СОДЕРЖАНИЕ

* Звездочкой отмечены разделы с продвинутым содержанием, которые можно пропустить при первом чтении.

ПРЕДИСЛОВИЕ

«Информация должна быть доступна для тех, кто хочет учиться и понимать; для программистов исходный код программы является единственным средством учиться искусству у своих предшественников. Было бы немыслимо, чтобы драматурги не позволяли другим драматургам читать тексты своих пьес на театральных представлениях и им было бы запрещено даже делать заметки. Любой достойный автор текста хорошо начитан, и каждый ребенок, который хочет научиться писать, должен читать в сотни раз больше, чем пишет. Вряд ли разумно требовать от будущих программистов, чтобы они сами изобрели алфавит и тут же научились писать большие романы. Программирование не может развиваться, если каждое следующее поколение программистов не будет иметь доступа к знаниям и информации, приобретенным другими программистами, их предшественниками».

– Эрик Наггум

Рендеринг является фундаментальной составляющей компьютерной графики. На самом высоком уровне абстракции рендеринг представляет собой процесс преобразования описания трехмерной сцены в изображение. Алгоритмы анимации, геометрического моделирования, текстурирования и других областей компьютерной графики должны пропускать свои результаты через процесс рендеринга, чтобы их можно было увидеть как изображение. Рендеринг стал вездесущим: от фильмов до игр и других областей; он открыл новые границы для творческого самовыражения, развлечений и рендеринга.

В первые годы исследования в области рендеринга были сосредоточены на решении фундаментальных проблем, таких как определение того, какие объекты видны с данной точки зрения. По мере того, как были найдены эффективные решения этих проблем, и по мере того, как благодаря постоянному прогрессу в других областях графики становились доступными более проработанные и реалистичные описания сцен, рендеринг стал включать в себя идеи из широкого круга дисциплин, в том числе физику и астрофизику, астрономию, биологию, психологию и изучение зрительного восприятия, а также абстрактную и прикладную математику. Междисциплинарный характер рендеринга является одной из причин того, что он стал настолько привлекательной областью для изучения.

В этой книге представлен набор современных алгоритмов рендеринга с помощью документированного исходного кода полной системы рендеринга. Почти все изображения в этой книге, включая изображение на обложке, были созданы описанным в книге

программным обеспечением. Все алгоритмы, которые были объединены для создания этих изображений, описаны на страницах этой книги. Система pbrt создана с использованием методологии программирования, называемой *грамотным программированием* (или *литературным программированием*, «literate programming»), которая объединяет прозаический текст, описывающий систему, с исходным кодом, который ее реализует. Мы считаем, что грамотное программирование является ценным способом реализации идей в компьютерной графике и информатике в целом. Часто некоторые тонкости алгоритма могут не проявить себя до тех пор, пока он не будет разработан, поэтому просмотр фактической реализации – хороший способ получить четкое представление о его деталях. Действительно, мы считаем, что такое глубокое понимание ряда тщательно отобранных нами алгоритмов обеспечивает лучшую основу для дальнейшего изучения компьютерной графики, чем поверхностное изучение многих из них.

Чтобы лучше продемонстрировать, как алгоритм реализуется на практике, мы представляем эти алгоритмы в контексте полной и нетривиальной программной системы, что также позволяет нам решать проблемы проектирования и реализации систем рендеринга среднего размера. Разработка основных абстракций и интерфейсов системы рендеринга имеет существенные последствия как на элегантность ее реализации, так и на возможность дальнейшего ее расширения, однако компромиссы в этом пространстве разработки обсуждаются редко.

Работа с системой pbrt, как и все содержание этой книги, сосредоточена исключительно на *фотореалистичном рендеринге*. Фотореалистичность можно определить по-разному: как задачу создания изображений, не отличимых от тех, что запечатлела бы камера на фотографии, или как создание изображений, реакция на которые у человека, рассматривающего их, не отличима от реакции человека, смотрящего на реальную сцену. Есть много причин сосредоточиться на фотореализме. Фотореалистичные изображения имеют решающее значение для спецэффектов в фильмах, потому что изображения, созданные компьютером, должны составлять естественную часть реального мира фильма. В таких приложениях, как компьютерные игры, где все изображения являются синтетическими, фотореализм выступает эффективным инструментом, позволяющим наблюдателю забыть о том, что он смотрит на сцену, которой на самом деле не существует. Наконец, фотореализм дает четкий критерий оценки качества результата рендеринга.

