CAD model inspection utility and prototyping framework based on OpenCascade

S. Slyadnev, A. Malyshev, V. Turlapov.

Abstract

We present an open-source, extensible software which assists in the development of CAD geometric algorithms based on Boundary Representation data structures. The software is coupled with OpenCascade library which is an open-source geometric modeling kernel offering a broad range of modeling operations for the development of multi-purpose CAD systems and utilities. The new software serves as a framework for constructing geometric modeling algorithms. We provide several examples of geometric utilities developed with the help of the present framework. Furthermore, our software offers some advanced facilities for inspection of geometric and topological structures of existing CAD parts which can be imported from any popular CAD system.

Keywords

CAD, geometric modeling, boundary representation, OpenCascade, VTK, software framework.

1. Introduction

Research and development in geometric modeling field for Computer-Aided Design is hardly possible without specialized tools for visualization and analysis of geometry defined in Boundary Representation (B-Rep) form. In this paper, we present an open-source software which assists in handling a certain amount of common geometric issues. The software is based on the OpenCascade library as it is the only currently available open-source geometric modeling kernel. Another reason to use OpenCascade is that the data structures of the library are compatible with ISO 10303-42 (STEP). The latter fact brings a certain level of generality to the present approach. The similar kind of software is known to exist for other geometric kernels. However, other kernels are closed-source and are not readily available for an unrestricted academic and commercial use.

The remainder of this paper is organized as follows. Section 2 describes the motivations for starting yet another open-source project and lists the problems the project should be able to solve. Section 3 provides an overview of similar existing tools. Section 4 describes B-Rep inspection facilities of the new software. Section 5 examines how to use the software as a framework for prototyping new CAD-oriented algorithms. Section 5 concludes with several examples illustrating some real use cases where the software has been successfully used as a prototyping framework. Section 6 outlines our plans for future work.

2. Motivations

There are two primary motivations behind the idea to start yet another free and open-source (FOSS) software project dealing with CAD geometry algorithms. The first reason is the lack of freely available inspection utilities for Boundary Representation (B-Rep) geometry. It is widely recognized that a better visualization of a problem leads to a better understanding of the underlying science (see [Jern 1989]). The traditional visualization engines for Boundary Representation models are CAD systems. However, such systems do not allow for an easy inspection of boundary elements and their geometries. The existing visualization engines are primarily designed for end users and not for developers, although advanced CAD systems may offer comprehensive development kits in addition to mainstream software. The second motivation for starting this FOSS project is to allow for the so-called *reproducibility of research results* which is similar to the motivation behind the popular open-source MeshLab project presented by [Cignoni et al. 2008]. Our software simplifies prototyping new CAD modeling algorithms and renders the results of the research group's work readily available for academic and engineering communities. Availability of the research results in the form of software prototypes is a key to solving the *technology transfer* problem raised by [Brown 1982].

Inspection of geometric and topological structures is of crucial importance when developing or profiling geometric modeling algorithms, such as Boolean Operations, fillets, offsets, etc. During such tedious working sessions, a lot of common questions may arise. Some of them are as follows: a) what is the host geometry (curve or surface type) of a particular edge or a face? b) what are the topological and geometric orientations of the geometric primitives? c) what is the topological structure of the model? d) how to find a particular boundary element in the model? e) what is the quality of a curve or a surface parameterization? f) is it necessary to repair the working geometry and how to do that? Answering these and many other similar geometric questions often requires laborious manual work. In this paper, we introduce a software which assists in geometric analysis and substantially reduces the efforts needed to figure out "what my model consists of".

In addition to its inspection functions, the software can also serve as a framework for the development of geometric modeling algorithms. The software provides STEP-compliant geometric and topological primitives. These primitives are rendered using a set of readily available VTK pipelines. An algorithm which is developed within the framework of our system may benefit from the convenience interfaces which facilitate error reporting and graphical dumps (similar in nature to the commonly used textual diagnostic outputs). Therefore, the software establishes a framework for implementing multipurpose geometric tools for Computer-Aided Design applications. The software permitted to develop several CAD algorithms including some commercial ones.

Furthermore, it was discovered that the developed inspection and prototyping facilities are helpful in teaching the basic CAD concepts. According to [Rossignac 2004], CAD learners suffer from the unavailability of simplistic and intuitive models of the concepts they have to deal with during the education process. For example, students and even experienced engineers often confront such common problem as a misleading visual appearance of a CAD model. The geometry may look visually perfect, but contain a variety of internal quality phenomena, including wild or irregular parametrization of curves and surfaces, presence of redundant topological primitives, large tolerance gaps, and overlaps, etc. The first stage is to explain that what is displayed on the screen is not the real definition of a CAD part (What You See is NOT What You Get). This is exactly where a boundary representation inspection utility can be advantageous. In this regard, the present software contributes to *the education-driven research in CAD* as formulated by [Rossignac 2004].

