a *reference frame* is chosen connected to and evolving with the body. Thus, the position of a rigid body can be described using two components, each represented by a vector: an orientation and a linear position. Computing mesh motion (i.e., the displacement of each mesh vertex) from object motion can be formulated as a discrete problem. It consists of solving a suitable elliptic partial differential equation to obtain the displacement vector u at all mesh vertices. Obviously, the main concern during such simulation is maintaining the overall mesh quality.

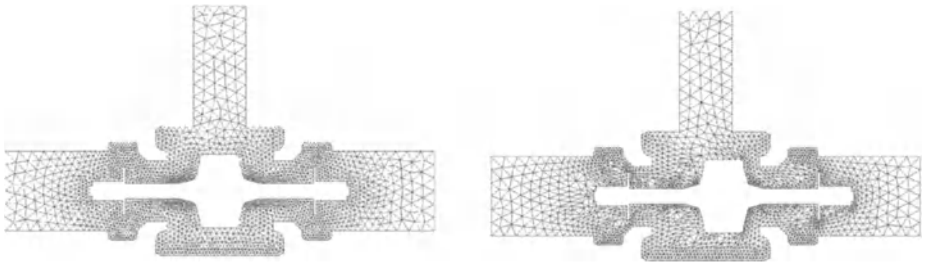


Figure 23.1: *Valve motion: initial position of the piston (left) and translatory moved piston with deformed mesh (reprinted from Int. J. Fluid Power).*

Mesh movement

We suppose that the boundary of the computational domain remains invariant in shape and that only the relative positions of the different parts may change in time. Many methods for defining mesh motion have been proposed in the past. Examples include the most popular spring analogy technique [Batina-1990] and [Zeng, Ethier 2005]. In this approach, vertex connections (edges) are replaced by linear springs and point motion is obtained as a response to the boundary loading. A review of the advantages and drawbacks of spring analogy is given by [Blom-2000]. To overcome the robustness problem, the addition of torsional springs to control and prevent invalidating any tetrahedral mesh element was suggested by [Farhat *et al.* 1998].

Several other approaches to providing a robust mesh motion solver involve the use of Laplacian smoothing [Löhner, Yang 1996], of a least-squares finite element formulation of the Navier-Stokes equations [Masud, Hughes 1997], or of a biharmonic equation [Helenbrook-2003]. In the following section, we will introduce a linear elasticity solver to help solve this motion problem.

Conceptually, the numerical solution is strongly influenced by the domain evolution in time to the point that the boundary geometry may be part of the solution. Thus, we must deal with boundary motion in a different way from internal vertex motion. In the applications we have in mind (pistons, valve motion, etc.), boundary motion is prescribed, for instance using a single displacement vector. The unknown of the problem is then the displacement field prescribed at internal mesh vertices. Internal mesh motion must cope with the changes in the domain shape as well as preserving the mesh quality and validity as it has an influence on the solution. It seems obvious to try reducing as much as possible the user intervention at this stage.

Spring analogy. As pointed out, in this approach, the mesh is considered a network of springs connecting vertices through edges. A force is induced in the spring assigned to an edge when a rigid body motion is applied. The strain energy in the spring system is given by the formula:

$$S = \sum_{ij \in E} K_{ij} \|u_i - u_j\|^2 , \tag{23.1}$$

where E denotes the set of mesh edges, K_{ij} represents the spring stiffness (that can be simply assigned to be the length of edge ij) and u_i is the displacement vector at node i [Batina-1990]. The minimum of the energy corresponds to a mesh where the strain, caused by the boundary displacement, is equidistributed. The model can be solved using a preconditioned conjugate gradient algorithm.

Mesh validity and quality. Preserving a valid mesh is an essential requisite for numerical simulations and this check is a critical step in mesh motion. Mesh validity, that corresponds to a conforming mesh without invalid elements, can be checked and enforced using topological and geometrical tests. Note that topological checks can be performed on the current mesh and do not involve mesh coordinates. Geometrical checks will check whether invalid elements may be produced by the displacement field, thus dealing with positivity of volumes and with orientation and convexity requirements (see Chapter 18 for more details).

Another concern during mesh motion, is that point locations are modified and will affect the element quality. Therefore, additional geometric tests must be used to preserve mesh quality during the motion. Once convexity and positivity of the mesh have been checked, quality checks are rather inexpensive to carry out.

The linear elasticity problem

Finding an admissible displacement field can be obtained by solving a simple Laplace (or Poisson) equation for each component or solving a modified version of the form:

$$\nabla \cdot (\tau \nabla u_i) = 0 \quad \forall i = 1, 2, 3 \tag{23.2}$$

where function τ represents a variable diffusivity, $\tau(K) = 1 + \dfrac{V_{max} - V_{min}}{V(K)}$, $V(K)$ being the volume of mesh element K and V_{min} (resp. V_{max}) denotes the minimal (resp. maximal) element volume. Solving Equation (23.2) provides an r-refinement scheme allowing for greater deformation in regions where elements have larger volume, [Baker-2001].

However, a drawback of this type of equation is that it lacks any dependence between the components of the displacement. For instance, if a domain region is moving along a fixed direction, all mesh vertices will be moving along this same direction. Hence, if the moving region is traversing a domain, say from the left to the right side, it tends to compress the mesh elements in the rightmost region while elements in the leftmost part become highly strechted. Therefore, a more versatile mesh motion scheme involves solving an elasticity problem.

It is well known that the equations of linear elasticity govern small displacements of an object subjected to internal body forces and surface traction. Here, we consider an elastic non-homogenous material filling a domain $\Omega \in \mathbb{R}^d$. The problem consists in seeking some apparent properties of this material. Here, the unknown u represents a field of elementary small displacements modelled by the linear elasticity equation. We will now present the governing equations and the weak formulation for finite element approximations of this problem.

Governing equations. Let consider Ω an open bounded domain with a Lipschitz boundary denoted as $\Gamma = \Gamma_0 \cup \Gamma_1$. The isotropic linear elasticity phenomenon is commonly described by the following system, where ρ represents the material density, σ_{ij} is the stress tensor and $f \in L^2(0,T,L^2(\Omega))$ is called a body force measured by unit volume of Ω:

$$\rho(x,t)\frac{\partial^2 u}{\partial^2 t}(x,t) - \nabla.\sigma(u(x,t)) = f(x,t)\,, \qquad \forall x \in \Omega\,,\, \forall t \in [0,T]\,. \tag{23.3}$$

For an homogenous isotropic media, the stress tensor σ is defined by means of Hooke's law as:

$$\sigma(u) = \lambda\, tr\varepsilon(u)\, I_d + 2\mu\,\varepsilon(u), \tag{23.4}$$

where $\varepsilon(u) = \frac{1}{2}(\nabla u + \nabla u^t)$ is the Green-Lagrange strain tensor, and λ and μ are the Lamé coefficients. This system is supplemented with the following set of boundary conditions:

$$\left\{\begin{array}{rcll} u(x,t) & = & u_0(x,t) & \forall x \in \Gamma_0\,, \forall t \in [0,T] \\ \sigma(u(x,t)).n(x) & = & g(x,t) & \forall x \in \Gamma_1\,, \forall t \in [0,T] \end{array}\right. \tag{23.5}$$

where $u_0 \in L^2(0,T,H^{1/2}(\Gamma_0))$ and $g \in H^1(0,T,H^{-1/2}(\Gamma_1))$ corresponds to a surface traction measured by unit area of Γ. In addition, a set of initial conditions is prescribed:

$$u(x,t=0) = u^0(x) \qquad \text{and} \qquad \frac{\partial u}{\partial t}(x,t=0) = w^0(x) \qquad \forall x \in \Omega\,, \tag{23.6}$$

where $u^0 \in H^1(\Omega)$ and $w^0 \in L^2(\Omega)$. As such, given Hooke's law, the problem is well-posed.

If we consider a steady-state elasticity model, the model can be reduced to the simple equation:

$$\nabla.\sigma(u(x)) = f(x)\,, \quad \forall x \in \Omega\,. \tag{23.7}$$

Finite element discretization. Equation (23.7) is then discretized on a simplicial mesh T_h using a Galerkin formulation. The related weak formulation of this problem can be written as: *find $u \in V + u_0$, such that for all $v \in V$*

$$\int_\Omega \sigma(u) : \varepsilon(v)\, dx = \int_\Omega f.v\, dx + \int_{\Gamma_1} g.v\, d\gamma\,, \tag{23.8}$$

where the space $V = \{v \in H^1(\Omega)\,, v = 0 \text{ on } \Gamma_0\}$.

Then, the corresponding discrete problem becomes: *Find $u_h \in V_h + u_0$, such that for all $v_h \in V_h$*

$$\int_\Omega \sigma(u_h) : \varepsilon(v_h)\, dx = \int_\Omega f.v_h\, dx + \int_{\Gamma_1} g.v_h\, d\gamma\,, \tag{23.9}$$

where V_h represents a suitable approximation function space.

We refer the reader to Chapter 20 for all the details about the implementation of this approach.

Moving mesh technique

Once a solution has been computed for the elasticity problem, a discrete field of displacement is defined at all mesh vertices. The meshing stage corresponds to moving each vertex to its new location provided the mesh elements do not become invalid or degenerated. Practically, mesh vertices are moved step by step toward their objective position, with the assumption that the mesh remains valid at each step. This iterative r-method is completed when all mesh vertices have reached their final position or when no mesh vertex can move without invalidating the mesh elements. A simple dichotomy scheme can be used to carry out this mesh modification procedure (Figure 23.2).

Other approaches involving topological and geometrical mesh modification operations can be used (see Chapter 18) to avoid creating tangled or invalid elements. However, such operations do not preserve the mesh cardinality (same number of vertices and elements).

Moreover, to optimize the numerical scheme, the elasticity problem can be resolved in a part of the domain only. This will significantly reduce the size of the linear system (the number of degrees of freedom) and dramatically speed up the overall procedure. Figure 23.3 shows an example of a rotating body where the active computational domain is notably reduced to a small region surrounding the blade.

Figure 23.4 illustrates the application of rigid body motion to an application where the local mesh density must be preserved. When the moving part of the domain crosses an area of anisotropic elements, the local anisotropic metric is kept and possibly intersected with the adaptive metric (for instance provided by an error estimate). To this end, a metric intersection procedure is employed (see Chapter 10).

23.2 ALE methods

Obviously, a chapter devoted to r-method would not be complete without a section devoted to arbitrary-Lagrangian-Eulerian methods.

Arbitrary-Lagrangian-Eulerian (ALE) methods were introduced by Hirt *et al.* to optimize the accuracy, robustness and efficiency of both the Lagrangian and Eulerian schemes [Hirt *et al.* 1974]. Classically, motion can be described in Eulerian or Lagrangian coordinates in continuum mechanics. In the Eulerian schemes,

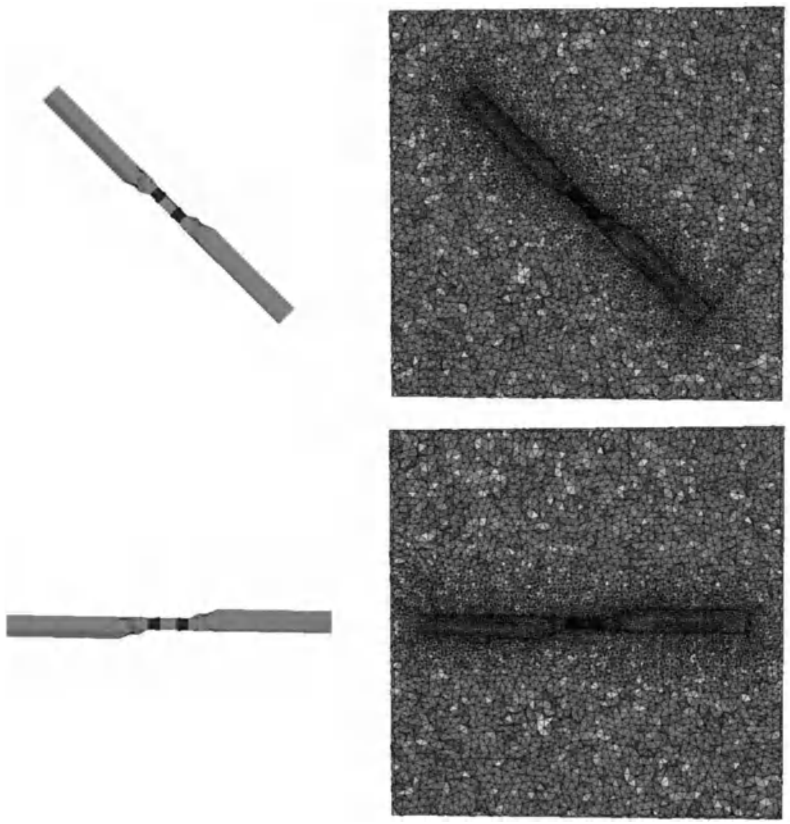


Figure 23.2: *Example of rigid body motion for a rotational three-dimensional domain (courtesy of C. Dobrzynski).*

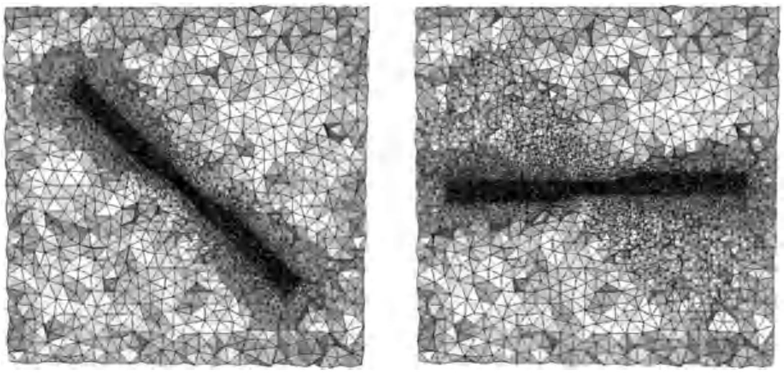


Figure 23.3: *Example of the restriction of the computational domain for a rotational three-dimensional domain (courtesy of C. Dobrzynski).*

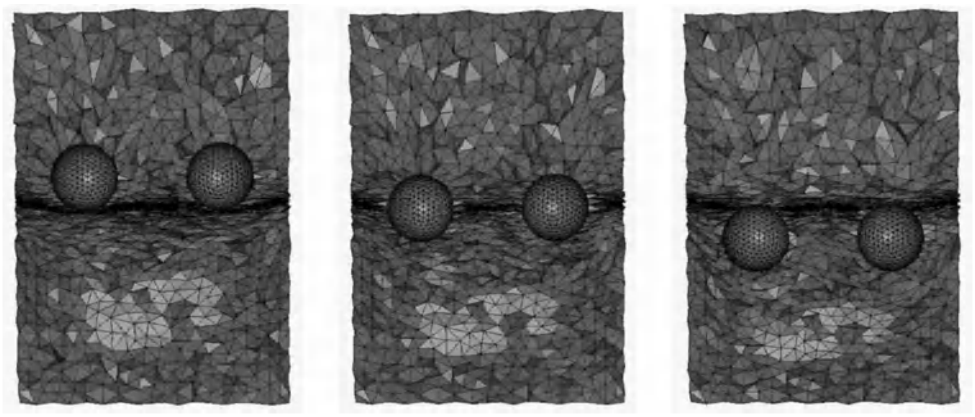


Figure 23.4: *Application of rigid body motion with density constraints (courtesy of C. Dobrzynski).*

largely used in computational fluid dynamics, the computational discretization of the domain (i.e., the mesh) remains fixed during the whole simulation; the medium moves with respect to the mesh. Hence, there is no real problem with large deformations in the motion, although the interface and boundaries must be accurately described to resolve flow details. On the other hand, in Lagrangian schemes, each mesh vertex is associated with a particle of material and follows the displacement of this particle during the time. As a consequence, large deformations of the mesh structure may be encountered and, in some peculiar cases, tangled or inverted elements may occur. With the ALE methods, the mesh vertices can move with the medium in the Lagrangian manner or can remain fixed in Eulerian manner or be moved freely in the domain. This gives the flexibility of handling much larger deformations than with any other approach while preserving the interfaces between materials or domains. Classically, the process consists of the following steps:

1. a classical Lagrangian step, in which the mesh moves along with the material particles,

2. a *rezoning* step, in which the mesh is modified so as to preserve good element quality, and

3. a *projection* step, in which the solution is transferred from the old mesh to the current mesh.

During the rezoning stage, care must be taken to avoid over-smoothing of the mesh leading to a loss of information (a dilution of the numerical solution). Moreover, due to the existence of invalid elements in the mesh, the regularization method must be robust enough to handle these situations and result in a valid mesh with convex untangled elements. Popular methods to regularize a mesh include Laplacian smoothings based on geometric criteria or methods based on a functional minimization (like the Winslow functional), usually more expensive and suffering from the existence of local minima and energy barriers [Winslow-1967].

Figure 23.5 shows an example of the computation of a cross-flow and rotational oscillations of a rectangular profile. The flow has been modelled using the incompressible Navier-Stokes equations and the numerical scheme uses an ALE formulation for fluid-structure interaction. The circle makes it possible to define two regions: the outer region where the mesh is Eulerian (no distortion) and the inner region where the mesh is prescribed to move rigidly attached to the rectangle.

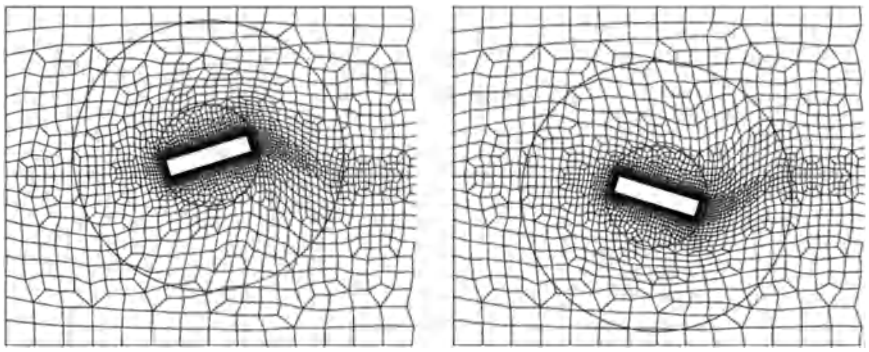


Figure 23.5: *Detail of a finite element mesh around a domain rotating in time in a fluid flow simulation (reprinted from [Donea* et al. *2004]).*

The ALE technique requires the formulation of a procedure to update the mesh vertices velocities at each time step of a computation. Consequently, the remeshing strategy strongly impacts on the efficiency of the ALE method and may be time consuming and error prone. Two choices are possible: mesh regularization (r-adaptation) or mesh adaptation. Regularization aims at preserving the regularity of the mesh as long as possible as well as to avoid the creation of tangled elements. On the other hand, if ALE is used as a mesh adaptation technique, it requires a suitable error estimate to concentrate the vertices in the regions of steep solution gradients.

Mesh regularization is a geometric operation and its objectives are purely of a geometric nature. It consists of keeping the mesh regular (as far as possible) during the computation, avoiding distorted or tangled elements. As mentioned in Chapter 18, it requires the update of the vertex coordinates at each time step through a displacement field (see the previous section) or from the current mesh velocities. Once the boundary motion has been computed, interpolation techniques are used to determine the mesh rezoning in the interior domain. As in mesh optimization, the rezoning of the mesh vertices consists of solving a Laplace (Poisson) equation for each component of the node velocity or position so that mesh edges coincide with lines of equipotential. It is close to what was described in Chapter 4 as *elliptic* mesh generation.

Laplacian smoothing

Discrete Laplacian operators have a long history in the context of mesh smoothing, although they were originally introduced for curve and surface denoising purposes [Field-1988]. Indeed, they can be seen as filters on the highest frequencies that are able to remove the noise.

The Laplacian smoothing, also called barycentric regularization (see Chapter 18) is easy to implement and its complexity is linear in the number of mesh edges. It consists of relaxing the position of a mesh vertex P_i in the direction ΔP_i of the weighted barycenter of its neighbors (the set $B(P_i)$, i.e, the ball of P_i):

$$\Delta P_i = \sum_{j \in B(P_i)} \omega_{ij}(P_j - P_i)\,, \tag{23.10}$$

where the ω_{ij} are positive weights and such that $\sum_{j \in B(P_i)} \omega_{ij} = 1$. All displacements being computed, the new positions of the mesh vertices are updated by adding the corresponding displacement vector to each vertex position:

$$P_i' = P_i + \alpha \Delta P_i$$

where $0 < \alpha < 1$ is a relaxation parameter, locally or globally specified. Obviously, if the ω_{ij} are all equal, each mesh vertex is moved towards the isobarycenter of its neighbors.

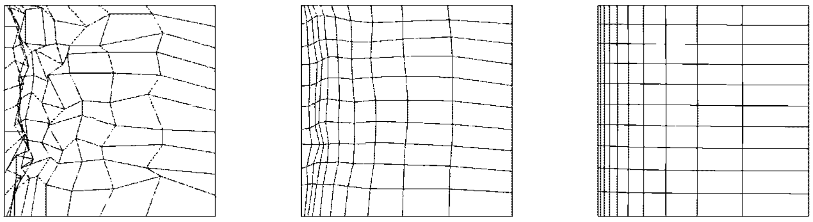


Figure 23.6: *Weighted Laplacian smoothing: initial tangled mesh, meshes after 3 and 300 iterations with a parameter* $\alpha = 0.5$ *(reprinted from [DeBuhan* et al. *2008]).*

As pointed out, too many iterations of Laplacian smoothing may lead to undesirable artifacts such as volume shrinkage or a loss of the mesh structure. To overcome this problem, Taubin [Taubin-1995], [Taubin *et al.* 1996] proposed combining two consecutive steps of Laplacian smoothing:

$$P_i' = P_i + \lambda \Delta P_i \quad \text{and} \quad P_i'' = P_i' - \mu \Delta P_i'\,.$$

A first Laplacian step is carried on with a positive scale factor λ on all mesh vertices and then, a second Laplacian step is applied to all mesh vertices, but with a negative scale factor $-\mu$ slightly greater in magnitude $0 < \lambda < \mu$. The Laplacian smoothing effect is obtained by repeating this alternate combination of positive and negative scale factors a number of times. Indeed, this method corresponds to a low pass filter, the scaling factors λ and μ determining the pass-band and stop-band limits [Taubin *et al.* 1996].

Note that the same procedure can be applied on surface meshes to remove undesirable artifacts related to implicit surface reconstruction methods (see Chapter 16).

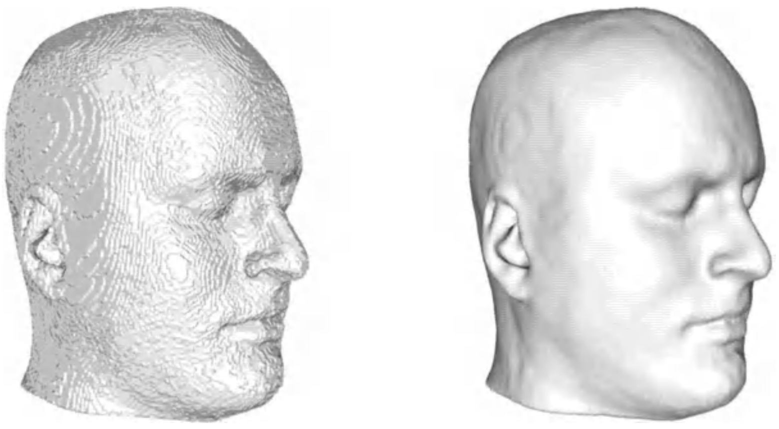


Figure 23.7: *Effect of bi-Laplacian smoothing on surface meshes: initial mesh and mesh after 100 iterations with a parameter* $\lambda = 0.33$, $\mu = 0.34$.

Area and Winslow functionals

There are obviously numerous parameters for measuring mesh quality (see Chapter 18). As far as numerical simulations are concerned, the orthogonality of the mesh lines and the mesh density in the region of large solution gradients are among the main criteria to be taken into account. Variational methods aim at optimizing the quality of a mesh using a mesh-based functional. Many functionals have been proposed that enjoy different properties such as mesh line orthogonality, equally sized elements, untangled elements, etc. (see [Tinoco *et al* 2001] for a description of area functionals).

For instance, the integral form of an area functional on a quadrilateral mesh is given by [Castillo-1991]:

$$\mathcal{J}_A = \frac{1}{2}\int_{[0,1]^2} |\mathbf{J}|^2 \, d\xi \, d\eta \,, \tag{23.11}$$

where $\mathbf{J} = [g_1 \; g_2]$ represents the Jacobian matrix based on the two covariants vectors g_1 and g_2 at the node considered (i.e., the columns of this matrix are the two vectors). In order to adapt a quadrilateral mesh, the author suggests using the following variation of the area functional:

$$F_A(x,y) = \sum_{k=1}^{n} \left(\sum_{i=1}^{4} s(k_i)|\mathbf{J}(k_i)|^2 \psi(k_i) \right) , \tag{23.12}$$

where $\mathbf{J}(k_i)$ is the Jacobian matrix at vertex k and for quadrilateral i, $|\mathbf{J}(k_i)|$ is the determinant and $\psi(k_i) > 0$ is the value of an adaptive function at the center of the cell i around vertex k. Indeed, the Jacobian is a measure of the area. The optimization of this functional will result in the equidistribution of the product of the cell area and the adaptive function. The critical feature of the functional is a mesh for which the product is the same for every element.

In the same spirit, the two-dimensional Winslow functional on quadrilaterals can be written as:

$$F(x,y) = \sum_{k=1}^{n} \left(\sum_{i=1}^{4} \frac{\|\mathbf{J}(k_i)\|^2}{|\mathbf{J}(k_i)|} \right) , \tag{23.13}$$

where the numerator is the Froebenius norm of the Jacobian matrix. Hence, the minimization of this functional is equivalent to the minimization of the condition number of the Jacobian matrix. A detailed description of the above analysis can also be found in [Knupp-2007].

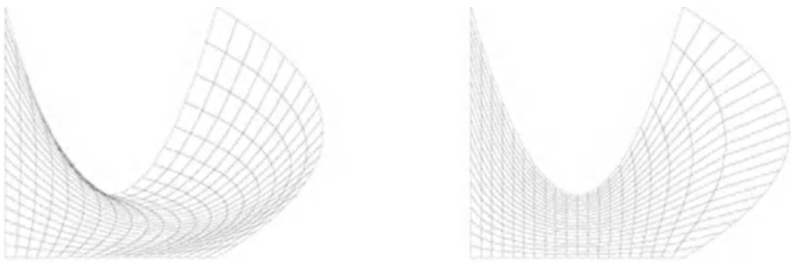


Figure 23.8: *Initial tangled mesh by transfinite interpolation (left) and final regularized mesh obtained using Winslow functional (right).*

23.3 Mesh deformation

In many applications, meshes undergo large or severe deformations during unsteady simulations. As a discretization tool, the finite element method has been largely employed to solve manufacturing problems and notably metal forming problems such as forging, extrusion, rolling, etc. In all these application fields, the spatial discretization, i.e., the mesh, evolves to represent and follow the material flow. When the distortion becomes too severe or when the topology of the domain changes, such a process requires the mesh to be enriched or coarsened to account for these modifications and to remain valid. Simultaneously, state variables must be transferred from the old mesh to the current mesh.

In contrast to the previous approaches, here the geometry of the domain is severely affected by the process and a geometric model is no longer available during the simulation. Figure 23.9 shows an example of an orthogonal cutting by chip formation. The material, similar to aluminium, is supposed isotropic elastoplastic, isothermal with non-linear isotropic hardening and isotropic ductile damage.

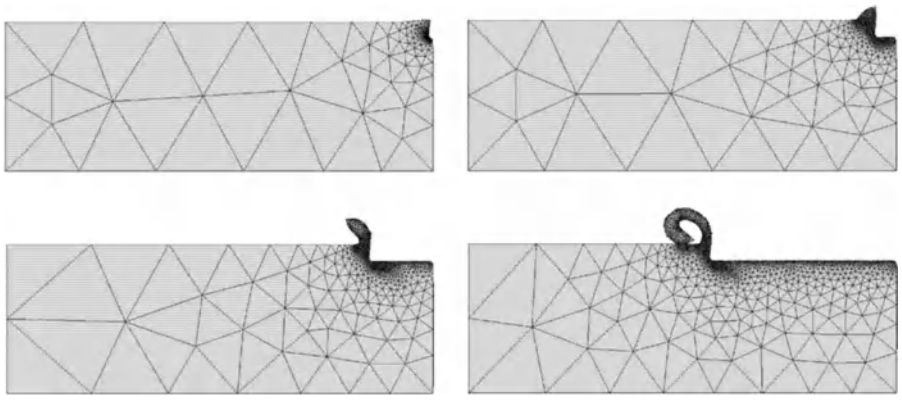


Figure 23.9: *Two dimensional example of orthogonal cutting by chip formation (reprinted from [Borouchaki* et al. *2002]).*

Many procedures have been devised to update the mesh, usually based on automatic mesh adaptation techniques including the following sequence:

1. update boundary discretization;
2. create a new mesh (or modify the old mesh);
3. project the state variables from the old mesh to the new mesh.

The new mesh can be obtained using global or local mesh generation or mesh modification procedures in a similar manner mesh adaptation is performed (see Chapter 21). Similarly, the amount of modifications can be decreased by employing local mesh modifications.

Geometry update

Obviously, the difficult part when considering the remeshing of a mechanical structure subjected to large plastic deformations, possibly with damage, is to follow the geometrical and topological changes of the domain shape. If remeshing is applied after each deformation increment (in a discrete time scheme), then the following steps are required:

- definition of the new geometry (after deformation),
- discretization of the boundary based on a geometric error estimation,

- internal mesh discretization based on a physical error estimate,
- adaptation of the current mesh to the deformed geometry.

The geometry evolution of the domain is concerned by different types of deformations leading to changes of the geometry and/or of the topology. Hence, all these deformations are related to topological and geometrical mesh modifications.