АУДИТОРИЯ КНИГИ

Эта книга предназначена для трех основных групп читателей. Первая группа – это аспиранты или студенты старших курсов высших учебных заведений, изучающие компьютерную графику. В этой книге предполагается, что у читателя уже есть знания основ компьютерной графики на уровне колледжа, но здесь будут рассмотрены также некоторые важнейшие понятия, такие как начала векторной алгебры и математические преобразования. Студентам, у которых нет опыта работы с программами, содержащими десятки тысяч строк исходного кода, «грамотное программирование» даст мягкое введение в эти сложности. Мы также уделили особое внимание объяснению некоторых внутренних взаимосвязей и абстракций в системе pbrt, чтобы дать читателям представление о том, почему она устроена так, как она есть, а не иначе.

Вторая предполагаемая группа читателей – это далеко продвинувшиеся в компьютерной графике аспиранты и исследователи данной области. Для тех, кто занимается исследованиями в области рендеринга, книга представляет собой широкое введение в эту дисциплину, а исходный код pbrt обеспечивает основу, на которой можно строить свои разработки (или, по крайней мере, использовать фрагменты исходного кода). Для тех, кто работает в других областях компьютерной графики, мы уверены, может оказаться полезным глубокое понимание технологии рендеринга.

Наша конечная аудитория – разработчики программного обеспечения в компьютерной индустрии. Хотя многие основные идеи книги им уже знакомы, просмотр комментариев к алгоритмам, представленных в грамотном стиле программирования, может

подсказать новые идеи. Система pbrt также содержит в себе тщательно разработанные и отлаженные реализации многих алгоритмов, которые могут быть сложными для корректной реализации; они должны представлять особый интерес для опытных практиков рендеринга. Мы надеемся, что изучение конкретной структуры полной и нетривиальной системы рендеринга может натолкнуть эту аудиторию на размышления.

ОБЗОР И ЦЕЛИ

Система pbrt основана на алгоритме *рейтрейсинга* (трассировки лучей). Трассировка лучей – это элегантная техника, которая берет свое начало в производстве линз; Карл Фридрих Гаусс описал прохождение лучей через линзы еще в XIX веке. Алгоритмы трассировки лучей на компьютерах следуют пути лучей света бесконечно малой толщины через сцену, пока они не пересекают какую-либо поверхность. Этот метод дает простой метод нахождения первого объекта, видимого с любой определенной позиции и направления, и является основой для многих алгоритмов рендеринга.

Система pbrt была разработана и реализована с учетом трех основных целей: она должна быть *полной*, *иллюстративной* и *физически корректной* (соответствующей законам физики).

Полнота подразумевает, что в системе не должно быть недостатка важнейших функций, необходимых для высококачественных коммерческих систем рендеринга. В частности, это означает, что все важные практические вопросы, такие как антиалайзинг, надежность, числовая точность и возможность эффективного рендеринга сложных сцен, должны быть детально решены. Было важно рассмотреть эти вопросы с самого начала проектирования системы, поскольку реализация данных функций может иметь отдаленные последствия для всех компонент системы, и их может быть весьма сложно модифицировать на более позднем этапе разработки системы.

Вторая цель определяла наш тщательный выбор алгоритмов, структур данных и методов рендеринга с прицелом на удобочитаемость и ясность. Поскольку их реализации будет изучать больше читателей, чем в случае с другими системами рендеринга, мы попытались выбрать самые элегантные алгоритмы, о которых мы знали, и реализовать их как можно лучше. Эта цель также требовала, чтобы система была достаточно маленькой – она должна быть в меру понятной, чтобы ее целиком мог охватить и один человек. Мы реализовали pbrt, используя расширяемую архитектуру, с ядром системы в виде набора тщательно разработанных классов интерфейсов и максимально конкретной функциональностью в реализациях этих интерфейсов. В результате у читателя нет необходимости разбираться во всех конкретных реализациях, чтобы понять базовую структуру системы. Это дает возможность углубиться в интересующие его части и пропустить другие, не теряя из виду работу системы как единого целого.

Существует противоречие между двумя качествами: полнотой и иллюстративностью. Описание и разработка всех возможных полезных техник не только сделали бы эту книгу неприемлемо длинной, но и представили систему pbrt чрезмерно сложной для большинства читателей. В тех случаях, когда в pbrt отсутствует необходимая функция, мы пытались спроектировать архитектуру таким образом, чтобы эту функцию можно было добавить, не изменяя общее устройство системы.