Finally, the efficient development of geometric modeling algorithms involves some mathematical abstractions which cannot be visualized directly in the modeling space. Such abstractions include adjacency graphs for boundary elements, parametric domains of faces, level sets of objective functions being optimized, etc. The visualization of such abstract data often gives a useful insight in a problematic situation where a human imagination may fail to recognize the cause of a problem. As mentioned by [Zenkin 1991], a system providing such cognitive visualization facilities is the so called "brainware" because it assists humans in their creative process.

3. Related work

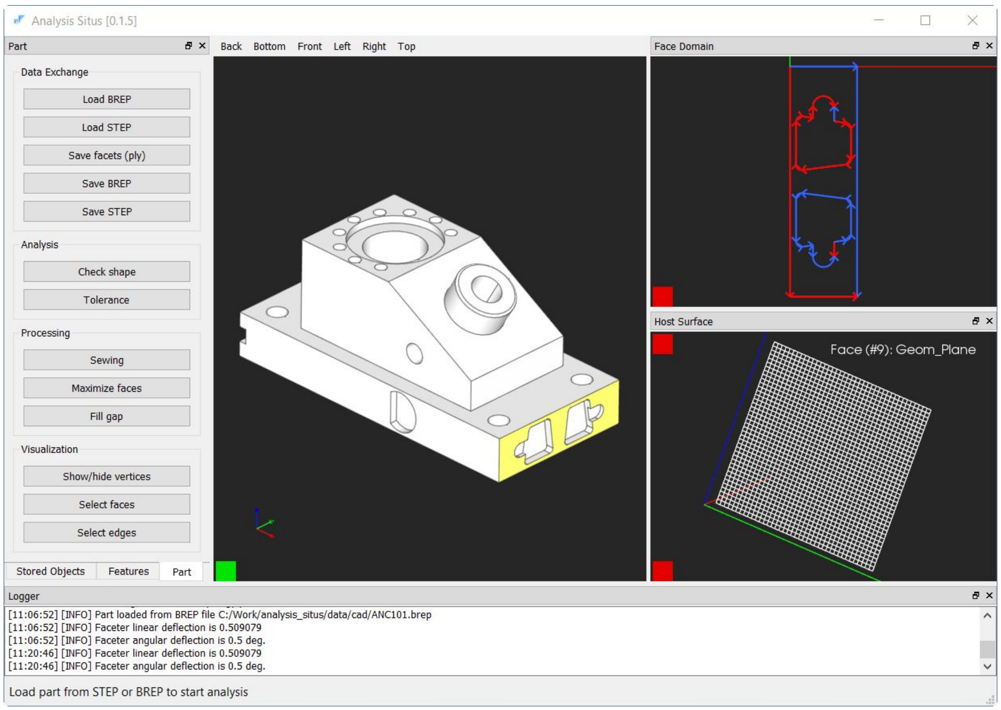
Among the available frameworks for geometry processing, it is worth mentioning MeshLab (see [Cignoni et al. 2008]) and OpenFlipper (see [Mobius and Kobbelt 2012]) software tools. Both systems were designed with the similar motivation to make the development of geometric algorithms more efficient, and to bridge the gap between academic researchers and industry. Our framework pursues the same goals, but focuses on precise CAD instead of polyhedral geometry. Although OpenFlipper provides a generic architecture which applies to curved geometry, it does not use a full-featured embedded CAD kernel. Therefore, the development of precise CAD algorithms will be problematic in such systems.

[Averbukh et al. 2015] presented a framework for deriving specialized systems for cognitive visualization. The cognitive visualization tools facilitate the investigation of subtle algorithmic problems, and even the creation of new knowledge. According to [Brooks 1996], such systems serve as "intelligence amplification" tools. The ability to visualize information which does not correspond to any real object but represents some abstract or multi-dimensional data is of great importance for the efficient and thoughtful development of geometric algorithms.

In practice, CAD systems and specialized CAD tools are often used for the development of geometric algorithms. These systems provide extensible frameworks which can be exploited by software developers and researchers. Data communication between an algorithm and a CAD system can be either direct (an algorithm calls the corresponding API of a system) or indirect with the use of exchange file formats (e.g. STEP). In the latter case, a geometric algorithm maintains a certain level of independence, as it can be used in a CAD-neutral environment. CAD software which integrates a newly developed geometric algorithm should offer sufficient functionality which is sometimes rather advanced. For example, [Lee et al. 2005] base their experiments on a commercial platform which provides enough functionality to prepare CAD models for the engineering analysis. Likewise, [Seo et al. 2005] base their algorithm on Parasolid kernel which offers an advanced face removal operator. These examples illustrate that a particular technology can be essential to conduct research in a specific field.

