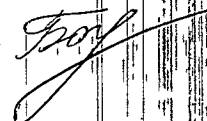


Федеральное государственное бюджетное образовательное
учреждение высшего профессионального образования
Нижегородский государственный университет им. Н.И. Лобачевского

На правах рукописи



БОГОЛЕПОВ Денис Константинович

**Методы глобального освещения для интерактивного синтеза
изображений сложных сцен на графических процессорах**

Шифр и наименование специальности
05.13.17 – «Теоретические основы информатики»
(технические науки)

Диссертация на соискание ученой степени
кандидата технических наук

Научный руководитель:
д.т.н. Турлапов Вадим Евгеньевич

Нижний Новгород – 2013

04201357234

26.06.2013

Содержание

Введение	4
Глава 1. Обзор публикаций и программного обеспечения по теме исследования	10
1.1. Трассировка лучей и глобальное освещение.....	10
1.2. Интерактивная трассировка лучей	13
1.3. Интерактивное глобальное освещение	53
1.4. Выводы к главе 1	59
Глава 2. Система моделей для алгоритмизации и конвейерной реализации лучевых методов синтеза изображений на массивно-параллельных вычислительных архитектурах	61
2.1. Элементы энергетического подхода к физически достоверному моделированию глобального освещения.....	61
2.2. Моделирование рассеяния света на поверхностях материалов.....	67
2.3. Исследование специализированных форм уравнения визуализации, подлежащих реализации в методах глобального освещения	70
2.4. Уравнение измерения как основа физически достоверного синтеза изображений	73
2.5. Моделирование пространства путей для решения уравнения измерения методом Монте-Карло	74
2.6. Компоненты генерации случайных путей для класса методов глобального освещения	76
2.7. Покомпонентное представление метода стохастической трассировки путей в системе моделей глобального освещения.....	81
2.8. Реализация и модификация двунаправленной трассировки путей на основе системы моделей глобального освещения	89
2.9. Выводы к главе 2	95

Глава 3. Универсальный конвейер трассировки лучей для интерактивного синтеза изображений методами глобального освещения на графических процессорах.....	96
3.1. Построение на графическом процессоре эффективной ускоряющей структуры для лучевых методов синтеза изображений	97
3.2. Метод представления и сериализации иерархической объектно-ориентированной модели сложной сцены.....	111
3.3. Компоненты универсальной системы синтеза изображений методами глобального освещения.....	123
3.4. Выводы к главе 3	146
Глава 4. Программный комплекс для интерактивного синтеза изображений сложных сцен на графических процессорах методами глобального освещения и его экспериментальное исследование.....	148
4.2. Состав и структура программного комплекса интерактивного синтеза изображений методами глобального освещения	148
4.3. Экспериментальная проверка корректности моделирования глобального освещения в различных условиях переноса световой энергии в сцене.....	151
4.4. Исследование алгоритма «усеченной» двунаправленной трассировки путей в различных условиях переноса световой энергии	155
4.5. Исследование производительности программного комплекса	163
4.6. Выводы к главе 4	170
Заключение.....	171
Список литературы.....	172

Введение

Актуальность темы диссертационной работы. Методы глобального освещения традиционно используются в компьютерной графике для синтеза фотореалистичных изображений. Программное обеспечение на их основе применяется в киноиндустрии, рекламе, проектировании архитектурных сооружений, дизайне помещений и офисов, компьютерных играх и симуляторах, а также во многих других системах виртуальной реальности. Расчет глобального освещения – вычислительно трудоемкая задача, для решения которой обычно применяются высокопроизводительные распределенные системы.

Традиционный способ визуализации динамических 3D сцен в реальном времени основан на алгоритме растеризации, который аппаратно ускоряется графическими процессорами и доступен через интерфейсы (API) OpenGL и Direct3D. Данный подход не позволяет обрабатывать вторичное освещение и физически достоверные модели материалов и источников света. К настоящему моменту предложены различные подходы для имитации тех или иных эффектов глобального освещения, среди которых следует отметить методы внешней преграды (ambient occlusion), отражающих теневых карт (reflective shadow maps), мгновенной излучательности (instant radiosity), а также методы фотонных карт в пространстве изображения (image space photon mapping). Перечисленные подходы позволяют добиться высокой производительности за счет аппаратной растеризации, однако воспроизводят только часть необходимых эффектов. Интерфейсы 3D графики не поддерживают трассировку лучей, на основе которой реализуются многие методы глобального освещения.

В последние годы появились и активно развиваются универсальные инструменты программирования (NVIDIA CUDA, OpenCL, Compute Shaders), которые позволяют задействовать вычислительные возможности графических ускорителей для широкого класса задач. Современный графический процессор NVIDIA GeForce GTX 680 имеет пиковую производительность свыше 3 Терафлопс, что позволяет говорить об эпохе персональных супервычислений. Основным препятствием на пути к использованию таких ресурсов является высокая сложность разработки для массивно-параллельных графических архитектур, связанная с необходимостью глубокой переработки традиционных алгоритмов.

Одна из актуальных задач современной компьютерной графики – портирование методов глобального освещения на графические процессоры с целью визуализации сложных 3D сцен в интерактивном режиме. Данная область активно исследуется как в России, так и за ее

пределами. Подтверждением этому является значительное число публикаций, среди которых следует отметить работы I.Wald, J. Gunther, S. Popov, H.-P. Seidel, S. Boulos, T. Ize, W. Hunt, C. Benthin, M. Wagner, V. Havran, C. Wachter, A. Keller, C. Lauterbach, T. Purcell, I. Buck, T. Foley, J. Sugerman, D. Horn, N. Thrane, L. Simonsen, W. Mark, M. Houston, P. Hanrahan, H. Jensen, S. Parker, P. Shirley, P. Slusallek, S. Woop, J. Schmittler, D. Hall, T. Aila, S. Laine, K. Zhou, R.Wang, а также Ю. Баяковского, В. Галактионова, А. Волобая, Б. Барладяна, Л. Шапиро, К. Гаранжи, К. Вострякова, А. Игнатенко, А. Адинца, В. Фролова и др. Публикации последних лет показывают устойчивую тенденцию к возрастанию роли графического процессора в расчете глобального освещения. Вместе с тем, остаются актуальными как достижение интерактивности в синтезе изображений 3D сцен (с учетом глобального освещения), так и разработка системного решения для графических процессоров. Поиску данного решения и посвящено настоящее исследование.

Цели и задачи исследования. Целью диссертационного исследования является разработка теоретических основ создания программных систем для информационных технологий виртуальной реальности и синтеза реалистичных изображений сложных трехмерных сцен (содержащих десятки миллионов треугольников и неполигональные объекты) методами глобального освещения на графических процессорах. Поставленная цель требует решения следующих задач:

- Исследовать существующие алгоритмы глобального освещения и ускоряющие структуры трассировки лучей для синтеза изображений сложных сцен, а также возможность их применения на графических процессорах.
- Развить существующие структуры представления компьютерной сцены, чтобы обеспечить эффективную передачу и потоковую обработку данных на графическом процессоре. В том числе, необходимо разработать:
 - Двухуровневое представление сцены, которое на верхнем уровне позволяет работать с иерархической объектно-ориентированной моделью сцены, а на нижнем уровне обеспечивает эффективное преобразование (сериализацию) данных в потоковое представление для передачи на графический процессор.
 - Расширяемую подсистему моделей материалов и источников света, которая обеспечивает Монте-Карло-ориентированное описание оптических свойств поверхностей (BSDF) и излучающих свойств источников.
 - Механизм описания 3D сцены, который позволяет сочетать в одной модели полигональные и нетесселированные объекты (фрагменты сплошных сред, неявно за-

данные и сплайновые поверхности, поверхности второго порядка, фрактальные множества).

- Конвейер трассировки лучей, допускающий конфигурацию для исполнения алгоритмов глобального освещения с разным балансом скорости и качества визуализации при сохранении физической корректности моделей.
- Ускоряющую структуру, которая обеспечивает эффективное параллельное построение на графических процессорах и высокую скорость визуализации.
- Разработать программный комплекс, реализующий предложенные решения с использованием инструментов параллельного программирования графических архитектур (NVIDIA CUDA, OpenCL, OpenGL).
- Экспериментально проверить корректность генерируемых изображений путем воспроизведения различных механизмов переноса световой энергии в сцене и сравнения результатов с эталонными изображениями.
- Оценить (и сравнить с аналогами) быстродействие программного комплекса, реализованных моделей и методов в задачах расчета глобального освещения.

Предметом исследования являются модели, методы и системные решения для интерактивного синтеза фотореалистичных изображений сложных 3D сцен методами глобального освещения; алгоритмизация методов глобального освещения; способы представления, хранения и передачи моделей 3D сцен, допускающие эффективную обработку на GPU; методы распараллеливания алгоритмов глобального освещения на массивно-параллельных GPU; современные технологии объектно-ориентированного программирования и метапрограммирования.

Методы исследования. Решение задач диссертационной работы базируется на теоретических основах информатики, методах компьютерной графики, численных методах, теории вероятностей и математической статистике, теоретических основах аналитической геометрии, теории алгоритмов и структур данных, а также методах параллельных вычислений.

Научная новизна работы. Получены следующие новые результаты в области интерактивного синтеза изображений 3D сцен методами глобального освещения на графическом процессоре:

- 1) Эффективная для статических и динамических сцен ускоряющая структура, на базе иерархии ограничивающих объемов, отличающаяся реализацией алгоритма построения

ния полностью на графическом процессоре (до 5-ти раз быстрее лучшей известной реализации на момент публикации результатов).

- 2) Структура хранения и обработки 3D сцен, отличающаяся наличием двухуровневого представления модели: на верхнем (прикладном) уровне функционирует как граф сцены, а на нижнем обеспечивает эффективную сериализацию данных для потоковой обработки на графическом процессоре.
- 3) Программный конвейер трассировки лучей, который реализует типовые блоки (операции) лучевых алгоритмов визуализации и отличается универсальностью на классе методов глобального освещения и ориентированностью на массивно-параллельные вычислительные архитектуры. На базе блоков данного конвейера построены методы: испускание лучей (ray casting), трассировка лучей Уиттеда (Whitted ray tracing), стохастическая трассировка путей в прямом и обратном направлении (light tracing и path tracing), двунаправленная трассировка путей (bidirectional path tracing), а также предлагаемый в настоящем исследовании ее «усеченный» вариант.
- 4) Подсистема материалов, отличающаяся построением на базе концепции Монте-Карло «атомарных» (событийно-атомарных) материалов, которая обеспечивает компактное описание и эффективную обработку на графических процессорах.
- 5) Метод «усеченной» двунаправленной трассировки путей, который строится на типовых операциях конвейера трассировки лучей и отличается фиксированным объемом потребляемой памяти при обработке путей любой длины, эффективной реализацией многократной выборки по значимости, полным исключением фазы соединения путей.

С точки зрения **практической значимости** интерес представляет разработанный программный комплекс и отдельные его компоненты:

- Высокоуровневая программная библиотека для расчета глобального освещения на GPU, которая может встраиваться в существующие программные решения и применяться для разработки общих и специализированных систем визуализации с заданными свойствами.
- Программная библиотека графа сцены, которая отличается от аналогов (таких как OpenSceneGraph и OpenSG) поддержкой эффективной сериализации данных и гибридных сцен (сочетающих полигональные и нетесселированные объекты).
- Встраиваемый модуль визуализации («плагин») для системы 3D моделирования Blender, который базируется на разработанной библиотеке расчета глобального освещения

(аналогичные модули могут быть разработаны для систем Autodesk 3D Studio Max и Maya, Maxon Cinema 4D, DAZ Studio, Smith Micro Poser).

- Автономная система синтеза изображений методами глобального освещения в интерактивном режиме с поддержкой основных (более 20) форматов хранения геометрии и описания материалов. Система может служить методическим пособием к вузовским курсам по компьютерной графике и физической оптике.

Таким образом, получен ряд практических результатов, востребованных в различных областях компьютерной графики. Программная библиотека лучевых методов синтеза изображений внедрена в комплекс научной визуализации ScView РФЯЦ-ВНИИЭФ. Лабораторные работы по лучевым методам, разработанные в процессе подготовки диссертации, внедрены в учебный процесс ННГУ.

Достоверность результатов подтверждена компьютерными экспериментами по моделированию различных аспектов переноса световой энергии в сцене и сравнением полученных результатов с эталонными изображениями.

Апробация работы. Основные результаты доложены и обсуждены на:

- 19-й международной конференции по компьютерной графике и зрению «GraphiCon-2009», Москва;
- 20-й международной конференции по компьютерной графике и зрению «GraphiCon-2010», Санкт-Петербург;
- 21-й международной конференции по компьютерной графике и зрению «GraphiCon-2011», Москва;
- 22-й международной конференции по компьютерной графике и зрению «GraphiCon-2012», Москва;
- Международной научной конференции «Параллельные Вычислительные Технологии», Нижний Новгород, 2009;
- Всероссийской конференции «Высокопроизводительные параллельные вычисления на кластерных системах», Нижний Новгород, 2011;
- Семинарах кафедры МО ЭВМ факультета ВМК ННГУ.

Лабораторные комплексы по трассировке лучей, глобальному освещению и вычислениям на GPU внедрены в курсы «Компьютерная графика» и «Современная компьютерная графика» на факультете ВМК ННГУ; использованы в образовательных проектах Intel Winter School

2008 и Intel Studio Graphics 2008; цикле Интернет-лекций и мастер-классов по компьютерной графике по гранту ФЦП СПбУ ИТМО (20 часов, ноябрь 2009); всероссийской научной школе для молодежи «Компьютерное зрение, 3D моделирование и компьютерная графика» 2010 (госконтракт №02.741.12.2170). Работа прошла конкурсный отбор для участия в программе У.М.Н.И.К (госконтракт №8753р/13130).

Публикации. Основные результаты исследования опубликованы в 13 работах, из них 4 – публикации в изданиях, рекомендованных ВАК.

Основные положения, выносимые на защиту:

- 1) Эффективная ускоряющая структура на базе иерархии объемов, отличающаяся наличием высокопроизводительного алгоритма ее параллельного построения на графическом процессоре.
- 2) Подход к созданию высокопроизводительных программных систем для синтеза изображений методами глобального освещения, отличающийся специализацией для графических процессоров как наличием конвейера трассировки лучей, так и представлением 3D сцены с помощью специальной технологии «граф сцены – сериализация».
- 3) Новый метод «усеченной» двунаправленной трассировки путей, реализуемый на типовых операциях конвейера, который отличается эффективной реализацией многократной выборки по значимости, фиксированным объемом потребляемой памяти при обработке путей произвольной длины и полным исключением фазы соединения путей.
- 4) Высокоуровневая библиотека для интерактивного синтеза изображений 3D сцен методами глобального освещения на графических процессорах и программный комплекс, реализующий выносимые на защиту положения и обеспечивающий производительность на уровне (а при увеличении глубины трассировки и до 2-х раз выше) библиотеки трассировки лучей NVIDIA OptiX.

Глава 1

Обзор публикаций и программного обеспечения по теме исследования

1.1. Трассировка лучей и глобальное освещение

1.1.1. Лучевые методы синтеза изображения

Трассировка лучей – популярный метод построения изображений компьютерных сцен посредством отслеживания траекторий распространения лучей света и их взаимодействий с поверхностями объектов. Оригинальный вариант метода был впервые сформулирован в 1980 году Т. Уиттедом [1] и оставался предметом активных исследований в течение следующих двадцати лет. Концепция трассировки лучей использовалась и ранее в различных областях науки и техники, однако для компьютерной графики она стала фундаментальной и открыла возможности синтеза реалистичных изображений. В работе Уиттеда лучи использовались не только для определения видимой через каждый пиксель точки (данный этап метода часто называют бросанием лучей), но и для расчета прямой освещенности и эффектов идеального зеркального отражения и преломления. Исходный алгоритм не обеспечивал моделирования глобального освещения, однако стал важной отправной точкой для дальнейших работ в этой области (подробный обзор развития метода изложен в книге [2]).

Для синтеза реалистичного изображения необходимо учитывать вторичное освещение, в результате которого одна поверхность получает свет, отраженный от другой поверхности. Типовыми проявлениями вторичного освещения являются эффекты «переноса» цвета между соседними поверхностями (англ. color bleeding) и каустики (англ. caustics). «Перенос» цвета часто возникает в случае, когда некоторая диффузная поверхность отражает свет на другую диффузную поверхность. При этом отражаемый свет «окрашивается» собственным цветом отражающей поверхности. Каустики наблюдаются как яркие световые кривые различного вида, которые возникают в результате фокусировании света отражающим или прозрачным объектом. Движущиеся каустики можно наблюдать на дне неглубокого бассейна, водная поверхность которого находится в движении. Радуга также служит примером разноцветной каустики, возникающей при преломлении солнечных лучей на дождевых каплях. Очень часто проявления вторичного освещения едва различимы, однако играют важную роль при синтезе реалистичных изображений (рис. 1.1).

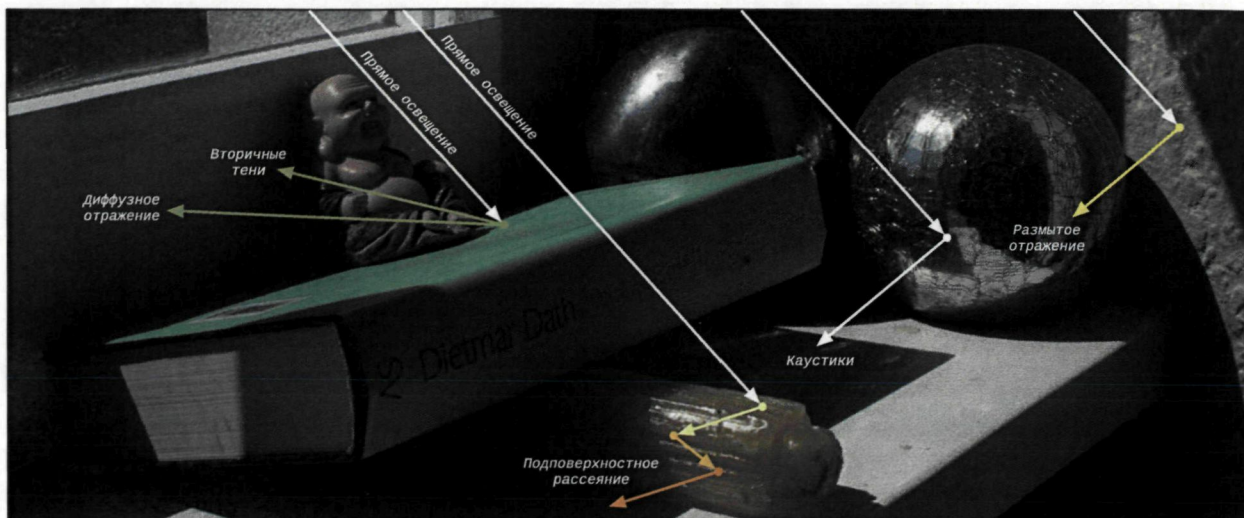


Рис. 1.1. Примеры различных эффектов глобального освещения (из работы [111])

Исследователи обратили внимание на важность столь тонких эффектов довольно рано. Предложенная Р. Куком распределенная трассировка лучей [3] (англ. distributed ray tracing) ввела в процесс расчета генерацию случайных направлений, что позволило учесть некоторые эффекты, включая глубину резкости (ГРИП), размытое изображение движущихся объектов, мягкие тени и нечеткие отражения. Распределенная трассировка лучей впоследствии была расширена и обобщена в методе стохастической трассировки путей [4] (англ. path tracing) Дж. Кайя. Данное обобщение позволило учесть все пути света в сцене за счет построения достаточного числа случайных направлений. Однако сходимость решения может достигаться при большом объеме вычислений. Типичными примерами служат сцены с преобладанием вторичного освещения (перекрытые источники света) или резкими колебаниями светового поля (каустики). В таких случаях крайне трудно построить корректные пути переноса света, выполняя трассировку только от объектива виртуальной камеры. Возможным решением проблемы является предварительная трассировка лучей из источников света отдельно от лучей из объектива камеры и последующая комбинация результатов расчета для создания финального изображения [5]. Данный подход служит основой широкого класса алгоритмов визуализации. В частности, на базе этого принципа строится двунаправленная трассировка путей (англ. bidirectional path tracing), которая была независимо предложена Э. Лафорчуном [6] и Э. Вичем [129]. Другим примером служит предложенный Йенсенем метод фотонных карт (англ. photon mapping), который в настоящее время активно используется для синтеза реалистичных изображений [7; 8]. Методы излучательности (англ. radiosity) также позволяют обрабатывать некоторые вторичные эффекты [9; 10], однако вместо прямого моделирования переноса света рассчитывают освещение путем решения систем линейных уравнений и не обсуждаются в настоящей работе.

1.1.2. Ускоряющая структура

Наиболее трудоемкой операцией в большинстве алгоритмов на базе трассировки лучей является поиск ближайшей точки пересечения луча с объектами сцены. Для эффективного выполнения этой операции предложены различные структуры данных, которые в контексте методов трассировки лучей принято называть ускоряющими структурами (англ. acceleration structure). Алгоритм трассировки практически не меняется при использовании ускоряющей структуры. Перед тестированием луча на пересечение с геометрией сцены определяется его пересечение с ускоряющей структурой (данная операция обычно называется «обходом»). В качестве результата ускоряющая структура возвращает ближайшую область пространства с подмножеством примитивов сцены. Процедура трассировки тестирует луч на пересечение с примитивами данного подмножества. Если точка соударения не обнаружена, то ускоряющая структура запрашивается повторно для получения нового подмножества примитивов. Этот процесс продолжается до тех пор, пока для луча не будет найдено соударение или не будет установлено, что данный луч не пересекает сцену.

Расчет точки пересечения луча с объектами сцены относится к задачам поиска, поэтому все ускоряющие структуры реализуют некоторый механизм пространственной сортировки объектов. С точки зрения базового подхода к сортировке все ускоряющие структуры можно разделить на два типа: разбиения пространства (англ. spatial subdivision) и иерархии объектов (англ. object hierarchy). Данные типы являются двойственными по своей природе. Разбиения пространства позволяют уникально представить каждую точку пространства, однако каждый примитив сцены может перекрываться любым числом ячеек. Иерархии объектов позволяют уникально представить каждый примитив, однако каждая точка сцены может перекрываться любым числом ячеек. Примерами разбиений пространства являются такие структуры, как регулярные [11] или иерархические [12] сетки, октодеревья [13] и k - d деревья [14], которые отличаются степенью регулярности [15]. Напротив, иерархии ограничивающих объемов [16] и их варианты (такие как иерархии ограничивающих интервалов [17; 18]) служат примерами иерархий объектов. Развитие методов трассировки лучей неразрывно связано с развитием ускоряющих структур, алгоритмов их построения и обхода. Подробному обзору ускоряющих структур и их относительной эффективности посвящена работа [19].

В большинстве случаев ускоряющие структуры строятся на этапе препроцессирования сцены, поэтому трудоемкость их построения не учитывается на этапе визуализации. Данный подход допустимо применять только для статических сцен (с неизменной геометрией), во время отображения которых возможны перемещения лишь виртуальной камеры и изменения источников света. Для обработки динамических сцен (с подвижной геометрией) необходимо

выполнять повторное построение или частичное обновление ускоряющей структуры каждый раз при модификации объектов. Исследования показали, что поддержка динамических сцен может быть реализована для всех основных структур, включая регулярные и иерархические сетки, k - d деревья и иерархии ограничивающих объемов [20; 21]. Однако в определенных случаях на анимацию накладываются ограничения, в частности, могут допускаться только иерархические движения примитивов или деформируемые модели. Разработка *эффективных* ускоряющих структур для сложных динамических сцен по-прежнему является актуальной областью исследований.

1.2. Интерактивная трассировка лучей

1.2.1. Параллельная трассировка лучей на пути к интерактивности

Метод трассировки лучей характеризуется высокой трудоемкостью, однако достаточно просто поддается распараллеливанию. Вклад каждого луча в результирующее изображение может вычисляться независимо от остальных лучей, что позволяет задействовать различные схемы декомпозиции вычислений. Большое число работ в области параллельной трассировки лучей систематизировано в публикации [22].

В простейшей параллельной реализации сцена дублируется на каждом вычислительном узле. При таком подходе основной проблемой является грамотная балансировка нагрузки. Если сцену не удастся разместить на одном узле или дублирование данных нежелательно, то возникает дополнительная проблема – распределение геометрии и лучей между отдельными вычислительными узлами. При дублировании данных на всех узлах или при использовании системы с разделяемой памятью балансировка нагрузки является довольно простой задачей. Управляющий процесс разбивает вычисления на отдельные порции, которые соответствуют фрагментам генерируемого изображения, и передает их на обработку свободным клиентам. При одинаковом размере фрагментов возможно нарушение балансировки, поскольку разные фрагменты могут потребовать разного времени на обработку. Типичное решение проблемы состоит в уменьшении размера фрагментов по мере приближения вычислений к концу кадра. Такой механизм балансировки дает хорошие результаты и обсуждается в работе [23]. Без полного дублирования данных параллельная трассировка значительно усложняется. В этом случае геометрия сцены разделяется между несколькими узлами. Если на узле отсутствует геометрия, необходимая для обработки луча, выполняется обмен данными с другими узлами сети. Данный обмен часто сопровождается значительными накладными расходами.

Несмотря на относительную простоту распараллеливания, было создано не так много систем, направленных на трассировку лучей в *интерактивном* режиме. Основная проблема

обусловлена тем, что интерактивный режим работы предполагает минимальные затраты на коммуникации. В результате наращивание числа процессоров далеко не всегда способствует повышению производительности. Первые интерактивные системы трассировки лучей были разработаны на массивно-параллельных суперкомпьютерах с разделяемой памятью. Данные вычислительные платформы сочетали быстрый межпроцессорный механизм коммуникации и высокую пропускную способность памяти. В работе [24] была представлена интерактивная трассировка лучей для простых сцен (описывались конструктивной блочной геометрией) на базе платформы SGI Power Challenge Array. В работе [25] была показана уже полноценная система трассировки лучей (Star-Ray), которая допускала моделирование теней, отражений и преломлений, имела поддержку текстур и сплайновых поверхностей. Система Star-Ray была ориентирована на массивно-параллельные суперкомпьютеры и масштабировалась до 1024-х процессоров. Высокая производительность достигалась посредством оптимизации структур данных под кэш-линии целевой аппаратуры, в качестве которой использовались сервера SGI Origin 2000. Значительное внимание уделялось балансировке нагрузки между процессорами, которая выполнялась за счет разделения набора лучей на порции различного размера.

Альтернативой суперкомпьютерам являются кластеры, объединяющие группу компьютеров стандартными каналами связи. Важными преимуществами данных систем являются доступные технологии построения и возможность экономически эффективного получения высокой производительности. Вместе с тем, стандартные интерфейсы коммуникации имеют высокую латентность и недостаточную пропускную способность, что приводит к падению производительности в определенных задачах. В работе [26] была описана одна из первых систем интерактивной трассировки лучей (RTRT) для кластера из обычных компьютеров. Данная система также характеризуется тщательной оптимизацией под размеры кэш-линий соответствующих процессоров (Intel Pentium III и Intel Pentium IV). Наряду с этим RTRT использует возможности исполнительного модуля SSE (англ. Streaming SIMD Extensions), который впервые появился на процессорах семейства Pentium III и позволяет на аппаратном уровне исполнять набор SIMD инструкций (англ. Single Instruction – Multiple Data). В RTRT несколько лучей группируются вместе, и значительная часть вычислений выполняется на модулях SSE. Система RTRT получила развитие в проекте OpenRT [27], в рамках которого велась разработка технологии распределенной интерактивной трассировки лучей. Данный проект был направлен на различные задачи и методы, включая классическую трассировку лучей, визуализацию сложных массивных сцен и интерактивное моделирование глобального освещения. Авторы показали, что система OpenRT на кластере из обычных компьютеров

способна достигать интерактивного режима для сцен с десятками и сотнями миллионов треугольников.

Следует отметить, что система Star-Ray вскоре также была частично портирована на кластер в работе [28]. При этом аппаратная межпроцессорная коммуникационная сеть архитектуры NUMA была заменена на программный слой распределенной общей памяти (англ. distributed shared memory – DSM). На этапе параллельной визуализации данный механизм позволяет получить доступ к необходимым частям сцены посредством обращения к другим вычислительным узлам через сетевой интерфейс. Организации эффективной распределенной общей памяти посвящена более поздняя работа [29], в рамках которой были получены два основных результата. Во-первых, посредством объединения памяти отдельных узлов удалось добиться визуализации сверхбольших сцен, которые невозможно разместить на одном узле. Во-вторых, использование объединенных вычислительных ресурсов позволило выполнять визуализацию в интерактивном режиме. Таким образом, авторы построили гибридную схему декомпозиции вычислений, при которой отдельные узлы кластера обрабатывают различные фрагменты изображения и хранят различные части сцены. Интерактивного режима удалось добиться за счет того, что в процессе визуализации распределенная память используется только для чтения. Данное обстоятельство позволило отказаться от полнофункциональных механизмов распределенной общей памяти [30; 31] и повысить производительность. Для повышения эффективности доступа к распределенной памяти авторы использовали принцип когерентности: наряду с необходимыми фрагментами сцены подгружались дополнительные «близкие» фрагменты. В работе предложен ряд оптимизаций, которые позволяют повысить вероятность повторного использования подгруженных данных, в результате чего механизм кэширования становится более эффективным.

Среди более поздних работ необходимо отметить систему интерактивной трассировки лучей Manta [32], которая развивается как проект с открытым исходным кодом [33]. Данный проект направлен на разработку универсальной системы, которая позволяет решать разные задачи (включая визуализацию полигональных трехмерных сцен и отображение объемных данных). Система Manta первоначально была ориентирована на рабочие станции и большие системы с разделяемой памятью. В основе Manta лежит настраиваемый распараллеленный конвейер, посредством которого осуществляется планирование задач и синхронизация после каждой стадии работы. На стадии визуализации каждый поток асинхронно выполняет стек визуализации, который состоит из модульных компонент. Гибкость системы достигается за счет использования функций обратного вызова и абстрактных интерфейсов для различных компонент конвейера и стека визуализации. Для достижения высокой производительности

разработчики используют пакетную трассировку, SIMD инструкции и когерентность лучей. Архитектура системы Manta показала хорошую масштабируемость на мультипроцессорных системах с общей памятью. В недавней работе [34] система Manta получила поддержку распределенных вычислений на кластерных системах. В основе данной версии лежит механизм распределенной общей памяти, реализованный на базе работы [29].

1.2.2. Специализированная аппаратура для трассировки лучей

Одним из наиболее ранних примеров специализированной аппаратуры для трассировки лучей является система LINKS-1 [35]. Строго говоря, данная система не являлась специализированной, поскольку состояла из нескольких узлов на базе стандартных процессоров Intel 8086, соединенных между собой сетевым интерфейсом. Тем не менее, данная система была спроектирована специально для задач трассировки лучей и является интересным и важным предшественником систем трассировки для кластеров, о которых ранее шла речь.

Основная часть специализированного аппаратного обеспечения для трассировки лучей, разработанного в последнее десятилетие, была построена на платах объемной визуализации. Ускорители VIRIM [36], VIZARD [37] и VolumePro [38] поддерживали простой алгоритм испускания (бросания) лучей и базовые алгоритмы расчета освещенности. Данные карты не позволяли выполнять общие алгоритмы трассировки лучей, с помощью которых могут быть смоделированы тени, отражения и преломления. Дальнейшее развитие специализированной аппаратуры позволило повысить качество синтезируемых изображений. Ускорители AR250 и AR350 [39] от Advanced Rendering Technologies полностью поддерживали классическую трассировку лучей и обеспечивали моделирование теней, отражений и преломлений. Данные процессоры обрабатывали сцены значительно быстрее традиционного оборудования, однако производительность была далека от реального времени.

Следующим этапом развития специализированной аппаратуры стала система SaarCOR [40]. Процессор SaarCOR не являлся программируемым и обеспечивал аппаратное ускорение только одного этапа трассировки лучей – расчета пересечений. Поскольку этот этап является наиболее ресурсоемким, аппаратура позволила добиться интерактивного режима для сцен средней сложности. При этом ускоритель SaarCOR содержал меньшее число транзисторов, чем графические процессоры того времени, и имел низкую пропускную способность памяти. Следующая версия процессора получила название Ray Processing Unit (RPU) и имела уже программируемую архитектуру [41], которая позволяла задавать нестандартные алгоритмы обработки материалов, геометрии и эффектов освещения. Несмотря на низкую частоту в 66

МГц прототип чипа в большинстве случаев значительно превосходил оптимизированные версии трассировки лучей для центральных и графических процессоров.

Среди недавних аппаратных решений следует отметить ускоритель CausticOne, разработанный компанией Caustic Professional [42]. Данный ускоритель не выполняет всех стадий трассировки лучей и используется в качестве сопроцессора, который работает совместно с обычным центральным или графическим процессором. Сгенерированные лучи передаются для обработки на ускоритель CausticOne, в то время как расчет освещенности выполняется на стороне управляющей системы. Ключевой особенностью данного сопроцессора является эффективная обработка вторичных лучей, которые зачастую не обладают пространственной когерентностью, однако играют важную роль при моделировании реалистичного освещения. Предложенная схема довольно удобна и позволяет реализовать любой алгоритм глобального освещения на базе трассировки лучей. При этом разработчики заявляют о двадцатикратном приросте производительности по сравнению с традиционными реализациями.

Последней разработкой в области специализированной аппаратуры является процессор RayCore, который был создан компанией Siliconarts [43]. Данный чип позволяет в реальном времени обрабатывать достаточно сложные статические и динамические сцены, и нацелен главным образом на рынок игровых консолей и мобильных устройств. Однако для расчета изображения используется трассировка лучей Уиттеда, которая обеспечивает расчет теней, отражений, преломлений и поддерживает текстурирование. Поэтому можно говорить лишь о повышении качества графики по сравнению с традиционным алгоритмом растеризации, но не о фотореалистичной визуализации в реальном времени.

1.2.3. Трассировка лучей на центральном процессоре

Трассировка лучей привлекала внимание разработчиков с первых дней существования персональных компьютеров. За прошедшие десятилетия появилось множество реализаций, некоторые из которых впечатляли своими возможностями. Одна из первых демонстраций трассировки лучей в реальном времени была показана в 1995 году [44]. Позднее появились другие любительские реализации, среди которых следует отметить демонстрацию Heaven Seven [45] и систему тестов RealStorm [46]. Большинство подобных реализаций основано на простых лучевых алгоритмах и геометрических моделях, которые тщательно адаптированы для целевой архитектуры.

К числу первых универсальных систем интерактивной трассировки относится проект OpenRT [47], в рамках которого разрабатывалось альтернативное решение традиционным методам растеризации. Система включала в себя оптимизированный движок трассировки

лучей и интерфейс прикладного программирования OpenRT-API, принцип работы которого аналогичен OpenGL. Хотя движок трассировки лучей был ориентирован на параллельные вычислительные системы, интерактивная визуализация простых сцен достигалась на одной рабочей станции. Высокая производительность обеспечивалась за счет обработки больших пакетов когерентных лучей (до 64-х), что позволило задействовать SIMD инструкции процессора, снизить нагрузку на подсистему памяти и повысить эффективность кэша [48; 49]. Техника пакетной обработки получила развитие в алгоритме многоуровневой трассировки лучей (англ. multi-level ray tracing) [50], который позволяет извлечь наибольшую выгоду от когерентности за счет использования геометрических свойств пирамид (англ. ray frustum) – выпуклых оболочек больших групп лучей. Алгоритм MLRT определяет оптимальную точку входа в k -d дерево для всех лучей внутри пирамиды, сокращая число шагов обхода на $\sim 2/3$ по сравнению с традиционной реализацией. В процессе своей работы алгоритм использует геометрические свойства только самой пирамиды и не зависит от распределения и числа лучей внутри нее. После выбора оптимальной точки входа в k -d дерево выполняется поиск в глубину для вычисления точек пересечения пакета лучей с треугольниками сцены [49]. На основе данного алгоритма была построена реализация простого метода испускания лучей для статических сцен (SAH k -d дерево генерировалось на этапе препроцессирования геометрии). На процессоре Intel Pentium 4 HT 3.2 ГГц система выполняла визуализацию достаточно сложных сцен (до 2×10^6 треугольников) с частотой 15–24 кадров в секунду в разрешении 1024×1024 пикселей. Для наиболее когерентных первичных и теневых лучей (от точечных источников света) производительность возросла в ~ 3 раза по сравнению с обычной пакетной трассировкой. Однако применение данного алгоритма для быстро расходящихся вторичных лучей может оказаться нецелесообразным.

Любая высокопроизводительная реализация трассировки лучей задействует ускоряющие структуры, такие как регулярные и иерархические сетки, k -d деревья и иерархии ограничивающих объемов. Долгое время данные структуры оптимизировались для максимальной скорости визуализации, которая зависит от числа необходимых тестов пересечения. Время построения ускоряющих структур часто игнорировалось, поскольку оно не играет заметной роли в задачах предварительной визуализации (англ. offline rendering). В итоге большинство первых систем интерактивной трассировки лучей поддерживало только статические сцены. Появление интерактивных реализаций привело к пересмотру традиционных ускоряющих структур, время генерации которых уже нельзя было игнорировать. Трассировка лучей стала привлекательной альтернативой алгоритму растеризации, который долгое время был единственно возможным для визуализации динамических сцен в реальном времени.

Среди первых реализаций трассировки лучей для произвольных анимированных сцен следует отметить работу [51]. В качестве ускоряющей структуры применялась регулярная сетка, которая допускала быстрое (за линейное время) перестроение на каждом кадре для достаточно сложных сцен (сотни тысяч треугольников). Для прохода сетки был разработан новый алгоритм, который вобрал ключевые оптимизации, получившие распространение для k - d деревьев: пакетная обработка лучей, ограничивающие пирамиды и SIMD-расширения процессора. По аналогии с работой [50] для большой группы когерентных лучей строится пирамида, которая задается четырьмя граничными плоскостями. В отличие от алгоритма 3DDDA-линии (англ. 3D Digital Differential Analyzer) пирамида проходит сетку по слоям. На каждом слое лучи внутри группы тестируются на пересечение с треугольниками всех ячеек, которые перекрываются с ограничивающей пирамидой (рис. 1.2). При таком подходе лучи группы могут обрабатывать лишние ячейки по сравнению с обычным алгоритмом 3DDDA-линии. Частично проблему удалось решить за счет быстрого отсека треугольников вне пирамиды [52] и идентификации уже протестированных треугольников (англ. mailboxing) [53]. В совокупности обе техники сократили число тестов пересечения в 8–14 раз. В целом производительность нового алгоритма прохода для пакетов размера 8×8 возросла в 7–20 раз по сравнению с алгоритмом 3DDDA-линии.

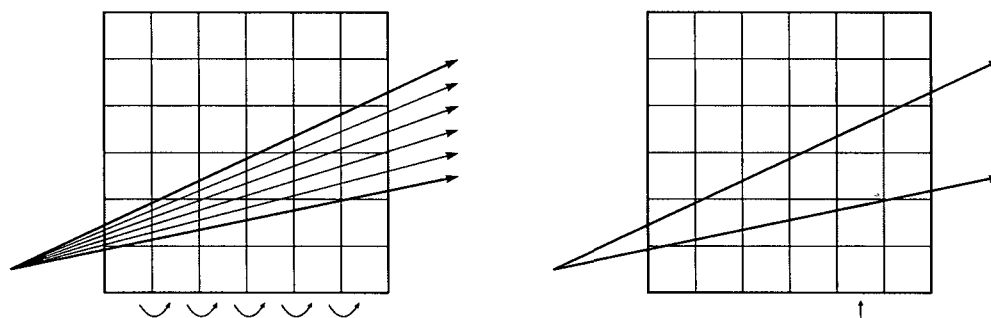


Рис. 1.2. Проход регулярной сетки по слоям пирамидой лучей

Предложенный алгоритм прохода лег в основу реализации испускания лучей. На процессоре Intel Xeon 3.2 ГГц анимированная сцена Fairy Forest (174К треугольников) отображалась с частотой 1–2 кадра в секунду в разрешении 1024×1024 для одного точечного источника света, при этом на построение сетки затрачивалось 68 мс. Несмотря на заметные улучшения, данный подход унаследовал все недостатки ускоряющей структуры, которая не учитывает распределение геометрии сцены, потребляет значительный объем памяти и не обеспечивает высокую производительность трассировки.

Потребность в интерактивной визуализации динамических сцен привела к разработке новых гибридных ускоряющих структур, сочетающих свойства иерархий ограничивающих

объемов и k -d деревьев. В работе [54] предложена так называемая иерархия ограничивающих интервалов (англ. bounding interval hierarchy), которая обеспечивает эффективную обработку динамической геометрии.

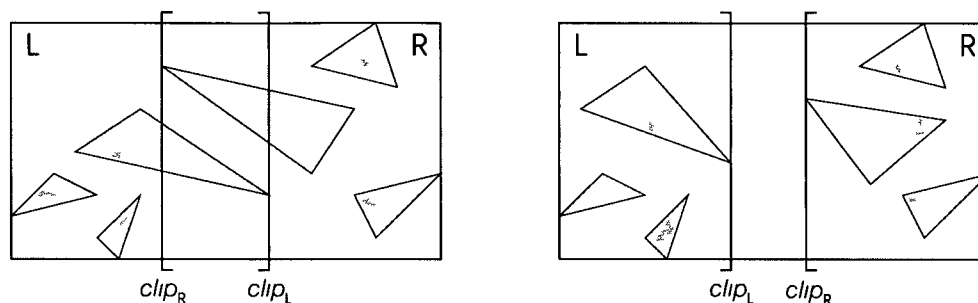


Рис. 1.3. Примеры разбиения узла на два дочерних в структуре ВИН

В отличие от иерархии объемов, данная структура содержит для каждого узла только две (из шести) параллельные плоскости, перпендикулярные оси x , y или z (рис. 1.3). Левый потомок образуется путем замены наибольшей стороны ограничивающего параллелепипеда первой плоскостью, а правый потомок – наименьшей стороны второй плоскостью (порядок сторон определяется осью разбиения). В результате дочерние узлы могут перекрываться (аналогия с иерархиями объемов) и становятся упорядоченными вдоль выбранной оси (аналогия с k -d деревьями). Обход данной структуры может выполняться с помощью обычного алгоритма для иерархии объемов, при этом упорядоченные потомки могут обрабатываться по ходу следования луча как при обходе k -d дерева. Если сегмент луча лежит в пустой области между двух плоскостей, то обработки не требует ни один дочерний узел. Для построения иерархии интервалов был предложен простой алгоритм на основе медианного разбиения пространства. С помощью корзинной сортировки (англ. bucket sorting) за время $O(N \log N)$ треугольники сцены распределяются по вокселям регулярной сетки, по которой за линейное время $O(N)$ строится иерархия интервалов. Такой способ генерации ведет к худшему качеству структуры по сравнению с методами, которые используют эвристику площадей поверхностей. На базе иерархии интервалов была построена реализация классической трассировки лучей, которая поддерживала тени, идеальные отражения и преломления (когерентные лучи обрабатывались пакетами размера 2×2). На процессоре Intel Pentium 4 HT 2.8 ГГц ускоряющая структура строилась со скоростью 1.0–1.3М треугольников в секунду. Визуализация анимированной сцены Fairy Forest (174К треугольников) выполнялась с частотой 1.8–2.0 кадра в секунду в разрешении 640×480 пикселей для одного точечного источника света.

Аналогичные гибридные структуры были независимо разработаны другими исследователями. В работе [55] рассматривается аппаратная реализация так называемых b - k -d деревьев

(англ. bounded k -d tree), каждый узел которых содержит две пары параллельных плоскостей, ограничивающих дочерние вершины. Аппаратная реализация на чипе FPGA обрабатывала простые динамические сцены с частотой до 35 кадров в секунду. В публикации [56] схожая структура получила название DE-дерева (англ. dual extent tree), при построении которого крупные примитивы сохранялись на верхних уровнях иерархии. В работе [57] предложены H-деревья (англ. H-tree), которые дополнили иерархию интервалов специальными узлами в виде ограничивающих параллелепипедов для эффективного отсечения пустого пространства. Вместе с тем, данный прием усложняет построение и обход дерева, исключая возможность унифицированной обработки всех узлов.

В отсутствие универсальных производительных решений исследователи разрабатывали алгоритмы для частных случаев динамических сцен. Значительное внимание уделялось так называемым деформируемым (англ. deformable) моделям, для обработки которых в работах [58; 59; 60; 61] были предложены схожие подходы на основе иерархии ограничивающих объемов. При таком типе движения меняются только координаты вершин полигональной сетки (число треугольников и связи между вершинами остаются неизменными). По сравнению со структурами разбиения пространства (к которым относятся регулярные сетки и k -d деревья) иерархии объемов являются более устойчивыми к малым перемещениям геометрии. Поэтому вместо полного перестроения структуры можно обновить только ограничивающие объемы узлов, сохранив топологию дерева неизменной. Данная операция выполняется за линейное время $O(N)$ посредством обхода всех узлов в обратном порядке. Эффективность модифицированного дерева может уступать полностью перестроенной иерархии, однако оно всегда останется корректным. Такой подход ранее использовался в задаче поиска соударений деформируемых объектов [62; 63] и показал свою состоятельность.

Применение данной техники к задаче трассировки лучей впервые описано в работе [58]. Для построения иерархии объемов использовался простой метод медианного разбиения с трудоемкостью $O(N \log N)$. Отказ от эвристики площадей поверхностей позволил сократить время построения более чем на порядок, при этом скорость визуализации упала на 50–90%. Обработка деформируемых сцен выполнялась за счет обновления дерева на каждом кадре, однако при значительных изменениях геометрии структуру необходимо строить заново. Для выбора подходящего момента перестроения в работе предложен критерий, в основе которого лежит отношение площади узла к суммарной площади его потомков. Сумма отношений по всем узлам дерева показывает, насколько «плотно» иерархия аппроксимирует геометрию сцены (эталонное значение соответствует последнему построенному дереву). В результате достигается баланс между временем перестроения ускоряющей структуры и временем ви-

зуализации, что значительно сокращает *среднее* время обработки одного кадра. На ЦП Intel Pentium 4 HT 2.8 ГГц система выполняла визуализацию простых деформируемых сцен (до ста тысяч треугольников) со средней частотой 6–13 кадров в секунду в разрешении 512×512 пикселей, при этом на обновление иерархии объемов затрачивалось 4–23 мс.

Минимизация среднего времени обработки кадра целесообразна для предварительной визуализации, при которой редкие трудоемкие перестроения усредняются по всем кадрам анимации. В интерактивных приложениях данный подход не работает, поскольку в момент перестроения пользователь не сможет взаимодействовать с виртуальной сценой. Для равномерного распределения трудоемких операций построения по кадрам анимации в работе [59] предложена схема выборочного перестроения иерархии объемов. На каждом кадре после обновления дерева оценивается степень деградации отдельных поддеревьев, которые при необходимости перестраиваются с помощью алгоритма из работы [58]. Для выявления таких поддеревьев авторы предложили две метрики. Первая базируется на эвристике площадей поверхностей и оценивает эффективность отсечения (англ. *culling efficiency*) геометрии на пути следования луча. Вторая позволяет оценить возможную пользу от перестроения дерева (англ. *restructuring benefit*) в терминах роста эффективности отсечения. На тестовых сценах такой метод позволил сократить общее время обработки анимационной последовательности до 2.5 раз по сравнению с публикацией [58].

Работа [60] направлена на повышение быстродействия визуализации деформируемых моделей. Для построения иерархии объемов был предложен оптимизированный алгоритм на основе эвристики площадей поверхностей. Треугольники сцены заменялись центрами своих ограничивающих параллелепипедов. Разбиение вокселя выбиралось из плоскостей, которые проходят через указанные центры ортогонально некоторой оси системы координат. Данный алгоритм не выполняет поиск оптимальных разбиений (список треугольников можно разбить на две части $O(2^N)$ способами), но обладает умеренной трудоемкостью $O(N \log^2 N)$. За счет оценки SAH-стоимости скорость трассировки возросла в 2 раза по сравнению с медианным разбиением пространства и в 6 раз по сравнению с медианным разбиением объектов. Наряду с этим в работе были предложены приемы и оптимизации для обработки больших пакетов лучей (16×16). Для популярной сцены Fairy Forest (174К треугольников) на процессоре AMD Opteron 2.6 ГГц время построения и обновления иерархии составило 3.2 секунды и 13 миллисекунд соответственно. За счет быстрого обновления данная сцена визуализировалась с частотой ~ 3.7 кадров в секунду в разрешении 1024×1024 с учетом теней (от точечного источника света).

В работе [61] для иерархии объемов была предложена асинхронная схема построения (параллельно с визуализацией сцены). В промежутках ожидания перестроенной структуры выполнялось обновление существующей иерархии. Данная техника позволила в несколько раз повысить производительность по сравнению с перестроением дерева на каждом кадре и избежать периодических «подвисаний» системы, характерных для метода из работы [58]. Однако, если перестроение занимает много времени, или сцена подвергается значительным изменениям, скорость обработки может существенно снижаться. Чтобы избежать сильного падения производительности во время ожидания новой структуры, в работе предложено оценивать эффективность иерархии объемов и при необходимости переключаться на быстрый алгоритм построения через медианные разбиения пространства. На рабочей станции с четырьмя двудерными процессорами AMD Opteron 2 ГГц данный подход позволил визуализировать анимированную сцену типа Fairy Forest (394К треугольников) с частотой 19–23 кадров в секунду в разрешении 1024×1024 с простым затенением (источник света в камере).

Для обработки деформируемых моделей исследователи предлагали подходы и на основе k -d деревьев. Выполнять стандартное построение k -d дерева на каждом кадре затруднительно, поскольку даже эффективный алгоритм [64] с трудоемкостью $O(N \log N)$ может потребовать секунды и минуты для достаточно сложных сцен. В публикации [65] предложен алгоритм обработки частного типа деформируемых сцен, для которых последовательность ключевых кадров задана заранее. Высокопроизводительная обработка таких сцен основана на двух концепциях: декомпозиция движения (англ. motion decomposition) и «размытые» k -d деревья (англ. fuzzy k -d tree). Для заданной последовательности ключевых кадров выполняется декомпозиция исходной полигональной сетки на кластеры с когерентным движением. На каждом ключевом кадре для каждой подсетки методом наименьших квадратов строится аффинная аппроксимация движения. Вычитание аффинной составляющей из исходного движения определяет локальное пространство подсетки, в котором остаточное движение вершин оказывается незначительным. Для каждого треугольника в локальном пространстве подсетки строится параллелепипед, который содержит возможные положения вершин во время анимации. «Размытые» k -d деревья формируются в локальном пространстве каждого кластера, при этом вместо треугольников используются построенные параллелепипеды. Для организации отдельных «размытых» k -d деревьев в ускоряющую структуру сцены на каждом кадре строится k -d дерево верхнего уровня для ограничивающих параллелепипедов подсеток. Поиск соударения луча начинается с обхода высокоуровневого дерева и продолжается в локальной системе координат пересекаемого кластера. При соударении с листовой вершиной «размытого» k -d дерева выполняется тест на пересечение луча с текущим экземпляром внут-

ренного треугольника. Предложенный подход обеспечил визуализацию простых деформируемых сцен (менее ста тысяч треугольников) в разрешении 1024×1024 пикселей с частотой 5–15 кадров в секунду на одном процессоре (модель не уточняется).

Рассмотренные решения на основе иерархий ограничивающих объемов и k -d деревьев обеспечили обработку деформируемых сцен, которым свойственны малые и когерентные перемещения. Однако в большинстве динамических сцен присутствуют объекты, движение которых имеет произвольный характер. Топология геометрической модели подвергается сильным изменениям в результате взрыва, разрезания, разрывания и разрушения. Данные сценарии часто реализуются в таких приложениях, как компьютерные игры, тренажеры и симуляторы. Универсальный способ обработки произвольных интерактивных окружений (с переменным числом объектов и произвольными перемещениями) состоит в перестроении ускоряющей структуры на каждом кадре.

Для повышения скорости формирования структур различными исследователями были предложены ячеечные (англ. binned) техники, которые обеспечивают разумный компромисс между временем построения и быстродействием трассировки. Изначально данные методы применялись для генерации k -d деревьев и впервые были описаны в работах [66; 67; 68], в основе которых лежали следующие общие принципы. Во-первых, игнорируются «идеальные разбиения» (англ. perfect splits), для обработки которых треугольники сцены предварительно отсекаются границами разбиваемого вокселя. Вместо этого используются ограничивающие параллелепипеды неотсеченных треугольников, что позволяет отказаться от ресурсоемкого теста пересечения треугольника с разделяющей плоскостью. Во-вторых, секущая плоскость выбирается среди K плоскостей, которые равномерно разбивают текущий воксель на $K + 1$ ячейку (корзину). На *первом* проходе подсчитывается число треугольников в каждой ячейке вокселя. Для каждой из K плоскостей вычисляется SAH-стоимость разбиения, и выбирается плоскость с наименьшей стоимостью (рис. 1.4).

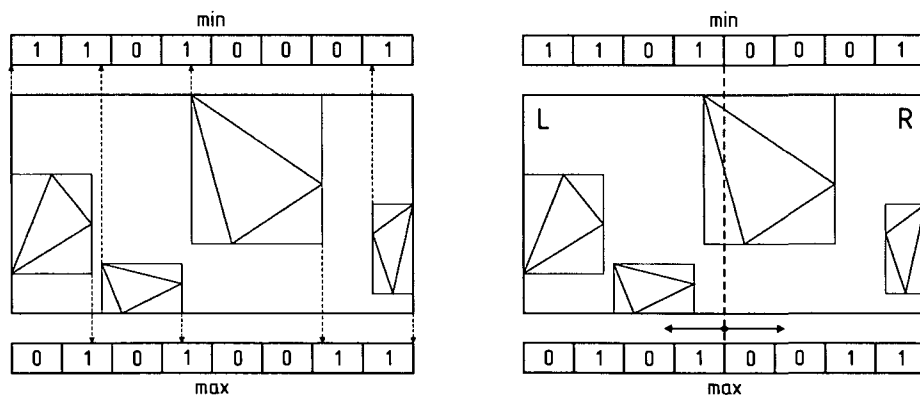


Рис. 1.4. Пример работы ячеечного алгоритма *min-max* для разбиения узла k -d дерева

На *втором* проходе формируются списки треугольников для левого и правого поддеревья, которые находятся соответственно слева и справа от выбранной секущей плоскости. Данный алгоритм значительно сокращает число вариантов разбиения (K плоскостей вместо $O(N)$ сторон ограничивающих параллелепипедов) и исключает сортировку тестовых плоскостей. Таким образом, разбиение узла выполняется за два быстрых прохода с трудоемкостью $O(N)$.

Хотя базовые концепции остаются неизменными для всех указанных работ, некоторые исследователи предложили дополнительные оптимизации и упрощения. Во-первых, процедура вычисления SAH-стоимости может быть упрощена за счет исключения членов, общих для всех плоскостей разбиения [66]. Во-вторых, выражение SAH-стоимости может быть аппроксимировано кусочно-квадратичной функцией, минимум которой уточняет положение секущей плоскости [67]. В-третьих, разбиение может выполняться только для оси, вдоль которой протяженность вокселя максимальна [67] (на тестовых сценах падение производительности не превышало 7%). В-четвертых, для каждого треугольника можно отмечать только начальную и конечную ячейку (англ. min-max binning), что позволяет точно определить число примитивов по обе стороны любой плоскости разбиения (независимо от размера треугольников) [66; 68]. В-пятых, вместо проектирования всех треугольников на ячейки вокселя можно ограничиться их подмножеством (или использовать менее детализированную модель сцены) [68]. В-шестых, если число треугольников вокселя оказывается меньше числа ячеек, выполняется стандартная процедура разбиения с помощью алгоритма подвижной плоскости (англ. plane sweep) [66].

В дополнение к перечисленным алгоритмическим изменениям в работах применяется тщательная программная оптимизация. Высокая производительность обеспечивается за счет оптимизированных структур хранения, работы с памятью и аппаратных SIMD-инструкций. Для использования ресурсов многоядерных процессоров в работах [66] и [68] предложены параллельные процедуры построения k -d дерева. В частности, в публикации [68] приводится схема разбиения ограничивающего параллелепипеда сцены на непересекающиеся области (на основе медианного разбиения списка примитивов). В полученных областях выполняется параллельная генерация k -d деревьев на множестве потоков с учетом балансировки нагрузки. После чего отдельные деревья объединяются в единое k -d дерево сцены. Данный подход обеспечивает хорошую масштабируемость (алгоритм тестировался на 4-х потоках), однако снижает производительность трассировки, поскольку на верхних уровнях не используется эвристика площадей поверхностей. С учетом указанных оптимизаций в работах [66] и [67] производительность построения k -d дерева достигает 300–400К треугольников в секунду, а в

работе [68] – 2М треугольников в секунду. В результате стала возможной интерактивная генерация SAH k -d деревьев для сцен с невысокой геометрической сложностью.

Поскольку ячеечные алгоритмы основаны на дискретизации функции SAH-стоимости, секущие плоскости не всегда выбираются оптимально. Вместе с тем, перечисленные работы сообщают о достаточно высоком качестве k -d деревьев даже при небольшом числе ячеек на воксель (в работе [67] реализована двухшаговая адаптивная ячеечная схема с 8 корзинами). Одна из основных причин уменьшения производительности (по сравнению со стандартным алгоритмом построения) связана с игнорированием «идеальных разбиений». В публикации [69] сообщается, что данная техника позволяет повысить быстродействие на 10–35%.

Аналогичные ячеечные алгоритмы могут использоваться и для построения иерархий ограничивающих объемов, некоторые особенности которых удачно сочетаются с данными техниками. Во-первых, иерархии объемов содержат значительно меньше узлов по сравнению с k -d деревьями, поэтому процедура построения выполняет меньше операций. Во-вторых, каждый треугольник сцены включается в дерево только один раз, поэтому общее число узлов ограничено величиной $2N - 1$ (где N – число треугольников). В-третьих, треугольники сцены включаются в узлы иерархии целиком, что исключает проблему «идеальных разбиений» (характерную для k -d деревьев). В-четвертых, алгоритмы построения иерархии объемов определяют положение треугольника относительно «секущей» плоскости по одной точке (обычно используется центроид треугольника или его ограничивающего параллелепипеда). Поэтому проектирование треугольников на корзины вокселя также упрощается, поскольку для каждого примитива нужно обновлять только одну ячейку.

Схожие ячеечные методы построения иерархии объемов были независимо предложены в работах [70] и [71]. Схема построения повторяет версию для k -d деревьев с минимальными изменениями (рис. 1.5). Для распределения треугольников сцены по ячейкам используются центроиды ограничивающих параллелепипедов. Поэтому (вместо исходного вокселя) на K равных ячеек разбивается параллелепипед, содержащий все центроиды примитивов (такой прием позволяет уплотнить ячейки за счет исключения пустого пространства). В случае иерархии объемов положение «секущей» плоскости уже не позволяет вычислить площади двух потомков, поскольку размеры дочерних узлов могут произвольно сокращаться вдоль любой оси. Поэтому наряду с числом треугольников каждая «корзина» должна содержать параллелепипед, который ограничивает все треугольники соответствующей ячейки. Размеры данных параллелепипедов наращиваются в процессе распределения примитивов по ячейкам. После проектирования треугольников выбирается плоскость с наименьшей SAH-стоимостью разбиения, и формируются потомки узла.