Основой физически корректного рендеринга являются законы физики и их математическая формулировка. Система pbrt была разработана с использованием адекватных физических единиц и понятий для вычисляемых величин и алгоритмов, которые она реализует. Она стремится визуализировать *физически корректные* изображения, которые точно отображают освещение, каким оно было бы в реальном воплощении сцены[1]. Одним из преимуществ решения использовать физическую основу является то, что она дает конкретный стандарт корректности программы: если для простых сцен, где ре-

[1] Конечно, любое компьютерное моделирование физики требует тщательного выбора приближений, которые сочетают требования точности с вычислительной эффективностью. См. раздел 1.2 для дальнейшего обсуждения выбора, сделанного в pbrt. – *Прим. авт.*

зультат может быть вычислен в общем виде, pbrt не представляет тот же результат, мы знаем, что в разработке есть ошибка. Точно так же если разные основанные на физике алгоритмы освещения в pbrt дают разные результаты для одной и той же сцены или если pbrt не дает таких же результатов, как другой основанный на законах физики визуализатор, то в одном из них определенно есть ошибка. Наконец, мы считаем, что этот физически обоснованный метод рендеринга является ценным, потому что он строгий. Когда не ясно, как следует выполнять то или иное вычисление, физика дает ответ, гарантирующий адекватный результат.

Эффективности уделялось меньше внимания, чем этим трем целям. Поскольку системам рендеринга часто требуются многие минуты или даже часы для рендеринга изображения, очевидно, что эффективность важна. Однако мы в основном ограничивались *алгоритмической* эффективностью, а не низкоуровневой оптимизацией кода. В некоторых случаях очевидные микрооптимизации уступают место чистому, хорошо организованному коду, и мы приложили усилия для оптимизации частей системы, в которых происходит большая часть вычислений.

В ходе представления pbrt и обсуждения его реализации мы надеемся поделиться некоторыми трудными уроками, усвоенными за годы исследований и разработок в области рендеринга. Написание хорошего средства рендеринга – это нечто большее, чем составление набора быстрых алгоритмов; сделать систему одновременно гибкой и надежной – сложная задача. Производительность системы по мере добавления к ней новых геометрических элементов, источников света или любого другого усложнения сцены должна снижаться плавно.

Вознаграждение за разработку системы, которая решает все эти задачи, огромно – написать новый модуль рендеринга или добавить новую функцию в существующий модуль рендеринга и использовать его для создания изображения, которое раньше нельзя было создать, будет огромным удовольствием. Нашей основной целью при написании данной книги было предоставить эту возможность более широкой аудитории. Читателям рекомендуется использовать систему pbrt для рендеринга примеров сцен, содержащихся в дистрибутиве pbrt, по мере того как они будут продвигаться в освоении материала книги. Упражнения в конце каждой главы предлагают модификации системы, которые помогут прояснить ее внутреннюю работу, и разработку более сложных проектов по расширению системы путем добавления новых функций.

Веб-сайт данной книги находится по адресу pbrt.org. Этот сайт содержит ссылки на исходный код pbrt, а также сцены, которые можно загрузить для рендеринга с помощью pbrt, средство трассировки багов и перечень ошибок в тексте. О любых ошибках в этом тексте, которые не указаны в данном списке, можно сообщить по адресу электронной почты authors@pbrt.org. Мы очень ценим ваши отзывы!

ИЗМЕНЕНИЯ МЕЖДУ ПЕРВЫМ И ВТОРЫМ ИЗДАНИЯМИ

Между публикацией первого издания этой книги в 2004 г. и второго издания в 2010 г. прошло шесть лет. За это время были проданы тысячи экземпляров книги, а программное обеспечение pbrt было загружено с веб-сайта книги тысячи раз. База пользователей pbrt дала нам значительное количество отзывов и рекомендаций, и наш опыт работы с системой повлиял на многие решения, которые мы приняли при внесении изменений между версией pbrt, представленной в первом издании, и версией второго издания. В дополнение к ряду исправлений багов мы также внесли несколько существенных изменений и улучшений в реализацию системы:

- *удаление архитектуры подключаемых модулей*. В первой версии pbrt использовалась архитектура подключаемых модулей среды выполнения для динамической загрузки при описании таких объектов, как формы, источники света, интеграторы, камеры и другие объекты, которые использовались в сцене, визуализируемой в данный момент. Такой метод позволял пользователям расширять pbrt новыми типами объектов (например, новыми примитивами формы) без перекомпиляции всей системы рендеринга. Этот метод изначально казался элегантным, но он

усложнял задачу поддержки pbrt на нескольких платформах и усложнял отладку. Единственный новый сценарий использования, который он действительно позволял (только бинарные дистрибутивы pbrt или бинарные плагины), на самом деле противоречил нашим педагогическим целям и целям открытого исходного кода. Поэтому в этом издании архитектура плагинов была исключена;