Local mesh modifications

Mesh modifications typically include the following operations:

- splitting of mesh edges,
- collapsing of mesh edge endpoints,
- swapping of mesh edges or faces, and
- vertex relocation.

In the case of simplicial (triangular or tetrahedral) elements, edge splitting or edge swapping are straightforward operations to implement. It is sufficient to define patterns to account for the introduction of a new vertex and to take into account node reconnections (in this case, a combinatorial problem). Edge collapsing is precluded if it results in inverting elements. If the edge to be collapsed is on the boundary, additional geometric constraints may be applied to control the accuracy of the geometric approximation.

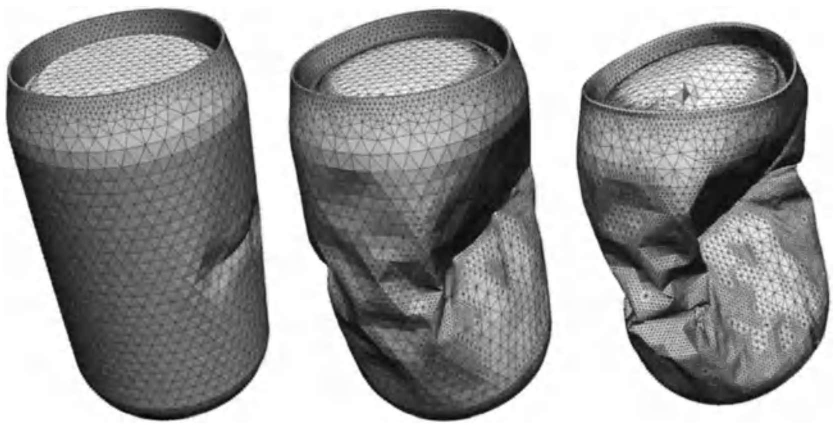


Figure 23.10: *Simulation of the crushing of an empty can using a template-based mesh adaptation (reprinted from [Giraud-Moreau* et al. *2005]).*

In some peculiar cases, it proved useful to use patterns (templates) to refine mesh elements when mesh distortion becomes too important. In such cases, refinement schemes such as those suggested by [Rivara-1997] are employed to ensure mesh conformity and validity. Figure 23.10 shows an example of an adaptive template-based refinement for simulating the crushing of an empty can.

Many of the known mesh coarsening procedures are usually restricted to the reversal of previous refinement procedures. However, more general procedures are required to handle the large variety of configurations that may arise. To this end, a general purpose mesh collapsing procedure is carried out, based on edge collapsing (see Chapter 18 for a description of this operation).

The next example concerns a fluid-structure interaction during an injection process. It deals with injection of a polymer in a cavity containing an elastic part in its centre. It is assumed that there are two injection surfaces and that the initial distribution of polymer is not homogenous. As the fluid polymer fills the cavity, the solid part moves under the fluid constraints as shown in Figure 23.11.

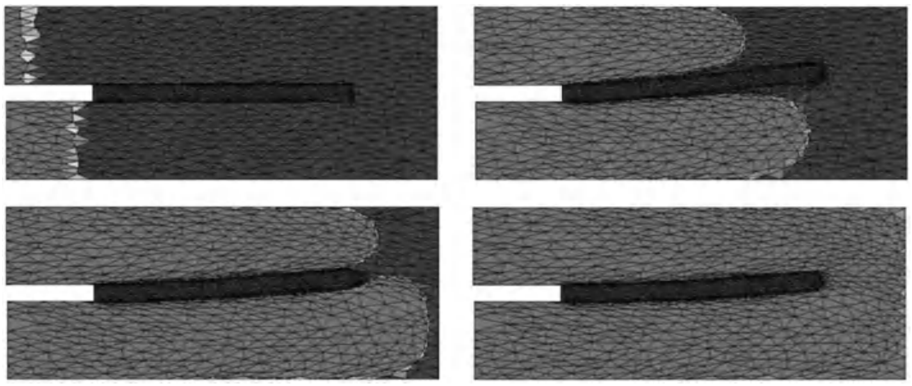


Figure 23.11: *Filling of a cavity containing a solid part inside (reprinted from [Coupez* et al. *2004]).*

A back extrusion problem is shown in Figure 23.12. The plastic behavior of the material is specified with a material flow stress function. The initial mesh of workpiece consists of 906 mesh vertices and 5,433 mesh elements. The simulation required 9 mesh enrichment steps. From the figure, it can be seen that the contact surface between the moving die and the workpiece is well captured by controlling the workpiece-die geometric interference.

23.4 Interface tracking

The numerical tracking of interfaces is an important part in simulations of many physical processes. In these applications, interfaces move according to physical laws. The key challenge is to keep updated with the arbitrary evolution of the interfaces, including changes of topology. For instance, in multiphase flows, an interface is a separation between two immiscible fluids. The interface is thus characterized by a discontinuity of physical quantities such as densities and viscosities.

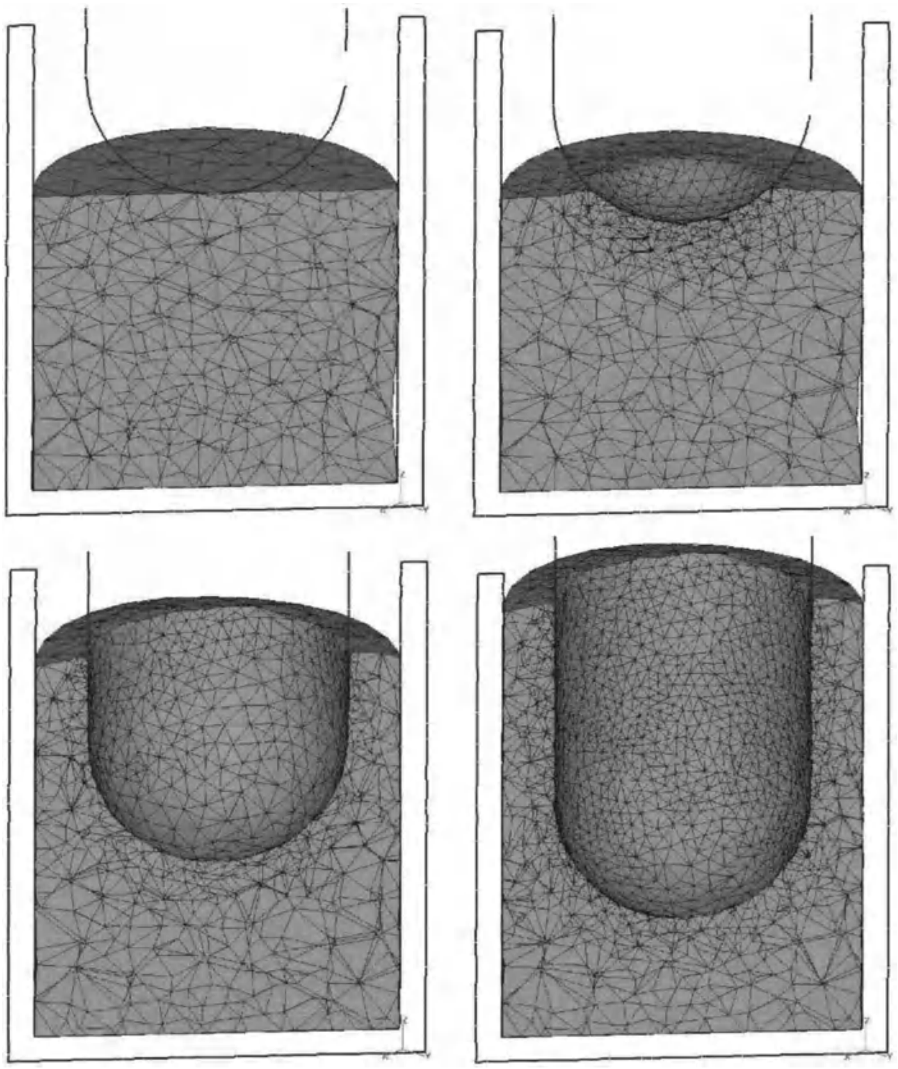


Figure 23.12: *Interior workpiece mesh produced by an automatic remeshing procedure (reprinted from [Wan* et al. *2003]).*

The law of motion may be directly connected to the shape of the interface as in flow motions by mean curvature. In the Lagrangian approach, markers or particles are used to define interfaces. A different challenge is offered by levelsets approaches which are based on a fixed Eulerian mesh. In this method, pioneered by Osher and Sethian [Osher, Sethian 1988], the interface is described by a zero levelset of a continuous function ϕ that is advected in time to follow the interface evolution.

General principle

Levelset methods are numerical techniques conceived and developed for tracking the evolution of interfaces for instance between two fluids. These interfaces can develop sharp corners, break apart, and merge together. As pointed out by [Sethian-1987], these techniques have a wide range of applications, including problems in computational fluid mechanics, combustion, manufacturing of computer chips, computer animation, image processing and the shape of soap bubbles. While most techniques attempted to follow moving boundaries by using a set of markers on the evolving front and then changing their positions to correspond to the moving front, levelset methods use a strong relationship between moving interfaces and equations from CFD.

By embedding the interfaces as a levelset of a higher dimensional function, the dimension of the advection problem is augmented by one. But, even if they are computationally more expensive than other methods, levelsets offer a convenient manner to handle topological changes.

Problem statement. Let consider the displacement of a surface at any point $x(t)$ according to a prescribed velocity field v. This problem can be related to the dynamics of an implicit surface. As mentioned before, a classical solution is offered by the Lagrangian schemes that consists of solving an ordinary differential equation with inital conditions to prescribe a mesh displacement field. From an Eulerian point of view, the dynamics revals two aspects:

i) if the velocity is independent of the surface, we have to resolve the PDE:

$$\frac{\partial \chi}{\partial t} + v \,.\, \nabla \chi = 0 \,, \tag{23.14}$$

where χ represents the characteristic function of one of the problem phases. This is usually solved by Volume of Fluids methods [Hirt-Nichols, 1981].

ii) if v depends on the surface geometry and if the domain $\Omega(t)$ evolves in time at a speed $v(t,x)$, for instance $v = (\alpha - \beta H)n$ (with n the normal to the surface and H the mean curvature), then $\phi(t, x(t)) = 0 \,,\ \forall x \in \partial(\Omega(t))$. Using the derivation rule will lead to the equation:

$$\frac{\partial \phi}{\partial t} + x'(t) \,.\, \nabla \phi = \frac{\partial \phi}{\partial t} + v\, n \,.\, \nabla \phi = 0 \,,$$

and as we can write:

$$n \,.\, \nabla\phi = \frac{\nabla\phi}{|\nabla\phi|} \,.\, \nabla\phi = \frac{|\nabla\phi|^2}{|\nabla\phi|} = |\nabla\phi| \,, \tag{23.15}$$

we finally obtain an Eikonal equation posed in $\mathbb{R}^n$:

$$\frac{\partial\phi}{\partial t} + v\,|\nabla\phi| = 0\,. \tag{23.16}$$

Levelset equation. In our problem, function ϕ is used both for representing the interface and to govern the evolution of the interface in time. From a practical point of view, the evolution of function ϕ is gouverned by the following convection equation:

$$\frac{\partial\phi}{\partial t} + v \,.\, \nabla\phi = 0\,, \tag{23.17}$$

the velocity field v can be prescribed in several ways. For instance, if $\phi(x) = 0$ represents the isosurface between two fluids, the speed of the interface is computed using Navier-Stokes equations (see below).

As n and $\nabla\phi$ are colinear vectors then $T \,.\, \nabla\phi = 0$ for all tangent vectors T. In two dimensions, $v = v_n\, n + v_t\, T$, the equation $\phi_t + (v_n\, n + v_t\, T) \,.\, \nabla\phi = 0$ gives:

$$\phi_t + v_n\, n \,.\, \nabla\phi = 0\,.$$

From Equation (23.15), we obtain the characteristic equation of levelsets:

$$\frac{\partial\phi}{\partial t} + v_n\,|\nabla\phi| = 0\,. \tag{23.18}$$

Equation (23.16) is a Hamilton-Jacobi equation posed in all $\mathbb{R}^n$. From the mathematical point of view, viscosity solutions have been proposed to this problem by [Crandall-Lions, 1984] to find a physically meaningfull solution to this problem. From the numerical point of view, this equation can be solved on unstructured meshes using high-order weighted schemes as suggested by [Abgrall, 2004]. One important point is to control the discretization of the interface and the mesh density in the vicinity of the interface.

Control of the geometric approximation

It is possible to control the construction of an accurate piecewise linear approximation Γ_h of the isocurve (isosurface) Γ describing an interface. In practice, in two dimensions, this control consists of defining a discrete anisotropic metric tensor field based on the intrinsic properties of the interface [Ducrot, Frey 2007]. To this end, the Hessian matrix of ϕ is decomposed as follows:

$$R \begin{pmatrix} 0 & 0 \\ 0 & \Delta\phi \end{pmatrix} R^t \quad \text{with} \quad R = \left(\nabla\phi \;\, \nabla\phi^{\perp}\right)$$

with

$$\frac{\kappa_2}{1+\kappa_2|\phi|} \leq \Delta\phi \leq \frac{\kappa_1}{1-\kappa_1|\phi|}$$

where κ is the local curvature of the interface, $\kappa_1 = \min_\Gamma \kappa$ and $\kappa_2 = \max_\Gamma \kappa$. The metric tensor is then defined as follows:

$$\mathcal{M} = R \begin{pmatrix} \dfrac{1}{\varepsilon^2} & 0 \\ 0 & \dfrac{|\Delta\phi|}{\varepsilon} \end{pmatrix} R^t \quad \text{with} \quad R = \left(\nabla\phi \ \nabla\phi^\perp\right) \tag{23.19}$$

From a practical point of view, the metric is defined at the vertices of a mesh of the domain. More precisely, it is prescribed at the vertices of the elements intersected by the isoline $\phi = 0$ and a matrix λI_2, $\lambda \in \mathbb{R}^+$ is prescribed at any other vertex of the mesh. This is sufficient to ensure that $\forall x \in \Gamma_h$:

$$d(x,\Gamma) < \varepsilon(1 - \frac{\kappa_2}{\kappa_1}) + \varepsilon^2\kappa_1) \, .$$

Mesh adaptation is based on iterative local modifications (see Chapters 16 and 18) in order to generate a unit mesh with respect to this Riemannian metric.

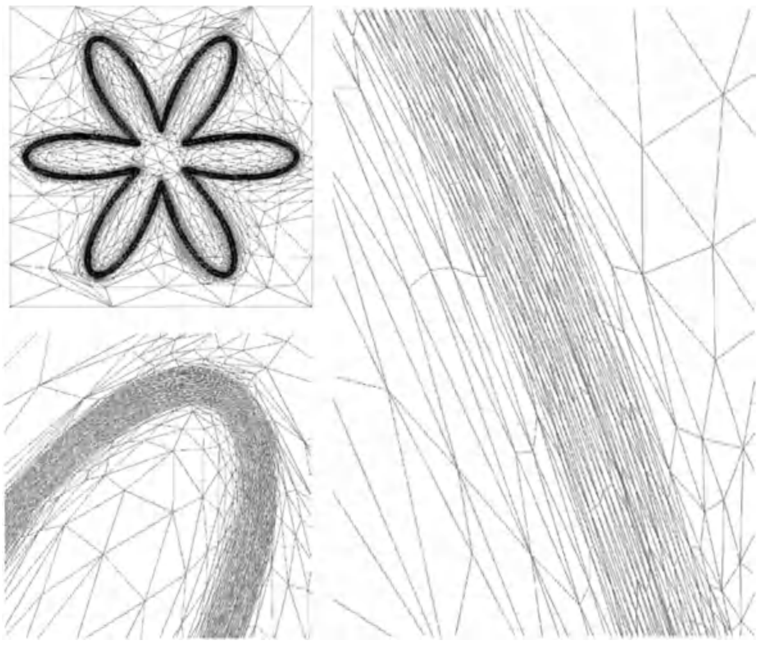


Figure 23.13: *Anisotropic control of a mesh in the vicinity of a time evolving interface (reprinted from [Ducrot, Frey 2007]).*

Figure 23.13 shows an example of an anisotropic mesh in the vicinity of an interface defined by the parametrized curve defined on $\Omega = [0,1] \times [0,1]$ by:

$$\begin{cases} x(t) = 0.5 + \cos(t)(0.27 + 0.18\cos(6t)) \\ y(t) = 0.5 + \sin(t)(0.27 + 0.18\cos(6t)) \end{cases} .$$

The resulting mesh has a mesh size of the order of 10^{-3} and contains 11,916 vertices. The L_∞ norm of the error between the curve and its discetization is $1.7\,10^{-4}$ for a theoretical gap smaller than $5\,10^{-4}$. A regular constant size mesh for this accuracy would contain about 10^6 vertices.

A similar control is possible in three dimensions as emphasized in Figure 23.14. Here, an anisotropic mesh has been generated near an interface corresponding to the following surface Σ defined on $\Omega = [-1;1]^3$ by $r = \cos(6\theta)\cos(3\phi)\cos(6\phi)$, where $\theta \in [0;2\pi]$ and $\phi \in [-\frac{\pi}{2};\frac{\pi}{2}]$.

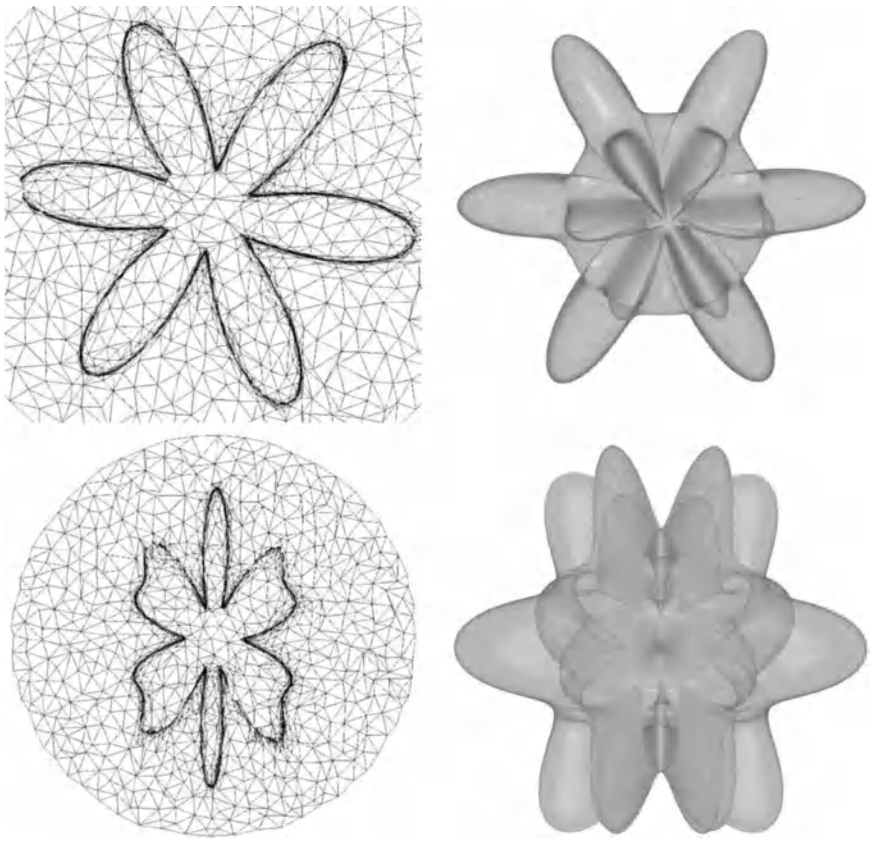


Figure 23.14: *Anisotropic control of interface approximation. Left: cut through tetrahedral meshes, right: corresponding levelset isosurfaces.*

A toy model for bifluid flows

As mentioned, the levelset method has been successfully applied to fluid dynamics simulations. As an example, we consider the following model based upon the levelset method for incompressible flows described by the Navier-Stokes equations:

$$\rho\left(\frac{\partial u}{\partial t}+u\cdot\nabla u\right)=-\nabla p+\nabla\cdot(\mu(\nabla u+\nabla u^t))+\rho g \qquad \forall(t,x)\in\mathbb{R}^+\times\Omega \tag{23.20}$$

$$\nabla\cdot u=0 \qquad \forall(t,x)\in\mathbb{R}^+\times\Omega \tag{23.21}$$

$$\frac{\partial\phi}{\partial t}+u\cdot\nabla\phi=0 \qquad \forall(t,x)\in\mathbb{R}^+\times\Omega \tag{23.22}$$

where Ω is the computational domain, the density ρ is assumed constant and μ is the variable viscosity. Dirichlet ($u=u_D$) or Neumann ($\sigma\cdot n=s_N$, where the stress tensor σ corresponds to $\mu(\nabla u+\nabla^t u)-pI$) boundary conditions and the following boundary conditions on the velocity u close the system:

$$\left\{\begin{array}{rll} u_1-u_2 & =0 & \text{on } \Gamma \\ (\sigma_1-\sigma_2)\cdot n & =\gamma\,\kappa\,n & \text{on } \Gamma \end{array}\right. \tag{23.23}$$

where the right-hand side term $\gamma\,\kappa\,n$ corresponds to a surface tension force involving the local curvature κ of the interface. The interface Γ is advected by the flow field $u(x,t)$ and, hence, changes in time.

The discretization of this problem shall account for the *inf-sup* condition and involves Lagrange $\mathbb{P}^1$ elements for the pressure p and augmented $\mathbb{P}^1$ elements for the velocity u. In this problem, the mass is assumed constant in each subdomain:

$$\rho=\rho_1\chi_{\Omega_1}+\rho_2(1-\chi_{\Omega_1})\,.$$

In practice, the characteristic function χ is replaced by a more regular function ϕ such as the distance function to interface Γ. The resulting equation can then be solved using a method of characteristics [Pironneau-1988].

Figure 23.15 shows an example of a circular bubble of viscous fluid A is immersed in a less viscous fluid B. In this sequence, the time stepping has been split between the computation of the velocity field and the subsequent advection of the interface represented by a levelset function ϕ. Initially, the velocity is zero $u=0$ at time $t=0$. The simulation has been run up to time $t=2$, where the bubble has moved more than a diameter and the mesh (the interface) has been largely deformed. The sequence shows the adapted meshes at iteration 1, 5, 10 and 20, for a time step $dt=0.1$. The adaptation was based on a geometric error estimate related to the local curvature of the interface as well as to the interpolation error on the velocity field.

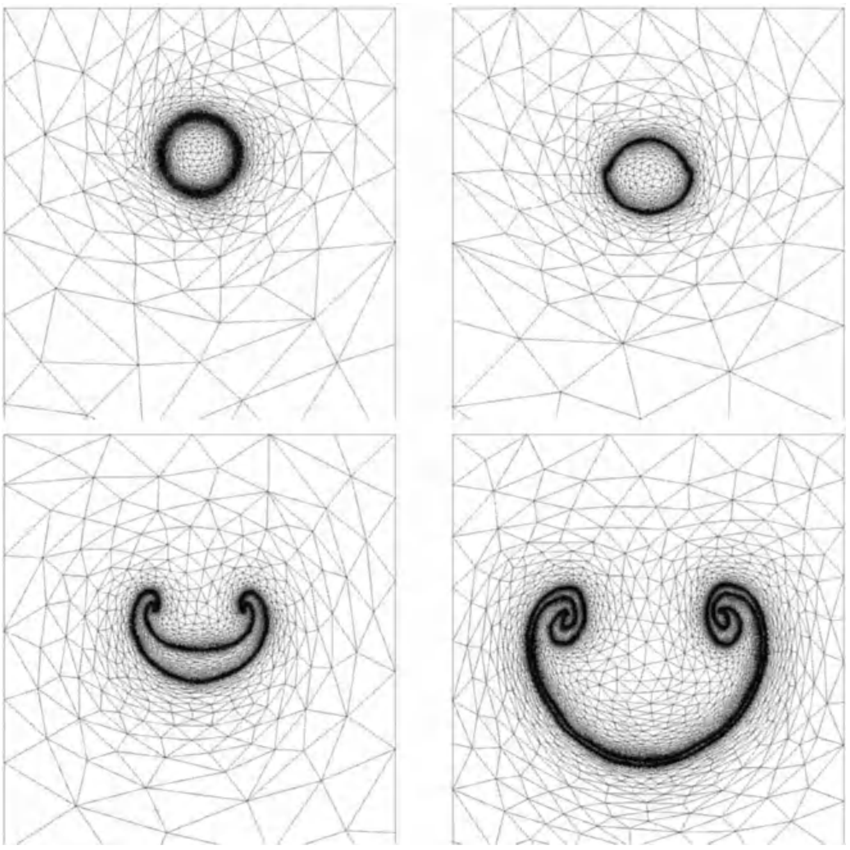


Figure 23.15: *A sequence of meshes adapted to the crvature of the interface in the Stokes problem.*

★
★ ★

The methods presented within this chapter do not claim to be exhaustive. However, most of the techniques employed to handle large changes or deformations in meshes can be attached to one of the previous categories. As it appears, mesh deformation is a hot topic, boosted by applications such as medical simulations (e.g. blood flows), chemical reaction simulations (where various species interact together), nuclear reactions (requiring a high level of accuracy), etc. Undoubtedly, this will be an active area of research for the next decade, possibly leading to the emergence of new meshing techniques.

Chapter 24

Parallel Computing and Meshing Issues

Parallel computing is a key issue for various categories of numerical problems. In practice, a numerical simulation may require a very fine mesh (i.e., one containing a large number of elements) and/or may be costly in terms of CPU time (when the computation is done in a serial way). Parallel computing[1] is an efficient solution for large size problems (i.e., with a large number of unknowns) that are impossible to carry out using classical facilities due to size requirements and/or CPU cost. In fact, parallelism consists of spreading (distributing) the computational effort and/or the memory requirements over the different computers available.

The notion of a parallel computing process can be conceived at various levels but, here, we will mainly focus on two of these levels. Obviously, the computational stage is concerned with parallelism. In such cases, a preliminary stage consists of partitioning the domain by means of sub-domains prior to dispatching these to different processors (each of them taking charge of one sub-domain). On the other hand, it could be of interest to see what degree of parallelism could be required at the mesh generation step itself.

At the solution step, a domain decomposition method requires a partition of the domain into several sub-domains. In these, the mesh that must be constructed must have certain properties. The solution method makes use of communication processes from one sub-domain to the others. At the meshing step, a parallel computational process consists of partitioning the domain into different sub-meshes in order to distribute the computation effort over several processors, each of them being responsible for computing the solution for one sub-domain. The global solution is then achieved, when all the sub-solutions have been completed, by merging all of these partial solutions.

Constructing a suitable domain partition, as well as constructing each of the corresponding sub-meshes, can be achieved in many different ways. These approaches can be broadly classified into two categories. Either *a posteriori* or *a priori* partitioning methods can be considered. *A posteriori* processing starts from a mesh of the entire domain while *a priori* processing uses the domain itself and

[1]For the sake of clarity, we are concerned here with a multi-processor architecture using a distributed memory system.

not a mesh of it. Note that *a posteriori* methods have received considerable attention and are widely used in practice whereas *a priori* methods are still a field of intensive research (particularly in order to automate the partitioning process).

When concerned with the mesh construction aspect, it is not easy to decide where precisely the parallel aspect should be included. In particular, a question could be: is it necessary to include parallel concepts in the mesh generation method or should the construction method remain serial and used in parallel? The following remark may throw some light on this. There are at present various mesh generation methods which are reliable, fast and able to complete several million elements within only a few minutes. Therefore, it might be thought that constructing[2] several million or a hundred million elements can be done in an indirect way by using a classical method (for constructing, say, one million elements) included in a parallel process. In other words, the global generation process is parallel but the construction method remains serial. This leads to distributing the computational effort over different processors and, this being done, sequentially considering one distinct task in one processor. Thus, following this approach, the parallel aspect acts at the global meshing level and not at the mesh generation method level.

Nevertheless, there have been a number of papers on parallel mesh generation methods. The aim here is to construct the mesh in parallel, usually using a classical meshing method which has been modified in order to incorporate some degree of parallelism. Despite the above remark, it should be noted that although this type of approach is possible, its interest seems more theoretical or academic.

★
★ ★

This chapter is subdivided into three parts. It successively discusses the different issues mentioned in the above introduction. Hence, at first, we recall the main domain partitioning methods. Several *a posteriori* methods are presented and then some *a priori* methods are described. We then turn to a parallel meshing process and, in particular, we discuss how to conceive a parallel loop of computation. To end, we give some indications about the possibility of mesh generation method parallelization.

24.1 Partition of a domain

Constructing a partition of a given domain consists of subdividing it into several disjoint sub-domains whose union forms a covering-up of the entire domain. In fact, we are not so much concerned with the domain or the above sub-domains but more with the meshes of these domains. First, we give some general indications about the question under investigation, then we discuss the different partitioning methods in greater detail.

[2]If this is strictly necessary. In fact, the concern is not necessarily to construct a "large" mesh but more probably to have such a mesh available.

Overview of partitioning methods

Partitioning or mesh splitting methods fall into two categories. *A posteriori* methods split a fine mesh (i.e., a large size mesh) into sub-meshes of a reduced size, and *a priori* methods construct the meshes related to sub-domains that are directly obtained (i.e., without having to construct a global fine mesh, but directly using the domain geometry or even a rather coarse mesh, which is easy and fast to complete and requires little memory).

***A posteriori* partitioning.** In such approaches, we are given a mesh of the entire domain and the partition method involves splitting this mesh into sub-meshes which define the partition. Several types of methods allow such a task. For details, the reader is referred to the *ad-hoc* literature (see [Simon-1991] or [Farhat, Lesoinne-1993], among many others) and, in what follows, we simply outline some of these methods.