The alternative approach consists in developing technology-neutral algorithms which communicate with an underlying geometric kernel using an abstract interface. Researchers have made several attempts to create such interface. Probably the most remarkable one is Djinn interface proposed by [Armstrong et al. 2002]. The Djinn interface eliminates dependency on the underlying geometric kernel. Moreover, it unifies representation of a geometric model with the use of the concept of cellular topology. According to [Stroud and Nagy 2011], Djinn's authors aimed to create a standard interface which could be used to share applications between researchers and to demonstrate research applications within a commercial environment. The existence of such interface could lead to a closer collaboration between researchers and industry. Therefore, Djinn contributes to solving the *technology transfer* problem raised by [Brown 1982]. Unfortunately, the concept of Djinn does not take into account commercial considerations. Indeed, software vendors prefer to have a code that is closely tied to their system. According to [Stroud and Nagy 2011], having Djinn interface in a CAD system would allow researchers to add new functions to a CAD system, such as feature recognition, analysis or manufacturing code. However, to develop a software system which is fully compliant with Djinn is a challenge.

We have chosen another approach which is essentially technology-dependent. At the same time, the employed technology (OpenCascade) is open-sourced and readily available for anyone from academia or industry. Moreover, this modeling kernel is known to provide only a small subset of functionality which is available in commercial libraries such as ACIS or Parasolid. Therefore, any algorithm based on OpenCascade can be transmitted to another library, at least in principle. From this point of view, OpenCascade-based algorithms can be considered as prototypes for exchange between academia and industry.



```
[11:06:52] [INFO] Part loaded from BREP file C:/Work/analysis_situs/data/cad/ANC101.brep
[11:06:52] [INFO] Faceter linear deflection is 0.509079
[11:06:52] [INFO] Faceter angular deflection is 0.5 deg.
[11:20:46] [INFO] Faceter linear deflection is 0.509079
[11:20:46] [INFO] Faceter angular deflection is 0.5 deg.
```

Load part from STEP or BREP to start analysis

Fig 1. The general view of the software. The desktop is composed of three viewers: part viewer, face domain viewer and host geometry viewer.

OpenCascade library provides its own prototyping framework called Draw Test Harness (or simply Draw). Although our software is not as multi-purpose as Draw, there are several advantages which make it more practical than Draw in some cases. First of all, it offers a combined layout of three principal geometric viewers, as shown in Figure 1. These viewers are interactive and do not require any console inputs like Draw. Among other features are explicit indexation of topological elements, availability of graph models, curvature maps and combs for geometric analysis, convexity analysis for edges, developer-oriented visualization and many other utilities which are not readily available in Draw.

4. Inspection facilities

For the three-dimensional visualization, VTK library by [Schroeder et al. 2006] was employed. The architecture of visualization pipelines offered by VTK allows for a clear distinction between the data transformation before rendering and the rendering itself. Another remarkable advantage of VTK is a broad set of available visualization features which are commonly used by many scientists and software engineers all over the world.

4.1. B-Rep inspection

According to [Corney and Lim 2001], although different geometric kernels have their subtleties in B-Rep realization, there is one consensus that emerged earlier on. The well-turned idea was to separate geometric definitions of boundary elements from the information about how these elements are connected. This idea which divorces topology from geometry is presented in any commercial multi-purpose CAD geometric kernel.