- *удаление конвейера обработки изображений*. В первой версии pbrt был интерфейс маппинга, который преобразовывал выходные изображения высокого динамического диапазона (HDR) с плавающей запятой непосредственно в TIFF с низким динамическим диапазоном отображения. Эта функциональность имела смысл в 2004 г., поскольку HDR-изображения еще были редкостью. Однако в 2010 г. достижения в области цифровой фотографии сделали HDR-изображения обычным явлением. Хотя теория и практика тонального маппинга элегантны и заслуживают изучения, мы решили сфокусировать новую книгу исключительно на процессе формирования изображения и пропустить тему его отображения. Заинтересованные читатели могут обратиться к книге, написанной Рейнхардом с соавт. (2010), про современную точную технологию обработки рендеринга HDR для его отображения;
- *параллелизм задач*. Многоядерные архитектуры стали общераспространенными, и мы посчитали, что pbrt не будет актуальна без возможности расширения процесса рендеринга на все локально доступные ядра. Мы также надеялись, что детали реализации параллельного программирования, описанные в этой книге, помогут программистам компьютерной графики понять некоторые тонкости и сложности написания масштабируемого параллельного кода;
- *пригодность для «производственного» рендеринга*. Первая версия pbrt предназначалась исключительно как педагогический инструмент и трамплин для проведения исследований. Действительно, при подготовке первого издания мы приняли ряд решений, противоречащих его использованию в производственном процессе, таких как ограниченная поддержка фонового освещения на основе изображений, отсутствие поддержки размытия в движении и ненадежная реализация фотонного маппинга при наличии комплексного освещения. Мы чувствуем, что со значительно улучшенной поддержкой этих функций, а также поддержкой подповерхностного рассеяния и светового переноса Метрополис второе издание pbrt стало гораздо более подходящим для рендеринга высококачественных изображений сложных сред.

ИЗМЕНЕНИЯ МЕЖДУ ВТОРЫМ И ТРЕТЬИМ ИЗДАНИЯМИ

По прошествии еще шести лет пришло время обновить и расширить и систему pbrt, и книгу. Мы продолжали учиться на опыте читателей и пользователей, чтобы лучше понять, какие темы наиболее важны, дабы их точнее проработать. Кроме того, быстрыми темпами продолжались исследования в области рендеринга; многие части книги должны были быть обновлены, чтобы отразить передовой опыт. Мы внесли значительные улучшения по ряду направлений:

- *двунаправленный световой перенос*. В третьей версии pbrt добавлен двунаправленный трассировщик пути, включающий полную поддержку объемного светового переноса и множественный семплинг[2] по значимости для взвешивания путей. Совершенно новый интегратор светового переноса Метрополис использовал компоненты двунаправленного трассировщика пути, что позволило реализовать этот алгоритм особенно лаконично;
- *подповерхностное рассеяние*. Внешний вид многих объектов, особенно кожи и полупрозрачных объектов, – это результат подповерхностного светового переноса.

[2] Наряду с «семплингом» также часто применяется термин «выборка», но в контексте данной книги это менее точно соответствует смыслу, так что применяется калька с английского, широко распространенная в профессиональном сообществе. – *Прим. ред.*

Наша реализация подповерхностного рассеяния во втором издании отражала состояние дел на начало 2000-х; мы тщательно обновили как модели BSSRDF, так и наши алгоритмы подповерхностного светового переноса, чтобы отразить прогресс, достигнутый за десять последующих лет исследований;

- *численно надежные пересечения*. Эффекты ошибки округления чисел с плавающей запятой в вычислениях пересечения геометрических лучей были давней проблемой в рейтрейсинге: они могут привести к небольшим ошибкам, присутствующим по всему изображению. Мы сосредоточились на этой проблеме и получили консервативные (но жесткие) границы данной ошибки, что сделало нашу реализацию более устойчивой к этой проблеме, чем предыдущие системы рендеринга;
- *представление передающих сред*. Мы значительно улучшили способ, каким рассеяние описано и представлено в системе; это позволяет получать более точные результаты с рассеивающими средами сцены. Новая техника семплинга («выборки», «пробы») позволила адекватно отображать гетерогенные среды таким образом, чтобы они были четко интегрированы со всеми другими частями объекта;
- *измеряемые материалы*. В этом издании добавлена новая техника для представления и измерения определенных материалов с использованием разреженной частотно-пространственной базы. Данный метод удобен тем, что позволяет проводить точный семплинг по значимости, что было невозможно при представлении, использовавшемся в предыдущем издании;
- *фотонный маппинг* (метод фотонных карт). Значительным шагом вперед для алгоритмов фотонного маппинга стала разработка переменных, не требующих хранения в памяти всех фотонов. Мы заменили алгоритм фотонного маппинга `pbrt`-реализацией, основанной на стохастическом прогрессивном фотонном маппинге, которая эффективно воспроизводит многие сложные эффекты светового переноса;
- *алгоритмы генерации семплов*. Распределение значений семплов, используемое для цифрового интегрирования в алгоритмы рендеринга, может иметь удивительно большое влияние на качество конечного результата (картинки). Мы тщательно обновили методику этой темы, охватив новые и эффективные методы реализации более подробно, чем раньше.