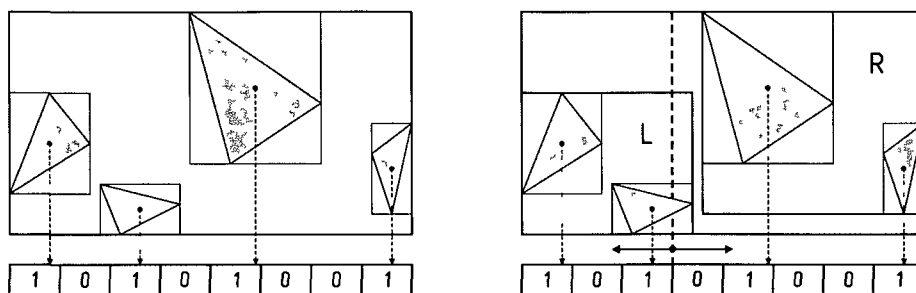


Рис. 1.5. Пример работы ячеечного алгоритма для разбиения узла BVH дерева

Ключевым параметром данных алгоритмов является число корзин на воксель, которое определяет баланс между трудоемкостью построения и эффективностью иерархии объемов. В работе [70] число корзин выбиралось пропорционально числу треугольников в диапазоне 4–32 (быстрый режим) или 8–128 (качественный режим). В публикации [71] каждый воксель разбивался ровно на 16 «корзин». Обе реализации обеспечили высокую скорость построения иерархии – 1.3–2.3М треугольников в секунду на одиночном ядре. При этом эффективность ускоряющих структур отвечала «аккуратным» SAH-деревьям из работы [60] (сравнивалась скорость трассировки лучей). Алгоритм [71] получил дальнейшее развитие в работе [72], где был адаптирован для параллельного построения иерархии объемов. На рабочей станции из 8-ми двоядерных процессоров AMD Opteron 3 ГГц данный алгоритм генерировал иерархию со скоростью ~5М треугольников в секунду (на 8 потоках).

Таким образом, возможность интерактивной трассировки лучей для динамических сцен была показана для всех основных структур, включая регулярные и иерархические сетки, k -d деревья, иерархии ограничивающих объемов и гибридные структуры (такие как s - k -d деревья и иерархии интервалов). Некоторые решения были ориентированы на частные типы динамических сцен, примерами которых являются иерархические движения примитивов и деформируемые модели. Для обработки таких сцен ускоряющая структура формируется с учетом возможных положений примитивов («размытые» k -d деревья) или адаптируется к деформациям модели (иерархии объемов). Наряду с этим были предложены универсальные подходы, которые выполняют полное перестроение ускоряющей структуры на каждом кадре. Среди таких методов высокую эффективность показали ячеечные техники, которые обеспечивают интерактивную генерацию ускоряющих структур и сохраняют принципы, лежащие в основе эвристики площадей поверхностей (SAH).

Несмотря на возможность обработки динамических сцен, большинство реализаций трассировки лучей генерировали изображения *низкого* качества, ограничиваясь обработкой только первичных и теневых лучей от точечных источников. Высокая производительность визуализации достигалась за счет предельного сокращения возможностей метода, который

терял свои основные преимущества перед аппаратно ускоренной растеризацией. Во многом данные ограничения были обусловлены специальными техниками, которые применялись для трассировки первичных и теневых лучей (от точечных источников света). Благодаря высокой когерентности (близости начальных точек и направлений) такие лучи могут обрабатываться большими группами, что значительно сокращает стоимость трассировки. Однако для получения реалистичного результата алгоритмы визуализации должны испускать большое число вторичных лучей, обработка которых занимает основное время расчета. Когерентность вторичных лучей оказывается значительно ниже и может отсутствовать вовсе. Более того, для синтеза реалистичных изображений часто используется модель камеры с протяженной диафрагмой и тонкой линзой (и более сложные модели), которые снижают когерентность даже первичных лучей. В итоге выигрыш в производительности от использования больших пакетов и ограничивающих пирамид может значительно снижаться или исчезать полностью. В этом смысле работы, направленные на максимально эффективную обработку первичных лучей, изначально преследовали неправильную цель.

Потенциал от использования пакетной обработки лучей для моделирования некоторых реалистичных эффектов впервые был исследован в работе [73]. Для построения изображения применялась классическая трассировка лучей Уиттеда (поддерживает идеальные отражения и преломления) и распределенная трассировка лучей (поддерживает мягкие тени, нечеткие отражения, глубину резкости и размытость движущихся объектов). Для ускорения поиска пересечений использовалась иерархия объемов и методы обработки больших пакетов лучей из работы [60]. Вторичные лучи, которые были получены в результате трассировки группы лучей, объединялись в новые пакеты по типу (теневые, отраженные и преломленные). Такой подход позволил извлечь некоторую пользу из остаточной когерентности вторичных лучей. Для пакетов размера 2×2 (так называемые SIMD-лучи) производительность возросла в 1.8–2 раза за счет использования аппаратных 4-х канальных SIMD-инструкций процессора (SSE). Пакеты большего размера (8×8 и 16×16) позволили увеличить производительность еще в ~ 1.5 раза. Тем не менее, по сравнению с первичными лучами производительность упала в 1.7–3.7 раз для вторичных лучей различного типа и составила 0.8–2М лучей в секунду на одном ядре процессора AMD Opteron 2.4 ГГц.

В работе [74] для повышения когерентности вторичных лучей применялись различные эвристики переупорядочивания. Авторами был предложен оригинальный метод организации вычислений: вторичные лучи накапливаются в специальном буфере, переупорядочиваются согласно некоторой эвристике и объединяются в когерентные пакеты (размера 4×4). При таком подходе в одном пакете могут оказаться лучи с разной «глубины» трассировки (спустя

различное число отражений), что обеспечивает широкие возможности сборки когерентных пакетов. Для сортировки применялись различные эвристики, которые группируют лучи по направлениям, начальным точкам или обоим признакам сразу. В ходе эксперимента ни одна эвристика не обеспечила преимущества над стандартным алгоритмом пакетной обработки, который выполняет простое маскирование неактивных лучей (такой метод использовался в высокопроизводительных системах [75; 76]). Данное исследование показало, что простые (и вычислительно эффективные) эвристики не позволяют формировать группы лучей, которые *будут* иметь схожие пути обхода ускоряющей структуры (применялось SAH k -d дерево).

Снижение производительности пакетной трассировки вызвано неактивными лучами, которые приводят к лишним операциям обхода. Повышение эффективности достигается за счет фильтрации пакетов – отключения или удаления неактивных лучей. Для одного SIMD-пакета потеря когерентности не является критичной, хотя приводит к снижению утилизации блоков SIMD. При обработке больших пакетов последствия принимают другой масштаб – значительное число неактивных лучей приводит к худшей производительности по сравнению с SIMD-лучами. В работах [77] и [78] предлагаются различные техники фильтрации пакетов, способные повысить быстродействие обработки вторичных лучей. В качестве ускоряющей структуры обе реализации используют иерархию ограничивающих объемов.

В работе [77] для исключения неактивных лучей использовалась «точная» фильтрация: для каждого узла дерева на пути обхода проверяется пересечение со всеми лучами пакета (с исходным размером 16×16). Активные лучи группируются в новый пакет (перемещаются в начало текущего пакета), который используется при обходе дочерних узлов вокселя. По мере продвижения «вглубь» дерева список активных лучей сокращается. В итоге при обработке листовых узлов с треугольниками пересекаются только активные лучи пакета. По сравнению с алгоритмом [60] данная процедура выполняет больше тестов соударения с вокселями (на верхних уровнях), однако значительно сокращает число тестов соударения с треугольниками (на нижних уровнях). В результате исходный алгоритм оптимально подходит для когерентных пакетов первичных и теневого лучей, а модифицированный – для расходящихся вторичных лучей. На базе нового решения была построена реализация классической трассировки лучей. Преимущество модифицированного алгоритма проявилось только на сценах со сложными многократными отражениями и преломлениями, для которых производительность возросла до 3.2–3.5 раз по сравнению с алгоритмом [60]. В абсолютном выражении быстродействие достигало 4–11 кадров в секунду в разрешении 1024×1024 пикселей на простых сценах (до 200 тысяч треугольников) и двух четырехъядерных процессорах Intel Xeon 2 ГГц (всего 8 ядер). Вероятно, в данной работе была представлена первая достаточно общая реализация

трассировки лучей Уиттеда, которая выполняется в реальном времени на доступной рабочей станции.

В публикации [78] техника фильтрации применялась к широкому классу вторичных лучей, включая размытые и диффузные отражения (имеют важное значение в глобальном освещении). В отличие от предыдущей работы переупорядочивание лучей выполнялось только в случае, если «утилизация» пакета (доля активных лучей) падала ниже некоторого порога (использовалось значение 50%). Такой подход позволяет сбалансировать затраты на переупорядочивание и потери от низкой утилизации блоков SIMD. Данный алгоритм был реализован на базе системы высокопроизводительной трассировки лучей Manta. Наилучшие результаты были получены для зеркальных отражений: после пятикратных отскоков прирост достигал 1.1–1.3 раз по сравнению с SIMD-лучами. Для диффузных и размытых отражений выигрыш оказался значительно скромнее: после двукратных отскоков прирост не превышал 1.3–1.4 раз. С ростом глубины трассировки преимущество метода полностью исчезало.

Таким образом, на некогерентных вторичных лучах методы пакетной трассировки не обеспечивают большого выигрыша в производительности. За счет аппаратных 4-канальных SIMD-инструкций пакеты размера 2×2 позволили повысить скорость визуализации до 2 раз. Пакеты большего размера обеспечили дополнительный прирост не более чем в 1.3–1.5 раз. И только в определенных случаях для когерентных зеркально отраженных лучей наблюдался 3-х кратный прирост производительности. Для сравнения относительно простой алгоритм пакетной трассировки [60] для первичных лучей достигал более чем 10-кратного прироста. Таким образом, в задачах синтеза реалистичных изображений пакетные техники оказались существенно менее эффективными. Для решения подобных задач необходимо использовать массивно-параллельные архитектуры с производительностью другого порядка. Подходящей платформой для дальнейшего развития стали программируемые графические процессоры.

1.2.4. Трассировка лучей на программируемой графической аппаратуре

За последнее десятилетие графическая аппаратура прошла путь от специализированных 3D ускорителей до программируемых массивно-параллельных процессоров, которые имеют высокую производительность и пропускную способность памяти и востребованы в решении целого ряда вычислительно трудоемких задач. С самого начала особенностью графических ускорителей являлась параллельная обработка данных и наличие графического конвейера. Такой способ построения изображения хорошо укладывается в модель потоковой обработки данных (англ. stream processing), которая стала ключевой в вычислительных алгоритмах для графической аппаратуры. В основе данной модели лежит понятие потока (англ. stream) как

последовательности элементов одного типа. Примером потока служит последовательность вершин геометрических примитивов, которые передаются на графический ускоритель, и последовательность фрагментов, которые генерируются на этапе растеризации. Обработка одного или нескольких потоков выполняется специальной функцией – ядром (англ. kernel). На каждом шаге ядро извлекает по одному элементу из входных потоков, обрабатывает их и записывает по одному элементу в каждый выходной поток. Важной особенностью потоковой модели является независимая обработка элементов потока. Данное обстоятельство позволяет выполнять вычисления параллельно на множестве независимых процессоров.

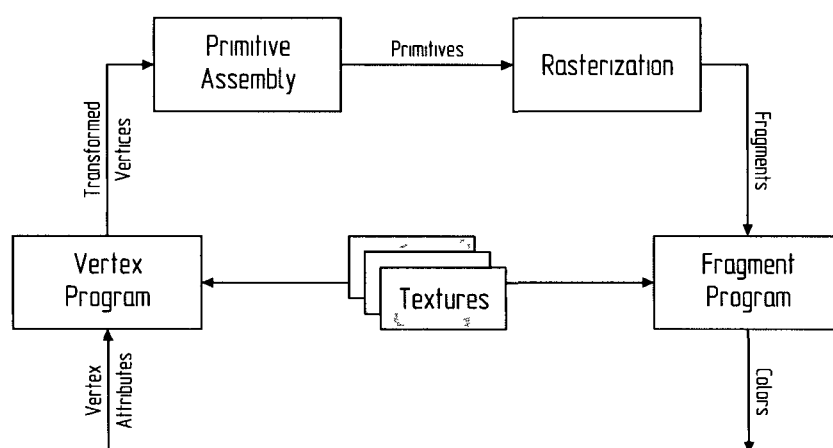


Рис. 1.6. Упрощенная схема графического конвейера

Сложные вычисления в потоковой модели строятся путем последовательного соединения нескольких ядер обработки. Примером такого соединения служит сам графический конвейер (рис. 1.6). На каждом этапе конвейера обработка данных реализована параллельно, что ведет к высокой эффективности использования аппаратуры.

Модель потоковой обработки естественным образом применялась для вычислений на первом программируемом графическом оборудовании¹. Ядра отображались на фрагментные шейдеры – специализированные программы, исполняемые непосредственно на графическом процессоре и предназначенные для независимой обработки пикселей изображения. Потоки данных отображались на текстуры, которые использовались в качестве обычных массивов вещественных чисел с плавающей точкой. Кроме того, графические ускорители допускали чтение данных из текстур по вычисляемому адресу, что позволяло их использовать во время исполнения ядра как константную память общего назначения.

Первая реализация трассировки лучей на программируемой графической аппаратуре была предложена в работе [79] и базировалась на потоковой модели вычислений. Алгоритм

¹ К числу первых графических процессоров с поддержкой OpenGL 2.0 и DirectX 9.0 относятся ATI Radeon 9700 (чип R300) и NVIDIA GeForce FX 5800 (чип NV30), которые были выпущены в 2002 и 2003 году соответственно.

трассировки лучей был представлен последовательностью ядер, соединенных между собой потоками данных (рис. 1.7). Потоковая схема трассировки содержала четыре ядра: генерация первичных лучей, обход ускоряющей структуры, пересечение луча с примитивами и расчет освещенности.

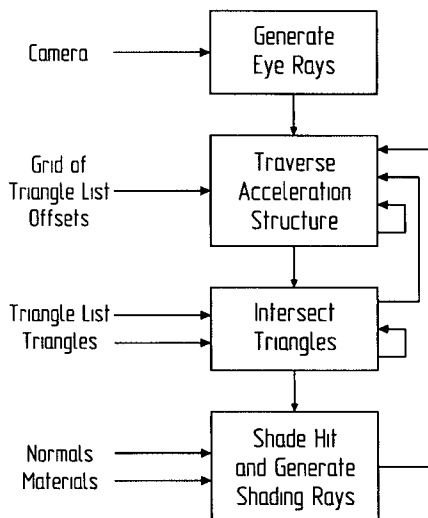


Рис. 1.7. Схема потоковой трассировки лучей

Такой набор ядер был обусловлен ограничениями первых программируемых графических процессоров, которые не поддерживали динамических ветвлений во фрагментных шейдерах. В качестве ускоряющей структуры использовалась регулярная сетка, которая имеет простой (нерекурсивный) алгоритм прохода на основе 3DDDA-линии и естественно отображается на трехмерную текстуру (рис. 1.8).

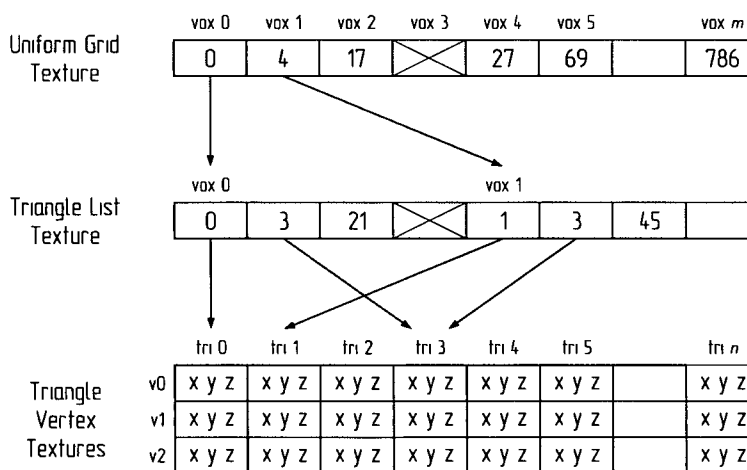


Рис. 1.8. Отображение регулярной сетки и списка треугольников на текстуры

Ядро генерации первичных лучей принимает на вход параметры камеры и координаты пикселей экранной плоскости и порождает поток первичных лучей, которые поступают на

вход ядра обхода ускоряющей структуры. Для каждого сгенерированного луча данное ядро выполняет проход регулярной сетки до пересечения с вокселем, содержащим непустой список треугольников. Луч и воксель передаются на вход ядра поиска точек соударения, которое тестирует луч на пересечение со всеми треугольниками соответствующего вокселя. Если соударение с треугольником обнаружено внутри вокселя, то луч и треугольник поступают на вход следующего ядра, которое вычисляет освещенность в точке пересечения. В противном случае луч поступает на вход ядра обхода ускоряющей структуры для поиска следующих вокселей с непустым списком треугольников. Ядро расчета освещенности вычисляет вклад в яркость соответствующего пикселя картинной плоскости. Если луч завершается в данной точке, то цвет заносится в результирующее изображение. Ядро расчета освещенности может генерировать дополнительные теньевые или вторичные лучи, которые передаются на вход ядра обхода ускоряющей структуры.

Рассмотренные ядра реализуются с помощью отдельных фрагментных программ. Для запуска вычислений устанавливается ортогональная проекция и выполняется рисование прямоугольника, который полностью заполняет окно просмотра. Входные потоки данных извлекаются из двумерных текстур, тексели которых взаимно однозначно отображаются на фрагменты генерируемого изображения. После завершения ядер выходные потоки данных также записываются в текстуры. Установка корректного теста трафарета позволяет задать для каждого ядра текстуры и пиксели, подлежащие обработке на данном проходе. Для этого каждому лучу приписывается 8-битное значение трафарета, указывающее на его состояние: «проход сетки», «пересечение», «освещение» и «завершен».

Для анализа потокового алгоритма трассировки лучей применялся симулятор, который был разработан на языке C++ и обеспечивал высокоуровневое моделирование графических архитектур следующего поколения. В работе дается сравнение двух типов архитектур:

- Многопроходная архитектура, в рамках которой ветвления и циклы организуются посредством нескольких проходов визуализации.
- Архитектура с динамическими ветвлениями, которая имеет поддержку инструкций для организации циклов и управления потоком.

Результаты исследования алгоритма на симуляторе показали: архитектура с динамическими ветвлениями значительно снижает требования к пропускной способности памяти, при этом скорость работы алгоритма ограничивается вычислительными возможностями аппаратуры. Доступное на тот момент графическое оборудование не поддерживало минимального набора необходимых функций, поэтому авторы не привели результатов экспериментов на реальной

аппаратуре. Два года спустя многопроходная версия была протестирована на графическом процессоре NVIDIA GeForce FX 5900 Ultra² и достигла производительности порядка 125К лучей в секунду.

В более поздней работе [80] потоковый алгоритм трассировки лучей был расширен для поддержки глобального освещения. В основу системы положен модифицированный метод фотонных карт [81], все этапы работы которого выполняются на графическом процессоре (рис. 1.9). В традиционном варианте метода для отыскания фотонов, близких к заданной точке сцены, используется сбалансированное k -d дерево. Эффективное построение данной структуры возможно при наличии операции записи по вычисляемому адресу, которая не поддерживалась доступной на тот момент графической аппаратурой³. Поэтому для хранения фотонной карты применялась регулярная сетка, которую можно построить непосредственно на графическом процессоре.

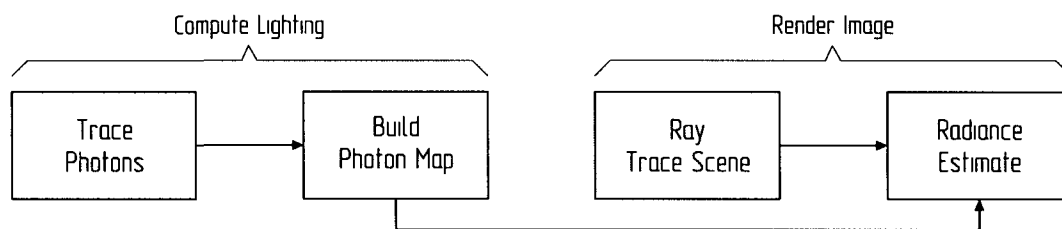


Рис. 1.9. Укрупненная схема визуализации на основе метода фотонных карт

Для поиска k ближайших фотонов для заданной точки сцены применялся адаптированный алгоритм Элиаса [82], который перебирает воксели сетки вокруг точки с последовательным увеличением радиуса поиска. Разработанный алгоритм сбора освещенности оказался весьма ресурсоемким, поэтому обработка даже простейших сцен занимала несколько минут. Вместе с тем, исследование показало принципиальную возможность расчета глобального освещения на программируемой графической аппаратуре.

Впоследствии появились другие исследования, в основе которых лежала потоковая схема трассировки лучей. Значительное внимание уделялось альтернативным ускоряющим структурам (включая k -d деревья и иерархии объемов) и повышению производительности трассировки лучей. В работе [83] регулярная сетка вокселей была дополнена картой близости (англ. proximity cloud), которая для каждой ячейки хранит расстояние (в системе координат сетки) до ближайшей ячейки с примитивами сцены [84]. Такой прием позволяет пропускать участки пространства с пустыми вокселями, однако влечет дополнительные затраты памяти и усложнение процедуры прохода (рис. 1.10).

² Имеет 3 вершинных и 4 пиксельных процессора с частотой 450 МГц. Полоса пропускания памяти – 27.2 Гб/с.

³ ATI Radeon 9800 Pro и NVIDIA GeForce FX 5900 Ultra.

Модифицированный алгоритм прохода был протестирован на ускорителе NVIDIA GeForce 6800 GT⁴. Интерактивного режима удалось достичь на простейших сценах из нескольких тысяч треугольников в разрешении 256 × 256 пикселей (производительность не превышала 200К лучей в секунду). На более сложных сценах скорость работы значительно уступала стандартному визуализатору Mental Ray для пакета моделирования 3D Studio Max.

5	4	3	2	2	1	1	1	2	2
5	4	3	2	1	1	0	1	1	1
5	4	3	2	1	0	0	0	0	1
5	4	3	2	1	0	0	0	0	1
5	4	3	2	1	0	0	0	1	1
5	4	3	2	1	1	1	1	1	2
5	4	3	2	2	2	2	2	2	2
5	4	3	3	3	3	3	3	3	3
5	4	4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5	5	5

Рис. 1.10. Пример карты близости для регулярной сетки

В работе [85] была предложена реализация стека на графическом процессоре, которая применялась при обходе k -д дерева в алгоритме потоковой трассировки лучей. Для хранения фрагментных стеков применялась текстура, разбитая на k блоков размера $N \times N$, одинаковые элементы которых соответствуют различным уровням одного фрагментного стека (рис. 1.11). Указатели на вершины стеков хранятся в отдельной текстуре размера $N \times N$. Все операции над системой стеков выполняются путем вывода прямоугольника, полностью покрывающего окно размера $N \times N$. В ходе обработки фрагментов каждый шейдер производит определенное действие над стеком соответствующего фрагмента. Для реализации операций добавления и извлечения из фрагментных стеков использовалась многопроходная техника. Данный подход хорошо укладывается в парадигму потокового программирования и сочетается с реализацией потоковой трассировки [79]. Однако многочисленные операции над системой стеков создают дополнительную нагрузку на подсистему памяти.

Level 0			Level 1			Level 2			Level 3		
A0	B0	C0	A1	B1	C1	A2	B2	C2	A3	B3	C3
D0	E0	F0	D1	E1	F1	D2	E2	F2	D3	E3	F3
G0	H0	I0	G1	H1	I1	G2	H2	I2	G3	H3	I3

Рис. 1.11. Пример текстуры для хранения 9 стеков (A...I) по 4 уровня каждый (0..3)

⁴ Имеет 6 вершинных и 16 пиксельных процессоров с частотой 350 МГц. Полоса пропускания памяти – 32 Гб/с.

На графическом ускорителе NVIDIA GeForce FX 5800⁵ производительность визуализации на простой сцене типа Cornell Box оказалась крайне низкой и более чем на порядок уступала оптимизированной системе для центрального процессора Intel Pentium 4.

В работе [86] были предложены модифицированные алгоритмы обхода k -d дерева, для работы которых не требуется стек. Первый алгоритм получил название *kd-restart* и решает задачу следующим образом. Если в процессе обхода луч достиг концевой вершины дерева, в которой не удалось найти точку соударения с треугольником, то обход запускается повторно с корневой вершины, а начальная точка луча смещается в точку выхода из концевого узла. Повторяя данный процесс, процедура обхода обработает все концевые вершины вдоль луча в порядке, соответствующем стандартной процедуре обхода на базе стека (рис 1.12).

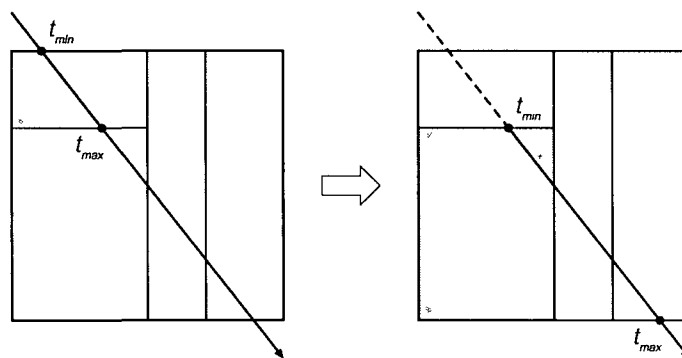


Рис. 1.12. Алгоритм *kd-restart*: переход к корневому узлу

Для сбалансированного по высоте k -d дерева такой алгоритм обхода имеет трудоемкость $O(N \log N)$, где N – число узлов. На практике процедура обхода посещает лишь некоторые концевые вершины, поэтому реальное число операций перехода сохраняет логарифмический характер.

Модификация данного алгоритма, получившая название *kd-backtrack*, снижает оценку трудоемкости до $O(\log N)$ за счет хранения дополнительной информации.

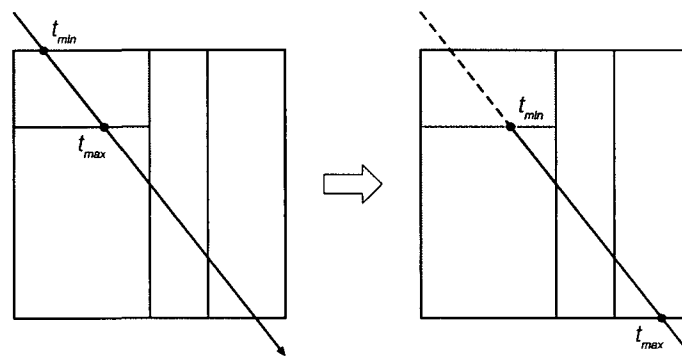


Рис. 1.13. Алгоритм *kd-backtrack*: переход к ближайшему предку узла

⁵ Имеет 3 вершинных и 4 пиксельных процессора с частотой 500 МГц. Полоса пропускания памяти – 16 Гб/с

Как и в предыдущем случае, после обработки концевой узла, в котором не удалось найти точку соударения, алгоритм перемещает начало луча в точку выхода из соответствующего узла. Однако поиск продолжается не с корневой вершины дерева, а с ближайшего предка, имеющего непустое пересечение с оставшимся сегментом луча. Для определения данного узла каждая вершина дерева должна дополнительно хранить указатель на родительский узел и ограничивающий параллелепипед (рис. 1.13). Нетрудно видеть, что такой алгоритм обхода посещает каждую вершину не более трех раз, поэтому для сбалансированного k - d дерева его оценка трудоемкости составит $O(\log N)$. К недостаткам алгоритма относится значительный объем дополнительной памяти, необходимой для хранения дерева. Помимо этого, процедура поиска предка требует дополнительных тестов пересечения луча с параллелепипедом, что вносит лишние вычислительные расходы.

На основе разработанных алгоритмов обхода была построена реализация потоковой трассировки лучей [79]. На графическом ускорителе ATI Radeon X800 XT⁶ при обработке сцен с неоднородной геометрией k - d деревья оказались до 8-ми раз эффективнее регулярных сеток. Однако производительность не превышала ~300К лучей в секунду и значительно уступала оптимизированным реализациям для центральных процессоров. Среди основных лимитирующих факторов отмечалась неэффективная балансировка нагрузки (между ядрами трассировки лучей) и накладные расходы, связанные с рециркуляцией данных в потоковых вычислениях.

Работа [87] направлена на исследование различных структур данных для трассировки на графическом ускорителе, включая регулярные сетки, k - d деревья и иерархии объемов (на основе выровненных по осям параллелепипедов). Реализация сеток и k - d деревьев повторяет работы [79] и [86] соответственно. Для обхода иерархии объемов был разработан элегантный бесстековый алгоритм, который естественно вытекает из особенностей этой структуры (рис. 1.14). Иерархия обходится в прямом порядке: каждая вершина обрабатывается до того, как будут обработаны дочерние узлы. При этом любой порядок обхода потомков обеспечивает корректный результат. Таким образом, на этапе построения дерева все вершины могут быть помечены последовательными номерами в соответствии с порядком их обхода. Для каждой вершины можно заранее вычислить и записать указатель перехода (англ. *escape index*) – номер узла, в который необходимо перейти, если тест пересечения завершился неудачно. Нетрудно видеть, что для листьев относительный указатель перехода всегда равен 1, за счет чего указатели можно хранить только во внутренних вершинах дерева и сократить объем потребляемой памяти.

⁶ Имеет 6 вершинных и 16 пиксельных процессоров с частотой 500 МГц. Полоса пропускания памяти – 32 Гб/с.

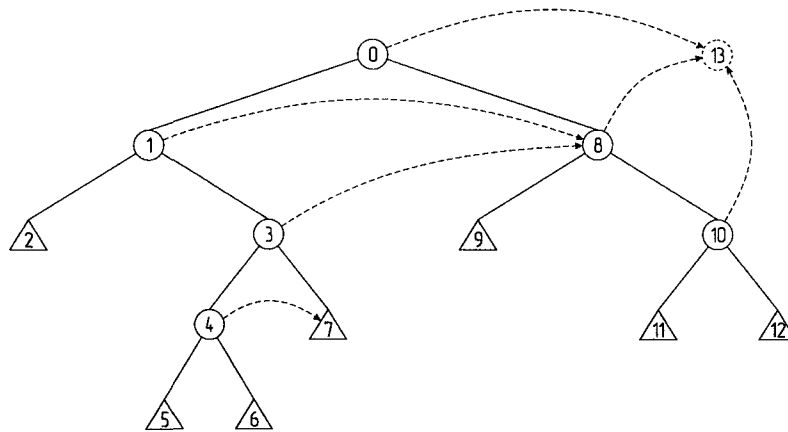


Рис. 1.14. Кодирование вершин дерева для обхода на графическом процессоре

Основной недостаток такого подхода состоит в фиксированном порядке обработки вершин дерева, который не учитывает направление хода луча. В худшем случае процедура обхода вырождается в полный перебор всех треугольников сцены. Исключить такую возможность можно за счет нумерации дочерних узлов в случайном порядке. Результаты экспериментов показали, что на некоторых сценах иерархии объемов обеспечивают девятикратный рост производительности по сравнению с регулярными сетками и k -d деревьями. На ускорителе NVIDIA GeForce 6800 Ultra система обрабатывала 0.5–2.5М лучей в секунду.

Все рассмотренные реализации основаны на оригинальной работе [79] и обеспечивают визуализацию только статических сцен. Даже при использовании адаптивных ускоряющих структур (k -d деревьев и иерархий объемов) исследователям не удалось добиться уровня производительности оптимизированных систем для центрального процессора⁷. Основные лимитирующие факторы были связаны как с ограничениями графических процессоров, так и с неэффективностью применяемой модели потоковых вычислений. Во-первых, разбиение вычислений на элементарные ядра с передачей промежуточных данных через текстуры предъявляет высокие требования к пропускной способности памяти. Данный подход был единственно возможным на первых программируемых графических процессорах, которые имели небольшое число регистров и исполняли короткие шейдерные программы. Однако для более современного оборудования⁸ столь «мелкая» декомпозиция алгоритма уже не являлась оптимальной. Во-вторых, такой подход приводит к большому числу проходов визуализации, большинство из которых являются зависимыми и требуют выполнения запроса на видимость (англ. occlusion query). В силу специфики графического процессора данный запрос обладает латентностью, что также создает дополнительные ограничения.

⁷ Хотя по уровню пиковой производительности графические процессоры на тот момент уже опережали центральные. Например, ускоритель NVIDIA GeForce 6800 Ultra имел пиковую производительность 53 GFLOPS и пропускную способность памяти 35.2 Гб/сек, в то время как процессор Intel Pentium 4 570 (3.8 ГГц) – 15 GFLOPS и 6.4 Гб/сек

⁸ Представленные в 2006 году графические процессоры NVIDIA GeForce 7800 GTX и ATI Radeon X1800 XT получили практически полную поддержку шейдерного языка OpenGL (GLSL), включая сложные инструкции управления потоком

Дальнейшее повышение производительности стало возможно за счет укрупнения ядер обработки и более полного использования возможностей аппаратуры нового поколения. Крайний пример такого подхода дает работа [88], в которой авторы полностью отказались от многопроходной техники и организовали все вычисления в одном фрагментном шейдере. Для поиска пересечений применялась двухуровневая регулярная сетка, которая позволила реализовать поддержку динамических объектов. Первый уровень сетки генерируется для ограничивающих параллелепипедов возможных экземпляров объектов (англ. instances). Каждый экземпляр хранит указатель на геометрический объект и матрицу преобразования, которая помещает данный объект в мировое пространство сцены. В локальном пространстве каждого объекта определяется ограничивающий параллелепипед, внутри которого строится второй уровень регулярной сетки. (Принцип работы аналогичен «размытым» k - d деревьям из публикации [65]). С одной стороны, такая организация ускоряющей структуры обеспечивает адаптацию к распределению геометрии. С другой стороны, в ходе визуализации экземпляры объектов могут подвергаться аффинным преобразованиям. На ускорителе ATI Radeon X1800 XL⁹ производительность трассировки достигала 3.3–6.7М лучей в секунду. Такой уровень быстродействия обеспечил визуализацию простых сцен (менее 35 тысяч треугольников) в реальном времени с учетом теней от точечных источников света. Исследование показало, что графическая аппаратура нового поколения позволяет обрабатывать шейдерные программы с большим числом ветвлений при достаточно высоком уровне производительности.

Оригинальный метод обработки деформируемых полигональных сеток был предложен в работе [89]. Для ускорения трассировки лучей использовалась иерархия объемов, которая формировалась непосредственно на графическом процессоре. На этапе препроцессирования для входной полигональной модели строится равномерная триангуляция и параметризация. Затем посредством стандартного алгоритма растеризации данная модель конвертируется в геометрическое изображение: с помощью вершинного шейдера мировые координаты и цвет вершины заменяются на параметрические и мировые координаты вершины соответственно. Геометрическое изображение определяет аппроксимацию исходной сетки, точность которой зависит от его разрешения. Для геометрического изображения создаются две MIP-пирамиды, в текстели которых заносятся минимальные и максимальные координаты ограничивающих параллелепипедов соответствующих вершин, при этом уровень пирамиды соответствует уровню иерархии объемов. Для обхода ускоряющей структуры применялся бесстековый алгоритм из [87]. На ускорителе NVIDIA GeForce 7800 GTX¹⁰ в зависимости от разрешения геометрического изображения система обрабатывала 1–4М лучей в секунду. Быстродействие

⁹ Имеет 8 вершинных и 16 пиксельных процессоров с частотой 500 МГц. Полоса пропускания памяти – 32 Гб/с.

¹⁰ Имеет 8 вершинных и 24 пиксельных процессора с частотой 430 МГц. Полоса пропускания памяти – 38 Гб/с.

оказалось достаточным для интерактивной визуализации простых деформируемых моделей. Основным недостатком подхода состоит в недостаточной эффективности иерархии объемов, для построения которой не используется эвристика площадей поверхностей (SAH). Кроме того метод поддерживает лишь одиночные полигональные сетки без острых углов, которые зачастую сложно получить из реальных моделей.

Значительным шагом на пути к трассировке лучей реального времени стала работа [90], в которой продолжилось исследование бесстековых алгоритмов обхода k -d деревьев [86]. За основу был взят алгоритм *kd-restart*, который в случае неудачного пересечения с концевой вершиной дерева переходит к корневой вершине, предварительно переместив начало луча в точку выхода из обработанного листового узла. Графическая аппаратура нового поколения позволила реализовать обход в одном фрагментном шейдере, отказавшись от оригинальной многопроходной схемы с балансировкой нагрузки на центральном процессоре. Кроме того, исходный алгоритм был дополнен тремя новыми техниками. Во-первых, была реализована пакетная трассировка лучей, которая к тому времени получила широкое распространение в системах для центрального процессора. Каждый пакет выполняет обработку четырех лучей, что обеспечивает утилизацию векторных инструкций графического процессора (ATI Radeon X1900 XTX¹¹) и снижает накладные расходы динамических ветвлений. Во-вторых, в работе применялась техника, которая позволяет начинать повторный обход с некоторого поддеревья, минуя вершины более высокого уровня (*push-down*). Данная оптимизация применяется лишь в том случае, если луч целиком находится в некотором поддереве исходного k -d дерева. В-третьих, был реализован короткий стек фиксированного размера, который функционирует следующим образом. Добавление нового элемента в полностью заполненный стек приводит к автоматическому удалению нижнего элемента. При извлечении элемента из пустого стека выполняется повторный обход k -d дерева аналогично исходной процедуре *kd-restart*. Таким образом, стек играет роль кэша, который позволяет значительно сократить число переходов к корневой вершине. Перечисленные оптимизации были реализованы на языке потокового программирования Brook GPU (транслируется в стандартные шейдеры) и протестированы на графическом процессоре ATI Radeon X1900 XTX. Переход к однопроходной реализации с инструкциями ветвления и циклами позволил повысить скорость работы до 25-ти раз (хотя производительность графического процессора возросла не более чем в 4 раза). Применение пакетов оказалось целесообразным лишь для первичных и теневых лучей, при этом заметный прирост скорости отмечался только на самых простых сценах. Техника *push-down* позволила

¹¹ Имеет 8 вершинных и 48 пиксельных процессоров с частотой 650 МГц. Полоса пропускания памяти – 49.6 Гб/с

сократить число обрабатываемых узлов на 3–22%, при этом короткий стек сокращал данное число на дополнительные 48–52%. С учетом всех оптимизаций алгоритм обхода k -d дерева на тестовых сценах обрабатывал всего на ~3% больше узлов по сравнению с классическим алгоритмом на базе полноценного стека. Для достаточно сложных сцен (десятки и сотни тысяч треугольников) система обеспечивала производительность на уровне ~15М лучей в секунду, что позволяет говорить о трассировке лучей в реальном времени. Среди основных лимитирующих факторов отмечался широкополосный SIMD (на тестовом процессоре размер «потока» составлял 16 пикселей), что приводило к снижению эффективности динамических ветвлений и ограничивало размер пакетов лучей (в одном фрагментном шейдере).

В работе [91] был описан альтернативный бесстековый алгоритм обхода k -d деревьев, который значительно сокращал число проверяемых узлов и допускал пакетную обработку лучей. Алгоритм основан на k -d деревьях со связками (англ. *gore trees*), которые ранее уже использовались для ускорения трассировки лучей [92]. При таком подходе концевые узлы дерева снабжаются связками (ссылками) для прямого перехода к соседним узлам (рис. 1.15). Для каждой грани концевой вершины связка указывает на смежную концевую вершину или, если таких вершин несколько, на наименьший узел дерева, который включает все смежные концевые вершины.

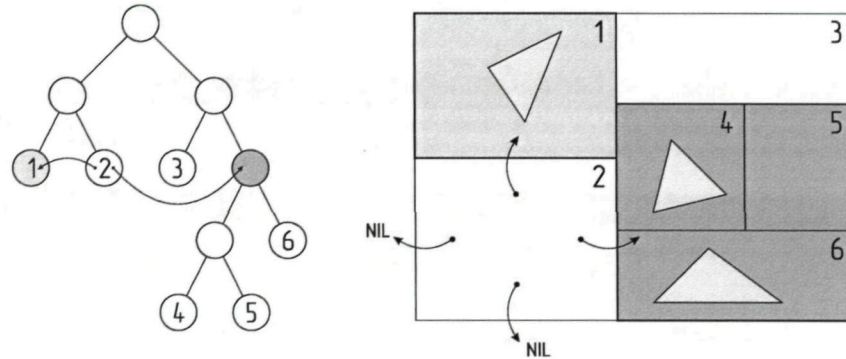


Рис. 1.15. Пример k -d дерева со связками

По аналогии с предыдущими исследованиями, SAH k -d дерево строилось на центральном процессоре, после чего снабжалось связками путем обхода всех вершин за линейное время. На стороне графического процессора для обхода k -d дерева применялась простая бесстековая процедура. Если поиск соударения с треугольниками концевого узла завершился неудачно, алгоритм вычисляет грань и точку выхода из этого листа. Обход дерева продолжается путем перехода к соседнему узлу по соответствующей связке. Если соседняя вершина не является концевой, алгоритм выполняет спуск по дереву до концевого узла, который содержит точку выхода (которая для нового узла будет являться уже точкой входа). Обход завершается при

обнаружении недопустимой связки (*NIL*), указывающей за пределы корневого узла дерева. Очевидно, что данный алгоритм позволяет начинать обход непосредственно с вершины, содержащей начальную точку луча. Эта особенность позволяет значительно оптимизировать обработку вторичных и других лучей с общей начальной точкой (первичных или теневого). В результате число посещаемых вершин сокращается на $\sim 2/3$ по сравнению с алгоритмом на базе стека и на $\sim 5/6$ по сравнению с бесстековым алгоритмом *kd-restart*. С другой стороны, значительным недостатком предложенного подхода является большой объем потребляемой памяти. На тестовых сценах *k-d* дерево со связками требовало до 3-х раз больше памяти по сравнению с обычным *k-d* деревом.

Рассмотренный алгоритм лег в основу классической трассировки лучей, которая была реализована на платформе NVIDIA CUDA [93] (англ. Compute Unified Device Architecture) и протестирована на графическом ускорителе NVIDIA GeForce 8800 GTX¹². Для одиночных первичных лучей производительность достигала 6М лучей в секунду на достаточно сложной сцене (более 300К треугольников). Обработка пакетов лучей (8×4) позволила увеличить быстродействие до 17М лучей в секунду (за счет когерентного доступа к памяти и операций ветвления). Производительность классической трассировки лучей (со вторичными лучами) оказалась примерно вдвое ниже. Несмотря на интерактивную визуализацию относительно сложных сцен, полученные результаты существенно уступают пиковой производительности графического процессора. Среди основных ограничивающих факторов отмечалось большое число регистров в скомпилированном коде, в результате чего занятость (англ. occupancy) графического процессора оказалась на уровне 33%.

В работе [70] для ускорения трассировки лучей применялась иерархия объемов, обход которой выполнялся посредством оптимизированного алгоритма для архитектуры CUDA. Авторы использовали традиционный способ обхода на основе стека, для хранения которого задействовалась разделяемая память (англ. shared memory). Алгоритм выполнял пакетную обработку лучей, используя один стек сразу для всего пакета. При таком подходе каждый луч отображается на поток (англ. thread), а пакет лучей – на свертку потоков¹³ (англ. warp). Потоки в пределах свертки работают синхронно и обрабатывают в каждый момент времени одну вершину дерева. Если текущий узел является концевым, то для всех лучей в пакете вычисляется ближайшее пересечение с его треугольниками. Если узел является внутренним, то для каждого луча в пакете определяется порядок обхода его дочерних узлов. Алгоритм переходит к тому дочернему узлу, который является предпочтительным для большинства

¹² Имеет 128 унифицированных шейдерных процессоров с частотой 1.35 ГГц. Полоса пропускания памяти – 86 Гб/с.

¹³ Свертка (англ. warp) – термин компании NVIDIA, который обозначает группу потоков, которые синхронно исполняются на одном SIMD модуле. В архитектуре NVIDIA G80 размер этой группы составляет 32 потока, на Intel Larrabee 16, AVX 8 и SSE 4.

лучей пакета, при этом адрес другого дочернего узла заносится в стек. Если оба дочерних узла не требуют обхода или был обработан лист, алгоритм извлекает следующий узел из стека. Если стек пуст, процедура обхода завершает свою работу.

На основе данного алгоритма была реализована простая трассировка лучей, все стадии которой выполнялись в одном ядре CUDA. При этом загрузка графического процессора составила 63% для первичных лучей и 38% с учетом теневых лучей и расчета освещения по Фонгу для нескольких источников света. Производительность системы оказалась на уровне предыдущей работы и достигала 17М лучей в секунду. Однако к важным преимуществам иерархии объемов относится низкое потребление памяти. На тестовых сценах для хранения данной структуры потребовалось в 3–4 раза меньше памяти по сравнению с k -d деревом (при использовании связок разница увеличивалась еще в 3 раза). В итоге при объеме графической памяти в 768 Мб система обеспечивала интерактивную обработку больших сцен (порядка 12 миллионов треугольников).

В работе [94] для ускорения трассировки лучей вновь применялось k -d дерево, однако в отличие от предыдущих исследований данная структура генерировалась непосредственно на графическом процессоре. Традиционные алгоритмы построения SAH k -d деревьев на основе «подвижной» плоскости являются вычислительно трудоемкими, однако простые эвристики могут значительно снижать производительность трассировки. В работе был найден удачный компромисс, который основан на разделении узлов на большие и малые в зависимости от числа примитивов (использовался порог $T = 64$). В отличие от параллельных реализаций для центрального процессора, алгоритм выполняет построение k -d дерева в ширину. На стадии обработки больших вершин вычисления распараллеливаются по треугольникам сцены. При этом для выбора секущей плоскости применяются две простые эвристики: отсечение пустого пространства (если оно превышает 25% от объема вокселя) и разбиение по пространственной медиане [95]. На стадии обработки малых вершин вычисления распараллеливаются по узлам, для разбиения которых используется алгоритм подвижной плоскости на базе функции SAH-стоимости. На тестовых сценах эффективность ускоряющей структуры оказалась не ниже стандартного SAH k -d дерева [64], при этом скорость построения на ускорителе NVIDIA GeForce 8800 Ultra¹⁴ достигала 2.7М треугольников в секунду. Для обхода k -d дерева на графическом процессоре использовался «полный» стек, который размещался в локальной памяти каждого потока (отображается на внешнюю память). Реализация трассировки лучей Уиттеда, которая поддерживала тени, отражения и преломления, позволяла обрабатывать до 20–40М «смешанных» лучей в секунду. За счет высокопроизводительного генератора k -d

¹⁴ Имеет 128 унифицированных шейдерных процессоров с частотой 1.5 ГГц. Полоса пропускания памяти – 103 Гб/с.

дерева система позволяла визуализировать простые (30–250К треугольников) динамические сцены с частотой 4–8 кадров в секунду в разрешении 1024 × 1024 пикселей. Кроме того, на базе реализации трассировки лучей и генератора *k-d* дерева был построен упрощенный метод фотонных карт, который обеспечивал визуализацию каустик в реальном масштабе времени для низкополигональных сцен (3–19К треугольников).

Таким образом, были предложены различные высокопроизводительные реализации трассировки лучей, которые базируются на разных ускоряющих структурах и алгоритмах обхода. Однако для достижения *максимальной* производительности необходимо тщательно проанализировать и определить основные ограничивающие факторы, которые могут быть связаны с производительностью вычислительных блоков, пропускной способностью памяти и другими особенностями аппаратуры. Поскольку трассировка лучей допускает множество реализаций, результат такого анализа может существенно зависеть от компьютерной сцены, ускоряющей структуры, точки обзора и типа обрабатываемых лучей. Впервые исследование различных высокопроизводительных решений было выполнено в работе [96], направленной на поиск наиболее эффективных реализаций базовых операций трассировки лучей – обход ускоряющей структуры и пересечение с треугольником (в совокупности образуют функцию *trace*). В рамках данной работы оптимизированные CUDA-версии различных вариантов *trace* исследовались на реальном оборудовании и симуляторе, позволяющем определить *верхнюю* границу быстродействия на графическом процессоре NVIDIA (и других широкополосных SIMD/SIMT архитектурах). Данный симулятор поддерживает различные алгоритмы работы диспетчера потоков и ширину SIMD, однако игнорирует некоторые особенности реальной аппаратуры. Таким образом, симулятор не позволяет сделать точный прогноз, но способен выявить сильные провалы в производительности и узкие места. В качестве ускоряющей структуры использовалась оптимизированная иерархия объемов, которая генерировалась на центральном процессоре. Обход данной структуры выполнялся следующими методами:

- *Пакетная обработка.* Группа лучей (свертка) обходит дерево в одинаковом порядке за счет использования общего стека [70]. Хотя лучи могут обрабатывать лишние узлы дерева, такой подход обеспечивает предельно когерентный доступ к памяти.
- *Одиночные лучи.* Для каждого луча используется отдельный стек, который хранится в локальной памяти потока (внешняя память) [94]. Такой подход исключает лишние операции обхода, однако снижает когерентность доступа к памяти.

Результаты исследования на симуляторе и графическом ускорителе NVIDIA GeForce GTX 285 показали, что трассировка одиночных лучей обеспечивает наилучшее быстродействие: пропускная способность памяти не является главным лимитирующим фактором. Авторы

выяснили, что на графических процессорах с микроархитектурой Tesla (серия 2xxx) скорость работы ограничивается диспетчером потоков, который оптимизирован для «однородных» заданий. Данное ограничение удалось обойти за счет использования постоянных потоков (англ. persistent threads), которые позволили достичь ~80% от теоретического максимума. На ускорителе GeForce GTX 285¹⁵ система обрабатывала 40–60М диффузных вторичных лучей в секунду и 75–140М первичных лучей в секунду.

В публикации [97] был предложен альтернативный конвейер высокопроизводительной трассировки лучей, который основан на обработке больших групп лучей и использовании ограничивающих пирамид. В рамках данного подхода функция *trace* выполняется в четыре этапа. На *первом* этапе лучи сортируются в когерентные пакеты (размера $T \leq 256$) согласно следующей схеме. Для каждого луча вычисляется 32-битный хеш-код, который получается в результате проецирования начальной точки и направления луча на ячейки регулярных сеток. (Лучи, получившие одинаковые хеш-коды, считаются когерентными в 3D пространстве.) Далее к списку лучей применяется метод группового кодирования (англ. run-length encoding), который каждую цепочку одинаковых хеш-кодов заменяет парой из индекса начального луча и числа идентичных хеш-кодов. Сжатые данные упорядочиваются с помощью параллельного алгоритма поразрядной сортировки и распаковываются в последовательность когерентных групп лучей, из которой формируются пакеты требуемого размера (рис. 1.16). Данная схема получила название CSD (англ. Compression-Sorting-Decompression).

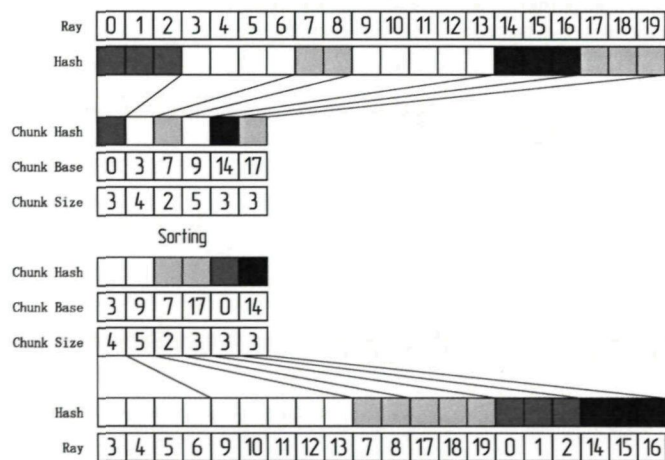


Рис. 1.16. Упорядочивание когерентных лучей по CSD-схеме

На *втором* этапе выполняется построение ограничивающих пирамид групп лучей. Каждая пирамида определяется доминирующей осью и двумя выровненными прямоугольниками, которые ортогональны данной оси и ограничивают все лучи внутри пакета. На *третьем*

¹⁵ Имеет 240 унифицированных шейдерных процессоров с частотой 1.5 ГГц. Полоса пропускания памяти – 159 Гб/с.

этапе пирамиды пересекаются с ускоряющей структурой, в качестве которой используется иерархия ограничивающих объемов в 8-арном представлении (строится на центральном процессоре на основе бинарного дерева). Обход иерархии выполняется в ширину, при этом на каждом уровне дерева узлы предварительно сортируются в соответствии с усредненным направлением лучей пирамиды. В результате для каждой пирамиды формируется список перекрываемых листовых узлов, все примитивы которых тестируются на пересечение с лучами пирамиды на *четвертом* этапе.

Разработанные алгоритмы были реализованы на платформе CUDA и протестированы на графическом ускорителе NVIDIA GeForce GTX 285. Производительность функции *trace* сравнивалась с собственной реализацией данной функции из работы [96]. Авторы сообщили результаты экспериментов только для первичных и теневых лучей от площадных источников света (использовалось 16 семплов на источник). Если на первичных лучах новая реализация оказалась на 12–100% быстрее, то на теневых лучах производительность возросла 3.2–4.6 раз. В абсолютных величинах быстродействие системы достигало 110–150М теневых лучей в секунду. Предложенные алгоритмы обеспечивают когерентный доступ к памяти и высокую эффективность использования SIMD. Однако для значительно менее когерентных вторичных лучей их эффективность может оказаться существенно ниже. Различные способы сортировки лучей исследовались в работе [74], однако не позволили извлечь «скрытую» когерентность и повысить производительность пакетной трассировки.

Исследование высокопроизводительных реализаций функции *trace* было продолжено в недавней работе [98], которая дополнила исходную публикацию [96] экспериментальными данными на микроархитектурах Fermi (GeForce GTX 480¹⁶) и Kepler (GeForce GTX 680¹⁷).

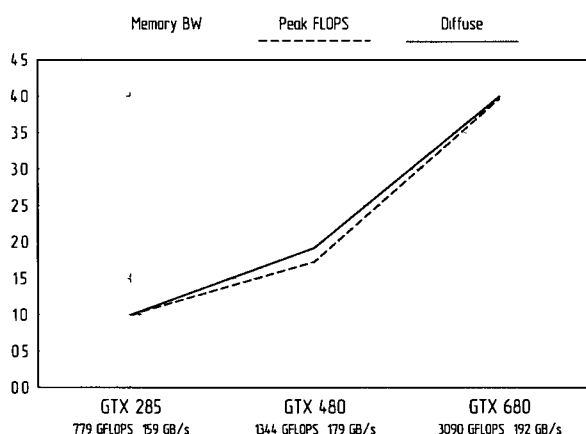


Рис. 1.17. Сравнение относительной скорости обработки диффузных лучей с полосой пропускания памяти и пиковой производительностью

¹⁶ Имеет 480 шейдерных процессоров с частотой 1.4 ГГц. Пиковая полоса пропускания памяти – 177 Гб/с

¹⁷ Имеет 1536 шейдерных процессоров с частотой 1.0 ГГц. Пиковая полоса пропускания памяти – 192 Гб/с

Для обхода оптимизированной иерархии объемов (с учетом «пространственных разбиений» из работы [106]) использовался стек, который размещался в локальной памяти потока. На графическом ускорителе NVIDIA GeForce GTX 680 производительность обработки наименее когерентных диффузных отскоков достигала 156–245М лучей в секунду на простых сценах (менее 300К треугольников) и 60М лучей в секунду на сложной сцене из 11М треугольников. Результаты экспериментов показывают, что быстродействие трассировки лучей (даже для некогерентных вторичных направлений и больших сцен) ограничивается, в первую очередь, производительностью вычислительных блоков, а не пропускной способностью памяти.

Новые алгоритмические решения и рост производительности графической аппаратуры превратили трассировку лучей в универсальный метод для визуализации сложных 3D сцен в реальном времени. Однако высокая трудоемкость построения ускоряющих структур часто ограничивала область применения метода только статическими сценами. Решение данной проблемы потребовало глубокой переработки структур данных, которые применялись для трассировки лучей более 20 лет. Исследователи предложили различные техники, которые были ориентированы на центральный процессор и обеспечивали интерактивную генерацию ускоряющих структур для простых сцен. С появлением гибких инструментов параллельного программирования (CUDA и OpenCL) данные техники были адаптированы для графической аппаратуры, что позволило повысить сложность сцен на порядок. Однако непосредственное портирование алгоритмов на массивно-параллельные графические архитектуры зачастую невозможно, поскольку эффективная реализация должна обладать достаточным объемом «мелкозернистого» параллелизма.

Вслед за работой [94] параллельный алгоритм построения k -d дерева был предложен в публикации [99]. Для генерации структуры применялась ячеечная («корзинная») техника, которая разбивает все узлы дерева на основе функции SAH-стоимости. Построение дерева выполняется сверху вниз, при этом узлы на каждом уровне обрабатываются множеством параллельных потоков. Процесс генерации разбит на пять стадий, которые обрабатывают сверхбольшие ($T > 512$), большие ($T > 512$), средние ($33 \leq T \leq 512$), малые ($33 \leq T \leq 512$) и сверхмалые ($T \leq 32$) узлы. Такая декомпозиция позволяет использовать индивидуальные оптимизации для обработки различных вершин дерева. На каждой стадии разбиение узлов выполняется в три этапа. На *первом* этапе с помощью ячеечной техники (использовались 32 корзины) определяется секущая плоскость с наименьшей SAH-стоимостью. (При обработке сверхмалых вершин применялся стандартный алгоритм подвижной плоскости). На *втором* этапе выполняется разбиение узла и распределение примитивов по его потомкам. Данный этап требует выделения дополнительной памяти для хранения примитивов дочерних узлов.

На *третьем* этапе треугольники, пересекаемые секущей плоскостью, отсекаются по границам соответствующих дочерних узлов. Данный алгоритм был протестирован на графическом процессоре NVIDIA GeForce GTX 285 и обеспечивал построение качественных SAH k -d деревьев со скоростью ~ 2.5 М треугольников в секунду. За счет модификации функции SAH-стоимости (которая привела к укрупнению листовых узлов) скорость построения возросла до ~ 4.1 М треугольников в секунду, при этом производительность трассировки упала на 15%.

Построению «точных» k -d деревьев посвящена работа [100], в которой при разбиении узла рассматривались все $6N$ потенциальных плоскостей (N – число треугольников узла). По аналогии с предыдущими исследованиями дерево строилось сверху вниз, при этом обработка каждого уровня распараллеливается по треугольникам сцены. Алгоритм работает в четыре этапа. На *первом* этапе выполняется инициализация, в рамках которой создается корневой узел дерева и формируются три списка потенциальных секущих плоскостей (по одному для каждой оси), которые упорядочиваются с помощью параллельной поразрядной сортировки. На *втором* этапе определяется плоскость с наименьшей SAH-стоимостью разбиения. Для подсчета числа треугольников по обе стороны от каждой плоскости применяется операция префиксной суммы, а для выбора плоскости с минимальной стоимостью – параллельная редукция. На *третьем* этапе треугольники сцены распределяются по двум дочерним узлам. Для повышения качества k -d дерева выполняется отсечение треугольников, перекрываемых секущей плоскостью, по соответствующим дочерним узлам (англ. split clipping). Данная операция может нарушать порядок в отсортированном списке плоскостей. Поскольку число отсекаемых треугольников ограничено величиной $O(\sqrt{N})$, порядок может быть эффективно восстановлен за счет вставки образованных секущих плоскостей в существующий список. Авторы реализовали данную процедуру на основе корзинной сортировки. На графическом процессоре NVIDIA GeForce GTX 280 построение SAH k -d дерева выполнялось со скоростью 1.5–1.7М треугольников в секунду, что оказалось в среднем на 8–30% быстрее генератора для 32-х ядерной системы из работы [[101].