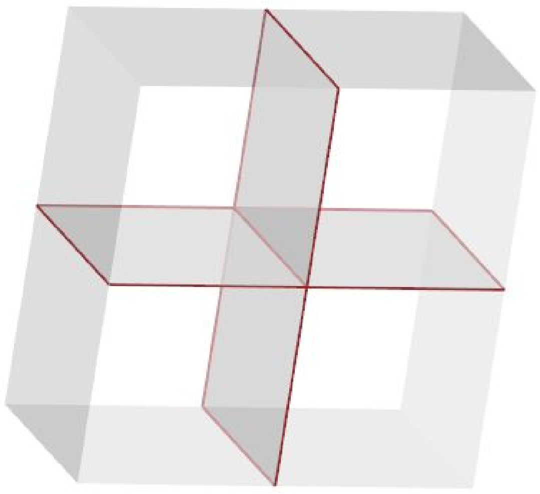


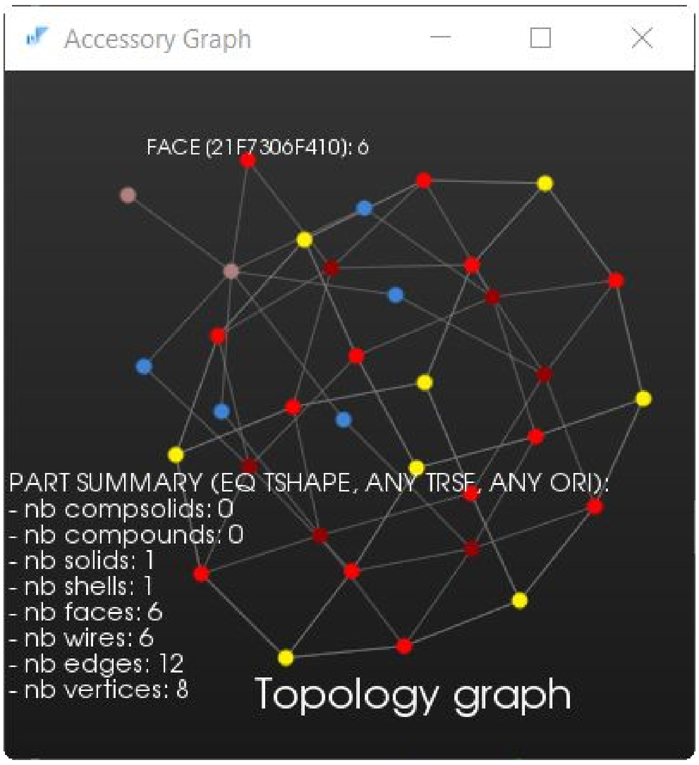
Fig 2. Non-manifold edges (dark-red color) in a model composed of four boxes sharing their faces.

The visualization facilities for B-Rep structures introduced in our software are not unique. Similar techniques for insightful rendering of boundary representation structures are given by [Corney and Lim 2001]. When designing a visualization system, it is first necessary to define how boundary representation structures of a particular modeling kernel are implemented. Modern libraries have to deal with both manifold and non-manifold objects. OpenCascade, as well as ACIS, utilize OBJECT/CO-OBJECT (e.g. EDGE/CO-EDGE) data structure which is capable of representing non-manifold models (Figure 2).

4.2. Adjacency graphs

According to [Corney and Lim 2001], the great value of graphs in geometric modeling comes from their ability to represent the connectivity between geometric elements. The Scheme AIDE utility of ACIS allows for a visualization of different graphs including vertex-edge and face adjacency graphs.

In our software, two graphs can be visualized for a part: the topology graph (Figure 3), which provides full information on how topological elements are nested into each other, and the attributed adjacency graph (Figure 4) representing adjacency relations between individual faces as explained by [Joshi and Chang 1988]. Both graphs give insight into the internal organization of the topological structure of a particular part (which is not visible without specialized inspection tools).



PART SUMMARY (EQ TSHAPE, ANY TRSF, ANY ORI):
- nb compsolids: 0
- nb compounds: 0
- nb solids: 1
- nb shells: 1
- nb faces: 6
- nb wires: 6
- nb edges: 12
- nb vertices: 8

Fig. 3. Topology graph of a simple box model. Different colors are used to distinguish between different types of topological elements (compounds, solids, shells, faces, etc.).

Both graphs are synchronized with the part viewer which allows for easy searching of the face of interest in the topological structure of the model.

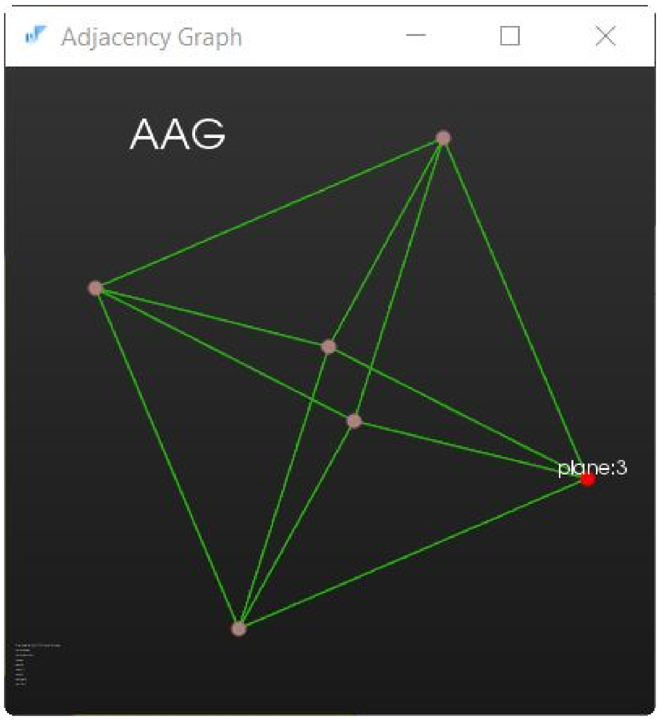
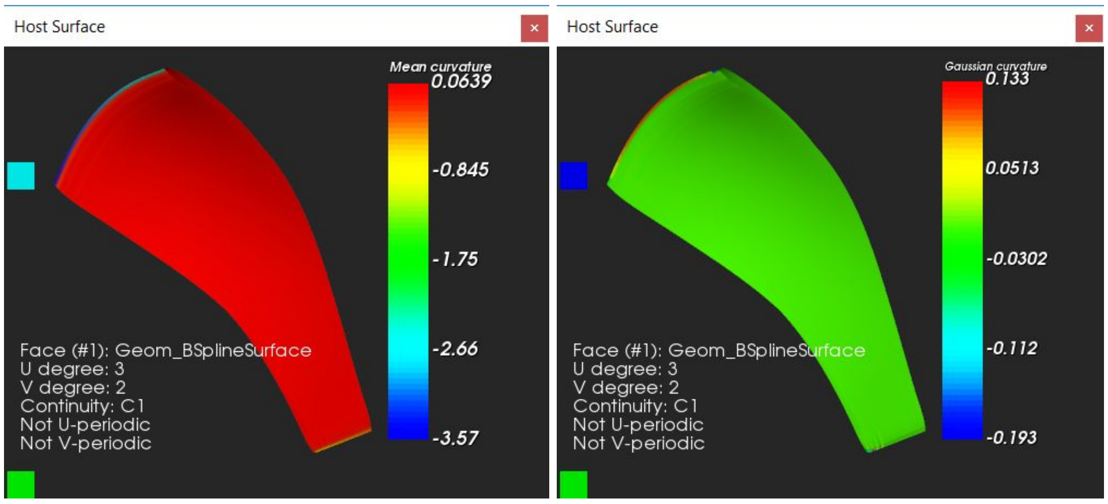