It is obvious that the weak point in such a direct way of addressing the problem is the problem of memory requirement. In fact, the required memory size is about the sum of the resources needed to store the complete mesh plus the resources required to store at least one of the sub-meshes. Moreover, classical problems inherent to the partition process must be considered (as they will be present whatever the chosen method). Among these, we find:

- the issue of *regularity* or *smoothness.* This mainly concerns the shape of the sub-meshes and thus includes the shape of the sub-mesh interfaces. A certain degree of smoothness is indeed required to guarantee a relative numerical accuracy (convergence) for the partial solutions with regard to the global solution. Indeed, some methods, such those using the Schur complement, lead to the solution of local problems coupled with a problem at the interface level. These methods require well conditioned local problems in order to guarantee the proper global convergence. Connexity or not of the sub-domains is therefore an important factor for this local convergence,

- the question of *interface size.* The size of the interface between two sub-domains directly influences the amount of data that must be exchanged and thus on the degree of saturation (bottleneck) in the network used to carry these messages. This size also affects the number of potentially redundant operations,

- concern about the number of connections from sub-domain to sub-domain,

- concern of balancing between the sub-meshes. The size of these meshes must be distributed in such a way as to balance the effort of each processor at the solution step,

- concern about the interface sizes (the number of nodes on these interfaces), etc.

Also, we do not consider the memory resources necessary to construct the initial mesh and those needed to store it (not to mention the CPU time required in the i/o).

Different classes of partition method are encountered. Among these, some are based on graph partitions and some are purely geometric methods directly based on mesh partitions. All these methods apply to finite element type meshes since a vicinity graph can be constructed based on the connections between the elements in a given mesh.

Before presenting some of the most frequently used algorithms, it is useful to consider the type of entities (vertices (nodes) or elements) on which the decomposition is applied. This point is of importance since it is related to the numerical software or the solution method that will use this decomposition. This choice may also have some effect on the splitting algorithms. Nevertheless, we do not go into more detail about the positive features or the weaknesses of these two approaches.

- Element based decomposition

This is the most frequent case. The mesh is partitioned by distributing the elements among the sub-domains. In other words, one element is logically associated with one and only one sub-domain. An example of such a decomposition can be seen in Figure 24.1, left-hand side. This partition has an interface consisting of vertices (nodes), edges and faces.

- Node based decomposition

In this case, the mesh is partitioned by distributing its vertices among the sub-domains. In other words, one vertex is logically associated with one and only one sub-domain. An example of such a decomposition is shown in Figure 24.1, right-hand side. This partition has an interface consisting now of elements (nodes, faces and segments), which are shared by the sub-domains while maintaining the initial connections.

Before describing these methods in greater detail, it may be noticed that this approach is probably the worst way to go about mesh parallelism. Indeed, for a large-scale problem, the construction of an initial mesh of the entire domain[3] may even be impossible. Nevertheless, this type of method, when suitable, gives nice results and is probably the most pragmatic (in fact, simply the only possible) way to address the parallel problem. To conclude, note that the creation of the initial mesh does not take benefit from the assumed advantages of parallelism.

***A priori* partitioning.** To avoid this memory requirement problem, it is natural to examine the *a priori* approaches, when they exist and can actually be implemented. In this case, it is not necessary to construct a large mesh, thus limiting the

[3]In earlier experiences (say at the turn of the century), we were limited to 10 million tets constructed by a Delaunay-type method, mainly due to the memory resources available in a classical workstation. Right now, this limit is not clearly different and no significative change is expected leading to favor *a priori* methods where the limit applies, in principle, to each processor (e.g. one at a time).

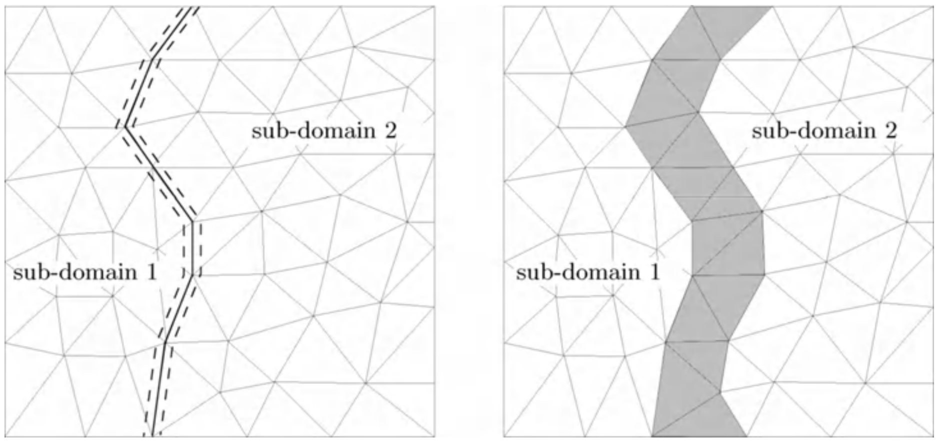


Figure 24.1: *Element based decomposition (EOD, element oriented decomposition), left-hand side and node based decomposition (VOD, vertex oriented decomposition), right-hand side.*

amount of memory available and, moreover, the parallel aspect directly appears in the construction, thus avoiding the main weakness of the previous methods.

Thus, we first create a partition of the domain. This step may start from a reasonable mesh (i.e., a relatively coarse mesh) of the entire domain. Several sub-domains are then identified and meshed, each on one processor. As such, parallelism plays a part from the mesh generation step. Another strategy consists of extracting the various sub-domains (their boundaries) starting from a mesh of the boundary (the surface) of the entire domain. The same method then applies.

Actually, the balancing between the sub-domains together with the smoothness of the interfaces may be considered by using the information now available (the coarse mesh in the first case or the surface mesh in the other approach).

In the first strategy, there are two ways of obtaining a coarse mesh: either an empty mesh (i.e., without any internal vertices) as defined in a Delaunay-based method (Chapter 7) or a mesh with a limited number of internal vertices constructed by one or other of the possible methods.

In the second strategy, the meshes are constructed using the meshes of the surfaces that constitute the interfaces between the sub-domains extracted from the given boundary mesh.

Interface regularity. Whatever the approach, interfaces are created and some degree of regularity or smoothness is expected. We saw that smooth interfaces avoid artefacts or indesirable behaviors in the further processing. Using local optimization tools as seen in Chapters 18 and 19. In the case of a purely internal interface, its geometry is not a constraint and its entities (vertices, edges, faces) can be freely modified (for instance, using swaps, point relocations, etc.). Conversely, a part of an interface member of a boundary must consider geometric constraints when applying any type of modification tool.

A posteriori partitioning

In this section, we describe some *a posteriori* partitioning methods. Specifically, there are four types of methods and some approaches based on the combination, to some degree, of two or more of these methods.

Greedy method. This approach makes use of a graph partition algorithm and is both very simple and quite intuitive. Its principle is similar to those of the node renumbering methods developed to limit the bandwidth of a finite element matrix (such as the Cuthill-McKee [Cuthill, McKee-1969] method, for instance, Chapter 17).

Given a neighborhood graph G, a starting entity $n1$, a number of expected sub-domains *ndsd* together with a targeted distribution (the number of expected entities in each sub-domain), the algorithm takes the following form:

```
set the parent entity to $n1$,
WHILE the number of sub-domain is less than $ndsd$, DO
   (A) initialize the front line to be one element, the parent entity
   FOR ALL entities in the current front line
      FOR ALL neighbors of the entity
         If the entity has not been previously marked, THEN
            put it into the current sub-domain
            add it to the list of the entities in the new front line
            under construction
            increment the number
         END IF
      END FOR
   END FOR
   If no entity is detected
      search for a new parent
      go to (A)
   IF the targeted number is that of the sub-domain
      IF the sub-domain is the last, END
      ELSE
         construct a new number 1
         return to WHILE
   END IF
   the current line is the thus-constructed line
   return to FOR ALL
END WHILE
```

This algorithm is very fast but its main drawback is the generation of non-connected sub-domains. Various variations exist, for instance, based on a recursive bisection, with adequate parents. This algorithm may also serve to construct a pre-partition used at a later time as the entry point of another partitioning method.

Partition by a spectral method. This graph partition method, described for example in [Simon-1991], [Gervasio *et al.* 1997] or in [Natarajan-1997] also referred to as the *Recursive Spectral Bisection* (RSB) is based on the properties of the eigenvalues and eigenvectors of a positive definite symmetric matrix.

For instance, let us consider the Laplace matrix associated with the graph dual of the mesh; then the matrix coefficients $a_{i,j}$ are defined as:

$$a_{i,j} = \begin{cases} -1 & \text{if} \quad (n_i, n_j) \quad \text{are connected} \\ deg(n_i) & \text{if} \quad i = j \\ 0 & \text{otherwise} \end{cases}$$

This matrix is useful when computing a separator between the nodes, as explained in [Donath, Hoffman-1972], [Donath-1973] or [Pothen *et al.* 1990]. We can prove that the eigenvector related to the second smaller eigenvalue of A (the so-called Fielder vector [Fielder-1975]) corresponds to minimizing a constraint about the number of connections.

If the vector components are partitioned in increasing order and if the graph is partitioned in this order, two sub-graphs with a close size are obtained in which the number of connections from one to the other is minimal. The corresponding algorithm may be written as follows:

```
$j = 1$
WHILE the number of sub-domains $j$ is less than the target
   construct the Laplace matrix of the dual graph
   compute the second smallest eigenvalue and the corresponding
   eigenvector (the Fielder vector)
   sort the graph according to the components of the Fielder vector
   distribute the entities between the two graphs
   $j = 2j$
END WHILE
```

This method is time-consuming. Indeed, it requires the solution of numerous eigenvalue problem for sparse matrices. Iterative methods are generally chosen as being better adapted to such problems (for instance, the Lanczos method [Cullum, Willoughby-1985], [Chatelin-1993]).

Nevertheless, this algorithm has two drawbacks related to memory requirements and the necessary CPU cost. A number of variations can be developed, for instance, based on a multi-level approach, which allows an improvement in costs.

Recursive partition using the inertia axis. The approach for partitioning is here related to the domain geometry. It is based on an analogy of the initial mesh with a mechanical system consisting of discrete points (i.e., the vertices) with which mass is associated. This mechanical device has various inertia components including a main component that corresponds to minimizing the rotation energies.

Intuitively, the main component gives the axis along which the set is the thickest. This idea, as applied in a partitioning problem, simply leads to trying to split the set of entities into two parts along this axis. Indeed, this is the place where the interface will be minimal. This algorithm is more efficient if the mesh is homogenous (in terms of element sizes) (and, *a contrario*, is less efficient when there is large variation in these sizes).

Computing the main inertia component involves finding the eigenvector associated with the largest eigenvalue of matrix $\mathcal{I} = {}^t\mathcal{A}\mathcal{A}$ where $\mathcal{A}$ has as its coefficients the $a_{i,j}$s[4] by:

$$a_{i,j} = x_j(i) \; - \; g_j \,,$$

with $x_j(i)$ the coordinate of index j of the node of index i and g_j the j-coordinate of the centroid G defined by:

$$g_j = \frac{1}{n} \sum_{1=1,n} x_j(i) \,,$$

n being the number of entities under consideration (the nodes, for instance). Then $\mathcal{I}$ is a $d \times d$ matrix where d is the spatial dimension. This matrix, in three dimensions, takes the form:

$$\mathcal{I} = \begin{vmatrix} \mathcal{I}_{x_1x_1} & \mathcal{I}_{x_1x_2} & \mathcal{I}_{x_1x_3} \\ \mathcal{I}_{x_2x_1} & \mathcal{I}_{x_2x_2} & \mathcal{I}_{x_2x_3} \\ \mathcal{I}_{x_3x_1} & \mathcal{I}_{x_3x_2} & \mathcal{I}_{x_3x_3} \end{vmatrix} \tag{24.1}$$

[4]To establish what the coefficients are, we turn to a minimization problem. If $X(i)$ stands for vector $P(i) - G$, where G is the centroid of the $P(i)$s and if V is an unknown unity vector, we consider the problem of minimizing the square of the distances between the $P(i)$s and G to a line segment aligned with V passing through G. This corresponds to the following problem:

$$\max_V \sum_i \langle X(i), V\rangle^2$$

under the constraint:

$$\langle V, V\rangle - 1 = 0 \,.$$

The solution corresponds to the case where the gradients of these two expressions are aligned. Then, we look for a coefficient λ such that:

$$\sum_i X(i) \, \langle X(i), V\rangle = \lambda \, V$$

In terms of the coordinates $x_j(i)$s of the $X(i)$s and v_js of V, this leads to:

$$\left(\sum_i (x_1(i)^2 - \lambda\right) \, v_1 + \sum_i x_1(i)x_2(i)\, v_2 + \sum_i x_1(i)x_3(i)\, v_3 = 0 \,,$$

$$\sum_i x_1(i)x_2(i)\, v_1 + \left(\sum_i x_2(i)^2 - \lambda\right) \, v_2 + \sum_i x_2(i)x_3(i)\, v_3 = 0 \,,$$

$$\sum_i x_1(i)x_3(i)\, v_1 + \sum_i x_2(i)x_3(i)\, v_2 + \left(\sum_i x_3(i)^2 - \lambda\right) \, v_3 = 0 \,,$$

which, in a matrix form, is:

$$\begin{pmatrix} \sum_i x_1(i)^2 - \lambda & \sum_i x_1(i)x_2(i) & \sum_i x_1(i)x_3(i) \\ \sum_i x_2(i)x_1(I) & \sum_i x_2(i)^2 - \lambda & \sum_i x_2(i)x_3(i) \\ \sum_i x_3(i)x_1(i) & \sum_i x_3(i)x_2(i) & \sum_i x_3(i)^2 - \lambda \end{pmatrix} \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} = 0 \,.$$

This system has a non-zero solution if its determinant is zero. Thus, λ is an eigenvalue of the previous matrix $\mathcal{I}$.

with:

$$\mathcal{I}_{x_j,x_k} = \sum_i (x_j(i) - g_j)(x_k(i) - g_k),$$

thus, in particular, we have as diagonal coefficients:

$$\mathcal{I}_{x_j,x_j} = \sum_i (x_j(i) - g_j)^2.$$

In three dimensions, the problem reduces to computing the largest eigenvalue of a 3×3 matrix and then computing the corresponding eigenvector. This task is easy for instance, by means of the power method. The partitioning algorithm then takes the form:

```
set $j = 1$ (initialization of the number of members in the partition)
WHILE $j$ is less than the target
   FOR $i = 1, j$ where $i$ denotes a sub-domain
      compute the centroid $G$ of sub-domain $i$
      compute the main inertia component of sub-domain $i$.
      (This involves computing $\mathcal{I}$ for the entities in $i$ and then
      picking the largest eigenvalue and eigenvector)
      project the entities in $i$ following this eigenvector
      sort the projected entities
      partition into two the sub-domain $i$ according to the
      sorted projected entities
   END FOR $i$
   $j = 2j$
END WHILE
```

This algorithm can be easily adapted to obtain an arbitrary number of sub-domains. To do so, it is sufficient to modify the way in which the partition into two sub-domains is made by partitioning not in a homogenous way but following a distribution which is proportional to the number of sub-domains expected when dealing with the current sub-domain, on one side or the other of the partition.

Remark about the inertia matrix. The above method makes use of the matrix defined in Relationship (24.1) which is not the inertia matrix widely used in mechanical engineering. The latter is written as follows:

$$\mathcal{I} = \begin{vmatrix} \mathcal{I}_{x_1x_1} & \mathcal{I}_{x_1x_2} & \mathcal{I}_{x_1x_3} \\ \mathcal{I}_{x_2x_1} & \mathcal{I}_{x_2x_2} & \mathcal{I}_{x_2x_3} \\ \mathcal{I}_{x_3x_1} & \mathcal{I}_{x_3x_2} & \mathcal{I}_{x_3x_3} \end{vmatrix} \tag{24.2}$$

but, now, with:

$$\mathcal{I}_{x_j,x_k} = -\sum_i (x_j(i) - g_j)(x_k(i) - g_k),$$

and, for the diagonal coefficients, expressions of the form:

$$\mathcal{I}_{x_j,x_j} = \sum_i \left((x_{j+1}(i) - g_{j+1})^2 + (x_{j+2}(i) - g_{j+2})^2\right),$$

with obvious notations about the indices $j+1$ and $j+2$, for a given j.

Clearly, in two dimensions, the eigen-elements of matrices (24.1) and (24.2) are the same. The same is true in three dimensions (or for higher dimensions). To be certain of this, just note that the sum of these two matrices (in three dimensions) is:

$$\sum_i \left(\sum_j (x_j(i) - g_j)^2 \right) \mathcal{I}_3 ,$$

in other words a diagonal matrix. A simple examination of the eigen elements of the two matrices (while using this relationship) allows for the proof.

K-means method. Initially developed in a different context (classification problems, [Celeux *et al.* 1989]), the aim is still the same, we try to balance in a certain number of classes the entities in a given set, here a mesh.

Let us consider S a cloud of points (the mesh points, in a vertex based algorithm or the element centroids, in an element based algorithm). An outline of the K-means method is as follows:

- choosing, using one method or another (random choice, inertia axis, octree type structure, etc.), k seeds (or kernels) where k is the number of classes to be constructed,
- (A) - associating the points in S with a seed based on a given criterion (for instance, a proximity criterion). This points-seeds relationship then defines k classes,
- computing, for every class, a new seed (for example, its centroid), then returning to (A).

The interesting feature in this method is, after some adequate assumptions, its convergence. Indeed, it builds a set of k stable classes. In addition, these classes satisfy (maximize) a criterion meaning that the classes and their seeds are in adequation and that the classes are well aggregated and well separated from one another.

Convergence results from observing that a series associated with the process is a decreasing series and thus the algorithm implies that a criterion decreases at each step until stability which, in turn, ensures that the solution is completed.

A few examples of partitioning output. We now give (Figures 24.2 to 24.5) some application examples resulting from the above methods. More precisely, Figure 24.2(i-iv) shows the partitioning of two dimensional domains as obtained by a greedy method with VOD or EOD choices.

Figure 24.3 depicts two partitions of a domain, in two dimensions, using *Recursive Spectral Bisection* methods, one vertex based (VOD), the other element based (EOD). We can see in this example that the (greedy) bisection method produces a fairly similar result.

Figure 24.4 (i-iv) shows, for the same case, some partitions into two, four and eight sub-domains as completed by an inertia axis method.

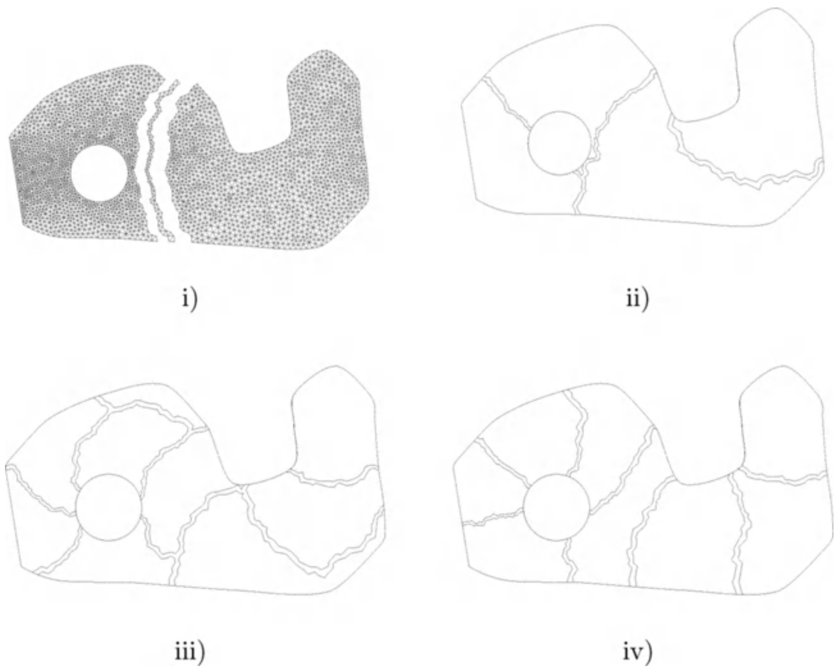


Figure 24.2: *Partition of a sub-domain in two dimensions using a VOD greedy method into, respectively, two i), four ii) and eight iii) sub-domains. Partition by a multigrid-greedy EOD method iv).*

The case of a three-dimensional domain is then depicted in Figure 24.5. The domain has been split into eight sub-domains using a VOD spectral method and an inertia axis EOD method.

A priori partitioning.

In what follows we discuss two particular *a priori* partitioning methods. Afterwards, some remarks about other methods will be given.

Method 1. In this approach the entry point is a relatively (or a really) coarse mesh of the entire domain. The technique may be classified as an *a priori* method in the sense that it is not necessary to mesh the domain with the fine mesh that will be the support of the envisaged computation in order to define the partition.

The coarse mesh, the basis of the partition, may be easily obtained using a Delaunay type method (Chapter 7), based on the discretization of the boundary of the entire domain. The meshing process is then stopped when the domain is covered, i.e., once the boundary points have been inserted and the boundary

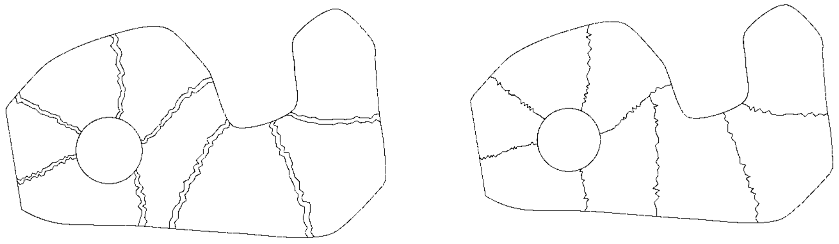


Figure 24.3: *Partition of a sub-domain into eight sub-domains, using a VOD RSB type method (left-hand side) and a EOD RSB method (right-hand side).*

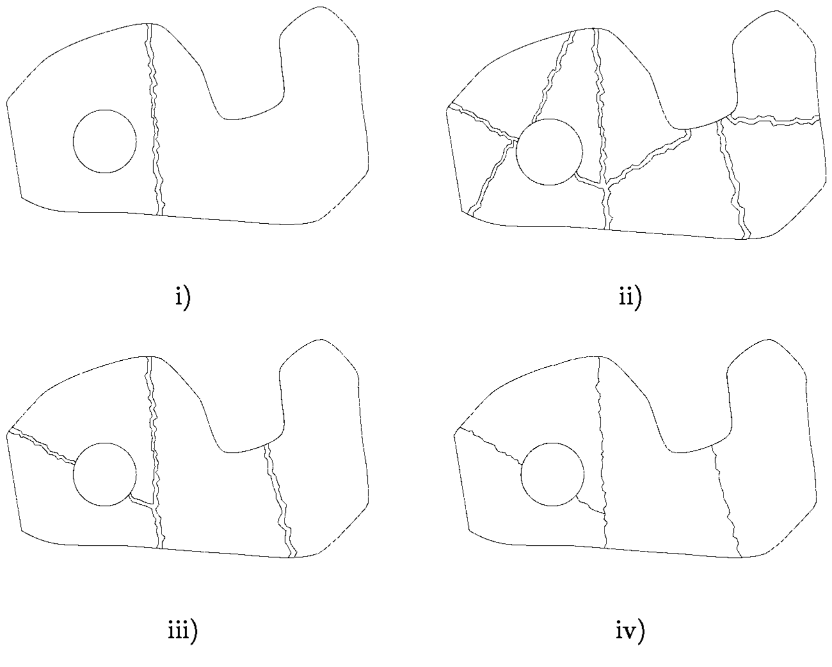


Figure 24.4: *Partition of a sub-domain by an* inertia axis *method (in the VOD case) into two i), four iii) and eight ii) sub-domains and in the EOD case in the case where four sub-domains are desired, iv).*

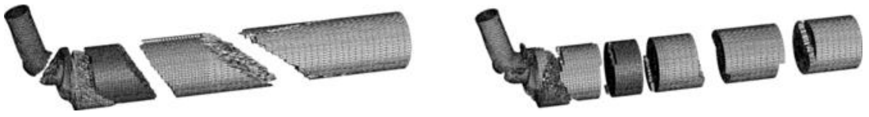


Figure 24.5: *Partition into eight sub-domains by a* Recursive Spectral Bisection *VOD type method (left-hand side) and by an* inertia axis *EOD method (right-hand side) (data courtesy of Renault).*

integrity step has been completed. In other words, no (or a few) internal vertices exist inside the domain.

We use a splitting algorithm based on this mesh, for instance, by a *greedy* type method (see above). This results in a partition of the domain into several sub-domains. However, it is tedious to balance this partition. In particular, two questions must be carefully addressed that concern the well-balancing aspect and, on the other hand, the shape and the smoothness of the interfaces.

A simple idea to obtain a nice balancing (say an equidistribution of the elements) is to introduce a weight to the vertices such as the average of the volumes (surfaces in two dimensions) of the elements sharing these vertices. In this way, we implicitly assume that the number of elements in the final mesh is proportional to the size of the initial elements. This choice is a reasonable idea when a uniform density is expected, but less reasonable in other situations. Thus, in such cases, it is necessary to adjust the weights with information of a metric nature. In other words, two initial elements of identical size do not necessarily lead to the same number of elements after meshing.

For a domain in two dimensions, the interface between sub-domains is basically formed by line segments which separate the domain from one side to the other. In three dimensions, the interfaces are composed of a series of triangular facets and the angles between two such facets may be arbitrary: thus, the geometry is likely to be badly-shaped (disturbed), resulting in a rather peculiar aspect at the interface levels.

Then, the method includes the following:

- interface meshing,
- sub-domain definitions and balancing these over the different processors,
- serial meshing for each sub-domain (one sub-domain in one processor).

Relationships between the different meshes are established at the end of the process in order to allow the transfer of data at a later stage.

Method 2. Following this approach, a mesh of the domain is not required and only a mesh of the domain boundary is used. The idea is then to directly build one or several separators (lines or curves, planes or surfaces) so as to define several sub-domains based on these boundaries.

The method primarily proposed by [Galtier, George-1996], [Galtier-1997] is based on the inductive Delaunay triangulation notion (termed projective Delaunay triangulation in the above references). To introduce this notion, let us recall that:

$$V_i = \{P \quad \text{such that} \quad d(P, P_i) \leq d(P, P_j), \quad \forall j \neq i\}$$

allows us to define the Voronoï cell associated with point P_i, a member of a given cloud of points (Chapter 7). The dual of these cells (which constitute the so-called Voronoï diagram) is the Delaunay triangulation of the convex hull of the points in the cloud. This triangulation satisfies the empty ball criterion.

By analogy, we now consider a plane Π and we define the set:

$$V_i^{(\Pi)} = \{P \in \Pi \ \text{ such that} \quad d(P, P_i) \leq d(P, P_j), \quad \forall j \neq i\}\,, \tag{24.3}$$

which is a polygon in $\mathbb{R}^3$. The dual of the different cells defined in this way is a set of triangular facets in $\mathbb{R}^3$ obtained by joining the points whose cells share a common edge. This set of facets (triangles) satisfies a criterion similar to that of the empty ball, referred to as the *inductive* criterion hereafter. This set of facets is denoted by $\mathcal{F}^{\Pi}$.

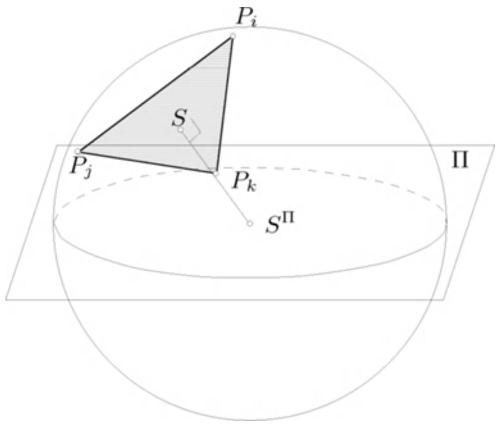


Figure 24.6: *The inductive empty ball criterion. The ball centered in the plane* Π *passing through the three vertices of facet* $P_iP_jP_k$ *is empty.*

In [Galtier-1997] it is proved that the facets constructed in this way form a separator in the domain (here, the convex hull of the cloud). Thanks to this result, it is possible to split this particular domain into two parts. The principle of this partitioning is to use this idea and to apply it to a domain in $\mathbb{R}^3$ (and not only to the convex hull of a set of points).

The proof is based on some assumptions that simplify the problem (for instance, about the point positions). First, we show that the set of facets $\mathcal{F}^{\Pi}$ is a planar graph. Then, we notice that a triangle $P_iP_jP_k$ is a facet in $\mathcal{F}^{\Pi}$ if and only if the three Voronoï cells, after Definition (24.3), are not empty and share a vertex. We then establish that these faces are Delaunay-admissible (following the definition in Chapter 9) and therefore will be formed in every Delaunay triangulation. We deduce in addition that the facets in $\mathcal{F}^{\Pi}$ define a separator in the convex hull of the domain.

The principle of the partitioning method is then to apply these issues to an arbitrary domain in $\mathbb{R}^3$ (and not only in a convex hull problem) and, in particular, to remove the above assumptions, which were made for the sake of simplification (the points are now assumed to be in an arbitrary position and not in a general position). The main steps of the splitting method are as follows (in the case where the separator entity is constructed from a plane):

- a polygonal line (a set of edges among the triangle edges in the mesh of the domain surface) is found such that this set of segments is Delaunay admissible for the inductive empty ball criterion,
- the points on this line are inserted using a Delaunay-type algorithm, thus providing a mesh of a surface whose boundary is this line.

In this way, an initial mesh of a surface that separates the domain into two parts is obtained. Then:

- some points are created on this surface (for instance, as in Chapter 7, by using the edges as a spatial support),
- these points are inserted.

On completion, we have a mesh of the separator surface. Thus, it is possible to define two sub-domains (through their boundary meshes) whose boundaries are composed of the union of the separator mesh with the adequate parts of the surface mesh of the entire domain.

By repeating this process, it is possible, step by step, to cut the initial domain into several sub-domains. As the interface between two sub-domains is defined in a unique way, a classical mesh construction algorithm (i.e., such that boundary integrity is maintained) can be suitably applied in each sub-domain[5].

Notice, and here is our concern, that the serial mesher is used in parallel meaning that the global meshing effort has been distributed over several processors (one per sub-domain).

A few remarks may be given. In specific, questions arise about the way in which plane Π mentioned above must be chosen and, this being done (a separator surface being available), about how to discretize this surface so as to obtain a mesh whose density reflects the desired element sizes. The recurrent problem of well-balancing the task must also be addressed.