В работе [102] были предложены два алгоритма для построения иерархии объемов на графическом процессоре. Первый алгоритм получил название LBVH (англ. Linear BVH) и сводит построение иерархии объемов к сортировке примитивов («линеаризации») вдоль пространственной кривой Мортонa (Z -кривой). В рамках данного подхода параллелепипед сцены накрывается «виртуальной» сеткой размера $2^k \times 2^k \times 2^k$ (число k определяет глубину дерева, в работе $k = 10$). Каждому примитиву по центру ограничивающего параллелепипеда присваивается его $3k$ -битный код Мортонa (рис. 1.18). После чего производится сортировка списка треугольников по данным кодам с помощью эффективной параллельной реализации

поразрядной сортировки (англ. radix sort) [103]. В итоге центры примитивов сцены будут упорядочены вдоль кривой Мортона. Отсортированный список треугольников позволяет быстро сформировать иерархию объемов. Для построения первого уровня дерева список примитивов делится на два непрерывных интервала на основе значения (0 или 1) самого старшего разряда в коде Мортона. Аналогичным образом строятся другие уровни дерева, при этом каждое новое разбиение выполняется на основе следующего разряда в коде Мортона. Таким образом, алгоритм генерирует иерархию объемов на основе медианного разбиения пространства с трудоемкостью $O(N)$. Основным недостатком «линейной» иерархии объемов – снижение производительности трассировки лучей по сравнению с SAH-деревом, которое в некоторых случаях может достигать 85%.

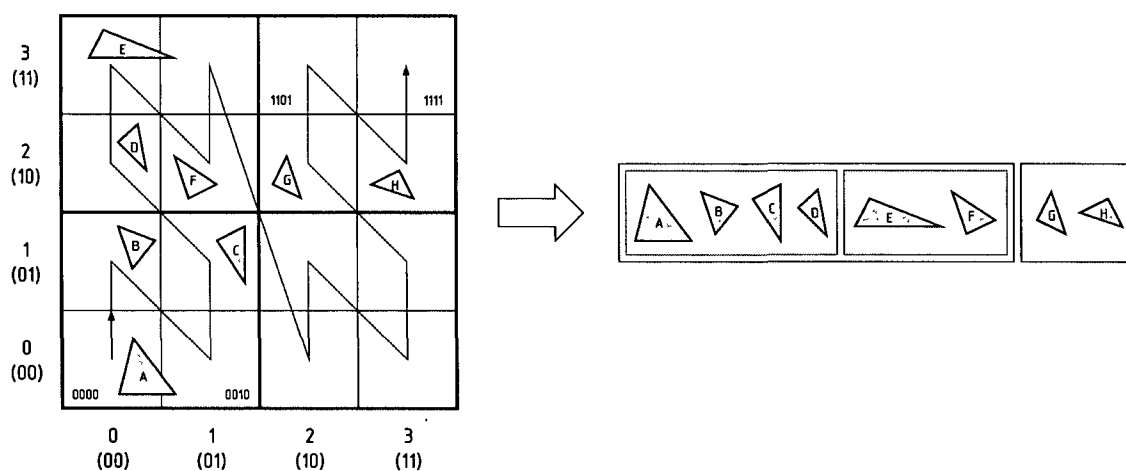


Рис. 1.18. Пример построения первых двух уровней иерархии по кривой Мортона

Второй алгоритм строит SAH-иерархию объемов с помощью ячеечной техники, которая была адаптирована для массивно-параллельных графических архитектур. Процесс генерации дерева представлен в виде набора заданий, параллельное исполнение которых реализуется с помощью концепции очереди. На каждом шаге работы алгоритма блоки потоков загружают из очереди доступные задания (по разбиению узлов), выполняют их обработку и записывают в очередь новые задания (если требуется разбить один или оба дочерних узла). Предлагаемая реализация делит каждый воксель на 32 ячейки, за счет чего обеспечивается эффективная утилизация 32-полосного SIMD («свертка» потоков) на CUDA-совместимых графических процессорах. Потоки в «свертке» последовательно загружают примитивы узла, определяют их положение относительно соответствующих границ ячеек и вычисляют SAH-стоимость разбиений. Затем с помощью алгоритма параллельной редукции определяется плоскость с наименьшей стоимостью, и исходный список треугольников разбивается на две части для формирования дочерних узлов (в отличие от $k-d$ дерева – без выделения дополнительной

памяти). Для лучшей утилизации графического процессора задания (по разбиению узлов) делятся на большие и малые в зависимости от числа примитивов (применялся порог $T = 32$). При обработке малых разбиений все данные размещаются в разделяемой памяти блока, за счет чего небольшие поддеревья строятся без обращений к глобальной памяти устройства. Основная проблема алгоритма проявляется на верхних уровнях дерева, в процессе обработки которых графический процессор остается незагруженным (из-за малого числа заданий).

Оба предложенных алгоритма («линейный» и «ячеечный») были протестированы на графическом процессоре NVIDIA GeForce 280 GTX. «Линейный» алгоритм генерировал иерархию объемов со скоростью 15–22М треугольников секунду, в то время как «ячеечный» – со скоростью ~0.6М треугольников в секунду. Наряду с этим в работе был протестирован гибридный подход, при котором узлы на верхних уровнях дерева разбиваются «линейным» алгоритмом, а для глубоких узлов применяется более точный «ячеечный» алгоритм. Данный подход позволил повысить скорость генерации дерева до 2.7–3.4М треугольников в секунду, при этом снижение производительности трассировки не превысило 8%.

«Линейный» алгоритм построения иерархии объемов (LBVH) был усовершенствован в работе [104]. Авторы предложили оптимизированную схему сортировки примитивов вдоль кривой Мортонa, которая активно использует пространственную когерентность в исходной полигональной сетке. В исходной работе $3k$ -битный код Мортонa задает регулярную сетку разрешения $2^k \times 2^k \times 2^k$ ($k = 10$). Несколько старших разрядов определяют грубую сетку верхнего уровня, которая разделяется на более мелкие ячейки младшими разрядами кода. Этот принцип лежит в основе нового алгоритма, который разделяет процедуру построения на «грубую» и «мелкую» стадию. На первой стадии треугольники сцены распределяются по вокселям грубой $3m$ -битной сетки ($m < k$) с помощью сортировки вдоль m -битной кривой Мортонa. На второй стадии примитивы упорядочиваются внутри вокселей грубой сетки по оставшимся $3(k - m)$ разрядам. Для достаточно малых m (5 или 6) соседние примитивы полигональной модели будут часто оказываться в одной ячейке грубой сетки (в этом состоит пространственная когерентность геометрических данных). Чтобы извлечь пользу из данного наблюдения, в работе применяется CSD-схема (англ. compress-sort-decompress) [97], которая использует групповое кодирование¹⁸ треугольников по $3m$ старшим битам для эффективной поразрядной сортировки. В каждой ячейке грубой сетки примитивы сцены сортируются в разделяемой памяти блока с помощью алгоритма четно-нечетных перестановок (англ. odd-even sort). По сравнению с оригинальным алгоритмом глобальной поразрядной сортировки такая схема позволила снизить нагрузку на память и уменьшить число глобальных точек

¹⁸ Метод сжатия данных, при котором последовательная серия одинаковых элементов заменяется на два символа – элемент и число его повторений.

синхронизации. Для преобразования отсортированного списка треугольников в иерархию объемов была предложена оптимизированная процедура, которая последовательно заменяет узлы иерархии небольшими поддеревьями глубины p (использовалось $p = 3$). Улучшенный «линейный» алгоритм построения структуры получил название HLBVH (англ. Hierarchical Linear Bounding Volume Hierarchy).

На основе алгоритма HLBVH был предложен метод формирования иерархий объемов, оптимизированных функцией SAH-стоимости. В рамках данного подхода алгоритм HLBVH используется для построения деревьев внутри ячеек грубой сетки, полученной в результате сортировки вдоль n -битной кривой Мортонa. Для построения верхнего уровня иерархии полученные деревья объединяются с помощью обычного алгоритма подвижной плоскости (англ. sweep plane), при этом используется *точная* SAH-стоимость отдельных поддеревьев. Такой подход является «обратным» к предложенным ранее гибридным алгоритмам, которые на верхних уровнях используют простые эвристики, а на нижних переходят к разбиениям на основе функции SAH-стоимости. Авторы утверждают, что такой подход позволяет строить более эффективные деревья, однако эксперименты говорят о схожих результатах.

Для тестирования алгоритмов построения иерархии объемов (HLBVH и SAH HLBVH) использовался графический ускоритель NVIDIA GeForce GTX 280. «Линейный» алгоритм HLBVH выполнял генерацию иерархии со скоростью 25–33М треугольников в секунду, при этом с учетом SAH-стоимости на верхних уровнях производительность понизилась до 6–7М треугольников в секунду. В работе не сообщается о производительности трассировки лучей, однако приводится SAH-стоимость сгенерированных деревьев. По сравнению с алгоритмом подвижной плоскости стоимость HLBVH-деревьев возросла на 4–33%, а стоимость SAH HLBVH-деревьев – на 1–14%. Данные показатели были получены на небольшом наборе тестовых сцен и могут оказаться заниженными (в работе [102] для «линейных» деревьев падение производительности достигало 85%).

Поиск эффективных параллельных алгоритмов для построения иерархии объемов был продолжен в работе [105]. В рамках данного исследования была предложена новая ячеечная техника, которая за счет системы вспомогательных регулярных сеток позволяет построить SAH-иерархию объемов за линейное время. Генерация ускоряющей структуры выполняется в несколько этапов. На *первом* этапе формируется детализированная сетка с разрешением $1024 \times 1024 \times 1024$, каждой ячейке которой присваивается 30-битный ключ. Компактное хранение в памяти достигается за счет двухуровневого представления: на верхнем уровне используется сетка $128 \times 128 \times 128$, непустые ячейки которой разбиваются сеткой $8 \times 8 \times 8$ на нижнем уровне. На *втором* этапе происходит распределение примитивов по вокселям

детализированной сетки. Центроидам треугольников сопоставляются 30-битные коды ячеек, которые используются в качестве ключей для поразрядной сортировки списка примитивов. На данном этапе вычисления ускоряются с помощью CSD-схемы, которая позволяет извлечь выгоду из пространственной когерентности примитивов. На *третьем* этапе строится MIP-пирамида регулярных сеток, каждая следующая сетка которой вдвое крупнее предыдущей (самая грубая сетка имеет разрешение $8 \times 8 \times 8$). На *четвертом* этапе выполняется генерация иерархии объемов. Данный этап аналогичен работе [102] и основан на очереди заданий, в которую помещаются разбиваемые вершины. Для выбора секущей плоскости используется самая крупная из сеток MIP-пирамиды, которая содержит не менее 5 вокселей вдоль каждой стороны разбиваемого узла. Вдоль каждой оси вычисляются SAH-стоимости плоскостей выбранной сетки (не более 7 для каждого измерения), и выбирается плоскость с наименьшей стоимостью. В отличие от большинства аналогичных работ, алгоритм разбиения игнорирует реальные ограничивающие параллелепипеды треугольников. Такой подход может приводить к довольно грубой аппроксимации SAH-стоимости [71] и в некотором смысле аналогичен работе [54]. Разбиение продолжается до тех пор, пока текущий узел покрывает более одного вокселя детализированной сетки. На *пятом* этапе вычисляются реальные ограничивающие объемы узлов путем обхода дерева в обратном порядке. В работе также уделено внимание проблеме длинных и широких треугольников, которые часто приводят к неэффективной иерархии с сильно перекрывающимися объемами. Возможное решение проблемы состоит в аппроксимации таких треугольников «плотным» набором параллелепипедов. Данный подход был реализован с помощью прохода по списку треугольников перед построением иерархии объемов. На графическом процессоре NVIDIA GeForce GTX 480 скорость генерации дерева составила 30–110М треугольников в секунду. Несмотря на ряд упрощений ячеечной техники, авторы сообщают о высоком качестве структуры: на тестовых сценах производительность трассировки оказалась всего на 4–10% ниже по сравнению с оптимизированной иерархией объемов из работы [106].

В работах [107] и [108] были предложены схожие параллельные алгоритмы построения регулярных сеток. Программа на центральном процессоре устанавливает разрешение сетки, которое выбирается адаптивно в зависимости от размера сцены и количества треугольников (по аналогии с публикацией [51]). На *первом* этапе запускается ядро, которое для каждого треугольника определяет число перекрываемых вокселей, и вычисляется суммарный объем памяти, необходимый для хранения регулярной сетки. На *втором* этапе выполняется ядро, которое загружает примитивы сцены и для каждой перекрываемой ячейки записывает пару, состоящую из номера ячейки сетки и индекса треугольника. На *третьем* этапе выполняется

сортировка списка пар по номеру ячейки с помощью параллельной реализации поразрядной сортировки (англ. radix sort), которая доступна в инструментарии разработчика CUDA SDK. На *четвертом* этапе на основе отсортированного списка пар («ячейка-примитив») строится регулярная сетка вокселей, которая отображается на трехмерную текстуру и используется для трассировки лучей на графическом процессоре. Основным недостатком алгоритма состоит в значительном объеме потребляемой памяти, которая необходима для хранения ячеек сетки и временного списка пар «ячейка-примитив». На графическом ускорителе NVIDIA GeForce GTX 280 1 Гб размер обрабатываемых сцен был ограничен ~10М треугольниками, при этом скорость построения сетки достигала 17–24М треугольников. Однако производительность трассировки лучей заметно уступала адаптивным ускоряющим структурам (*k-d* деревьям и иерархиям ограничивающих объемов) и находилась в диапазоне 3.5–8.1М лучей в секунду даже для простых сцен (менее 300К треугольников).

Алгоритм генерации регулярных сеток получил дальнейшее развитие в работе [109], где была предложена концепция двухуровневых вложенных сеток для трассировки лучей. Двухуровневая сетка определяется как иерархия фиксированной глубины, на верхнем уровне которой строится относительно грубая регулярная сетка. Каждая ячейка данной сетки, в свою очередь, разбивается регулярной сеткой некоторого разрешения. Данная ускоряющая структура принадлежит к классу иерархических сеток, введенному в работе [110], однако в таком варианте ранее не исследовалась. Генерация двухуровневой сетки выполняется сверху вниз. Сначала строится регулярная сетка верхнего уровня. Затем выполняется построение всех сеток нижнего уровня. По аналогии с предыдущей работой данная задача была сведена к сортировке. На графическом процессоре NVIDIA GeForce GTX 285 скорость построения такой структуры достигала 33–40М треугольников в секунду. При этом производительность трассировки по сравнению с обычной регулярной сеткой возросла на 34–100%.

1.3. Интерактивное глобальное освещение

Корректный расчет глобального освещения характеризуется высокой вычислительной трудоемкостью, поэтому большинство интерактивных систем базируется на упрощенных моделях, которые сочетают приемлемое быстродействие и высокое качество визуализации. Для интерактивного расчета глобального освещения применяются различные подходы [111], такие как метод излучательности (англ. radiosity), стохастическая трассировка путей (англ. stochastic path tracing), метод фотонных карт (англ. photon mapping), метод «мгновенной» излучательности (англ. instant radiosity), «многосветовые» (англ. many-light-based) подходы, точечные подходы (англ. point-based) или «микрорендеринг» (англ. micro-rendering) и метод

дискретных координат (англ. discrete ordinate). Кроме того, для повышения быстродействия практически всех перечисленных алгоритмов могут применяться схемы кэширования.

В настоящей работе для расчета глобального освещения используется (стохастическая) трассировка путей, которая решает уравнение визуализации методом Монте-Карло. Хотя современные графические ускорители обрабатывают свыше 100 миллионов случайных лучей в секунду, некогерентность вторичных отскоков является основной проблемой эффективных реализаций [112]. Быстрое разветвление путей ведет к плохой утилизации вычислительных ресурсов и снижает эффективность доступа к памяти.

В типичной реализации трассировки путей на графическом процессоре каждый поток обрабатывает один путь, который останавливается в произвольной точке методом «русской рулетки». Такой подход неизбежно ведет к ситуации, когда некоторые потоки продолжают трассировать пути, в то время как другие уже завершили работу и «вхолостую» занимают вычислительные ресурсы. В большинстве реализаций данная проблема устраняется за счет установки фиксированной глубины трассировки. Данное решение неэффективно, поскольку приводит к игнорированию важных путей переноса света или, наоборот, к обработке путей с ничтожным вкладом. В работе [113] было предложено альтернативное решение, в рамках которого выполняется перезапуск («регенерация») путей, которые были завершены или не имеют пересечений с геометрией сцены. Регенерация путей выполняется в специальной процедуре, которая запускается сразу же после применения «русской рулетки». Поскольку трассировать большое число первичных лучей на каждый пиксель нецелесообразно, новые пути испускаются из точек соударения предыдущих первичных лучей. Для тестирования алгоритма применялся графический процессор NVIDIA GeForce GTX 285. Регенерация путей привела к повышению производительности в 1.2–2.2 раза. В абсолютных величинах скорость трассировки достигала 10–17М *некогерентных* лучей в секунду для простых сцен (менее 300 тысяч треугольников). Несмотря на очевидные преимущества, предложенный подход имеет два недостатка. Во-первых, в каждой свертке потоков завершаются только некоторые пути, поэтому загрузка SIMD-модулей на этапе самой регенерации оказывается довольно низкой (встречаются оценки на уровне 30–35%). Во-вторых, новые лучи «разбрасываются» по всему потоку обрабатываемых лучей, что нарушает (потенциальную) когерентность в свертках (и приводит к снижению эффективности SIMD-модулей).

В работе [114] предложен другой подход к решению проблемы, который основан на уплотнении активных (незавершенных) потоков (англ. active thread compaction). В процессе уплотнения исходная совокупность частично занятых свертков сжимается путем исключения завершенных потоков, что приводит к меньшему числу полностью утилизируемых свертков.

В работе реализовано «полноэкранное» уплотнение (англ. whole-frame compaction), которое выполняется после продолжения путей на одну вершину и применения «русской рулетки». Процедура уплотнения использует стандартный примитив из библиотеки CUDA Performance Primitives (CUPP). К недостаткам данного подхода относится вынужденная декомпозиция трассировки на несколько ядер, управляющая логика которых переносится на центральный процессор. Уплотнение невозможно начать до тех пор, пока последний поток не завершит обработку отскока луча. Кроме того, уплотнение приводит к наполнению сверток полностью некогерентными лучами, что снижает эффективность доступа к памяти и приводит к плохой утилизации SIMD-модулей. В результате на графическом процессоре NVIDIA GeForce GTX 480 производительность увеличилась всего на 12–16% по сравнению с обычной реализацией.

Для эффективной утилизации SIMD-модулей графического процессора в работе [115] предложено совместить обе техники – уплотнение потоков и регенерацию путей. На вход каждой итерации алгоритма поступает большой поток *семплов*, которые хранят последние вершины соответствующих путей. Для каждого семпла строится не более одного теневого и вторичного луча, которые записываются в поток *соединений* (англ. connection stream) и поток *продолжений* (англ. extension stream) соответственно (рис. 1.19).

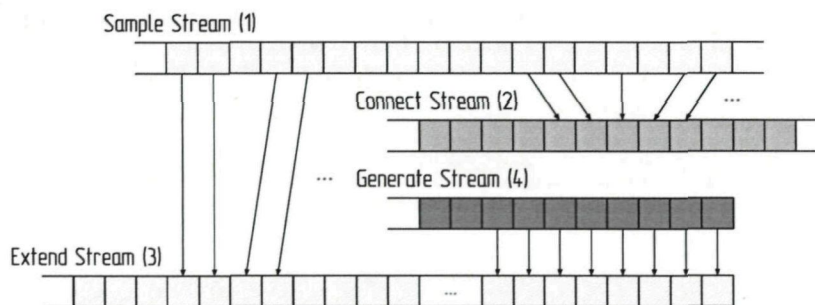


Рис. 1.19. Обработка потоков на одной итерации трассировки путей

Поскольку некоторые семплы не соединяются с источником света или завершаются русской рулеткой, потоки соединений и продолжений могут содержать пустые элементы, которые удаляются с помощью операции уплотнения. В конец уплотненного списка расширений записываются вновь сгенерированные семплы. Далее выполняется обработка всех элементов в списке соединений и расширений. Для каждого соединения (теневого луча) вычисляется функция видимости и вклад в яркость соответствующего пикселя. Для каждого расширения (вторичного луча) определяется следующая вершина пути. Список новых вершин формирует входной поток семплов для следующей итерации трассировки путей. Такой подход позволил повысить производительность на 16–48% по сравнению с техникой регенерации путей [113].

Аналогичным образом была реализована двунаправленная трассировка путей. В начале алгоритма создаются потоки световых и видовых путей, каждая пара которых образует один

семпл. В отличие от обычной трассировки путей каждый семпл должен хранить *все* вершины обоих путей, что приводит к резкому росту объема потребляемой памяти (более 20-ти раз). На практике потоки световых и видовых путей содержат информацию только о последних точках соударения, которая необходима для вычисления следующих вершин путей. Список сгенерированных вершин хранится в отдельной области памяти, которая выделяется заранее с учетом предельной глубины трассировки. Построение путей выполняется в соответствии с рассмотренной ранее схемой (рис. 1.20). После завершения большинства двунаправленных путей методом «русской рулетки» (использовался порог в 60%) запускается фаза обработки соединений. Поскольку число вершин в семплах может существенно различаться, загрузка графического процессора оказывается крайне неоднородной.

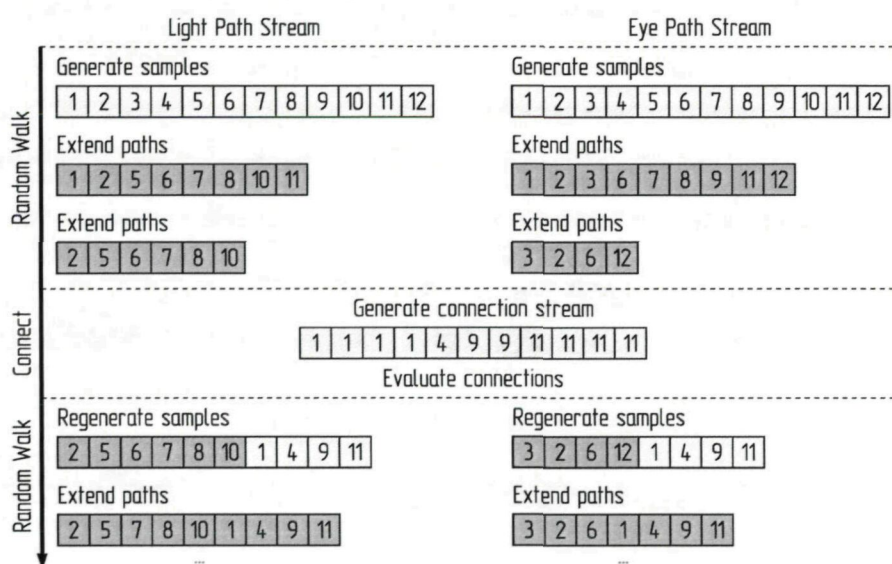


Рис. 1.20. Обработка потоков в двунаправленной трассировке путей

Данную проблему удалось решить за счет распараллеливания вычислений по отдельным соединениям – теневым лучам. Для этого формируется новый поток, в который каждый *завершенный* семпл записывает определенное число соединений. В процессе обработки каждого соединения его вес определяется с помощью многократной выборки по значимости (англ. multiple importance sampling). Вместо традиционной процедуры вычисления MIS-весов используется рекурсивная схема из работы [116]. При построении путей в каждой вершине вычисляется дополнительная величина, которая позволяет определить MIS-вес, используя данные только соединяемых вершин (при этом требуется фиксированное число операций независимо от длины путей). Предложенная реализация двунаправленной трассировки путей была протестирована на графическом ускорителе NVIDIA GeForce GTX 480 и обеспечила более чем 10-кратный рост производительности по сравнению с оптимизированной версией для центрального процессора.

В работе [117] предложена схема управления памятью, которая обеспечивает подкачку данных из системной памяти в собственную память графического процессора, за счет чего максимальный объем обрабатываемых данных возрастает более чем на порядок. В рамках данного подхода исходный набор данных делится на страницы – минимальные единицы запрашиваемой памяти. Программа на графическом процессоре выполняет вычисления над большим набором данных в цикле, посылая запросы к менеджеру внешней памяти. Если необходимые данные находятся во внешней памяти, менеджер загружает их во внутреннюю память, замещая данные с наиболее длительным отсутствием обращений. Для повышения скорости передачи страницы группируются в пакеты (1 Мб), которые копируются в память графического процессора за один вызов. В результате на следующих итерациях программа получит доступ к запрошенным ранее данным. Вычисления продолжают до тех пор, пока запрашиваются новые страницы (рис. 1.21).

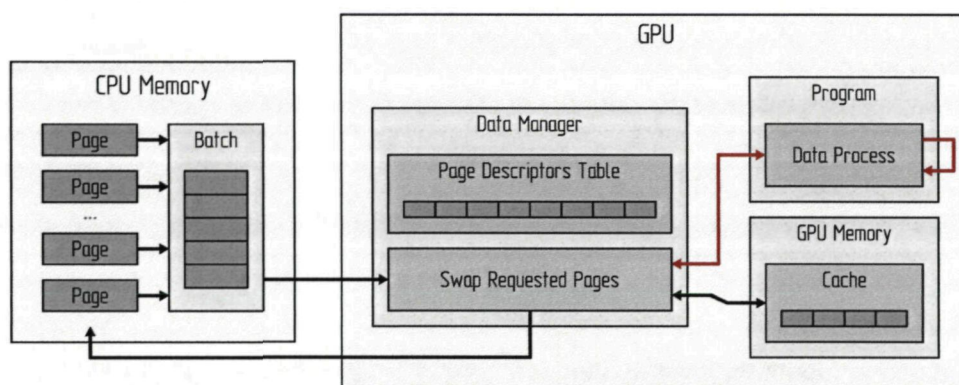


Рис. 1.21. Схема управления данными во внешней памяти

На базе менеджера виртуальной памяти была построена трехуровневая ускоряющая структура, которая позволяет выполнять трассировку для сверхбольших моделей (рис. 1.22). Входная модель разбивается на небольшие блоки, которые содержат менее 4К примитивов. Для этой цели строится временная иерархия объемов, большие узлы которой разбиваются на основе функции SAH-стоимости, а малые – путем сортировки вдоль кривой Мортон.

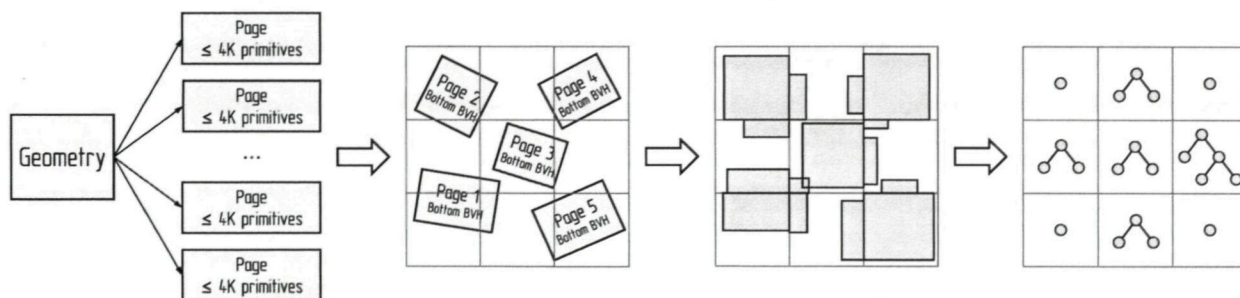


Рис. 1.22. Построение трехуровневой ускоряющей структуры сцены

Для каждого блока строится локальная иерархия объемов, совокупность которых образует *нижний* уровень ускоряющей структуры. Затем ограничивающие параллелепипеды блоков отображаются в мировое пространство согласно заданным аффинным преобразованиям и накрываются регулярной сеткой (с разрешением 16^3 – 128^3). Внутри каждого вокселя сетки строится иерархия объемов для перекрываемых сегментов блоков, которые заключаются в ограничивающие параллелепипеды. Данные иерархии объемов образуют *средний* уровень ускоряющей структуры. При этом глобальная регулярная сетка, каждый воксель которой содержит иерархию среднего уровня, является *верхним* уровнем ускоряющей структуры.

В процессе трассировки луча первые два уровня ускоряющей структуры находятся в памяти графического процессора, а иерархии нижнего уровня подгружаются менеджером виртуальной памяти. Производительность предложенных алгоритмов была протестирована на графическом процессоре NVIDIA GeForce GTX 480 1.5 Гб. Для компьютерных сцен, которые целиком помещаются в видеопамять (менее 10М треугольников), быстродействие оказалась в 1.5–2 раза ниже по сравнению с реализацией [96], что позволяет оценить эффект от накладных расходов процедуры трассировки. На сверхбольшой модели самолета Boeing (360М треугольников) система сохраняла интерактивный режим работы, обрабатывая ~2М лучей в секунду.

Хотя расчет глобального освещения в интерактивном режиме стал возможен совсем недавно (и остается широкий простор для исследований), данные технологии уже вышли за пределы академических проектов. В 2010 году компания NVIDIA представила платформу OptiX – высокоуровневый инструментарий для разработки алгоритмов трассировки лучей на базе NVIDIA CUDA [118]. OptiX позиционируется как гибкий «движок», который допускает тонкую настройку и адаптацию к конкретным задачам. Область применения охватывает различные вычислительно трудоемкие задачи, которые можно решать трассировкой лучей (как универсальным методом анализа и исследования различных геометрических систем). OptiX автоматизирует такие типовые операции, как распараллеливание вычислений (как внутри графического процессора, так и между центральным и графическим процессором), построение ускоряющих структур (*k-d* деревьев и иерархий ограничивающих объемов) и различные алгоритмы обхода. Эффективная утилизация ресурсов графического процессора контролируется встроенным в OptiX модулем балансировки нагрузки. Наряду с этим, OptiX поддерживает графические интерфейсы типа OpenGL, что обеспечивает дополнительную гибкость и позволяет комбинировать традиционную растеризацию и трассировку лучей.

Первым коммерческим визуализатором, выполняющим расчет глобального освещения на графическом процессоре, стал Octane Render компании Refractive Software [126], который

поступил в продажу в конце 2012 года (тестовая версия была доступна ранее). Среди систем с открытым исходным кодом следует отметить проекты LuxRender [127] и Cycles Render [128]. В настоящее время открытые системы визуализации, имеющие поддержку вычислений на графических процессорах, уступают в функциональности и стабильности коммерческим решениям (к качеству которых у пользователей также есть нарекания). Краткая информация об основных доступных на данный момент системах визуализации дана в табл. 1.1.

Таблица 1.1. Основные характеристики современных систем визуализации:

«+» – поддерживается, «○» – поддерживается частично или легко реализуется, «-» – не поддерживается

Особенности	OptiX (NVIDIA)	iRay (Mental Images)	Octane (Refractive Software)	Aion 2 (RandomCentral)	Cycles (Blender Foundation)	FurryBall 3 (Art And Animation)	Indigo Renderer (Gloare Technologies)	Maxwell Render (Next Limit Technologies)	LuxRender	V-Ray RT (Chaos Group)	Mental Ray (Mental Images)
	[118]	[119]	[126]	[120]	[128]	[121]	[122]	[123]	[127]	[124]	[125]
Несмещенная визуализация	○	*	*	*	*		*	*	*	*	
Вычислительная платформа	GPU	Hybrid	GPU	Hybrid	GPU/CPU	GPU	Hybrid	CPU	Hybrid	CPU/GPU	CPU
Модели графических карт	NV	NV	NV	NV	ATI/NV	ATI/NV	ATI/NV	None	ATI/NV	ATI/NV	None
Открыты API визуализации	*	*			*			*	*		
Плагины к популярным системам		*	*	*	*	*	*	*	*	*	*
Автономный визуализатор		*	*	*			*	*	*	*	*
Цена	Free	Часть [125]	199	195	Open Source	999	595	995	Open Source	970	От 995

1.4. Выводы к главе 1

1. Трассировка лучей – прямая и обратная – как способ моделирования распространения света является вычислительной основой и для моделирования глобального освещения. Многие ускоряющие структуры и техники, предложенные для достижения реального времени в трассировке лучей, могут применяться и в глобальном освещении. Богатая история развития трассировки лучей на традиционных архитектурах создала хорошую базу для дальнейшего развития методов на графических процессорах.
2. С появлением первых программируемых графических процессоров (GPU) появилась и развивается тенденция к переносу ускоряющих структур и вычислений трассировки лучей на данную платформу, чему противостоит сложность адаптации наработанных алгоритмов (ограниченность ресурсов и потоковая модель вычислений). Наибольшую сложность для переноса представляют методы построения ускоряющих структур.

3. Наиболее важным направлением в развитии глобального освещения являются методы физически достоверного синтеза несмещенных изображений, которые обеспечивают фотореалистичный результат и имеют принципиальное значение для моделирования виртуальной реальности. Физически обоснованному расчету глобального освещения способствует и усложнение применяемых физико-математических моделей, которые корректно описывают перенос света в сцене. На порядки увеличивается и сложность вычислений, в процессе которых все большую роль играет метод Монте-Карло.
4. Целью многих исследований последних лет, посвященных глобальному освещению, является интерактивный синтез изображений. И здесь также существует тенденция постепенного переноса все большей части вычислений на графические процессоры. Появились и развиваются коммерческие и свободно распространяемые программные комплексы визуализации, которые выполняют значительный объем вычислений на GPU. Однако сложности полного переноса вычислений на GPU в задаче глобального освещения также на порядок выше, чем в задаче трассировки лучей. Значительные трудности возникают при обработке сложных сцен, которые состоят из миллионов треугольников и сочетают традиционные полигональные модели с полупрозрачными средами и другими нетесселированными объектами.

В связи с перечисленным актуальна разработка теоретических основ создания программных систем для информационных технологий виртуальной реальности, которые базируются на интерактивном моделировании глобального освещения сложных сцен (включающих десятки миллионов треугольников и неполигональные объекты), и основ системных решений для их высокопроизводительной реализации на графических процессорах. Наряду с этим, актуально создание и экспериментальное исследование возможностей данных систем и новых методов глобального освещения, построенных на их основе.

Глава 2

Система моделей для алгоритмизации и конвейерной реализации лучевых методов синтеза изображений на массивно-параллельных вычислительных архитектурах

Первым шагом в разработке системного решения для синтеза изображений методами глобального освещения на графическом процессоре служит построение обеспечивающей его системы математических моделей. Данная система должна:

- Основываться на энергетическом подходе и обеспечивать синтез изображений на базе интегрального уравнения визуализации и интегрального уравнения измерения.
- Использовать вычислительно выгодные формы уравнения визуализации и физически корректные расширяемые (универсальные) модели материалов и источников света.
- Содержать аппарат для эффективного вычисления интегралов высоких кратностей на основе методов Монте-Карло, включая аппарат выборки по значимости.
- Быть полной по отношению к проблеме моделирования глобального освещения и, как следствие, способной к развитию и конструированию новых алгоритмов глобального освещения с заданными свойствами.

2.1. Элементы энергетического подхода к физически достоверному моделированию глобального освещения

В фотометрии применяются две системы единиц. *Энергетическая* система служит для количественной оценки фотометрических величин во всем оптическом диапазоне. *Световая* система величин связана с глазом человека и позволяет количественно оценить возникающее субъективное ощущение света при его облучении. Для описания системы световых величин необходимо ввести относительную спектральную чувствительность человеческого глаза, так как в зависимости от длины волны близкие по силе зрительные ощущения возникают при различной интенсивности излучения. Стандартная функция спектральной чувствительности была построена Международной комиссией по освещению (МКО) и обозначается символом $V(\lambda)$. Данная функция обратно пропорциональна монохроматическим мощностям, которые дают схожие по силе зрительные ощущения. Воздействие потока излучения с длиной волны

$\lambda = 555$ нм принимается за единицу. Функция относительной спектральной чувствительности максимальна в области желто-зеленого области спектра (550–570 нм) и спадает до нуля для красных (380 нм) и фиолетовых (770 нм) лучей. Для аналогичных энергетических и световых величин применяются одинаковые буквенные обозначения, которые помечаются индексами e и v соответственно. Когда нет необходимости различать данные величины, индексы могут быть опущены.

2.1.1. Две системы фотометрических величин в моделировании глобального освещения

Лучистый поток и световой поток. Исходным физическим понятием при построении системы энергетических и световых величин является переносимая излучением энергия (или лучистая энергия). На практике чаще приходится рассматривать излучаемую, переносимую или воспринимаемую мощность. В системе энергетических или лучистых величин мощность излучения называют лучистым потоком или потоком излучения (англ. radiant flux или radiant power) и обозначают через Φ_e . Для измерения лучистого потока используется общепринятая единица мощности – ватт (Вт, W). Если излучение является сложным (состоит из нескольких монохроматических излучений), то полный лучистый поток есть сумма монохроматических потоков. В частности, для линейчатого спектра можно записать:

$$\Phi_e = \sum_{i=1}^n \Phi_e(\lambda_i) \quad (2.1)$$

Здесь $\Phi_e(\lambda_i)$ – лучистые потоки линий с длинами волн λ_i . Суммирование производится по всем n линиям, излучаемым источником в диапазоне длин волн от нуля до бесконечности. Для сплошного спектра полный лучистый поток определяется интегрированием:

$$\Phi_e = \int_0^{\infty} \Phi_{\lambda} d\lambda \quad (2.2)$$

Здесь Φ_{λ} – спектральная плотность лучистого потока, которая выражает поток на единичный интервал длин волн:

$$\Phi_{\lambda} = \frac{d\Phi_e}{d\lambda} \quad (2.3)$$

В системе световых величин лучистому потоку соответствует световой поток – мощность излучения, оцениваемая по его действию на «средний» человеческий глаз. Для линейчатого спектра световой поток определяется по формуле:

$$\Phi_v = K_m \sum_{i=1}^n \Phi_e(\lambda_i) V(\lambda_i) \quad (2.4)$$

Суммировать нужно во всем диапазоне, в котором произведение $\Phi_e(\lambda_i)V(\lambda_i)$ отличается от нуля. Очевидно, что излучение в невидимой области спектра не влияет на световой поток. Коэффициент K_m является нормирующим множителем, который зависит от выбора единицы светового потока. Исторически для светового потока была принята независимая единица – люмен (лм, lm), который равен 1/683 ватта при $\lambda = 555$ нм. Поэтому для получения светового потока в люменах (когда лучистые потоки линий заданы в ваттах) необходимо принять $K_m = 683$ лм/Вт. Для сплошного спектра световой поток определяется выражением:

$$\Phi_v = K_m \int_0^{\infty} \Phi_\lambda V(\lambda) d\lambda \quad (2.5)$$

Последние две формулы показывают, почему условный характер видимой области спектра практически не сказывается на определении световых величин. Функция $V(\lambda)$ вблизи границ уменьшается очень быстро, поэтому изменение пределов суммирования (интегрирования) на некоторый интервал около границы незначительно влияет на результат.

Сила излучения и сила света. Большинство реальных источников испускают излучение в различных направлениях неодинаково. Для характеристики пространственного (углового) распределения лучистого потока используется сила излучения или энергетическая сила света (англ. radiant intensity):

$$I_e = \frac{d\Phi_e}{d\omega} \quad (2.6)$$

Данная величина определяется как отношение потока излучения, распространяющегося от источника внутри телесного угла, к величине этого телесного угла. Единица измерения силы излучения – ватт на стерадиан (Вт/ср).

Соответствующая световая величина называется силой света (англ. luminous intensity) и определяется аналогичным образом:

$$I_v = \frac{d\Phi_v}{d\omega} \quad (2.7)$$

Единица силы света – люмен на стерадиан – имеет специальное название кандела (кд, cd).

Для наглядного представления распределения силы света (излучения) в пространстве пользуются понятием фотометрического тела излучателя. Фотометрическое тело образуется

поверхностью, являющейся геометрическим местом концов всех радиус-векторов, которые представляют силу света (излучения) источника. В зависимости от формы фотометрического тела источники подразделяют на симметричные (фотометрическое тело которых имеет ось или плоскость симметрии) и несимметричные (фотометрическое тело которых не является симметричным). Для симметричных источников достаточно определить меридиональное (продольное) сечение фотометрического тела – кривую силы света.

Облученность и освещенность. Для описания поверхностной плотности лучистого и светового потока вводится облученность (англ. irradiance) и освещенность (англ. illuminance) соответственно. Указанные величины определяются как отношение потока, падающего на малый участок поверхности, к площади этого участка:

$$E = \frac{d\Phi}{dA} \quad (2.8)$$

Облученность измеряется в ваттах на квадратный метр. Единица измерения освещенности – люмен на квадратный метр – носит название люкс (лк, lx).

Светимость и энергетическая светимость. Для некоторых практических задач нужно знать распределение потока от источника по его поверхности – поверхностную плотность потока на источнике:

$$M = B = \frac{d\Phi}{dA} \quad (2.9)$$

Для светового потока данная величина называется светимостью (англ. luminous emittance), а для лучистого – плотностью излучения, излучательностью или энергетической светимостью (англ. radiant exitance или radiosity). Данные величины имеют те же самые размерности, что и освещенность и облученность. Для энергетической светимости – ватт на квадратный метр, для светимости – люмен на квадратный метр (название люкс в этом случае не применяется).

2.1.2. Яркость и энергетическая яркость

Светимость определяет поток, излучаемый единицей поверхности источника в данной точке, но не дает информации о том, как этот поток распределяется по направлениям. Любой элемент поверхности может излучать в пределах половины всех возможных в пространстве направлений, которые заключены в телесном угле 2π (если элемент поверхности источника излучает только во внешнюю половину пространства). При этом распределение излучения в указанных пределах может быть различным. Для характеристики данного распределения в

системе световых величин вводится понятие яркости (англ. luminance), а в энергетической системе – лучистой или энергетической яркости (англ. radiance).

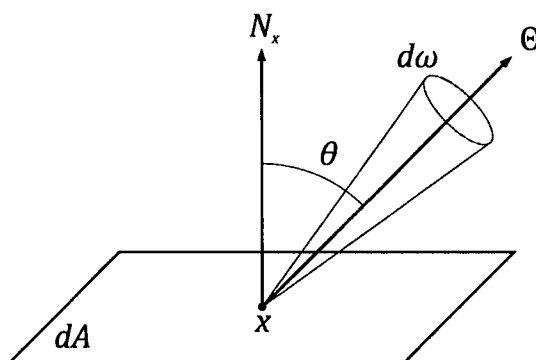


Рис. 2.1. Определение (энергетической) яркости

Яркость данной точки источника в данном направлении определяется как отношение силы света (излучения) элемента поверхности в выбранном направлении к площади его проекции на плоскость, перпендикулярную этому направлению (рис. 2.1):

$$L = \frac{dI}{dA^\perp} \quad (2.10)$$

Здесь dA^\perp – площадь проекции элемента поверхности dA . Если θ – угол между нормалью к поверхности и выбранным направлением, то

$$L = \frac{dI}{dA \cos \theta} \quad (2.11)$$

Раскрывая в данном выражении силу света, получим другое определение яркости:

$$L = \frac{d^2\Phi}{d\omega dA \cos \theta} \quad (2.12)$$

Энергетическая яркость измеряется в ваттах настерадиан и на квадратный метр. Единицей яркости служит кандела на квадратный метр.

В отличие от всех описанных выше величин яркость является второй производной от потока (по углу и по площади), и ее физический смысл не так нагляден, как смысл первых производных. Однако именно эту величину непосредственно оценивает глаз – поверхности с равной яркостью представляются глазу одинаково светлыми. Иными словами, для расчета изображения алгоритмы глобального освещения должны вычислять энергетическую яркость.

2.1.1. Взаимосвязь яркости с остальными фотометрическими величинами и ее свойства в глобальном освещении

Из сформулированных определений фотометрических величин вытекают следующие соотношения:

$$\Phi = \int_A \int_H L(\mathbf{x} \rightarrow \Theta) \cos \theta d\omega_\Theta dA_x \quad (2.13)$$

$$E(\mathbf{x}) = \int_H L(\mathbf{x} \leftarrow \Psi) \cos \theta d\omega_\Psi \quad (2.14)$$

$$B(\mathbf{x}) = \int_H L(\mathbf{x} \rightarrow \Theta) \cos \theta d\omega_\Theta \quad (2.15)$$

Здесь через A обозначена площадь поверхности, а через H – полусфера направлений вокруг точки \mathbf{x} . В настоящей работе используется следующая общепринятая символика: $L(\mathbf{x} \rightarrow \Theta)$ обозначает энергетическую яркость, покидающую точку \mathbf{x} в направлении Θ , а $L(\mathbf{x} \leftarrow \Psi)$ – энергетическую яркость, падающую в точку \mathbf{x} по направлению Ψ .

Таким образом, все остальные фотометрические величины могут быть выражены через энергетическую яркость. Алгоритмы синтеза изображений опираются на следующие важные свойства яркости.

Инвариант яркости вдоль луча. Яркость постоянна (или инвариантна) вдоль луча при отсутствии потерь энергии:

$$L_e = const \quad (2.16)$$

Если среда неоднородна (показатель преломления меняется), то используется приведенная яркость (инвариант яркости):

$$\frac{L_e}{\eta^2} = const \quad (2.17)$$

Из инварианта яркости вытекают два важных для геометрической оптики следствия:

- Яркость является основной характеристикой передачи световой энергии оптической системой.
- Оптическая система в принципе не может увеличить яркость проходящего через нее излучения (она может лишь уменьшить яркость за счет поглощения или рассеяния света).

Таким образом, в отсутствие поглощающей среды яркость достаточно вычислить только на *поверхностях* компьютерной сцены. Данный принцип лежит в основе большинства лучевых алгоритмов глобального освещения (и применяется в настоящей работе).

Чувствительность оптических датчиков к яркости. Реакция оптического датчика (в качестве которого выступает виртуальная камера или человеческий глаз) пропорциональна падающей яркости. В совокупности оба свойства объясняют, почему воспринимаемый цвет или яркость объекта не меняются с расстоянием.

2.2. Моделирование рассеяния света на поверхностях материалов

Испускаемая источниками энергия взаимодействует с объектами различным образом. В общем случае некоторая часть света отражается или проходит сквозь поверхность, а другая поглощается и рассеивается в виде тепла (данное явление явно не моделируется в методах визуализации). В данном разделе рассматриваются основные модели взаимодействия света с поверхностями, необходимые для алгоритмизации задач глобального освещения. Принято допущение, что падающая световая энергия покидает поверхность немедленно и с прежней длиной волны (игнорируются такие явления как флуоресценция и фосфоресценция).

2.2.1. Моделирование оптических свойств поверхностей на основе двулучевой функции распределения рассеяния (BSDF)

В наиболее общем случае световая энергия падает на поверхность в некоторую точку P с направления Ψ и покидает поверхность в другой точке Q по направлению Θ . Для описания зависимости между падающей и отраженной световой энергией в общем случае применяется двулучевая функция поверхностно-рассеянной отражающей способности (англ. Bidirectional Surface Scattering Reflectance Distribution Function, BSSRDF) [131].

Довольно часто можно принять допущение, что падающая в некоторую точку энергия покидает поверхность из этой же точки, при этом игнорируется эффект подповерхностного рассеяния (англ. Subsurface Scattering, SSS). В этом случае отражающие свойства могут быть описаны более простой двулучевой функцией отражающей способности (англ. Bidirectional Reflectance Distribution Function, BRDF). BRDF в каждой точке \mathbf{x} поверхности определяется как отношение приращения (энергетической) яркости, отраженной вдоль направления Θ , к приращению облученности, падающей с направления Ψ (рис. 2.2):

$$f_r(\mathbf{x}, \Psi \rightarrow \Theta) = \frac{dL(\mathbf{x} \rightarrow \Theta)}{dE(\mathbf{x} \leftarrow \Psi)} = \frac{dL(\mathbf{x} \rightarrow \Theta)}{L(\mathbf{x} \leftarrow \Psi) \cos(N_{\mathbf{x}}, \Psi) d\omega_{\Psi}} \quad (2.18)$$

Через $\cos(N_x, \Psi)$ обозначен косинус угла между вектором нормали в точке x и направлением падающего света.

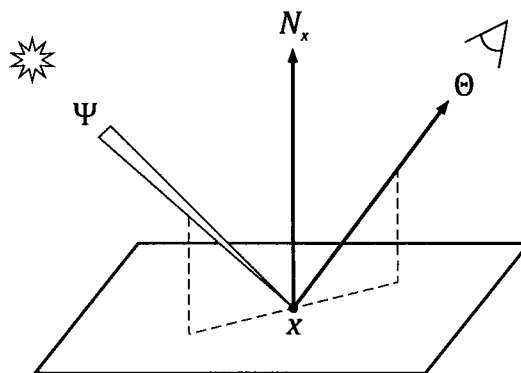


Рис. 2.2. Определение двулучевой функции отражающей способности (BRDF)

Аналогичным образом для обратной стороны поверхности определяется двулучевая функция пропускающей способности (англ. Bidirectional Transmittance Distribution Function, BTDF). В большинстве случаев обе функции – BRDF и BTDF – обрабатываются одинаково, поэтому для их обозначения удобно применять общий термин V_xDF .

В некоторых источниках для совместного описания отражаемой и пропускаемой части падающей световой энергии вводится двулучевая функция распределения рассеяния (англ. Bidirectional Scattering Distribution Function, BSDF). Данная функция определяется для всей сферы направлений вокруг точки x и включает в себя четыре отдельные функции: по одной двунаправленной функции отражающей и пропускающей способности для *каждой* стороны поверхности. В настоящем исследовании BSDF используется в качестве основной модели материала.

2.2.2. Классы и свойства V_xDF , применяемые в методах глобального освещения

V_xDF называется *изотропной*, если она описывает отражающие (или пропускающие) свойства, инвариантные относительно поворота вокруг нормали в точке x . Иными словами, при вращении поверхности вокруг нормали значение V_xDF (а также доля отражаемого или пропускаемого света) не меняется. Для описания реальных материалов часто применяются *анизотропные* V_xDF , которые могут менять свое значение при повороте поверхности вокруг нормали. Тем не менее, понятие изотропии широко применяется в компьютерной графике, и многие аналитические модели V_xDF попадают в этот класс.

Далее перечисляются основные свойства BRDF, которые используются в алгоритмах глобального освещения. Данные свойства справедливы и для BTDF, если явным образом не указано обратное.

Размерность. BRDF является четырехмерной функцией, заданной для каждой точки \mathbf{x} на поверхности материала. Первые два измерения определяют направление падающего света Ψ , а вторые два – направление отраженного света Θ .

Область значений. BRDF может принимать любое неотрицательное значение и может зависеть от длины волны λ . В предельном случае (идеальное отражение) BRDF выражается δ -функцией, которая отлична от нуля только для одного направления, где она обращается в бесконечность.

Симметричность. Значение BRDF остается неизменным при обращении падающего и отраженного направления. Данное свойство называют принципом обратимости Гельмгольца и записывают следующим образом:

$$f_r(\mathbf{x}, \Psi \rightarrow \Theta) = f_r(\mathbf{x}, \Theta \rightarrow \Psi) \quad (2.19)$$

Иными словами, при обращении направления распространения света количество отражаемой энергии не меняется. В силу этого для обозначения BRDF применяют запись $f_r(\mathbf{x}, \Psi \leftrightarrow \Theta)$, которая явным образом указывает на возможность перестановки направлений.

Для BTDF принцип обратимости соблюдается не всегда. Переход узкого пучка света из среды оптически менее плотной в более плотную среду приводит к его «уплотнению». Такое поведение является прямым следствием закона преломления Снеллиуса, согласно которому луч отклоняется в сторону нормали. Поток излучения на единицу перпендикулярной пучку площади увеличивается, поэтому возрастает и энергетическая яркость. Когда луч переходит из среды оптически более плотной в менее плотную среду, происходит обратный процесс. Изменение «плотности» луча определяется квадратом отношения показателей преломления двух сред – $(\eta_2/\eta_1)^2$. При расчете освещения сцен с прозрачными объектами данный фактор необходимо учитывать с помощью соответствующего весового коэффициента.

Связь между падающим и отраженным излучением. Значение BRDF для заданной пары направлений не зависит от облучения по другим возможным направлениям. В этом смысле BRDF выступает как линейная функция по отношению к различным направлениям падения. Данное свойство позволяет вычислить суммарную отраженную яркость через интеграл по всем возможным направлениям облучения:

$$L(\mathbf{x} \rightarrow \Theta) = \int_H f_r(\mathbf{x}, \Psi \rightarrow \Theta) dE(\mathbf{x} \leftarrow \Psi) \quad (2.20)$$

$$L(\mathbf{x} \rightarrow \Theta) = \int_H f_r(\mathbf{x}, \Psi \rightarrow \Theta) L(\mathbf{x} \leftarrow \Psi) \cos(N_{\mathbf{x}}, \Psi) d\omega_{\Psi} \quad (2.21)$$

Сохранение энергии. В каждой точке \mathbf{x} поверхности суммарная отраженная энергия не может превосходить суммарной падающей энергии (часть энергии преобразуется в тепловую и другие формы). Данный принцип накладывает ограничение и на функции, которые могут быть использованы в качестве BRDF:

$$\forall \Psi: \int_H f_r(\mathbf{x}, \Psi \rightarrow \Theta) \cos(N_{\mathbf{x}}, \Theta) d\omega_{\Theta} \leq 1 \quad (2.22)$$

В алгоритмах глобального освещения часто применяются аналитические модели BRDF различного типа. При построении физически достоверных моделей необходимо учитывать закон сохранения энергии и принцип обратимости Гельмгольца.

2.3. Исследование специализированных форм уравнения визуализации, подлежащих реализации в методах глобального освещения

Уравнение визуализации определяет стационарное распределение световой энергии в сцене и впервые было предложено в середине 1980-х годов в публикации [4]. В классическом варианте данное уравнение не учитывает воздействие среды, которая может поглощать или рассеивать свет. Кроме того, принимается допущение, что свет распространяется мгновенно (стационарное состояние достигается немедленно). В данной работе уравнение визуализации применяется в нескольких специализированных формах, которые позволяют оптимизировать вычисления. Данные формы являются компонентами универсальной модели.

2.3.1. Полусферическая форма уравнения визуализации для специализации интегрирования по направлениям

Из закона сохранения энергии следует, что полная исходящая яркость $L(\mathbf{x} \rightarrow \Theta)$ может быть представлена суммой излучаемой яркости $L_e(\mathbf{x} \rightarrow \Theta)$ и суммарной отражаемой яркости $L_r(\mathbf{x} \rightarrow \Theta)$:

$$L(\mathbf{x} \rightarrow \Theta) = L_e(\mathbf{x} \rightarrow \Theta) + L_r(\mathbf{x} \rightarrow \Theta) \quad (2.23)$$

Уравнение визуализации в *полусферической* форме выражает отражаемую яркость $L_r(\mathbf{x} \rightarrow \Theta)$ через интеграл по полусфере H вокруг точки \mathbf{x} :

$$L(\mathbf{x} \rightarrow \Theta) = L_e(\mathbf{x} \rightarrow \Theta) + \int_H f_r(\mathbf{x}, \Psi \rightarrow \Theta) L(\mathbf{x} \leftarrow \Psi) \cos(N_x, \Psi) d\omega_\Psi \quad (2.24)$$

Уравнение визуализации принадлежит к классу интегральных уравнений Фредгольма второго рода. Незвестная величина – энергетическая яркость – содержится одновременно в левой части уравнения и в правой части под знаком интеграла Фредгольма.

2.3.2. Площадная форма уравнения визуализации для специализации интегрирования по поверхностям

В данной форме уравнения суммарная отражаемая яркость $L_r(\mathbf{x} \rightarrow \Theta)$ выражается через облученность от всех поверхностей сцены, видимых из точки \mathbf{x} . При этом интегрирование по полусфере заменяется интегрированием по поверхностям (рис. 2.3).

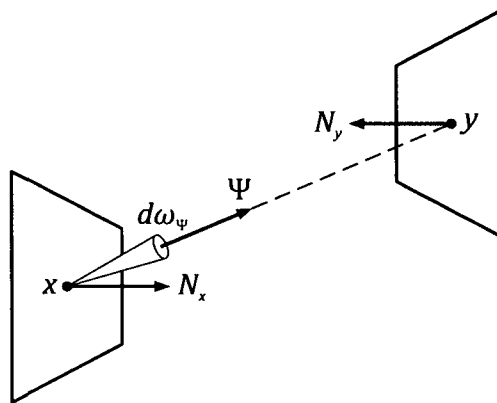


Рис. 2.3. Вывод площадной формы уравнения визуализации

Чтобы записать уравнение в этой форме, необходимо определить операцию испускания (или бросания) луча $R(\mathbf{x} \rightarrow \Psi)$. Данная операция вычисляет ближайшую точку соударения луча с началом в точке \mathbf{x} и направлением Ψ с объектами сцены:

$$R(\mathbf{x} \rightarrow \Psi) = \{\mathbf{y} \mid \mathbf{y} = \mathbf{x} + \tau\Psi\} \quad (2.25)$$

$$\tau = \min_{t>0} \{t \mid \mathbf{x} + t\Psi \in A\}$$

Через A здесь обозначена совокупность всех поверхностей сцены. В компьютерной графике разработаны эффективные техники трассировки лучей, в основе которых лежат различные ускоряющие структуры. В настоящем исследовании для этих целей применяется иерархия ограничивающих объемов, подробная реализация которой рассматривается далее. Операция бросания луча позволяет задать функцию видимости $V(\mathbf{x} \leftrightarrow \mathbf{y})$ между точками \mathbf{x} и \mathbf{y} , которая равна единице, если точки \mathbf{x} и \mathbf{y} взаимно видимы, и нулю в противном случае. Очевидно, что

точки \mathbf{x} и \mathbf{y} взаимно видимы тогда и только тогда, когда существует такое направление Ψ , что $R(\mathbf{x} \rightarrow \Psi) = \mathbf{y}$.

При отсутствии поглощающей среды энергетическая яркость инварианта вдоль любого луча, поэтому для взаимно видимых точек \mathbf{x} и \mathbf{y} справедливо следующее равенство:

$$L(\mathbf{x} \leftarrow \Psi) = L(\mathbf{y} \rightarrow -\Psi) \quad (2.26)$$

Элементарный телесный угол $d\omega_\Psi$, ориентированный в направлении Ψ и опирающийся на малую площадку dA_y , может быть выражен следующим образом:

$$d\omega_\Psi = \cos(N_y, -\Psi) \frac{dA_y}{\|\mathbf{x} - \mathbf{y}\|^2} \quad (2.27)$$

Подстановка данных выражений в полусферическую форму уравнения визуализации позволяет выразить отраженную яркость $L_r(\mathbf{x} \rightarrow \Theta)$ через интеграл по поверхностям сцены:

$$\begin{aligned} L(\mathbf{x} \rightarrow \Theta) &= L_e(\mathbf{x} \rightarrow \Theta) + L_r(\mathbf{x} \rightarrow \Theta) \\ &= \int_A f_r(\mathbf{x}, \Psi \rightarrow \Theta) L(\mathbf{y} \rightarrow -\Psi) V(\mathbf{x} \leftrightarrow \mathbf{y}) \frac{\cos(N_x, \Psi) \cos(N_y, -\Psi)}{\|\mathbf{x} - \mathbf{y}\|^2} dA_y \end{aligned} \quad (2.28)$$

Для сокращения данной записи используется геометрический член $G(\mathbf{x} \leftrightarrow \mathbf{y})$, который определяет взаимное расположение поверхностей в точках \mathbf{x} и \mathbf{y} :

$$G(\mathbf{x} \leftrightarrow \mathbf{y}) = V(\mathbf{x} \leftrightarrow \mathbf{y}) \frac{\cos(N_x, \Psi) \cos(N_y, -\Psi)}{\|\mathbf{x} - \mathbf{y}\|^2} \quad (2.29)$$

В настоящей работе в геометрический член включена и функция видимости между двумя точками (такое определение не является общепринятым), что позволяет записать *площадную* форму уравнения в более компактном виде:

$$L(\mathbf{x} \rightarrow \Theta) = L_e(\mathbf{x} \rightarrow \Theta) + \int_A f_r(\mathbf{x}, \Psi \rightarrow \Theta) L(\mathbf{y} \rightarrow -\Psi) G(\mathbf{x} \leftrightarrow \mathbf{y}) dA_y \quad (2.30)$$

2.3.3. Трехточечная форма уравнения визуализации для специализации интегрирования по траекториям переноса излучения

Данная форма уравнения является специальной записью площадной формы, в которой все направления заменяются точками. Такая форма записи была впервые предложена Вичем в работе [129] и служит удобной отправной точкой для универсального описания различных лучевых алгоритмов визуализации.