Fig. 4. Attributed adjacency graph of a simple box model. The nodes correspond to faces; the arcs represent adjacency relations.

Graph representations are especially useful in algorithms which involve analysis of the connection between boundary elements of a model. Such information is fully used in feature recognition scenarios to extract the engineering semantics of a model. Other applications are model simplification for CAE, local operations (see [Fahlbusch and Roser 1995] for introduction), etc.

4.3. Geometry analysis

As mentioned by [Greiner et al. 1995], visualization is the indispensable tool for controlling the quality of geometric objects including their differential and aesthetic properties. A shaded display is usually insufficient to draw a conclusion on the quality of a modeling result. Therefore, additional techniques such as curvature maps (Figure 5), or isoparametric wireframe views are employed to detect surface irregularities.



```
Face (#1): Geom_BSplineSurface
U degree: 3
V degree: 2
Continuity: C1
Not U-periodic
Not V-periodic
```

```
Face (#1): Geom_BSplineSurface
U degree: 3
V degree: 2
Continuity: C1
Not U-periodic
Not V-periodic
```

Fig. 5. Mean (on the left) and Gaussian (on the right) curvature maps for a B-spline blade surface.

5. Prototyping framework

According to [Mobius et al. 2013], researchers tend to confine their development efforts to implementing the algorithmic core and to spend less time on software engineering. Therefore, the authors of the OpenFlipper framework have implemented a set of common operations for data processing, visualization, I/O and many other components, including data structures, which are constantly and vainly re-invented by researchers and software engineers. Having all such components readily available allows developers to implement their innovative ideas in a rapid and goal-oriented way. Our main driving force was the idea to create new software, which will provide a similar framework for CAD-oriented research.

5.1. General architecture

Our software is designed to operate with a single CAD part. However, it can also import CAD assemblies without their component structure. This restriction simplifies the general architecture of the software, thus allowing it to operate as a test bench for a single part geometry. The software is also capable of working with meshes and point clouds at the basic level. The framework provides such commonly used features as hierarchical organization of data objects, open/save in binary format, undo/redo, multi-purpose 3D and 2D viewers, etc.

Compared to MeshLab or OpenFlipper utilities which offer a plugin mechanism for software extension, we use a more straightforward prototyping approach. Our software provides a collection of components, including data structures, basic algorithms, visualization pipelines and typical GUI controls. A minimalistic application is provided to put these components together and to enable interactive B-Rep inspection. Every new algorithm being implemented within the framework should be supplied with its own independent executable. A prototype developer is free to choose particular GUI controls and to configure their communication. Such flexibility comes at a price. The developer is forced to write some sort of non-scientific code to bring everything together.

5.2. Use cases

In the following subsections, we present several use cases where our system helped to facilitate coding and debugging. Some of the mentioned algorithms were integrated into commercial CAD systems.

5.3.1. Feature recognition

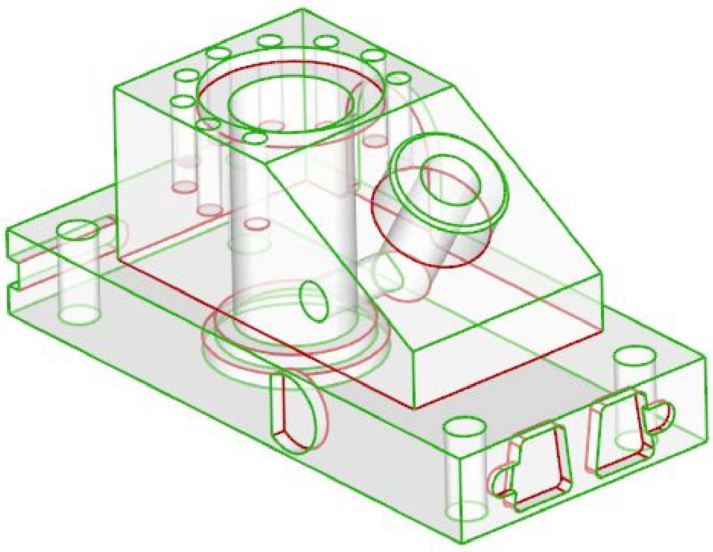


Fig. 6. Visualization of convex (green) and concave (red) edges for ANC101 model.