Regarding this last question, notice that if it is possible to evaluate, even in a rough way, the volume of the sub-domains (with regard to the density of the points in their boundaries, which is the only data available at this time), then it

[5]Thus, due to the required properties, it seems natural to use a Delaunay-type meshing method.

is possible to predict the number of elements (its order of magnitude) that will be constructed within each sub-domain. For more details about this method, we refer the reader to the already-mentioned references.

Other methods. Multi-block type methods (Chapter 4) as well as methods based on a space decomposition (Chapter 5) are also essentially decomposition methods. Other methods can also be considered, for example, at the level of the CAD system (thus, before any meshing concerns).

The most favorable and automated method is most probably the *quadtree-octree* type method which is based on a recursive decomposition of a box enclosing the domain of interest. The decomposition here is directly connected with the underlying tree structure.

This approach will be discussed below, seen as a parallel meshing method, in the sense where the same principle applies even if some modifications are made.

24.2 Parallel meshing process

Throughout this section, we will consider two cases. First, we are concerned with the design of a static process of parallel meshing, i.e., used once, then, we will see the case of a computation loop. The computational process is then a dynamic task where at each iteration step of the loop, the issue of parallelism must be examined.

One-step process

The load balancing between the members in the partition must be ensured at the beginning (i.e., when constructing the partition) or, this having been constructed as well as possible, before using the resulting meshes (i.e., before any actual computation). Thus, we can distinguish between two types of load balancing approaches, *a priori* load balancing and *a posteriori* load balancing.

***A priori* load balancing.** When the processors used at a later stage for the computation have comparable speed-up, the effort balancing can be done *a priori* by distributing identical numbers of vertices (elements) to each processor. In contrast, when the speed-up varies (from processor to processor) or when some processors are busy with other tasks, an equidistribution is not the optimal solution. It is then necessary to re-balance the loads during the computation.

As a general rule, such a load balancing is tedious to obtain. To realize this, it is only necessary to keep in mind the different partitioning approaches previously discussed. Hence, for example, the key idea to ensure a good load balancing is to introduce some weights during the partitioning stage. This idea implicitly assumes that the number of elements in the final mesh is proportional to the size of the initial mesh. When the element density is not uniform (which, in practice, is frequent), such an assumption is obviously wrong. Then, the weights have to be adjusted to take into account metric specifications, which makes the load balancing stage more delicate.

***A posteriori* load balancing.** The load balancing per processor is basically obtained by *migration.* The migration step consists of moving one element or a set of elements from one processor to another (or to several processors), if the load from one processor to the others is rather different [Coupez-1996]. Moreover, the migration step may serve some other purposes such as obtaining a nice smoothness at the sub-domain interfaces and can also be used as a way to deal with non-connected sub-domains.

The goal of the partitioning algorithms is to minimize data exchange between the processors and to balance the computational effort in each processor. The data migration actually consists of collecting and updating the links between the mesh entities and the processors when some entities are assigned to one (or several) different processors. This is generally done in three steps:

- a *source* processor sends some data items to a *target* processor,
- the source and target processors update the information related to the transferred data items,
- the source and target processors indicate this change to all the processors with access to the data items that are exchanged (see below).

The efficiency of the migration procedure then greatly depends on the volume of the data items exchanged as well as on the data structure used in this task.

Data structures. (Adaptive) mesh generation methods for unstructured mesh construction when used in distributed memory architecture require specific data structures suitable for efficiency queries and data item exchanges from processor to processor. In particular, information about adjacency relationships must be considered.

At a given mesh entity level, it is necessary to know all the relationships (links) between this entity and the different processors. In principle, only a single processor really *possesses* this entity but several other processors may have some links to it (thus, this information is not duplicated in each processor). The edges and faces of the sub-domain boundaries are then shared by several processors.

It is clear that in the first step of the migration process the amount of data item exchange is proportional to the number of entities exchanged (thus, to np the number of vertices). On the other hand, the two other steps depend on the number of entities in the sub-domain boundaries (thus, typically in $\mathcal{O}(np^{\frac{1}{2}})$).

Parallel adaptation loop

Here, load balancing is a *dynamic* task that must be adapted at each new iteration step (or, at least, when the sizes of the members in the partition become rather different). This load balancing is a fundamental issue since a physical problem with a solution that has a large variation may, during the iteration in an adaptive computational process, lead to a large variation in the number of elements in a given region (thus, in a given processor). However, in contrast to the previous case, it is possible to collect some information from the current partition and the behavior of the solution related to it in order to deduce the next partition in the iterative process in progress.

Without mentioning here the probabilist style methods, some algorithms may use a *defect* function that reflects the difference in the load balancing between neighboring elements [Löhner, Ramamurti-1993], in order to govern data exchanges between processors. We may also use some variations of the partitioning algorithms previously described that operate on distributed data. Typically, the partitioning algorithms are either of a geometrical nature (inertia axis based method for instance) or of a topological nature (using the adjacency relationships of a tree structure; see below).

24.3 Parallel meshing techniques

First, it could be observed that the parallel aspect can be present at the input data level, the data being distributed between the processors, or at the level of the tasks (which are then distributed) or, again, in a combination of these two levels.

Distributing the data (without task exchange) involves making the task in each processor independent of the others. Distributing the tasks leads to sending some requests from processor to processor when dealing with an entity (located in one processor), and implies some processing from the proprietary processor, but also from some neighboring processors.

Various classical techniques for mesh generation may be performed in parallel. Among these, the Delaunay method is a potentially interesting candidate. Indeed, this method widely uses a *proximity criterion*, namely the empty ball criterion. Given a set of points, we may consider separating these points into several disjoint sub-sets and thus make the point insertion procedure uncoupled. In other words, the mesh can be completed in parallel.

In practice, this point separation could either be decided *a priori*, which is in general a tedious issue, or prescribed by means of a constraint about edges (resp. faces) in a coarse triangulation used as a separator. Once this separator has been defined, the problem turns to meshing with consistency the sub-domains identified in this way, while their interface (i.e., the separator) could be meshed:

- before constructing the mesh of the sub-domains,
- at the same time as the sub-domains,
- after constructing the mesh of the sub-domains,

which, in fact, leads to three classes of parallel meshing methods.

Another class of methods for parallel use is based on the spatial decomposition of a box enclosing the computational domain. Methods like *quadtree-octree* (Chapter 5) are then natural candidates since they make use of a hierarchical tree structure and thus could be used for domain partitioning purposes.

Are any other meshing methods candidates for parallelism ? The answer is not so trivial. In particular, it is of interest to examine whether an advancing-front type method (Chapter 6) may include some degree of parallelism.

Within this section, we give some indications about these various approaches and briefly examine their ability to include some parallel aspect.

Remark 24.1 *Note that the concern here is to construct a mesh in parallel and not to define a parallel mesh construction or computational process. In the present context, the wish is to have a complete mesh available (after merging together the sub-meshes) while in the other case, it is not strictly necessary (see the previous sections) or we do not want to have, at a given time, the complete mesh of the domain.*

•Delaunay-type method

Let us recall that the key point of a Delaunay-style method is the point insertion process, the *Delaunay kernel*, as introduced in Chapter 7. This process is a local one in the sense that it involves the cavity of the point to be inserted. Depending on whether the elements in this cavity all belong to the same processor or are common to two or more processors, it is easy to imagine that the degree of parallelism could be different. Simply remark, in addition, that finding this locality could be a non-trivial problem (or, at least, a time consuming task).

Various ways to implement the Delaunay kernel have been proposed based on the various possible configurations (in this respect, see [Okusanya, Peraire-1996], [Chrisochoides, Sukupp-1996] or [Chew *et al.* 1997], for example).

General principle. For the sake of simplicity, let us consider a domain Ω in two dimensions, provided through a discretization $\Gamma(\Omega)$ of its boundary. Using this sole data, an initial mesh is constructed by inserting, one at a time, the points in $\Gamma(\Omega)$, and then by enforcing the boundary entities that are missing, if any[6].

In this way, an *empty* mesh is obtained (i.e., without any internal points) whose internal edges join one domain side to another. We pick one of these edges, say AB, which separates Ω into two sub-domains, Ω_1 and Ω_2, of approximately the same size. Each of these two domains is then assigned to one processor. Now, let us look at a condition to ensure the independence of a point with respect to the sub-domains:

- every point P inside Ω_1 is independent of Ω_2 if the cavity $\mathcal{C}_P$ is entirely included in Ω_1: $\mathcal{C}_P \subset \Omega_1 \quad \text{and} \quad \mathcal{C}_P \not\subset \Omega_2$,
- on the other hand, if $\mathcal{C}_P \cap \Omega_2 \neq \emptyset$, point P depends on Ω_1 and Ω_2 and its insertion affects the mesh both in Ω_1 and in Ω_2.

Note that a (sufficient) condition for independence is that the circle of diameter AB is empty (Chapter 9). After this assumption, the mesh is carried out in parallel without any exchange of tasks.

[6]Thus, the case of a *constrained* Delaunay mesh (Chapter 7).

Meshing the separator. It is easy to see that, in general, the output completed using the previous algorithm will be rather poor because the constraint implies that there is no point in some vicinity of AB (i.e., its open ball) and only some post-processing may be able to fill up this region (and thus to split AB).

Using a constrained Delaunay algorithm by prescribing AB as a constraint leads to the same issue and the global process remains "processor-independent". The presence of points in some vicinity of AB necessarily gives a bad quality mesh and some post-processing is again required in order to enhance this result (and, in particular, to split AB).

At last, a "classical" meshing is possible, while using data and task exchanges. This can be done in two ways. On the one hand, we maintain AB as a constraint but we may even split this segment (by means of point insertion) in such a way as to preserve the global quality of the mesh. In such a situation, the processor responsible for this point creation indicates this request to the other processor in order to make the mesh construction synchronous at the interface level (then a delay must be expected for the simultaneous updating of the two meshes because the interface must remain coherent).

The other way to proceed is more tedious. Edge AB (or any interface edge) is no longer a constraint and may disappear. The interface is then modified during the processing which impedes the global process in terms of data and task organization.

After this discussion about the principles, we may observe that there are only a few examples of implementations of these ideas in two dimensions, for architectures with a limited number of processors (actually, from 2 to 16). The extension of these approaches to three dimensions remains a relatively open problem. In particular, constraining a mesh (Chapter 7) is a tedious problem which, in practice, is widely based on heuristics. Moreover, identifying a separator is not obvious because even a face joining two "opposite" sides of the domain does not in general separate this domain into two parts.

• *Quadtree-octree*-type method

Here two approaches will be discussed. The first, which is mainly of academic interest, assumes the tree structure to be known and, using this information, simply seeks to balance the terminal cells (the leaves) in the available processors. The second one, which is more interesting from a practical point of view, considers the tree structure construction in parallel.

Distribution of the nodes in a given tree. For the sake of simplicity, we assume the tree to be balanced by means of the [2:1] rule (Chapter 5), thus every cell edge is shared by at most three cells, in two dimensions, while a cell facet is shared by five cells at most in three dimensions. Following on from this, the natural link between the number of terminal cells and the number of elements in the resulting mesh[7] could be observed.

[7]In fact, we saw in Chapter 5 that the cell sizes are locally compatible with the element size distribution function.

Therefore, obtaining a nice load balancing in the processors involves partitioning the tree in such a way that the number of leaves assigned to each processor leads to the same number of elements. Meshing each sub-domain is then performed in parallel, with no communication between the processors. Since a quadrant boundary edge (resp. face) may belong to several processors, special care is necessary regarding the meshing algorithm used to mesh these interfaces (as the resulting meshes must be conformal). On the other hand, if the leaves in the interface are inside the domain, predefined patterns (or *templates*) can be used. The adjacency relationships between the tree cells must be enriched with information about the processor to which the cell belongs [Saxena, Perucchio-1992].

Following these remarks, the general scheme of a tree partitioning method simply consists of the following:

```
$ne \leftarrow 0$
FOR each leaf $c$ in the tree
   compute the number of elements that must be created $ne(c)$
   $ne \leftarrow ne + ne(c)$, (increment the total number of elements)
END FOR
compute the load per processor:   $cpp = \frac{ne}{nbproc}$
traverse the tree
$nep \leftarrow 0$
$p \leftarrow 1$
FOR each leaf $c$
   IF $nep < cpp$
      $nep \leftarrow nep + ne(c)$
      assign the $ne(c)$ elements to processor $p$
   OTHERWISE
      $p \leftarrow p + 1$
   END IF
END FOR
```

To take advantage of the adjacency relationships in the tree (and thus obtain sub-domains that are as connected as possible), one may traverse the tree at first (using a pre-order, as seen in Chapter 2).

Distributed construction of the tree. A more interesting problem is the design in parallel of the whole meshing process, i.e., the tree construction and its use to form the mesh elements.

The method proposed in [deCougny *et al.* 1996] constructs an *octree* which is balanced in parallel. The entry data in this algorithm is a discretization of the domain surface. With each vertex in this triangulation is associated a size (that corresponds to the average of the lengths of the incident edges), thus defining a discrete size map. This scalar value h is then modified in an integer value related to a level l in the tree (i.e., the level of the cell within which the vertex falls), by means of the formula (Chapter 5):

$$l = log_2\left(\frac{b}{h}\right) ,$$

where b is the length of a side of the enclosing box.

Conceptually, the tree construction includes two stages: the construction of local sub-trees and the refinement of these sub-trees. The four (resp. eight) initial cells are assigned to four (eight) different processors. Then, all the terminal cells are iteratively subdivided once or more (if necessary) and the cells resulting from the decomposition are assigned to the different processors according to the load they imply. This effort approximatively corresponds to the number of vertices to be inserted (i.e., this is the stopping criterion of the process, Chapter 5) inside the volume related to one cell. Notice that the non-terminal cells are then replaced by the leaves. In this way, on completion of the construction (once all the processors have been assigned an equivalent load), each terminal cell corresponds to the root of a sub-tree and exists in only one processor. The sub-tree construction is in this way balanced in a natural way, each processor having approximately the same number of vertices to be inserted.

Then sub-trees are then constructed independently in each processor by subdividing their terminal cells until a level related to the desired local size is achieved.

The meshes of the sub-trees are then completed in parallel (with no communication between the processors) by using in each sub-tree a technique similar to that described in Chapter 5.

Remark 24.2 *In three dimensions, a tree re-balancing step may be made after the mesh of the terminal cells inside the sub-domains has been completed. This is due to the fact that an advancing-front type algorithm is often used to mesh the boundary leaves (the number of elements in these cells being tedious to evaluate* a priori*).*

Complexity. The complexity in time of the construction process is of the order of $\mathcal{O}(np/nproc \log(nproc))$, where $np/nproc$ is the load per processor and where the term in $\log(np)$ corresponds to the number of iteration steps in the algorithm (and thus to the number of refinement levels in the tree). While the sub-tree refinement is of a complexity $\mathcal{O}\left(\frac{np}{nproc}\log\left(\frac{np}{nproc}\right)\right)$.

•Other methods?

The third category of automated mesh generation methods not yet discussed is that of the advancing-front type methods. At present, we have no knowledge of developments or specific issues for this type of approach. We may just mention that the advancing front strategy may be used for a parallel mesh construction of sub-domains while the underlying decomposition is completed using a tree structure or a Delaunay-type empty mesh (with no internal point); see, for instance, [Shostko, Löhner-1995] or [Löhner-1998]).

★

★ ★

Parallelism appears to be a practical solution to handle systems with several million (or dozens of millions of) unknowns[8].

The impact on meshing technologies seems to be, as we have seen, more about the parallelization of the meshing processes (partitioning, load balancing and communication) than the parallelization of the meshing methods themselves. It is likely that such parallel methods will be subjected to important developments in the coming future and, as a consequence, significant advances are expected. Thus this subject appears to be both open and very promising.

[8]Such systems are nowadays used in computational fluid dynamics where the order of magnitude of the physics varies greatly.

Bibliography

[Abgrall, 2004] R. ABGRALL (2004), Numerical discretisation of boundary conditions for first order Hamilton Jacobi equations, *SIAM Journal on Numerical Analysis.*

[Aho *et al.* 1983] A. AHO, J. HOPCROFT AND J. ULLMAN (1983), *Data structures and algorithms*, Addison-Wesley, Reading, Mass.

[Alauzet *et al.* 2007] F. ALAUZET, P. FREY, P.L. GEORGE AND B. MOHAMMADI (2007), 3D transient fixed point mesh adaptation for time-dependent problems: Application to CFD simulations, *J. Comp. Phys.*, **222**, 592-623.

[Allgower, Schmidt-1985] E.L. ALLGOWER AND P.H. SCHMIDT (1985), An algorithm for piecewise-linear approximation of an implicitly defined manifold, *SIAM J. Numer. Anal.*, **22**, 322-346.

[Allgower, Gnutzmann-1987] E.L. ALLGOWER AND S. GNUTZMANN (1987), An algorithm for piecewise linear approximation of implicitly defined two-dimensional surfaces, *SIAM J. Numer. Anal.*, **24**(2), 452-469.

[Allgower, Georg-1990] E.L. ALLGOWER AND K. GEORG (1990), *Numerical continuation methods: an introduction, Springer-Verlag.*

[Allgower, Gnutzmann-1991] E.L. ALLGOWER AND S. GNUTZMANN (1991), Simplicial pivoting for mesh generation of implicitly defined surfaces, *CAGD*, **8**, 305-325.

[Amenta *et al.* 1997] N. AMENTA, M. BERN, AND D. EPPSTEIN (1997), Optimal point placement for mesh smoothing, *8th ACM-SIAM Symp. Discrete Algorithms*, New Orleans, 528-537.

[Amezua *et al.* 1995] E. AMEZUA, M.V. HORMAZA, A. HERNANDEZ AND M.B.G. AJURIA (1995), A method for the improvement of 3D solid finite-element meshes, *Advances in Engineering Software*, **22**, 45-53.

[Anglada *et al.* 1999] M.V.ANGLADA, N.P. GARCIA AND P.B. CROSA (1999), Directional adaptive surface triangulation, *Computer Aided Geometric Design*, **16**, 107-126.

[Armstrong *et al.* 1993] C.G. ARMSTRONG, D.J. ROBINSON, R.M. MCKEAG, D. SHEEHY, C. TOH AND T.S. LI (1993), Abstraction and meshing of 3D stress analysis models, *Proc. ACME Research Conf.*, Sheffield Univ., England.

[Armstrong *et al.* 1995] C.G. ARMSTRONG, D.J. ROBINSON, R.M. MCKEAG, T.S. LI, S.J. BRIDGETT AND R.J. DONAGHY (1995), Applications of the medial axis transform in analysis modelling, *Proc. Fifth International Conference on Reliability of Finite Element Methods for Engineering Applications*, Nafems, 415-426.

[Armstrong *et al.* 1995] C.G. ARMSTRONG, D.J. ROBINSON, R.M. MC KEAG, T.S. LI, S.J. BRIDGETT, R.J. DONAGHY AND C.A. MC GLEENAN, Medials for meshing and more, *Proc. 4th Int. Meshing Roundtable*, 277-288.

[Attili-1997] B.S. ATTILI (1997), Tracing implicitly defined curves and the use of singular value decomposition, *Applied Num. Math.*, **25**, 1-11.

[Aurenhammer-1987] F. AURENHAMMER (1987), Power diagrams properties, algorithms and applications, *SIAM J. Comput.*, **16**, 78-96.

[Aurenhammer-1990] F. AURENHAMMER (1990), A new duality result concerning Voronoï diagrams, *Discrete Comput. Geom.*, **5**, 243-254.

[Aurenhammer-1991] F. AURENHAMMER (1991), Voronoï diagrams: a survey of a fundamental geometric data structure, *ACM Comput. Surv.*, **23**, 345-405.

[Avis, Bhattacharya-1983] D. AVIS AND B.K. BHATTACHARYA (1983), Algorithms for computing *d*-dimensional Voronoï diagrams and their duals, in F.P. Preparata eds. *Advances in Computing Research*, **1**, 159-188.

[Avis, ElGindy-1987] D. AVIS AND H. ELGINDY (1987), Triangulating point sets in space, *Discrete Comput. Geom.*, **2**, 99-111.

[Ayers *et al.* 1999] AYERS R., HASSAN O. , MORGAN K. AND WEATHERILL N. P. (1999), The role of computational fluid dynamics in the design of the Thrust supersonic car, *Design Optimization*, **1**, 106-126.

[Azevedo, Simpson-1989] E.F. D'AZEVEDO AND B. SIMPSON (1989), On optimal interpolation triangle incidences, *Siam J. Sci. Stat. Comput.*, **10**, 1063-1075.

[Azevedo-1991] E.F. D'AZEVEDO (1991), Optimal triangular mesh generation by coordinate transformation, *Siam J. Sci. Stat. Comput.*, **12**, 755-786.

[Azevedo, Simpson-1991] E.F. D'AZEVEDO AND B. SIMPSON (1991), On optimal triangular meshes for minimizing the gradient error, *Numerische Mathematik*, **59**(4), 321-348.

[Avnaim *et al.* 1994] F. AVNAIM, J.D. BOISSONNAT, O. DEVILLERS, F.P. PREPARATA AND M. YVINEC (1994), Evaluating signs of determinants using single-precision arithmetic, *RR INRIA* **2306**.

[Babuska, Aziz-1976] I. BABUŠKA AND A. AZIZ (1976), On the angle condition in the finite element method, *SIAM J. Numer. Analysis*, **13**, 214-227.

[Babuska, Rheinboldt-1978] I. BABUŠKA AND W.C. RHEINBOLDT (1978), A posteriori error estimates for the finite element method, *Int. J. Numer. Methods Eng.*, **12**, 1597-1615.

[Babuska, Guo-1988] I. BABUŠKA AND B.Q. GUO (1988), The hp version of the finite element method for domains with curved boudaries, *SIAM J. Numer. Anal.*, **25**(4), 837-861.

[Babuska *et al.* 1989] I. BABUŠKA, M. GRIEBEL AND J. PITKARANTA (1989), The problem of selecting the shape functions for p-type finite elements, *Int. J. Numer. Methods Eng.*, **28**, 1891-1908.

[Babuska, Suri-1990] I. BABUŠKA AND M. SURI (1990), The p and the hp versions of the finite element method : An overview, *Comp. Meth. Appl. Mech. Engng.*, **80**(1-3), 5-26.

[Babuska *et al.* 1994] I. BABUŠKA, T. STROUBOULIS, C.S. UPADHYAY, S.K. GANGARAJ AND K. COPPS (1994), Validation of *a posteriori* error estimation by numerical approach, *Int. J. Numer. Methods Eng.*, **37**, 1073-1123.

[Babuska, Guo-1996] I. BABUŠKA AND B.Q. GUO (1996), Approximation properties of the hp version of the finite element method, *Comp. Meth. Appl. Mech. Engng.*, **133**, 319-346.

[Baehmann *et al.* 1987] P.L. BAEHMANN, S.L. WITTCHEN, M.S. SHEPHARD, K.R. GRICE AND M.A. YERRY (1987), Robust geometrically-based automatic two-dimensional mesh generation, *Int. j. numer. methods eng.*, **24**, 1043-1078.

[Bai, Brandt-1987] D. BAI AND A. BRANDT (1987), Local mesh refinement multilevel techniques, *Siam J. Sci. Stat. Comput.*, **8**(2), 109-134.

[Bajaj *et al.* 1993] C. BAJAJ, I. IHM AND J. WARREN, Higher-order interpolation and least-squares approximation using implicit algebraic surfaces, *ACM Trans. Graph.*, **12**, 327-347.

[Bajaj *et al.* 1997] C. BAJAJ, J. BLINN, M.P. GASCUEL, A. ROCKWOOD, B. WYVILL AND G. WYVILL, *Introduction to Implicit Surfaces*, J. Bloomenthal ed., Morgan Kaufman.

[Baker *et al.* 1988] B.S. BAKER, E. GROSS AND C. RAFFERTY (1988), Nonobtuse triangulation of polygons, *Discrete and Comp. Geometry*, **3**, 147-168.

[Baker-1986] T.J. BAKER (1986), Three dimensional mesh generation by triangulation of arbitrary point sets, *Proc. AIAA 8th Comp. Fluid Dynamics Conf.* Honolulu, HI, **87**-1124-CP.

[Baker-1988] T.J. BAKER (1988), Generation of tetrahedral meshes around complete aircraft, *Numerical grid generation in computational fluid mechanics '88*, Miami, FL.

[Baker-1989a] T.J. BAKER (1989), Developments and trends in three-dimensional mesh generation, *Appl. Num. Math.*, **5**, 275-304.

[Baker-1989b] T.J. BAKER (1989), Element quality in tetrahedral meshes, *Proc. 7th Int. Conf. on Finite Element Methods in Flow Problems*, Huntsville, AL.

[Baker-1989c] T.J. BAKER (1989), Automatic Mesh Generation for Complex Three-Dimensional Regions Using a Constrained Delaunay Triangulation, *Eng. Comp.*, **5**, 161-175.

[Baker-1989d] T.J. BAKER (1989), Mesh generation by a sequence of transformations, *Appl. Numer. Math.*, **2**(6), 515-528.

[Baker-1991a] T.J. BAKER (1991), Shape Reconstruction and Volume Meshing for Complex Solids, *Int. J. Numer. Methods Eng.*, **32**, 665-675.

[Baker-1991b] T.J. BAKER (1991), Single block mesh generation for a fuselage plus two lifting surfaces, *Proc. Third Int. Conf. Numerical Grid Generation*, Barcelona, Spain, 261-272.

[Baker-1992] T.J. BAKER (1992), Mesh Generation for the Computation of Flowfields over Complex Aerodynamic Shapes, *Comp. Math. Applic.*, **24**(5-6), 103-127.

[Baker-1997] T.J. BAKER (1997), mesh adaptation strategies for problems in fluid dynamics, *Finite Elements in Analysis and Design*, **25**(3-4), 243-273.

[Baker-2001] T.J. BAKER (2001), Mesh movement and metamorphosis, *Proc. 10th Int. Meshing Roundtable*, 387-396.

[Bakhvalov-1973] N. BAKHVALOV (1973), *Méthodes numériques*, Eds. Mir, Moscou.

[Bank *et al.* 1983] R.E. BANK, A.H. SHERMAN, A. WEISER (1983), Refinement algorithms and data structure for regular local mesh refinement, *Scientific Computing*, R. Stepleman et al. (eds), IMACS, North Holland.

[Bank, Chan-1986] R. E. BANK AND T.F. CHAN (1986), PLTMGC: a multigrid continuation program for parametrized nonlinear elliptic system, *Siam J. Sci. Stat. Comput.*, **7**(2), 540-559.

[Bank-1990] R. E. BANK (1990), *PLTMG: a software package for solving elliptic partial differential equations*, Frontiers in applied mathematics, Siam.

[Bank-1997] R. E. BANK (1997), Mesh smoothing using *a posteriori* estimates, *Siam J. Numer. Anal.*, **34**(3), 979-997.

[Bänsch-1991] E. BÄNSCH (1991), Local Mesh Refinement in 2 and 3 Dimensions, *Impact of Comp. in Sci. and Eng.*, **3**, 181-191.

[Barequet *et al.* 1996a] , G. BAREQUET, M. DICKERSON AND D. EPPSTEIN (1996), On triangulating three-dimensional polygons, *Computational Geometry'96*, 38-47, Philadelphia, PA, USA.

[Barequet *et al.* 1996b] , G. BAREQUET, B. CHAZELLE, L.J. GUIBAS, J. MITCHELL AND A. TAL (1996), Boxtree: a hierarchical representation for surfaces in 3D, *Eurographics'96*, **15**(3), 387-396.

[Barequet *et al.* 1998] G. BAREQUET, C.A. DUNCAN AND S. KUMAR (1998), RSVP: A geometric toolkit for controlled repair of solid models, *IEEE Trans. on Visualization and Computer Graphics*, **4**(2), 162-177.

[Barfield-1970] W.D. BARFIELD (1970), An optimal mesh generator for Lagrangian hydrodynamic calculations in two space dimensions, *J. Comput. Physics*, **6**, 417-429.

[Bartels *et al.* 1987] R. H. BARTELS, J.C. BEATY AND B.A. BARSKY (1987), *An Introduction to Splines for Use in Computer Graphics and Geometric Modeling*, Morgan Kaufmann.

[Bartels *et al.* 1988] R. H. BARTELS, J.C. BEATY AND B.A. BARSKY (1988), *B-splines, Mathématiques et CAO*, **6**, Hermès, Paris.

[Barth-1994] T.J. BARTH (1994), Aspects of unstructured grids and finite-volume solvers for the Euler and Navier-Stokes equations, *Technical report*, VKI Lecture Series 1994-05.

[Bathe, Chae-1989] K.J. BATHE, S.W. CHAE (1989), On automatic mesh construction and mesh refinement in Finite Element analysis, *Computers & Structures*, **32**(3/4), 911-936.

[Batina-1990] J. BATINA (1990), Unsteady Euler airfoil solutions using unstructured dynamic meshes, *AIAA Journal*, **28**(8), 1381-1388.

[Baumgart-1974] B.G. BAUMGART (1974), Geometric Modeling for Computer Vision, *AIM-249, CS-TR-74-463*, Stanford Artificial Intelligence Lab., Stanford University.

[Baumgart-1975] B.G. BAUMGART (1975), A Polyhedron Representation for Computer Vision., in *National Computer Conference*, Montvale, N.J.: AFIPS Press, 589-596.

[Beall, Shephard-1997] M.W. BEALL AND M.S. SHEPHARD (1997), A general topology-based mesh data structure, *Int. J. Numer. Methods Eng.* , **40**(8), 1573-1596.

[Benzley *et al.* 1995] S.E. BENZLEY, E. PERRY, M. MERKLEY, B. CLARK AND G. SJAARDEMA, A comparison of all-hexahedral and all-tetrahedral finite element meshes for elastic and elasto-plastic analysis, *Proc 4th Int. Meshing Roundtable*, 179-191.

[de Berg *et al.* 1997] M. DE BERG, M. VAN KREVELD, M. OVERMARS AND O. SCHWAZKOPF (1997), *Computational geometry: algorithms and applications*, *Springer-Verlag*, Berlin Heidelberg.