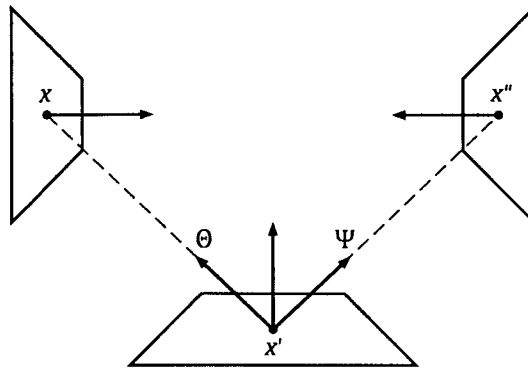


Рис. 2.4. Трехточечная форма уравнения визуализации

Введем другую символику для обозначения энергетической яркости и BRDF, которая вместо направлений использует точки на поверхностях сцены (рис. 2.4):

$$L(\mathbf{x}' \rightarrow \mathbf{x}) = L(\mathbf{x}' \rightarrow \Theta) \quad (2.31)$$

$$f_r(\mathbf{x}'' \rightarrow \mathbf{x}' \rightarrow \mathbf{x}) = f_r(\mathbf{x}, \Psi \rightarrow \Theta) \quad (2.32)$$

Данная запись явным образом указывает на путь переноса излучения. Принятые обозначения позволяют переписать площадную форму уравнения в виде:

$$L(\mathbf{x}' \rightarrow \mathbf{x}) = L_e(\mathbf{x}' \rightarrow \mathbf{x}) + \int_A f_r(\mathbf{x}'' \rightarrow \mathbf{x}' \rightarrow \mathbf{x}) L(\mathbf{x}'' \rightarrow \mathbf{x}') G(\mathbf{x}'' \leftrightarrow \mathbf{x}') dA_{\mathbf{x}''} \quad (2.33)$$

Полученное выражение называется *трехточечной* формой уравнения визуализации.

2.4. Уравнение измерения как основа физически достоверного синтеза изображений

Задачу синтеза изображения можно рассматривать как проблему *измерения* яркости каждого пикселя. Отдельные пиксели экранной плоскости выступают в качестве датчиков, которые фиксируют падающую яркость. Результат измерения зависит от заданной функции отклика W_j , которая определяет чувствительность датчика j к облучению. Конкретной выбор данной функции определяется полнотой моделирования оптической системы.

В общем случае чувствительность пикселя меняется в зависимости от точки (аргумент \mathbf{x}), состояния диафрагмы (аргумент Ψ) и положения затвора (аргумент t). В данной работе зависимость от времени игнорируется, поэтому суммарный отклик датчика j на падающую яркость можно выразить интегралом по его площади I и по полусфере H всех возможных направлений облучения (рис. 2.5):

$$M_j = \int_I \int_H W_j(\mathbf{x} \leftarrow \Psi) L(\mathbf{x} \leftarrow \Psi) \cos(N_{\mathbf{x}}, \Psi) d\omega_{\Psi} dA_{\mathbf{x}} \quad (2.34)$$

Данное выражение называется *уравнением измерения*. Это уравнение играет ключевую роль в компьютерной графике – формулирует основную задачу алгоритмов визуализации.

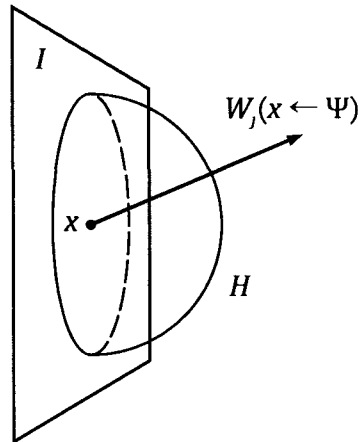


Рис. 2.5. Измерение яркости пикселя для заданной функции отклика

Уравнение измерения допускает различные формы записи (по аналогии с уравнением визуализации). В настоящей работе из альтернативных форм применяется трехточечная:

$$M_j = \int_I \int_A W_j(\mathbf{x}_1 \rightarrow \mathbf{x}_0) L(\mathbf{x}_1 \rightarrow \mathbf{x}_0) G(\mathbf{x}_1 \leftrightarrow \mathbf{x}_0) dA_{\mathbf{x}_1} dA_{\mathbf{x}_0} \quad (2.35)$$

Для сокращения записи подынтегральное выражение заменяется совокупной *функцией измерения* f_j датчика j :

$$f_j(\mathbf{x}_0, \mathbf{x}_1) = W_j(\mathbf{x}_1 \rightarrow \mathbf{x}_0) L(\mathbf{x}_1 \rightarrow \mathbf{x}_0) G(\mathbf{x}_1 \leftrightarrow \mathbf{x}_0) \quad (2.36)$$

$$M_j = \int_{I \times A} f_j(\mathbf{x}_0, \mathbf{x}_1) dA_{\mathbf{x}_0} dA_{\mathbf{x}_1} \quad (2.37)$$

2.5. Моделирование пространства путей для решения уравнения измерения методом Монте-Карло

Уравнение визуализации описывает распределение световой энергии через *локальное* равновесие в каждой точке компьютерной сцены. В классической форме данное уравнение рекурсивно, поскольку неизвестная величина – энергетическая яркость – содержится в обеих

частях уравнения. При отсутствии поглощающей среды энергетическую яркость под знаком интеграла можно также выразить через уравнение визуализации. Рекурсивное применение данной процедуры бесконечное число раз приводит к следующей форме уравнения:

$$\begin{aligned}
 L(\mathbf{x}_1 \rightarrow \mathbf{x}_0) &= L_e(\mathbf{x}_1 \rightarrow \mathbf{x}_0) + L_r(\mathbf{x}_1 \rightarrow \mathbf{x}_0) \\
 &= L_e(\mathbf{x}_1 \rightarrow \mathbf{x}_0) + \sum_{k=2}^{\infty} \int_{M^{k-1}} l(\mathbf{x}_0 \mathbf{x}_1 \dots \mathbf{x}_k) dA_{\mathbf{x}_2} \dots dA_{\mathbf{x}_k}
 \end{aligned}
 \tag{2.38}$$

Данная запись была впервые предложена Э. Вичем в работе [129] и называется уравнением визуализации в *пространстве путей* (переноса излучения). Символом l здесь обозначена *функция переноса излучения*, $l: \bigcup_{i=2}^{\infty} M^{i+1} \rightarrow \mathbb{R}$, которая определяется как (рис. 2.6)

$$l(\mathbf{x}_0 \mathbf{x}_1 \dots \mathbf{x}_k) = \prod_{i=1}^{k-1} f_r(\mathbf{x}_{i+1} \rightarrow \mathbf{x}_i \rightarrow \mathbf{x}_{i-1}) G(\mathbf{x}_i \leftrightarrow \mathbf{x}_{i+1}) L_e(\mathbf{x}_k \rightarrow \mathbf{x}_{k-1})
 \tag{2.39}$$

Уравнение визуализации в пространстве путей больше не является рекурсивным. Суммарная отражаемая яркость здесь выражается через излучение от всех поверхностей сцены, которое достигает заданную точку спустя *любое* число отскоков.

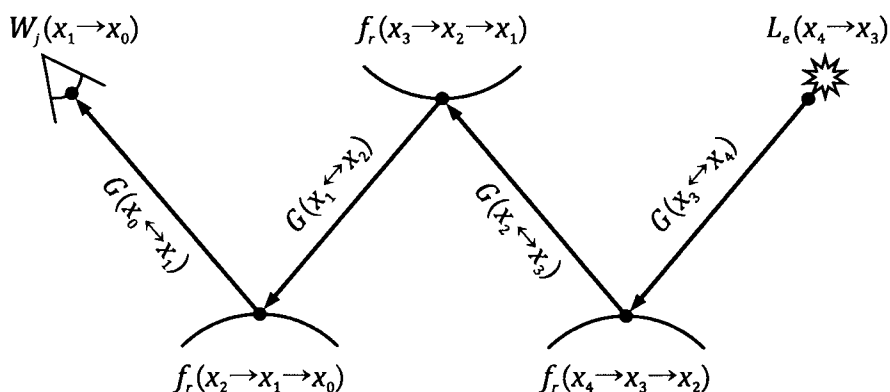


Рис. 2.6. Вычисление вклада пути с помощью функции измерения

Путь переноса излучения \mathbf{X}^k определяется произвольной последовательностью точек $\mathbf{x}_0 \mathbf{x}_1 \dots \mathbf{x}_k \in I \times A^k$, первая вершина которой лежит на картинной плоскости I , а остальные – на поверхностях сцены A . Совокупность всех путей длины k обозначается символом $\Omega_k = I \times A^k$, а пространство всех возможных путей – символом $\Omega = \bigcup_{i=1}^{\infty} \Omega_i$.

Рассмотренное ранее уравнение измерения также может быть записано в пространстве путей. Для этого на базе функции переноса излучения l определяется *обобщенная функция измерения* $f_j: \Omega \rightarrow \mathbb{R}$, которая задана на пространстве путей:

$$\begin{aligned}
f_j(\mathbf{x}_0 \dots \mathbf{x}_k) &= W_j(\mathbf{x}_1 \rightarrow \mathbf{x}_0) G(\mathbf{x}_0 \leftrightarrow \mathbf{x}_1) \\
&\times \prod_{i=1}^{k-1} f_r(\mathbf{x}_{i+1} \rightarrow \mathbf{x}_i \rightarrow \mathbf{x}_{i-1}) G(\mathbf{x}_i \leftrightarrow \mathbf{x}_{i+1}) L_e(\mathbf{x}_k \rightarrow \mathbf{x}_{k-1}) \\
&= W_j(\mathbf{x}_1 \rightarrow \mathbf{x}_0) \\
&\times \prod_{i=0}^{k-1} G(\mathbf{x}_i \leftrightarrow \mathbf{x}_{i+1}) \prod_{i=1}^{k-1} f_r(\mathbf{x}_{i+1} \rightarrow \mathbf{x}_i \rightarrow \mathbf{x}_{i-1}) L_e(\mathbf{x}_k \rightarrow \mathbf{x}_{k-1})
\end{aligned} \tag{2.40}$$

Данная функция позволяет выразить результат измерения яркости датчика j через интеграл по пространству путей переноса излучения:

$$M_j = \int_{\Omega} f_j(\mathbf{X}) d\Omega(\mathbf{X}) \tag{2.41}$$

$$d\Omega(\mathbf{x}_0 \dots \mathbf{x}_k) = dA_{x_0} \times dA_{x_1} \times \dots \times dA_{x_k} \tag{2.42}$$

Реализованные в рамках настоящего исследования алгоритмы глобального освещения строят приближенную оценку уравнения измерения в пространстве путей методом Монте-Карло.

2.6. Компоненты генерации случайных путей для класса методов глобального освещения

В настоящей работе приняты обозначения: $p_A(\mathbf{x})$ – плотность вероятности, отнесенная к единичной площади dA , $p_{\omega^\perp}(\mathbf{x}' \rightarrow \mathbf{x})$ – плотность вероятности, отнесенная к единичному спроектированному телесному углу $d\omega^\perp$, а $p_\omega(\mathbf{x}' \rightarrow \mathbf{x})$ – плотность вероятности, отнесенная к единичному телесному углу $d\omega$.

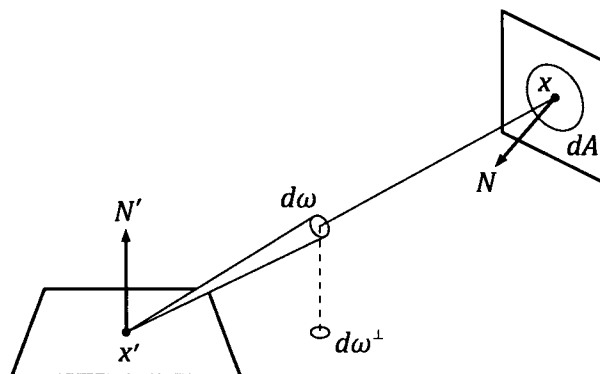


Рис. 2.7. Взаимосвязь между телесным углом, спроектированным телесным углом и площадью поверхности

Между площадью поверхности dA , телесным углом $d\omega$ и спроектированным телесным углом $d\omega^\perp$ действуют соотношения (рис. 2.7):

$$d\omega^\perp = |N_{x'} \cdot (\mathbf{x} - \mathbf{x}')| d\omega = G(\mathbf{x} \leftrightarrow \mathbf{x}') dA \quad (2.43)$$

Таким образом, указанные выше плотности вероятности связаны следующими формулами:

$$p_A(\mathbf{x}) = p_{\omega^\perp}(\mathbf{x}' \rightarrow \mathbf{x}) G(\mathbf{x}' \leftrightarrow \mathbf{x}) \quad (2.44)$$

$$p_{\omega^\perp}(\mathbf{x}' \rightarrow \mathbf{x}) = \frac{p_\omega(\mathbf{x}' \rightarrow \mathbf{x})}{(N_{x'} \cdot (\mathbf{x} - \mathbf{x}'))} \|\mathbf{x} - \mathbf{x}'\| \quad (2.45)$$

2.6.1. Компонент моделирования камеры с протяженной диафрагмой и тонкой линзой

Как правило, в лучевых алгоритмах синтеза изображений процедура генерации первых двух вершин \mathbf{x}_0 и \mathbf{x}_1 зависит от принятой модели оптической системы. В настоящей работе в качестве основной применяется камера с протяженной диафрагмой и тонкой линзой [130].

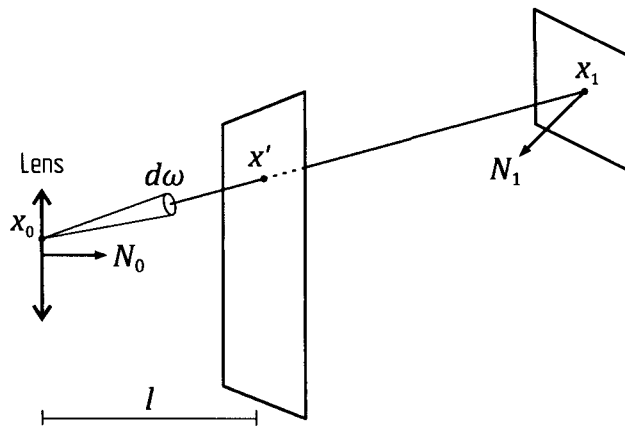


Рис. 2.8. Схема камеры с протяженной диафрагмой и тонкой линзой

Вершина \mathbf{x}_0 выбирается на диафрагме камеры с плотностью вероятности $p_l(\mathbf{x}_0)$. Для вычисления вершины \mathbf{x}_1 на экранной плоскости выбирается случайная точка \mathbf{x}' с плотностью вероятности $p_v(\mathbf{x}')$. Через выбранную точку \mathbf{x}' из вершины \mathbf{x}_0 трассируется луч, ближайшее соударение которого определяет вторую вершину \mathbf{x}_1 . Плотность вероятности $p_v(\mathbf{x}_1)$ задана относительно единичной площади экранной плоскости, однако для вычисления уравнения измерения методом Монте-Карло вероятности всех вершин необходимо задать относительно единичной площади поверхности. Если экранная плоскость находится на расстоянии l от диафрагмы камеры, то можно получить следующие равенства (рис. 2.8):

$$\frac{p_v(\mathbf{x}_1)}{(N_0 \cdot (\mathbf{x}_1 - \mathbf{x}_0))^3} = \frac{p_\omega(\mathbf{x}_0 \rightarrow \mathbf{x}_1)}{l^2 \|\mathbf{x}_1 - \mathbf{x}_0\|^3} = \frac{p_A(\mathbf{x}_1)}{l^2 (N_1 \cdot (\mathbf{x}_0 - \mathbf{x}_1))} \quad (2.46)$$

Данные соотношения позволяют записать плотность вероятности точки \mathbf{x}_1 , отнесенную к единичной площади поверхности dA :

$$p_A(\mathbf{x}_1) = p_V(\mathbf{x}_1) \frac{l^2(N_1 \cdot (\mathbf{x}_0 - \mathbf{x}_1))}{(N_0 \cdot (\mathbf{x}_1 - \mathbf{x}_0))^3} \quad (2.47)$$

Вслед за другими исследованиями, в настоящей работе вместо явного задания функции отклика используется следующее выражение:

$$\widehat{W}_j(\mathbf{x}_1 \rightarrow \mathbf{x}_0) = W_j(\mathbf{x}_1 \rightarrow \mathbf{x}_0) G(\mathbf{x}_0 \leftrightarrow \mathbf{x}_1) \frac{1}{p_I(\mathbf{x}_0) p_A(\mathbf{x}_1)} \quad (2.48)$$

Для камеры с протяженной диафрагмой и тонкой линзой данное выражение принимает вид:

$$\widehat{W}_j(\mathbf{x}_1 \rightarrow \mathbf{x}_0) = W_j(\mathbf{x}_1 \rightarrow \mathbf{x}_0) G(\mathbf{x}_0 \leftrightarrow \mathbf{x}_1) \frac{(N_0 \cdot (\mathbf{x}_1 - \mathbf{x}_0))^3}{l^2(N_1 \cdot (\mathbf{x}_0 - \mathbf{x}_1))} \frac{1}{p_I(\mathbf{x}_0) p_V(\mathbf{x}_1)} \quad (2.49)$$

2.6.2. Компонента генерации случайных путей как основа реализации и вариации алгоритмов глобального освещения

Для применения метода Монте-Карло к уравнению измерения необходимо определить процедуру генерации случайных путей $\mathbf{X}^k = \mathbf{x}_0 \dots \mathbf{x}_k$ согласно некоторой функции плотности вероятности. При этом указанная плотность должна быть задана относительно площадной меры $d\Omega(\mathbf{X}^k)$. Как правило, лучевые алгоритмы визуализации последовательно наращивают путь, присоединяя к нему новые вершины. В результате плотность вероятности, с которой выбирается очередная вершина \mathbf{x}_i , может зависеть от сгенерированных ранее вершин.

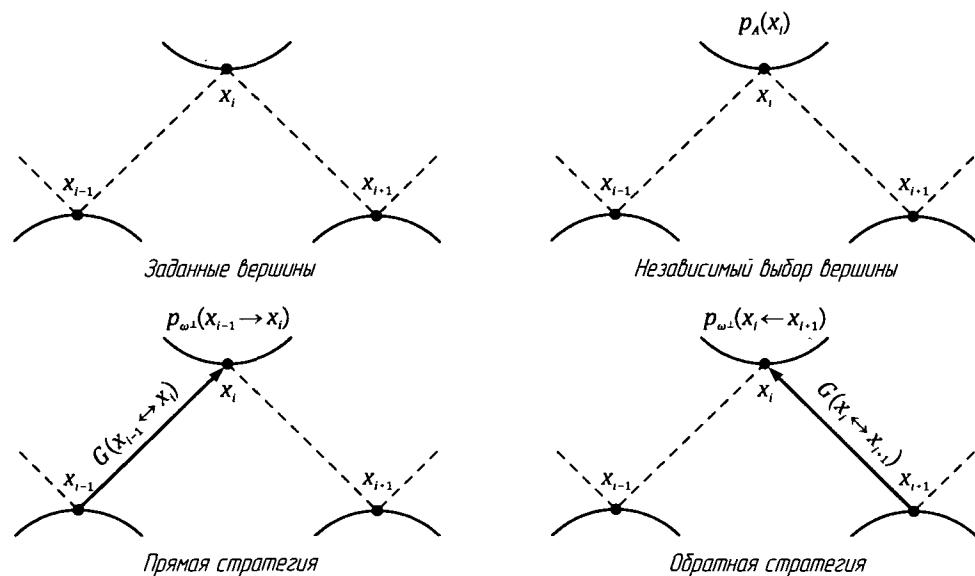


Рис. 2.9. Три стратегии генерации вершины пути

При наличии заданных вершин \mathbf{x}_{i-1} и \mathbf{x}_{i+1} для выбора вершины \mathbf{x}_i применяются три основные стратегии (рис. 2.9). При использовании *первой* стратегии вершина \mathbf{x}_i выбирается на поверхностях сцены случайным образом, независимо от других вершин. В ходе *второй* («прямой») стратегии из точки \mathbf{x}_{i-1} в случайном направлении испускается луч, ближайшая точка пересечения которого определяет вершину \mathbf{x}_i . Аналогично вершина \mathbf{x}_i выбирается и при использовании *третьей* («обратной») стратегии, однако в этом случае луч испускается из точки \mathbf{x}_{i+1} . Все методы генерации случайных путей, которые применяются в настоящем исследовании, являются комбинацией трех перечисленных стратегий.

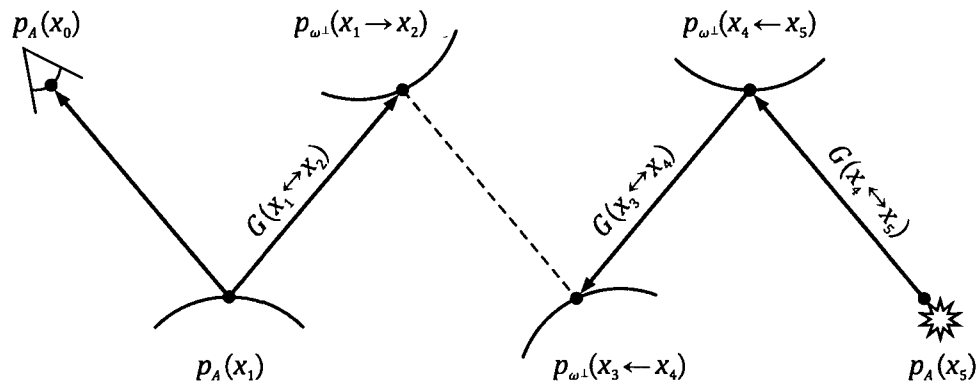


Рис. 2.10. Пример генерации двунаправленного пути

Обычно в случае «прямой» или «обратной» генерации вершины \mathbf{x}_i направление луча выбирается согласно плотности вероятности, отнесенной к единичному спроектированному телесному углу (если плотность вероятности пропорциональна BSDF). При использовании «прямой» стратегии плотность выбора вершины \mathbf{x}_i обозначим символом $p_{\omega^\perp}(\mathbf{x}_{i-1} \rightarrow \mathbf{x}_i)$, а при «обратной» – символом $p_{\omega^\perp}(\mathbf{x}_i \leftarrow \mathbf{x}_{i+1})$. Чтобы получить плотность вероятности $p_A(\mathbf{x}_i)$, отнесенную к единичной площади поверхности, необходимо выполнить преобразования:

$$p_{\omega^\perp}(\mathbf{x}_{i-1} \rightarrow \mathbf{x}_i)G(\mathbf{x}_{i-1} \leftrightarrow \mathbf{x}_i) = p_A(\mathbf{x}_i) = p_{\omega^\perp}(\mathbf{x}_i \leftarrow \mathbf{x}_{i+1})G(\mathbf{x}_i \leftrightarrow \mathbf{x}_{i+1}) \quad (2.50)$$

Если для заданной вершины \mathbf{x}_{s-1} был построен случайный путь $\mathbf{x}_s \dots \mathbf{x}_t$, все вершины которого генерировались «прямым» методом, то его плотность вероятности (отнесенную к единичной площади поверхности) можно выразить следующим образом:

$$p_A(\mathbf{x}_s \dots \mathbf{x}_t) = \prod_{i=s}^t G(\mathbf{x}_i \leftrightarrow \mathbf{x}_{i-1}) \prod_{i=s}^t p_{\omega^\perp}(\mathbf{x}_{i-1} \rightarrow \mathbf{x}_i) \quad (2.51)$$

Аналогично, если для заданной вершины \mathbf{x}_{t+1} случайный путь $\mathbf{x}_s \dots \mathbf{x}_t$ был сгенерирован при помощи «обратного» метода, то его плотность вероятности равна

$$p_A(\mathbf{x}_s \dots \mathbf{x}_t) = \prod_{i=s}^t G(\mathbf{x}_i \leftrightarrow \mathbf{x}_{i+1}) \prod_{i=s}^t p_{\omega^\perp}(\mathbf{x}_i \leftarrow \mathbf{x}_{i+1}) \quad (2.52)$$

Двунаправленные алгоритмы визуализации используют сразу две стратегии построения путей: последовательность $\mathbf{x}_2 \dots \mathbf{x}_s$ генерируется «прямым» методом, а последовательность $\mathbf{x}_t \dots \mathbf{x}_{s+1}$ – «обратным». Плотность вероятности такого комбинированного пути (отнесенную к единичной площади поверхности) можно вычислить по формуле (рис. 2.10):

$$p_A(\mathbf{x}_2 \dots \mathbf{x}_t) = \prod_{i=2}^s G(\mathbf{x}_i \leftrightarrow \mathbf{x}_{i-1}) \prod_{i=s+1}^t G(\mathbf{x}_i \leftrightarrow \mathbf{x}_{i+1}) \times \prod_{i=2}^s p_{\omega^\perp}(\mathbf{x}_{i-1} \rightarrow \mathbf{x}_i) \prod_{i=s+1}^t p_{\omega^\perp}(\mathbf{x}_i \leftarrow \mathbf{x}_{i+1}) \quad (2.53)$$

2.6.3. Компонент многократной выборки по значимости на основе энергетической эвристики

Для интегрирования уравнения измерения применяется метод Монте-Карло, который в базовом виде генерирует N псевдослучайных точек $\mathbf{X}_1 \dots \mathbf{X}_N$ в пространстве путей согласно некоторой плотности вероятности p и строит оценку интеграла по формуле:

$$\langle M_j \rangle = \frac{1}{N} \sum_{i=1}^N \frac{f_j(\mathbf{X}_i)}{p(\mathbf{X}_i)} \quad (2.54)$$

Скорость сходимости данного метода сравнительно мала и определяется величиной $N^{-1/2}$. Чтобы получить следующий знак после запятой (уменьшить погрешность примерно в 10 раз) необходимо в 100 раз увеличить число испытаний N . Однако скорость сходимости $N^{-1/2}$ не зависит от размерности пространства (и сохраняется для интегралов счетной кратности). Эта особенность является ключевой и объясняет широкую популярность метода Монте-Карло в задачах моделирования глобального освещения.

Для уменьшения дисперсии оценки уравнения измерения Э. Вичем в работе [129] был предложен метод многократной выборки по значимости (англ. Multiple Importance Sampling, MIS), который позволяет скомбинировать результаты нескольких способов выборки в одну *несмещенную* оценку. В данной системе моделей для генерации каждой точки пространства путей \mathbf{X}_i используется своя функция плотности вероятности p_i . На основе сгенерированных точек строится оценка интеграла:

$$\langle M_j \rangle = \sum_{i=1}^N w_i(\mathbf{X}_i) \frac{f_j(\mathbf{X}_i)}{p_i(\mathbf{X}_i)} \quad (2.55)$$

В данной формуле символом $w_i(\mathbf{X}_i)$ обозначена весовая функция, которая определяет вклад каждой точки в приближенное значение интеграла. Чтобы оценка оставалась несмещенной, весовые функции должны удовлетворять двум условиям:

$$f_j(\mathbf{X}) \neq 0 \Rightarrow \sum_{i=1}^N w_i(\mathbf{X}) = 1 \quad (2.56)$$

$$p_i(\mathbf{X}) = 0 \Rightarrow w_i(\mathbf{X}) = 0$$

Иными словами, должна существовать хотя бы одна стратегия для выбора каждой значимой точки (в которой функция $f_j(\mathbf{X})$ отлична от нуля). Если существует несколько стратегий для выбора точки \mathbf{X} , то сумма их весов должна быть равна единице.

Для построения корректных весовых функций применяются различные подходы, среди которых естественным выбором является балансовая эвристика (англ. balance heuristic):

$$w_i(\mathbf{X}) = \frac{p_i(\mathbf{X})}{\sum_{j=0}^N p_j(\mathbf{X})} \quad (2.57)$$

При таком выборе каждая точка вносит вклад в результирующую оценку в соответствии со своей «относительной значимостью». В работе Э. Вича было показано, что при отсутствии дополнительной информации балансовая эвристика является наилучшей комбинацией весов среди всех возможных. В настоящей работе применяется обобщенный вариант балансовой эвристики:

$$w_i(\mathbf{X}) = \frac{p_i(\mathbf{X})^\beta}{\sum_{j=0}^N p_j(\mathbf{X})^\beta} \quad (2.58)$$

Данная комбинация весов называется энергетической эвристикой (англ. power heuristic).

2.7. Покомпонентное представление метода стохастической трассировки путей в системе моделей глобального освещения

2.7.1. Компонента расчета вклада явного и неявного путей переноса излучения

Стохастическая трассировка путей является одним из первых алгоритмов глобального освещения, который генерирует несмещенную оценку изображения. В данном методе пути

генерируются с помощью «прямой» стратегии, при этом начальная вершина выбирается на диафрагме виртуальной камеры. Путь $x_0 \dots x_k$ строится за счет поэтапного присоединения новых вершин и завершается в произвольной точке (критерии завершения рассматриваются далее). Каждая вершина x_i соединяется теневым лучом со случайной точкой z на источнике света, за счет чего формируется корректная траектория переноса излучения $x_0 \dots x_i z$. Такие пути называются *явными*. Если в процессе расширения пути очередная вершина x_k оказалась на источнике света, то промежуточная последовательность вершин $x_0 \dots x_k$ также является корректной траекторией переноса излучения. Такие пути называются *неявными*.

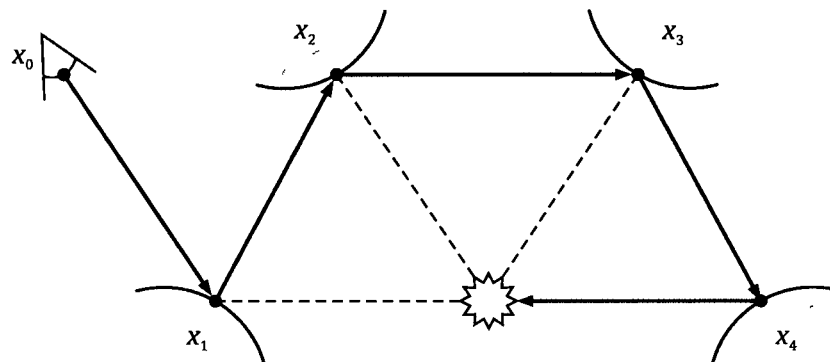


Рис. 2.11. Понятие семпла в стохастической трассировке путей

Для расчета вклада пути (по методу Монте-Карло) необходимо вычислить плотность вероятности, с которой производилась генерация последовательности вершин. Поскольку уравнение измерения выражается через интеграл по площадной мере, указанная плотность должна быть отнесена к единичной площади поверхности. Плотность вероятности *неявного* пути $x_0 \dots x_k$ выражается через плотности выбора его вершин следующим образом:

$$p_I(x_0 \dots x_k) = p_I(x_0) \cdot \prod_{i=1}^k p_A(x_i) = \frac{l^2(N_1 \cdot (x_0 - x_1))}{(N_0 \cdot (x_1 - x_0))^3} \times p_I(x_0)p_V(x_1) \prod_{i=1}^{k-1} G(x_i \leftrightarrow x_{i+1}) \prod_{i=1}^{k-1} p_{\omega^\perp}(x_i \rightarrow x_{i+1}) \quad (2.59)$$

Плотность вероятности *явного* пути $x_0 \dots x_k$ записывается аналогично, однако последняя вершина x_k уже не требует преобразования, поскольку генерируется на источнике света:

$$p_E(x_0 \dots x_k) = p_I(x_0) \cdot \prod_{i=1}^k p_A(x_i) = \frac{l^2(N_1 \cdot (x_0 - x_1))}{(N_0 \cdot (x_1 - x_0))^3} \times p_I(x_0)p_V(x_1) \prod_{i=1}^{k-2} G(x_i \leftrightarrow x_{i+1}) \prod_{i=1}^{k-2} p_{\omega^\perp}(x_i \rightarrow x_{i+1}) p_A(x_k) \quad (2.60)$$

Вклад *неявного* пути $\mathbf{x}_0 \dots \mathbf{x}_k$ в оценку уравнения измерения (по методу Монте-Карло) записывается следующим образом:

$$\frac{f_j(\mathbf{x}_0 \dots \mathbf{x}_k)}{p_I(\mathbf{x}_0 \dots \mathbf{x}_k)} = \widehat{W}_j(\mathbf{x}_1 \rightarrow \mathbf{x}_0) \prod_{l=1}^{k-1} \frac{f_r(\mathbf{x}_{l+1} \rightarrow \mathbf{x}_l \rightarrow \mathbf{x}_{l-1})}{p_{\omega^\perp}(\mathbf{x}_l \rightarrow \mathbf{x}_{l+1})} L_e(\mathbf{x}_k \rightarrow \mathbf{x}_{k-1}) \quad (2.61)$$

В данном выражении геометрические члены отсутствуют, поскольку входят одновременно в числитель и знаменатель дроби. Для сокращения записи вклада применяется введенная ранее модифицированная функция чувствительности \widehat{W}_j . Аналогичным образом можно выразить вклад *явного* пути переноса излучения:

$$\begin{aligned} \frac{f_j(\mathbf{x}_0 \dots \mathbf{x}_k)}{p_E(\mathbf{x}_0 \dots \mathbf{x}_k)} = & \widehat{W}_j(\mathbf{x}_1 \rightarrow \mathbf{x}_0) \prod_{l=1}^{k-2} \frac{f_r(\mathbf{x}_{l+1} \rightarrow \mathbf{x}_l \rightarrow \mathbf{x}_{l-1})}{p_{\omega^\perp}(\mathbf{x}_l \rightarrow \mathbf{x}_{l+1})} \\ & \times \frac{f_r(\mathbf{x}_k \rightarrow \mathbf{x}_{k-1} \rightarrow \mathbf{x}_{k-2}) G(\mathbf{x}_k \leftrightarrow \mathbf{x}_{k-1})}{p_A(\mathbf{x}_k)} L_e(\mathbf{x}_k \rightarrow \mathbf{x}_{k-1}) \end{aligned} \quad (2.62)$$

Поскольку для выбора последней вершины используется плотность вероятности, отнесенная к единичной площади, геометрический член между двумя последними точками присутствует только в числителе и сохраняется в окончательном выражении.

2.7.2. Компонент управления глубиной трассировки, реализующий различные критерии завершения пути

Процедура генерации пути, которая лежит в основе стохастической трассировки путей, нуждается в критерии остановки. В противном случае длина генерируемых путей окажется бесконечной, и алгоритм не завершит свою работу. Следует учитывать, что световая энергия может отражаться от поверхностей сцены произвольное число раз, за счет чего даже очень длинные траектории переноса излучения могут вносить значительный вклад в изображение. Необходим механизм эффективного ограничения длины путей, который позволяет сохранить *несмещенную* оценку изображения. Существует три классических критерия остановки.

Фиксированная глубина трассировки. При таком подходе длина путей ограничивается наперед заданным числом отскоков. Обычно обрабатывается порядка 4–5 отражений, однако оптимальное значение зависит от конкретной сцены. Для обработки значительного числа зеркальных или пропускающих поверхностей необходима большая длина путей, в то время как диффузные сцены позволяют ограничиться всего несколькими отскоками. Данный метод не позволяет получить *несмещенную* оценку и может игнорировать важные пути переноса света (или обрабатывать пути с ничтожным вкладом).

Адаптивная глубина трассировки. Решение об остановке пути принимается на основе его веса, который определяется произведением BSDF в точках соударения, отнесенному к произведению соответствующих плотностей вероятности. Данная величина накапливается в процессе построения пути и позволяет прогнозировать вклад в яркость пикселя. Если вес оказывается меньше заданного порогового значения, то процедура построения останавливает путь. Данная техника значительно эффективнее предыдущего метода, поскольку позволяет быстро останавливать «неперспективные» пути, не внося при этом больших ошибок. Однако данный метод по-прежнему не обеспечивает несмещенную оценку изображения, поскольку не учитывает произвольные пути переноса света.

Метод «русской рулетки». В данном случае длина путей также выбирается адаптивно, однако сохраняется возможность обработки траекторий с любым числом отскоков. Таким образом, метод обеспечивает несмещенную оценку изображения. Принцип работы «русской рулетки» можно продемонстрировать на примере вычисления одномерного интеграла:

$$I = \int_0^1 f(x) dx \quad (2.63)$$

Стандартный метод Монте-Карло генерирует выборочные значения x_i в промежутке $[0, 1]$ и вычисляет среднее для найденных значений функции $f(x_i)$. Перепишем интеграл I в виде:

$$\bar{I} = \frac{1}{P} \int_0^P f\left(\frac{x}{P}\right) dx, \quad P \leq 1 \quad (2.64)$$

Если применить метод Монте-Карло к преобразованному интегралу \bar{I} , то для выборочных значений $x_i > P$ подынтегральное выражение обращается в ноль. Если функция $f(x)$ задана рекурсивно через другой интеграл (по аналогии с классическим уравнением визуализации), то данная рекурсия остановится с вероятностью $\alpha = 1 - P$. Путем варьирования величины α можно добиться оптимального баланса между быстродействием и точностью алгоритма.

В данной работе в качестве основного механизма остановки путей используется метод «русской рулетки». В каждой вершине \mathbf{x}_i вероятность продолжения пути $1 - \alpha$ выбирается пропорционально текущему весу и включается в плотность вероятности выбора следующей вершины $p_{\omega^+}(\mathbf{x}_i \rightarrow \mathbf{x}_{i+1})$.

2.7.3. Компонент выборки лучей при расчете прямого освещения

В процессе генерации пути каждая вершина соединяется теневым лучом со случайной точкой на источнике света. Как правило, компьютерные сцены содержат несколько (много)

источников света. В настоящей работе реализовано два метода расчета прямого освещения для нескольких источников света.

Независимые источники света. При таком подходе каждый источник обрабатывается независимо (для каждого источника генерируется отдельный теневой луч). Результирующий вклад от прямого освещения вычисляется как сумма вкладов от каждого источника. В рамках данной стратегии для каждого источника можно генерировать разное число теневых лучей (пропорционально мощности источника или обратно пропорционально квадрату расстояния до источника).

Один логический источник. В большинстве случаев предпочтительно рассматривать все источники света как один логический источник (с разрывной областью интегрирования). В процессе вычисления прямого освещения выборочные точки могут генерироваться на любом реальном источнике света. Данный подход позволяет рассчитывать освещение от любого числа источников, используя только один теневой луч (при этом получается несмещенная оценка изображения). В общем случае для генерации теневого луча применяется следующая двухшаговая процедура:

- На *первом* шаге с помощью дискретного распределения $p_L(k)$ выбирается источник света k . Для этого каждому источнику приписывается определенная вероятность, с которой он участвует в процессе генерации теневых лучей.
- На *втором* шаге на поверхности источника k выбирается случайная точка \mathbf{z} согласно некоторой условной плотности вероятности $p_A(\mathbf{z}|k)$. Конкретный вид данной функции зависит от выбранного источника.

Комбинированная функция плотности вероятности для выбора случайной точки \mathbf{u} на поверхности логического источника $p_A(\mathbf{z}) = p_L(k)p_A(\mathbf{z}|k)$. Любой выбор функций $p_L(k)$ и $p_A(\mathbf{z}|k)$ обеспечивает генерацию несмещенного изображения. Вместе с тем, конкретный вид данных распределений оказывает значительное влияние на дисперсию получаемых оценок (и на уровень шума в изображении). На практике часто применяются следующие варианты.

Равномерный выбор источника – равномерная выборка точек. В данном случае обе функции задают равномерное распределение: $p_L(k) = 1/N_L$ и $p_A(\mathbf{z}|k) = 1/A_k$, где через N_L обозначено общее число источников, а через A_k – площадь источника k . При таком выборе для каждого источника генерируется примерно одинаковое число теневых лучей, которые равномерно распределяются по его поверхности. Данный метод легко реализуется и является

основным в настоящей работе, однако для некоторых условий освещения может оказаться неэффективным (поскольку игнорирует относительную значимость источников света).

Выбор источника по мощности – равномерная выборка точек. В этом случае источник выбирается согласно дискретному распределению $p_L(k) = \Phi_k/\Phi$, где через Φ_k обозначена мощность источника k , а через Φ – суммарная мощность всех источников сцены. Если все источники являются диффузными, то мощность можно определить по формуле $\Phi_k = \pi A_k L_k$, и плотность вероятности $p_A(\mathbf{z}) = \pi L_k/\Phi$. Обычно данный подход дает хорошие результаты, однако может приводить к медленной сходимости в областях, где яркие источники не видны, и освещение поступает от слабых источников. Для устранения данной проблемы необходимо учитывать видимость источников света, что значительно повышает трудоемкость процедуры генерации теневых лучей. В силу перечисленных особенностей в предлагаемой реализации трассировки путей на графическом процессоре данный метод выборки не применяется.

2.7.4. Компонент выборки лучей при расчете вторичного освещения

Для каждой вершины пути необходимо сгенерировать один вторичный луч, который определяет дальнейший маршрут переноса световой энергии. В отличие от расчета прямого освещения область интегрирования невозможно заранее ограничить, поскольку излучение может падать вдоль любого направления. В общем случае для оценки вторичного освещения случайные лучи необходимо генерировать по всей полусфере направлений. Для этой цели применяются следующие основные методы.

Равномерная выборка направлений. В простейшем случае для генерации направления вторичного луча Ψ используется равномерное распределение $p_\omega(\Psi) = 1/2\pi$. Данный подход полностью игнорирует особенности подынтегрального выражения и приводит к высокому уровню шума. Для понижения дисперсии оценки применяются различные подходы на основе выборки по значимости.

Относительно косинуса. Плотность вероятности $p_\omega(\Psi)$ в некоторой точке \mathbf{x} строится пропорционально косинусу:

$$p_\omega(\Psi) = \frac{(N_x \cdot \Psi)}{\pi} \quad (2.65)$$

Такой подход является универсальным способом выборки при отсутствии дополнительной информации, поскольку косинус в подынтегральном выражении присутствует всегда (входит в геометрический член). В настоящей работе данный метод применяется по умолчанию.

Относительно BSDF. Плотность вероятности $p_\omega(\Psi)$ в некоторой точке \mathbf{x} выбирается пропорционально BSDF поверхности (либо строится «похожая» функция). Данный подход позволяет значительно снизить уровень шума для глянцевых или зеркальных поверхностей.

Как правило, рассеивающие свойства поверхностей описываются *комбинированными* функциями BSDF (f_s), которые включают в себя несколько VxDF-компонент (f):

$$f_s(\mathbf{x}, \Psi \rightarrow \Theta) = \sum_{i=1}^N \alpha_i f_i(\mathbf{x}, \Psi \rightarrow \Theta) \quad (2.66)$$

В настоящей работе для генерации направления вторичного луча применяется следующий трехшаговый алгоритм:

- На *первом* шаге конструируется дискретное распределение $p_F(k)$ для $N + 1$ событий, вероятности которых соответственно равны $q_1 \dots q_{N+1}$. События $1 \dots N$ определяют VxDF-компоненту, относительно которой выбирается направление луча. Последнему событию $N + 1$ соответствует остановка пути ($q_{N+1} = 1 - \sum_{i=1}^N q_i$).
- На *втором* шаге разыгрывается равномерно распределенная случайная величина, по значению которой согласно распределению $p_F(k)$ выбирается VxDF-компонента k .
- На *третьем* шаге для VxDF-компоненты k генерируется случайное направление Ψ согласно условной плотности вероятности $p_\omega(\Psi|k)$, пропорциональной f_k .

Комбинированная плотность вероятности для выбора случайного направления Ψ имеет вид $p_\omega(\Psi) = p_F(k)p_\omega(\Psi|k)$.

Конкретные значения $q_1 \dots q_N$ оказывают значительное влияние на дисперсию оценки по методу Монте-Карло (которая в любом случае будет несмещенной). В рамках наиболее эффективной стратегии данные вероятности определяются пропорционально максимальной отражаемой энергии VxDF-компонент для заданного входного направления. Такой подход работает только для самых простых моделей VxDF (Ламберта или Блинна-Фонга), которые позволяют оценить данное значение аналитически. В настоящей работе вероятности $q_1 \dots q_N$ задаются пропорционально коэффициентам $\alpha_1 \dots \alpha_N$, скорректированным с учетом текущего веса луча. Данный подход является вычислительно эффективным и обеспечивает достаточно хорошие результаты.

Относительно падающей яркости. В рамках данного подхода случайные направления генерируются согласно плотности вероятности, пропорциональной энергетической яркости, отраженной от других поверхностей сцены в точку \mathbf{x} . Поскольку данная величина неизвестна

на этапе построения пути, необходимо использовать различные методы аппроксимации. Для этой может применяться метод фотонных карт, который позволяет быстро получить грубое распределение световой энергии в сцене. Данные подходы достаточно сложны в реализации и встречаются относительно редко. В настоящей работе подобные техники не используются.

2.7.5. Компонент многократной выборки по значимости для комбинирования явных и неявных путей

В трассировке путей каждая траектория переноса излучения может быть сгенерирована «явной» или «неявной» стратегией (в зависимости от способа выбора последней вершины). Согласно многократной выборке по значимости, при вычислении вклада пути необходимо учитывать относительную «значимость» применяемой стратегии. Нетрудно видеть, что все вершины явного и неявного пути, за исключением последней, генерируются одинаково. За счет этого в выражениях для весовых функций (2.58) большинство членов сокращается, и остаются только плотности вероятности последних вершин. Весовая функция для *неявных* путей $\mathbf{x}_0 \dots \mathbf{x}_k$ переноса излучения имеет вид:

$$\begin{aligned} w_I(\mathbf{x}_0 \dots \mathbf{x}_k) &= \frac{p_I(\mathbf{x}_0 \dots \mathbf{x}_k)^\beta}{p_E(\mathbf{x}_0 \dots \mathbf{x}_k)^\beta + p_I(\mathbf{x}_0 \dots \mathbf{x}_k)^\beta} \\ &= \frac{[p_{\omega^\perp}(\mathbf{x}_{k-1} \rightarrow \mathbf{x}_k)G(\mathbf{x}_{k-1} \leftrightarrow \mathbf{x}_k)]^\beta}{[p_A(\mathbf{x}_k)]^\beta + [p_{\omega^\perp}(\mathbf{x}_{k-1} \rightarrow \mathbf{x}_k)G(\mathbf{x}_{k-1} \leftrightarrow \mathbf{x}_k)]^\beta} \end{aligned} \quad (2.67)$$

Аналогичным образом записывается весовая функция для *явных* путей переноса излучения:

$$\begin{aligned} w_E(\mathbf{x}_0 \dots \mathbf{x}_k) &= \frac{p_E(\mathbf{x}_0 \dots \mathbf{x}_k)^\beta}{p_E(\mathbf{x}_0 \dots \mathbf{x}_k)^\beta + p_I(\mathbf{x}_0 \dots \mathbf{x}_k)^\beta} \\ &= \frac{[p_A(\mathbf{x}_k)]^\beta}{[p_A(\mathbf{x}_k)]^\beta + [p_{\omega^\perp}(\mathbf{x}_{k-1} \rightarrow \mathbf{x}_k)G(\mathbf{x}_{k-1} \leftrightarrow \mathbf{x}_k)]^\beta} \end{aligned} \quad (2.68)$$

В работе [129] было показано, что оптимальная комбинация явных и неявных стратегий реализуется для показателя $\beta = 2$.

2.7.6. Реализация трассировки лучей Уиттеда в системе моделей глобального освещения

Классическая трассировка лучей (или трассировка лучей Уиттеда) остается одним из самых популярных методов визуализации. Данная техника поддерживает базовые эффекты глобального освещения и является частным случаем стохастической трассировки путей.

Прямое освещение. Расчет прямого освещения выполняется аналогично стохастической трассировке путей, что обеспечивает корректный расчет теней. Обычно допускаются только точечные и направленные источники света, поэтому стохастические методы моделирования заменяются детерминированными.

Идеальные отражения и преломления. Обычно данные типы отражения и преломления являются единственными вторичными эффектами, которые можно получить в классической трассировке лучей. Для моделирования данных эффектов применяются детерминированные методы. В настоящей работе трассировка лучей Уиттеда применяется как вспомогательный метод для быстрого отображения массивных компьютерных сцен.

2.8. Реализация и модификация двунаправленной трассировки путей на основе системы моделей глобального освещения

2.8.1. Содержание метода двунаправленной трассировки путей

В стохастической трассировке путей все траектории переноса излучения генерируются «прямым» методом, начиная от диафрагмы виртуальной камеры. Данная стратегия не всегда является наиболее эффективной. Характерными примерами служат сцены с преобладанием вторичного освещения (перекрытые источники света) или резкими колебаниями светового поля (каустики). В подобных случаях крайне сложно построить корректные пути переноса излучения, выполняя трассировку от виртуальной камеры.

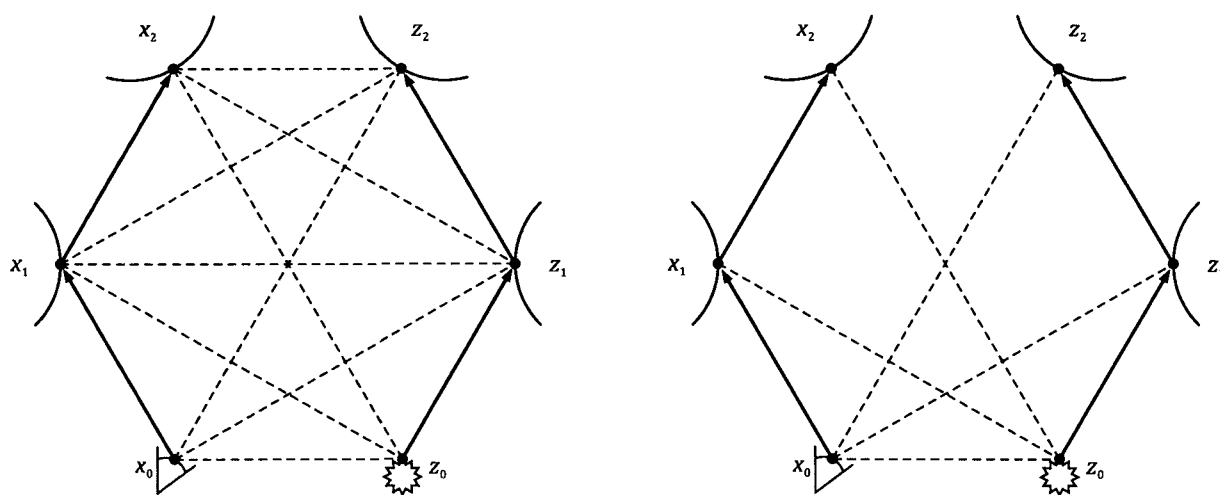


Рис. 2.12. Семпл в «классической» и «усеченной» двунаправленной трассировке путей

Решить данную проблему можно за счет *двунаправленной* трассировки путей, которая была независимо предложена Э. Вичем [129] и Э. Лафорчуном [6]. Данный метод генерирует

пути из источника света («световой» путь) и диафрагмы виртуальной камеры («видовой» путь). Затем все вершины построенных путей соединяются теньевыми лучами, за счет чего образуется множество корректных путей переноса излучения.

В двунаправленной трассировке каждый путь $\mathbf{X}^k = \mathbf{x}_0 \dots \mathbf{x}_k$ длины k можно построить $k + 1$ способом, соединяя «видовой» путь $\mathbf{Y}^s = \mathbf{y}_0 \dots \mathbf{y}_s$ длины $s \geq 0$ со «световым» путем $\mathbf{Z}^t = \mathbf{z}_1 \dots \mathbf{z}_t$ длины $t \geq 0$, где $k = s + t$. При этом любой из двух путей может быть нулевой длины. Если длина «светового» пути равна нулю, то последовательность «видовых» вершин является неявным путем и должна заканчиваться на источнике света. Если «видовой» путь имеет нулевую длину (содержит только одну вершину), то «световой» путь явно соединяется с виртуальной камерой. В настоящей работе не рассматривается ситуация, когда «световой» путь непосредственно пересекает диафрагму виртуальной камеры. Как правило, вклад таких путей в окончательное изображение пренебрежимо мал.

Обозначим символом $p_s(\mathbf{X}^k)$ плотность вероятности пути переноса излучения, который был построен соединением «видового» пути длины s и «светового» пути длины $t = k - s$. Различные способы соединения ($s = 0 \dots k$) определяют различные стратегии генерации пути \mathbf{X}^k , которым соответствуют определенные плотности вероятности. Согласно многократной выборке по значимости, при вычислении вклада пути необходимо учитывать относительный вес реализованной стратегии, который определяется энергетической эвристикой:

$$w_s(\mathbf{X}) = \frac{p_s(\mathbf{X})^\beta}{\sum_{i=0}^k p_i(\mathbf{X})^\beta} \quad (2.69)$$

Стандартный вариант двунаправленной трассировки является достаточно ресурсоемкой техникой для реализации на графическом процессоре. Для вычисления вкладов отдельных комбинированных путей необходимо сгенерировать и сохранить *все* вершины «светового» и «видового» пути. В работе [115] для хранения двунаправленного семпла при максимальной длине пути 16 (в каждом направлении) требуется около 3.5 Кб памяти. Данный показатель более чем в 20 раз превосходит соответствующие затраты памяти для обычной трассировки путей. В результате для синтеза изображения размером 1024×1024 пикселей необходимо свыше 3 Гб памяти только для хранения двунаправленных семплов. Проблема может быть частично решена за счет обработки изображения порциями небольшого размера. Однако для эффективной балансировки нагрузки и утилизации ресурсов графический процессор должен обрабатывать потоки данных, число элементов которых как минимум в несколько раз выше по сравнению с максимальным числом одновременно работающих потоков (десятки тысяч). С учетом этого нижнего ограничения затраты памяти остаются весьма значительными и не

опускаются ниже ~500 Мб. Следует учитывать, что для обработки некоторых сцен (таких как крупный план прозрачного объекта) максимальной длины 16 может оказаться недостаточно, что приведет к смещенной оценке изображения (или к росту потребления памяти). С другой стороны, современные графические карты снабжаются относительно небольшим объемом памяти (1–4 Гб), которую необходимо максимально экономно использовать для хранения геометрии сцены, ускоряющей структуры и текстур.

2.8.2. Построение нового алгоритма «усеченной» двунаправленной трассировки путей на основе компонентов системы моделей

В силу перечисленных обстоятельств в настоящей работе предлагается «усеченный» вариант двунаправленной трассировки путей, который по затратам сопоставим с обычной трассировкой и сохраняет основные преимущества исходного алгоритма. Данный алгоритм формирует изображение за два этапа. На *первом* этапе выполняется обратная трассировка от объектива виртуальной камеры, в ходе которой генерируются явные и неявные траектории переноса излучения. На *втором* этапе выполняется прямая трассировка от источника света, в процессе которой все вершины пути явно соединяются с камерой. Вклады различных путей комбинируются в одну несмещенную Монте-Карло оценку с использованием многократной выборки по значимости. Данный метод можно рассматривать как частный случай «полной» двунаправленной трассировки, при котором один из соединяемых путей содержит не более одной вершины ($s \leq 1$ или $t \leq 1$). Основные преимущества такого подхода определяются следующими особенностями:

- На стадии прямой и обратной трассировки для обработки путей *произвольной* длины требуется фиксированный объем памяти. Данное обстоятельство позволяет загрузить графический процессор большим числом «легковесных» путей и снимает ограничения при синтезе несмещенных изображений (когда необходимо большое число отскоков).
- Трассировка путей в каждом направлении выполняется независимо, что обеспечивает широкие возможности распараллеливания. Например, «прямой» и «обратный» проход визуализации можно выполнять на различных графических процессорах без какого-либо взаимодействия в ходе вычислений (для получения результата два изображения достаточно просто сложить).
- Полностью исключается трудоемкая фаза соединения «прямого» и «обратного» путей (требует информации обо всех вершинах каждого пути).

- Значительно упрощается реализация многократной выборки по значимости, которая позволяет оптимально скомбинировать вклады прямых и обратных путей.

В методе «усеченной» двунаправленной трассировки каждая траектория $\mathbf{X}^k = \mathbf{x}_0 \dots \mathbf{x}_k$ может быть построена тремя способами: как *явный* путь $\mathbf{y}_0 \dots \mathbf{y}_{k-1} \mathbf{z}$ или *неявный* путь $\mathbf{y}_0 \dots \mathbf{y}_k$ в ходе обратной трассировки или как *световой* путь $\mathbf{z}_0 \dots \mathbf{z}_{k-1} \mathbf{y}$ в ходе прямой трассировки.

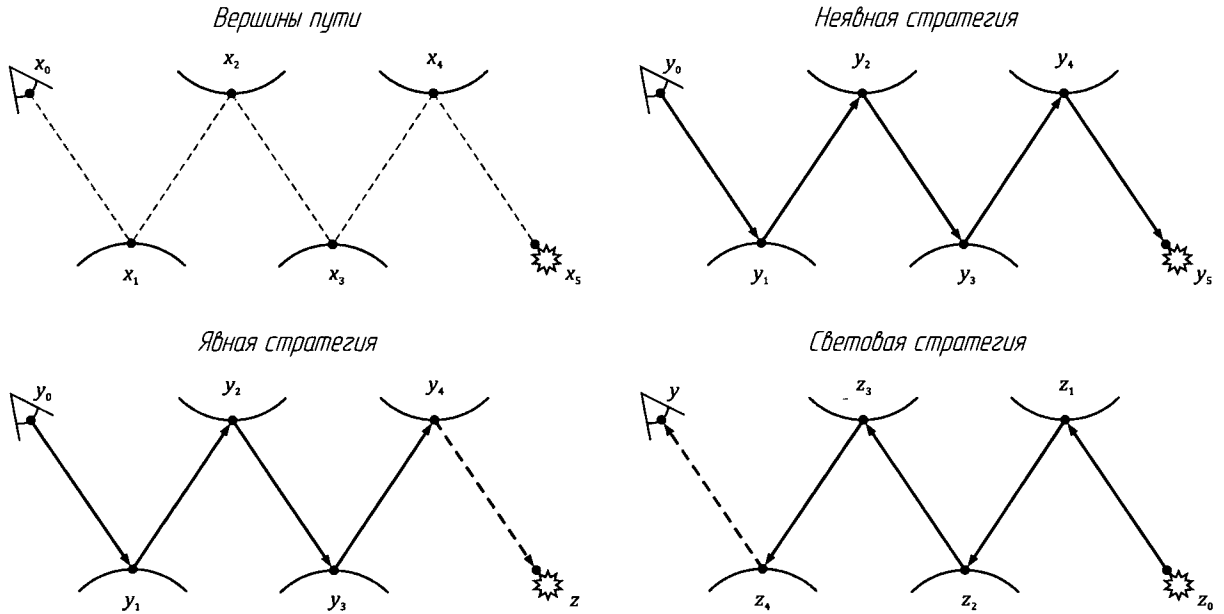


Рис. 2.13. Три стратегии построения корректных путей: неявный и явный (обратная трассировка) и световой (прямая трассировка)

Вклады явного и неявного пути определяются по формулам (2.59)–(2.62). Для расчета вклада светового пути выразим его плотность вероятности p_L :

$$p_L(\mathbf{x}_0 \dots \mathbf{x}_k) = p_A(\mathbf{x}_k) \prod_{i=1}^{k-1} G(\mathbf{x}_i \leftrightarrow \mathbf{x}_{i+1}) \prod_{i=2}^k p_{\omega^\perp}(\mathbf{x}_i \rightarrow \mathbf{x}_{i-1}) p_l(\mathbf{x}_0) \quad (2.70)$$

Поскольку вершина \mathbf{x}_1 генерируется как часть светового пути, соответствующая плотность вероятности не требует преобразования по формуле (2.47). Вклад светового пути по методу Монте-Карло запишется следующим образом:

$$\begin{aligned} \frac{f_j(\mathbf{x}_0 \dots \mathbf{x}_k)}{p_L(\mathbf{x}_0 \dots \mathbf{x}_k)} &= \widehat{W}_j(\mathbf{x}_1 \rightarrow \mathbf{x}_0) \frac{l^2(N_1 \cdot (\mathbf{x}_0 - \mathbf{x}_1))}{(N_0 \cdot (\mathbf{x}_1 - \mathbf{x}_0))^3} p_V(\mathbf{x}_1) \\ &\times f_r(\mathbf{x}_2 \rightarrow \mathbf{x}_1 \rightarrow \mathbf{x}_0) \prod_{i=2}^{k-1} \frac{f_r(\mathbf{x}_{i+1} \rightarrow \mathbf{x}_i \rightarrow \mathbf{x}_{i-1})}{p_{\omega^\perp}(\mathbf{x}_i \rightarrow \mathbf{x}_{i-1})} \frac{L_e(\mathbf{x}_k \rightarrow \mathbf{x}_{k-1})}{p_A(\mathbf{x}_k) p_{\omega^\perp}(\mathbf{x}_k \rightarrow \mathbf{x}_{k-1})} \end{aligned} \quad (2.71)$$

Для унификации записи здесь применяется модифицированная функция чувствительности \widehat{W}_j , которая позволяет включать в предыдущие формулы плотность вершины \mathbf{x}_1 , отнесенную к единичной площади экранной плоскости. В данном случае указанное преобразование не требуется, поэтому возникает дополнительный коэффициент перехода.