Feature recognition allows to recover the semantics of a geometric model when the information about its features is unavailable. Feature recognition is a primary application for model inspection utilities, including edge and face indexation, edge convexity analysis (Figure 6), traversing adjacency graphs, etc. This sort of functionality has found a broad use in manufacturing planning where such features as holes, pockets, and bosses are detected in an input 3D model. Figure 7 illustrates a sheet metal part (which was imported from a STEP file without the information on features) before and after recognition.

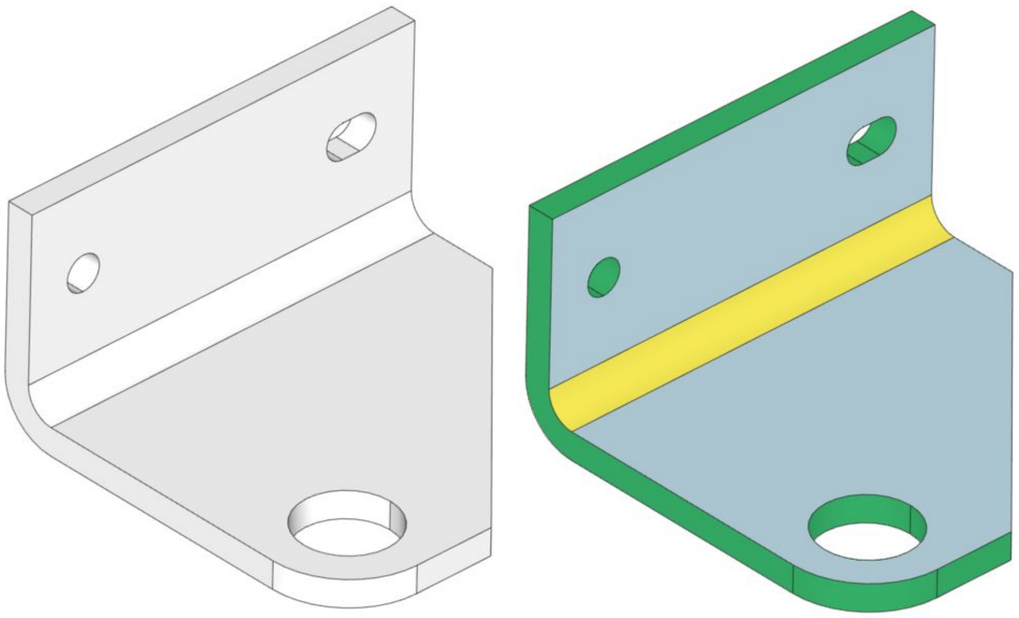


Fig. 7. The sheet metal part imported from STEP file (on the left) and feature recognition result (on the right) shown in different colors depending on the feature type.

In sheet metal manufacturing, the information about features can be used for fabrication cost estimation, validation of design by unfolding, preparation of 2D drawings, etc.

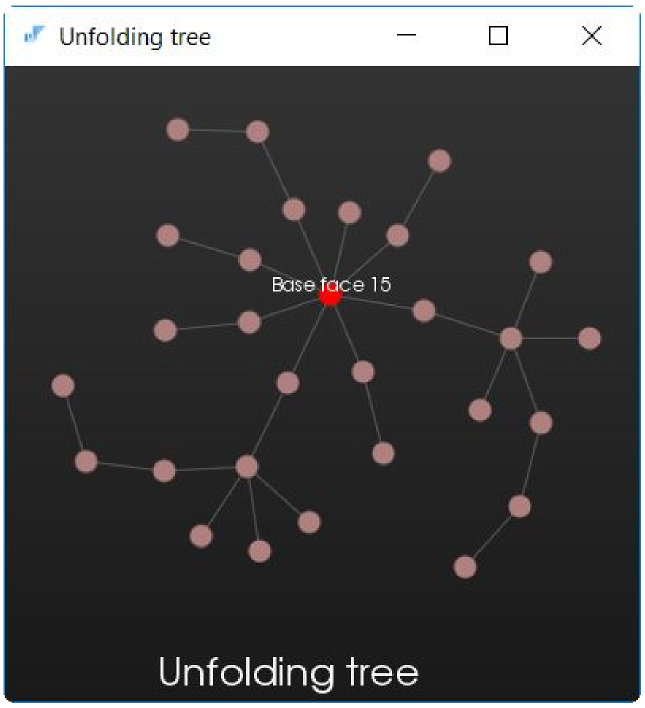


Fig. 8. Flattening graph Sheet metal part. Each node corresponds to a flange being unfolded to a flat pattern.