[Berger-1978] M. BERGER (1978), *Géométrie tome 3 : convexes et polytopes, polyèdres réguliers, aires et volumes*, Fernand Nathan, Paris.

[Berger, Gostiaux-1987] M. BERGER ET B. GOSTIAUX (1987), *Géométrie différentielle : variétés, courbes et surfaces*, PUF, Paris.

[Berger, Colella-1989] M.J. BERGER AND P. COLELLA (1989), Local adaptive mesh refinement for shock hydrodynamics, *J. Comput. Physics*, **82**, 64-84.

[Berger, Jameson-1985] M.J. BERGER AND A. JAMESON (1985), Automatic adaptive grid refinement for Euler equations, *AIAA J.*, **23**(4), 561-568.

[Berger, Oliger-1984] M.J. Berger and J. Oliger (1984), Adaptive mesh refinement for hyperbolic partial differential equations, *J. Comput. Physics*, **53**, 484-512.

[Bern *et al.* 1990] M. Bern, D. Eppstein and J. Gilbert (1994), Provably good mesh generation, *J. Comp. Sys. Sci.*, **48**, 384-409.

[Bern *et al.* 1991] M. Bern, D. Eppstein and F. Yao. (1991), The expected extremes in a Delaunay triangulation, *Int. J. Comp. Geom. & Appl.*, **1**, 79-92.

[Bern *et al.* 1991] M. Bern, H. Edelsbrunner, D. Eppstein, S. Mitchell and T.S. Tan (1991), Edge insertion for optimal triangulations, *Disc. & Comp. Geom.*, **10**, 47-65.

[Bern-1995] M. Bern (1995), Compatible tetrahedralizations, *Fundamenta Informaticae*, **22**, 371-384.

[Bern, Eppstein-1995] M. Bern and D. Eppstein (1995), Mesh generation and optimal triangulation, *Computing in Euclidean Geometry*, 2nd ed., D.-Z. Du and F.K. Hwang, eds., World Scientific, 47-123.

[Bern, Eppstein-1997] M. Bern and D. Eppstein (1997), Quadrilateral meshing by circle packing, *Proc. 6th Int. Meshing Roundtable*, 7-19.

[Bern, Plassmann-1998] M. Bern and P. Plassmann (1998), *Mesh generation*, Joerg Sack ed., Elsevier.

[Bernardi-1996] C. Bernardi (1996) Estimations *a posteriori* : une mode ou un outil de base, *Matapli*, **48**, 19-30.

[Bernadou *et al.* 1988] M. Bernadou, P.L. George, A. Hassim, P. Joly, P. Laug, B. Muller, A. Perronnet, E. Saltel, D. Steer, G. Vanderborck et M. Vidrascu (1988), Modulef: une bibliothèque modulaire d'éléments finis, INRIA (Eds.).

[Berzins-1999] M. Berzins (1999), Mesh quality: a function of geometry, error estimates or both?, *Engineering with Computers*, **15**, 236-247.

[Bézier-1986] P. Bézier (1986), *Courbes et surfaces, Mathématiques et CAO*, **4**, Hermès, Paris.

[Blacker, Stephenson-1991] T.D. Blacker and M.R. Stephenson (1991), Paving: a new approach to automated quadrilateral mesh generation, *Int. J. Numer. Methods Eng.*, **32**, 811-847.

[Blacker, Meyers-1993] T.D. Blacker and R.J. Meyers (1993), Seams and wedges in plastering: a 3d hexahedral mesh generation algorithm, *Engr. with Computers*, **9**, 83-93.

[Blinn-1982] J.F. Blinn(1982), A generalization of algebraic surface drawing, *ACM Trans. on Graphics*, **1**(3), 235-256.

[Blom-2000] F.J. Blom (2000), Considerations on the spring analogy, *Int. J. Numer. Methods Fluids*, **32**, 647-668.

[Bloomenthal-1988] J. BLOOMENTHAL (1988), Polygonization of implicit surfaces, *CAGD*, **5**, 341-355.

[Bloomenthal *et al.* 1997] J. BLOOMENTHAL, C. BAJAJ, J. BLINN, M.P. CANI-GASCUEL, A. ROCKWOOD, B. WYWILL, G. WYVILL (1997), *Introduction to implicit surfaces*, Morgan Kaufmann Publishers, San Francisco.

[Blum-1967] H. BLUM, A transformation for extracting new descriptors of shape, in *Models for the perception of speech and visual form*, W. Whaten Dunn eds., MIT Press, 362-380.

[Boissonnat, Yvinec-1995] J.D. BOISSONNAT AND M. YVINEC (1997), *Algorithmic Geometry*, Cambridge University Press.

[Bonet, Peraire-1991] J. BONET AND J. PERAIRE (1991), An alternate digital tree for geometric searching and interaction problems, *Int. J. Numer. Methods eng.*, **31**, 1-17.

[Borgers-1990] C. BORGERS (1990), Generalized Delaunay triangulations of non convex domains, *Comput. Math. Applic.*, **20**(7), 45-49.

[Borouchaki *et al.* 1995] H. BOROUCHAKI, F. HECHT, E. SALTEL AND P.L. GEORGE (1995), Reasonably efficient Delaunay based mesh generator in 3 dimensions, *Proc. 4th Int. Meshing Roundtable*, 3-14.

[Borouchaki, George-1996a] H. BOROUCHAKI ET P.L. GEORGE (1996), Triangulation de Delaunay et métrique riemannienne. Applications aux maillages éléments finis, *Revue européenne des éléments finis*, **5**(3), 323-340.

[Borouchaki, George-1996b] H. BOROUCHAKI ET P.L. GEORGE (1996), Mailleur de Delaunay gouverné par une carte, *C. R. Acad. Sci. Paris*, t. 323, Series I, 1141-1146.

[Borouchaki *et al.* 1996] H. BOROUCHAKI, P.L. GEORGE AND S.H. LO (1996), Optimal Delaunay point insertion, *Int. J. Numer. Methods Eng.*, **39**(20), 3407-3438.

[Borouchaki, George-1997a] H. BOROUCHAKI AND P.L. GEORGE (1997), Aspects of 2D Delaunay mesh generation, *Int. J. Numer. Methods Eng.*, **40**, 1957-1975.

[Borouchaki, George-1997b] H. BOROUCHAKI ET P.L. GEORGE (1997), Maillage des surfaces paramétriques. Partie I: Aspects théoriques, *C.R. Acad. Sci. Paris*, t 324, Series I, 833-837.

[Borouchaki *et al.* 1997a] H. BOROUCHAKI, P.L. GEORGE, F. HECHT, P. LAUG AND E. SALTEL (1997), Delaunay mesh generation governed by metric specifications. Part I: Algorithms, *Finite Elements in Analysis and Design*, **25**(1-2), 61-83.

[Borouchaki *et al.* 1997b] H. BOROUCHAKI, P.L. GEORGE AND B. MOHAMMADI (1996), Delaunay mesh generation governed by metric specifications. Part II: Application examples, *Finite Elements in Analysis and Design*, **25**(1-2), 85-109.

[Borouchaki, Frey-1998] H. BOROUCHAKI AND P.J. FREY (1998), Adaptive triangular-quadrilateral mesh generation, *Int. J. Numer. Methods Eng.*, **41**, 915-934.

[Borouchaki *et al.* 1998] H. BOROUCHAKI, F. HECHT AND P.J. FREY (1997), Mesh gradation control, *Int. J. Numer. Methods Eng.*, **43**(6), 1143-1165.

[Borouchaki *et al.* 1999] H. BOROUCHAKI, P. LAUG AND P.L. GEORGE (1999), About parametric surface meshing, 2^{nd} *Symposium on Trends in Unstructured Mesh Generation, USNCCM'99*, University of Colorado, Boulder, CO.

[Borouchaki *et al.* 2000] H. BOROUCHAKI, P. LAUG AND P.L. GEORGE (2000), Parametric surface meshing using a combined advancing-front-generalized-Delaunay approach, *Int. J. Numer. Methods Eng.*, **49**(1), 233-259.

[Borouchaki *et al.* 2002] H. BOROUCHAKI, A. CHEROUAT, P. LAUG AND K. SAANOUNI (2002), Adaptive remeshing for ductile fracture prediction in metal forming, *C.R. Acad. Sci. Paris*, C.R. Mécanique, t. 330, 709-716.

[Borouchaki *et al.* 2005] H. BOROUCHAKI, N. FLANDRIN ET CH. BENNIS (2005), Diagramme de Laguerre, *C.R. Acad. Sci. Paris*, C.R. Mécanique, **10**, 762-767.

[Bowyer-1981] A. BOWYER (1981), Computing Dirichlet tesselations, *The Comp. J.*, **24**(2), 162-167.

[Brière, George-1995] E. BRIÈRE DE L'ISLE AND P.L. GEORGE (1995), Optimization of tetrahedral meshes, *IMA Volumes in Mathematics and its Applications*, I. Babuska, W.D. Henshaw, J.E. Oliger, J.E. Flaherty, J.E. Hopcroft and T. Tezduyar (Eds.), **75**, 97-128.

[Bristeau, Periaux-1986] M.O. BRISTEAU AND J. PERIAUX (1986), Finite element methods for the calculation of compressible viscous flows using self-adaptive refinement, *VKI lecture notes on CFD.*

[Brown-1979] K.Q. BROWN (1979), Voronoi diagrams from convex hull, *Inform. Process. Lett.*, **9**, 223-228.

[Buratynski-1990] E.K. BURATYNSKI (1990), A fully automatic three-dimensional mesh generator for complex geometries, *Int. J. Numer. Methods Eng.*, **30**, 931-952.

[Bykat-1976] A. BYKAT (1976), Automatic generation of triangular grid-subdivision of a general polygon into convex subregions. Part II: triangulation of convex polygons, *Int. J. Numer. Methods Eng.*, **10**, 1329-1342.

[Canann et al.1993] S. CANANN, M. STEPHENSON AND T. BLACKER (1993), Optismoothing: An optimization-driven approach to mesh smoothing, *Finite Elements in Analysis and Design*, **13**, 185-190.

[Canann et al.1996] S. CANANN, S.N. MUTHUKRISHNAN AND R.K. PHILIPS (1996), Topological refinement procedures for triangular finite element meshes, *Engineering with Computers*, **12**, 243-255.

[Carey-1997] G.F. CAREY (1997), Computational grids: *generation, adaptation and solution strategies,* Taylor and Francis.

[Carey, Oden-1984] G.F. CAREY AND J.T. ODEN (1984), *Finite Elements: computational aspects,* Prentice-Hall.

[do Carmo-1976] M.P. DO CARMO (1976), *Differential geometry of curves and surfaces,* Prentice Hall.

[Carnet-1978] J. CARNET (1978), Une méthode heuristique de maillage dans le plan pour la mise en œuvre des éléments finis, Thesis, Paris.

[Cass *et al.* 1996] R.J. CASS, S.E. BENTLEY, R. MEYERS AND T.J. BAKER (1996), Generalized 3D paving: an automated quadrilateral surface mesh generation algorithm, *Int. J. Numer. Methods Eng.*, **39**, 1475-1489.

[Castillo-1991] J. E. CASTILLO (1991), A discrete variational grid generation method, *SIAM J. Sci. Statist. Comput.*, **12**(2), 454-468.

[Castro-1994] M.J. CASTRO-DIAZ (1994), Mesh refinement over triangulated surfaces, *RR INRIA* **2462**.

[Castro, Hecht-1995] M.J. CASTRO-DIAZ AND F. HECHT (1995), Anisotropic surface mesh generation, *RR INRIA* **2672**.

[Castro *et al.* 1995] M.J. CASTRO-DIAZ, F. HECHT AND B. MOHAMMADI (1995), New Progress in Anisotropic Mesh Adaption for Inviside and Viscous Flow Simulations, *RR INRIA* **2671**.

[Catmull-1974] E. CATMULL (1974), A subdivision Algorithm for Computer Design of curved Surfaces, Univ. Utah Comp. Sci. Dept., UTEC-CSC, 74-133.

[Cavendish-1974] J.C. CAVENDISH (1974), Automatic Triangulation of Arbitrary Planar Domains for the Finite Element Method, *Int. J. Numer. Methods Eng.*, **8**, 679-696.

[Cavendish-1975] J.C. CAVENDISH (1975), Local mesh refinement using rectangular blended finite elements, *J. of Comp. Phys.*, **19**, 211-228.

[Cavendish *et al.* 1985] J.C. CAVENDISH, D.A. FIELD AND W.H. FREY (1985), An approach to automatic three-dimensional finite element mesh generation, *Int. J. Numer. Methods Eng.*, **21**, 329-347.

[Cazals, Puech-1997] F. CAZALS, C. PUECH (1997), Bucket-like space partitioning data structures with applications to ray-tracing, *13th ACM Synposium on Computational Geometry*, Nice.

[Cebral, Löhner-1999] CEBRAL, J., AND R. LÖHNER (1999), From Medical Images to CFD Meshes, *Proc. 8th Int. Meshing Roundtable.*

[Celeux *et al.* 1989] G. CELEUX, E. DIDAY, G. GOVAERT, Y LECHEVALLIER ET H. RALAMBONDRAINY (1989), *Classification automatique des données*, Dunod, Paris.

[Cendes *et al.* 1985] Z.J. CENDES, D.N. SHENTON AND H. SHAHNASSER (1985), Magnetic field computations using Delaunay triangulations and complementary finite element methods, *IEEE Trans. Magnetics*, **21**.

[Cendes, Shenton-1985a] Z.J. Cendes, D.N. Shenton (1985), Adaptive mesh refinement in the finite element computation of magnetic fields, *IEEE Trans. Magnetics*, **21**.

[Cendes, Shenton-1985b] Z.J. Cendes, D.N. Shenton (1985), Complementary error bounds for foolproof finite element mesh generation, *Math. and Comp. in Simulation*, **27**, 295-305, North Holland.

[Cendes, Shenton-1985c] Z.J. Cendes, D.N. Shenton (1985), Three-dimensional finite element mesh generation using Delaunay tesselation, *IEEE Trans. Magnetics*, **21**, 2535-2538.

[Chan, Anatasiou-1997] C.T. Chan and K. Anatasiou (1997), An automatic tetrahedral mesh generation scheme by the advancing-front method, *Commun. Numer. Methods Eng.*, **13**, 33-46.

[Chatelin-1993] F. Chatelin(1993), *Eigenvalues of matrices*, Wiley.

[Chazelle-1984] B. Chazelle (1984), Convex partitions of polyhedra: a lower bound and worst-case optimal algorithm, *SIAM J. Comput*, **13**, 488-507.

[Chazelle, Palios-1990] B. Chazelle and L. Palios (1990), Triangulating a nonconvex polytope, *Discrete Comput. Geom.*, **5**, 505-526.

[Chazelle-1991] B. Chazelle (1991), Triangulating a simple polygon in linear time, *Discrete Comput. Geom.*, **6**, 485-524.

[Chen, Schmitt-1992] X. Chen and F. Schmitt (1992), Intrinsic surface properties from surface triangulation, *Rapport de Recherche*, ENST-92-D-004.

[Cherfils, Hermeline-1990] C. Cherfils and F. Hermeline (1990), Diagonal swap procedures and characterizations of 2D-Delaunay triangulations, *Rairo, Math. Mod. and Num. Anal.*, **24**(5), 613-626.

[Chew, Drysdale-1985] L.P. Chew and R.L. Drysdale (1985), Voronoï diagrams based on convex distance functions, *ACM 0-89791-163-6*, 235-244.

[Chew-1989a] L.P. Chew (1989), Constrained Delaunay Triangulations, *Algorithmica*, **4**, 97-108.

[Chew-1989b] L.P. Chew (1989), Guaranteed-Quality Triangular Meshes, *Dept. of Computer Science Tech Report 89-983*, Cornell University.

[Chew-1993] L.P. Chew (1993), Guaranteed-Quality Mesh Generation for Curved Surfaces, *Proc. 9th ACM Computational Geometry*, 274-280.

[Chew *et al.* 1997] L.P. Chew, N. Chrisochoides and F. Sukupp (1997), Parallel constrained Delaunay meshing, *Joint ASME/ASCE/SES Summer Meeting McNu'97*, Northwestern Univ., USA, June 29-July 2, 89-96.

[Chrisochoides, Sukupp-1996] N. Chrisochoides and F. Sukupp (1996), Task parallel implementation of the Bowyer-Watson algorithm, *Proc. 5th Int. Conf. Numerical grid Generation in Computational Field Simulations*, Mississippi State Univ., 773-782.

[Ciarlet-1978] P.G. CIARLET (1978), *The Finite Element Method*, North Holland.

[Ciarlet-1986] P.G. CIARLET (1986), *Elasticité Tridimensionnelle*, RMA 1, Masson, Paris.

[Ciarlet-1988] P.G. CIARLET (1988), *Mathematical Elasticity, Vol 1: Three-Dimensional Elasticity*, North-Holland.

[Ciarlet-1991] P.G. CIARLET (1991), Basic Error Estimates for Elliptic Problems, in *Handbook of Numerical Analysis, vol II, Finite Element Methods* (Part 1), P.G. Ciarlet and J.L. Lions Eds, North Holland, 17-352.

[Ciarlet, Raviart-1973] P.G. CIARLET AND P.A. RAVIART (1973), Maximum principle and uniform convergence for the finite element method, *Computer Methods in Appl. Mechanics and Engineering*, **2**, 17-31.

[Cignoni *et al.* 1996] P. CIGNONI, C. MONTANI, E. PUPPO AND R. SCOPIGNO (1996), Optimal isosurface extraction from irregular volume data, *IEEE/ACM Symp. on Volume Visualization*, San Francisco, CA.

[Clarkson-1987] K. CLARKSON (1987), New applications of random sampling in computational geometry, *Discrete Comput. Geom.*, **2**, 195-222.

[Clarkson, Shor-1989] K. CLARKSON AND P.W. SHOR (1989), Applications of random sampling in computational geometry, *Discrete Comput. Geom.*, **4**, 387-421.

[Clarkson *et al.* 1989] K. CLARKSON, R.E. TARJAN AND C.J. VAN WYK (1989), A fast Las Vegas algorithm for triangulating a simple polygon, *Discrete Comput. Geom.*, **4**, 423-432.

[Cook-1974] W.A. COOK (1974), Body oriented coordinates for generating 3-dimensional meshes, *Int. J. Numer. Methods Eng.*, **8**, 27-43.

[Coorevits *et al.* 1995] P. COOREVITS, P. LADEVÈZE AND J.P. PELLE (1995), Au automatic procedure for finite element analysis in 2d elasticity, *Comput. Methods Appl. Mech. Engrg.*, **121**, 91-120.

[Cormen *et al.* 1990] T. CORMEN, C. LEISERSON AND R. RIVEST (1990), *Introduction to Algorithms,* MIT Press.

[deCougny *et al.* 1990] H. DECOUGNY, M.S. SHEPHARD AND M.K. GEORGES (1990), Explicit Node Point smoothing within Octree, *Scorec Report* **10**, RPI, Troy, NY.

[deCougny, Shephard 1996] H. DECOUGNY AND M.S. SHEPHARD (1996), Surface Meshing Using Vertex Insertion, *Proc. 5th Int. Meshing Roundtable*, 243-256.

[deCougny *et al.* 1996] H. DECOUGNY, M.S. SHEPHARD AND C. ÖZTURAN (1996), Parallel three-dimensional mesh generation on distributed memory MIMD computers, *Engineering with Computers*, **12**, 94-106.

[deCougny-1997] H. DECOUGNY (1997), Distributed parallel mesh generation, *PhD thesis*, Scorec, Rensselaer Polytechnic Inst., Troy, NY.

[deCougny-1998] H. DECOUGNY (1998), Refinement and coarsening of surface meshes, *Engineering with Computers*, **14**, 214-222.

[deCougny, Shephard-1999] H. DECOUGNY AND M.S. SHEPHARD (1999), Parallel refinement and coarsening of tetrahedral meshes, *Int. J. Numer. Methods Eng.*, **46**, 1101-1125.

[Coulomb-1987] J.L. COULOMB (1987), Maillage 2D et 3D. Experimentation de la triangulation de Delaunay, *Conf. on Automated mesh generation and adaptation*, Grenoble.

[Coupez-1991] T. COUPEZ (1991), Grandes transformations et remaillage automatique, Thèse ENSMP, CEMEF, Sophia Antipolis.

[Coupez-1995] T. COUPEZ (1995), Automatic Remeshing in Threedimensional Moving Mesh Finite Element Analysis of Industrial Forming, in *Simulation of Material Processing: Theory, Practice, Methods and Applications*, Balkema, Rotterdam, S.F. Shen and P.R. Dawson (editors), 407-412.

[Coupez-1996] T. COUPEZ (1996), Parallel Adaptive Remeshing in 3D Moving Mesh Finite Element, *5th Int. Conf. on Grid Generation in Comp. Field Simulations*, MS, USA, 783-792.

[Coupez-1997] T. COUPEZ (1997), Mesh generation and adaptive remeshing by a local optimization principle, *Proc. Nafems World Congress'97*, Stuttgart.

[Coupez-2000] COUPEZ T. (2000), Génération de maillage et adaptation de maillage par optimisation locale, *Revue Européenne des éléments finis*, **49**(4), 403-423.

[Coupez *et al.* 2004] T. COUPEZ, C. GRUAU, CH. PEQUET AND J. BRUCHON (2004), Metric map and anisotropic mesh adaptation for static and moving surfaces, *World Congress on Computational Mechanics*, Beijing (China), 5-10.

[Coxeter *et al.* 1959] H.S.M. COXETER, L. FEW AND C.A. ROGERS (1959), Covering space with equal spheres, *Mathematika*, **6**, 147-151.

[Coxeter-1963] H.S.M. COXETER (1963), *Regular polytopes,* Macmillan, New York.

[Crandall-Lions, 1984] M. CRANDALL AND J.L. LIONS (1984), Two approximations of solutions of Hamilton-Jacobi equations, *Math. Comp.*, **43**, 1-19.

[Cullum, Willoughby-1985] J. CULLUM AND R.A. WILLOUGHBY (1985), Lanczos algorithms for large symmetric eigenvalue computations, in *Progress in Scientific Computing*, **3-4**, Biekhauser, London.

[Cuthill-1972] E. CUTHILL (1972), Several strategies for reducing the bandwidth of sparse symmetric matrices, in *Sparse matrices and their applications*, Plenum Press, New York.

[Cuthill, McKee-1969] E. CUTHILL, J. MCKEE (1969), Reducing the bandwidth of sparse symmetric matrices, *Proc. 24th Nat. Conf. Assoc. Comput. Mach.*, 157-172.

[Dahlquist, Bjorck-1974] G. DAHLQUIST AND A. BJORCK (1974), *Numerical Methods,* Prentice-Hall, Englewood Cliffs, N.J.

[Dannelongue, Tanguy-1990] H. DANNELONGUE AND P.A. TANGUY (1990), Efficient data structures for adaptive remeshing with the FEM, *J. Comput. Physics*, **91**, 94-109.

[Dannelongue, Tanguy-1991] H. DANNELONGUE AND P.A. TANGUY (1991), Three-dimensional adaptive finite element computations and applications to non-Newtonian flows, *Int. J. Numer. Methods Fluids*, **13**, 145-165.

[DeBuhan *et al.* 2008] M. DE BUHAN *et al.* (2008), Mesh adaptation for an Arbitrary Lagrangian Eulerian (ALE) method, *ESAIM Proc. Cemracs 2007*, to appear.

[Delaunay-1934] B. DELAUNAY (1934), Sur la sphère vide, *Bul. Acad. Sci. URSS, Class. Sci. Nat.*, 793-800.

[Demengel, Pouget-1998] G. DEMENGEL, J.P. POUGET (1998), Modèles de Bézier, des B-Splines et des NURBS, *Ellipses*, Paris.

[Deriche-1987] R. DERICHE (1987), Using Canny's criteria to derive a recursively implemented optimal edge detector, *Int. J. Comput. Vision*, 167-187.

[Dervieux-Thomasset, 1981] A. DERVIEUX AND F. THOMASSET (1981), Multifluid incompressible flows by a finite element method, *Lecture Notes in Physics*, **11**, 158-163.

[Désidéri-1998] J.A DÉSIDÉRI (1998), *Modèles discrets et schémas itératifs*, Hermès.

[Devillers-1992] O. DEVILLERS (1992), Randomization yields simple $O(nlog^*n)$ algorithms for difficult $\Omega(n)$ problems, *Internat. J. Comput. Geom. Appl.*, **2**(1), 97-111.

[Devillers-1997] O. DEVILLERS (1997), Improved incremental randomized Delaunay triangulation, *RR INRIA*, **3298**.

[Devillers *et al.* 1992] O. DEVILLERS, S. MEISER AND M. TEILLAUD (1992), Fully dynamic Delaunay triangulation in logarithmic expected time per operation, *Comput. Theory Appl.* **2**(2), 55-80.

[Devillers-Preparata 1998] O. DEVILLERS, AND F. PREPARATA (1998), A probalistic analysis of the power of arithmetic filters, *Discrete and Computational geometry.*, 20.

[Devroye-1986] L. DEVROYE (1986), *Lecture Notes on Bucket Algorithms,* Birkauser.

[Dey-1997] S. DEY (1997), Geometry-Based Three-Dimensional *hp*-Finite Element Modelling And Computations, PhD thesis, Scorec, Rensselaer Polytechnic Inst., Troy, NY.

[Dey *et al.* 1997] S. DEY, M.S. SHEPHARD AND J.E. FLAHERTY (1997), Geometry representation issues associated with p-version finite lement computations, *Comp. Meth. Appl. Mech. and Eng.*, **150**(1-4), 39-56.

[Dirichlet-1850] G.L. Dirichlet (1850), Über die reduction der positiven quadratischen formen mit drei understimmten ganzen zahlen, *Z. Angew Math. Mech.*, **40**(3), 209-227.

[Dobrzynski-2005] C. Dobrzynski, (2005), Adaptation de maillage anisotrope 3D et application à l'aéro-thermique des bâtiments, PhD thesis, Université Pierre et Marie Curie, Paris VI.

[Dobrzynski *et al.* 2005] C. Dobrzynski, P. Frey, B. Mohammadi and O. Pironneau, (2005), Fast and accurate simulations of air-cooled structures, *Comp. Meth. Appl. Mech. and Eng.*, **195**(23-24), 3168-3180.

[Doi, Koide-1991] A. Doi and A. Koide (1991), An efficient method of triangulating equi-valued surfaces by using tetrahedral cells, *IEICE Trans.*, **E74**(1), 214-224.

[Dolenc, Makela-1990] A. Dolenc and I. Mäkelä (1990), Optimized triangulation of parametric surfaces, *Mathematics of Surfaces*, **IV**.

[Donaghy *et al.* 1996] R.J. Donaghy, W.M. Cune, S.J. Bridgett, C.G. Armstrong, D.J. Robinson and R.M. Keag (1996), Dimensional reduction of analysis models, *Proc. 5th Int. Meshing Roundtable*, 307-320.

[Donath, Hoffman-1972] W.E. Donath and A.J. Hoffman (1972), Algorithms for Partitioning of Graphs and Computer Logic based on Eigenvectors of Connection matrices, *IBM Technical Disclosure Bulletin*, **15**, 938-944.

[Donath-1973] W.E. Donath (1973), Lower Bounds for the partitioning of Graphs, *IBM J. Res. Develop.*, **17**, 420-425.

[Donea et al. 2004] J. Donea, A. Huerta, J.-P. Ponthot, A. Rodriguez-Ferran (2004), Arbitrary Lagrangian-Eulerian methods, in *Encyclopedia of Computational Mechanics*, E. Stein, R. de Borst and T.J.R. Hughes eds, **1**, 413-437.

[Ducrot, Frey 2007] V. Ducrot, P. Frey (2007), Contrôle de l'approximaion géométrique d'une interface par une métrique anisotrope, *C.R. Acad. Sci. Paris*, t. 345, Série I, 537-542.

[Dürst-1988] M.J. Dürst (1988), Letters: additional reference to "Marching Cubes", *ACM Comp. Graphics*, **22**(4), 72-73.

[Dyn, Goren-1993] N. Dyn and I. Goren (1993), Transforming triangulations in polygonal domains, *Computer Aided Geometric Design*, **10**, 531-536.

[Dwyer-1987] R.A. Dwyer (1987), A faster Divide-and-Conquer algorithm for constructing Delaunay triangulations, *Algorithmica*, **2**, 137-151.

[Dwyer-1991] R.A. Dwyer (1991), Higher-Dimensional Voronoï diagrams in linear expected time, *Discrete Comput. Geom.*, **6**, 342-367.

[Ecer et al. 1985] A. Ecer, J.T. Spyropoulos, J.D. Maul (1985), A three-dimensional block-structured Finite Element Grid generation scheme, *AIAA J.*, **23**, 1483-1490.

[Edelsbrunner-1987] H. EDELSBRUNNER (1987), *Algorithms in Combinatorial Geometry*, **10**, EATCS Monographs on Theoretical Computer Science, Springler-Verlag.

[Edelsbrunner *et al.* 1990] H. EDELSBRUNNER, F.P. PREPARATA AND D.B. WEST (1990), Tetrahedrizing point sets in three dimensions, *J. Symbolic Computation*, **10**, 335-347.

[Eiseman-1979] P.R. EISEMAN (1979), A multi-surface method of coordinate generation, *J. Comput. Phys.*, **33**.

[Eiseman-1985] P.R. EISEMAN (1985), Alternating direction adaptive grid generation, *AIAA J.*, **23**.

[Eiseman, Erlebacher-1987] P.R. EISEMAN AND G. ERLEBACHER, Grid generation for the solution of partial differential equations, *ICASE report 87-57*.

[Eppstein-1992] D. EPPSTEIN (1992), The farthest point Delaunay triangulation minimizes angles, *Comp. Geom. Theory & Applications*, **1**, 143-148.

[Eriksson-1982] L.E. ERIKSSON (1982), Generation of boundary-conforming grids around wing-body configurations using transfinite interpolation, *AIAA J.*, **20**.