В настоящей работе предлагается оригинальный метод вычисления *обратных* весов путей на основе многократной выборки по значимости. Для *явных* путей переноса излучения энергетическая эвристика (2.58) имеет вид:

$$\frac{1}{w_E(\mathbf{x}_0 \dots \mathbf{x}_k)} = 1 + \frac{p_I(\mathbf{x}_0 \dots \mathbf{x}_k)^\beta}{p_E(\mathbf{x}_0 \dots \mathbf{x}_k)^\beta} + \frac{p_L(\mathbf{x}_0 \dots \mathbf{x}_k)^\beta}{p_E(\mathbf{x}_0 \dots \mathbf{x}_k)^\beta} \quad (2.72)$$

Поскольку явный и неявный пути различаются только способом генерации последней вершины \mathbf{x}_k , отношение их плотностей вероятности определяется по формуле:

$$\frac{p_I(\mathbf{x}_0 \dots \mathbf{x}_k)}{p_E(\mathbf{x}_0 \dots \mathbf{x}_k)} = \frac{p_{\omega^\perp}(\mathbf{x}_{k-1} \rightarrow \mathbf{x}_k) G(\mathbf{x}_{k-1} \leftrightarrow \mathbf{x}_k)}{p_A(\mathbf{x}_k)} \quad (2.73)$$

Для светового и явного пути отношение плотностей вероятности может быть получено из формул (2.60) и (2.70):

$$\begin{aligned} \frac{p_L(\mathbf{x}_0 \dots \mathbf{x}_k)}{p_E(\mathbf{x}_0 \dots \mathbf{x}_k)} &= \frac{p_A(\mathbf{x}_k) \prod_{i=1}^{k-1} G(\mathbf{x}_i \leftrightarrow \mathbf{x}_{i+1}) \prod_{i=2}^k p_{\omega^\perp}(\mathbf{x}_i \rightarrow \mathbf{x}_{i-1}) p_I(\mathbf{x}_0)}{p_I(\mathbf{x}_0) p_A(\mathbf{x}_1) \prod_{i=1}^{k-2} G(\mathbf{x}_i \leftrightarrow \mathbf{x}_{i+1}) \prod_{i=1}^{k-2} p_{\omega^\perp}(\mathbf{x}_i \rightarrow \mathbf{x}_{i+1}) p_A(\mathbf{x}_k)} \\ &= \frac{G(\mathbf{x}_k \leftrightarrow \mathbf{x}_{k-1}) p_{\omega^\perp}(\mathbf{x}_k \rightarrow \mathbf{x}_{k-1}) p_{\omega^\perp}(\mathbf{x}_{k-1} \rightarrow \mathbf{x}_{k-2})}{p_A(\mathbf{x}_1) p_{\omega^\perp}(\mathbf{x}_1 \rightarrow \mathbf{x}_2)} \prod_{i=2}^{k-2} \frac{p_{\omega^\perp}(\mathbf{x}_i \rightarrow \mathbf{x}_{i-1})}{p_{\omega^\perp}(\mathbf{x}_i \rightarrow \mathbf{x}_{i+1})} \end{aligned} \quad (2.74)$$

Данное выражение допускает эффективную *итеративную* процедуру вычисления. Для этого его необходимо переписать в другой форме, разделив на отдельные компоненты с помощью следующих обозначений:

$$\begin{aligned} P_0^E &= 1/p_A(\mathbf{x}_1) \\ P_1^E &= 1/p_{\omega^\perp}(\mathbf{x}_1 \rightarrow \mathbf{x}_2) \\ P_i^E &= \prod_{i=2}^{k-2} \frac{p_{\omega^\perp}(\mathbf{x}_i \rightarrow \mathbf{x}_{i-1})}{p_{\omega^\perp}(\mathbf{x}_i \rightarrow \mathbf{x}_{i+1})} \\ P_{k-1}^E &= p_{\omega^\perp}(\mathbf{x}_{k-1} \rightarrow \mathbf{x}_{k-2}) \\ P_k^E &= p_{\omega^\perp}(\mathbf{x}_k \rightarrow \mathbf{x}_{k-1}) G(\mathbf{x}_k \leftrightarrow \mathbf{x}_{k-1}) \end{aligned} \quad (2.75)$$

В данных формулах величину P_i^E можно вычислить сразу после выборки $(i + 1)$ -ой вершины пути. С использованием данных обозначений отношение (2.74) принимает вид:

$$\frac{p_L(\mathbf{x}_0 \dots \mathbf{x}_k)}{p_E(\mathbf{x}_0 \dots \mathbf{x}_k)} = \prod_{i=0}^k P_i^E \quad (2.76)$$

Произведение в правой части данного выражения *накапливается* в процессе генерации пути. Таким образом, для вычисления отношения (2.74) в произвольной вершине пути необходимо поддерживать всего *одну* скалярную переменную на путь. В отношении затрат памяти такой алгоритм практически не отличается от обычной (однонаправленной) трассировки путей.

Аналогично определяется обратный вес *неявного* пути переноса света (для сокращения записи опущены аргументы функций):

$$\frac{1}{w_l} = 1 + \frac{p_E^\beta}{p_l^\beta} + \frac{p_L^\beta}{p_l^\beta} = 1 + \frac{p_E^\beta}{p_l^\beta} \left(1 + \frac{p_l^\beta p_L^\beta}{p_E^\beta p_l^\beta} \right) = 1 + \frac{p_E^\beta}{p_l^\beta} \left(1 + \frac{p_L^\beta}{p_E^\beta} \right) \quad (2.77)$$

Таким образом, вычисление веса неявного пути также сводится к формуле (2.76). Наконец, в процессе прямой трассировки (от источника света) обратный вес *светового* пути может быть вычислен следующим образом:

$$\frac{1}{w_l} = 1 + \frac{p_l^\beta}{p_L^\beta} + \frac{p_E^\beta}{p_L^\beta} = 1 + \frac{p_E^\beta}{p_L^\beta} \left(1 + \frac{p_L^\beta p_l^\beta}{p_E^\beta p_L^\beta} \right) = 1 + \frac{p_E^\beta}{p_L^\beta} \left(1 + \frac{p_l^\beta}{p_E^\beta} \right) \quad (2.78)$$

Следовательно, для расчета веса светового пути алгоритм в ходе прямой трассировки должен накапливать величины $P_i^L = 1/P_i^E$ в *обратном* порядке:

$$\begin{aligned} P_0^L &= \frac{1}{p_{\omega^\perp}(\mathbf{x}_k \rightarrow \mathbf{x}_{k-1})G(\mathbf{x}_k \leftrightarrow \mathbf{x}_{k-1})} \\ P_1^L &= \frac{1}{p_{\omega^\perp}(\mathbf{x}_{k-1} \rightarrow \mathbf{x}_{k-2})} \\ P_i^L &= \prod_{i=2}^{k-2} \frac{p_{\omega^\perp}(\mathbf{x}_i \rightarrow \mathbf{x}_{i+1})}{p_{\omega^\perp}(\mathbf{x}_i \rightarrow \mathbf{x}_{i-1})} \\ P_{k-1}^L &= p_{\omega^\perp}(\mathbf{x}_1 \rightarrow \mathbf{x}_2) \\ P_k^L &= p_A(\mathbf{x}_1) \end{aligned} \quad (2.79)$$

В результате весовые функции всех стратегий переноса излучения эффективно вычисляются за счет поддержки единственной скалярной переменной на путь (независимо от его длины).

Подробное исследование сходимости алгоритма «усеченной» двунаправленной трассировки путей дано в 4-ой главе.

2.9. Выводы к главе 2

1. На основе энергетического подхода построена система моделей, которая обеспечивает физически обоснованный синтез изображений компьютерных сцен. Система моделей поддерживает алгоритмизацию и параметризацию целого класса методов глобального освещения. В настоящем исследовании данная система использована для построения двух основных методов глобального освещения: стохастической трассировки путей и двунаправленной трассировки путей.
2. Построенная система моделей способна к развитию и встраиванию новых методов глобального освещения: на ее основе предложен новый метод глобального освещения, который является усеченным вариантом метода двунаправленной трассировки путей и отличается следующими особенностями:
 - На стадиях «прямой» и «обратной» трассировки для обработки путей любой длины требует фиксированного объема памяти (стандартный алгоритм двунаправленной трассировки сохраняет все вершины путей). Это позволяет эффективно загрузить графический процессор большим числом «легковесных» путей и снять возможные ограничения на длину путей при синтезе несмещенных изображений.
 - Трассировка путей в каждом направлении выполняется независимо, что позволяет выполнить «прямой» и «обратный» этап визуализации на различных графических процессорах (для получения результата два изображения необходимо сложить).
 - Полностью исключается ресурсоемкая фаза соединения «прямого» и «обратного» путей (требует информации обо всех вершинах каждого пути).
 - Упрощается многократная выборка по значимости, для реализации которой была предложена оптимизированная схема, позволяющая хранить всего одну скалярную переменную на путь (независимо от его длины).

В результате подготовлено построение универсального графического конвейера трассировки лучей, который может быть настроен для использования алгоритмов с различным балансом скорости и качества синтезируемых изображений при сохранении физической корректности моделей.

Глава 3

Универсальный конвейер трассировки лучей для интерактивного синтеза изображений методами глобального освещения на графических процессорах

Целью данной главы исследования является построение основ создания программных систем виртуальной реальности, обеспечивающих интерактивный синтез фотореалистичных изображений методами глобального освещения на современных графических процессорах. В ходе достижения этой цели должен быть решен целый ряд проблем, которые связаны как с содержанием задачи, так и с особенностями графических процессоров:

- Разработать метод построения ускоряющей структуры на графическом процессоре (и других массивно-параллельных вычислительных архитектурах), который обеспечивает высокое быстродействие трассировки лучей для сложных динамических сцен.
- Разработать метод представления иерархической объектно-ориентированной модели сцены, позволяющий эффективно сконвертировать (сериализовать) сцену в потоковое представление для передачи и обработки на графическом процессоре.
- Разработать метод представления геометрических объектов сцены, который допускает сочетание в одной модели полигональных и нетесселированных данных (поверхности второго порядка, неявно заданные поверхности, фрактальные множества и фрагменты сплошных сред).
- Разработать универсальный конвейер трассировки лучей для интерактивного синтеза изображений на основе методов глобального освещения, который отличается полнотой реализации на массивно-параллельных графических архитектурах. Следует обеспечить возможность конфигурирования конвейера для применения алгоритмов с различным балансом скорости и качества визуализации при сохранении физической корректности моделей.
- Построить расширяемые подсистемы материалов (BSDF) и источников света, которые могут быть дополнены необходимыми моделями рассеивающих свойств поверхностей и излучающих свойств источников света.

3.1. Построение на графическом процессоре эффективной ускоряющей структуры для лучевых методов синтеза изображений

3.1.1. Выбор эффективной ускоряющей структуры для динамических сцен в условиях ограничения памяти

В основе высокопроизводительной реализации трассировки лучей лежит эффективная ускоряющая структура, которая позволяет оптимизировать поиск точки соударения луча с ближайшим видимым объектом сцены. На протяжении последних трех десятилетий было предложено большое разнообразие структур данных, которые реализуют различные подходы к пространственной сортировке объектов сцены [15; 95; 132; 133]. С точки зрения базового подхода к сортировке все ускоряющие структуры можно разделить на два класса.

- *Разбиения пространства.* Позволяют уникальным образом представить каждую точку пространства, однако каждый примитив сцены может перекрываться любым числом ячеек. К разбиениям пространства относятся регулярные сетки (одноуровневые или иерархические), октодеревья, k - d деревья и ряд других типов двоичных разбиений пространства.
- *Иерархии объектов.* Позволяют уникальным образом представить каждый примитив сцены, однако каждая точка пространства может перекрываться любым числом ячеек. К иерархиям объектов относятся такие ускоряющие структуры данных, как иерархии ограничивающих объемов и их разнообразные варианты (иерархии интервалов, s - kd деревья, b - kd деревья, DE-деревья, H-деревья, SBVH-деревья).

Достоинства и недостатки различных структур данных во многом вытекают из особенностей подхода к пространственной сортировке объектов:

- *Поиск точки соударения.* Данная операция является базовой для лучевых алгоритмов синтеза изображений. В разбиениях пространства каждая часть пространства сцены представлена единственным образом, поэтому алгоритм поиска пересечения может обходить ячейки в строгом порядке их следования вдоль луча и совершать «ранний выход» (англ. “early exit”) сразу после обнаружения точки соударения. В иерархиях объектов узлы дерева могут произвольно перекрываться, поэтому точка соударения в одном поддереве может впоследствии замещаться точкой из другого поддерева, что потенциально ведет к дополнительным вычислениям. С другой стороны, разбиения пространства допускают многократную обработку одних и тех же примитивов сцены,

что исключается в случае иерархии объектов. Аналогичное замечание справедливо и в отношении пустых узлов, которые в иерархии объектов отсутствуют.

- *Плотность аппроксимации геометрии.* Как правило, разбиения пространства позволяют получить более мелкое разбиение, которое хорошо ограничивает геометрию сцены. За счет этого максимально сокращается число тестов пересечения, однако повышается трудоемкость генерации структуры и возрастает число операций обхода (внутренних узлов дерева). С другой стороны, иерархии объектов эффективнее аппроксимируют геометрию сцены на каждом уровне дерева (за счет использования большего числа плоскостей разбиения), однако минимальный размер ячейки определяется размером конкретного примитива (за исключением некоторых гибридных структур). В итоге сокращается трудоемкость построения структуры данных и число операций обхода, однако повышается число тестов пересечения.
- *Потребляемая память.* Другое важное замечание относится к объему потребляемой памяти. Поскольку иерархии объектов содержат значительно меньше узлов и ссылок на примитивы сцены, данные структуры довольно компактны. Для хранения типовой иерархии (выровненных) параллелепипедов требуется в 3–4 раза меньше памяти по сравнению с k - d деревом. Данный показатель имеет решающее значение при выборе структуры для графических процессоров, которые оснащаются относительно малым объемом памяти (1–4 Гб). Кроме того, поскольку каждый примитив сцены входит в иерархию объектов ровно один раз, общее число узлов дерева ограничено величиной $2N - 1$ (N – число примитивов сцены). В результате построение иерархии объектов может быть выполнено непосредственно на массиве примитивов (англ. in-place) без выделения дополнительной памяти для разбиения вокселей. Данное обстоятельство позволяет разрабатывать эффективные алгоритмы построения иерархии объектов на графическом процессоре.
- *Обновление структуры.* По сравнению с разбиениями пространства иерархии объектов более устойчивы к малым перемещениям геометрии сцены. Благодаря этому вместо полного перестроения структуры можно обновить только ограничивающие объемы узлов, сохранив топологию дерева неизменной. Данное обстоятельство может быть полезно при обработке деформируемых сцен. Наконец, естественное представление объектов сцены ограничивающими объемами позволяет строить иерархию объектов на основе графа сцены (при его наличии). В ряде случаев данный подход позволяет повысить качество и скорость построения ускоряющей структуры.

В данном исследовании ставилась задача разработки компактной и высокопроизводительной структуры данных, которая допускает эффективный параллельный алгоритм построения на графическом процессоре. В силу перечисленных обстоятельств естественным выбором стала иерархия ограничивающих объемов (выровненных параллелепипедов), которая обеспечивает удачное сочетание выбранных критериев.

3.1.2. Применение эвристики площадей поверхностей для построения иерархических ускоряющих структур

Задача построения *эффективных* иерархических структур (обеспечивающих высокую производительность трассировки лучей) является классической проблемой в компьютерной графике [95; 134; 135; 136]. На данный момент наилучшим известным критерием качества структуры является эвристика площадей поверхностей (англ. Surface Area Heuristic, SAH). Данная эвристика описывает *ожидаемую* стоимость обхода вокселя V , который разбивается секущей плоскостью p на два дочерних вокселя V_L и V_R . Вывод данной эвристики основан на следующих допущениях:

1. Лучи являются прямыми бесконечными линиями, которые равномерно распределены в пространстве и не могут блокироваться внутри вокселя V .
2. Известны вычислительные стоимости одного шага обхода K_T и теста пересечения с примитивом K_I (обычно используются треугольники).
3. Суммарная стоимость тестов пересечения для N примитивов равна NK_I (такая оценка может оказаться заметно завышенной на SIMD-архитектурах).

Данные предположения позволяют формально определить стоимость разбиения вокселя V секущей плоскостью p . В случае равномерно распределенных линий и выпуклых вокселей геометрическая вероятность пересечения лучом подвокселя $V' \in V$ при условии пересечения вокселя V выражается формулой:

$$P(V'|V) = \frac{S(V')}{S(V)} \quad (3.1)$$

Символом $S(V)$ здесь обозначена площадь поверхности вокселя V . Ожидаемая стоимость обхода вокселя V при разбиении секущей плоскостью p складывается из стоимости одного шага обхода и ожидаемой стоимости пересечения дочерних вокселей:

$$C_V(p) = K_T + P(V_L|V)C(V_L) + P(V_R|V)C(V_R) \quad (3.2)$$

Рекурсивно разворачивая данное выражение, получим ожидаемую стоимость обхода всего дерева T :

$$C(T) = \sum_{n \in Nodes} \frac{S(V_n)}{S(V_S)} K_T + \sum_{l \in Leaves} \frac{S(V_l)}{S(V_S)} K_I \quad (3.3)$$

Символом V_S здесь обозначен ограничивающий параллелепипед для всех примитивов сцены S . Максимальную производительность для трассировки лучей должно обеспечивать дерево, которое доставляет минимум данной функции. Поскольку число возможных деревьев растет экспоненциально в зависимости от числа примитивов, на практике построить оптимальное дерево невозможно (за исключением тривиальных случаев).

Вместо поиска глобального минимума используются жадные алгоритмы построения, которые производят локально оптимальные разбиения вокселей в предположении, что оба дочерних потомка являются листьями:

$$\begin{aligned} C_V(p) &\approx K_T + P(V_L|V)|T_L|K_I + P(V_R|V)|T_R|K_I \\ &= K_T + K_I \left(\frac{S(V_L)}{S(V)} |T_L| + \frac{S(V_R)}{S(V)} |T_R| \right) \end{aligned} \quad (3.4)$$

В данном выражении символами $|T_L|$ и $|T_R|$ обозначено число примитивов в левом и правом дочернем узле соответственно. Данная аппроксимация является довольно грубой и сильно переоценивает фактическую стоимость дочерних узлов, которые в действительности почти всегда оказываются поддеревьями (а не листовыми вершинами). Однако на практике такой подход дает хорошие результаты и является предпочтительным по сравнению с некоторыми более сложными эвристиками.

Эвристика площадей позволяет получить простой и эффективный критерий остановки разбиения. Если стоимость наилучшей секущей плоскости p оказывается выше стоимости обработки листовой вершины, то процесс разбиения останавливается:

$$\min_p C_V(p) > K_I T \quad (3.5)$$

При использовании данной эвристики разбиение узла может прекратиться даже тогда, когда *фактическая* стоимость разбиения указывает на необходимость дальнейшей декомпозиции узла. Тем не менее, данный критерий обеспечивает высокую эффективность по сравнению с некоторыми модификациями, что свидетельствует о достаточной степени точности.

3.1.3. Схема алгоритма построения иерархии объемов

В рамках настоящего исследования был разработан высокопроизводительный алгоритм для построения иерархии объемов на графическом процессоре. Данный алгоритм развивает подходы [71; 102; 137], однако не задействует простые эвристики, подобные медианному разбиению объектов (на которой базируется «линейный» алгоритм LBVN). Вместо этого был предложен набор методик, которые позволяют эффективно отобразить исходный алгоритм построения SAH дерева на массивно-параллельные графические процессоры. Разработанный алгоритм позволил получить 5-кратный прирост производительности по сравнению с лучшей известной реализацией [102] (на момент публикации результатов).

«Ячеечный» (англ. binned) алгоритм построения SAH иерархии объемов можно разбить на отдельные задания, последовательное исполнение которых реализуется через концепцию очереди. Построение дерева описывается последовательностью шагов:

1. В очередь заданий добавляется корневой узел иерархии объемов, который совпадает с (выровненным) ограничивающим параллелепипедом сцены и содержит полный набор геометрических примитивов.
2. Первый узел очереди помечается как активный.
3. Активный узел разбивается на K равных ячеек вдоль осей x , y и z . Для каждой ячейки подсчитывается число геометрических примитивов и вычисляется выпуклая оболочка (выровненный ограничивающий параллелепипед).
4. На внутренних границах ячеек вычисляется стоимость разбиения по SAH. Плоскость с минимальной стоимостью выбирается в качестве секущей.
5. Активный узел разбивается на два дочерних узла, которые содержат геометрические примитивы слева и справа от секущей плоскости соответственно.
6. Для каждого нового узла число геометрических примитивов сравнивается с заданным пороговым значением T . Если число примитивов превышает заданный порог, то узел помещается в очередь заданий для дальнейшего разбиения.
7. Активный узел удаляется из очереди заданий. Если очередь содержит невыполненные задания, то происходит переход к шагу 2.

Основным параметром данного алгоритма является число ячеек K по каждой оси вокселя. В работе [71] сообщается, что уже 4 «корзины» обеспечивали достаточно хорошие результаты. Дальнейшее наращивание числа ячеек позволило повысить качество структуры, однако при

использовании более 16 «корзин» дополнительный прирост эффективности не фиксировался. Таким образом, 16 ячеек (по каждой стороне вокселя) обеспечивают оптимальный результат, соответствующий «полноценному» алгоритму на основе подвижной плоскости [60].

В настоящей работе для получения качественной ускоряющей структуры используется $K = 16$. Максимальное число примитивов на листовую вершину определяется эмпирически. На практике значение $T = 4$ обеспечивает хорошие результаты.

3.1.4. Адаптация алгоритма построения для графического процессора

Эффективное отображение базового алгоритма построения на графический процессор предполагает декомпозицию задачи на «порции» определенного размера, которые позволяют максимально задействовать аппаратный параллелизм. Кроме того, для достижения высокой производительности необходимо использовать оптимизированные схемы работы с памятью.

Наиболее простой способ реализации алгоритма был описан в работе [102]. В рамках данного подхода одно задание (из очереди) отображается на одну группу работ (в терминах стандарта OpenCL). Такая схема приводит к низкой производительности на верхних уровнях дерева (поскольку в вычислениях задействуется только часть доступных ядер) и к падению производительности на нижних уровнях (за счет роста накладных расходов на поддержку большого числа мелких заданий).

Следующий относительно простой подход был предложен в работе [137] и реализован на архитектуре Intel MIC¹: для больших узлов (число примитивов больше заданного порога) используются ресурсы всего процессора, в то время как остальные узлы обрабатываются по предыдущей схеме – одна вершина на группу работ. Данный подход оказался эффективным для архитектуры Intel MIC, но при отображении на архитектуру графического процессора выявляется ряд недостатков. По аналогии с предыдущей работой, здесь возникает падение производительности на нижних уровнях. Кроме того, появляется «промежуточный слой» – часть уровней, для которых одно задание не способно загрузить весь графический процессор, однако число заданий мало для отображения на отдельные группы работ.

В данном исследовании предложен альтернативный подход к отображению алгоритма построения на графический процессор, который снимает указанные проблемы. Общая схема построения состоит в следующем:

- На каждом шаге построения дерева выполняются все доступные задания независимо от их размера. В результате достигается *полная* утилизация ресурсов графического

¹ Intel MIC (Intel Many Integrated Core Architecture) – архитектура многоядерной процессорной системы, которая вводит применение очень широких векторных АЛУ (512-разрядные SIMD) в микропроцессоры с архитектурой x86.

процессора, а также снижаются накладные расходы, вызванные передачей заданий с центрального процессора.

- Для обработки узлов с большим числом геометрических примитивов (использовался порог $N \geq 32K$) применяется несколько групп работ на узел (в текущей реализации – $N/512$). В сочетании с оптимизацией из предыдущего пункта такой подход позволяет обеспечить *оптимальное* использование аппаратных ресурсов.
- Построение дерева разбивается на три стадии: обработка больших узлов (более 32K примитивов), средних узлов (от 256 до 32K примитивов) и малых узлов (менее 256 примитивов). Такой подход позволяет применять специализированные модификации алгоритма на каждом уровне дерева.

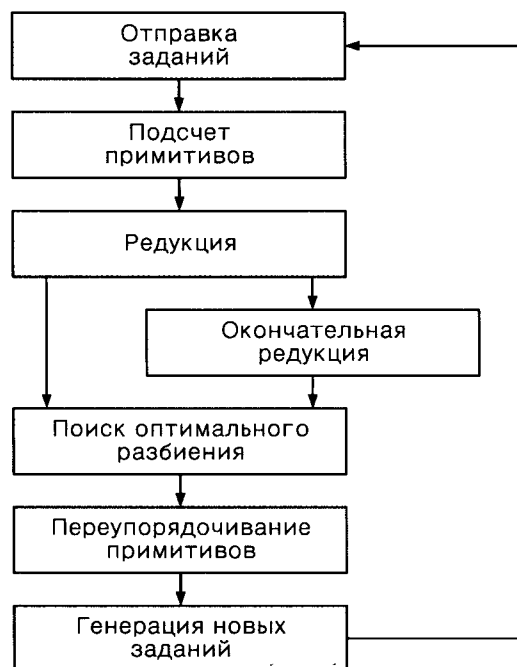


Рис. 3.1. Общая схема построения уровня дерева

На рис. 3.1 изображена наиболее общая схема построения уровня иерархии (используется на стадии обработки больших вершин). Из представленных этапов на центральном процессоре выполняется только отправка заданий и генерация новых заданий. Данные этапы обладают наименьшей трудоемкостью, поэтому их реализация на графическом процессоре не является оправданной.

3.1.5. Эффективная реализация этапов алгоритма построения

Генерация заданий по разбиению узлов

В предлагаемой реализации для обработки одного узла дерева может использоваться несколько групп работ (при этом параллельно обрабатывается несколько узлов). При таком подходе достигается высокая загрузка графического процессора, однако возникает проблема определения «объема работы» (диапазона обрабатываемых данных) для каждой конкретной группы работ. Проблема решается за счет передачи двух дополнительных массивов величин, которые хранят для каждой группы номер обрабатываемого узла и номер данной группы в узле. Эта информация в сочетании с данными об узлах (число примитивов, номер первого примитива, ограничивающий параллелепипед узла) позволяет однозначно определить объем работы для каждой группы работ. Интерпретация указанных параметров (на графическом процессоре) может быть описана следующим псевдокодом²:

```
void DefineWork( out int readOffset, out int workSize, out vec3 boxMin, out vec3 boxMax )
{
    nodeIndex = nodeIndices[groupID]
    nodeGroup = groupID - nodeOffsets[nodeIndex]

    readOffset = readOffsets[nodeIndex] + nodeGroup * nodeWorkSize

    groupWorkSize = min( nodeWorkSize,
                        nodeSizes[nodeIndex] - nodeGroup * nodeWorkSize )

    boxMin = nodeMinPoints[nodeIndex]
    boxMax = nodeMaxPoints[nodeIndex]
}
```

Подсчет числа примитивов в ячейках вокселя

На втором этапе построения уровня дерева для каждой ячейки (корзины) выполняется подсчет числа примитивов и вычисление ограничивающих параллелепипедов. В настоящей работе разбиение вокселя производится по всем трем координатным осям (часто разбиение выполняется только по наибольшей стороне). Для некоторых сцен данный подход позволяет получить существенный (до 10-ти раз) прирост производительности. Этап подсчета можно условно разделить на две части: получение данных о ячейках узла на основе информации о примитивах и применение к этим данным классического алгоритма параллельной редукции.

² Здесь и далее в псевдокоде принимаются следующие обозначения: *threadID* – глобальный идентификатор потока, *localID* – локальный идентификатор потока (внутри группы работ), *groupID* – идентификатор группы работ.

Реализация первой части аналогична работам [102] и [71]. Примитивы каждой группы работ делятся на части по 16 элементов (в соответствии с числом ячеек), каждая из которых обрабатывается 16-ю последовательными потоками (в терминологии NVIDIA CUDA такой набор потоков называется «полусверткой» – *halfwarp*). Реализация данного алгоритма может быть описана следующим псевдокодом:

```
void UpdateBins( Bin bins[16] )
{
    for( i = 0; i < 16; i++ )
        if( bin of triangle[i] == threadID )
            bins[threadID].append( triangle[i] )
}
```

Использование 16 проходов на 16 элементов может показаться излишним, однако подобная схема подсчета оптимально отображается на SIMD-архитектуру графического процессора (в работе использовались карты NVIDIA) и сводит к минимуму конфликты при обращении к памяти. При альтернативном решении задачи каждый поток вычисляет данные о «корзине» для ассоциированного треугольника, после чего выполняется редукция полученных данных. Такой подход приводит к росту потребляемой памяти (порядка 16-ти раз) и дополнительным вычислительным затратам, связанным с последующей редукцией.

Параллельная редукция

При реализации данного этапа был использован классический алгоритм параллельной редукции [138]. Однако, в отличие от типовой реализации, в данной работе не используется переменное число итераций алгоритма. При обработке «средних» уровней дерева достаточно всего одной итерации. При обработке «больших» уровней применяется дополнительное ядро «окончательной редукции», которое выполняет редукцию всех имеющихся комплектов ячеек в один. Для «малых» уровней данная стадия не требуется вовсе. Потери производительности при таком подходе несущественны, поскольку значительная часть вычислительной нагрузки приходится на этап подсчета. На выходе получается более простой алгоритм, и уменьшается объем передаваемых данных между центральным и графическим процессором. Реализация данного этапа может быть описана следующим псевдокодом (предполагается размер группы работ 256):

```
void Reduce( local Bin bins[] )
{
    if( localID < 128 )
        bins[localID].append( bins[localID + 128] )
    if( localID < 64 )
```

```

    bins[localID].append( bins[localID + 64] )
if( localID < 32 )
    bins[localID].append( bins[localID + 32] )
if( localID < 16 )
    bins[localID].append( bins[localID + 16] )
}

```

Поиск оптимального разбиения

На данном этапе информация о ячейках вокселя используется для выбора наилучшей секущей плоскости (с минимальной SAH-стоимостью разбиения). Данная операция является наименее ресурсоемкой и реализована на графическом процессоре для минимизации объема данных, передаваемых между центральным процессором и ускорителем. Псевдокод данного этапа выглядит следующим образом (предполагается размер группы работ 48):

```

void FindBestSplit( Bin bins[16][3], out Split bestSplit ) // 16 bins × 3 axes
{
    int axis = localID / 16
    int item = localID & 15

    local Split splits[16][3] // split candidates

    if( item > 0 ) splits[localID][axis].left.append( bins[localID - 1][axis] )
    if( item > 1 ) splits[localID][axis].left.append( bins[localID - 2][axis] )
    if( item > 3 ) splits[localID][axis].left.append( bins[localID - 4][axis] )
    if( item > 7 ) splits[localID][axis].left.append( bins[localID - 8][axis] )

    if( item < 15 ) splits[localID][axis].right.append( bins[localID + 1][axis] )
    if( item < 14 ) splits[localID][axis].right.append( bins[localID + 2][axis] )
    if( item < 12 ) splits[localID][axis].right.append( bins[localID + 4][axis] )
    if( item < 8 ) splits[localID][axis].right.append( bins[localID + 8][axis] )

    splits[localID][axis].computeSAH()

    if( item < 14 and splits[localID + 1][axis].cost < splits[localID][axis].cost )
        splits[localID][axis] = splits[localID + 1][axis]
    if( item < 13 and splits[localID + 2][axis].cost < splits[localID][axis].cost )
        splits[localID][axis] = splits[localID + 2][axis]
    if( item < 11 and splits[localID + 4][axis].cost < splits[localID][axis].cost )
        splits[localID][axis] = splits[localID + 4][axis]
    if( item < 7 and splits[localID + 8][axis].cost < splits[localID][axis].cost )
        splits[localID][axis] = splits[localID + 8][axis]

    if( localID == 0 )
    {

```

```

bestSplit = splits[0][0] // best X split

if( splits[0][1].cost < bestSplit.cost ) // compare to best Y split
    bestSplit = splits[0][1]
if( splits[0][2].cost < bestSplit.cost ) // compare to best Z split
    bestSplit = splits[0][2]
}
}

```

Переупорядочивание примитивов

Последним ресурсоемким этапом построения является переупорядочивание элементов узлов в соответствии с найденными секущими плоскостями (примитивы слева от плоскости перемещаются в начало массива, примитивы справа – в конец). Основой для решения задачи послужила параллельная реализация поразрядной сортировки [139]. Данный алгоритм можно разделить на две стадии: префиксное суммирование индексов записи для всех примитивов узла (поиск подходящей позиции в массиве) и фактическое переупорядочивание элементов массива. Традиционно указанные стадии выполняются в несколько проходов, что влечет за собой дополнительные затраты памяти (для передачи временных данных между проходами) и лишние вычисления (связанные с запуском префиксного суммирования на «глобальном» уровне [140]).

В настоящей работе вместо «глобального» префиксного суммирования применяются атомарные операции. Таким образом, дополнительные шаги вычисления префиксной суммы на глобальном уровне были заменены одной атомарной операцией на каждую группу работ, за счет чего сокращается количество вычислений и объем потребляемой памяти. Алгоритм переупорядочивания может быть описан следующим псевдокодом.

```

// Initial interval = [0, NodeSize - 1]
void Split( Triangle inTriangles[], Triangle outTriangles[], Interval interval, int splitIndex )
{
    local writeIndices[groupSize]

    if( inTriangles[globalID].splitIndex < splitIndex )
        right = 0
    else
        right = 1

    writeIndices[localID] = right

    scan( writeIndices ) // compute output primitive indices

    local int localStart

```

```

local int localFinal

if( localID == 0 ) {
    localStart = atomicAdd( interval.start,
                           groupSize - writeIndices[localSize - 1] )
    localFinal = atomicSub( interval.final,
                           writeIndices[localSize - 1] )
}

int unitIndex = localFinal - writeIndices[localID]
int zeroIndex = localStart - writeIndices[localID] + localID + right

outTriangles[ right ? unitIndex : zeroIndex ] = inTriangles[ globalID ]
}

```

Генерация новых заданий

На данной стадии построенные узлы добавляются в дерево, и выполняется генерация новых заданий. Если в результате обработки больших и средних узлов число примитивов в одном из дочерних узлов оказывается ниже заданного порога (32К и 256 соответственно), то такой узел перемещается в очередь заданий для следующего этапа. Если после обработки малых узлов число примитивов в дочернем узле не превышает заданный порог (применялось значение 4), то такие узлы помечаются как листовые вершины дерева. Данный этап является наименее ресурсоемкой частью алгоритма и описывается следующим псевдокодом:

```

void GenerateTasks( Node node, Child lChild, Child rChild )
{
    node.setChildren( lChild, rChild )

    lChild.setParent( node )
    rChild.setParent( node )

    nodes.add( left, right )

    if( lChild.size ≥ minSize )
        currTasks.add( lChild )
    else
        nextTasks.add( lChild )

    if( rChild.size ≥ minSize )
        currTasks.add( rChild )
    else
        nextTasks.add( rChild )
}

```

Полный исходный код (на базе OpenCL) алгоритма построения иерархии объемов доступен по адресу: <https://code.google.com/p/rtrt>.

3.1.6. Алгоритмы обхода иерархии объемов со стеком и без стека для использования на графическом процессоре

Изначально для обхода иерархии объемов на графическом процессоре был предложен «бесстековый» алгоритм [87], который активно использовался на ранней программируемой графической аппаратуре. Традиционные инструменты программирования (GLSL и HLSL) не поддерживали запись в память по произвольному адресу (необходима для реализации стека). Хранение стека в регистрах слишком накладно и применимо только для k -d деревьев (обход которых можно реализовать на базе «короткого» стека [90]). Наконец, реализация «полного» стека на основе многопроходной техники имела крайне низкую производительность [85]. В такой ситуации «бесстековый» алгоритм оказался оптимальным вариантом. Традиционный вариант обхода дерева с использованием указателей перехода (англ. *escape indices*) можно описать следующим псевдокодом (рис. 1.14):

```
void Traverse( Ray ray, out Intersection hit )
{
    int index = 0
    while( index < number of nodes ) {
        Node node = nodes[index]

        if( ray intersects node ) {
            index = index + 1
            if( node is leaf ) {
                Intersect ray with node geometry and update hit
            }
        }
        else
            index = escapeIndex( node )
    }
}
```

Данный алгоритм предполагает фиксированную схему размещения иерархии в массиве, при которой первый дочерний узел всегда следует за родительским узлом. Тогда относительные указатели перехода к первому потомку и соседнему листу всегда равны 1.

На новом поколении графической аппаратуры стали доступны универсальные средства программирования – NVIDIA CUDA и OpenCL. Данные инструменты позволяют построить эффективную реализацию стека, поскольку глобальная память устройства доступна ядрам на чтение и запись. Наиболее производительная реализация обхода иерархии объемов (на базе

стека) описана в работе [96]. За счет глубокой программной оптимизации для архитектуры NVIDIA CUDA скорость трассировки достигает ~200М некогерентных лучей в секунду. На псевдокоде алгоритм обхода со стеком выглядит следующим образом:

```
void Traverse( Ray ray, out Intersection hit )
{
    Node node = root
    while( true ) {
        if( node is leaf ) {
            Intersect ray with node geometry and update hit
            if( stack is empty )
                break
            node = stack.pop()
        }
        else {
            intersect( ray, node.L, out timeLin, out timeLout )
            intersect( ray, node.R, out timeRin, out timeRout )

            bool L = ( timeLin < timeLout ) and ( timeLin < hit.time ) and ( timeLout ≥ 0 )
            bool R = ( timeRin < timeRout ) and ( timeRin < hit.time ) and ( timeRout ≥ 0 )

            if( L and R ) {
                if( timeLin < timeRin ) {
                    stack.push( node.R )
                    node = node.L
                }
                else {
                    stack.push( node.L )
                    node = node.R
                }
            }
            else if( L )
                node = node.L
            else if( R )
                node = node.R
            else {
                if( stack is empty )
                    break
                node = stack.pop()
            }
        }
    }
}
```

В настоящем исследовании были реализованы оба варианта обхода, однако базовым является алгоритм на основе стека, оптимизированный согласно рекомендациям работы [96].

3.2. Метод представления и сериализации иерархической объектно-ориентированной модели сложной сцены

Для генерации изображения необходимо подготовить детальное *описание* трехмерной компьютерной сцены, которое включает в себя расположение геометрических примитивов, рассеивающие свойства поверхностей, положение и излучающие свойства источников света. Форма представления данной информации определяет эффективность системы визуализации и является ключевой проблемой при проектировании программной архитектуры. Замечание особенно актуально при разработке систем, которые ориентированы на обработку *больших* объемов *разнородных* данных с использованием графического процессора. В традиционных инструментах объектно-ориентированного программирования разработчик может построить структуру произвольной сложности, которая естественным образом описывает предметную область и допускает эффективную обработку. Современные инструменты программирования для графического процессора (CUDA, OpenCL) используют потоковую модель вычислений, в которой данные представлены большими массивами. Таким образом, возникает проблема разработки универсальной формы хранения и обработки компьютерной сцены, которая на «внешнем» уровне представлена объектно-ориентированной структурой, а на «внутреннем» уровне обеспечивает эффективную сериализацию данных в потоки.

Внешний уровень. Внешним уровнем системы следует считать интерфейс прикладного программиста (API), который обеспечивает необходимые функциональные возможности для работы с компьютерной сценой. Данный интерфейс позволяет добавлять и удалять объекты сцены, редактировать различные параметры (например, излучающие свойства источников и рассеивающие свойства поверхностей) и настраивать отношения между объектами. На базе «внешнего» интерфейса разрабатываются такие прикладные программы, как редактор сцены, тестовые приложения, автономный визуализатор и «плагины» для систем трехмерного моделирования и анимации. Данный интерфейс должен в максимальной степени отражать представления пользователей о структуре сцены и способствовать сокращению трудозатрат на разработку перечисленных прикладных программ.

Внутренний уровень. Внутренним уровнем системы следует считать интерфейс (API) для разработки лучевых алгоритмов визуализации. Функциональность данного интерфейса должна охватывать базовые операции трассировки лучей, к числу которых относится поиск (ближайших) точек пересечения с геометрическими примитивами, вычисление нормалей и касательного пространства, вычисление BSDF поверхностей, генерация лучей (первичных,

теневого и вторичных) и ряд других вспомогательных операций. На базе данного интерфейса строится реализация таких методов визуализации, как испускание лучей, трассировка лучей Уиттеда, стохастическая трассировка путей и двунаправленная трассировка путей (включая предложенный «усеченный» вариант). Реализация интерфейса «внутреннего» уровня должна обеспечивать эффективный доступ к данным компьютерной сцены и экономно расходовать видеопамять.

3.2.1. Анализ структур хранения сцены в виде графа и набора массивов

Граф сцены. В настоящее время для описания компьютерных сцен широко применяется иерархический объектно-ориентированный подход, в рамках которого сцена представляется в виде графа – ориентированного дерева. Обычно узлы графа служат только для организации объектов сцены в иерархию и выступают в качестве универсальных контейнеров данных. Для хранения конкретных сущностей используются *ядра*, которые могут включаться в один или несколько узлов. Как правило, ядра задают аффинные преобразования, геометрические объекты (такие как полигональные сетки), визуальные свойства материалов, источники света и объекты других типов (рис. 3.2).

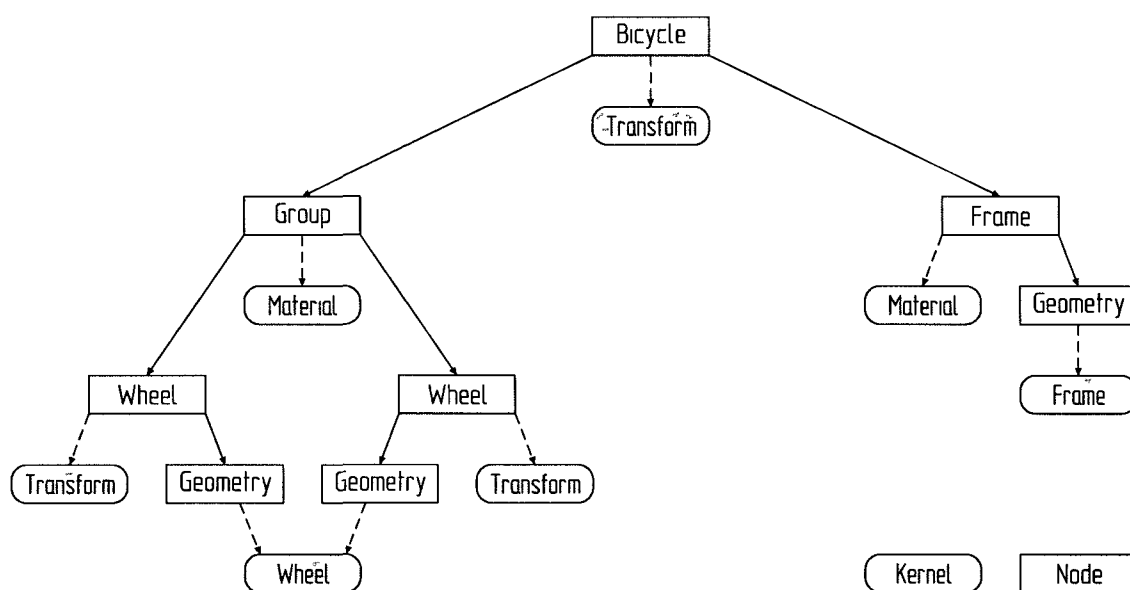


Рис. 3.2. Пример графа сцены для модели велосипеда

Важной возможностью графа сцены является *группирование* объектов, которое позволяет строить узлы с несколькими потомками (при этом каждый потомок может иметь несколько родительских узлов). Если подмножество объектов (например, жестко скрепленные детали) обладает одинаковыми свойствами, группирование позволяет установить данные свойства в

единственном узле. Другое применение группирования связано с ситуацией, когда имеется несколько различающихся по свойствам или расположению вариантов некоторого элемента (например, автомобильное колесо). Тогда геометрия элемента представляется в одном узле графа, но этот узел присоединяется к разным группам.

Использование графа сцены позволяет свести реализацию графических приложений к порождению *прикладной модели*. Данная модель (помимо визуализации сцены) позволяет решать и другие практически полезные задачи. Примером подобной возможности является универсальный редактор, который получает на вход описание сцены и позволяет изменять расположение объектов, визуальные свойства материалов, устанавливать источники света и выполнять другие операции.

Наиболее адекватный стиль для работы с графом сцены – объектно-ориентированный, в рамках которого узлы и ядра графа рассматриваются как объекты языка программирования и обладают совокупностью применимых к ним методов. Типичными представителями таких систем служат проекты с открытым исходным кодом OpenSG [141] и OpenSceneGraph [142], которые применяются для разработки высокопроизводительных 3D приложений. Структура графа сцены широко используется в различных форматах для хранения и обработки 3D сцен, включая язык моделирования виртуальной реальности VRML (англ. Virtual Reality Modeling Language) и формат обмена данными COLLADA (основан на стандарте XML).

Для задач настоящего исследования предпочтительным способом представления сцены также является граф. Однако данная форма описания не допускает эффективного отображения на архитектуру графических процессоров, которые ориентированы на потоковую обработку больших массивов данных.

Набор массивов. Многочисленные исследования в области высокопроизводительной трассировки лучей на графической аппаратуре показали, что по скорости доступа и объему потребляемой памяти оптимальной структурой хранения является набор *массивов*. Данная структура обеспечивает одинаковое время доступа ко всем элементам данных, не требует дополнительных накладных расходов (все элементы находятся в информационном поле) и позволяет эффективно задействовать аппаратный кэш. Использование массивов не вызывает проблемы при хранении одной полигональной модели или списка лучей (типичный подход для многих реализаций). Однако представление большого объема *разнородной* информации сцены (что имеет место в настоящей работе) вызывает определенные сложности, поскольку необходимо поддерживать иерархическую систему массивов и обеспечивать эффективную локализацию данных в такой системе. Хотя структура сцены на базе массивов обеспечивает

высокую производительность визуализации, она практически не пригодна для прикладного программирования, поскольку не позволяет работать с отдельными объектами сцены.

3.2.2. Универсальный механизм хранения и обработки компьютерных моделей: «граф сцены – сериализация»

Перечисленные выше особенности не позволяют остановиться только на одной форме представления сцены. В рамках данного исследования разработан универсальный механизм хранения и обработки компьютерных моделей, который на пользовательском («верхнем») уровне имеет функционал графа сцены, а на нижнем («внутреннем») уровне поддерживает эффективную сериализацию данных в потоки (массивы). Внутреннее представление в виде массивов может быть выгружено для обработки на графический процессор или отправлено по сети на другие вычислительные узлы (при вычислениях на кластерной системе). Данное решение позволяет сочетать лучшие качества обоих способов представления, однако ведет к дополнительным расходам памяти (центрального процессора) для хранения информации в различных формах. Кроме того, для эффективного конвертирования графа сцены в массивы необходимо снабдить все объекты механизмом *сериализации*.

В настоящей работе принята следующая иерархическая модель сцены. Все визуальные объекты представлены *узлами (вершинами)* графа, которые применяются исключительно для построения иерархического описания сцены и выступают в роли универсальных контейнеров данных. Для хранения конкретных сущностей используются *ядра*, которые могут включаться в один или несколько узлов графа. Примерами ядер служат виртуальные камеры, источники света, различные геометрические объекты, аффинные преобразования («трансформации») и рассеивающие свойства поверхностей («материалы»).

Параметры некоторых ядер (таких как трансформации) должны распространяться на дочерние вершины узла (и «накапливаться» от корня к листьям дерева). Для этой цели узлы графа в автоматическом режиме снабжаются набором *атрибутов*, которые применяются для «накопления» соответствующих данных. Примерами атрибутов являются «трансформации» (содержат матрицу аффинного преобразования), параметры отображения/сокрытия объектов (содержат флаг видимости) и «материалы» (содержат идентификатор некоторого материала). Каждому узлу графа сцены может быть присвоено несколько атрибутов (например, материал и трансформация). Механизм атрибутов работает автоматически и не требует вмешательства со стороны пользователя графа сцены (который оперирует привычными узлами и ядрами).

Далее обсуждаются основные архитектурные и программные решения, которые лежат в основе разработанного графа сцены.

3.2.3. Программные интерфейсы узлов, ядер и атрибутов графа сцены

Интерфейс узла. Компьютерная сцена представлена ориентированным деревом, каждая вершина (или узел) которого описывается следующим программным интерфейсом (показаны основные методы и аргументы функций):

- `Kernel kernel()`
Возвращает ядро данного узла.
- `void setKernel(Kernel kernel)`
Назначает данному узлу ядро `kernel`.
- `void removeKernel()`
Удаляет связь ядра с данным узлом (само ядро не удаляется).
- `void addChild(Node node)`
Добавляет к данному узлу дочерний узел `node`. В результате на дочернюю вершину распространяются все атрибуты текущего узла. Действует следующее ограничение: один и тот же узел не должен включаться в граф сцены более одного раза.
- `void removeChild(Node node)`
Исключает узел `node` из списка дочерних вершин данного узла.
- `Node parent()`
Возвращает родительскую вершину данного узла.
- `void update(real time)`
Обновляет состояние ядра данного узла и ядер всех дочерних узлов в момент времени `time` (если ядра зависят от времени) и выполняет их сериализацию (запись данных во внутренние хранилища-массивы).
- `Attribute[] attributes()`
Возвращает список атрибутов данного узла (позволяет редактировать, присоединять новые и удалять существующие атрибуты).
- `string name()`
Возвращает символьное имя данного узла.
- `void setName(string name)`
Назначает данному узлу символьное имя `name`.

Интерфейс ядра. Конкретные сущности сцены представлены ядрами, которые могут включаться в несколько вершин графа и описываются следующим базовым программным интерфейсом:

- `Node node()`
Возвращает узел, с которым связано данное ядро.
- `void assignNode(Node node)`
Устанавливает связь данного ядра с узлом `node`.
- `void update(real time)`
Обновляет состояние данного ядра в момент времени `time`.
- `string typeName()`
Возвращает символьное имя типа данного ядра.
- `Property[] properties()`
Возвращает список *свойств* данного ядра, которые выступают в роли стандартного механизма доступа и модификации параметров различных объектов графа сцены. С каждым свойством связано символьное имя и некоторое поле класса. Универсальная подсистема свойств была реализована с помощью шаблонов языка C++.

Основная функциональность ядра зависит от типа сущности. Например, ядро трансформации предоставляет набор методов для удобного задания аффинного преобразования.

Интерфейс атрибутов. Атрибуты определяют параметры узлов графа сцены и должны распространяться на все дочерние вершины. Реализация универсальной системы атрибутов сопряжена с рядом трудностей. Во-первых, все возможные варианты атрибутов заранее не известны (новые типы могут вводиться прикладным программистом). Во-вторых, некоторые типы атрибутов могут быть применимы только к определенным ядрам. В результате задание фиксированного набора атрибутов (например, в виде некоторого агрегатного типа данных) является крайне негибким подходом. Для решения указанной проблемы в системе вводится базовый интерфейс атрибутов, который реализует следующие операции:

- `Attribute defaultValue()`
Возвращает значение «по умолчанию» для атрибута данного класса (необходимо для «обнуления» атрибута при удалении ядра, чтобы старое значение не распространялось на новое ядро). Например, для атрибута трансформации базовым значением является единичное преобразование.

- Attribute clone()

Возвращает копию данного атрибута. Необходимость указанного метода обусловлена тем, что дочерние узлы наследуют атрибуты родителя, однако не должны зависеть от атрибутов «братьев». Иными словами, атрибуты родительской вершины должны быть скопированы.

- void merge(Attribute attribute)

Комбинирует (объединяет) данный атрибут с атрибутом *attribute*. На основе данной операции значения атрибутов «накапливаются» при движении от корневой вершины к листьям графа сцены. Реализация процедуры объединения зависит от типа атрибута. Для трансформаций производится перемножение матриц аффинных преобразований, для атрибутов сокрытия/отображения – логическое умножение, а для материалов – замена текущего значения новым.

Данный подход обеспечивает: 1) поддержку атрибутов произвольных типов; 2) возможность их «накапливания» в дочерних поддеревьях; 3) корректное изменение значений атрибутов в узлах-«братьях». Прикладной программист может вводить в систему дополнительные ядра и атрибуты, сохраняя остальные классы неизменными.

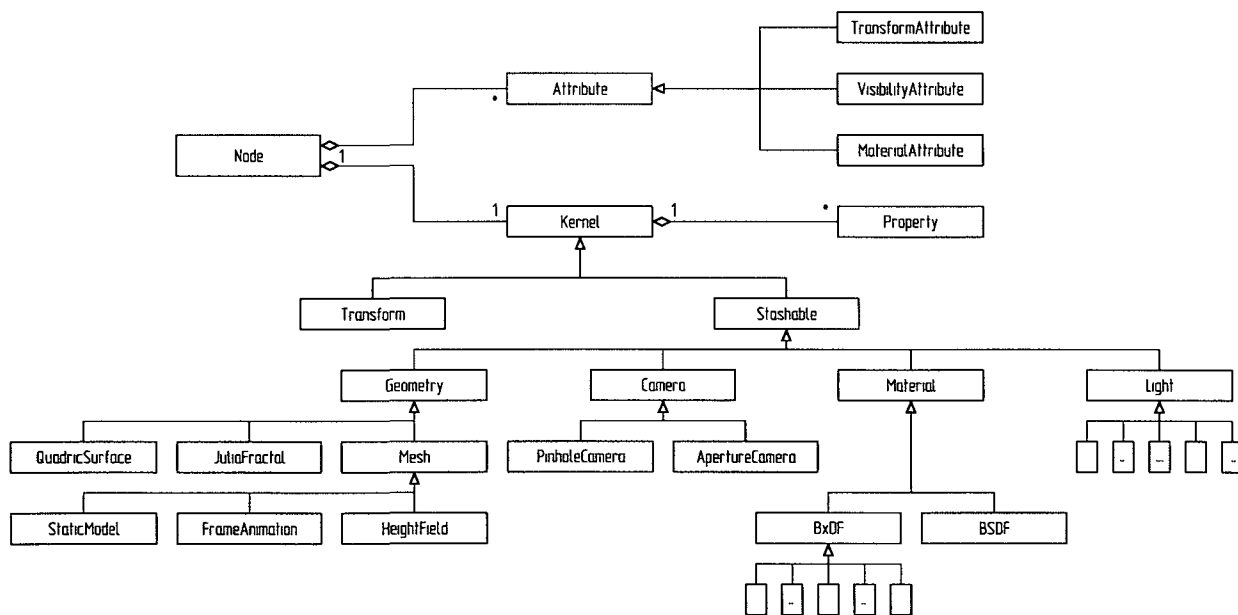


Рис. 3.3. Диаграмма основных классов графа сцены

(промежуточные классы опущены или заменены укрупненными блоками)

На рис. 3.3 изображена диаграмма основных классов подсистемы графа сцены, которая отражает взаимосвязь между узлами, ядрами и атрибутами. Ядра графа сцены, подлежащие

сериализации, отнаследованы от базового класса *Stashable* (реализация данного механизма рассматривается далее).

3.2.4. Механизм эффективной сериализации графа сцены

Как упоминалось выше, одной из ключевых особенностей разработанного графа сцены является возможность *сериализации* построенной иерархии объектов в потоки (массивы) для эффективной обработки на графическом процессоре. Для данной возможности существуют и другие применения. Например, подобный функционал позволяет пересылать данные по сети для организации распределенной визуализации. Широко распространенные системы графов сцены (OpenSG и OpenSceneGraph) не обеспечивают эффективную сериализацию иерархии объектов (для анимированных сцен данную процедуру следует выполнять на *каждом* кадре). Принципиальная необходимость механизма сериализации послужила основной мотивацией при разработке специализированной системы графа сцены.

С программной точки зрения для реализации сериализации необходимо поддерживать представление сцены в виде хранилищ-массивов (помимо представления в виде узлов графа) и обеспечить запись информации об объектах в данные хранилища. В системе применяются специализированные хранилища для объектов различных типов, базовая функциональность которых описывается следующим программным интерфейсом:

- `void addObject(Node node)`
Добавляет информацию об узле *node* в хранилище (однако память для записи данных объекта не выделяется).
- `void removeObject(Node node)`
Удаляет информацию об узле *node* из хранилища. Если для узла *node* была выделена память, то выполняется ее освобождение.
- `void allocateObject(Node node)`
Выделяет память для размещения данных узла *node* в хранилище (объем необходимой памяти определяется на основе типа ядра). При этом данные из ядра *не* записываются в хранилище.
- `void writeObject(Node node, Attribute attributes[])`
Записывает данные из ядра узла *node* в хранилище (память для хранения ядра должна выделяться заблаговременно). В ходе записи к данным ядра применяются *текущие* значения атрибутов узла *attributes* (например, выполняется трансформация вершин

полигональной сетки из локального пространства в мировое). Таким образом, на этапе сериализации выводится потоковое представление сцены в текущий момент времени. При альтернативном подходе наряду с данными узла необходимо записывать полный набор атрибутов, что приведет к дополнительному расходу памяти (в том числе и на графическом процессоре) и снизит производительность системы.

- `void freeObject(Node node)`

Освобождает память, которая была выделена в хранилище для хранения данных узла *node*. При этом информация об узле из хранилища не удаляется.

- `string typeName()`

Возвращает символьное имя типа данного хранилища.

При таком определении интерфейса хранилища идентификация объектов осуществляется по *узлам*. На практике данный подход является удобным и обеспечивает лаконичный механизм взаимодействия с хранилищами. Однако возникает дополнительный запрет на многократное включение узла в граф сцены. В рамках разработанной архитектуры данное ограничение не является существенным, поскольку узлы служат только для описания иерархии. Конкретные сущности хранятся в ядрах, которые могут быть включены в произвольное число узлов.