The prototyping system has not only helped to perform an interactive inspection of sheet metal features, but has also assisted in the construction of explicit unfolding graphs (Figure 8). Such graphs provide information about the order of flattening and transformations associated with every flange.

5.3.2. As-built reconstruction

The present framework is used in research and development for automated reconstruction of CAD models from three-dimensional point clouds and meshes. We use VTK to visualize massive point clouds and to let users interactively operate on their portions for segmentation and surface fitting. One example is the as-built reconstruction of industrial pipelines from laser-scan data (Figure 9).

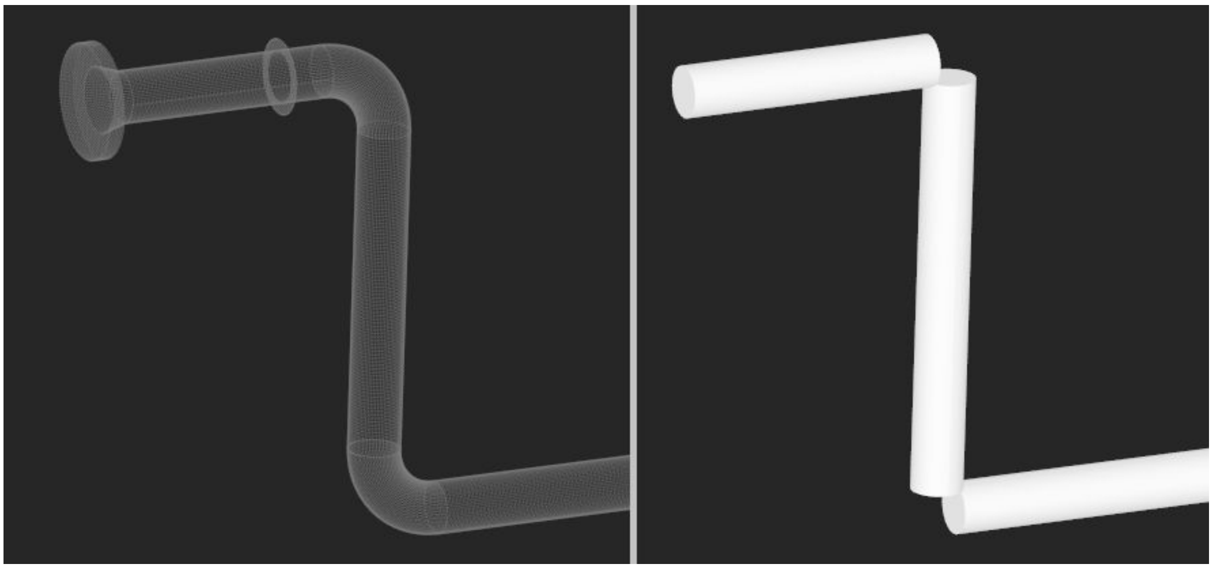


Fig. 9. The initial point cloud (on the left) and the extracted cylindrical primitives (on the right).

5.3.3. Contour capture

Contour capture algorithm is a variation of non-solid Boolean operation used to extract a certain piece of the two-dimensional (shell) model. The user interactively constructs a closed contour which follows the input geometry with a prescribed tolerance. The resulting polyline is then smoothed out using a dedicated reapproximation technique. Once the contour is smoothed, it can be then imprinted on the model by projection. At the last stage, the piece of geometry limited by the imprinted contour is extracted (Figure 10). Like any Boolean operation, this algorithm consists of several internal stages which can be implemented and tested separately. To facilitate the maintenance of this algorithm, we have used the novel framework which allowed to save a significant amount of time on debugging.

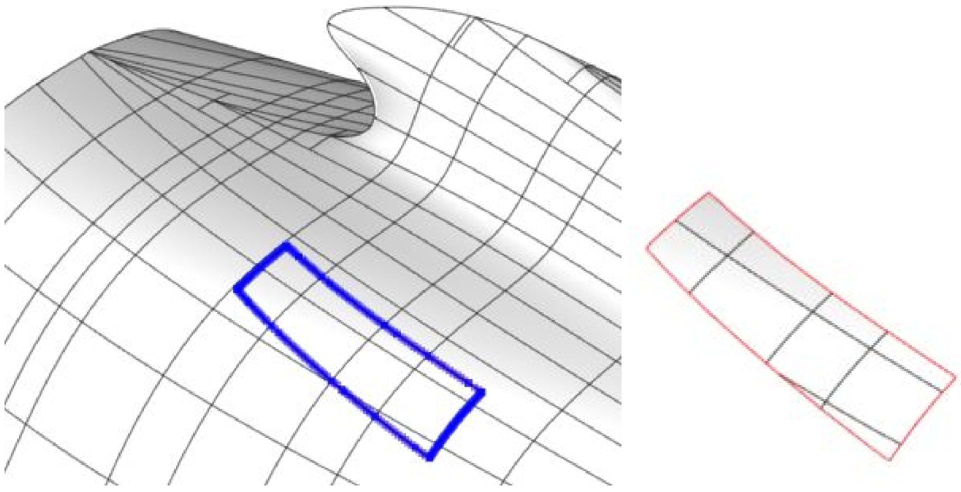


Fig. 10. Manually specified contour to capture (closed loop of line segments) and the result of capturing algorithm.