[Eriksson-1983] L.E. ERIKSSON (1983), Practical three-dimensional mesh generation using transfinite interpolation, Von Karman Inst. for Fluids Dynamics, Lecture Series Notes.

[Eriksson-1984] L.E. ERIKSSON (1984), Transfinite mesh generation and Computer-Aided-Analysis of mesh effects, PhD thesis, Universitetshuset, Uppsala, Sweden.

[Faddeev *et al.* 1992] D.K. FADDEEV, N.P. DOLBILIN, S.S. RYSHKOV AND M.I. SHTOGRIN (1992), B.N. Delone (on his life and creative work), *Proc. the Steklov Inst. of Math.*, **4**, 1-9.

[Farestam, Simpson-1993] S. FARESTAM AND R.B. SIMPSON (1993), On correctness and efficiency for advancing-front techniques of finite element mesh generation, *RR*, Univ. Waterloo, Canada.

[Farhat, Lesoinne-1993] C. FARHAT AND M. LESOINNE (1993), Automatic partitioning of unstructured meshes for the parallel solution of problems in computational mechanics, *Int. J. Numer. Methods Eng.*, **36**, 745-764.

[Farhat *et al.* 1998] C. FARHAT, C. DEGAND, B. KOOBUS AND M. LESOINNE (1998), Torsional springs for two-dimensional dynamic unstructured fluid meshes, *Comput. Methods Appl. Mech. Eng.*, **163**, 231-45.

[Farin-1983] G. FARIN (1983), Smooth interpolation to scattered 3D data, in R.E. Barnhill and W. Boehm (Eds.) *Surfaces in Computer Aided Geometric Design*, North-Holland, 43-63.

[Farin-1985] G. FARIN (1985), A modified Clough-Tocher interpolant, *Computer Aided Geometric Design*, **2**, 19-27.

[Farin-1986] G. FARIN (1986), Triangular Bernstein-Bézier patches, *Computer Aided Geometric Design*, **3**(2), 83-127.

[Farin-1987] G. FARIN (1987), *Geometric Modeling: Algorithms and new trends,* SIAM.

[Farin-1997] G. FARIN (1997), *Curves and surfaces for CAGD. A practical guide.* Academic Press.

[Faroukin, Rajan-1987] R.T. FAROUKI AND V.T. RAJAN (1987), On the numerical condition of algebraic curves and surfaces. 1 : implicit equations, *IBM Research Report*, **RC 13263**.

[Fezoui *et al.* 1991] L. FEZOUI, F. LORIOT, M. LORIOT AND J. RÉGÈRE J. (1991), A 2D Finite volume/Finite element Euler Solver on a MIMD parallel machine, *Proc. 2nd Symp. on High Performance Computing*, Montpellier, France.

[Field-1988] D.A. FIELD (1988), Laplacian smoothing and Delaunay triangulations, *Commun. Numer. Methods Eng.*, **4**, 709-712.

[Field, Smith-1991] D.A. FIELD AND W.D. SMITH (1991), Graded tetrahedral finite element meshes, *Int. J. Numer. Methods Eng.*, **31**, 413-425.

[Field, Yarnall-1989] D.A. FIELD AND K. YARNALL (1989), Three dimensional Delaunay triangulations on a Cray X-MP, in Supercomputing 88, vol 2, Science et Applications, *IEEE C.S. and ACM Sigarch.*

[Field-1995] D.A. FIELD (1995), The legacy of automatic mesh generation from solid modeling, *Computer Aided Geometric Design*, **12**, 651-673.

[Field-2000] D.A. FIELD (2000), Quality measures for initial meshes, *Int. J. Numer. Methods Eng.*, **47**, 887-906.

[Fielder-1975] M. FIELDER (1975), A Property of Eigenvectors of Nonnegative Symmetric Matrices and its Application to Graph Theory, *Czech. Math.*, **25**, 619-633.

[deFigueiredo *et al.* 1992] L.H. DEFIGUEIREDO, J.DEMIRANDA GOMEZ, D. TERZOPOULOS AND L. VELHO (1992), Physically-based methods for polygonization of implicit surfaces, in *Proc. Graphics Interfaces'92*, 250-257.

[deFigueiredo-1992] L.H. DEFIGUEIREDO (1992), Computational morphology of implicit curves, *PhD thesis*, IMPA, Brazil.

[Filip *et al.* 1986] D. FILIP, R. MAGEDSON AND R. MARKOT (1986), Surface algorithm using bounds on derivatives, *Computer Aided Geometric Design*, **3**, 295-311.

[Filipiak-1996] M. FILIPIAK (1986), *Mesh generation,* Edinburgh Parallel Comp. Center, University of Edinburgh.

[Fiorot, Jeannin-1989] J.C. FIOROT, P. JEANNIN(1989), *Courbes et Surfaces Rationnelles: Applications à la CAO*, RMA 12, Masson, Paris.

[Folwell, Mitchell-1998] N.T. FOLWELL AND S.A. MITCHELL (1998), Reliable Whisker Weaving via curve contraction, *Proc. 7th International Meshing Roundtable*, 365-378.

[Fortin-2000] M. FORTIN (2000), Estimation a posteriori et adaptation de maillages, *Revue Européenne des Eléments Finis*, **9**(4).

[Fortune-1987] S.J. FORTUNE (1987), A sweepline algorithm for Voronoi diagrams, *Algorithmica*, **2**(2), 153-174.

[Fortune-1992] S.J. FORTUNE (1992), Voronoï diagrams and Delaunay triangulations. In D.Z. Du and F.K. Hwand, eds., *Computing in Euclidean Geometry*, vol 1 of *Lecture Notes Series on Computing*, World Scientific, Singapore, 193-233.

[Fortune, Van Wyk-1993] S.J. FORTUNE AND C.J. VAN WYK (1993), Efficient exact arithmetic for computational geometry, *Proc. 9th ACM Sympos. Comput. Geom.*, 163-172.

[Freitag, Gooch-1996] L. FREITAG AND C.O. GOOCH (1996), A comparison of tetrahedral mesh improvement techniques, *Proc. 5th International Meshing Roundtable*, 87-100.

[Freitag, Gooch-1997] L. FREITAG AND C.O. GOOCH (1997), The effect of mesh quality on solution efficiency, *Proc. 6th International Meshing Roundtable*, 249.

[Freudenthal-1942] H. FREUDENTHAL (1942), Simplizialzerlegungen von Beschränker Flachheit, *Annals of Mathematics*, **43**, 580-582.

[Frey-1987] W.H. FREY (1987), Selective refinement procedure: a new strategy for automatic node placement in graded triangular meshes, *Int. J. Numer. methods eng.*, **24**, 2183-2200.

[Frey, Field-1991] W.H. FREY AND D.A. FIELD (1991), Mesh relaxation: a new technique for improving triangulations, *Int. J. Numer. Methods Eng.*,**31**, 1121-1131.

[Frey-1993] P.J. FREY (1993), Génération automatique de maillages 3D dans des ensembles discrets. Application biomédicale aux méthodes d'éléments finis. Thesis,Univeristy of Strasbourg I.

[Frey-2000] P.J. FREY (2000), About surface remeshing, *Proc. 9th International Meshing Roundtable*, 123-136.

[Frey-2004] P.J. FREY (2004), Generation and adaptation of computational surface meshes from discrete anatomical data, *Int. J. Numer. Methods Engng.*, **60**, 1049-1074.

[Frey, Alauzet-2003] P.J. FREY AND F. ALAUZET (2003), Anisotropic Mesh Adaptation for Transient Flows Simulations, *Proc. 12th International Meshing Roundtable.*

[Frey, Alauzet-2005] P.J. FREY AND F. ALAUZET (2005), Anisotropic mesh adaptation for CFD computations, *Comp. Meth. Appl. Mech. and Eng.*, **194**(48-49), 5068-5082.

[Frey *et al.* 1994] P.J. FREY, B. SARTER AND M. GAUTHERIE (1994), Fully automatic mesh generation for 3-D domains based upon voxel sets, *Int. J. Numer. Methods Eng.*, **37**, 2735-2753.

[Frey *et al.* 1996] P.J. FREY, H. BOROUCHAKI AND P.L. GEORGE (1996), Delaunay tetrahedralization using an advancing-front approach, *Proc. 5th International Meshing Roundtable*, 31-43.

[Frey *et al.* 1998] P.J. FREY, H. BOROUCHAKI AND P.L. GEORGE (1998), 3D Delaunay mesh generation coupled with an advancing-front approach, *Comput. Methods Appl. Mech. Engrg.*, **157**, 115-131.

[Frey, Borouchaki-1996] P.J. FREY ET H. BOROUCHAKI (1996), Triangulation des surfaces implicites, *C.R. Acad. Sci. Paris*, t. 325, Serie I, 101-106.

[Frey, Borouchaki-1996] P.J. FREY ET H. BOROUCHAKI (1996), Texel: triangulation des surfaces implicites. Partie I: aspects théoriques, *RR INRIA*, **3066**.

[Frey, Borouchaki-1997] P.J. FREY AND H. BOROUCHAKI (1997), Unit surface mesh simplification, *Joint ASME/ASCE/SES Summer Meeting McNu'97*, Northwestern Univ., USA, June 29-July 2.

[Frey, Borouchaki-1998] P.J. FREY AND H. BOROUCHAKI (1998), Geometric evaluation of finite element surface meshes, *Finite Element in Analysis and Design*, **31**, 33-53.

[Frey, Borouchaki-1998] P.J.FREY AND H. BOROUCHAKI (1998), Geometric surface mesh optimization, *Computing and Visualization in Science*, **1**, 113-121.

[Frey, George-1999] P.J.FREY ET P.L. GEORGE (1999), *Maillages. applications aux éléments finis*, Hermès Science, Paris.

[Frey, George-2003] P.J.FREY ET P.L. GEORGE (2003), *Le maillage facile*, Hermès Science, Paris.

[Frey, Maréchal-1998] P.J. FREY AND L. MARÉCHAL (1998), Fast Adaptive Quadtree Mesh Generation, *Proc. 7th International Meshing Roundtable*, 211-224.

[Frey, Borouchaki-1999] P.J. FREY AND H. BOROUCHAKI(1999), Surface mesh quality evaluation, *Int. J. Numer. Methods Engng.*, **45**, 101-118.

[Frykestig-1994] J. FRYKESTIG (1994), Advancing-front mesh generation techniques with application to the finite element method, *Research Report* **94-10**, Chalmers Univ. of Technology, Gothenburg, Sweden.

[Galtier-1997] J. GALTIER (1997), Structures de données irrégulières et architectures haute performance. Une étude du calcul numérique intensif par le partitionnement de grappes, Thesis, Université de Versailles Saint-Quentin.

[Galtier, George-1996] J. GALTIER AND P.L. GEORGE (1996), Prepartitioning as a way to mesh subdomains in parallel, *Proc. 5th International Meshing Roundtable*, 107-121.

[Garey *et al.* 1978a] M.R. GAREY, D.S. JOHNSON, F.P. PREPARATA AND R.E. TARJAN (1978), Triangulating a simple polygon, *Inform. Proc. Letters*, **7**(4), 175-180.

[Gargantini-1982] I. GARGANTINI (1982) Linear octtrees for fast processing of three-dimensional objects, *CVGIP*, **20**, 365-374.

[Garimella, Shephard-1998] R.V. GARIMELLA AND M.S. SHEPHARD (1998), Boundary layer meshing for viscous flows in complex domains, *Proc. 7th International Meshing Roundtable*, 107-118.

[Gaudel *et al.* 1987] M.C. GAUDEL, M. SORIA ET C. FROIDEVAUX (1987), *Types de Données et Algorithmes : Recherche, Tri, Algorithmes sur les Graphes*, collection didactique INRIA, **4**(2).

[van Gelder, Wilhelms-1992] A. VANGELDER AND J. WILHELMS (1992), Octrees for faster isosurface generation, *Trans. on Graphics*, **11**(3), 201-227.

[A.George-1971] J.A. GEORGE (1971), Computer implementation of the finite element method, PhD thesis, Dept. of Computer Science, Stanford Univ.

[A.George-1973] J.A. GEORGE (1973), Nested dissection of a regular finite element mesh, *SIAM J. Num. Anal.*, **10**, 1345-367.

[A.George, Liu-1978] J.A. GEORGE, J.W. LIU (1978), An automatic nested dissection algorithm for irregular finite element problems, *SIAM J. Num. Anal.*, **15**, 1053-1069.

[A.George, Liu-1979] J.A. GEORGE, J.W. LIU (1979), An implementation of a pseudoperipheral node finder, *ACM Tras. Math. Software*, **5**(3), 284-295.

[A.George, Liu-1981] J.A. GEORGE, J.W. LIU (1981), *Computer solution of large sparse positive definite systems*, Prentice Hall.

[George-1991] P.L. GEORGE (1991), *Automatic mesh generation. Applications to finite element methods*, Wiley.

[George-1993] P.L. GEORGE (1993), Construction et Modification de Maillages, *Guide Module* **3**.

[George-1996a] P.L. GEORGE (1996), Automatic Mesh Generation and Finite Element Computation, in *Handbook of Numerical Analysis*, vol IV, Finite Element methods (Part 2), Numerical Methods for Solids (Part 2), P.G. Ciarlet and J.L. Lions Eds, North Holland, 69-190.

[George-1997] P.L. GEORGE (1997), Improvement on Delaunay based 3D automatic mesh generator, *Finite Elements in Analysis and Design*, **25**(3-4), 297-317.

[George-2001] P.L. GEORGE (2001), *Maillage et adaptation*, in MIM series, Hermès Lavoisier, Paris.

[George *et al.* 1990] P.L. GEORGE, F. HECHT AND E. SALTEL (1990), Fully automatic mesh generator for 3D domains of any shape, *Impact of Comp. in Sci. and Eng.*, **2**, 187-218.

[George *et al.* 1991a] P.L. GEORGE, F. HECHT AND E. SALTEL (1991), Automatic mesh generator with specified boundary, *Comput. Methods Appl. Mech. Engrg.*, **92**, 269-288.

[George *et al.* 1991b] P.L. GEORGE, F. HECHT AND M. G. VALLET (1991), Création of internal points in Voronoi's type method, Control and adaptation, *Adv. in Eng. Soft.*, **13**(5/6), 303-313.

[George, Hermeline-1992] P.L. GEORGE AND F. HERMELINE (1992), Delaunay's mesh of a convex polyhedron in dimension d. Application to arbitrary polyhedra, *Int. J. Numer. Methods Eng.*, **33**, 975-995.

[George, Seveno-1994] P.L. GEORGE AND E. SÉVENO (1994), The advancing-front mesh generation method revisited, *Int. J. Numer. Methods Eng.*, **37**, 3605-3619.

[George, Borouchaki-1997] P.L. GEORGE AND H. BOROUCHAKI (1998), *Delaunay triangulation and meshing. Applications to Finite Elements*, Hermès, Paris.

[George, Borouchaki-1998] P.L. GEORGE ET H. BOROUCHAKI (1998), Génération automatique de maillages tridimensionnels respectant une carte de taille, *Revue européenne des éléments finis* **7**(4), 339-363.

[George, Borouchaki-2002] P.L. GEORGE AND H. BOROUCHAKI (2002), "Ultimate" robustness in meshing an arbitrary polyhedron, *Int. J. Numer. Methods Eng.*, **58**(7), 1061-1089.

[Gervasio *et al.* 1997] P. GERVASIO, E. OVTCHINNIKOV AND A. QUARTERONI (1997), The Spectral Projection Decomposition Method for Elliptic Equations in Two Dimensions, *SIAM Journal on Numerical Analysis*, **34**(4), 1616-1639.

[Gibbs *et al.* 1976a] N.E. GIBBS, W.G. POOLE, P.K. STOCKMEYER (1976), An algorithm for reducing the bandwidth and profile of a sparse matrix, *SIAM J. Num. Anal.*, **13**(2), 236-250.

[Gibbs *et al.* 1976b] N.E. GIBBS, W.G. POOLE, P.K. STOCKMEYER (1976), A comparison of several bandwidth and profile reduction algorithm, *ACM Trans. on Math. Software*, **2**, 322-330.

[Giraud-Moreau *et al.* 2005] L. GIRAUD-MOREAU, H. BOROUCHAKI AND A. CHEROUAT (2005), A remeshing procedure for numerical simulation of forming processes in three dimensions, *Proc. 15th Int. Meshing Roundtable*, 127-144.

[Glowinski-1973] R. GLOWINSKI (1973), Approximations externes par éléments finis de lagrange d'ordre un et deux du problème de Dirichlet pour l'opérateur biharmonique. Méthode itérative de résolution des problèmes approchés, *Topics in numerical analysis*, J.J.H. Miller ed., Academic Press, 123-171.

[Goldberg-1991] D. GOLDBERG (1991), What every computer scientist should know about floating-point arithmetic, *ACM Computing surveys*, **23**(1).

[Golgolab-1989] A. GOLGOLAB (1989), Mailleur tridimensionnel automatique pour des géométries complexes, *RR INRIA* **1004**.

[Golub, VanLoan-1983] G.H. GOLUB AND C.F. VAN LOAN (1983), *Matrix computations*, John Hopkins Univ. Press.

[Gonnet *et al.* 1991] G. GONNET AND R. BAEZA-YATES (1991), *Handbook of Algorithms and Data-Structures*, Addison-Wesley.

[Gordon, Hall-1973] W.J. GORDON, C.A. HALL (1973), Construction of curvilinear coordinate systems and applications to mesh generation, *Int. J. Numer. Methods Eng.*, **7**, 461-477.

[Green, Sibson-1978] P. GREEN AND R. SIBSON (1978), Computing Dirichlet tesselations in the plane, *Comput. J.*, **21**, 168-173.

[Gregory-1974] J.A. GREGORY (1974), Smooth interpolation without twist constraints, in R.E. Barnhill and R.F. Riesenfed (Eds.), *Computer Aided Geometric Design*, Academic Press, 71-87.

[Grice *et al.* 1988] K.R. GRICE, M.S. SHEPHARD, C.M. GRAICHEN (1988), Automatic, topologically correct, three-dimensional mesh generation by the finite octree technique, *Technical Report*, RPI Center for Interactive Computer Graphics.

[Grooms-1972] H.R. GROOMS (1972), Algorithm for matrix bandwidth reduction, *J. of Structural division*, **98**, no. ST1, 203-214.

[Grunbaum-1967] B. GRUNBAUM (1967), *Convex Polytopes*, Wiley, New York.

[Guéziec, Hummel-1995] A. GUÉZIEC AND R. HUMMEL 1995), Exploiting triangulated surface extraction using tetrahedral decomposition, *IEEE Trans. on Visualization and Comput. Graphics*, 1(4), 328-342.

[Guibas *et al.* 1989] L.J. GUIBAS, D. SALESIN AND J. STOLFI (1989), Epsilon geometry: building robust algorithm from imprecise computations, *Proc. 5th ACM Sympos. Comput. Geom.*, 208-217.

[Guibas *et al.* 1992] L.J. GUIBAS, D.E. KNUTH AND M. SHARIR (1992), Randomized incremental construction of Delaunay and Voronoï diagrams, *Algorithmica*, **7**, 381-413.

[Hackbush, Trottenberg-1982] W. HACKBUSCH, U. TROTTENBERG (1982), Multigrid methods, *Lect. Notes in Mathematics* **960**, Springer Verlag.

[Hacon, Tomei-1989] D. HACON, C. TOMEI (1989), Tetrahedral decompositions of hexahedral meshes, *Europ. J. Combinatorics*, **10**, 435-443.

[Hall-1976] C.A. HALL (1976), Transfinite interpolation and applications to engineering problems, *Law and Sahney, Theory of approximation*, Academic Press, 308-331.

[Hamann-1993] B. HAMANN (1993), Curvature approximation for triangulated surfaces, in *Geometric Modelling*, Computing Suppl. 8, Farin, Hagen, Noltmeier and Knodel eds., Springer, NY.

[Hansbo-1995] P. HANSBO (1995), Generalized Laplacian smoothing of unstructured grids, *Comm. Numer. Methods Eng.*, **11**, 455-464.

[Hansen *et al.* 2005] G Hansen, A. Zardecki, D. Greening and R. Bos (2005), A finite element method for three-dimensional unstructured grid smoothing, *J. Comp. Phys.*, **202**(1), 281-297.

[Hart-1993] J. Hart (1993), Sphere tracking: a geometric method for the antialiased ray tracing of implicit surfaces, *Research Report*, EECS-93-015, Washington State Univ.

[Hartmann-1990] E. Hartmann (1990), Blending of implicit surfaces with functional splines, *Comput. Aided Design*, **22**, 500-506.

[Hartmann-1998a] E. Hartmann (1998), A marching method for the triangulation of surfaces, *The Visual Computer*, **14**, 95-108.

[Hartmann-1998b] E. Hartmann (1998), Numerical implicitization for interpolation and G^n-continuous blending of surfaces, *Computer Aided Geometric Design*, **15**, 377-397.

[Hassan *et al.* 1996] O. Hassan, K. Morgan, E.J. Probert and J. Peraire (1996), Unstructured tetrahedral mesh generation for three-dimensional viscous flows, *Int. J. Numer. Methods Eng.*, **39**, 549-567.

[Hauser, Taylor-1986] J. Hauser, C. Taylor (1986), *Numerical grid generation in computational fluid dynamics*, Pinebridge Press.

[Hecht, Saltel-1989] F. Hecht et E. Saltel (1990), Emc2 : Un logiciel d'édition de maillages et de contours bidimensionnels, *RT INRIA* **118**.

[Helenbrook-2003] B.T. Helenbrook (2003), Mesh deformation using the biharmonic operator, *Int. J. Numer. Methods Eng.*, **56**(7), 1007-1021.

[Hermeline-1980] F. Hermeline (1980), Une méthode automatique de maillage en dimension n, Thesis, Université Paris VI, Paris.

[Hermeline-1982] F. Hermeline (1982), Triangulation automatique d'un polyèdre en dimension N, *Rairo, Analyse numérique*, **16**(3), 211-242.

[Hirt *et al.* 1974] C.W. Hirt, A.A. Amsden and J.L. Cook (1975), An arbitrary Lagrangian-Eulerian computing method for all flow speeds, *J. Comp. Physics* **14**, 227-253.

[Hirt-Nichols, 1981] C.W. Hirt and B.D. Nichols (1981), Volume of fluid methods for the dynamics of free boundaries, *J. Comp. Phys.*, **30**, 201-255.

[Ho Le-1988] K. Ho Le (1988), Finite element mesh generation methods: a review and classification, *Comp. Aided Design*, **20**, 27-38.

[Hoffmann-1993] c. M. Hoffmann, Implicit curves and surfaces in CAGD, *IEEE Comp. Graphics Appl.*, **13**, 79-88.

[Hoit, Garcelon-1989] M. Hoit and J.H. Garcelon (1989), Updated profile front minimization algorithm, *Computers & Structures*, **33**(3), 903-914.

[Holmes, Snyder-1988] D.G. Holmes and D.D. Snyder (1988), The generation of unstructured triangular meshes using Delaunay triangulation, *Numerical grid generation in computational fluid mechanics'88*, Miami, 643-652.

[Hoppe *et al.* 1991] H. HOPPE, T. DEROSE, T. DUCHAMP, J. MCDONALD AND W. STUETZLE (1991), Surface reconstruction from unorganized points, *technical report 91-12-03*, Dept. of Comp. Science, U. of Washington.

[Hoppe-1994] H. HOPPE (1994), Surface reconstruction from unorganized points, *PhD thesis*, Dept. of Computer Science and Engineering, University of Washington.

[Hosaka-1992] M. HOSAKA (1992), *Modeling of Curves and Surfaces in CAD/CAM*, Springer-Verlag.

[Hughes-1988] T.J.R. HUGHES (1988), *The Finite Element Method: linear static and dynamic finite element analysis*, Prentice-Hall Inc, NJ.

[Ibaroudene, Acharya-1991] D. IBAROUDENE AND R. ACHARYA (1991), Coordinate relationships between vertices of linear octree nodes and corners of the universe, *Comput. and Graphics*, **15**(3), 375-381, 1991.

[Jacquotte-1992] O.P. JACQUOTTE AND G. COUSSEMENT (1992), Structured mesh adaptation: space accuracy and interpolation methods, *Comput. Methods Appl. Mech. Engrg.*, **101**, 397-432.

[Jameson *et al.* 1986] A. JAMESON, T.J. BAKER AND N.P. WEATHERILL (1986), Calculation of inviscid transonic flow over a complete aircraft, *Proc AIAA 24th Aerospace Meeting*, Reno.

[Jasak, Tukoví 2007] H.JASAK AND Z. TUKOVÍ (2007), Automatic Mesh Motion for the Unstructured Finite Volume Method, *Trans. of FAMENA*, **30** (2).

[Jin, Tanner-1993] H. Jin and R.I. Tanner (1993), Generation of unstructured tetrahedral meshes by advancing-front technique, *Int. J. Numer. Methods Eng.*, **36**, 1805-1823.

[Joe-1989] B. JOE (1989), Three-dimensionnal triangulations from local transformations, *SIAM J. Sci. Stat. Comput.*, **10**(4), 718-741.

[Joe-1991] B. JOE (1991), Construction of three-dimensionnal Delaunay triangulations using local transformations, *Comput. Aided Geom. Design*, **8**, 123-142.

[Joe-1991a] B. JOE (1991), Delaunay versus max-min solid angle triangulations for three-dimensionnal mesh generation, *Int. J. Numer. Methods Eng.*, **31**(5), 987-997.

[Joe-1995] B. JOE (1995), Quadrilateral mesh generation in polygonal regions, *Computer Aided Design*, **27**(3), 209-222.

[Johnson-1995] A. A. JOHNSON (1995), Mesh Generation and Update Strategies for Parallel Computation of Flow Problems with Moving Boundaries and Interfaces, Thesis lecture, U. of Minnesota at Minneapolis.

[Johnson, Hansbo-1992] C. JOHNSON AND P. HANSBO (1992), Adaptive finite element methods in computational mechanics, *Comput. Methods Appl. Mech. Engrg.*, **101**, 143-181.

[Johnston *at al.* 1991] B.P. JOHNSTON, J.M. SULLIVAN AND A. KWASNIK (1991), Automatic conversion of triangular finite element meshes to quadrilateral elements, *Int. J. Numer. Methods Eng.*, **31**, 67-84.

[Johnston, Sullivan-1993] B.P. JOHNSTON AND J.M. SULLIVAN (1993), A normal offsetting technique for automatic mesh generation in three dimensions, *Int. J. Numer. Methods Eng.*, **36**, 1717-1734.

[Jones, Plassmann-1997] M.T. JONES AND P.E. PLASSMANN (1997), Adaptive refinement of unstructured finite-element meshes, *Finite Element in Analysis and Design*, **25**(1-2) 41-60.

[Jung, Lee-1991] Y.H. JUNG AND K. LEE(1991), Tetrahedron-based octree encoding for automatic mesh generation, *Technical Report*, Seoul National Univ.

[Kahan-1996] W. KAHAN (1996), Lecture notes on the status of IEEE standard 754 for binary floating-point arithmetic, University of Berkeley, 1996.

[Kallinderis *et al.* 1995] Y. KALLINDERIS, A. KHAWAJA AND H. MCMORRIS (1995), Hybri prismatic/tetrahedral grid generation for complex geometries, *AIAA paper* 95-0211.

[Kalvin, Taylor-1996] A.D. KALVIN AND R.H. TAYLOR (1996), Superfaces: polygonal mesh simplification with bounded error, *IEEE Comp. Graphics and App.*, 64-77.

[Karron-1992] D. KARRON(1992), The "SpiderWeb" surface construction algorithm for building triangle mesh surfaces in noisy volume data, *Proc. 14th Annual Int. Conf. of the IEEE Engineering in Medicine and Biology Soc.*, Paris.

[Kaufmann, Nielson-1993] A.E. KAUFMANN AND G.M. NIELSON (1993), Tutorial on volume visualization techniques and applications, *Computer Graphics Int. 93*, École Polytechnique fédérale de Lausanne.

[Kela *et al.* 1986] A. KELA, R. PERUCCHIO AND H. VOELCKER (1986), Toward automatic finite element analysis, *Comp. Mech. Eng.*, 57-71.

[Kela-1989] A. KELA (1989), Hierarchical octree approximations for boundary representation-based geometric models, *Comp. Aided Des.*, **21**, 355-362.

[Kettner-1998] L. KETTNER (1998), Designing a data structure for polyhedral surfaces. *ACM Symposium on Computational Geometry*, Minneapolis.

[Khawaja *et al.* 1995] A. KHAWAJA, H. MCMORRIS AND Y. KALLINDERIS (1995), Hybrid grids for viscous flows around complex 3D geometries including multiple bodies, *Proc. 12th AIAA CFD Conf.*, AIAA paper-1685.

[Klee-1966] V. KLEE (1966), Convexe polytopes and linear programming, *Proc. IBM Sci. Comput. Symp: Combinatorial Problems*, 123-158.

[Klee-1980] V. KLEE (1980), On the complexity of d-dimensional Voronoï diagrams, *Archiv der Mathematik* **34**, 75-80.

[Klein-1989] R. KLEIN (1989), Concrete and Abstract Voronoï diagrams, *Lecture Notes in Computer Science*, **400**, Springer Verlag.

[Klein *et al.* 1993] R. KLEIN, K. MEHLHORN AND S. MEISER (1993), Randomized incremental construction of abstract Voronoi diagrams, *Comput. Geom. Theory Appl.*, **3**(3), 157-184.

[Knupp, Steinberg-1993] P. KNUPPE AND S. STEINBERG (1993), *The fundamentals of grid generation*, CRC press.

[Knupp-2001] P.M. KNUPP (2001), Algebraic mesh quality metrics, *SIAM J. Sci. Comput.*, **23**, 193-218.

[Knupp-2007] P. KNUPP (2007), Updating meshes on deforming meshes: an application of the target-matrix paradigm, *Comm. Numer. Methods in Engng.*, in press.