Для каждого класса *сериализуемых* объектов (на диаграмме отнаследованы от базового типа *Stashable*) в системе определяется специальный тип хранилища. Каждое хранилище содержит данные всех экземпляров соответствующего класса согласно принципу SoA (англ. Structure of Arrays). Реализация хранилищ основана на *контейнерах* стандартной библиотеки шаблонов STL (главным образом применяется динамический массив произвольного доступа *vector*). Перед визуализацией очередного кадра выполняется обход графа сцены, в процессе которого данные обновленных узлов записываются в соответствующие хранилища (список ассоциированных хранилищ определяется на уровне базового класса *Stashable*).

3.2.5. Программный интерфейс объектов графа сцены для разработки лучевых алгоритмов визуализации на графическом процессоре

В основе программного интерфейса визуализации на стороне графического процессора (применяется для разработки лучевых алгоритмов) лежит разделение объектов графа сцены на следующие категории:

Геометрические объекты. Элементы данного типа определяют некоторый трехмерный объект сцены. Важным преимуществом алгоритмов на основе трассировки лучей является возможность прямой визуализации *неполигональных* данных (поверхности второго порядка,

сплайновые и неявно заданные поверхности, объемные данные и фракталы). Чтобы в полной мере задействовать данную возможность в системе определяется следующий программный интерфейс геометрического объекта (на стороне ГПУ):

- `bool intersectNearest(Ray ray, out Intersection hit)`

Вычисляет *ближайшую* точку пересечения луча *ray* с геометрическими примитивами данного типа (набор примитивов хранится в массивах соответствующего хранилища). Информация о найденной точке пересечения (идентификатор примитива в хранилище и расстояние до соударения) записывается в переменную *hit*. Для поиска пересечения данный метод может задействовать различные ускоряющие структуры. В настоящей работе для эффективной обработки полигональных сеток применяется рассмотренная ранее иерархия объемов. Принципиальная возможность работы с неполигональными геометрическими объектами показана на примере поверхностей второго порядка и 4D фрактальных множеств Джулия.

- `bool intersectAny(Ray ray)`

Выполняет проверку *любого* пересечения луча *ray* с геометрическими примитивами данного типа. Указанная функция является оптимизированной версией предыдущей и позволяет повысить скорость обработки теневого луча (выполняет выход сразу после обнаружения *первого* соударения).

Такой подход к определению геометрии выгодно отличает настоящее решение от аналогов. На данный момент немногочисленные коммерческие системы визуализации (Octane Render, Arion 2, iRay, V-Ray RT) обеспечивают обработку только полигональных моделей.

Источники света. Элементы данного типа задают методы обработки источников света и описывают распределение излучаемой энергии. В настоящем исследовании источник света определяется следующим программным интерфейсом (на стороне ГПУ):

- `bool intersect(Ray ray, out vec3 point, out vec3 normal)`

Выполняет тест пересечения луча *ray* с источником света. В случае положительного исхода координаты и нормаль в точке пересечения заносятся в переменные *point* и *normal* соответственно. Данная функция необходима для обработки неявных путей переноса излучения.

- `color sample(vec3 ref, out vec3 point, out vec3 normal, out real pdf)`

Генерирует случайный теневой луч из точки соударения *ref* в направлении источника освещения. Начальная точка луча и нормаль к источнику записываются в переменные

point и *normal* соответственно. «Площадная» плотность вероятности (отнесенная к единичной площади) записывается в переменную *pdf*. В качестве результата функция возвращает «вес» теневого луча (включает плотность вероятности и геометрический член). Данная функция применяется для вычисления прямого освещения (на основе площадной формы уравнения визуализации).

- `color sample(out vec3 point, out vec3 normal, out vec3 direct, out real pdfA, out real pdfΩ)`

Генерирует произвольный луч из источника света, начальная точка и направляющий вектор которого заносятся в переменные *point* и *direct* соответственно. Нормаль к источнику света (в точке *point*) записывается в переменную *normal*. «Площадная» и «угловая» (отнесенная к единичному телесному углу) плотности вероятности (выбора начальной точки и направления) заносятся в переменные *pdfA* и *pdfΩ* соответственно. В качестве результата метод возвращает «вес» выбранного светового луча (включает «площадную» и «угловую» плотности вероятности). Данная функция применяется в двунаправленных лучевых алгоритмах для генерации световых путей.

- `real pdfPosition(vec3 point)`

Вычисляет «площадную» плотность вероятности, с которой точка *point* выбирается на поверхности источника света. Данный метод используется при вычислении веса «обратных» путей (явного и неявного) переноса излучения.

- `real pdfDirection(vec3 point, vec3 direct)`

Вычисляет «угловую» плотность вероятности, с которой в точке *point* на источнике света генерируется исходящее направление *direct*. Данный метод используется при вычислении веса «обратных» путей (явного и неявного) переноса излучения.

«Материалы». Элементы данного типа определяют визуальные свойства поверхностей сцены, которые часто описываются комбинированными BSDF (в прикладном программном обеспечении используется термин «*материал*»). В настоящей работе материал определяется следующим программным интерфейсом (на стороне ГПУ):

- `color handle(vec3 point, vec3 Ψ, vec3 Θ)`

Вычисляет BSDF в точке соударения *point* для заданной пары направлений Ψ и Θ. В текущей версии системы применяется 3-х канальная модель цвета (RGB), однако при необходимости она может быть расширена на *N*-полосный спектр.

- `void sample(vec3 point, vec3 Θ , out vec3 Ψ , out real pdf)`

Генерирует направление Ψ вторичного луча в некоторой точке *point* для заданного исходящего направления Θ . Значение «угловой» плотности вероятности записывается в переменную *pdf*.

- `real pdf(vec3 point, vec3 Ψ , vec3 Θ)`

Возвращает значение «угловой» плотности вероятности, с которой в заданной точке *point* для исходящего направления Θ может быть выбрано входящее направление Ψ .

Система позволяет задействовать любую модель материала, которая допускает реализацию указанного интерфейса (в данной работе применяются комбинированные BSDF).

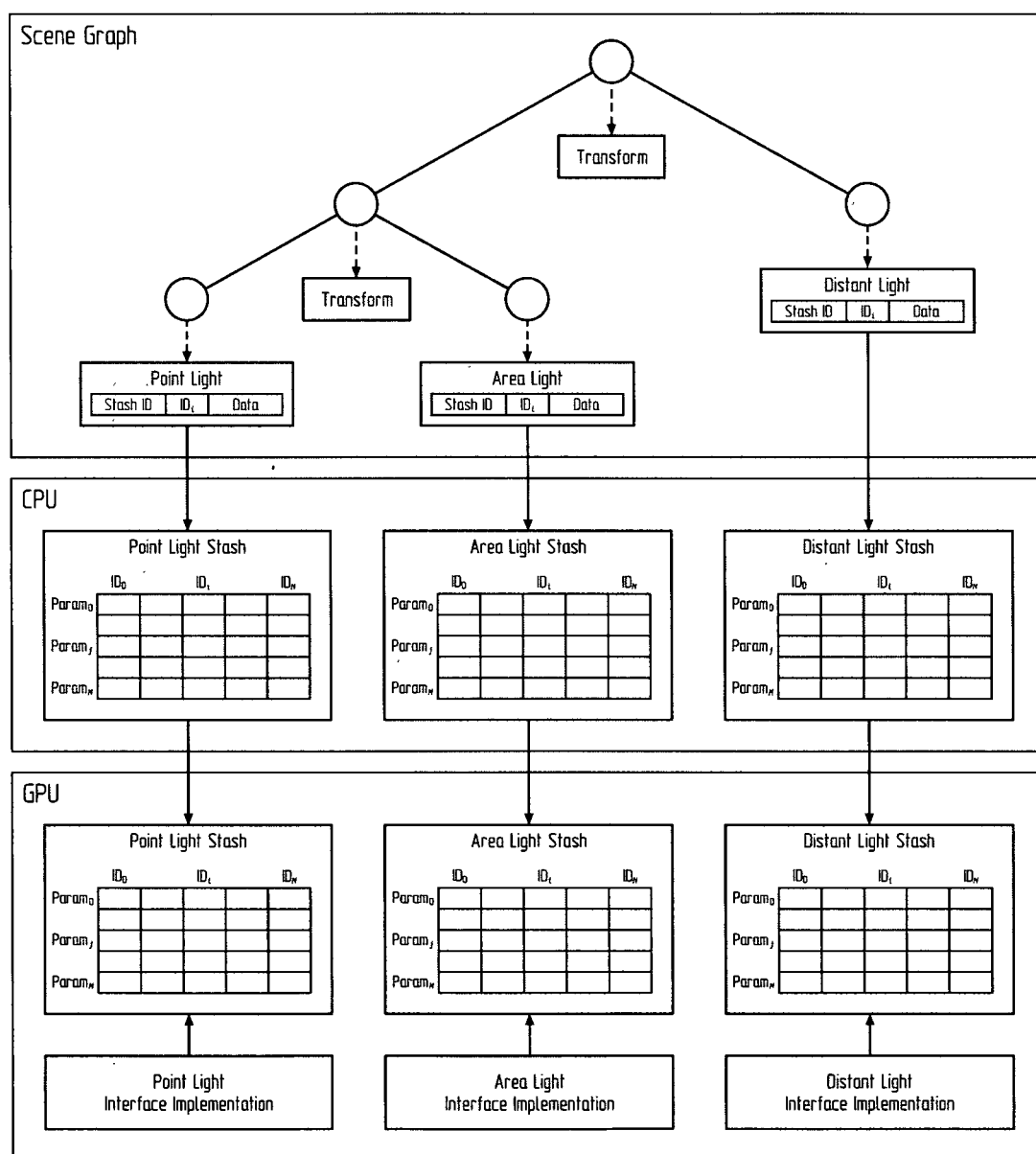


Рис. 3.4. Схема сериализации данных на примере некоторых источников света

Все перечисленные интерфейсы должны быть реализованы на стороне графического процессора, где использование виртуальных функций нежелательно (хотя и возможно). Для реализации концепции интерфейсов в данной работе применяются указатели на функции, с помощью которых записывается универсальный программный код визуализации (на языке CUDA C). Все *типы* геометрических объектов, материалов и источников света снабжаются уникальными идентификаторами и реализациями соответствующих интерфейсов. На этапе визуализации по идентификатору (через таблицу соответствия) определяется указатель на функцию, косвенное обращение по которому приводит к вызову нужного обработчика.

Входные данные для функций-обработчиков (такие как параметры источников света и материалов поверхностей) доступны посредством обращения к *массивам* соответствующих хранилищ (рис. 3.4). Для локализации данных конкретного объекта в хранилище функциям-обработчикам передается соответствующий идентификатор (смещение в массивах). В итоге обеспечивается универсальная платформа для разработки произвольных лучевых алгоритмов визуализации.

3.3. Компоненты универсальной системы синтеза изображений методами глобального освещения

3.3.1. Реализация компонентов для синтеза изображений в виде вычислительных блоков конвейера трассировки лучей

В рамках данного исследования ставилась задача разработки универсальной системы синтеза изображений методами глобального освещения, которая обеспечивает исполнение различных алгоритмов на основе трассировки лучей. Данные алгоритмы строятся на едином наборе операций, определенном в главе 2, и различаются типом генерируемых путей. Для описания траекторий переноса света в работе [143] была предложена компактная нотация, которая используется далее. Путь определяется последовательностью вершин, тип которых зависит от способа образования: L – вершина на источнике света, E – вершина на диафрагме камеры, S – зеркальный отскок от поверхности, D – диффузный (или «размыто-зеркальный») отскок от поверхности (рис. 3.5). Для записи путей используются регулярные выражения³. В частности, трассировка лучей Уиттеда строит пути ES^*L и ES^*DL , которые начинаются на диафрагме виртуальной камеры и после нескольких зеркальных отскоков завершаются либо на источнике света, либо на диффузной поверхности. Обратная стохастическая трассировка путей генерирует траектории вида $E(S|D)^*L$, которые начинаются на диафрагме виртуальной

³ Хопкрофт Д., Мотвани Р., Ульман Д. *Введение в теорию автоматов, языков и вычислений*. – Изд. Вильямс, 2002. – 528 с. – ISBN: 5-8459-0261-4, 0-201-44124-1.

камеры и спустя произвольное число любых отскоков достигают источника света. Прямая трассировка путей строит аналогичные траектории, которые начинаются от источника света – $L(S|D)^*E$.

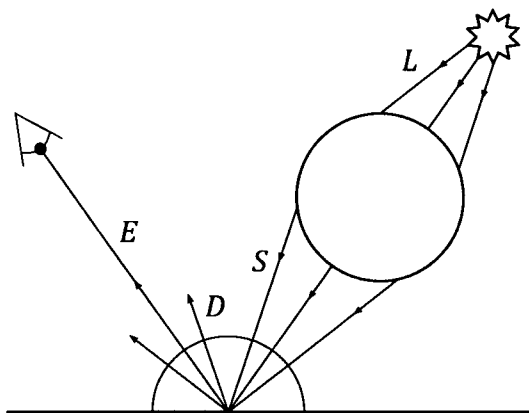


Рис. 3.5. Классификация точек соударения

Таким образом, можно определить набор универсальных операций (блоков) для генерации и обработки путей в лучевых алгоритмах визуализации. Блочное представление может быть проиллюстрировано на примере двунаправленной трассировки путей (рис. 3.6).

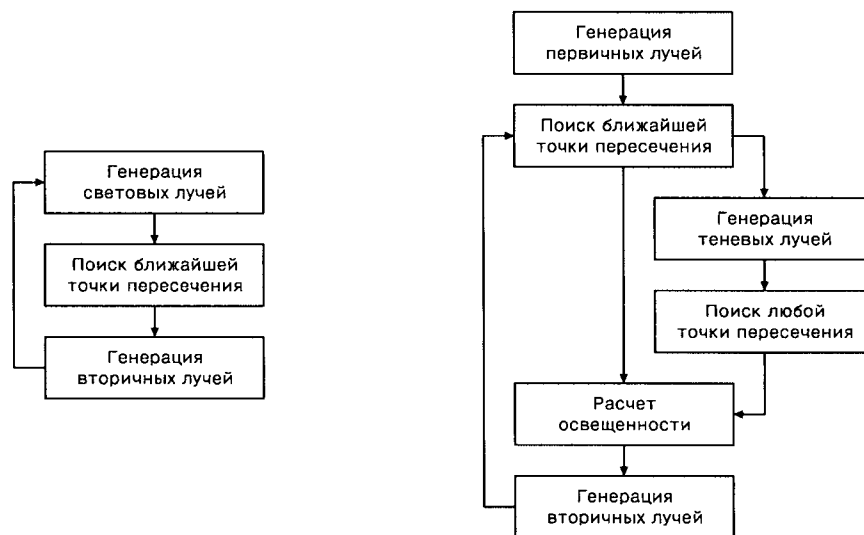


Рис. 3.6. Укрупненная блок-схема двунаправленной трассировки путей

Данный алгоритм генерирует пути одновременно из источника света («световой» путь типа $L(S|D)^*$) и диафрагмы виртуальной камеры («видовой» путь типа $E(S|D)^*$). Затем вершины построенных путей соединяются теневыми лучами, и производится расчет (потенциальных) вкладов в яркость некоторого пикселя. Такое представление двунаправленной трассировки путей позволяет выделить следующие базовые блоки.

Блок генерации первичных лучей. Выполняет генерацию первичных лучей из камеры. Способ выборки лучей определяется моделью оптической системы. В простейшем случае применяется модель камеры с точечной диафрагмой (стеноп), однако более универсальной является модель камеры с протяженной диафрагмой и тонкой линзой.

Блок генерации световых лучей. Выполняет генерацию лучей, исходящих из источника света (используется в двунаправленных алгоритмах расчета изображения). Способ выборки лучей зависит от модели источника света.

Блок расчета пересечений. Отвечает за вычисление точек пересечения с (ближайшим) объектом сцены. Как правило, данная операция выполняется с использованием ускоряющих структур. Однако для поиска соударения с неполигональными объектами могут применяться другие техники (например, при расчете пересечения с 4D фракталом Джулия используется метод «внешних» объемов – *unbounding volumes*⁴).

Блок генерации теневых лучей. Выполняет генерацию теневых лучей, соединяющих вершину «видового» пути либо с вершинами «светового» пути, либо со случайной точкой на поверхности источника света. Способ выборки точки зависит от модели источника света.

Блок расчета освещенности. Производит расчет вклада путей (методом Монте-Карло) в оценку яркости соответствующих пикселей. Для вычисления вкладов требуется доступ к моделям материалов, которые в общем случае описываются функциями BSDF/BSSRDF.

Блок генерации вторичных лучей. Выполняет генерацию вторичных лучей, с помощью которых происходит расширение пути. Способ выборки лучей зависит от модели материала.

Набор перечисленных блоков является достаточно универсальным и позволяет строить различные лучевые алгоритмы визуализации. Полное отключение фазы построения световых путей соответствует обычной стохастической трассировке путей. Если при этом генерация вторичных лучей выполняется только для расчета идеального отражения или преломления, то последовательность блоков реализует трассировку лучей Уиттеда. Полное отключение блока генерации вторичных лучей соответствует алгоритму испускания лучей.

В результате набор перечисленных блоков допускает адаптацию к различным задачам: – от полноценного моделирования глобального освещения до обработки динамических сцен в реальном времени и упрощенным освещением.

⁴ Описание “4D Quaternion Julia Set Ray Tracer” http://www.subblue.com/blog/2009/9/20/quaternion_julia

3.3.2. Архитектура универсального конвейера трассировки лучей для синтеза изображений методами глобального освещения

Центральным компонентом разработанной системы является универсальный конвейер трассировки лучей (по аналогии с традиционным графическим конвейером). Блоки данного конвейера позволяют строить различные алгоритмы визуализации, включая бросание лучей, классическую трассировку лучей, (прямую и обратную) стохастическую трассировку путей и («усеченную») двунаправленную трассировку путей (рис. 3.7). Переход между указанными алгоритмами выполняется путем простой настройки параметров конвейера (включения или отключения отдельных блоков). В совокупности с расширяемыми подсистемами объектов, материалов и источников света разработанный инструментарий становится универсальной платформой для исследования и построения лучевых алгоритмов синтеза изображений.

Каждый блок конвейера реализуется отдельным вычислительным ядром (или набором ядер), которые в настоящем исследовании были разработаны на основе технологии NVIDIA CUDA. Такой подход отличается от библиотеки трассировки лучей NVIDIA OptiX, в которой все этапы визуализации реализованы в одном ядре (“mega kernel”).

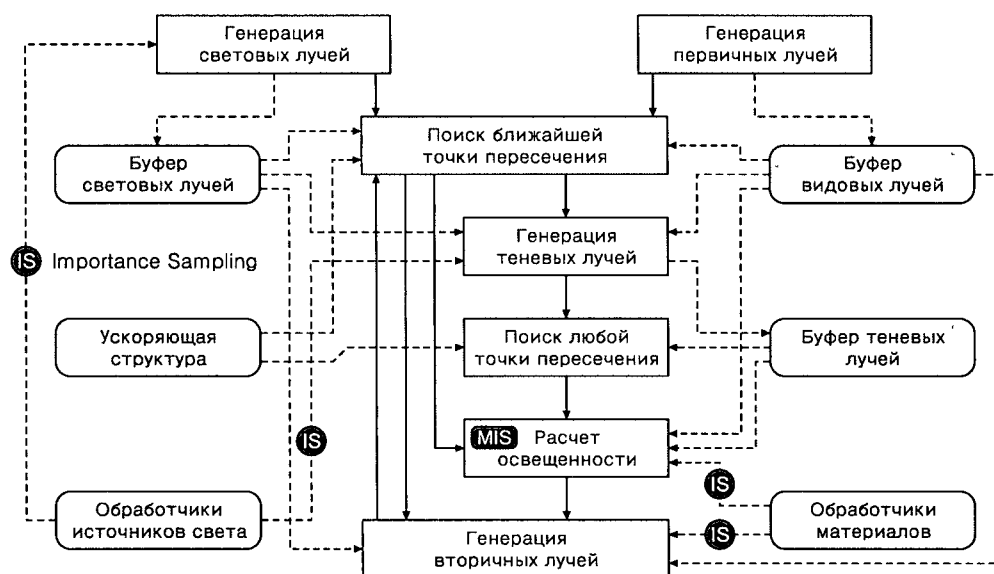


Рис. 3.7. Схема конвейера трассировки лучей и основных потоков данных

Все блоки разработанного конвейера используют рассмотренные выше универсальные ГПУ-интерфейсы для обработки геометрических примитивов, материалов и источников света. В результате конвейер получается независимым по отношению к конкретным типам указанных объектов и не требует модификаций при расширении возможностей системы (например, при добавлении новых моделей материалов и источников света). Далее обсуждаются ключевые особенности реализации основных ГПУ-интерфейсов.

3.3.3. Интерфейсы блока расчета пересечений луча с тесселированными и нетесселированными геометрическими примитивами

Для поиска точек соударения с произвольными геометрическими примитивами блоки расчета пересечений были разбиты на последовательность процедур, которые обрабатывают определенный тип примитивов и вызывают соответствующие реализации ГПУ-интерфейса объектов. Разработанная ускоряющая структура на основе иерархии объемов применяется только для поиска точек соударения с *полигональными* сетками, которые задаются набором треугольников. Для других объектов (ближайшее) пересечение вычисляется путем полного перебора. Такой подход приемлем в ситуации, когда объекты других типов присутствуют в незначительном количестве и встраиваются в полигональное окружение (в противном случае необходимо применять ускоряющие структуры).

Для демонстрации возможности обработки различных нетесселированных объектов в настоящей работе реализована поддержка поверхностей второго порядка и 3D-проекций 4D фрактальных множеств Джулиа. Данные фрактальные множества являются естественным обобщением двумерных фракталов Джулиа на четырехмерное пространство кватернионов. Фрактал Джулиа определяется совокупностью точек сходимости последовательности:

$$z_{n+1} = z_n^2 + c \quad (3.6)$$

В данной формуле z_n и c – кватернионы. В качестве z_0 задается точка, в которой исследуется сходимость последовательности (рис. 3.8).

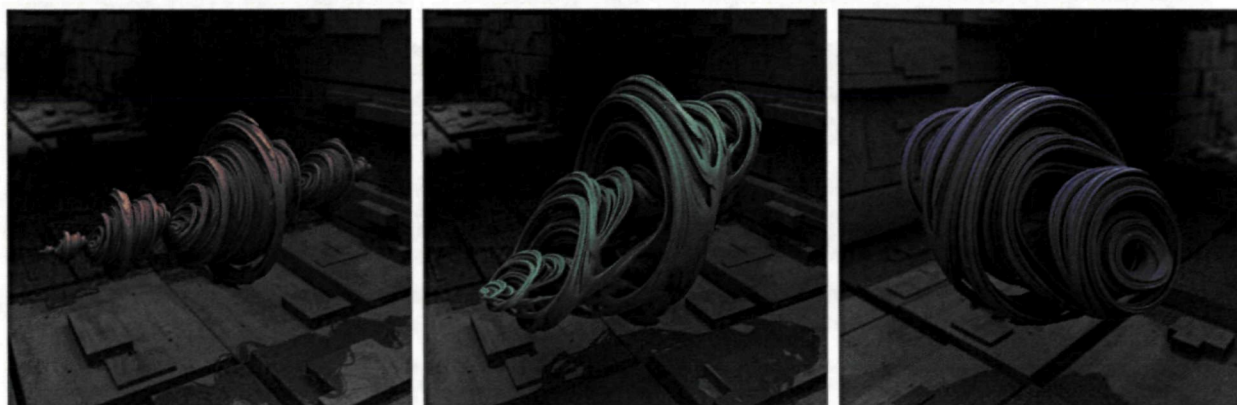


Рис. 3.8. Пример визуализации неполигональных объектов (проекция 4D фрактала Джулиа)

Для визуализации данных множеств используется принцип редукции размерности: четвертая координата заменяется некоторым фиксированным значением и строится трехмерный «срез» фрактала (аналогично сечениям трехмерных фигур).

3.3.4. Построение расширяемой подсистемы источников света. Классы и модели типовых источников света

В рамках настоящего исследования разработана расширяемая подсистема источников света. Для повышения наглядности и удобства реализации в работе применяется следующая классификация. На верхнем уровне иерархии все источники разделяются на два класса:

- *Локальные.* К данному классу относятся источники, расположение и размеры которых определяются конечными координатами. Данный класс подразделяется на *точечные* и *протяженные* (площадные) источники света.
- *Бесконечно удаленные.* К данному классу относятся источники, которые расположены на бесконечном расстоянии от сцены (и могут иметь бесконечные размеры). Данный класс подразделяется на *направленные* и *бесконечно-площадные* источники света.

На стороне центрального процессора указанная классификация реализована в виде иерархии классов (на языке C++), которые предназначены для хранения параметров соответствующих источников освещения (рис. 3.9). Указанные параметры доступны для редактирования как традиционным способом (посредством методов доступа), так и через список свойств.

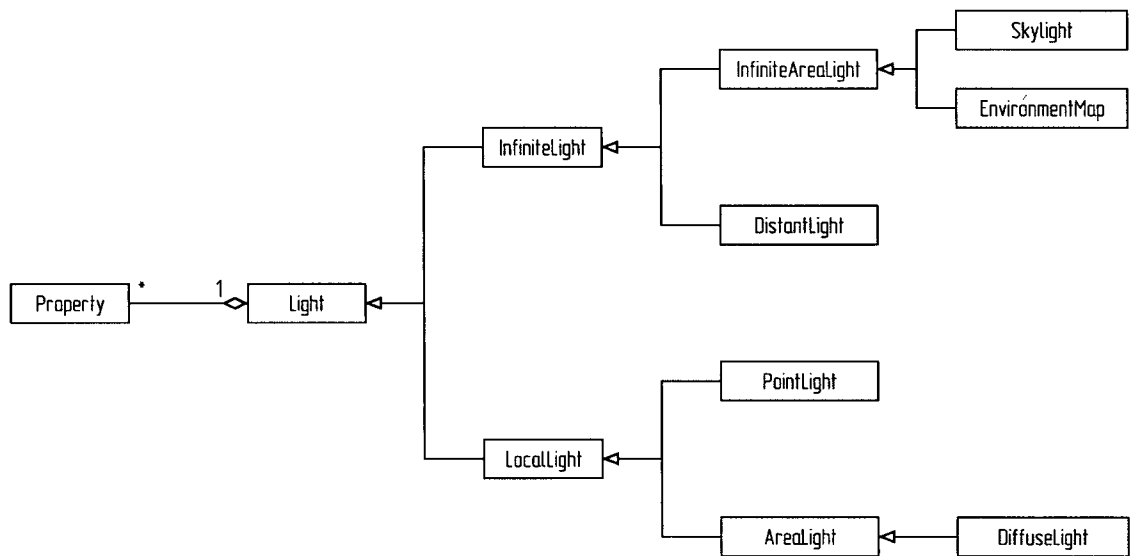


Рис. 3.9. Диаграмма реализованных (в текущей версии) источников света

Конкретные функции для обработки каждого типа источника света определяются на стороне графического процессора (CUDA C) и реализуют указанный выше программный интерфейс. Далее рассматриваются ключевые особенности реализации данного интерфейса.

Точечный источник света. Изотропный точечный источник определяется *силой света* I , которая одинакова во всех направлениях пространства. Однако реализованные алгоритмы

направлены на обработку *энергетической яркости* L . Поэтому в настоящей работе точечные источники трактуются как сферические диффузные источники, размеры которых ничтожно малы по сравнению с геометрией сцены. При таком подходе задание яркости «точечного» источника вполне корректно.

Для генерации теневого луча или начальной вершины светового пути на «поверхности» точечного источника необходимо выбрать случайную точку \mathbf{z} . Соответствующую плотность вероятности можно выразить δ -функцией:

$$p_A(\mathbf{z}) = \delta(\mathbf{z} - \mathbf{z}_p) \quad (3.7)$$

Через \mathbf{z}_p здесь обозначено положение источника в пространстве. Строго говоря, при таком определении плотности обнуляются вклады явных и «световых» путей переноса излучения, поскольку δ -функция принимает бесконечно большое значение. Формально проблему можно устранить за счет записи излучаемой яркости в виде:

$$L_e(\mathbf{z} \rightarrow \Psi) = \delta(\mathbf{z} - \mathbf{z}_p)L_e \quad (3.8)$$

В результате одинаковые δ -функции (в числителе и знаменателе вкладов) будут сокращены, поэтому в ходе вычислений достаточно учитывать лишь коэффициенты перед δ -функциями. «Вес» теневого луча (учитывает плотность вероятности и геометрический член) для точки \mathbf{x} поверхности записывается следующим образом:

$$\left(N_{\mathbf{x}} \cdot \frac{\mathbf{z}_p - \mathbf{x}}{\|\mathbf{z}_p - \mathbf{x}\|} \right) \frac{L_e}{\|\mathbf{z}_p - \mathbf{x}\|^2} \quad (3.9)$$

Для выбора направления случайного луча, исходящего из точечного источника света, применяется равномерное распределение:

$$p_{\omega}(\Psi) = \frac{1}{4\pi} \quad (3.10)$$

«Вес» исходящего луча вычисляется согласно следующей формуле (учитывает «площадную» и «угловую» плотность вероятности):

$$4\pi L_e \quad (3.11)$$

Направленный источник света. Данный источник света расположен в бесконечности и имеет единственное направление D падающего излучения. Примером такого источника (до определенной степени точности) является естественный солнечный свет.

Для генерации направления теневого луча из некоторой точки соударения применяется следующая плотность вероятности:

$$p_{\omega}(\Psi) = \delta(\Psi + D) \quad (3.12)$$

«Вес» теневого луча (учитывает плотность вероятности и геометрический член) для точки \mathbf{x} поверхности вычисляется следующим образом:

$$(N_{\mathbf{x}} \cdot (-D))L_e \quad (3.13)$$

Для генерации случайной точки на «поверхности» направленного источника в данной работе используется адаптированный метод из [144]. Строится круг, радиус которого равен радиусу ограничивающей сферы сцены, а нормаль ориентирована в направлении падающего излучения D . Круг выносится за границы сцены и располагается в касательной плоскости к ограничивающей сфере. Внутри круга с помощью равномерного распределения выбирается точка \mathbf{z} , которая возвращается в качестве искомой случайной точки. Плотность вероятности определяется по формуле:

$$p_A(\mathbf{z}) = \frac{1}{\pi R_s^2} \quad (3.14)$$

Символом R_s здесь обозначен радиус ограничивающей сферы сцены (рис. 3.10).

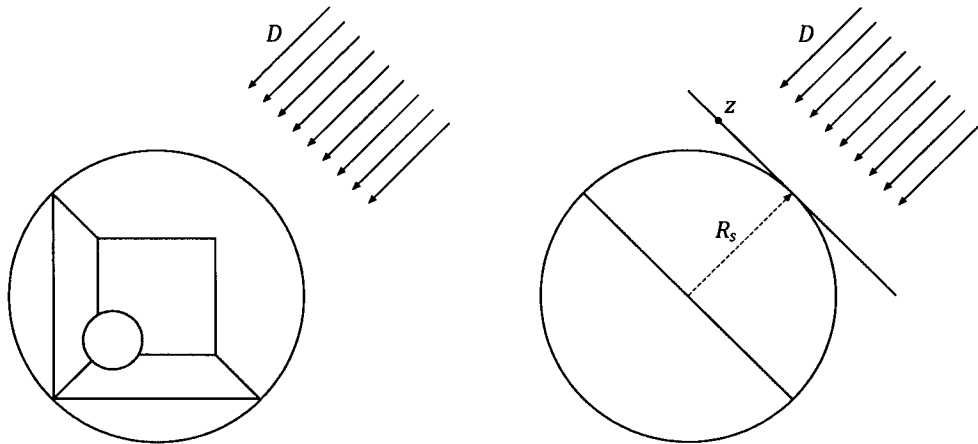


Рис. 3.10. Генерация точки на «поверхности» направленного источника

Для генерации направления случайного луча, исходящего из направленного источника света, используется плотность вероятности:

$$p_{\omega}(\Psi) = \delta(\Psi - D) \quad (3.15)$$

«Вес» исходящего луча (включает в себя «площадную» и «угловую» плотности вероятности) вычисляется согласно следующей формуле:

$$\pi R_s^2 L_e \quad (3.16)$$

Протяженный диффузный источник света. В общем случае площадные (протяженные) источники света определяются некоторой фигурой (конечных размеров), которая излучает энергетическую яркость. В текущей версии системы реализованы *диффузные* протяженные источники, которые излучают энергию одинаково в каждой точке и по всем направлениям полупространства. Для генерации случайной точки \mathbf{z} на поверхности площадного источника используется равномерное распределение:

$$p_A(\mathbf{z}) = \frac{1}{A_L} \quad (3.17)$$

Символом A_L здесь обозначена площадь протяженного источника. Сгенерированная точка \mathbf{z} применяется для построения теневого луча или используется в качестве начальной вершины «светового» пути. «Вес» теневого луча (учитывает плотность вероятности и геометрический член) для точки \mathbf{x} поверхности имеет вид:

$$\left(N_x \cdot \frac{\mathbf{z} - \mathbf{x}}{\|\mathbf{z} - \mathbf{x}\|}\right) \left(N_z \cdot \frac{\mathbf{x} - \mathbf{z}}{\|\mathbf{x} - \mathbf{z}\|}\right) \frac{A_L L_e}{\|\mathbf{x} - \mathbf{z}\|^2} \quad (3.18)$$

Для генерации направления случайного луча, исходящего из протяженного источника света, используется следующая плотность вероятности:

$$p_\omega(\Psi) = \frac{(N_z \cdot \Psi)}{\pi} \quad (3.19)$$

«Вес» исходящего луча (включает в себя «площадную» и «угловую» плотности вероятности) вычисляется согласно следующей формуле:

$$\pi A_L L_e \quad (3.20)$$

Верхний свет. Данный тип источника света моделируется сферой бесконечно большого радиуса, которая окружает сцену. Некоторые системы 3D визуализации содержат сложную модель «небесного света», которая учитывает различные эффекты освещения атмосферой и солнцем в заданное время суток. В настоящей работе (для демонстрации принципиальной возможности обработки таких источников) окружающая сфера моделируется как *диффузный* источник света. Для генерации направления теневого луча из некоторой точки соударения применяется равномерное распределение:

$$p_\omega(\Psi) = \frac{1}{4\pi} \quad (3.21)$$

«Вес» теневого луча (учитывает плотность вероятности и геометрический член) в некоторой точке \mathbf{x} поверхности имеет вид:

$$4\pi(N_x \cdot \Psi)L_e \quad (3.22)$$

По аналогии с направленным источником света, для генерации точки на «поверхности» бесконечно-площадного источника используется ограничивающая сфера сцены. Поскольку данный источник света является диффузным, для выбора точки \mathbf{z} применяется равномерное распределение:

$$p_A(\mathbf{z}) = \frac{1}{4\pi R_S^2} \quad (3.23)$$

Для выбора направления случайного луча, исходящего из точки \mathbf{z} ограничивающей сферы, используется следующая плотность вероятности:

$$p_\omega(\Psi) = \frac{(N_z \cdot \Psi)}{\pi} \quad (3.24)$$

«Вес» исходящего луча (включает в себя «площадную» и «угловую» плотности вероятности) вычисляется согласно следующей формуле:

$$4\pi^2 R_S^2 L_e \quad (3.25)$$



Рис. 3.11. Результаты визуализации для различных источников света (точечный, направленный, верхний свет, площадной, карта окружения и комбинация)

Карта окружения. Карта окружения является частным случаем бесконечно-площадного источника света, для которого задана HDRI текстура (определяет яркость источника вдоль различных направлений). Для генерации исходящего или теневого луча в настоящей работе применяется выборка по значимости. Стандартный подход к построению и использованию соответствующих функций плотности вероятности описан в книге [144].

3.3.5. Построение расширяемой подсистемы материалов. Концепция «атомарных» Монте-Карло материалов и ее реализация. Модели атомарных материалов

Для описания свойств поверхностей разработана расширяемая подсистема *материалов*, в основе которой лежит декомпозиция механизмов рассеяния света на диффузное, размытое зеркальное и зеркальное. Для описания конкретных типов рассеяния в работе применяются «атомарные» (неделимые) двунаправленные функции распределения BRDF и BTDF. С одной стороны, такой подход позволяет построить гибкую систему описания материалов на основе комбинирования произвольных «атомарных» функций. С другой стороны, данные функции позволяют реализовать для составных материалов выборку по значимости и используются в качестве базовых блоков метода Монте-Карло (схема выборки по значимости дана в главе 2).

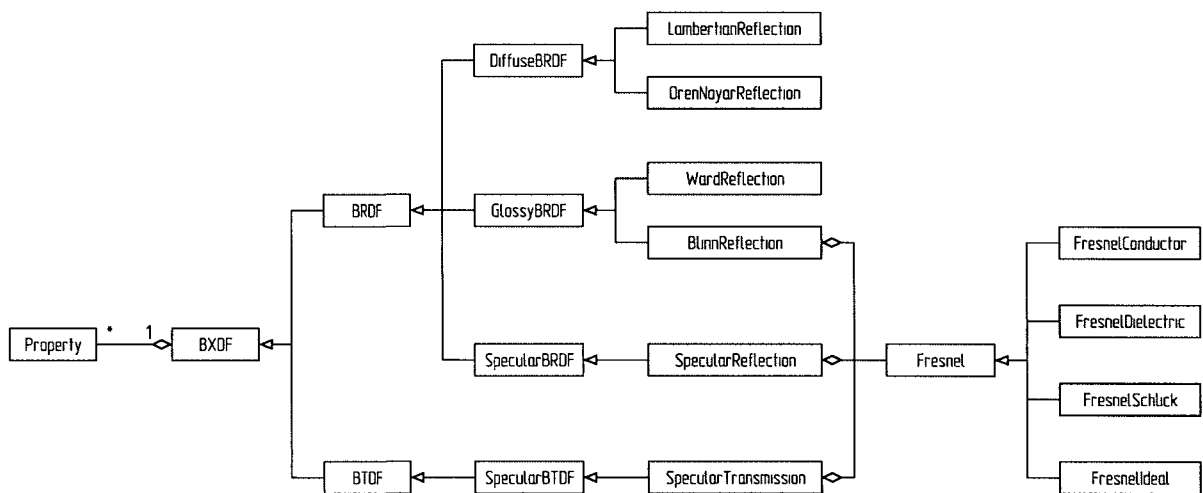


Рис. 3.12. Диаграмма реализованных (в текущей версии) «атомарных» материалов

По аналогии с источниками света, для работы с материалами на стороне центрального процессора разработана иерархия классов C++ (рис. 3.12). Данные классы предназначены для хранения параметров материалов и реализуют следующий программный интерфейс:

- `color weight()`

Возвращает весовой коэффициент данной BxDF.

- `void setWeight(color weight)`
Устанавливает весовой коэффициент `weight` для данной VxDF.
- `Texture texture()`
Возвращает текстурную карту данной VxDF.
- `void setTexture(Texture texture)`
Устанавливает текстурную карту `texture` для данной VxDF.
- `Property[] properties()`
Возвращает список свойств данной VxDF (зависит от конкретной реализации, однако всегда включает весовой коэффициент и текстурную карту).

Конкретные функции для работы с «атомарными» материалами в алгоритмах визуализации определяются на стороне графического процессора (на языке CUDA C) и реализуют базовый ГПУ-интерфейс материала.

В текущей версии системы текстурные данные передаются на графический процессор в виде стандартных массивов CUDA. Отказ от использования специальной текстурной памяти ведет к некоторой потере производительности, однако позволяет снять ограничения на число обрабатываемых текстур, их размер и формат пикселя. В рамках данного подхода методы фильтрации реализуются программно. На данный момент поддерживается интерполяция по ближайшему элементу и билинейная интерполяция.

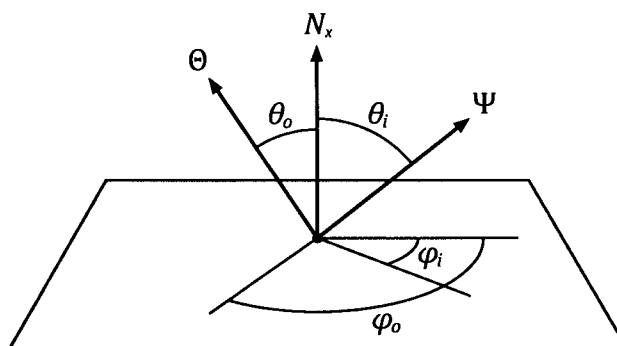


Рис. 3.13. Используемые обозначения для записи VxDF

Коэффициенты Френеля. Для моделирования реалистичных отражений и преломлений необходимо учитывать тот факт, что доля отраженной (и преломленной) световой энергии не является постоянной величиной. Она зависит от угла падения и показателей преломления на границе двух сред и описывается формулами (коэффициентами) Френеля. Данные формулы определяются отдельно для *проводников* (англ. conductor) и *диэлектриков* (англ. dielectric) и

зависят от поляризации падающего света. Для упрощения вычислений в настоящей работе принимается допущение, что свет является *неполяризованным* (определяется как смесь волн со случайными равновероятными направлениями поляризации). В этом случае коэффициент Френеля равен полусумме квадратов коэффициентов для параллельной и перпендикулярной поляризации света (*p*-поляризация и *s*-поляризация соответственно):

$$F_r = \frac{1}{2}(r_{\parallel}^2 + r_{\perp}^2) \quad (3.26)$$

Для диэлектрика (*FresnelDielectric*) доля отраженной световой энергии вычисляется с помощью следующей аппроксимации:

$$r_{\parallel} = \frac{\eta_o \cos \theta_i - \eta_i \cos \theta_o}{\eta_o \cos \theta_i + \eta_i \cos \theta_o} \quad (3.27)$$

$$r_{\perp} = \frac{\eta_i \cos \theta_i - \eta_o \cos \theta_o}{\eta_i \cos \theta_i + \eta_o \cos \theta_o}$$

Через η_i и η_o здесь обозначены показатели преломления на границе двух сред, Ψ и Θ задают направления падения и преломления соответственно.

Для описания проводников (*FresnelConductor*) наряду с показателем преломления η вводится коэффициент поглощения k (часть падающей энергии поглощается и преобразуется в тепловую форму). При расчете коэффициентов Френеля для проводников использовалась следующая аппроксимация:

$$r_{\perp}^2 = \frac{(\eta^2 + k^2) - 2\eta \cos \theta_i + \cos^2 \theta_i}{(\eta^2 + k^2) + 2\eta \cos \theta_i + \cos^2 \theta_i} \quad (3.28)$$

$$r_{\parallel}^2 = \frac{(\eta^2 + k^2) \cos^2 \theta_i - 2\eta \cos \theta_i + 1}{(\eta^2 + k^2) \cos^2 \theta_i + 2\eta \cos \theta_i + 1}$$

Наряду с достаточно аккуратной обработкой проводников и диэлектриков в настоящей работе реализована универсальная аппроксимация Шлика (*FresnelSchlick*):

$$F_r = R_0 + (1 - R_0)(1 - \cos \theta_i)^5 \quad (3.29)$$

Единственным параметром указанной аппроксимации является коэффициент отражения R_0 при нормальном падении. Если известны показатели преломления на границе двух сред, то данную величину можно вычислить по формуле:

$$R_0 = \left(\frac{\eta_i - \eta_o}{\eta_i + \eta_o} \right)^2 \quad (3.30)$$

Для моделирования идеального зеркального отражателя реализован специальный тип коэффициента Френеля (*FresnelIdeal*), который независимо от угла падения отражает всю световую энергию ($F_r = 1$). Этот тип материала не является физически корректным, однако применяется в системе при отсутствии дополнительной информации (например, при импорте 3D моделей из большинства форматов хранения).



Рис. 3.14. Пример визуализации с учетом коэффициентов Френеля

Модель отражения Ламберта. Данная модель применяется для описания поверхностей, которые отражают падающий свет равномерно по всем направлениям полупространства (так называемые «диффузные» отражатели). Для наблюдателя диффузная поверхность выглядит одинаково под любым направлением взгляда. Значение BRDF не зависит от конфигурации направлений и определяется некоторой константой ρ :

$$f_r(\mathbf{x}, \Psi \rightarrow \Theta) = \frac{\rho}{\pi} \quad (3.31)$$

Коэффициент ρ («вес» BRDF) характеризует отражательную способность поверхности и для корректных материалов должен находиться в диапазоне $[0, 1]$.

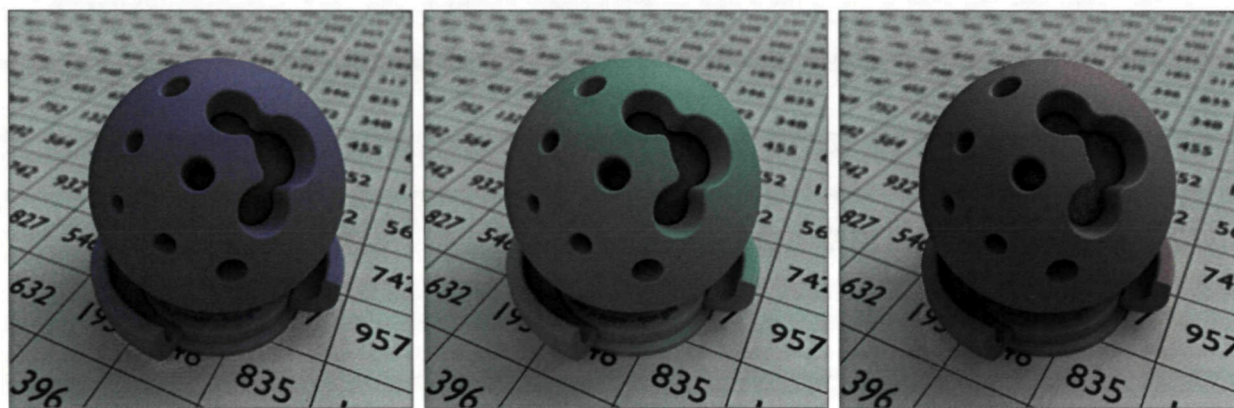


Рис. 3.15. Результаты визуализации для BRDF Ламберта

Модель Ламберта является идеализацией, однако достаточно хорошо описывает некоторые реальные поверхности (подобные матовой краске). Для генерации направления вторичного луча в данном случае применяется универсальная стратегия:

$$p_{\omega}(\Psi) = \frac{(N_x \cdot \Psi)}{\pi} \quad (3.32)$$

Модель отражения Орена-Найара. Альтернативная модель диффузного отражения была предложена в работе [146] Ореном и Найаром и описывает рассеяние света на «шершавых» поверхностях. В данной модели такие поверхности представляются набором симметричных V-образных микро-бороздок, при этом каждая грань (бороздки) рассеивает падающий свет по закону Ламберта.

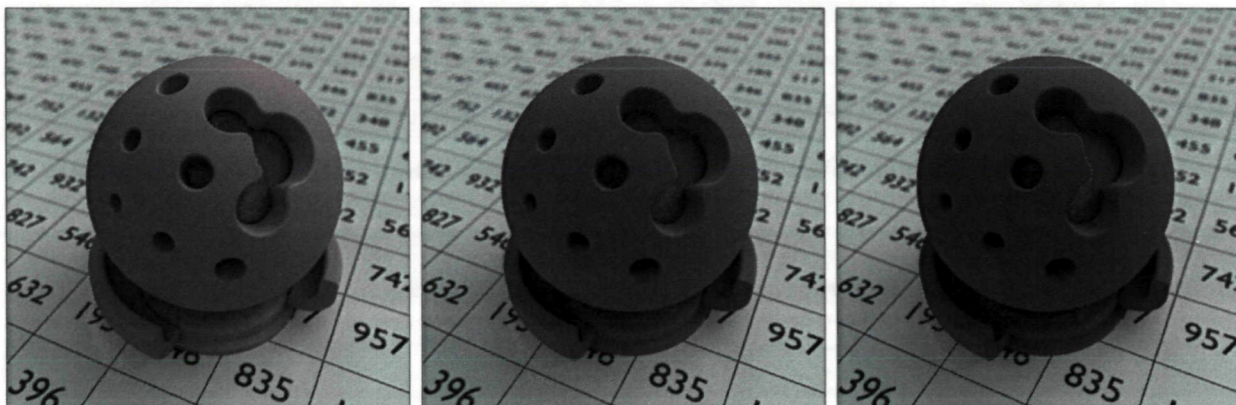


Рис. 3.16. Результаты визуализации для BRDF Орена-Найара при различных коэффициентах шероховатости ($\sigma = 0.1$, $\sigma = \pi/4$, $\sigma = \pi/2$)

Возможные ориентации нормалей к микрограням описываются гауссовским распределением со стандартным отклонением σ (от усредненной нормали к поверхности). Тогда отражение света от «шершавой» поверхности *приближенно* определяется BRDF Орена-Найара:

$$f_r(\mathbf{x}, \Psi \rightarrow \Theta) = \frac{\rho}{\pi} \left[A + B \max(0, \cos(\phi_i - \phi_o)) \sin(\alpha) \tan(\beta) \right] \quad (3.33)$$

В данном выражении

$$\begin{aligned} A &= 1 - 0.5 \frac{\sigma^2}{\sigma^2 + 0.33} \\ B &= 0.45 \frac{\sigma^2}{\sigma^2 + 0.09} \\ \alpha &= \max(\theta_i, \theta_o) \\ \beta &= \min(\theta_i, \theta_o) \end{aligned} \quad (3.34)$$

Стандартное отклонение σ выражается в радианах и находится в диапазоне $\left[0, \frac{\pi}{2}\right]$. Нетрудно видеть, что при $\sigma = 0$ модель Орена-Найара соответствует модели Ламберта. На практике для вычисления данной BRDF применяется тождество:

$$\cos(\phi_i - \phi_o) = \cos \phi_i \cos \phi_o + \sin \phi_i \sin \phi_o \quad (3.35)$$

Тогда все тригонометрические функции в выражении BRDF эффективно вычисляются через операцию скалярного произведения.

Следует подчеркнуть, что понятие «шершавости» в модели Орена-Найара не означает *визуально* наблюдаемой нерегулярности материала (по аналогии с наждачной бумагой). Речь идет о *микроструктуре* поверхности, которая порождает определенный характер рассеяния света (слабо зависит от направления падающего излучения). В этом смысле пластик является относительно гладким материалом, а резина или камень – шершавым.

Для генерации направления вторичного отскока при обработке BRDF Орена-Найара в настоящей работе применяется плотность вероятности (3.32).

Модель идеального отражения. Данный тип BRDF описывается δ -функцией, которая обращается в ноль всюду, кроме направления идеального зеркального отражения R_Θ (где она принимает бесконечно большое значение):

$$f_r(\mathbf{x}, \Psi \rightarrow \Theta) = \rho F_r(\Psi) \frac{\delta(R_\Theta - \Psi)}{|N_x \cdot \Psi|} \quad (3.36)$$

Доля отраженной световой энергии определяется коэффициентом Френеля F_r . Направление R_Θ вычисляется по закону идеального отражения:

$$\Theta_R = 2(N_x \cdot \Theta)N_x - \Theta \quad (3.37)$$

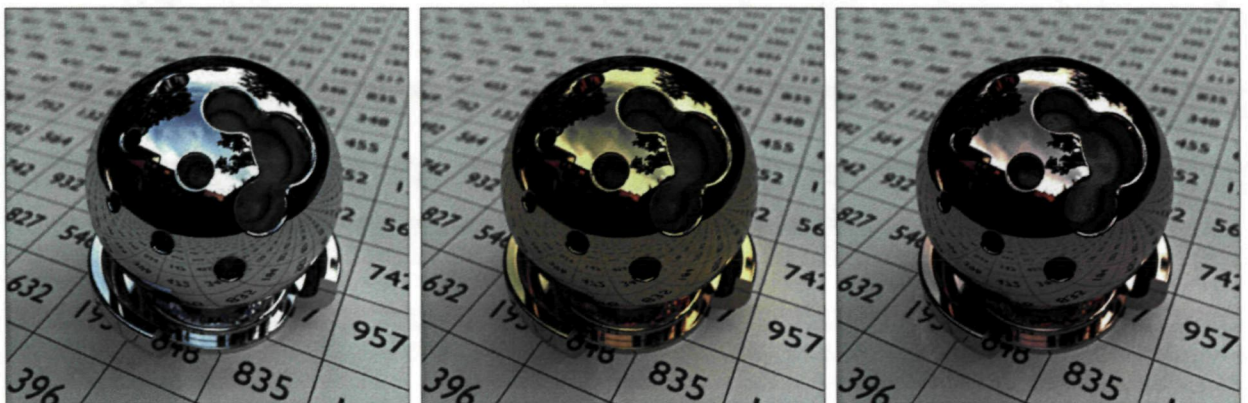


Рис. 3.17. Результаты визуализации различных металлов (*серебро, золото, медь*) для BRDF идеального отражения

На практике принято считать, что для произвольной пары направлений BRDF идеального зеркального отражения равна нулю. В результате вклад в отражаемую яркость вносят только *вторичные* лучи. При этом плотность вероятности для выбора случайных направлений также выражается через δ -функцию:

$$p_{\omega}(\Psi) = \delta(R_{\Theta} - \Psi) \quad (3.38)$$

При расчете вклада пути с «зеркальным» отскоком сокращаются все члены соответствующей BRDF, за исключением коэффициентов отражения, поскольку

$$p_{\omega^{\perp}}(\Psi) = \frac{\delta(R_{\Theta} - \Psi)}{|N_{\mathbf{x}} \cdot \Psi|} \quad (3.39)$$

Таким образом, в формулах (3.36) и (3.38) достаточно учитывать лишь *коэффициенты* перед δ -функциями (сами функции можно опустить).

Модель идеального преломления. Аналогично «зеркальной» BRDF данный тип BTDF описывается δ -функцией, которая обращается в ноль всюду, кроме направления идеального преломления T_{Θ} :

$$f_r(\mathbf{x}, \Psi \rightarrow \Theta) = \rho \frac{\eta_{\Theta}^2}{\eta_{\Psi}^2} (1 - F_r(\Psi)) \frac{\delta(T_{\Theta} - \Psi)}{|N_{\mathbf{x}} \cdot \Psi|} \quad (3.40)$$

При отсутствии потерь энергии коэффициент пропускания выражается через коэффициент отражения Френеля F_r . Изменение яркости преломленного луча Ψ определяется квадратом отношения коэффициентов преломления двух сред $\eta_{\Theta}^2 / \eta_{\Psi}^2$.



Рис. 3.18. Результаты визуализации для BTDF идеального преломления при различных коэффициентах преломления ($\eta = 1.3, \eta = 1.6, \eta = 1.9$)

Направление идеального преломления T_{Θ} на границе двух сред описывается законом Снелла. На практике данный закон удобно применять в векторной форме:

$$T_{\Theta} = -\frac{\eta_{\Theta}}{\eta_{\Psi}} \Theta + N_x \left[\frac{\eta_{\Theta}}{\eta_{\Psi}} (N_x \cdot \Theta) - \sqrt{1 - \frac{\eta_{\Theta}^2}{\eta_{\Psi}^2} (1 - (N_x \cdot \Theta)^2)} \right] \quad (3.41)$$

Если световая энергия падает из оптически более плотной среды в оптически менее плотную, то вместо преломления может возникать полное внутреннее отражение в исходную плотную среду. Эффект наблюдается при углах падения, которые превышают критический угол θ_c :

$$\theta_c = \arcsin\left(\frac{\eta_{\Psi}}{\eta_{\Theta}}\right) \quad (3.42)$$

В рамках принципа «атомарных» (неделимых) VxDF случай полного внутреннего отражения реализован как отдельная BRDF идеального отражения, а в VTDF идеального преломления в случае полного внутреннего отражения выполняется остановка пути (за счет чего возникают черные области на изображении, полученном с помощью VTDF). Для генерации вторичного луча в VTDF идеального преломления применяется следующая плотность вероятности:

$$p_{\omega}(\Psi) = \delta(T_{\Theta} - \Psi) \quad (3.43)$$

По аналогии с «зеркальной» BRDF в формулах (3.40) и (3.43) достаточно вычислять только коэффициенты δ -функций.

Модель размытого зеркального отражения Блинна. Данная BRDF является типичным представителем «микрограневых» моделей, в которых поверхность представляется набором мельчайших V-образных бороздок. В отличие от модели Орена-Найара каждая микрогрань выступает в роли идеального *зеркального* отражателя. Торренсом и Спарроу был предложен общий вид «микрограневой» BRDF:

$$f_r(\mathbf{x}, \Psi \rightarrow \Theta) = \rho \frac{D(H) \cdot G(\Theta, \Psi) \cdot F(\Theta, H)}{4(N_x \cdot \Psi)(N_x \cdot \Theta)} \quad (3.44)$$

В данном выражении H – вектор половинного направления, который определяет ориентацию микрограней, вносящих вклад в отраженную яркость:

$$H = \frac{\Theta + \Psi}{\|\Theta + \Psi\|} \quad (3.45)$$

Символом $D(H)$ обозначен закон распределения, который определяет долю микрограней с нормалью H . Наряду с этим модель Торренса и Спарроу содержит коэффициент отражения Френеля $F(\Theta, H)$ и геометрический фактор $G(\Theta, \Psi)$, учитывающий эффекты *экранирования, самозатенения и переотражения*:

$$G(\theta, \psi) = \min \left(1, \min \left(2 \frac{(N_x \cdot H)(N_x \cdot \Theta)}{(\Theta \cdot H)}, 2 \frac{(N_x \cdot H)(N_x \cdot \Psi)}{(\Theta \cdot H)} \right) \right) \quad (3.46)$$

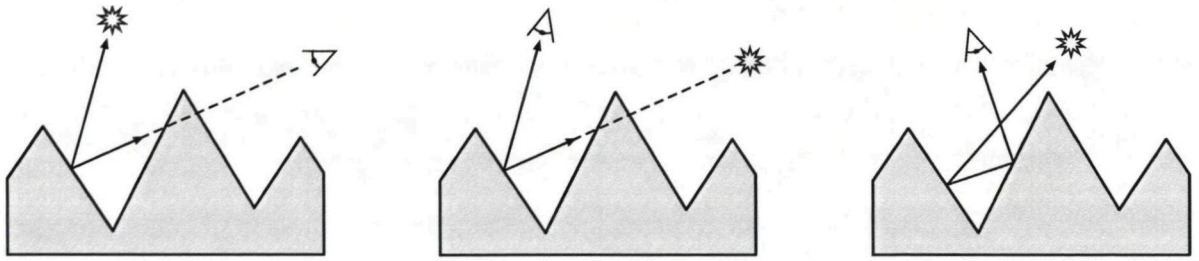


Рис. 3.19. Эффекты (слева направо) экранирования, самозатенения и переотражения

В модели Блинна доля микрограней с нормалью H экспоненциально убывает по мере отклонения от усредненной нормали N_x к поверхности:

$$D(H) = \frac{e + 2}{2\pi} (H \cdot N_x)^e \quad (3.47)$$

Показатель e иногда называется коэффициентом резкости отражений и определяет степень «шероховатости» материала (большие значения соответствуют гладким поверхностям).



Рис. 3.20. Результаты визуализации для BRDF Блинна при различных коэффициентах резкости ($e = 64$, $e = 512$, $e = 2048$)

Для выбора направления вторичного луча в настоящей работе применяется плотность вероятности, приближенно пропорциональная распределению $D(H)$:

$$p_\omega(\Psi) = \frac{e + 1}{4(\Theta \cdot H)} (H \cdot N_x)^e \quad (3.48)$$

Модель размытого зеркального отражения Уорда. Данная BRDF является достаточно удобной эмпирической моделью, которая позволяет имитировать анизотропные поверхности

и хорошо согласуется с результатами измерений (при правильной настройке параметров). В наиболее общем виде BRDF определяется следующим образом [146]:

$$f_r(\mathbf{x}, \Psi \rightarrow \Theta) = \frac{\rho}{4\pi\alpha_x\alpha_y\sqrt{\cos\theta_i\cos\theta_o}} e^{-\tan^2\theta_h\left(\frac{\cos^2\phi_h}{\alpha_x^2} + \frac{\sin^2\phi_h}{\alpha_y^2}\right)} \quad (3.49)$$

В данной формуле параметры α_x и α_y определяют степень «шероховатости» поверхности в вдоль осей x и y (в касательном пространстве).