6. Conclusion and further work

The present software is available online at http://www.analysissitus.org. In future versions, we are planning to add new inspection facilities and basic modeling functionality. Particular attention will be paid to the extension of analysis tools for geometric properties, i.e. fairness of curves and surfaces following the ideas presented in [Burchard et al. 1994], [Greiner et al. 1995], [Hagen et al. 1995], [Sapidis and Koras 1997] and other works.

References

1. Armstrong, C.G., Corney, J.R., Salmon, J.C., and Cameron, S. 2002. Djinn. A Geometric Interface for Solid Modelling.
2. Averbukh, V.L., Bakhterev, M.O., Vasev, P.A., Manakov, D. V, and Starodubtsev, I.S. 2015. Development of approaches to realization of specialized visualization systems. GraphiCon, 17–21.
3. Brooks, F.P. 1996. Toolsmith II. Communications of the ACM 39, 3, 61–68.
4. Brown, C.M. 1982. PADL-2: A Technical Summary.
5. Burchard, H.G., Ayers, J.A., Frey, W.H., and Sapidis, N.S. 1994. Approximation with Aesthetic Constraints. Designing Fair Curves and Surfaces, 3–28.
6. Cignoni, P., Callieri, M., Corsini, M., Dellepiane, M., Ganovelli, F., and Ranzuglia, G. 2008. MeshLab: an Open-Source Mesh Processing Tool. Sixth Eurographics Italian Chapter Conference, 129–136.
7. Corney, J.R. and Lim, T. 2001. 3D Modelling with ACIS.

8. Fahlbusch, K.-P. and Roser, T.D. 1995. HP PE / SolidDesigner : Dynamic Modeling for Three-Dimensional Computer-Aided Design. Hewlett-Packard Journal, 6–13.

9. Greiner, G., Kolb, A., Pfeifle, R., Seidel, H., Encarna, M., and Klein, R. 1995. A Platform for Visualizing Curves and Surfaces. Computer-Aided Design 27, 7, 559–566.

10. Hagen, H., Hahmann, S., and Schreiber, T. 1995. Visualization and computation of curvature behaviour of freeform curves and surfaces. Computer-Aided Design 27, 7, 545–552.

11. Jern, M. 1989. Visualization of Scientific Data. Proceeding Advances in Computer Graphics V (Tutorials from Eurographics'89 Conf.), 1–17.

12. Joshi, S. and Chang, T.C. 1988. Graph-based heuristics for recognition of machined features from a 3D solid model. Computer-Aided Design 20, 2, 58–66.

13. Lee, K., Armstrong, C., and Price, M. 2005. A small feature suppression/unsuppression system for preparing B-rep models for analysis. Proceedings of the 2005 1, 212, 113–124.

14. Mobius, J. and Kobbelt, L. 2012. OpenFlipper: An open source geometry processing and rendering framework. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 6920 LNCS, 488–500.

15. Mobius, J., Kremer, M., and Kobbelt, L. 2013. OpenFlipper - A highly modular framework for processing and visualization of complex geometric models. 2013 6th Workshop on Software Engineering and Architectures for Realtime Interactive Systems, SEARIS 2013; Co-located with the 2013 Virtual Reality Conference - Proceedings, 25–32.

16. Rossignac, J. 2004. Education-driven research in CAD. CAD Computer Aided Design 36, 14, 1461–1469.

17. Sapidis, N.S. and Koras, G.D. 1997. Visualization of curvature plots and evaluation of fairness: an analysis of the effect of ‘scaling.’ Computer Aided Geometric Design 14, 96, 299–311.

18. Seo, J., Song, Y., Kim, S., Lee, K., Choi, Y., and Chae, S. 2005. Wrap-around operation for multi-resolution CAD model. Computer-Aided Design and Applications 2, 1-4, 67–76.

19. Schroeder, W., Martin, K., Lorensen, B. 2006. Visualization Toolkit: An Object-Oriented Approach to 3D Graphics. Kitware, Colombia.

20. Stroud, I. and Nagy, H. 2011. Solid Modelling and CAD Systems: How to Survive a CAD System.

21. Zenkin, A. 1991. Cognitive Computer Graphics.