[Knuth-1975] D.E. KNUTH (1975), The Art of Computer Programming, 2nd ed., *Addison-Wesley*, Reading, Mass.

[Knuth-1998a] D.E. KNUTH (1998), The Art of Computer Programming, Vol I: Fundamental algorithms, *Addison-Wesley*, Reading, Mass.

[Knuth-1998b] D.E. KNUTH (1998), The Art of Computer Programming, Vol III: Sorting and Searching, *Addison-Wesley*, Reading, Mass.

[Kohli, Carey-1993] H.S. KOHLI AND G.F. GAREY (1993), Shape optimization using adaptive shape refinement, *Int. J. Numer. Methods Eng.*, **36**, 2435-2451.

[Krause, Rank-1996] R. KRAUSE AND E. RANK (1996), A Fast Algorithm for Point-Location in a Finite Element Mesh, *Computing*, **57**, 49-62.

[Kreiner, Kröplin-1994] R. KREINER AND B. KRÖPLIN (1994), Unstructured quadrilateral mesh generation on surfaces from CAD, *Proc. Fourth Int. Conf. Numerical Grid Generation*, Swansea, UK.

[Kriegman, Ponce-1990] D.J. KRIEGMAN AND J. PONCE (1990), On recognizing and positioning curved 3D objects from image contours, *IEEE Trans. on Pattern Analysis and Machine Intelligence*, **12**, 1127-1137.

[Krysl-1996] P. KRYSL (1996), Computational Complexity of the Advancing Front Triangulation, *Engineering with Computers*, **12**, 16-22.

[Kwak *et al.* 2002] D.Y. KWAK, J.S. CHEON AND Y.T. IM (2002), Remeshing for Metal Forming Simulations. Part I: Twodimensional Quadrilateral Remeshing, *Int. J. Numer. Methods Eng.*, **53**, 2463-2500.

[Kwok *et al.* 1995] W. KWOK, K. HAGHIGHI AND E. KANG, An efficient data structure for the advancing-front triangular mesh generation technique, *Comm. Numer. Methods Eng.*, **11**, 465-473.

[Labbé *et al.* 1999] P. LABBÉ, J. DOMPIERRE, F. GUIBAULT AND R. CAMARERO (1999), On element shape measures for mesh optimization, *2nd Symposium on Trends in Unstructured Mesh Generation.*

[Labelle, Shewchuk-2003] F. LABELLE AND J.R. SHEWCHUK (2003), Anisotropic Voronoi Diagrams and Guaranteed-Quality Anisotropic Mesh Generation, in *SoCG'03.*

[Ladevèze *et al.* 1991] P. LADEVÈZE, J.P. PELLE AND P. ROUGEOT (1991), Error estimates and mesh optimization for finite element computation, *Eng. Comput.*, **8**, 69-80.

[Lai-1998] Y.C. LAI (1998), A three-step renumbering procedure for high-order finite element analysis, *Int. J. Numer. Methods Eng.*, **41**, 127-135.

[Lantéri, Loriot-1996] S. LANTERI ET M. LORIOT, Large-scale solutions of Three-dimensional compressible flows using the parallel N3S-MUSCL solver, *Concurrency: Pract. Exp.*, **8**(10), 769-798.

[Laug, Borouchaki-1996] P. LAUG AND H. BOROUCHAKI (1996), The BL2D Mesh Generator, Beginner's Guide, User's and Programmer's Manual, *RT INRIA* **0194** (**0185** in French).

[Laug *et al.* 1996] P. LAUG, H. BOROUCHAKI ET P.L. GEORGE (1996), Maillage de courbes gouverné par une carte de métriques, *RR INRIA* **2818**.

[Laug, Borouchaki-1999] P. LAUG AND H. BOROUCHAKI (1999), BLSURF, Mesh Generator for Composite Parametric Surfaces. User's Manual, *RT INRIA* **0235**.

[Laug, Borouchaki-2002] P. LAUG AND H. BOROUCHAKI (2002), Molecular Surface Modeling and Meshing, *Eng. Comput.*, **18**(3), 199-210.

[Laug, Borouchaki-2003a] P. LAUG AND H. BOROUCHAKI (2003), Interpolating and Meshing 3-D Surface Grids, *Int. J. Numer. Methods Eng.*, **58**(2), 209-225.

[Laug, Borouchaki-2003b] P. LAUG AND H. BOROUCHAKI (2003), Generation of Finite Element Meshes on Molecular Surfaces, *Int. J. Numer. Methods Eng.*, **93**(2), 131-138.

[Lawson-1972] C.L. LAWSON (1972), Generation of a triangular grid with application to contour plotting, California Institute of Technology, JPL, 299.

[Lawson-1972] C.L. LAWSON (1972), Transforming triangulations, *Discrete Mathematics*, **3**, 365-372.

[Lawson-1977] C.L. LAWSON (1977), *Software for C^1 surface interpolation,* Math. Soft., 3, J. Rice ed., Academic Press, New York.

[Lawson-1986] C.L. LAWSON (1986), Properties of n-dimensional triangulations, *Computer Aided Geometric Design*, **3**, 231-246.

[Lee-1999] C.K. LEE (1999), Automatic adaptive mesh generation using metric advancing-front approach, *Engineering Computations*, **16**(2), 230-263.

[Lee-2000] C.K. LEE (2000), Automatic metric advancing-front triangulation over curved surfaces, *Engineering Computations*, **17**(1), 48-74.

[Lee-1980] D.T. LEE (1980), Two-dimensional Voronoï diagrams in Lp-Metric, *J. of the ACM*, **27**(4), 604-618.

[Lee, Schachter-1980] D.T. LEE AND B.J. SCHACHTER (1980), Two algorithms for constructing a Delaunay Triangulation, *Int. J. Comp. Inf. Sci.*, **9**(3), 219-242.

[Lee, Lo-1994] C.K. LEE AND S.H. LO (1994), A new scheme for the generation of a graded quadrilateral mesh, *Comp. Struct.*, **52**, 847-857.

[Lee, Wong-1980] D.T. LEE AND C.K. WONG (1980), Voronoi diagrams in L_1 (L_∞) metrics with 2-dimensional storage applications, *SIAM J. Comput.*, **9**, 200-211.

[Lee, Lin-1988] D.T. LEE AND A.K. LIN (1988), Generalized Delaunay triangulation for planar graphs, *Discrete Comput. Geom.*, **1**, 201-217.

[Leibon, Letscher-2000] G. LEIBON AND D. LETSCHER (2000), Delaunay triangulation and Voronoi Diagrams for Riemannian Manifolds, in 16th *Annual Symposium on CG*, 341-349.

[Lelong-Ferrand, Arnaudies-1977] J. LELONG-FERRAND AND J.M. ARNAUDIÈS (1977), *Cours de Mathématiques*, Tome 3, *Géométrie et cinématique*, Dunod Université, Bordas.

[Lennes-1911] N.J. LENNES (1911), Theorems on the simple finite polygon and polyhedron, *Am. J. Math.*, **33**, 37-62.

[Léon-1991] J.C. LÉON (1991), *Modélisation et construction des surfaces pour la CFAO*, Éditions Hermès, Paris.

[Lewis-1982] J.G. LEWIS (1982), Implementation of the Gibss-Poole-Stockmeyer and Gibbs-King algorithms, *ACM Trans. Math. Software*, **8**(2), 180-189.

[Lewis *et al.* 1996] R.W. LEWIS, Y. ZHENG AND D.T. GETHIN (1996), Three-dimensional unstructured mesh generation: Part 3. Volume meshes, *Comput. Methods Appl. Mech. Engrg.*, **134**, 285-310.

[Lewis *et al.* 1995] R.W. LEWIS, Y. ZHENG AND A.S. USMANI (1995), Aspects of adaptive mesh generation based on domain decomposition and Delaunay triangulation., *Finite Elements in Anal. and Design.*, **20**, 47-70.

[Li *et al.* 1997] T.S. LI, C.G. ARMSTRONG AND R.M. MCKEAG (1997), Quad mesh generation for k-sided faces and hex mesh generation for trivalent polyhedra, *Finite Elements in Analysis and Design*, **26**(4), 279-301.

[Liao, Anderson 1991] G. LIAO AND D.A. ANDERSON (1991), A New Approach to Grid Generation, *Applicable Analysis*, 44.

[Liseikin-2000] V.D. LISEIKIN (2000), Grid Generation Methods *Scientific Computation*, Springer.

[Liu, Joe-1994] A. LIU AND B. JOE (1994), On the shape of tetrahedra from bisection, *Mathematics of Computation*, **63**(207), 141-154.

[Lo-1985] S.H. LO (1985), A new mesh generation scheme for arbitrary planar domains, *Int. J. Numer. Methods Eng.*, **21**, 1403-1426.

[Lo-1989] S.H. LO (1989), Delaunay triangulation of non-convex planar domains, *Int. J. Numer. Methods Eng.*, **28**, 2695-2707.

[Lo-1989] S.H. LO (1989), Generating quadrilateral elements on plane and over curved surfaces, *Comp. Struct.*, **31**, 421-426.

[Löhner-1988] R. LÖHNER (1988), Some useful Data Structures for the Generation of unstructured grids, *Commun. Numer. Methods Eng.*, **4**, 123-135.

[Löhner-1989] R. LÖHNER (1989), Adaptive remeshing for transient problems, *Comput. Methods Appl. Mech. Engrg.*, **75**, 195-214.

[Löhner-1993] R. LÖHNER (1993), Matching semi-structured and unstructured grids for Navier-Stokes calculations, *AIAA paper 93-3348*, july.

[Löhner-1995] R. LÖHNER (1995), Surface gridding from discrete data, *Proc. 4th International Meshing Roundtable*, 29-45.

[Löhner-1996a] R. LÖHNER (1996), Regridding Surface Triangulations, *Jour. of Comput. Phys.*, **126**, 1-10.

[Löhner-1996b] R. LÖHNER (1996), Extensions and improvements of the advancing-front grid generation technique, *Commun. Numer. Methods Eng.*, **12**, 683-702.

[Löhner-1996c] R. LÖHNER (1996), Progress in grid generation via the advancing-front technique, *Engineering with Computers*, **12**, 186-210.

[Löhner-1997] R. LÖHNER (1997), Automatic Unstructured Grid Generators, *Finite Elements in Analysis and Design*, **25**(3-4), 111-134.

[Löhner-1998] R. LÖHNER (1998), Renumbering strategies for unstructured-grid solvers operating on shared memory, cache-based parallel machines, *Comput. Methods Appl. Mech. Engrg.*, **163**(1-4), 95-110.

[Löhner-2000] R. LÖHNER (2000), A Parallel Advancing Front Grid Generation Scheme, *AIAA*-00-1005.

[Löhner, Parikh-1988] R. LÖHNER AND P. PARIKH (1988), Three-Dimensional Grid Generation by the Advancing Front Method, *Int. J. Numer. Methods Fluids*, **8**, 1135-1149.

[Löhner et al.1992] R. LÖHNER, J. CAMBEROS AND M. MERRIAM (1992), Parallel Unstructured Grid Generation, *Comput. Methods Appl. Mech. Engrg.*, **95**, 343-357.

[Löhner *et al.* 1992] R. LÖHNER, J. CAMBEROS AND M. MERRIAM (1992), Parallel Unstructured Grid Generation, *Comput. Methods Appl. Mech. Engrg.*, **95**, 343-357.

[Löhner, Ramamurti-1993] R. LÖHNER AND R. RAMAMURTI (1993), A parallelizable load balancing algorithm, *Proc. of the AIAA 31st Aerospace Sciences Meeting and Exhibit.*

[Löhner, Yang 1996] R. Löhner and C. Yang (1996), Improved ALE mesh velocities for moving bodies, *Commun. in Numer. Methods Eng.*, **12**, 599-608.

[Lorensen, Cline-1987] W.E. LORENSEN AND H.E. CLINE (1987), Marching cubes: a high resolution 3D surface construction algorithm, *Comput. Graphics*, **21**(4), 163-169.

[Lu *et al.* 1999] Y. LU, G. RAJIT AND T.J. TAUTGES (1999), Volume decomposition and feature recognition for hexahedral mesh generation, *Proc 8th Int. Meshing Roundtable*, 269-280.

[Mackerle-1993] J. MACKERLE (1993), Mesh generation and refinement for FEM and BEM - a bibliography, *Finite Elem. Anal. Design*, 177-188.

[McMorris, Kallinderis-1997] H. MACMORRIS AND Y. KALLINDERIS (1997), Octree-advancing front method for generation of unstructured surface and volume meshes, *AIAA Journal*, **35**(6), 976-984.

[Mahmoud-1992] H. MAHMOUD (1992), *Evolution of random search trees*, Wiley, New York.

[Mäntylä-1988] M. MÄNTYLÄ (1988), An Introduction to Solid Modeling. Rockville, MD.: Computer Science Press.

[Manzi *et al.* 2000] C. MANZI, F. RAPETTI AND L. FORMAGGIA (2000), Function approximation on triangular grid: some numerical results using adaptive techniques, *Appl. Numer. Math.*, **32**(4), 389-399.

[Marchant et al.1997] M.J. MARCHANT, N.P. WEATHERILL AND O. HASSAN (1997), The adaptation of unstructured grids for transonic viscous flow simulation, *Finite Elements in Analysis and Design*, **25**(3-4), 199-217.

[Marcum-1996] D.L. MARCUM (1996), Unstructured grid generation components for complete systems, *5th Int. Conf. on Grid Generation in Comp. Field Simulations*, MS, USA, 1-5 April.

[Marcum, Weatherill-1995] D.L. MARCUM AND N.P. WEATHERILL (1995), Unstructured grid generation using iterative point insertion and local reconnection, *AIAA Journal.*, **33**(9), 1619-1625.

[Marechal-2001] L. MARÉCHAL (2001) A new approach to octree hexahedral meshing, *Proc. 10th International Meshing Roundtable*, 209-221.

[Masud, Hughes 1997] A. MASUD AND T.J.R. HUGHES (1997), A space-time Galerkin/least-squares finite element formulation of the Navier-Stokes equations for moving domain problems, *Comput. Methods Appl. Mech. Engrg.*, **146**, 91-126.

[Matveyev-1994] S.V. MATVEYEV (1994), Approximation of isosurface in the Marching Cube: ambiguity problem, *Visualization'94*, Conf. Proc., IEEE Computer Society Press, 288-292.

[Mavriplis-1990] D.J. MAVRIPLIS (1990), Adaptive mesh generation for viscous flows using Delaunay triangulation, *J. Comput. Physics*, **90**(2), 271-291.

[Mavriplis-1992] D.J. MAVRIPLIS (1992), An advancing front Delaunay triangulation algorithm designed for robustness, *ICASE report 92-49.*

[Mavriplis-1995] D.J. MAVRIPLIS (1995), Unstructured mesh generation and adaptivity, *ICASE report 95-26*

[Meagher-1982] D. MEAGHER (1982), Geometric modeling using octree encoding, *Comput. Graphics and Image Proc.*, **19**, 129-147.

[Mehlhorn *et al.* 1991] D. MEHLHORN, S. MEISER AND C. O'DUNLAING (1991), On the construction of abstract Voronoi diagrams, *Discrete Comput. Geom.*, **6**, 211-224.

[Melhem-1987] R. MELHEM (1987), Towards efficient implementation of preconditioned conjugate gradient methods on vector supercomputers, *Int. Jour. of Supercomp. Appl.*, **1**.

[Merriam-1991] M.L. MERRIAM (1991), An efficient advancing front algorithm for Delaunay triangulation., *AIAA 91-0792*.

[Meshkat, Talmor-1999] S. MESHKAT AND D. TALMOR (1999), Generating a mixed mesh of hexahedra, pentahedra and tetrahedra from an underlying tetrahedral mesh, *2nd Symposium on Trends in Unstructured Mesh Generation.*

[Miller *et al.* 1991] J.V. MILLER, D.E. BREN, W.E. LORENSEN, R.B. O'BARA AND M.J. WOZNY (1991), Geometrically deformed models: a method for extracting closed geometric models from volume data, *Computer Graphics*, **25**(4), 217-225.

[Miller *et al.* 1997] G.L. MILLER, D. TALMOR AND S.H. TENG (1997), Optimal coarsening of unstructured meshes, *Proc. 8th ACM-SIAM Symp. Disc. Algorithms.*

[Mitchell, Vavasis-1992] S. MITCHELL AND S. VAVASIS (1992), Quality Mesh Generation in Higher Dimensions, *Proc. 8th ACM Symp. Comp. Geometry*, 212-221.

[Mitty *et al.* 1993] T.J. MITTY, A. JAMESON AND T. J. BAKER (1993), Solution of three-dimensionnal supersonic flowfields via adapting unstructured meshes, *Computer Fluids*, **22**(2/3), 271-283.

[Mohammadi-1994] B. MOHAMMADI (1994), Fluid dynamics computation with NSC2KE - a User-Guide, Release 1.0, *RT INRIA* **164**.

[Mohammadi, Pironneau-1994] B. MOHAMMADI AND O. PIRONNEAU (1994), *Analysis of the K-Epsilon Turbulence Model*, Wiley and Masson (Eds.).

[Möller, Hansbo-1995] P. MÖLLER AND P. HANSBO (1995), On advancing-front mesh generation in three dimensions, *Int. J. Numer. Methods Eng.*, **38**, 3551-3569.

[Monga, Benayoun-1995] O. MONGA AND S. BENAYOUN (1995), Using partial derivatives of 3D images to extract typical surface features, *Comput. Vision and Image Understanding*, **61**(2), 171-189.

[Montani *et al.* 1994] C. MONTANI, R. SCATENI AND R. SCOPIGNO (1994), A modified look-up table for implicit disambiguation of Marching Cubes, *The Visual Computer*, **10**.

[Moreton-1992] H.P. MORETON (1992), Minimum curvature variation curves, networks and surfaces for fair free-form shape design, PhD thesis, Dept. of Computer science, Univeristy of California, Berkeley.

[Mortenson-1985] M.E. MORTENSON (1985), *Geometric modeling*, Wiley.

[Muller *et al.* 1992] J.D. MULLER, P.L. ROE AND H. DECONINCK (1992), A frontal approach for node generation in Delaunay triangulations., *Unstructured grid methods for advection dominated flows.*, VKI Lecture notes, 91-97, AGARD Publication R-787.

[Muller, Stark-1993] H. MULLER AND M. STARK (1993), Adaptive generation of surfaces in volume data, *The Visual Computer*, **9**(4), 182-199.

[Mulmuley-1993] K. MULMULEY (1993), *Computational Geometry: An Introduction Through Randomized Algorithms*, Prentice Hall, New York.

[Murdoch, Benzley-1995] P. MURDOCH AND S. BENZLEY (1995), The Spatial Twist Continuum, *Proc. 4th International Meshing Roundtable*, 243-251.

[Nakahashi, Sharov-1995] K. NAKAHASHI AND D. SHAROV (1995), Direct surface triangulation using the advancing-front method, *AIAA-95-1686-CP*, 442-451.

[Natarajan-1997] R. NATARAJAN (1997), Domain Decomposition using Spectral Expansions of Steklov-Poincar Operators II: A Matrix Formulation, *SIAM Journal on Scientific Computing*, **18**(4), 1187-1199.

[Nielson, Hamann-1991] G.M. NIELSON AND B. HAMANN (1991), The asymptotic decider: resolving the ambiguity in Marching Cubes, *Visualization'91*, Conf. Proc., IEEE Computer Society Press, 83-90.

[Ning, Bloomenthal-1993] P. NING AND J. BLOOMENTHAL (1993), An evaluation of implicit surface tilers, *IEEE Computer Graphics & Applications*, **13**(6), 3341.

[Noël *et al.* 1995] NOËL F., LÉON J. C., TROMPETTE P. (1995), A New Approach to Free-form Surface Mesh Control in a CAD Environment, *Int. j. numer. methods eng.*, **38**(18), 3121-3142.

[Oden *et al.* 1995] J.T. ODEN, W. WU AND M. AINSWORTH (1995), Three-step hp adaptive strategy for the incompressible Navier-Stockes equations, *IMA Volumes in Mathematics and its Applications*, I. Babuska, W.D. Henshaw, J.E. Oliger, J.E. Flaherty, J.E. Hopcroft and T. Tezduyar (Eds.), **75**, 347-366.

[Ohtake, Bogaevski 2001] Y. OHTAKE AND I. BOGAEVSKI (2001), Mesh regularization and adaptive smoothing, *Computer-Aided Design* **33**.

[Okusanya, Peraire-1996] T. OKUSANYA AND J. PERAIRE (1996), Parallel unstructured mesh generation, *Proc. 5th Int. Conf. Numerical Grid Generation in Computational Field Simulations*, Mississippi State Univ., 719-729.

[deOliveira *et al.* 1997] M. DE OLIVEIRA, A.C. SALGADO, R.A. FEIJÓO, M.J. VÉNERE AND E.A. DARI, An object oriented Tool for automatic surface mesh generation using the Advancing Front technique, *Latin American Applied Research*, **27**(1/2), 39-49.

[O'Rourke-1994] J. O'ROURKE (1994), *Computational geometry in C*, Cambridge University Press.

[Osher, Sethian 1988] S. OSHER, J.A. SETHIAN (1988), Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations, *J. Comp. Phys.*, **79**, 12-49.

[Owen *et al.* 1998] S.J. OWEN, M.L. STATEN, S.A. CANANN AND S. SAIGAL (1998), Q-Morph: an indirect approach to advancing-front quad meshing, *Int. J. Numer. Meth. Eng.*, **44**(9), 1317-1340.

[Owen-1999a] S.J. OWEN (1999), Non-Simplicial Unstructured Mesh Generation, PhD thesis, Carnegie Mellon Univ., Pittsburg, PA.

[Owen-1999b] S.J. OWEN (1999), Constrained triangulation: application to hex-dominant mesh generation, *Proc. 8th Meshing Roundtable.*

[Owen, Saigal-2000] S.J. OWEN AND S. SAIGAL (2000), Surface mesh sizing control, *Int. J. Numer. Meth. Eng.*, **46**, 497-511.

[Pain *et al.* 2001] C.C PAIN, A.P. HUMPLEBY, C.R.E. DE OLIVEIRA AND A.J.H. GODDARD (2001), Tetrahedral mesh optimisation and adaptivity for steady-state and transient finite element calculations, *Comput. Methods Appl. Mech. Engrg.*, **190**, 3771-3796.

[Palmerio-1994] B. PALMERIO (1994), An attraction-repulsion mesh adaption model for flow solution on unstructured grid, *Comp. and Fluids*, **23**(3), 487-506.

[Parthasarathy *et al.* 1993] V.N. PARTHASARATHY, C.M. GRAICHEN AND A.F. HATHAWAY (1993), A comparison of tetrahedron quality measures, *Finite Elements in Analysis and Design*, **15**, 255-261.

[Pasko *et al.* 1995] A. PASKO, V. ADZHIEV, A. SOURIN AND V. SAVCHENKO, Function representation in geometric modeling: concepts, implementation and applications, *The Visual Computer*, **11**, 429-446.

[Pebay-1998a] P. PEBAY (1998), Construction d'une triangulation surfacique Delaunay admissible, *RR INRIA* **3369**.

[Pebay-1998b] P. PEBAY (1998), Construction d'une contrainte Delaunay admissible en dimension 2, *RR INRIA* **3492**.

[Peraire *et al.* 1987] J. PERAIRE, M. VAHDATI, K. MORGAN AND O.C. ZIENKIEWICZ (1987), Adaptive remeshing for compressible flow computations, *Jour. of Comput. Phys.*, **72**, 449-466.

[Peraire *et al.* 1988] J. PERAIRE, J. PEIRO, L. FORMAGGIA, K. MORGAN, O.C. ZIENKIEWICZ (1988), Finite element Euler computations in three dimensions, *Int. J. Numer. Methods Eng.*, **26**, 2135-2159.

[Peraire *et al.* 1992] J. PERAIRE, J. PEIRO, K. MORGAN (1992), Adaptive remeshing for three-dimensional compressible flow computations, *Jour. of Comput. Phys.*, **103**, 269-285.

[Peraire, Morgan-1997] J. PERAIRE AND K. MORGAN (1997), Unstructured mesh generation including directional refinement for aerodynamic flow simulation, *Finite Elements in Analysis and Design*, **25**(3-4), 343-355.

[Perronnet-1984] A. PERRONNET (1984), Logical and physical representation of an object, modularity for the programming of finite element methods, *PDE Software, Interfaces and Systems*, Elsevier, IFIP.

[Perronnet-1988a] A. PERRONNET (1988), Tétraèdrisation d'un objet multimatériaux ou de l'extérieur d'un objet, L.a.n. 189, Université Paris 6.

[Perronnet-1988b] A. PERRONNET (1988), A generator of tetrahedral finite elements for multi-material object and fluids, Numerical grid generation in computational fluid mechanics'88, Miami, FL.

[Perronnet-1992] A. PERRONNET (1992), Triangulation par arbre-4 de triangles équilatéraux et maximisation de la qualité, *Rapport de Recherche*, R-92015, Univ. Paris 6.

[Perronnet-1993] A. PERRONNET (1993), Tetrahedrization by the 5-14-OTT-tree technique and the Delaunay's criterion, *Proc. 8th Conf. on Finite Elements in Fluids*, Barcelona, Spain.

[Perronnet-1998] A. PERRONNET (1998), Interpolation transfinie sur le triangle, le tetraèdre et le pentaèdre. Application à la création de maillages et à la condition de Dirichlet. *C.R. Acad. Sci. Paris*, t 326, Series I, 117-122.

[Perucchio *et al.* 1989] R. PERUCCHIO, M. SAXENA AND A. KELA (1989), Automatic mesh generation from solid models based on recursive spatial decompositions, *Int. J. Numer. Meth. Eng.*, **28**, 2469-2501.

[Piegl, Richard-1995] L.A. PIEGL AND A.M. RICHARD (1995), Tesselations of trimmed NURBS surfaces, *Comput. Aided Des.*, **7**(1), 12-26.

[Piper-1987] B.R. PIPER (1987), Visually smooth interpolation with triangular Bézier patches, in *Geometric modeling: algorithms and new trends*, G. Farin Ed., SIAM, 221-233.

[Pironneau-1988] O. PIRONNEAU (1988), *Méthodes d'éléments finis pour les fluides*, Masson, Paris.

[Pirzadeh-1994] S. PIRZADEH (1994), Viscous unstructured three-dimensional grids by the advancing-layers method, *AIAA-94-0417*.

[Pothen *et al.* 1990] A. POTHEN, H. D. SIMON AND K.-P. LIOU (1990), Partitioning Sparse Matrices with eigenvectors of graphs, *SIAM Journal Matrix Analysis Application*, **11**(3), 430-252.

[Preparata, Hong-1977] F.P. PREPARATA AND S.J. HONG (1977), Convex hull of finite sets of points in two and three dimension, *Com. of the ACM*, **2**(20), 87-93.

[Price *et al.* 1995] M.A. PRICE, C.G. ARMSTRONG AND M.A. SABIN (1995), Hexahedral mesh generation by medial surface subdivision; Part I. Solids with convex edges, *Int. J. Numer. Methods Eng.*, **38**, 3335-3359.

[Price, Armstrong-1997] M.A. Price and C.G. Armstrong (1997), Hexahedral mesh generation by medial surface subdivision; Part II. Solids with flat and concave edges, *Int. J. Numer. Methods Eng.*, **40**, 111-136.

[Rajan-1994] V.T. Rajan (1994), Optimality of the Delaunay Triangulation in $\mathbb{R}^d$, *Discrete Comput. Geom.*, **12** , 189-202.

[Raviart, Thomas-1983] P.A. Raviart et J.M. Thomas (1983), *Introduction à l'analyse numérique des équations aux dérivées partielles*, Masson, Paris.

[Ravichandran, Shephard-1995] R. Ravichandran and M.S. Shephard (1995), Mesh searching structures for adaptive finite element analysis, *Scorec Report* **26**, RPI, Troy, NY.

[Ravindranath, Kumar 2000] M. N. Ravindranath and R. K. Kumar (2000), Simulation of Cold Forging using Contact and Practical Adaptive Meshing Algorithms, *Journal of Material Process Technology*, **104**, 110-126.

[Rank *et al.* 1993] E. Rank, M. Schweingruber and M. Sommer (1993), Adaptive mesh generation and transformation of triangular to quadrilateral meshes, *Commun. Numer. Methods Eng.*, **9**(2), 121-129.

[Rassineux-1995] A. Rassineux (1995), Maillage automatique tridimensionnel par une méthode frontale pour la méthode des éléments finis, Thesis, Nancy I.

[Rassineux-1997] A. Rassineux (1997), Maillage automatique tridimensionnel par une technique frontale et respect d'une carte de taille, *Revue européenne des éléments finis* **6**(1), 43-70.

[Rassineux *et al.* 2000] Rassineux A., Villon P., Savignat J-M, Stab O. (2000), Surface remeshing by local Hermite diffuse interpolation, *Int. J. Numer. Methods Eng.*, **49**, 31-49.

[Rebay-1993] S. Rebay (1993), Efficient unstructured mesh generation by means of Delaunay triangulation and Bowyer/Watson algorithm, *J. Comput. Physics*, **106**, 125-138.

[Reddy, Turkiyyah-1995] J.M. Reddy and G.M. Turkiyyah (1995), Computatin of 3D skeletons using a generalized Delaunay triangulation technique, *Computer Aided Design*, **27**(9), 677-694.

[vanRens *et al.* 1998] B.J.E. vanRens, D. Brokken, W.A.M. Brekelmans and F.P.T. Baaijens (1998), A two-dimensional paving mesh generator for triangles with controllable aspect ratio and quadrilaterals with high-quality, *Engineering with Computers*, **14**, 248-259.

[Requicha-1980] A.A.G. Requicha (1980), Representations for rigid solids: theory, methods and systems, *Comput. Surveys*, **12**, 437-464.

[Ricci-1972] A. Ricci (1972), A constructive geometry for computer graphics, *The Computer Journal*, **16**, 157-160.

[Rippa-1990] S. Rippa (1990), Minimal roughness property of the Delaunay triangulation, *Comput. Aided Geom. Design.*, **7**, 489-497.