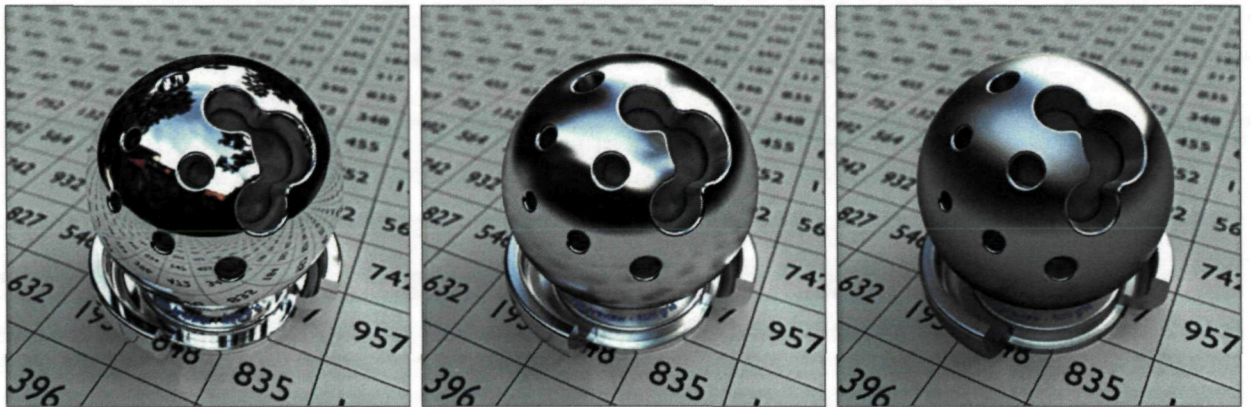


Рис. 3.21. Результаты визуализации для изотропной BRDF Уорда ($\alpha_x = \alpha_y$) при различных коэффициентах «шероховатости» ($\alpha = 0.01, \alpha = 0.1, \alpha = 0.3$)

Для генерации направления вторичного луча в данной работе применяется плотность вероятности, приблизительно пропорциональная BRDF:

$$p_\omega(\Psi) = \frac{\rho}{4\pi\alpha_x\alpha_y(H \cdot \Theta) \cos^3\theta_h} e^{-\tan^2\theta_h\left(\frac{\cos^2\phi_h}{\alpha_x^2} + \frac{\sin^2\phi_h}{\alpha_y^2}\right)} \quad (3.50)$$

3.3.6. Построение расширяемой подсистемы материалов. Составные материалы

Большинство реалистичных материалов описываются комбинацией нескольких типов отражения или пропускания света. В настоящей работе свойства поверхностей описываются *составными* BSDF (f_s), которые являются взвешенной суммой «атомарных» BxDF (f):

$$f_s(\mathbf{x}, \Psi \rightarrow \Theta) = \sum_{i=1}^N \alpha_i f_i(\mathbf{x}, \Psi \rightarrow \Theta) \quad (3.51)$$

Весовые коэффициенты α_i в общем случае являются векторными и определяются отдельно для каждой полосы спектра (в текущей версии системы применяется трехканальный спектр).

Программный интерфейс комбинированного материала (BSDF) определяется следующим образом:

- `Material create(VxDF bxdf)`
Создает комбинированный материал из одной VxDF компоненты *bxdf*.
- `Material create(VxDF bxdfs[])`
Создает комбинированный материал из списка VxDF компонент *bxdfs*.
- `void insert(VxDF bxdf)`
Добавляет к данному комбинированному материалу VxDF компоненту *bxdf*.
- `void remove(VxDF bxdf)`
Исключает из данного комбинированного материала VxDF компоненту *bxdf*.
- `VxDF[] components()`
Возвращает список всех VxDF компонент данного комбинированного материала.
- `color emission()`
Возвращает (энергетическую) яркость, которая *равномерно* излучается материалом во всех направлениях полупространства.
- `color setEmission()`
Устанавливает (энергетическую) яркость, которая *равномерно* излучается материалом во всех направлениях полупространства.
- `Property[] properties()`
Возвращает список свойств данного комбинированного материала (состоит из списка VxDF компонент и излучаемой яркости).

Поскольку для каждой VxDF компоненты наряду с весовым коэффициентом α_i может быть определена текстурная карта T_i и коэффициент Френеля F_i , фактические коэффициенты $\tilde{\alpha}_i$ (на момент обработки материала) определяются следующим образом:

$$\tilde{\alpha}_i = \alpha_i T_i F_i \quad (3.52)$$

Для сохранения физической корректности материала необходимо обеспечить, чтобы сумма всех фактических весовых коэффициентов не превышала единицы (за соблюдением данного правила должен следить пользователь системы визуализации). Например, материал стекла

может быть смоделирован за счет комбинации VxDF идеального *отражения* и *преломления* с единичными весовыми коэффициентами:

$$f_s(\mathbf{x}, \Psi \rightarrow \Theta) = F_r(\Psi) \frac{\delta(R_\Theta - \Psi)}{|N_x \cdot \Psi|} + \frac{\eta_\Theta^2}{\eta_\Psi^2} (1 - F_r(\Psi)) \frac{\delta(T_\Theta - \Psi)}{|N_x \cdot \Psi|} \quad (3.53)$$

В данном случае правильный баланс между компонентами соблюдается за счет применения коэффициентов Френеля.

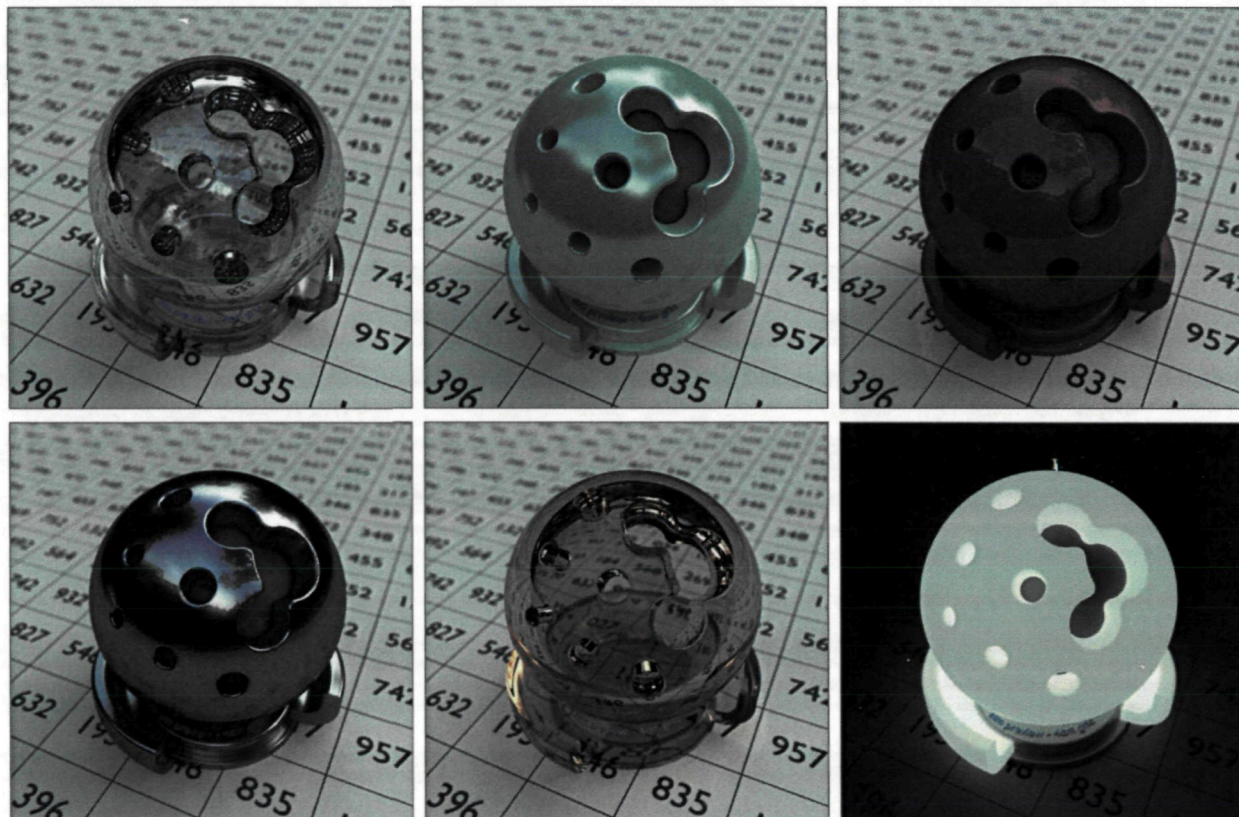


Рис. 3.22. Результаты визуализации для некоторых составных BSDF

Другим типовым материалом является комбинация VxDF Ламберта и Блинна (по аналогии с моделью материалов OpenGL):

$$f_s(\mathbf{x}, \Psi \rightarrow \Theta) = \frac{k_d}{\pi} + \frac{k_s(e+2)}{8\pi} \frac{G(\Theta, \Psi) \cdot F(\Theta, H)}{(N_x \cdot \Psi)(N_x \cdot \Theta)} (H \cdot N_x)^e \quad (3.54)$$

Здесь весовые коэффициенты k_d и k_s в сумме не должны превышать единицы (для каждого цветового канала). Еще одним примером комбинированного материала является простейшая модель автомобильной краски. Данное покрытие можно рассматривать как слой прозрачного лака, за которым следует слой относительно матовой краски. Лак выступает в роли идеально гладкой прозрачной поверхности, для которой доля отраженной и преломленной световой

энергии выражается коэффициентами Френеля. При этом преломленный свет рассеивается на диффузном слое краски:

$$f_s(\mathbf{x}, \Psi \rightarrow \Theta) = F_r(\Psi) \frac{\delta(R_\Theta - \Psi)}{|N_x \cdot \Psi|} + (1 - F_r(\Psi)) \frac{k_d}{\pi} \quad (3.55)$$

В данном выражении коэффициент k_d определяет цвет матовой краски под слоем лака.

Для генерации направления вторичного отскока относительно комбинированной BSDF в работе применяется принцип выборки по значимости (соответствующий алгоритм описан в главе 2).

3.3.7. Оптимизационные техники уплотнения потоков и контроля сходимости

«Уплотнение» потоков. Для оптимизации вычислений в настоящей работе применяется техника «уплотнения» потоков [114]. Данная процедура запускается перед каждой итерацией алгоритма и перемещает лучи с ненулевыми весами в начало потока обрабатываемых лучей. В результате из вычислительного процесса исключаются «непродуктивные» лучи, которые были остановлены методом «русской рулетки» или покинули пределы сцены.

Контроль сходимости. Из практических расчетов методом Монте-Карло известно, что при большом числе выборочных значений N поведение погрешности δ_N может быть описано следующей формулой:

$$\delta_N \sim \frac{H}{\sqrt{N}} \quad (3.56)$$

Здесь H – некоторая константа, которая зависит от дисперсии оценки. Обозначим символом I_N приближенное значение искомого интеграла (уравнение измерения) для N выборочных значений. Тогда для достаточно больших N выполняется:

$$|I_{2N} - I_N| \sim \frac{2H}{\sqrt{N}} = \frac{C}{\sqrt{N}} \quad (3.57)$$

Данный факт позволяет определять и исключать из вычислительного процесса пиксели, для которых уже достигнута сходимость (с некоторой точностью). За счет этого время обработки изображения может многократно сокращаться (существенно зависит от конкретной сцены). На практике проверку сходимости следует запускать каждые K итераций и спустя некоторое число T начальных итераций. Величины K и T являются параметрами и позволяют добиться оптимального компромисса между качеством и временем расчета изображения. В настоящей

работе по умолчанию приняты значения $K = 50$ и $T = 200$. Процедуру контроля сходимости можно выразить следующим псевдокодом:

```
void Estimate( Image currImage, Image prevImage, real error, out bool mask[][] )
{
    for( int i = 0; i < width; i++ )
        for( int j = 0; j < height; j++ )
        {
            color prevPixel = prevImage[i][j]
            color currPixel = currImage[i][j]

            if( norm( currPixel - prevPixel ) / norm( prevPixel ) < error )
                mask[i][j] = 0
            else
                mask[i][j] = 1
        }
}
```

В данном псевдокоде можно использовать любую норму цвета. В настоящей работе для этой цели применяется евклидова векторная норма:

$$\|\mathbf{x}\|_2 = \sqrt{\sum_i |\mathbf{x}_i|^2} \quad (3.58)$$

На практике для получения качественного результата «ошибку» (*error*) требуется задавать в диапазоне 0.01 – 0.03.

3.4. Выводы к главе 3

Результаты исследования, представленные в главе 3, создают достаточно эффективную основу для создания программных систем виртуальной реальности, обеспечивающих синтез реалистичных изображений методами глобального освещения на современных графических процессорах. Ведущими положениями этой основы являются следующие:

1. Эффективная ускоряющая структура на основе иерархии объемов, которая отличается наличием высокопроизводительного ячеечного алгоритма построения на графическом процессоре. Алгоритм выполняет разбиение всех узлов дерева с помощью эвристики площадей поверхностей (SAH) и обеспечивает 5-кратное повышение быстродействия по сравнению с наилучшими известными результатами [102] (на момент подготовки публикации [150]).

2. Универсальный графический конвейер трассировки лучей, полностью реализованный на графическом процессоре (при этом на центральном процессоре сохраняется только управляющая логика). За счет настройки данный конвейер позволяет воспроизводить различные лучевые методы визуализации, включая испускание лучей, классическую трассировку лучей, прямую и обратную трассировку путей, а также двунаправленную трассировку путей (в том числе предложенный в главе 2 «усеченный» вариант).
3. Универсальная структура представления и обработки компьютерных сцен гибридной природы (сочетают тесселированные и нетесселированные данные), которая включает в себя специализированный граф сцены, расширяемые подсистемы источников света, материалов (BSDF) и геометрических объектов, ускоряющую структуру.

Глава 4

Программный комплекс для интерактивного синтеза изображений сложных сцен на графических процессорах методами глобального освещения и его экспериментальное исследование

Данная глава посвящена экспериментальному исследованию разработанного комплекса и отвечает следующим целям:

- Представить описание состава и структуры разработанного программного комплекса для интерактивного синтеза изображений методами глобального освещения, который реализует положения глав 2 и 3 с использованием инструментов программирования параллельных графических архитектур.
- Экспериментально проверить корректность моделирования глобального освещения в различных условиях переноса световой энергии в сцене путем сравнения результатов моделирования с эталонными изображениями.
- Дать оценку производительности программного комплекса, реализованных моделей и методов в задачах интерактивного синтеза фотореалистичных изображений, а также оценку эффективности предложенных решений относительно аналогов.

4.1. Состав и структура программного комплекса интерактивного синтеза изображений методами глобального освещения

4.1.1. Высокопроизводительные свободно распространяемые библиотеки нижнего уровня

В разработанном программном комплексе активно применяются следующие свободно распространяемые библиотеки:

- *Eigen* [<http://eigen.tuxfamily.org>]. Библиотека шаблонов на языке C++ для линейной алгебры. Характеризуется универсальностью, высоким уровнем производительности, лаконичностью программных интерфейсов (вектора и матрицы) и поддержкой многих

компиляторов. Высокое быстродействие достигается посредством применения SIMD-расширений ЦПУ и техник метапрограммирования (таких как Expression Templates). Библиотека доступна с исходным кодом по лицензиям LGPL3 и GPL2.

- *Boost* [<http://www.boost.org>]. Собрание библиотек, расширяющих язык C++ подобно STL (Standard Template Library). Библиотеки *Boost* ускоряют массу задач прикладного программирования, предоставляя типичные алгоритмы, контейнеры данных, «умные указатели», примитивы многопоточного программирования и многое другое. Данные библиотеки доступны с исходным кодом по лицензии Boost Software License.
- *Assimp* [<http://assimp.sourceforge.net>]. Мультиплатформенная библиотека на языке C++ для загрузки и препроцессирования моделей 3D сцен из популярных форматов хранения, включая Collada, Blender 3D, 3D Studio Max 3DS/ASE, DirectX X, Wavefront Object, Stanford PLY и ряда других. Среди возможностей препроцессирования стоит отметить построение касательного пространства, «геометрических» и «сглаженных» нормалей. Библиотека распространяется с исходным кодом по лицензии BSD.
- *FreeImage* [<http://freeimage.sourceforge.net>]. Вспомогательная библиотека на языке C для загрузки изображений (текстур) из популярных форматов графических файлов, включая некоторые форматы с широким диапазоном (OpenEXR и HDR). Библиотека доступна с исходным кодом по лицензиям GPL2+ и FreeImage Public License (FIPL).
- *Qt* [<http://qt-project.org>]. Кросс-платформенный инструментарий разработки ПО на языке C++. Содержит основные классы, которые могут потребоваться при разработке прикладного программного обеспечения, включая элементы графического интерфейса пользователя. Инструментарий Qt является полностью объектно-ориентированным и поддерживает технологию компонентного программирования. Библиотека доступна с исходным кодом по лицензиям GPL3 и LGPL.
- *GLog* [<http://code.google.com/p/google-glog>]. Вспомогательная библиотека на языках C/C++, которая позволяет записывать в журнал приложения указанные разработчиком события (application-level logging). Данный функционал позволяет упростить отладку и выявление проблем. Библиотека распространяется с исходным кодом по лицензии New BSD.
- *GLEW* [<http://sourceforge.net/projects/glew>]. Мультиплатформенная библиотека на языке C для работы с расширениями OpenGL (поддерживает версию 4.3). Доступна с исходным кодом по лицензиям BSD и MIT.

- *GLFW* [<http://www.glfw.org/index.html>]. Мультиплатформенная библиотека на языке C для создания контекста OpenGL и управления вводом, включая клавиатуру, мышь и джойстик. Распространяется с исходным кодом по лицензии zlib/libpng.

Наряду с указанными библиотеками в разработанном комплексе применяются инструменты программирования графических процессоров NVIDIA CUDA и OpenCL. Для визуализации 3D сцены в отладочном режиме применяется интерфейс программирования OpenGL.

4.1.2. Структура программного комплекса и состав его библиотек прикладных программ

Разработанный программный комплекс состоит из набора модулей, которые основаны на перечисленных выше базовых библиотеках (рис. 4.1). На нижнем слое функциональности расположены библиотеки *Graphics* и *Property*. Библиотека *Graphics* отвечает за поддержку типовых задач программирования 3D графики OpenGL, включая управление виртуальными камерами и устройствами ввода, загрузку изображений, настройку шейдерных программ и буферов кадров. Библиотека *Property* реализует универсальный механизм свойств классов (на основе шаблонов языка C++), которые применяются для описания параметров объектов графа сцены. На среднем слое расположены библиотеки *Compute* и *Scene Graph*. Библиотека *Compute* содержит вспомогательные классы для работы с NVIDIA CUDA, которые упрощают работу с буферами (массивами), поиск ошибок и профилировку CUDA-программ. Наряду с этим в данной библиотеке определяются наборы функций CUDA C для работы с векторами, генерации случайных величин, выполнения операций в касательном пространстве. Основная функциональность программного комплекса содержится в библиотеке *Scene Graph*, которая реализует рассмотренное в главе 3 иерархическое объектно-ориентированное представление сцены, расширяемые подсистемы геометрических объектов, материалов и источников света, а также программный конвейер трассировки лучей (путей). Верхний слой функциональности представлен библиотеками *Convert* и *Qt GUI*, которые отвечают за импорт моделей 3D сцен из популярных форматов хранения (во внутренний формат системы) и содержат некоторые элементы графического интерфейса Qt (многорежимное окно визуализации с возможностью переключения между различными лучевыми алгоритмами глобального освещения).

На основе верхнего уровня функциональности разрабатываются различные клиентские программы, включая тестовые и демонстрационные приложения, «плагины» к системам 3D моделирования и специализированные системы визуализации. На данный момент разработан автономный визуализатор с возможностью редактирования графа сцены и ранний прототип «плагина» к системе моделирования Blender.

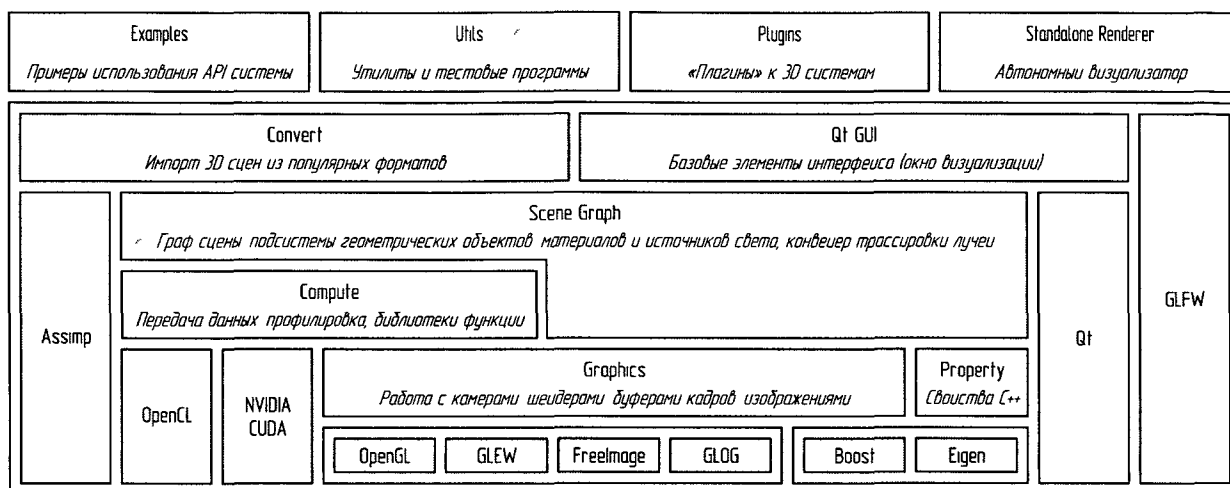


Рис. 4.1. Основные компоненты разработанного программного комплекса

В совокупности разработанные библиотеки и приложения содержат свыше 500 файлов программного кода (более 45 000 строк).

4.2. Экспериментальная проверка корректности моделирования глобального освещения в различных условиях переноса световой энергии в сцене

Проверка синтезированных изображений на физическую корректность (при отсутствии фактических экспериментальных данных) довольно проблематична. Алгоритмы глобального освещения воспроизводят такие сложные явления, как многократные отражения, мягкие тени и каустики. В подобной ситуации разработчик склонен объяснять «артефакты» изображения неочевидными проявлениями данных эффектов. Ситуация дополнительно усложняется при расчете методом Монте-Карло, который вносит в изображение случайное отклонение. Часто подобные отклонения проявляются в виде «светящихся» пикселей (англ. «fireflies»), которые субъективно неотличимы от многих типовых проблем реализации. Поэтому при разработке алгоритмов глобального освещения значительное внимание следует уделять тестированию.

4.2.1. Методика проверки корректности

В настоящей работе корректность реализованных алгоритмов проверялась с помощью сравнения полученных изображений с результатами промышленной системы визуализации. В качестве данной системы был выбран визуализатор Cycles Render [128], который входит в стандартную поставку бесплатного пакета 3D моделирования Blender (версия 2.66). Cycles построен на основе стохастической трассировки путей и обеспечивает расчет несмещенного

изображения. Данная система прошла тестирование большим сообществом пользователей и разработчиков и продолжает активно развиваться. Среди альтернативных визуализаторов с открытым исходным кодом стоит отметить LuxRender [127], который изначально направлен на синтез физически достоверных изображений. Однако данная система не допускает вывод *линейных* значений яркости в HDR изображение (соответствующих настроек обнаружить не удалось). Внутренние значения яркости конвертируются в выбранное цветовое пространство и могут подвергаться другим преобразованиям, поэтому результаты визуализации не удастся сравнить объективно. Для визуализатора Cycles система Blender позволяет настроить вывод изображения в линейном пространстве. Тестирование разработанной системы выполнялось согласно следующей схеме:

- Для каждой тестовой сцены средствами Cycles вычислялось «эталонное» изображение (размера 256×256), при этом число *семплов* (случайных путей) на каждый пиксель устанавливалось равным 2×10^5 .
- Средствами разработанной системы визуализации генерировалась серия изображений (размера 256×256) с различным числом семплов на пиксель. Сходимость данных изображений к «эталону» исследовалась путем попиксельного вычисления разности согласно некоторой норме.

Конкретный выбор нормы не имеет большого значения, поэтому в настоящем исследовании используется простейшая норма L2 (англ. Sum of Squared Differences, SSD). На псевдокоде алгоритм сравнения изображений выглядит следующим образом:

```
real Compare( Image image, Image reference )
{
    real diff = 0
    for( int x = 0; x < image.width; x++ )
        for( int y = 0; y < image.height; y++ ) {
            color delta = image( x, y ) - reference( x, y )
            diff += ( delta.R )2 + ( delta.G )2 + ( delta.B )2
        }
    return sqrt( diff )
}
```

Далее приводятся результаты измерения сходимости для некоторых компьютерных сцен.

4.2.2. «Классический» тест на диффузных объектах сцены Cornell Box

В данном тесте моделируется классическая сцена типа *Cornell Box*. Поверхности всех геометрических объектов являются диффузными. В «цвет» окрашены только боковые стенки

комнаты. Данная сцена позволяет проверить общую корректность системы визуализации и воспроизводит такие важные эффекты, как мягкие тени от протяженных источников света и диффузные отражения («перенос цвета»).

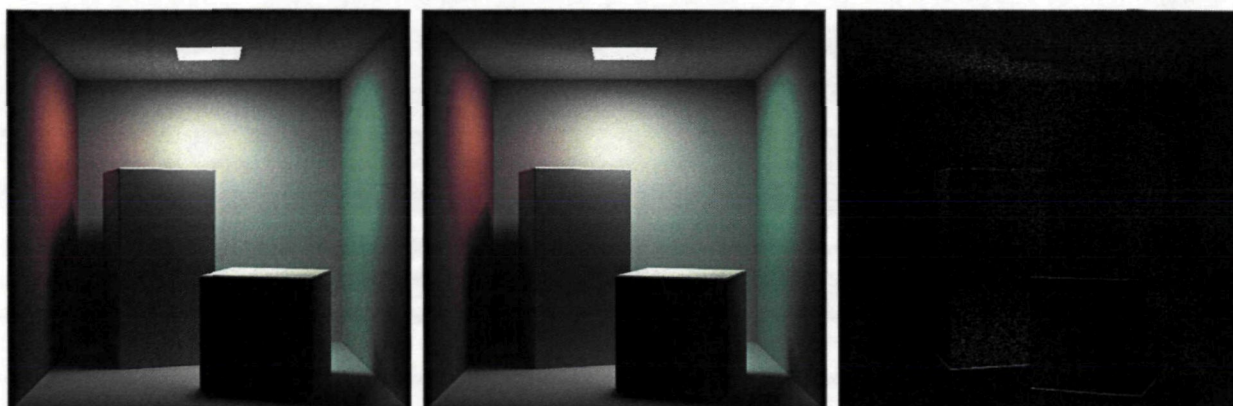


Рис. 4.2. Результат визуализации Cycles (слева, 10K SPP) и разработанной системы (центр, 2K SPP).

На изображении справа показана 10-кратно увеличенная попиксельная разница

Сходимость последовательности сгенерированных изображений к эталонному соответствует закону $N^{-1/2}$ (N – число семплов на пиксель). Иными словами, четырехкратное повышение числа семплов приводит к двукратному понижению ошибки.

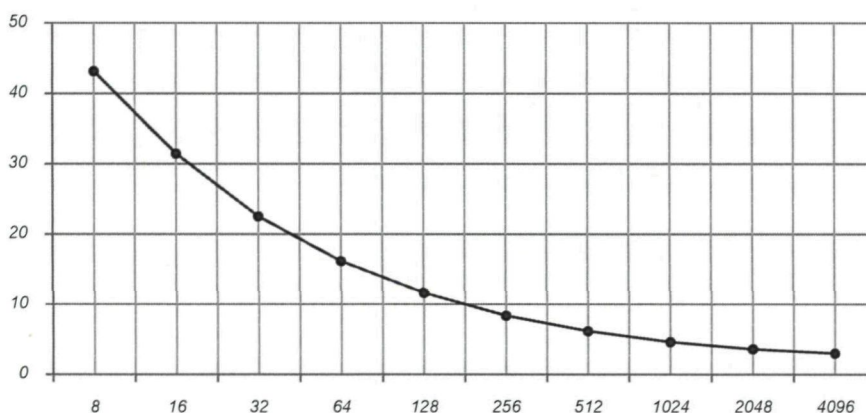


Рис. 4.3. Разница изображений (по норме L2) в зависимости от числа семплов на пиксель

Отличия от эталонного изображения (после снижения общего уровня «шума») проявляются на границах объектов, что обусловлено особенностями обработки геометрии и реализацией функций для генерации первичных и теневых лучей. Данный эксперимент показывает, что система воспроизводит изображение, идентичное эталонному.

4.2.3. Мелкомасштабный «геометрический» тест многократных диффузных отражений

Данный тест является модифицированным вариантом предыдущего и усложняет сцену мелкомасштабной геометрией. Стенки комнаты рассеивают падающий свет на небольших

неровностях и создают условия для многократных отражений. Для корректной визуализации сцены необходимо аккуратно обрабатывать геометрию и вычислять точки соударения (иначе могут возникать «утечки» света и теней).

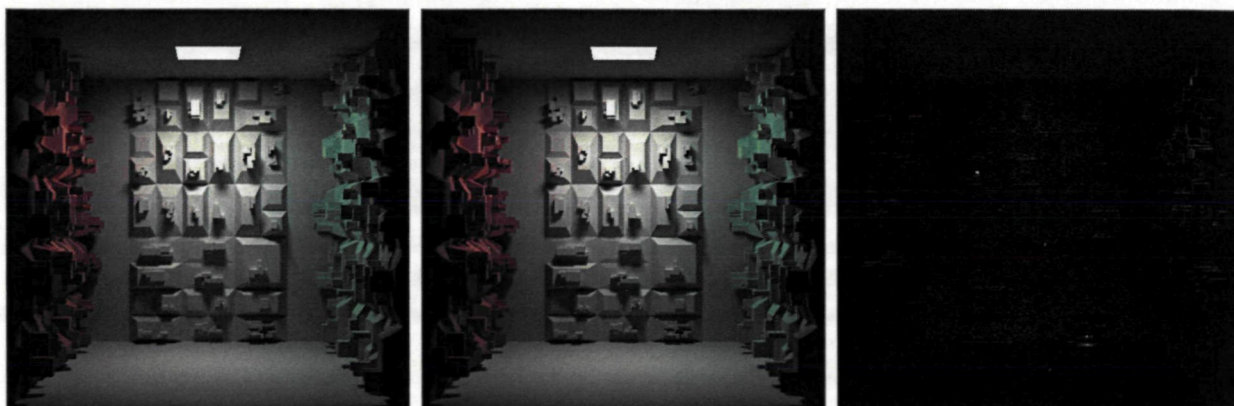


Рис. 4.4. Результат визуализации Cycles (слева, 10K SPP) и разработанной системы (центр, 2K SPP).

На изображении справа показана 10-кратно увеличенная попиксельная разница

Аналогично предыдущему тесту основные отличия от эталонного изображения проявляются на границах объектов, обилие которых снижает скорость сходимости (которая по-прежнему сохраняет характер $N^{-1/2}$).

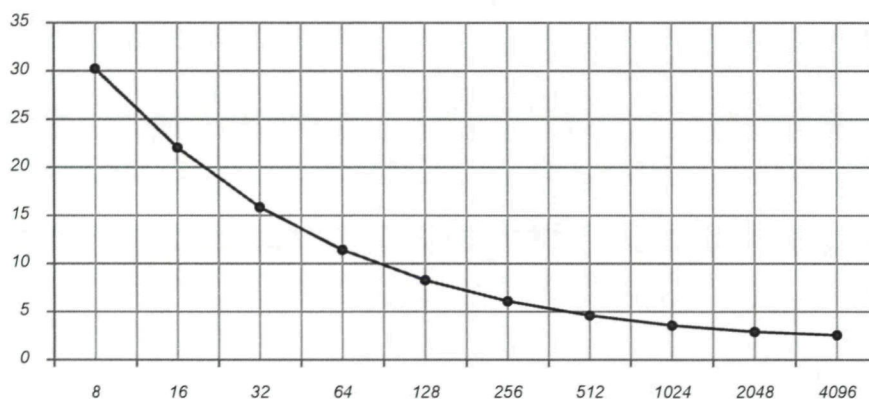


Рис. 4.5. Разница изображений (по норме L2) в зависимости от числа семплов на пиксель

Данный эксперимент показывает, что система правильно обрабатывает мелкомасштабную геометрию и воспроизводит изображение, идентичное эталонному.

4.2.4. «Зеркальный» тест на объектах сцены Cornell Box

Данный тест проверяет корректность обработки идеальных отражающих поверхностей, которые порождают каустики (на стенах и потолке комнаты). Для моделирования данных поверхностей использовалась специальная реализация «идеального» коэффициента Френеля (*FresnelIdeal*), который отражает всю падающую световую энергию. Прямое соотнесение

результатов визуализации для более сложных материалов не представляется возможным из-за специфики реализации (например, формул Френеля или микрограневых моделей).

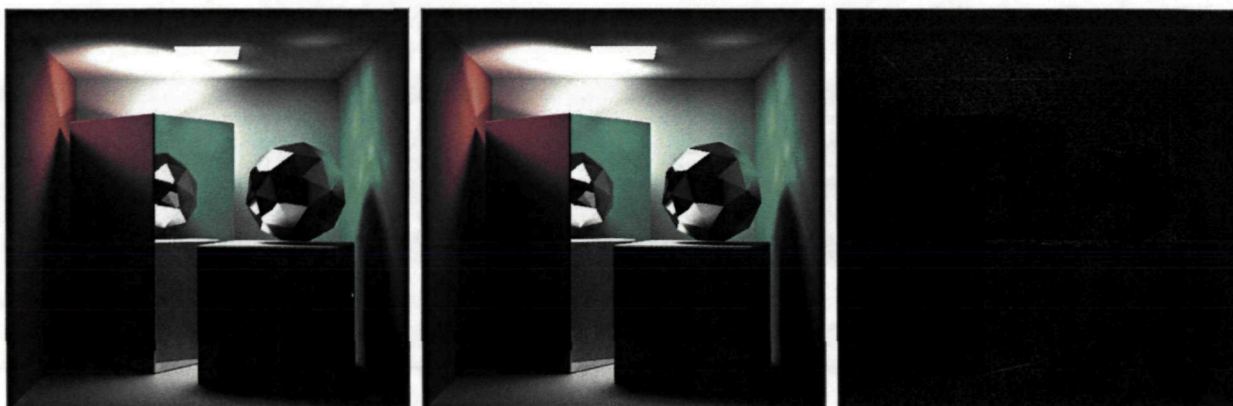


Рис. 4.6. Результат визуализации Cycles (слева, 10K SPP) и разработанной системы (центр, 2K SPP).

На изображении справа показана 10-кратно увеличенная попиксельная разница

Минимальные отличия от эталонного изображения проявляются на границах объектов, при этом скорость сходимости подчиняется закону $N^{-1/2}$.

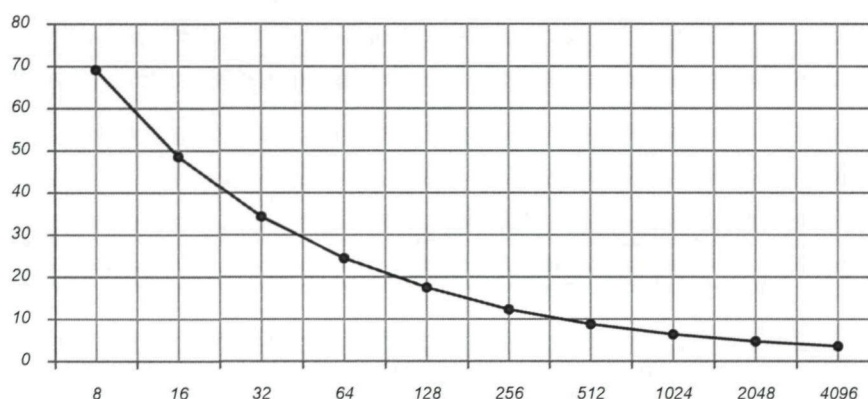


Рис. 4.7. Разница изображений (по норме L2) в зависимости от числа семплов на пиксель

Данный эксперимент показывает, что система обеспечивает корректное моделирование идеальных зеркальных поверхностей и воспроизводит изображение, идентичное эталонному.

4.3. Исследование алгоритма «усеченной» двунаправленной трассировки путей в различных условиях переноса световой энергии

Предложенный алгоритм «усеченной» двунаправленной трассировки путей повышает скорость сходимости (по сравнению с обычной однонаправленной трассировкой) в случаях, когда построение корректных путей переноса излучения от объектива виртуальной камеры затруднено. Типичными примерами служат сцены с преобладанием вторичного освещения

(прикрытые источники освещения) или резкими колебаниями светового поля (каустики). Для исследования алгоритма был построен набор 3D сцен, которые обладают перечисленными особенностями.

4.3.1. Методика исследования

Для тестового набора сцен алгоритм «усеченной» двунаправленной трассировки путей сравнивался с MIS-алгоритмом обычной трассировки путей (комбинирует вклады «явных» и «неявных» стратегий с помощью многократной выборки по значимости). Данное сравнение уместно, поскольку «усеченный» алгоритм по уровню вычислительных затрат значительно ближе к обычной трассировке путей, нежели к двунаправленной. Сравнение двух алгоритмов выполнялось согласно следующей схеме:

- Для каждой тестовой сцены алгоритмом однонаправленной трассировки путей (MIS) генерировалось эталонное изображение (размера 256×256), при этом число семплов на каждый пиксель устанавливалось равным 2×10^5 .
- Вычислялась серия изображений (размера 256×256) с различным числом семплов на пиксель с помощью однонаправленного и «усеченного» двунаправленного алгоритма. Сходимость изображений к эталонному (для каждого алгоритма) исследовалась путем попиксельного вычисления разности согласно норме L2.

Строго говоря, N семплов «усеченного» двунаправленного алгоритма по объему вычислений соответствует $2N$ семплам однонаправленного алгоритма трассировки путей. С учетом этого замечания формулируются все выводы к результатам экспериментов.

В ходе исследования следует учитывать тот факт, что «усеченный» двунаправленный алгоритм повышает сходимость именно в проблемных областях сцены (таких как каустики). При расчете «простых» областей (таких как прямая освещенность диффузной поверхности) сходимость практически не растет, поскольку эти области уже эффективно обрабатываются MIS трассировкой путей. В тестовых сценах для «усеченного» двунаправленного алгоритма не строились исключительные условия – каустики и другие эффекты вторичного освещения выглядят естественно и не занимают значительной части кадра.

4.3.2. Тест «Каустика от зеркальной поверхности»

Данный тест воспроизводит классический пример каустик, возникающих в результате отражения света от идеальной зеркальной поверхности – кольца (рис. 4.8). В этом примере

классическая сцена несколько усложнена за счет введения 3 сферических источников света (модель 3D сцены заимствована из инструментария разработчика NVIDIA OptiX).

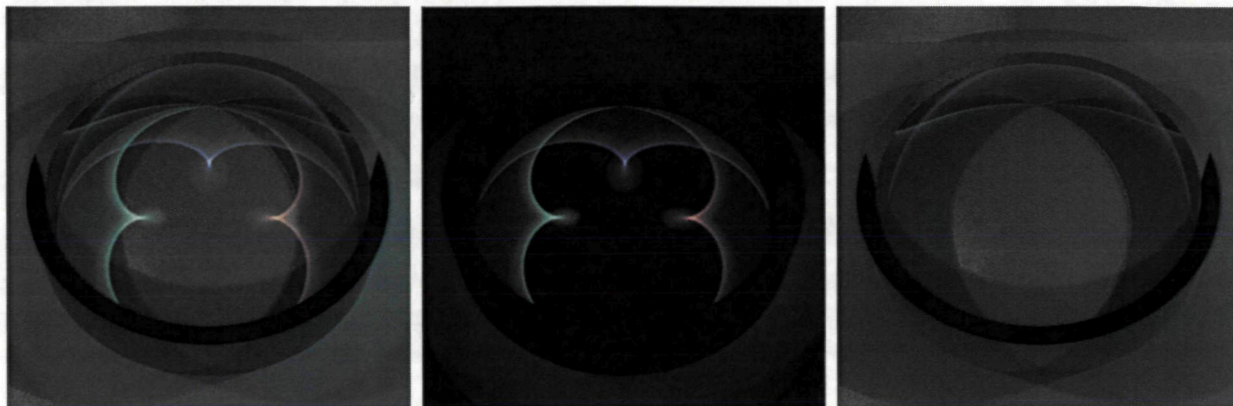


Рис. 4.8. Результаты визуализации «усеченным» двунаправленным алгоритмом (слева) и вклады прямой (центр) и обратной (справа) трассировки путей

Графики сходимости (рис. 4.9) показывают, что метод «усеченной» двунаправленной трассировки для 1024 семплов (на пиксель) обеспечивает уровень ошибки, соответствующий обычной MIS трассировке для 4096 семплов. Таким образом, на данной сцене «усеченный» двунаправленный алгоритм позволяет сократить вычислительные затраты в ~ 2 раза.

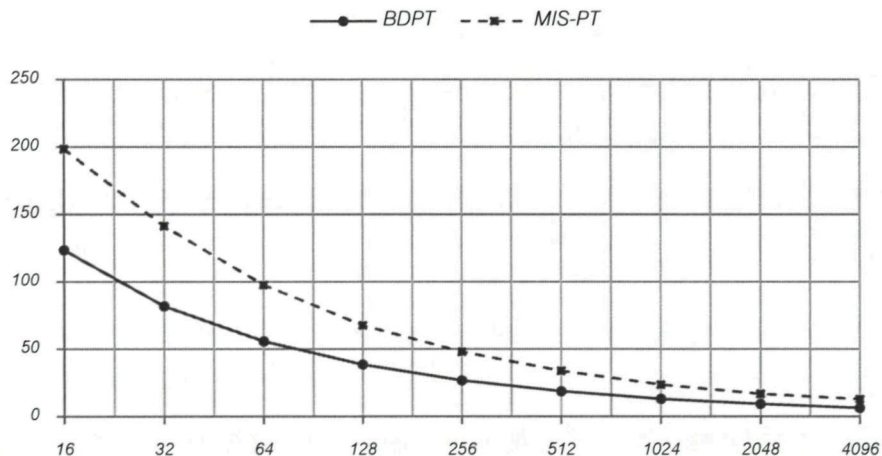


Рис. 4.9. Разница изображений (по норме L2) в зависимости от числа семплов на пиксель

Вместе с тем, при использовании автоматического контроля сходимости (описан в главе 3) прирост производительности окажется существенно большим. За счет быстрого исключения из вычислений сошедшихся пикселей (в изображении каустик) метод переходит к обработке изображения самого кольца, что приводит к многократному ускорению вычислений (до 10-ти раз). Конкретный показатель прироста производительности зависит от «агрессивности» настроек блока контроля сходимости (поэтому здесь не приводится).

Промежуточные изображения обоих методов визуализации, полученные вскоре после начала счета, показаны на *рис. 4.10*. Можно заметить, что уже для 64 семплов (на пиксель) «усеченный» двунаправленный алгоритм обеспечивает достаточно проработанную каустику.

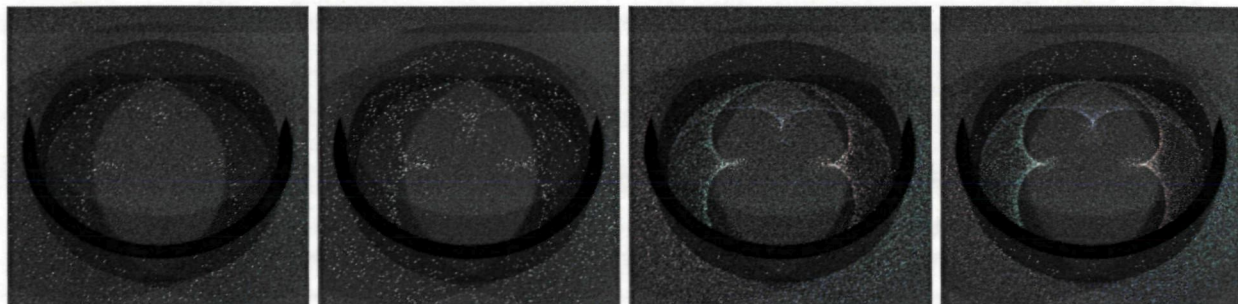


Рис. 4.10. Первые итерации счета (слева направо): PT 64 SPP, PT 128 SPP, BDPT 32 SPP, BDPT 64 SPP

Яркие точки на изображении самого кольца (“fireflies”) заслуживают отдельного обсуждения и являются основным препятствием к убыванию ошибки на графике сходимости (*рис. 4.9*). Данные точки порождаются достаточно сложными (в обработке) путями переноса излучения вида $E(S(D)?)^+SL$, которые начинаются и завершаются зеркальным отскоком. Такие пути в «усеченной» двунаправленной трассировке могут быть обработаны только *одной* стратегией из трех – «обратными» неявными путями. Остальные пути дадут нулевой вклад, поскольку зеркальная BRDF выражается δ -функцией и обнуляет вклады всех теневых лучей (которые применяются для построения явных и световых путей).

Другое важное замечание состоит в том, что и «полная» двунаправленная трассировка не решит данной проблемы: весь набор дополнительных стратегий переноса излучения даст нулевой вклад. Единственным эффективным способом обработки подобных путей являются методы на базе Metropolis Light Transport (MLT) [129]. «Усеченный» алгоритм может быть положен в основу эффективной реализации такого метода, что является целью дальнейших исследований. Среди альтернативных способов решения проблемы можно отметить технику «объединения вершин» (англ. vertex merging), которая была предложена в недавней работе [147]. Однако данный подход уже не обеспечивает *несмещенной* оценки изображения.

4.3.3. Тест «Каустика от прозрачной поверхности»

Данный тест дает пример достаточно сложных преломлений света, которые порождают каустики (*рис. 4.11*). Физически обоснованный расчет отражений и преломлений достигается за счет применения коэффициента Френеля для диэлектриков (*FresnelDielectric*). Глубина путей переноса излучения, которые формируют каустики, достигает 10 и более отскоков (с учетом полных внутренних отражений).



Рис. 4.11. Результаты визуализации «усеченным» двунаправленным алгоритмом (слева) и вклады прямой (центр) и обратной (справа) трассировки путей

Графики сходимости (рис. 4.12) показывают, что метод «усеченной» двунаправленной трассировки для 1024 семплов на пиксель обеспечивает уровень ошибки, соответствующей обычной MIS трассировке для 4096 семплов.

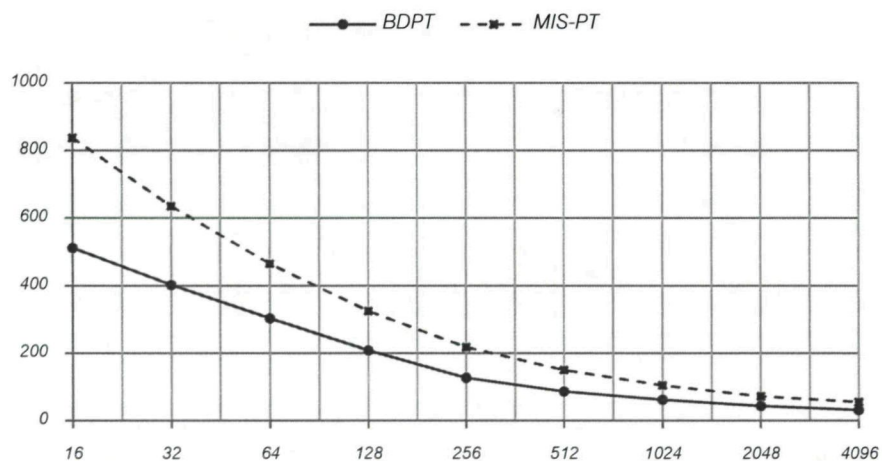


Рис. 4.12. Разница изображений (по норме L2) в зависимости от числа семплов на пиксель

Таким образом, в данной сцене «усеченный» двунаправленный алгоритм позволяет понизить вычислительные затраты в ~2 раза (и более при использовании контроля сходимости).

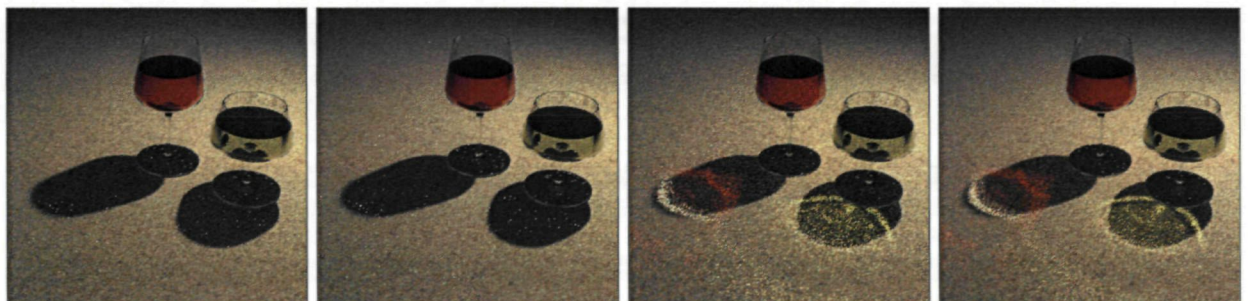


Рис. 4.13. Первые итерации счета (слева направо): PT 64 SPP, PT 128 SPP, BDPT 32 SPP, BDPT 64 SPP

Промежуточные изображения обоих методов визуализации, полученные вскоре после начала вычислений, показаны на *рис. 4.13*. Нетрудно видеть, что двунаправленный алгоритм быстро обеспечивает достаточно проработанное изображение каустики. Как и в предыдущем тесте, основным фактором «ошибки» на графике сходимости (*рис. 4.12*) являются пути вида $E(S(D)?)^+SL$, за счет которых формируется изображение незатененных частей оснований бокалов (на *рис. 4.13* данные области еще не проработаны). Очевидно, что «полноценная» двунаправленная трассировка здесь также не решит проблемы.

4.3.4. Тест «Вторичное освещение через диффузное отражение»

Данный тест моделирует ситуацию, когда изображение сцены формируется в основном за счет вторичного освещения. Все поверхности сцены являются диффузными (*рис. 4.14*).

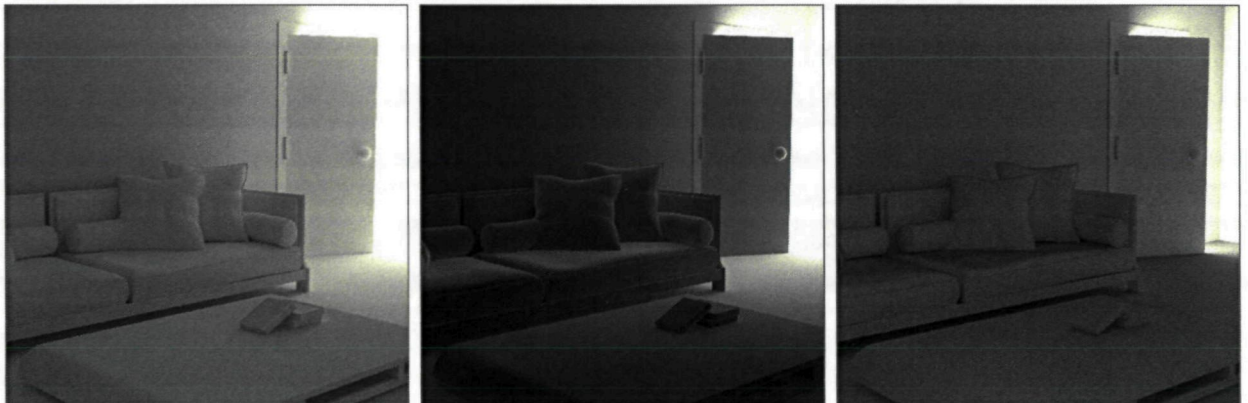


Рис. 4.14. Результаты визуализации «усеченным» двунаправленным алгоритмом (*слева*) и вклады прямой (*центр*) и обратной (*справа*) трассировки путей

Площадной источник света расположен за прикрытой дверью, что создает примерно равные условия для «прямых» и «обратных» путей переноса излучения.

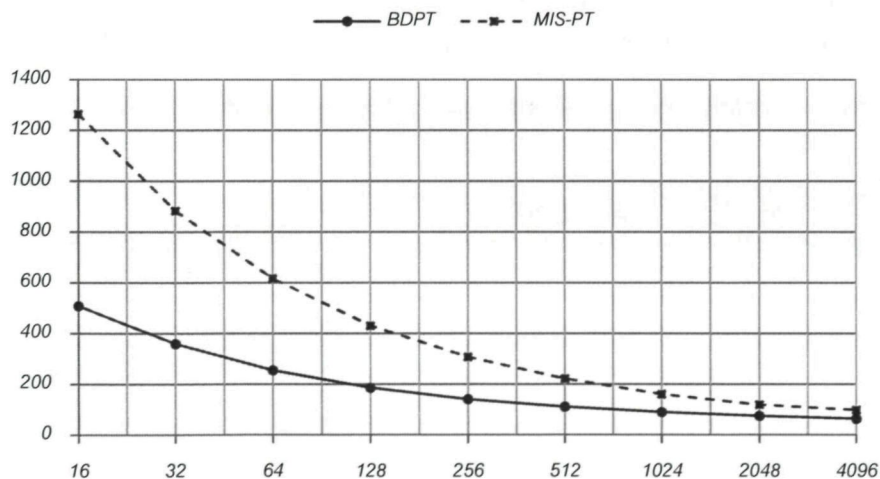


Рис. 4.15. Разница изображений (по норме L2) в зависимости от числа семплов на пиксель

Графики сходимости (рис. 4.15) показывают, что метод «усеченной» двунаправленной трассировки для 512 семплов (на пиксель) обеспечивает уровень ошибки, приблизительно соответствующий обычной MIS трассировке для 4096 семплов. В итоге, на данной тестовой сцене «усеченный» двунаправленный алгоритм обеспечивает сокращение вычислительных затрат в 3–4 раза.

Промежуточные изображения обоих методов визуализации, полученные вскоре после начала вычислений, показаны на рис. 4.16. Нетрудно видеть, что двунаправленный алгоритм довольно быстро достигает проработанного изображения сцены (с учетом сложных условий освещения).

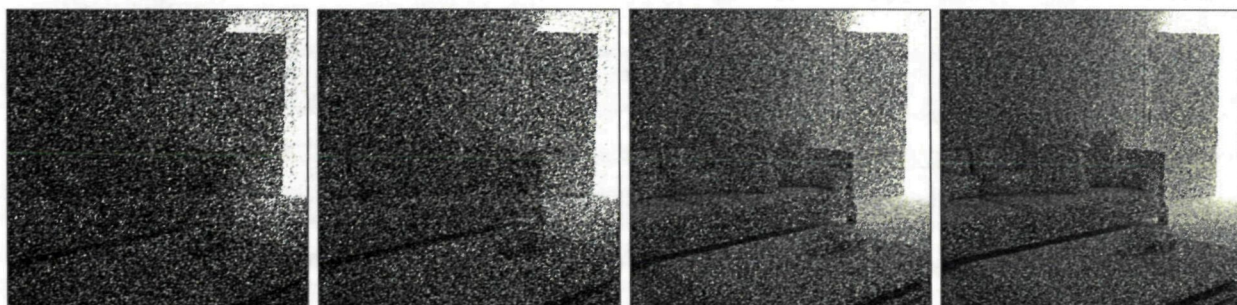


Рис. 4.16. Первые итерации счета (слева направо): PT 256 SPP, PT 512 SPP, BDPT 128 SPP, BDPT 256 SPP

Поскольку все поверхности являются диффузными, «полноценный» двунаправленный метод позволит существенно уменьшить дисперсию оценки (однако каждый семпл будет требовать значительно больше вычислений).

4.3.5. Тест «Вторичное освещение через прозрачную поверхность»

В данной тестовой сцене изображение формируется исключительно за счет вторичного освещения (рис. 4.17) – сферические источники света перекрыты прозрачными стенками.

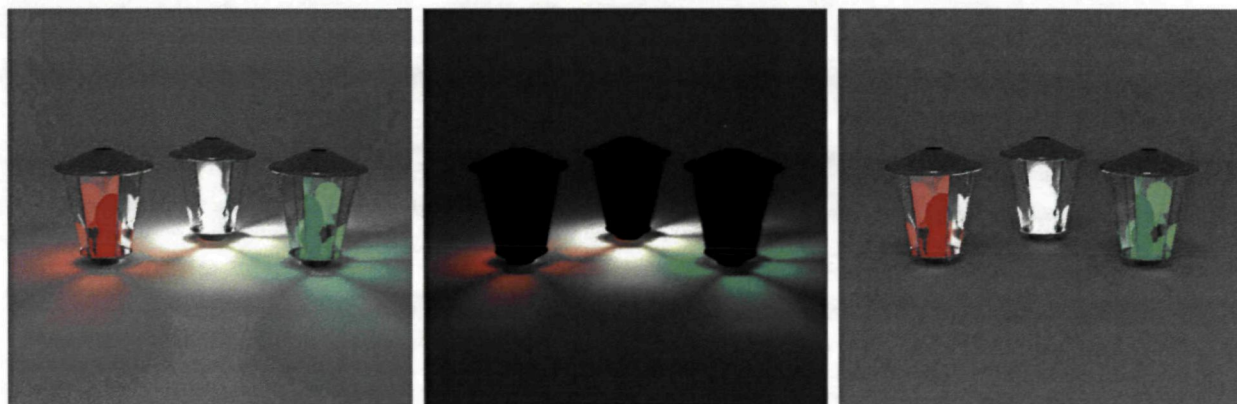


Рис. 4.17. Результаты визуализации «усеченным» двунаправленным алгоритмом (слева) и вклады прямой (центр) и обратной (справа) трассировки путей

Отражение и преломление света на стенках плафона физически достоверно моделируется с помощью коэффициентов Френеля для диэлектриков (*FresnelDielectric*).

Графики сходимости (рис. 4.18) показывают, что метод «усеченной» двунаправленной трассировки для 512 семплов обеспечивает уровень ошибки, соответствующей обычной MIS трассировке для 4096 семплов. Таким образом, в данной сцене «усеченный» метод позволяет сократить вычислительные затраты в 3–4 раза.

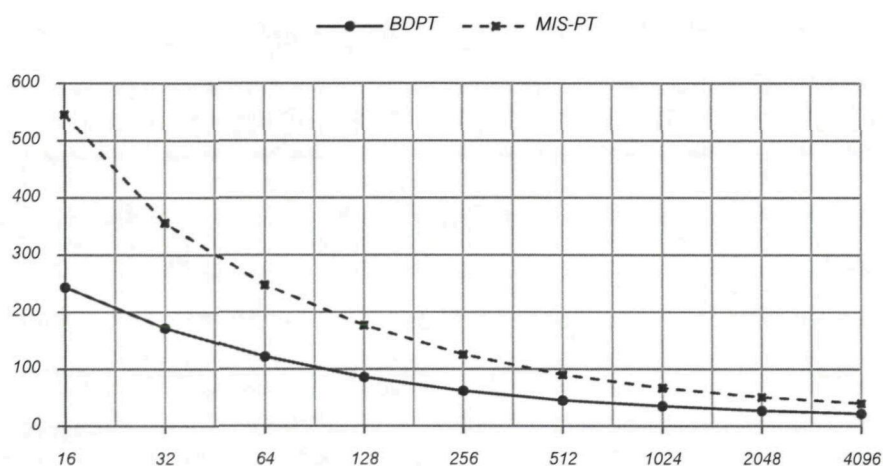


Рис. 4.18. Разница изображений (по норме L2) в зависимости от числа семплов на пиксель

Промежуточные изображения двух методов визуализации, полученные вскоре после начала счета, показаны на рис. 4.19. Уже для 128 семплов (на пиксель) двунаправленный алгоритм обеспечивает достаточно проработанную картину освещения от напольных светильников.

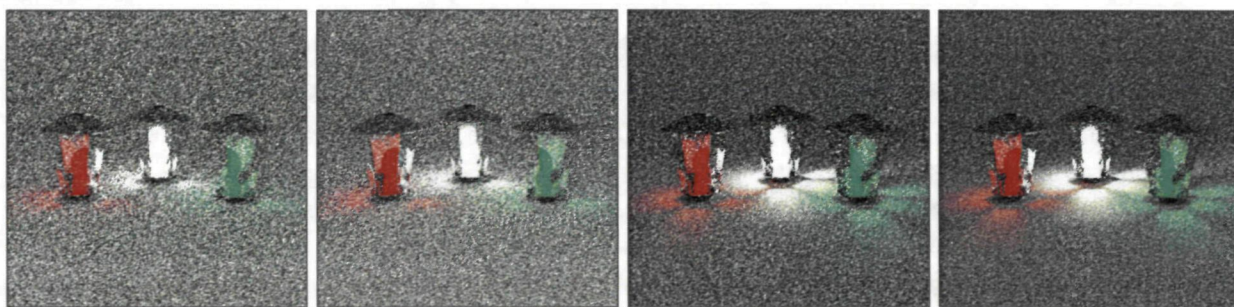


Рис. 4.19. Первые итерации счета (слева направо): PT 128 SPP, PT 256 SPP, BDPT 64 SPP, BDPT 128 SPP

Для этой сцены также характерны пути переноса излучения вида $E(S(D))^+SL$, посредством которых формируются отражения источников света на стеклянных стенках плафонов (в силу чего замедляется сходимость соответствующих фрагментов изображения). Подобно первым двум тестам, «полноценная» двунаправленная трассировка здесь не сможет решить проблему — необходимо использовать методы на основе Metropolis Light Transport (MLT).

4.4. Исследование производительности программного комплекса

4.4.1. Производительность построения ускоряющей структуры

Быстродействие разработанного алгоритма построения иерархии объемов замерялось на тестовых сценах различной сложности (табл. 4.1). На момент подготовки публикации [162] наилучшая ячеечная SAH реализация была описана в работе [102], скорость работы которой измерялась на графическом процессоре NVIDIA GeForce GTX 285 1 Гб. Наряду с этим были доступны результаты исследования схожего ячеечного алгоритма на прототипе процессора Intel MIC [137], который содержит 32 полноценных x86 ядра с частотой 1 ГГц. Поскольку (возможные) продукты на базе Intel MIC должны были конкурировать с производительными графическими процессорами, сравнение результатов с указанной реализацией представляет интерес. Тестирование алгоритма построения выполнялось на ускорителе NVIDIA GeForce GTX 480 1.5 Гб под операционной системой Linux (версия видеодрайвера 270.41.06). Данный графический процессор построен на микроархитектуре Fermi и имеет 480 вычислительных (CUDA) ядер с частотой 1.4 ГГц. По сравнению с графическим ускорителем предыдущего поколения NVIDIA GeForce GTX 285 производительность вычислительных блоков возросла примерно в 2 раза, а пропускная способность памяти – на 25%.

Таблица 4.1. Время построения ускоряющей структуры в сравнении с другими ячеечными SAH реализациями (на момент публикации результатов)

Тестовая сцена	Число примитивов	GPU реализация <i>Lauterbach</i> [102]	Intel MIC реализация <i>Wald</i> [137]	Предложенная реализация [162]
<i>Toasters</i>	11K	Недоступно	11 мс	13 мс
<i>Bunny</i>	69K	Недоступно	Недоступно	20 мс
<i>Cloth</i>	92K	Недоступно	19 мс	19 мс
<i>Fairy Forest</i>	174K	488 мс	31 мс	40 мс
<i>Dragon/Bunny</i>	252K	403 мс	43 мс	49 мс
<i>Welsh-dragon</i>	2.2M	Недоступно	Недоступно	362 мс
<i>Cathedral</i>	3.2M	Недоступно	Недоступно	697 мс

Результаты экспериментов показывают, что предложенный алгоритм построения иерархии объемов обеспечивает до 10 раз большую производительность по сравнению с реализацией

[102] (с учетом разницы в оборудовании можно говорить о 5-кратном приросте). Кроме того, производительность построения находится на уровне 32-х ядерного процессора Intel MIC (даже с учетом использования устаревшей карты NVIDIA GeForce GTX 480). В абсолютном выражении скорость построения достигает 6М треугольников в секунду (для «аккуратных» SAN деревьев).

4.4.2. Производительность алгоритмов визуализации

Быстродействие разработанных алгоритмов визуализации исследовалось на популярных сценах для тестирования методов глобального освещения (рис. 4.20). Большинство моделей содержат различные типы материалов, а также диффузные и зеркальные текстурные карты высокого разрешения. Все изображения генерировались в разрешении 1024×768 методом стохастической трассировки путей (является основой и для «усеченного» двунаправленного алгоритма). Для каждой тестовой сцены замерялось время трассировки первичных, теневых и вторичных лучей, а также общее время формирования промежуточного кадра (1 семпл на пиксель). В последнем случае «глубина» семпла определялась методом «русской рулетки», который запускался с первого соударения.

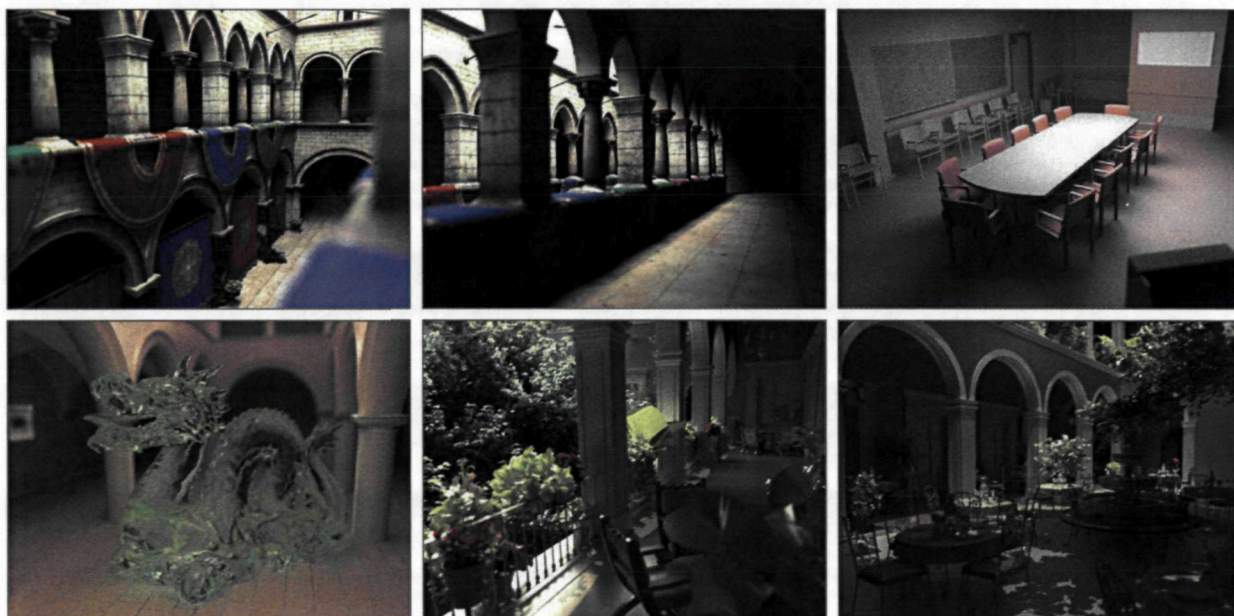


Рис. 4.20. Тестовые сцены и ракурсы камеры (слева направо): *Crytek Sponza (Outside)*, *Crytek Sponza (Inside)*, *Conference*, *Sponza + Dragon*, *San-Miguel (Upper Floor)*, *San-Miguel (Ground Floor)*

Скорость обработки тестовых сцен измерялась на графическом процессоре NVIDIA GeForce GTX 580 3 Гб под операционной системой Linux. Данный графический процессор построен на микроархитектуре Fermi и имеет 512 вычислительных (CUDA) ядер с частотой 1.54 ГГц.

Таблица 4.2. Результаты измерения производительности на тестовых сценах

Тестовая сцена	Число примитивов	Время трассировки (мс)			Миллионов лучей в секунду		Число обработанных вторичных лучей (после 1 отскока)
		Первичные лучи	Теневые лучи	Вторичные лучи	Первичные лучи	Вторичные лучи	
<i>Sponza + Dragon</i>	143K	99	260	319	794	24.6	785963
<i>Crytek Sponza (Outside)</i>	262K	126	184	338	624	22.7	769634
<i>Crytek Sponza (Inside)</i>	262K	77	123	24.9	1021	316	786432
<i>Conference</i>	283K	64	4.9	155	1228	412	639756
<i>San-Miguel (Ground Floor)</i>	10.5M	273	30.3	65.9	288	119	785967
<i>San-Miguel (Upper Floor)</i>	10.5M	15.8	23.2	51.8	497	15.1	783896

Полученные экспериментальные данные (табл. 4.2) позволяют сделать вывод о достаточно высокой производительности трассировки лучей. На сложной сцене San Miguel, содержащей свыше 10М треугольников, скорость обработки достигает соответственно 50 и 15 миллионов первичных и (некогерентных) вторичных лучей в секунду. Данного уровня быстродействия достаточно для интерактивного моделирования глобального освещения: система генерирует промежуточные изображения с частотой 2.5–6 кадров в секунду (табл. 4.3).

Таблица 4.3. Время обработки одного промежуточного изображения (1 семпл на пиксель)

Тестовая сцена	Число примитивов	Время итерации (мс)	Тестовая сцена	Число примитивов	Время итерации (мс)
<i>Sponza + Dragon</i>	143K	2616	<i>Conference</i>	283K	2206
<i>Crytek Sponza (Outside)</i>	262K	166.4	<i>San-Miguel (Ground Floor)</i>	10.5M	3712
<i>Crytek Sponza (Inside)</i>	262K	143.9	<i>San-Miguel (Upper Floor)</i>	10.5M	410.9

Производительность может быть существенно повышена за счет сокращения максимальной длины путей до 2–3 отскоков с сохранением большинства эффектов глобального освещения (однако в этом случае уже нельзя гарантировать несмещенную оценку изображения).

Распределение времени генерации одного промежуточного кадра (1 семпл на пиксель) между различными задачами показано на следующей диаграмме (рис. 4.21). Основная масса вычислительных затрат связана с трассировкой различного типа лучей, однако существенная доля расчетов уходит и на другие операции (семплирование и обработка источников света и материалов поверхностей).

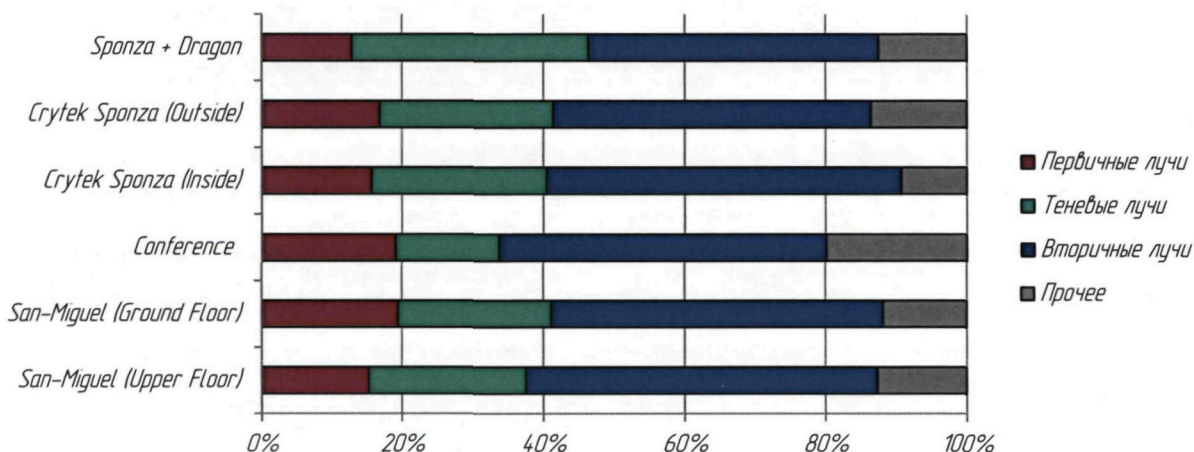


Рис. 4.21. Распределение времени генерации одного кадра (1 семпл на пиксель)

В определенном смысле данная диаграмма позволяет оценить накладные расходы, связанные с обеспечением универсальных моделей геометрических объектов, материалов и источников света. Поскольку основные вычислительные затраты уходят на трассировку, данные расходы следует признать приемлемыми (оправдывающими возможности системы).

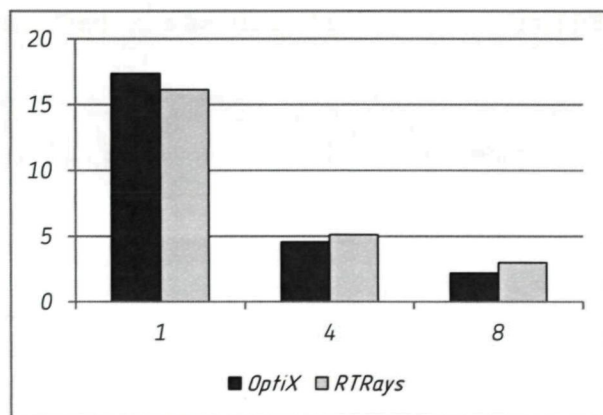
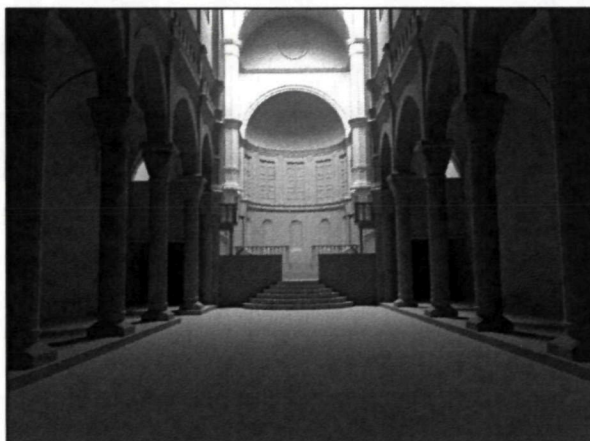
4.4.3. Сравнение производительности с аналогами: NVIDIA OptiX

В широком смысле аналогом разработанной системы является любой инструментарий для высокопроизводительной трассировки лучей. К настоящему моменту доступно широкое разнообразие подобных систем для традиционных вычислительных архитектур. В частности, следует отметить популярный образовательный проект PBRT [144], который стал основой для целого семейства систем физически корректной визуализации, включая LuxRender [127] и Mitsuba Renderer [148]. К высокопроизводительным решениям можно отнести библиотеку трассировки лучей Intel Embree [149]. Для графических архитектур относительно известным инструментарием является NVIDIA OptiX [118], который позиционируется как программный движок для ускорения базовых операций трассировки лучей. OptiX не реализует конкретных алгоритмов визуализации и позволяет программисту задавать свои программы для генерации лучей, пересечения с примитивами или расчета освещенности (на языке CUDA C). Наряду с разработкой систем визуализации OptiX может применяться в задачах, которые задействуют

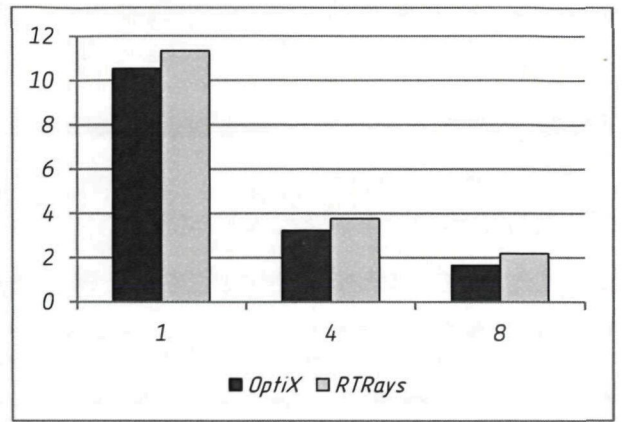
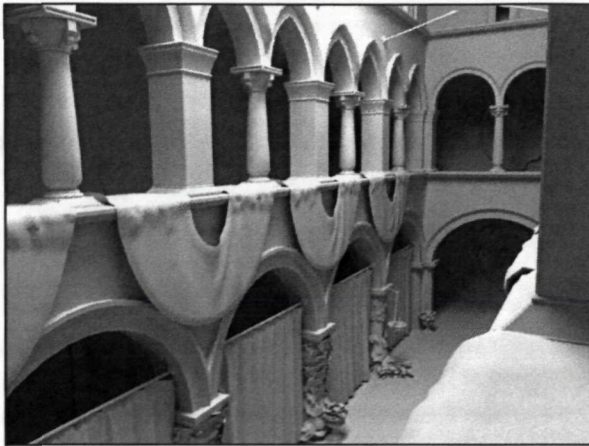
трассировку лучей как универсальный геометрический метод моделирования: оптический и акустический дизайн, радиационные исследования, анализ соударений (физические движки) и другие приложения. Таким образом, библиотека OptiX является низкоуровневым аналогом разработанной системы. Среди других инструментариев можно отметить проект с открытым кодом LuxRays [127], который разработан на OpenCL и обеспечивает ускорение поиска точек пересечения (все остальные стадии необходимо реализовывать на центральном процессоре). В рамках настоящего исследования быстродействие разработанной системы целесообразно сопоставлять с программным движком OptiX.

Для сравнения использовалась наиболее актуальная (на момент подготовки материала) версия OptiX 3.0. На основе примера *Path Tracing* (входит в OptiX SDK) была разработана базовая реализация стохастической трассировки путей для произвольных сцен, загружаемых из стандартного формата (OBJ). Вместо реализации универсальной подсистемы материалов все поверхности сцены обрабатывались как диффузные. Для освещения компьютерных сцен были портированы (прямоугольные) протяженные источники, «верхний свет» (бесконечная сфера с постоянным излучением) и направленные источники. Данный функционал позволяет воспроизвести некоторые примеры расчета глобального освещения и является достаточным для целей эксперимента. Быстродействие измерялось для различной глубины генерируемых путей (метод «русской рулетки» отключался) – исследовались случаи 1, 4 и 8 «отскоков». В точках соударения выполнялось *явное* соединение с источником, поэтому соответствующее число обрабатываемых лучей равно 4, 10 и 18. В тестовых сценах глубина пути может быть уменьшена только за счет выхода за пределы сцены. Эксперимент проводился на процессоре NVIDIA GeForce GTX 680 4 Гб под управлением операционной системой Ubuntu Linux 12. Результаты эксперимента показаны на следующих диаграммах (рис. 4.22).

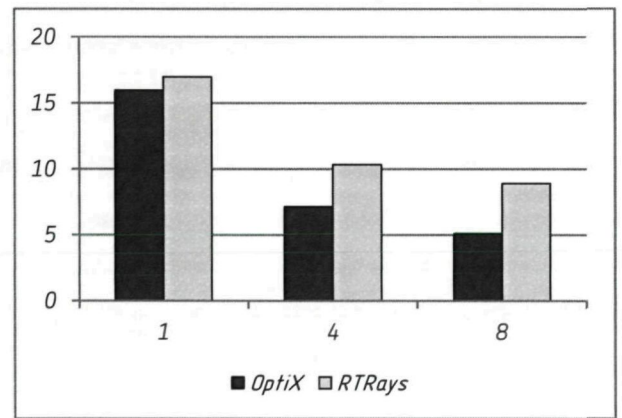
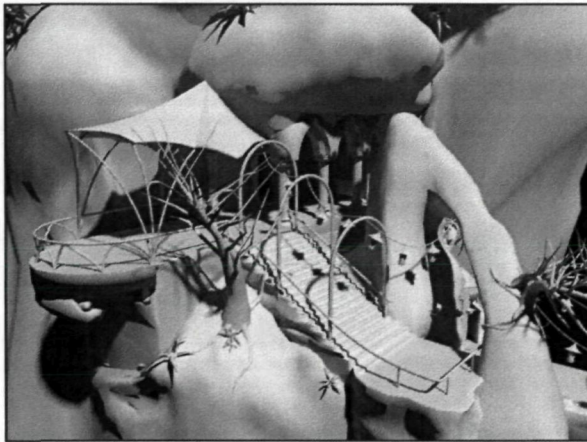
Sibenik Cathedral
75K Triangles



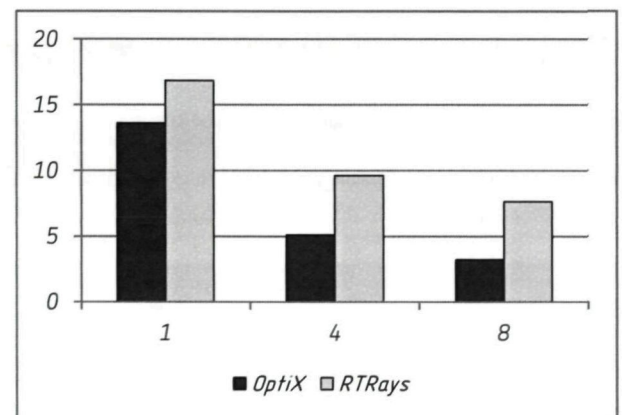
Crytek Sponza
262K Triangles



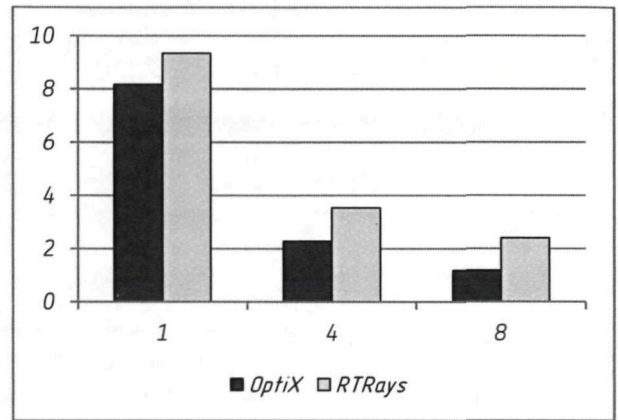
Outbound
1.1M Triangles



City
3.3M Triangles



San Miguel (Upper Floor)
10.5M Triangles



San Miguel (Ground Floor)
10.5M Triangles

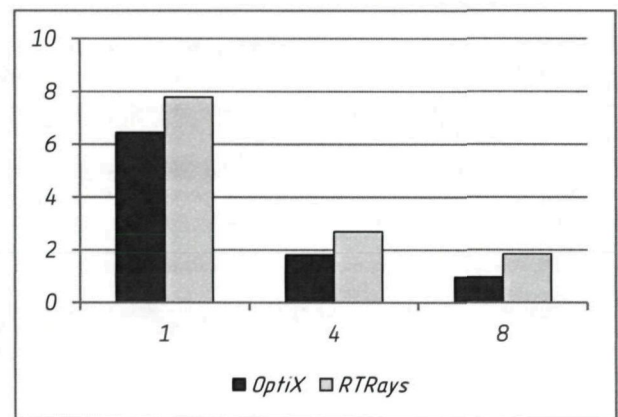


Рис. 4.22. Результаты замера производительности (миллионы семплов в секунду) для движка *OptiX* и разработанной системы *RTRays*

Результаты эксперимента позволяют сделать вывод, что производительность разработанного комплекса не уступает инструментарию NVIDIA OptiX, а с ростом «глубины» трассировки опережает его (до 2-х раз). Вероятно, основная причина такого результата связана с тем, что конвейер трассировки лучей OptiX реализован целиком в одном CUDA ядре. В результате по мере расхождения (или остановки) вторичных лучей значительно падает их когерентность в «свертках» потоках, что приводит к низкой утилизации SIMT архитектуры ГПУ.

4.5. Выводы к главе 4

1. В результате настоящего исследования разработаны высокоуровневые программные библиотеки и комплекс, который выполняет интерактивный синтез изображений 3D сцен методами глобального освещения на графических процессорах и реализует все выносимые на защиту положения.
2. Экспериментально показана корректность моделирования глобального освещения для различных условий переноса световой энергии в сцене путем сравнения результатов моделирования с эталонными изображениями.
3. Производительность ускоряющей структуры и программного комплекса в целом для сцен размером более 100К треугольников в эквивалентных условиях во всех случаях не ниже возможных аналогов (таких как NVIDIA OptiX).

Задачи исследования, поставленные во введении, выполнены полностью.

Заключение

В результате настоящего исследования разработаны теоретические основы создания программных систем виртуальной реальности в части интерактивного синтеза изображений 3D сцен методами глобального освещения на графических процессорах, в том числе:

- 1) Эффективная для статических и динамических сцен ускоряющая структура, на основе иерархии объемов (SAH BVH), отличающаяся высокопроизводительным построением, реализованным полностью на графическом процессоре.
- 2) Структура хранения и обработки 3D сцен, отличающаяся наличием двухуровневого представления модели: на уровне прикладного программиста функционирует как граф сцены, а на нижнем уровне обеспечивает эффективное преобразование (сериализацию) данных для потоковой обработки на графическом процессоре.
- 3) Расширяемая подсистема источников света и материалов, отличающаяся построением на основе концепции «Монте-Карло атомарных» (событийно-атомарных) материалов, которая обеспечивает компактное описание и эффективную обработку на графических процессорах.
- 4) Программный конвейер трассировки лучей, который реализует стандартные операции (блоки) лучевых алгоритмов визуализации и отличается универсальностью на классе методов глобального освещения и ориентированностью на массивно-параллельные вычислительные архитектуры. На базе блоков данного конвейера построены методы: испускание лучей, трассировка лучей Уиттеда, стохастическая трассировка путей в прямом и обратном направлении, двунаправленная трассировка путей и предлагаемый в настоящем исследовании ее «усеченный» вариант.
- 5) Метод «усеченной» двунаправленной трассировки путей, который строится на типовых операциях конвейера трассировки лучей и отличается фиксированным объемом потребляемой памяти при обработке путей любой длины, эффективной реализацией многократной выборки по значимости, полным исключением фазы соединения путей.
- 6) Высокоуровневая библиотека для интерактивного синтеза изображений компьютерных сцен на графических процессорах методами глобального освещения и программный комплекс, который реализует все выносимые на защиту положения и обеспечивает производительность на уровне (а при увеличении глубины трассировки и до 2-х раз выше) библиотеки трассировки лучей NVIDIA OptiX.

Литература

- [1] Whitted T. An improved illumination model for shaded display // *Communications of the ACM*. – 1980. – June. – Vol. 23, no. 6. – Pp. 343–349.
- [2] Glassner A.S. *An Introduction to Ray Tracing*. – Academic Press, London, 1989.
- [3] Cook R.L., Porter T., Carpenter L. Distributed Ray Tracing // *Computer Graphics (Proceedings of SIGGRAPH '84)*. – 1984. – July. – Vol. 18, no. 3. – Pp. 137–145.
- [4] Kajiya J.T. The Rendering Equation // *Computer Graphics (Proceedings of SIGGRAPH '86)*. – 1986. – August. – Vol. 20, no. 4. – Pp. 143–150.
- [5] Chen S., Rushmeier H.E., Miller G., Turner D. A Progressive Multi-Pass Method for Global Illumination // *Computer Graphics (Proceedings of SIGGRAPH '91)*. – 1991. – July. – Vol. 25, no. 4. – Pp. 165–174.
- [6] Lafortune E.P., Willems Y.D. Bi-Directional Path Tracing // *Proceedings of Compugraphics '93*. – Alvor, Portugal, 1993. – December. – Pp. 145–153.
- [7] Jensen H.W. Global Illumination using Photon Maps // *Eurographics Rendering Workshop*. – 1996. – June. – Pp. 21–30.
- [8] Jensen H.W. *Realistic Image Synthesis using Photon Mapping*. – A K Peters, 2001. – ISBN 156881-147-0.
- [9] Goral C., Torrance K., Greenberg D., Battaile B. Modeling the Interaction of Light between Diffuse use Surfaces // *Computer Graphics (Proceedings of SIGGRAPH '84)*. – 1984. – July. – Vol. 18, no. 3. – Pp. 213–222.
- [10] Nishita T., Nakamae E. Continuous Tone Representation of Three-Dimensional Objects Taking Account of Shadows and Interreflection // *Computer Graphics (Proceedings of SIGGRAPH '85)*. – 1985. – July. – Vol. 19, no. 3. – Pp. 23–30.
- [11] Fujimoto A., Tanaka T., Iwata K. ARTS: Accelerated Ray-Tracing System // *IEEE Computer Graphics & Applications*. – 1986. – April. – Pp. 16–26.
- [12] Cosenza B. A Survey on Exploiting Grids for Ray Tracing // *Eurographics Italian Chapter Conference*. – 2008. – Pp. 89–96.
- [13] Glassner A.S. Space Subdivision For Fast Ray Tracing // *IEEE Computer Graphics & Applications*. – 1984. – October. – Vol. 4, no. 10. – Pp. 15–22.

- [14] Bentley J.L. Multidimensional binary search trees used for associative searching // Communications of the ACM. – 1975. – September. – Vol. 18, no. 9. – Pp. 509–517.
- [15] Havran V. About the Relation between Spatial Subdivisions and Object Hierarchies Used in Ray Tracing // Proceedings of conference SCCG 2007, Budmerice, Slovakia. – 2007. – April. – Pp. 55–60.
- [16] Rubin S.M., Whitted J.T. A 3-Dimensional Representation for Fast Rendering of Complex Scenes // Computer Graphics (Proceedings of SIGGRAPH '80). – 1980. – July. – Vol. 14, no. 3. – Pp. 110–116.
- [17] Ooi B.C., Sacks-Davis R., McDonell K.J. Spatial k-dtree: An indexing mechanism for spatial databases // Proceedings of the IEEE COMPSAC Conference. – 1987. – Pp. 433–438.
- [18] Zachmann G. Minimal Hierarchical Collision Detection // Proceedings of ACM Symposium on Virtual Reality Software and Technology (VRST), Hong Kong, China. – 2002. – November. – Pp. 121–128.
- [19] Havran V., Prikryl J., Purgathofer W. Statistical Comparison of Ray-Shooting Efficiency Schemes: Tech. rep. / Institute of Computer Graphics / Vienna University of Technology. – 2000. – TR-186-2-00-14.
- [20] Reinhard E., Smits B., Hansen C. Dynamic Acceleration Structures for Interactive Ray Tracing // Rendering Techniques 2000: 11th Eurographics Workshop on Rendering. – 2000. – June. – Pp. 299–306.
- [21] Wald I., Mark W.R., Gunther J., Boulos S., Ize T., Hunt W., Parker S.G., Shirley P. State of the Art in Ray Tracing Animated Scenes // Computer Graphics Forum. – 2009. – Vol. 28, no. 6. – Pp. 1691–1722.
- [22] Chalmers A., Davis T., Reinhard E., editors. Practical Parallel Rendering. – A K Peters, 2002. – ISBN 156881-179-9.
- [23] Parker S., Parker M., Livnat Y., Sloan P., Hansen C., Shirley P. Interactive Ray Tracing for Volume Visualization // IEEE Transactions on Visualization and Computer Graphics. – 1999. – July. – Vol. 5, no. 3. – Pp. 238–250.
- [24] Muuss M.J. Towards Real-Time Ray-Tracing of Combinatorial Solid Geometric Models // Proceedings of BRL-CAD Symposium. – 1995. – June.
- [25] Parker S., Martin W., Sloan P., Shirley P., Smits B., Hansen C. Interactive Ray Tracing // Proceedings of the 1999 Symposium on Interactive 3D graphics. – 1999. – April. – Pp. 119–126.

- [26] Wald I., Kollig T., Benthin C., Keller A., Slusallek P. Interactive Global Illumination using Fast Ray Tracing // Proceedings of the 13th Eurographics workshop on Rendering. – 2002. – Pp. 15–24.
- [27] Wald I., Benthin C., Dietrich A., Slusallek P. Interactive Ray Tracing on Commodity PC Clusters State of the Art and Practical Applications // Lecture Notes in Computer Science. – 2003. – Vol. 2790. – Pp. 499–508.
- [28] DeMarle D.E., Parker S.G., Hartner M., Gribble C., Hansen C.D. Distributed Interactive Ray Tracing for Large Volume Visualization // Proceedings of the 2003 IEEE Symposium on Parallel and Large-Data Visualization and Graphics. – 2003. – Pp. 87–94.
- [29] DeMarle D.E., Gribble C., Boulos S., Parker S. Memory sharing for interactive ray tracing on clusters // Parallel Computing. – 2005. – February. – Vol. 31, no. 2. – Pp. 221–242.
- [30] Carter J.B., Khandekar D., Kamb L. Distributed shared memory: Where we are and where we should be headed // Proceedings of the Fifth Workshop on Hot Topics in Operating Systems. – 1995. – Pp. 119–122.
- [31] Bershad B.N., Zekauskas M.J. Shared memory parallel programming with entry consistency for distributed memory multiprocessors: Tech. rep. / Carnegie Mellon University, Pittsburgh. – 1991. – CMU-CS-91-170.
- [32] Bigler J., Stephens A., Parker S.G. Design for Parallel Interactive Ray Tracing Systems // Proceedings of the IEEE Symposium on Interactive Ray Tracing. – 2006. – Pp. 187–196.
- [33] Manta Open Source Interactive Ray Tracer. URL:
http://mantawiki.sci.utah.edu/manta/index.php/Main_Page
- [34] Ize T., Brownlee C., Hansen C.D. Real-Time Ray Tracer for Visualizing Massive Models on a Cluster // Proceedings of the 2011 Eurographics Symposium on Parallel Graphics and Visualization. – 2011. – Pp. 61–69.
- [35] Nishimura H., Ohno H., Kawata T., Shirakawa I., Omura K. Links-1 - a parallel pipelined multimicrocomputer system for image creation // Proceedings of the 10th annual international symposium on Computer architecture. – 1983. – June. – Vol. 11, no. 3. – Pp. 387–394.
- [36] Gunther T., Poliwoda C., Reinhart C., Hesser J., Manner R., Meinzer H.-P., Baur H.-J. VIRIM: A massively parallel processor for real-time volume visualization in medicine // Computers & Graphics. – 1995. – Vol. 19, no. 5. – Pp. 705–710.

- [37] Knittel G., Straer W. VIZARD: Visualization Accelerator for Realtime Display // Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware. – 1997. – August. – Pp. 139–147.
- [38] Pster H., Hardenbergh J., Knittel J., Lauer H., Seiler L. The VolumePro Real-Time Ray-casting System // Proceedings of the 26th annual conference on Computer graphics and interactive techniques (SIGGRAPH '99). – Pp. 251–260.
- [39] Hall D. The AR350: Today's Ray Trace Rendering Processor // Proceedings of the EUROGRAPHICS/SIGGRAPH workshop on Graphics Hardware Hot 3D Session 1. – 2001.
- [40] Schmittler J., Wald I., Slusallek P. SaarCOR: a hardware architecture for ray tracing // Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware. – 2002. – Pp. 27–36.
- [41] Woop S., Schmittler J., Slusallek P. RPU: A Programmable Ray Processing Unit for Realtime Ray Tracing // ACM SIGGRAPH 2005 Papers. – Pp. 434–444.
- [42] Официальный сайт компании Caustic Professional. URL:
<http://www.caustic.com>
- [43] Официальный сайт компании Siliconarts URL:
<http://www.siliconarts.com>
- [44] Abaddon. Chrome Real-time Ray Tracing Demo. Presentation at the Assembly '95 demo party, Helsinki, Finland, 1995. URL:
<http://www.demoparty.net/assembly1995>
- [45] Exceed. Heaven seven. Presentation at the Mekka and Symposium demo party, 2000. URL:
<http://www.demoparty.net/ms2000>
- [46] Federation Against Nature. RealStorm Benchmark 2004. URL:
<http://www.realstorm.com>
- [47] Wald I., Benthin C., Slusallek P. OpenRT - A Flexible and Scalable Rendering Engine for Interactive 3D Graphics: Tech. rep. / Saarland University – 2002. URL:
<http://graphics.cg.uni-saarland.de/2002/wald02openrt>
- [48] Wald I., Benthin C., Slusallek P., Wagner M. Interactive rendering with coherent ray tracing // Computer Graphics Forum. – 2001. – Vol. 20, no. 3. – Pp. 153–164.
- [49] Wald I. Realtime Ray Tracing and Interactive Global Illumination: PhD thesis / Saarland University / Computer Graphics Group. – 2004.
- [50] Reshetov A., Soupikov A., Hurley J. Multi-level ray tracing algorithm // ACM SIGGRAPH 2005 Papers (SIGGRAPH '05)

- [51] Wald I., Ize T., Kensler A., Knoll A., Parker S.G. Ray tracing animated scenes using coherent grid traversal // ACM SIGGRAPH 2006 Papers (SIGGRAPH '06). – 2006. – July. – Vol. 25, no. 3. – Pp. 485– 493.
- [52] Dmitriev K., Havran V., Seidel H.-P. Faster Ray Tracing with SIMD Shaft Culling: Tech. rep. / Max Planck Institute for Computer Science. – 2004. –MPI-I-2004-4-006
- [53] Kirk D., Arvo J. Improved ray tagging for voxel-based ray tracing // Graphics Gems II (Academic Press, 1991). – Pp. 264 – 266.
- [54] Wachter C., Keller A. Instant Ray Tracing: The Bounding Interval Hierarchy // Rendering Techniques 2006 (Proceedings of 17th Eurographics Symposium on Rendering). – 2006. – Pp. 139–149.
- [55] Woop S., Marmitt G., Slusallek P. B-KD Trees for Hardware Accelerated Ray Tracing of Dynamic Scenes // Proceedings of the 21st ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware. – 2006. – Pp. 67–77.
- [56] Zuniga M.R., Uhlmann J.K. Ray queries with wide object isolation and the detree // Journal of Graphics Tools. – 2006. – Vol. 11, no. 3. – Pp. 27–45.
- [57] Havran V., Herzog R., Seidel H.-P. On Fast Construction of Spatial Hierarchies for Ray Tracing // Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing. – 2006. – September. – Pp. 71–80.
- [58] Lauterbach C., Yoon S.-E., Tuft D., and Manocha D. RT-DEFORM: Interactive Ray Tracing of Dynamic Scenes Using BVHs // Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing. – 2006. – Pp. 39–45.
- [59] Yoon S.-E., Curtis S., Manocha D. Ray tracing dynamic scenes using selective restructuring // ACM SIGGRAPH 2007 sketches (SIGGRAPH '07). – Article 55.
- [60] Wald I., Boulos S., Shirley P. Ray tracing deformable scenes using dynamic bounding volume hierarchies // ACM Transactions on Graphics. – 2007. – January. – Vol. 26, no. 1. – Article 6.
- [61] Ize T., Wald I., Parker S.G. Asynchronous BVH construction for ray tracing dynamic scenes on parallel multi-core architectures // Proceedings of the 7th Eurographics conference on Parallel Graphics and Visualization. – 2007. – Pp. 101–108.
- [62] Bergen G. Efficient collision detection of complex deformable models using AABB trees // Journal of Graphics Tools. – 1997. – April. – Vol. 2, no. 4. – Pp. 1–13.
- [63] Schmidl H., Walker N., Lin M. CAB: Fast update of OBB trees for collision detection between articulated bodies // Journal of Graphics Tools. – 2004. – Vol. 9, no. 2. – Pp. 1–9.

- [64] Wald I., Havran V. On building fast kd-trees for ray tracing, and on doing that in $O(N \log N)$: Tech. rep. / University of Utah / SCI Institute. – 2006.
- [65] Gunther J., Friedrich H., Wald I., Seidel H.-P., Slusallek P. Ray tracing animated scenes using motion decomposition // Proceedings of Eurographics. – 2006. – Pp. 517–525.
- [66] Popov S., Gunther J., Seidel H.-P., Slusallek P. Experiences with Streaming Construction of SAH KD-Trees // Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing. – 2006. – Pp. 89–94.
- [67] Hunt W., Stoll G., Mark W. Fast kd-tree Construction with an Adaptive Error-Bounded Heuristic // Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing. – 2006. – Pp. 81–88.
- [68] Shevtsov M., Soupikov A., Kapustin A. Fast and scalable kd-tree construction for interactively ray tracing dynamic scenes // Computer Graphics Forum. – 2007. – Vol. 26, no. 3. – Pp. 395–404.
- [69] Havran V., Bittner J. On improving kd-trees for ray shooting // Proceedings of WSCG. – 2002. – Pp. 209–217.
- [70] Gunther J., Popov S., Seidel H.-P., Slusallek P. Realtime Ray Tracing on GPU with BVH-based Packet Traversal // Proceedings of the 2007 IEEE Symposium on Interactive Ray Tracing. – 2007. – Pp. 113–118.
- [71] Wald I. On fast Construction of SAH-based Bounding Volume Hierarchies // Proceedings of the IEEE Symposium on Interactive Ray Tracing. – 2007. – Pp. 33–40.
- [72] Wald I., Ize T., Parker S.G. Fast, Parallel, and Asynchronous Construction of BVHs for Ray Tracing Animated Scenes // Computers and Graphics. – 2008. – Vol. 32, no. 1. – Pp. 3–13.
- [73] Boulos S., Edwards D., Lacewell J.D., Kniss J., Kautz J., Shirley P., Wald I. Packet-based Whitted and Distribution Ray Tracing // Proceedings of Graphics Interface 2007. – Pp. 177–184.
- [74] Mansson E., Munkberg J., Akenine-Moller T. Deep Coherent Ray Tracing // Proceedings of the 2007 IEEE Symposium on Interactive Ray Tracing. – 2007. – Pp. 79–85.
- [75] Benthin C. Realtime Ray Tracing on current CPU Architectures: PhD thesis / Saarland University. – 2006.
- [76] Bikker J. Arauna Ray Tracer. URL:
<http://lgad.nhtv.nl/~bikker>
- [77] Overbeck R., Ramamoorthi R., Mark W.R. Large Ray Packets for Real-time Whitted Ray Tracing // Proceedings of the IEEE Symposium on Interactive Ray Tracing. – 2008. – Pp. 41–48.

- [78] Boulos S., Wald I., Benthin C. Adaptive Ray Packet Reordering // Proceedings of the IEEE Symposium on Interactive Ray Tracing. – 2008. – Pp. 131–138.
- [79] Purcell T.J., Buck I., Mark W.R., Hanrahan P. Ray Tracing on Programmable Graphics Hardware // ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2002). – 2002. – Vol. 21, no. 3. – Pp. 703–712.
- [80] Purcell T.J., Donner C., Cammarano M., Jensen H.W., Hanrahan P. Photon Mapping on Programmable Graphics Hardware // Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware. – 2003. – Pp. 41–50.
- [81] Jensen H.W. The Photon Map in Global Illumination: PhD thesis / Technical University of Denmark. – 1996.
- [82] Cleary J.G. Analysis of an algorithm for finding nearest neighbors in Euclidean space // ACM Transactions on Mathematical Software (TOMS). – 1979. – Vol. 5, no. 2. – Pp. 183–192.
- [83] Karlsson F., Ljungstedt C.J. Ray tracing fully implemented on programmable graphics hardware: Master's thesis / Chalmers university of technology. – 2004.
- [84] Cohen D., Sheffer Z. Proximity Clouds – An Acceleration Technique for 3D Grid Traversal // The Visual Computer. – 1994. – Vol. 11, no. 1. – Pp. 27–38.
- [85] Ernst M., Vogelgsang C., Greiner G. Stack Implementation on Programmable Graphics Hardware // Proceedings of Vision Modeling and Visualization. – 2004.
- [86] Foley T., Sugerman J. KD-Tree Acceleration Structures for a GPU Raytracer // Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware. – Pp. 15–22.
- [87] Thrane N., Simonsen L.O. A Comparison of Acceleration Structures for GPU Assisted Ray Tracing: Master's thesis / University of Aarhus. – 2005.
- [88] Adinetz A.V., Berezin S.B. Implementing Classical Ray Tracing on GPU – a Case Study of GPU Programming // Proceedings of Graphicon. – 2006.
- [89] Carr N.A., Hoberock J., Crane K., Hart J.C. Fast GPU Ray Tracing of Dynamic Meshes using Geometry Images // Proceedings of Graphics Interface 2006. – Pp. 203–209.
- [90] Horn D.R., Sugerman J., Houston M., Hanrahan P. Interactive k-d tree GPU raytracing // Proceedings of the 2007 symposium on Interactive 3D graphics and games. – Pp. 167–174.
- [91] Popov S., Gunther J., Seidel H.-P., Slusallek P. Stackless KD-Tree Traversal for High Performance GPU Ray Tracing // Computer Graphics Forum. – 2007. – Vol. 26, no. 3. – Pp. 415–424.

- [92] Havran V., Bittner J., Zara J. Ray tracing with rope trees // Proceedings of Spring Conference on Computer Graphics. – 1998. – Pp. 130–140.
- [93] NVIDIA: The CUDA Homepage. URL:
<http://developer.nvidia.com/cuda>
- [94] Zhou K., Hou Q., Wang R., Guo B. Real-time KD-tree construction on graphics hardware // ACM Transactions on Graphics. – 2008. – December. – Vol. 27, no. 5. – Article 126.
- [95] Havran V. Heuristic Ray Shooting Algorithms: PhD thesis / Czech Technical University in Prague. – 2001.
- [96] Aila T., Laine S. Understanding the Efficiency of Ray Traversal on GPUs // Proceedings of the Conference on High Performance Graphics. – 2009. – Pp. 145–149.
- [97] Garanzha K. Loop C. Fast Ray Sorting and Breadth-First Packet Traversal for GPU Ray Tracing // Computer Graphics Forum. – 2010. – Vol. 29, no. 2. – Pp. 289–298.
- [98] Aila T., Laine S., Karras T. Understanding the Efficiency of Ray Traversal on GPUs – Kepler and Fermi Addendum: Tech. rep. / NVIDIA. – 2012. –NVR-2012-02
- [99] Danilewski P., Popov S., Slusallek P. Binned SAH KD-Tree Construction on a GPU // Saarland University. – 2010. – Pp. 1–15.
- [100] Wu Z., Zhao F., Liu X. SAH KD-Tree Construction on GPU // Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics. – 2011. – Pp. 71–78.
- [101] Choi B., Komuravelli R., Lu V., Sung H., Bocchino R.L., Adve S.V., Hart J.C. Parallel SAH k-D tree construction // Proceedings of the Conference on High Performance Graphics. – 2010. – Pp. 77–86.
- [102] Lauterbach C., Garland M., Sengupta S., Luebke D., Manocha D. Fast BVH Construction on GPUs // Computer Graphics Forum. – 2009. – Vol. 28, no. 2. – Pp. 375–384.
- [103] Satish N., Harris M., Garland M. Designing efficient sorting algorithms for manycore GPUs // Proceedings of the IEEE International Symposium on Parallel&Distributed Processing. – 2009. – Pp. 1–10.
- [104] Pantaleoni J., Luebke D. HLBVH: hierarchical LBVH construction for real-time ray tracing of dynamic geometry // Proceedings of the Conference on High Performance Graphics. – 2010. – Pp. 87–95.
- [105] Garanzha K., Premoze S., Bely A., Galaktionov V. Grid-based SAH BVH construction on a GPU // The Visual Computer. – 2011. – June.– Vol. 27, no. 6-8. – Pp. 697–706.

- [106] Stich M., Friedrich H., Dietrich A. Spatial splits in bounding volume hierarchies // Proceedings of the Conference on High Performance Graphics. – 2009. – Pp. 7–13.
- [107] Kalojanov J., Slusallek P. A Parallel Algorithm for Construction of Uniform Grids // Proceedings of the Conference on High Performance Graphics. – 2009. – Pp. 23–28.
- [108] Ivson P., Duarte L., Celes W. Gpu-accelerated uniform grid construction for ray tracing dynamic scenes: Tech. rep. / Pontifícia Universidade Católica do Rio de Janeiro / Departamento de Informatica. – 2009. – June.
- [109] Kalojanov J., Billeter M., Slusallek P. Two-Level Grids for Ray Tracing on GPUs // Computer Graphics Forum. – 2011. – Vol. 30, no. 2. – Pp. 307–314.
- [110] Jevans D., Wyvill, B. Adaptive voxel subdivision for ray tracing // Proceedings of Graphics Interface 1989. – Pp. 164–172.
- [111] Ritschel T., Dachsbacher C., Grosch T., Kautz J. The State of the Art in Interactive Global Illumination // Computer Graphics Forum. – 2012. – February. – Vol. 31, no. 1. – Pp. 160–188.
- [112] Aila T., Karras T. Architecture considerations for tracing incoherent rays // Proceedings of the Conference on High Performance Graphics. – 2010. – Pp. 113–122.
- [113] Novak J., Havran V., Dachsbacher C. Path Regeneration for Interactive Path Tracing // Proceedings of EUROGRAPHICS (Short Papers). – 2010. – Pp. 61–64.
- [114] Wald I. Active Thread Compaction for GPU Path Tracing // Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics. – 2011. – Pp. 51–58.
- [115] Antwerpen D.V. Improving SIMD Efficiency for Parallel Monte Carlo Light Transport on the GPU // Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics. – 2011. – Pp. 41–50.
- [116] Antwerpen D. V. Recursive MIS Computation for Streaming BDPT on the GPU: Tech. rep. / Delft University of Technology. – 2011. – September.
- [117] Garanzha K., Bely A., Premoze S., Galaktionov V. Out-of-core GPU ray tracing of complex scenes // ACM SIGGRAPH 2011 Talks. – 2011. – Article 21.
- [118] Parker S.G., Bigler J., Dietrich A., Friedrich H., Hoberock J., Luebke D., McAllister D., McGuire M., Morley K., Robison A., Stich M. OptiX: A General Purpose Ray Tracing Engine // ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2010). – 2010. – July. – Vol. 29, no. 4. – Article 66.

- [119] Система визуализации NVIDIA iRay. URL:
<http://www.irayrender.com>
- [120] Официальный сайт компании RandomControl. URL:
<http://www.randomcontrol.com>
- [121] Система визуализации FurryBall. URL:
<http://furryball.aaa-studio.eu>
- [122] Система визуализации Indigo Renderer. URL:
<http://www.indigorenderer.com>
- [123] Система визуализации Maxwell Render. URL:
<http://www.maxwellrender.com>
- [124] Официальный сайт компании Chaos Group. URL:
<http://www.chaosgroup.com>
- [125] Система визуализации NVIDIA mental ray. URL:
<http://www.nvidia-arc.com>
- [126] Система визуализации Octane Render. URL:
<http://render.otoy.com>
- [127] Система визуализации LuxRender. URL:
<http://www.luxrender.net>
- [128] Система визуализации Cycles Render (компонента системы Blender). URL:
<http://www.blender.org>
- [129] Veach E. Robust Monte Carlo Methods for Light Transport Simulation: PhD thesis / Stanford University. – 1997. – December.
- [130] Antwerpen D.V. A Survey of Importance Sampling Applications in Unbiased Physically Based Rendering (2011). URL:
<http://graphics.tudelft.nl/~dietger/survey.pdf>
- [131] Nicodemus F.E., Richmond J.C., Hsia J.J., Ginsberg I.W., Limperis T. Geometrical considerations and nomenclature for reflectance // Radiometry. – 1992. – Pp. 94–145.
- [132] Samet H. Foundations of Multidimensional and Metric Data Structures. – Morgan Kaufmann, San Francisco, 2006.
- [133] Gaede V., Gunther O. Multidimensional access methods // ACM Computing Surveys. – 1998. – June. – Vol. 30, no. 2. – Pp. 170–231.
- [134] Goldsmith J., Salmon J. Automatic creation of object hierarchies for ray tracing // IEEE Computer Graphics and Applications. – 1987. – May. – Vol. 7, no. 5. – Pp. 4–20.

- [135] MacDonald J.D., Booth K.S. Heuristics for ray tracing using space subdivision // The Visual Computer. – 1990. – Vol. 6, no. 3. – Pp. 153–166.
- [136] Subramanian K. R. A Search Structure based on kd-Trees for Efficient Ray Tracing: PhD thesis / University of Texas at Austin. – 1990.
- [137] Wald I. Fast Construction of SAH BVHs on the Intel Many Integrated Core (MIC) Architecture // IEEE Transactions on Visualization and Computer Graphics. – 2012. – January. – Vol. 18, no. 1. – Pp. 47–57.
- [138] Harris M. Optimizing Parallel Reduction in CUDA. URL:
http://www.uni-graz.at/~haasegu/Lectures/GPU_CUDA/Lit/reduction.pdf
- [139] Satish N., Harris M., Garland M. Designing efficient sorting algorithms for manycore GPUs // Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing. – 2009. – Pp. 1–10.
- [140] Harris M., Sengupta S., Owens J.D. Parallel prefix sum (scan) with CUDA // GPU Gems 3. – 2007. – August. – Chapter 39.
- [141] Проект OpenSG. URL:
<http://www.opensg.org>
- [142] Проект OpenSceneGraph. URL:
<http://www.openscenegraph.org/projects/osg>
- [143] Heckbert P.S. Adaptive radiosity textures for bidirectional ray tracing // Proceedings of the 17th annual conference on Computer graphics and interactive techniques (SIGGRAPH '90). – 1990. – August. – Vol. 24, no. 4. – Pp. 145–154.
- [144] Pharr M., Humphreys G. Physically Based Rendering, Second Edition: From Theory to Implementation. – Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2010. – ISBN 0123750792 9780123750792.
- [145] Oren M., Nayar S.K. Generalization of Lambert's reflectance model // Proceedings of the 21st annual conference on Computer graphics and interactive techniques (SIGGRAPH '94). – 1994. – Pp. 239–246.
- [146] Walter B. Notes on the Ward BRDF: Tech. rep. / Cornell University. – 2005. – April. – PCG-05-06. URL: *<http://www.graphics.cornell.edu/~bjw/wardnotes.pdf>*
- [147] Georgiev I., Křivánek J., Slusallek P. Bidirectional light transport with vertex merging // SIGGRAPH Asia 2011 Sketches (SA '11). – 2011. – Article 27.

- [148] Волобой А.Г., Галактионов В.А., Дмитриев К.А., Копылов Э.А. Двухнаправленная трассировка лучей для интегрирования освещенности методом квази- Монте Карло // Программирование. – 2004. – № 5. – С. 25–34.
- [149] Игнатенко А. Задачи синтеза фотореалистичных изображений // Сетевой журнал "Компьютерная графика и мультимедиа". – 2008. – № 6(1). URL:
http://cgm.computergraphics.ru/issues/issue16/render_tasks
- [150] Адинец А. Игнатенко А. Методы Монте-Карло и быстрой излучательности для интерактивной визуализации глобального освещения // Сетевой журнал "Компьютерная графика и мультимедиа". – 2004. – № 2(3). URL:
<http://cgm.computergraphics.ru/content/view/64>
- [151] Фролов В., Игнатенко А. Интерактивная трассировка лучей и фотонные карты на GPU // Труды конференции GraphiCon'2009. – 2009. – С. 255–262.
- [152] Frolov V., Vostriakov K., Kharlamov A., Galaktionov V. Irradiance cache for a GPU ray tracer // Proceedings of 22-th International Conference on Computer Graphics and Vision. – 2012. – Pp. 39–44.
- [153] Кулясов П.С., Никулин Е.А. Ускорение вычислений форм-факторов при расчете освещенности сцены методом излучательности // Вестник Нижегородского университета им. Н.И. Лобачевского. – 2012. – Вып. 3(1). – С. 184–188.
- [154] Боголепов Д.К., Сопин Д.С., Турлапов В.Е. Упрощенный метод фотонных карт для визуализации каустик в реальном времени // Программирование. – 2011. – № 5. – С. 3–12. – ISSN 0132-3474.
- [155] Боголепов Д.К., Турлапов В.Е. Моделирование каустик в реальном времени на основе комбинированных возможностей OpenGL и шейдеров // Вестник Нижегородского университета им. Н.И. Лобачевского. – 2011. – Вып. 3(2). – С. 180–186.
- [156] Боголепов Д.К., Сопин Д.П., Ульянов Д.Я., Турлапов В.Е. Построение SAN BVH деревьев для трассировки лучей с использованием графических процессоров // Научно-технический журнал «Приборостроение». – 2011. – Вып. 10. – С. 92–94.
- [157] Боголепов Д.К., Сопин Д.П., Ульянов Д.Я., Турлапов В.Е. Интерактивное моделирование глобального освещения на GPU для анимированных гетерогенных сцен // Вестник Нижегородского университета им. Н.И.Лобачевского. – 2012. – Вып. 5(2). – С. 251–259.
- [158] Боголепов Д.К., Турлапов В.Е. Вычисления общего назначения на графических процессорах с использованием шейдерных языков // Труды международной научной

- конференции «Параллельные вычислительные технологии». – Челябинск: Изд-во ЮУрГУ, 2009. – С. 40–47.
- [159] Боголепов Д., Трушанин В., Турлапов В. Интерактивная трассировка лучей на графическом процессоре // Труды 19-ой международной конференции по компьютерной графике и зрению (GraphiCon'2009). – М.: Изд-во МГУ, 2009. – С. 263–267.
- [160] Боголепов Д., Удалова Т., Турлапов В. Интерактивная трассировка лучей на графическом процессоре // Труды 19-ой международной конференции по компьютерной графике и зрению (GraphiCon'2009). – М.: Изд-во МГУ, 2009. – С. 392–394.
- [161] Боголепов Д.К., Блохин О.Д., Захаров М.М., Сопин Д.П. Исследование высокопроизводительного решения задачи N тел на базе платформы OpenCL // Труды всероссийской конференция молодых ученых «Технологии Microsoft в теории и практике программирования». – Нижний Новгород: Изд-во ННГУ, 2010. – С. 34–37.
- [162] Боголепов Д.К., Блохин О.Д., Захаров М.М., Сопин Д.П. GLSL и OpenCL – сравнительный опыт вычислений // Тезисы докладов научно-практической конференции «Вычисления с использованием графических процессоров в молекулярной биологии и биоинформатике». – М.: Изд-во МГУ, 2010. – С. 44–46.
- [163] Боголепов Д., Сопин Д., Турлапов В. Моделирование каустик в реальном времени // Труды 20-ой международной конференции по компьютерной графике и зрению (GraphiCon'2010). – СПб: Изд-во СПбГУ ИТМО, 2010. – С. 253–256.
- [164] Sopin D., Bogolepov D., Ulyanov D. Real-time SAH BVH construction for ray tracing dynamic scenes // Proc. of the 21th International Conference on Computer Graphics and Vision «GraphiCon'2011». – Moscow, 2011. – Pp. 74–78.
- [165] Сопин Д.П., Боголепов Д.К., Ульянов Д.Я., Турлапов В.Е. Построение SAH BVH деревьев для трассировки лучей на GPU в реальном времени // Материалы XI Всероссийской конференции «Высокопроизводительные параллельные вычисления на кластерных системах». – Нижний Новгород: Изд-во ННГУ, 2011, – С. 298–303.
- [166] Боголепов Д., Бугаев И., Сопин Д., Ульянов Д., Турлапов В. Высококачественная объемная визуализации в реальном времени // Труды 22-ой международной конференции по компьютерной графике и зрению (GraphiCon'2012). – М.: МАКС Пресс, 2012. – С. 169–174.
- [167] Боголепов Д., Сопин Д., Ульянов Д., Турлапов В. Система интерактивного расчета глобального освещения для гибридных сцен // Труды 22-ой международной конференции

по компьютерной графике и зрению (GraphiCon'2012). – М.: МАКС Пресс, 2012. – С. 227–233.