[Rippa-1992] S. RIPPA (1992), Long and thin triangles can be good for linear interpolation, *SIAM J. Numer. Analysis*, **29**, 257-270.

[Risler-1991] J.J. RISLER (1991), *Méthodes mathématiques pour la C.A.O.*, RMA 18, Masson, Paris.

[Rivara-1984a] M.C. RIVARA (1984), Mesh refinement processes based on the generalised bisection of simplices, *SIAM J of Num. Ana.*, **21**, 604-613.

[Rivara-1984b] M.C. RIVARA (1984), Algorithms for refining triangular grids suitable for adaptive and multigrid techniques, *Int. J. Numer. Methods Eng.*, **21**, 745-756.

[Rivara-1986] M.C. RIVARA (1986), Adaptive finite element refinement and fully irregular and conforming triangulations, in *Accuracy Estimates and Adaptive Refinements in Finite Element Computations*, I. Babuska *et al.*, John Wiley and Sons.

[Rivara-1990] M.C. RIVARA (1990), Selective refinement/derefinement algorithms for sequences of nested triangulations, *Int. J. Numer. Methods Eng.*, **28**(12), 2889-2906.

[Rivara-1991] M.C. RIVARA (1991), Local modification of meshes for adaptive and/or multigrid finite element methods, *J. Comp. and Appl. Math.*, **36**, 79-89.

[Rivara, Levin-1992] M.C. RIVARA AND C. LEVIN (1992), A 3D refinement algorithm suitable for adaptive and multigrid techniques, *J. Comp. and Appl. Math.*, **8**, 281-290.

[Rivara-1997] M.C. RIVARA (1997), New longest-edge algorithms for the refinement and/or improvement of unstructured triangulations, *Int. J. Numer. Methods Eng.*, **40**, 3313-3324.

[Robinson-1987] J. ROBINSON (1987), Some new distorsion measures for quadrilaterals, *Finite Elements in Analysis and Design*, **3**, 183-197.

[Robinson-1988] J. ROBINSON (1988), Distorsion measures for quadrilaterals with curved boundaries, *Finite Elements in Analysis and Design*, **4**, 115-131.

[Rogers, Adams-1989] D.F. ROGERS AND J.A. ADAMS (1989), *Mathematical Elements for Computer Graphics*, McGraw-Hill.

[Ruppert, Seidel-1992] J. RUPPERT AND R. SEIDEL (1992), On the difficulty of triangulating three-dimensional nonconvex polyhedra, *Discrete Comput. Geom.*, **7**, 227-253.

[Ruppert-1995] J. RUPPERT (1995), A Delaunay refinement algorithm for quality 2-dimensional mesh generation, *J. Algorithms*, **18**(3), 548-585.

[Rypl, Krysl-1994] D. RYPL AND P. KRYSL (1994), Triangulation of 3-D surfaces, *Tech. Report*, Czech Tech. Univ., Prague.

[Rypl, Krysl-1997] D. RYPL AND P. KRYSL (1997), Triangulation of 3-D surfaces, *Engineering with Computers*, **13**, 87-98.

[Rypl-1998] D. Rypl (1998), Sequential and parallel generation of unstructured 3D meshes, PhD thesis, Faculty of Civil Engrg., Czech Tech. Univ., Prague.

[Rvachev-1963] V.L. Rvachev (1963), On the analytical description of some geometric objects, *Report of the Ukrainian Academy of Sciences*, **153**, 765-767.

[Said *et al.* 1999] Said R., Weatherill N. P., Morgan K. and Verhoeven N. A. (1999), Distributed parallel Delaunay mesh generation, *Comput. Methods Appl. Mech. Engrg,* , **177**, 109-125.

[Salmon, 1885] G. Salmon (1885), *Modern Higher Algebra*, Longmans, Greens and Co., London.

[Samareh-Abolhassani, Stewart-1994] J. Samareh-Abolhassani and J.E. Stewart (1994), Surface grid generation in parametric space, *J. Comput. Physics*, **113**, 112-121.

[Samet-1984] H. Samet (1984), The Quadtree and Related Hierarchical Data Structures, *ACM Computing Surveys*, **16**(2), 187-260.

[Samet-1989] H. Samet (1989), Neighbor finding in images represented by octrees, *CVGIP*, **46**, 367-386.

[Samet-1990] H. Samet (1990), *Design and analysis of spatial data structures*, Addison Wesley.

[Sapidis, Perucchio-1991] N. Sapidis and R. Perucchio (1991), Delaunay triangulation of arbitrarily shaped planar domains, *Comput. Aided Geom. Des.*, **8**, 421-437.

[Sapidis, Perucchio-1993] N. Sapidis and R. Perucchio (1993), Combining recursive spatial decompositions and domain Delaunay tetrahedrizations for meshing arbitrarily shaped curved solid models, *Comput. Methods Appl. Mech. Engrg.*, **108**, 281-302.

[Savchenko *et al.* 1995] V.V. Savchenko, A.A. Pasko, O.G. Okunev and T.L. Kunii (1995), Function representation of solids reconstructed from scattered surface points and contours, *Computer Graphics Forum*, **14**(4), 181-188.

[Saxena, Perucchio-1992] M. Saxena and R. Perucchio (1992), Parallel FEM algorithms based on recursive spatial decomposition. I: automatic mesh generation, *Computers & Structures*, **45**(5-6), 817-831.

[Saxena *et al.* 1995] M. Saxena, P.M. Finnigan, C.M. Graichen, A.F. Hathaway and V.N. Parthasarathy (1995), Octree-based automatic mesh generation for non-manifold domains, *Engineering with Computers*, **11**, 11-14.

[Schaeffer-1979] H. G. Schaeffer(1979), MSC/NASTRAN Primer. Static and Normal Modes Analysis, *Schaeffer Analysis*, Mont Vernon.

[Schmidt-1993] M.F.W. Schmidt (1993), Cutting cubes - visualizing implicit surfaces by adaptive polygonization, *The Visual Computer*, **10**, 101-115.

[Schneiders *et al.* 1996] R. SCHNEIDERS, R. SCHINDLER AND F. WEILER (1996), Octree-based generation of hexahedral element meshes, in *Proc. 5th International Meshing Roundtable*, 205-215.

[Schneiders-1996a] R. SCHNEIDERS (1996), A grid-based algorithm for generation of hexahedral element meshes, *Engineering with Computers*, **12**, 168-177.

[Schneiders-1996b] R. SCHNEIDERS (1996), Refining quadrilateral and hexahedral element meshes, *Proc. 5th Int. Conf. on Numerical Grid Generation in Computational Field Simulations*, 679-688.

[Schöberl-1997] J. SCHÖBERL (1997), NETGEN: An advancing-front 2D/3D-mesh generator based on abstract rule, *Comput Visual Sci.*, **1** 41-52.

[Schönhardt-1928] E. SCHÖNHARDT (1928), Über die Zerlegung von Dreieckspolyedern, *Mathematish Annalen*, **98**, 309-312.

[Schroeder, Shephard-1988] W.J. SCHROEDER AND M.S. SHEPHARD (1988), Geometry-based fully automatic mesh generation and the Delaunay triangulation *Int. J Numer. Meth. Eng.*, **26**, 2503-2515.

[Schroeder, Shephard-1989] W.J. SCHROEDER AND M.S. SHEPHARD (1989), An $O(N)$ algorithm to automatically generate geometric triangulations satisfying the Delaunay circumsphere criteria, *Engrg. with Computers*, **5**, 177-193.

[Schroeder, Shephard-1990] W.J. SCHROEDER AND M.S. SHEPHARD (1990), A combined octree/Delaunay method for fully automatic automatic 3D mesh generation, *Int. J. Numer. Meth. Eng.*, **29**, 37-55.

[Schroeder *et al.* 1992] W.J. SCHROEDER, J.A. ZARGE AND W. LORENSEN, Decimation of triangle mesh, *ACM Comput. Graphics*, **26**(2), 65-70.

[Sclaroff, Pentland-1991] S. SCLAROFF AND A. PENTLAND, generalized implicit functions for computer graphics, *Comput. Graph.*, **25**, 247-250.

[Schoofs *et al.* 1979] A.J.G. SCHOOFS, L.H.T. VANBEUKERING AND M.L.C. SLUITER (1979), A general purpose two-dimensional mesh generator, *Adv. Eng. Software*, **1**, 131-136.

[Sederberg-1983] T.W. SEDERBERG (1983), Implicit and parametric curves and surfaces for Computer-Aided Design, PhD thesis, Purdue Univ., W. Lafayette, Ind.

[Sederberg, Parry-1986] T. W. SEDERBERG AND S. R. PARRY (1986), Comparison of three curve intersection algorithms, *Computer Aided Design*, **18**(1), 58-63.

[Sederberg-1987] T. W. SEDERBERG (1987), Algebraic geometry for surface and solid modeling, in *Geometric Modeling: algorithms and new trends*, G.E. Farin, Ed. SIAM, 29-42.

[Sederberg, Chen-1995] T.W. SEDERBERG AND F. CHEN (1995), Implicitization using moving curves and surfaces, *Computer Graphics Annual Conf. Series*, 301-308.

[Sedgewick-1988] R. SEDGEWICK (1988), *Algorithms*, Addison-Wesley.

[Sedgewick, Flajolet-1996] R. SEDGEWICK AND PH. FLAJOLET (1996), *Lecture Notes on Bucket Algorithms,* Birkauser.

[Seidel-1982] R. SEIDEL (1982), The complexity of Voronoi diagrams in higher dimensions, *Proc. 20th Ann. Allerton Conf. Commun., Control, Comput.*, 94-95.

[Seidel-1991] R. SEIDEL (1991), Small-dimensional linear programming and convex hulls made easy, *Discrete Comput. Geom.*, **6**, 423-434.

[Sethian-1987] J. SETHIAN (1987), Numerical methods for propagating fronts, in *Variational Methods for Free Surface Interfaces*, *Proc. 1985 Vallambrosa Conference, E*ds. P. Concus and R. Finn, Springer-Verlag, NY.

[Seveno-1997] E. SEVENO (1997), Towards an adaptive advancing-front mesh generation, *Proc. 6th Int. Meshing Roundtable*, 349-360.

[Seveno-1998] E. SEVENO (1998), Génération automatique de maillages tridimensionnels isotropes par une méthdoe frontale, Thesis d'Université, Paris 6.

[Shamos, Preparata-1985] F.P. PREPARATA AND M.I. SHAMOS (1985), *Computational geometry, an introduction*, Springer-Verlag.

[Shapiro-1994] V. SHAPIRO (1994), Real functions for representation of rigid solids, *CAGD*, **11**, 153-175.

[Sheehy *et al.* 1996] D.S. SHEEHY, G.C. ARMSTRONG AND D.J. ROBINSON (1996), Shape description by medial surface construction, *IEEE Trans. on Vizualization and Comput. Graph.*, **2**(1), 62-72.

[Sheng, Hirsch-1992] X. SHENG AND B.E. HIRSCH (1992), Triangulation of trimmed surfaces in parametric space, *Comput. Aided Des.*, **24**(8), 437-444.

[Shenton *et al.* 1985] D.N. SHENTON AND Z.J. CENDES (1985), Three-dimensional finite element mesh generation using Delaunay tesselation, *IEEE Trans. Magnetics*, **21**(6), 2535-2538.

[Shephard-1988] M.S. SHEPHARD (1988), Approaches to the automatic generation and control of finite element meshes, *Applied Mechanics Reviews*, **41**, 169-185.

[Shephard-2000] M.S. SHEPHARD (2000), Meshing environment for geometry-based analysis, *Int. J. Numer. Methods Eng.* **47**, 169-190.

[Shephard *et al.* 1988] M.S. SHEPHARD, F. GUERINONI, J.E. FLAHERTY, R.A. LUDWIG, P.L. BAEHMANN (1988), Finite octree mesh generation for automated adaptive 3D flow analysis, *Numerical grid generation in computational fluid mechanics '88*, Miami.

[Shephard *et al.* 1988a] M.S. SHEPHARD, F. GUERINONI, J.E. FLAHERTY, R.A. LUDWIG, P.L. BAEHMANN (1988), Adaptive solutions of the Euler equations using finite quadtree and octree grids, *Computers & Structures*, **30**, 327-336.

[Shephard *et al.* 1988b] M.S. SHEPHARD, K.R. GRICE, J.A. LO, W.J. SCHROEDER (1988), Trends in automatic three-dimensional mesh generation, *Comp. and Struct.*, **30**(1/2), 421-429.

[Shephard, Georges-1991] M.S. SHEPHARD AND M.K. GEORGES (1991), Automatic three-dimensional Mesh Generation by the Finite Octree Technique, *Int. J. Numer. Meth. Eng.*, **32**, 709-749.

[Shephard, Georges-1992] M.S. SHEPHARD AND M.K. GEORGES (1992), Reliability of automatic 3D mesh generation, *Computer Methods in Appl. Mechanics and Engineering*, **101**, 443-462.

[Shephard *et al.* 1996] M.S. SHEPHARD, S. DEY AND M.K. GEORGES (1996), Automatic meshing of curved three-dimensional domains: curving finite elements and curvature-based mesh control, *Technical Report*, RPI Scientific Computation Research Center.

[Shewchuk-1997a] J.R. SHEWCHUK (1997), Delaunay Refinement Mesh Generation, PhD thesis, Technical Report CMU-CS-97-137, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania.

[Shewchuk-1997b] J.R. SHEWCHUK (1997), Adaptive precision floating-point arithmetic and fast robust geometric predicates, *Discrete and Computational Geometry*, **18**.

[Shirman, Séquin-1991] L.A. SHIRMAN AND C.H. SÉQUIN (1991), Procedural construction of patch-boundary-curves, in *Curves and surfaces*, Academic Press, Boston.

[Shmit-1982] R.E. SHMIT (1982), Algebraic grid generation, *Numerical grid generation*, Ed. J.F. Thompson, North Holland, 137.

[Shostko, Löhner-1995] A. SHOSTKO AND R. LÖHNER (1995), Three-Dimensional Parallel Unstructured Grid Generation, *Int. J. Numer. Methods Eng.*, **38**, 905-925.

[Simon-1991] H. SIMON (1991), Partitioning of unstructured problems for parallel processing, *Comp. Systems in Eng.*, **2**, 135-148.

[Simpson-1994] R.B. SIMPSON (1994), Anisotropic mesh transformation and optimal error control, *Applied Num. Math.*, **4**, 183-198.

[Simpson-1998] R.B. SIMPSON (1998), C++ classes for 2D unstructured mesh programming, *RR INRIA* **3592**.

[Sloan-1987] S.W. SLOAN (1987), A fast algorithm for constructing Delaunay triangulations in the plane, *Adv. Eng. Soft.*, **9**(1), 34-55.

[Sloan, Houlsby-1984] S.W. SLOAN AND G.T. HOULSBY (1984), An implementation of Watson's algorithm for computing 2-dimensional Delaunay triangulations, *Adv. Eng. Soft.*, **6**(4), 192-197.

[Sloan-1986] S.W. SLOAN (1986), An algorithm for profile and wavefront reduction of sparse matrices, *Int. J. Numer. Methods Eng.*, **23**, 239-251.

[Smith *et al.* 1996] B. SMITH, P. BJØRSTAD AND W. GROPP (1996), *Domain decomposition*, Cambridge Univ. Press, Cambridge.

[Stander, Hart-1997] B.T. STANDER AND J.C. HART (1997), Guaranteeing the topology of an implicit surface polygonization for interactive modeling, *Computer Graphics Proc.*, 279-286.

[Staten *et al.* 1998] M.L. STATEN, S.A. CANANN AND S.J. OWEN (1998), BMsweep: Locating Interior Nodes During Sweeping, *Proc 7th Int. Meshing Roundtable*, 7-18.

[Steger, Sorenson-1980] J.L. STEGER AND R.L. SORENSON (1980), Use of hyperbolic partial differential equations to generate body-fitted coordinates, *Proc. NASA Langley workshop on Numerical grid generation techniques.*

[Stein *et al.* 2004] E. STEIN, R. DE BORST AND T.J.R. HUGHES (2004), Encyclopedia of Computational Mechanics, E. Stein, R. de Borst and T.J.R. Hughes Eds, Wiley.

[Tacher, Parriaux-1996] L. TACHER AND A, PARRIAUX (1996), Automatic nodes generation in N-dimensional space *Commun. Numer. Methods Eng.*, **12**, 243-248.

[Talbert, Parkinson-1991] J.A. TALBERT AND A.R. PARKINSON (1991), Development of an automatic two-dimensional finite element mesh generator using quadrilateral elements and Bezier curve boundary definition, *Int. J. Numer. Methods Eng.*, **29**, 1551-1567.

[Talon-1987a] J.Y. TALON (1987), Algorithmes de génération et d'amélioration de maillages en 2D, Rapport Technique Artemis-Imag **20**.

[Talon-1987b] J.Y. TALON (1987), Algorithmes d'amélioration de maillages tétraèdriques en 3 dimensions, Rapport Technique Artemis-Imag **21**.

[Tam, Armstrong-1991] T.K. TAM AND G.C. ARMSTRONG (1991), 2D finite element mesh generation by medial axis subdivision, *Adv. in Engrg. Soft.*, **13**, 313-324.

[Tamminen, Jansen-1985] M. TAMMINEN AND F.W. JANSEN (1985), An integrity filter for recursive subdivision meshes, *Computers and Graphics*, **9**(4), 351-363.

[Tanaka *et al.* 1990] H. TANAKA, O. KLING AND D.T. LEE (1990), On surface curvature computation from level set contours, *Proc. 10th Int. Conf. Pattern Recognition*, Atlantic City, NJ, 155-160.

[Tanemura *et al.* 1983] M. TANEMURA, T. OGAWA AND N. OGITA (1983), A new algorithm for three-dimensional Voronoï tesselation, *J. Comp. Phys.*, **51**, 191-207.

[Tarjan, Van Wyk-1988] R.E. TARJAN AND C.J. VAN WYK (1988), An $O(nlog\,logn)$-time algorithm for triangulating a simple polygon, *SIAM J. Comput.*, **17**, 143-178.

[Taubin-1992] G. TAUBIN (1992), Rasterizing implicit curves by space subdivision, *IBM Research Report*, **RC 17913**.

[Taubin-1995] G. TAUBIN (1995), Curve and surface smoothing without shrinkage, *Int. Conf. Computer Vision.*

[Taubin *et al.* 1996] G. TAUBIN, T. SHANG AND G. GOLUB (1996), Optimal surface smoothing as filter design, *Proc. 4th European Conf. Computer Vision.*

[Taubin, Rossignac-1998] G. TAUBIN AND J. ROSSIGNAC (1998), Geometric compression through topological surgery, *ACM Trans. on Graphics*, **17**(2), 84-115.

[Thacker-1980] W.C. THACKER (1980), A brief review of techniques for generating irregular computational grids, *Int. J. Numer. Methods Eng.*, **15**, 1335-1341.

[Thiriet *et al.* 1998] M. THIRIET, P. BRUGIÈRES, F. RICOLTI, A. GASTON AND J. BITTOUN (1998), Modélisation numérique de l'écoulement dans les anévrismes cérébraux, *Proc. SNFR*, Paris.

[Thiriet *et al.* 1998] M. THIRIET, S. PIPERNO, G. MALLANDAIN, P. FREY, J. BITTOUN AND A. GASTON (1998), Computational models of flow in cerebral aneuvrisms, in *Proc. 11th Conf. of the ESB*, Toulouse, July 8-11, 1998.

[Thirion-1993] J.P. THIRION (1993), The marching lines algorithm: new results and proofs, *RR INRIA*, **1881**.

[Thompson-1982a] J.F. THOMPSON (1982), Numerical grids generation, *Appl. Math. and Comp.*, **10-11**.

[Thompson-1982b] J.F. THOMPSON (1982), *Elliptic grids generation, numerical grids generation*, Elsevier Science, 79-105.

[Thompson *et al.* 1985] J.F. THOMPSON, Z.U.A. WARSI, C.W. MASTIN (1985), *Numerical grids generation, foundations and applications*, North Holland.

[Thompson-1987] J.F. THOMPSON (1987), A general three dimensional elliptic grid generation system on a composite block-structure, *Comp. Meth. Appl. Mech. and Eng.*, **64**.

[Thompson *et al.* 1999] J.F. THOMPSON, B.K. SONI AND N.P. WEATHERILL (1999), *Handbook of grid generation*, CRC Press.

[Tinoco *et al* 2001] J.G. TINOCO, P. BARRERA AND A. CORTÉS (2001), Some Properties of Area Functionals in Numerical Grid Generation, *Proc. 10th Int. Meshing Roundtable.*

[Todd, McLeod-1986] P.H. TODD AND R.J.H. MC LEOD (1986) Numerical estimation of the curvature of surfaces, *Computer Aided Design*, **18**, 33-37.

[Tomasi *et al.* 2003] J.TOMASI, B.MENNUCCI AND P.LAUG (2003), The modeling and simulation of the liquid phase, *Handbook of Numerical Analysis*, Vol. X, Special Volume on Computational Chemistry, P.G. Ciarlet and C. Le Bris eds., North-Holland, Amsterdam, Netherlands, 271-375.

[Turk-1992] G. TURK (1992), Re-tiling polygonal surfaces, *Computer Graphics*, **26**(2), 55-64.

[Turkiyyah *et al.* 1997] G.M. TURKIYYAH, D.W. STORTI, M. GANTER, H. CHEN AND M. VIMAWALA (1997), An accelerated triangulation method for computing the skeletons of free-form solid models, *Computer Aided Design*, **29**(1), 5-19.

[Vallet-1990] M.G. VALLET (1990), Génération de maillages anisotropes adaptés - Application à la capture de couches limites, *RR INRIA* **1360**.

[Vavasis, Ye-1996] S. VAVASIS AND Y. YE (1996), A primal dual accelerated interior point method whose running time depends only on A, *Math. Progr.*, **74**, 79-120.

[Verfurth-1996] R. VERFÜRTH (1996), *A review of a posteriori error estimation and adaptive refinement techniques*, Wiley Teubner.

[Velho *et al.* 1997] L. VELHO, L.H. DEFIGUEIREDO AND J. GOMES (1997), A methodology for piecewise linear approximation of surfaces, *Journal of the Brazilian Computer Society*, **3**(3), 30-42.

[Voronoï-1908] G. VORONOÏ (1908), Nouvelles applications des paramètres continus à la théorie des formes quadratiques. Recherches sur les parallélloèdres primitifs. *Journal Reine Angew. Math.*, **134**.

[Walton, Meek-1996] D.J. WALTON AND D.S. MEEK (1996), A triangular G^1 patch from boundary curves, *Comp. Aided Design*, **28**, 113-123.

[Wan *et al.* 2003] J. WAN, S. KOCAK AND M.S. SHEPHARD (2003), Automated adaptive forming simulations, *Proc. 12th Int. Meshing Roundtable*, 323-334.

[Wang, Liang-1989] Y.F. WANG AND P. LIANG (1989), A new method for computing intrinsic surface properties, *Proc. Conf. Computer Vision and Pattern Recognition*, San Diego, CA, 235-240.

[Watson-1981] D.F. WATSON (1981), Computing the n-dimensional Delaunay Tesselation with applications to Voronoï polytopes, *Computer Journal*, **24**(2), 167-172.

[Weatherill-1985] N.P. WEATHERILL (1985), The generation of unstructured grids using Dirichlet tesselation, MAE report no. 1715, Princeton Univ.

[Weatherill-1988] N.P. WEATHERILL (1988), A method for generating irregular computational grids in multiply connected planar domains, *Int. J. Numer. Methods Fluids*, **8**.

[Weatherill-1988] N.P. WEATHERILL (1988), A strategy for the use of hybrid structured-unstructured meshes in CFD, *Num. Meth. for Fluids Dynamics*, Oxford University Press.

[Weatherill-1990] N.P. WEATHERILL (1990), The integrity of geometrical boundaries in the 2-dimensional Delaunay triangulation, *Commun. Numer. Methods Eng.*, **6**, 101-109.

[Weatherill, Hassan-1994] N.P. WEATHERILL AND O. HASSAN (1994), Efficient three-dimensionnal Delaunay triangulation with automatic point creation and imposed boundary constraints, *Int. J. Numer. Methods Eng.*, **37**, 2005-2039.

[Weatherill *et al.* 1994] N.P. WEATHERILL, M.J. MARCHANT, O. HASSAN AND D.L. MARCUM (1994), Grid adaptation using a distribution of sources applied to inviscid compressible flow simulations, *Int. Jour. Num. Methods Eng.*, **19**, 739-764.

[Weatherill *et al.* 1998] WEATHERILL N. P., SAID R. AND MORGAN K. (1998), The construction of large unstructured grids by parallel Delaunay grid generation, in *Proceedings of the 6th Inter. Conf. on Numerical Grid Generation in Computational Field Simulation*, pub. NSF Research Center, M.S.U., USA. 53-78.

[Weiler-1985] K. WEILER (1985), Edge-based data structure for solid modeling in curved-surface environments, *IEEE Computer Graphics and Applications*, **5**(1), 21-40.

[Weiler-1986] K. WEILER (1986), Topological structures for geometric modeling, *PhD thesis*, Computer and System Engineering, Rensselaer Polytechnic Institute, Troy, NY.

[Wilhelms, van Gelder-1990] J. WILHELMS AND A. VAN GELDER (1990), Topological considerations in isosurface generation, *Comput. Graphics*, **24**(5), 79-86.

[Winslow-1964b] A.M. WINSLOW (1964), Equi-potential zoning for two-dimensional meshes, *R. no. UCRL-7312.*

[Winslow-1966] A.M. WINSLOW (1966), Numerical solution of the quasilinear Poisson equations in a non-uniform triangle mesh, *J. Computat. Physics*, **1**, 149-172.

[Winslow-1967] A.M. WINSLOW (1967), Equipotential zoning of two dimensional meshes, *J. Computat. Physics*, **1**.

[Winslow-1967] A.M. WINSLOW (1967), Numerical solution of the quasilinear Poisson equation in a non uniform triangle mesh, *J. Comp. Phys.*, **1**

[Wirth-1986] N. WIRTH (1986), *Algorithms and Data-Structures*, Prentice Hall.

[Wyvill *et al.* 1986] G. WYVILL, C. MCPHEETERS AND B. WYVILL (1986), Data structure for soft objects, *The Visual Computer*, **2**, 227-234.

[Wordenweber-1984] B. WORDENWEBER (1984), Finite element mesh generation, *Computer Aided Design*, **16**, 285-291.

[Wright, Jack-1994] J.P. WRIGHT AND A.G. JACK(1994), Aspects of three-dimensional constrained Delaunay meshing, *Int. J. Numer. Methods Eng.*, **37**, 1841-1861.

[Wu, Houstis-1996] P. WU AND E.N. HOUSTIS (1996), Parallel adaptive mesh generation and decomposition, *Engineering with Computers*, **12**, 155-167.

[Yang *et al.* 2006] X. YANG *et al.* (2006), An adaptive coupled level-set/volume-of-fluid interface capturing method for unstructured triangular grids, *J. Comp. Phys.*, **217**, 364-394.

[Yao-1981] A.C. YAO (1981), A lower bound to finding convex hulls, *J. ACM*, **28**, 780-787.

[Yerry, Shephard-1983] M.A. YERRY AND M.S. SHEPHARD (1983), A modified-quadtree approach to finite element mesh generation, *IEEE Computer Graphics Appl.*, **3**(1), 39-46.

[Yerry, Shephard-1984] M.A. YERRY AND M.S. SHEPHARD (1984), Automatic three-dimensional mesh generation by the modified-octree technique, *Int. J. Numer. Meth. Eng.*, **20**, 1965-1990.

[Yerry, Shephard-1985] M.A. YERRY AND M.S. SHEPHARD (1985), Automatic three-dimensional mesh generation for three-dimensional solids, *Comp. Struct.*, **20**, 31-39.

[Yu *et al.* 1991] X. YU, J.A. GOLDAK AND L. DONG (1991), Constructing 3D discrete medial axis, *Proc. ACM Symp. Solid Modeling Found. and CAD/CAM Applic.*, Austin, 481-492.

[Zavattieri *et al.* 1996] P.D. ZAVATTIERI, E.A.DARI AND G.C. BUSCAGLIA (1996), Optimization strategies in unstructured mesh generation, *Int. J. Numer. Methods Eng.*, **39**, 2055-2071.

[Zeng, Ethier 2005] D. ZENG AND C.R. ETHIER (2005), A semi-torsional spring analogy model for updating unstructured meshes in 3D moving domains, *Finite Elements in Analysis and Design*, **41**(11-12), 1118-1139.

[Zheng *et al.* 1996a] Y. ZHENG, R.W. LEWIS AND D.T. GETHIN (1996), Three-dimensional unstructured mesh generation: Part 1. Fundamental aspects of triangulation and point creation, *Comput. Methods Appl. Mech. Engrg.*, **134**, 249-268.

[Zheng *et al.* 1996b] Y. ZHENG, R.W. LEWIS AND D.T. GETHIN (1996), Three-dimensional unstructured mesh generation: Part 2. Surface meshes, *Comput. Methods Appl. Mech. Engrg.*, **134**, 269-284.

[Zhu *et al.* 1991] J.Z. ZHU, O.C. ZIENKIEWICZ, E. HINTON AND J. WU (1991), A new approach to the development of automatic quadrilateral mesh generation, *Int. J. Numer. Methods Eng.*, **32**, 849-866.

[Zienkiewicz-2005] O.C. ZIENKIEWICZ (2005), *The Finite Element Method*, Butterworth-Heinemann, 6th ed.

[Zienkiewicz, Phillips-1971] O.C. ZIENKIEWICZ, D.V. PHILLIPS (1971), An automatic mesh generation scheme for plane and curved surfaces by isoparametric co-ordinates, *Int. J. Numer. Methods Eng.*, **3**, 519-528.

[Zienkiewicz-Zhu-1991] O.C. ZIENKIEWICZ AND X. ZHU (1991), Adaptivity and mesh generation, *Int. J. Numer. Methods Eng.*, **32**, 783-810.

Index