Многие другие части системы были улучшены и обновлены, чтобы отразить прогресс в соответствующих областях: модели микрофасеточного отражения были проработаны более глубоко, с гораздо лучшей техникой семплинга; добавлена новая форма «curve» («кривая») для моделирования волос и другой тонкой геометрии; стала доступной новая модель камеры, которая имитирует объективы реальных производителей оптического оборудования. На протяжении всей книги мы внесли множество небольших изменений, чтобы более четко объяснить и проиллюстрировать главные концепции физически корректных систем рендеринга, таких как `pbrt`.

ИЗМЕНЕНИЯ МЕЖДУ ТРЕТЬИМ И ЧЕТВЕРТЫМ ИЗДАНИЯМИ

Инновации в алгоритмах рендеринга не демонстрируют признаков замедления, поэтому в 2019 году мы начали целенаправленную работу над четвертым изданием книги. Мало того, что почти каждая глава содержит существенные дополнения, мы также изменили порядок представленных глав и идей, выдвинув интегрирование методом Монте-Карло и основные идеи трассировки пути на передний план, переместив их из конца оглавления.

Возможности системы, которые претерпели действительно значительные улучшения, включают:

- *объемное рассеяние*. Мы обновили алгоритмы, которые моделируют рассеяние сред, до современного состояния, добавив поддержку излучающих объемов, эффективный семплинг объемов с различной плотностью и надежную поддержку хроматических сред, где характер рассеяния зависит от длины волны;

- *спектральный рендеринг*. Мы исключили любое использование цветов RGB для вычислений освещения; `pbrt` теперь выполняет их исключительно для семплинга спектральных распределений интенсивностей по длине волны. Этот метод не только более физически точен, но и позволяет `pbrt` точно моделировать такие эффекты, как дисперсия;
- *модели отражения*. Наше описание основ BSDF и моделей отражения было значительно пересмотрено, и мы расширили диапазон рассматриваемых функций BSDF, включив в него одну, которая точно моделирует отражение от волос, и другую, которая моделирует рассеяние от слоистых материалов. Измеренное значение BRDF следует новому методу, который охватывает широкий набор спектров отражения материалов;
- *семплинг источников света*. Мы не только улучшили алгоритмы семплинга точек для отдельных источников света, чтобы лучше отражать современное состояние разработок, но эта редакция также содержит поддержку семплинга множественных источников света, что позволяет эффективно визуализировать сцены с тысячами или миллионами источников света, тщательно семплируя лишь некоторые из них;
- *рендеринг на GPU*. В этой версии `pbrt` добавлена поддержка рендеринга на GPU (графическом процессоре), который обеспечивает в 10–100 раз более высокую производительность рейтрейсинга, чем CPU. Мы реализовали эту возможность таким образом, что почти весь код, представленный в книге, работает как на центральных процессорах, так и на графических, что позволило перенести обсуждение вопросов, связанных с графическими процессорами, в главу 15.

Система претерпела множество других улучшений и дополнений, в том числе новую билинейную форму патча, множество обновлений алгоритмов генерации семплов, самой сердцевины интегрирования Монте-Карло, поддержку вывода вспомогательной информации в каждом пикселе о видимой геометрии поверхности и параметрах отражения, а также множество других мелких улучшений системы.

БЛАГОДАРНОСТИ

Пэт Ханрахан внес в эту книгу больший вклад, чем мы могли надеяться; мы в огромном долгу перед ним. Он неустанно выступал за ясные интерфейсы и поиск правильных абстракций для использования во всей системе, и его понимание и метод рендеринга глубоко повлияли на ее разработку. Его готовность использовать `pbrt` и эту рукопись в своем курсе рендеринга в Стэнфорде была чрезвычайно полезна, особенно в первые годы, когда она была еще в грубой форме; обратная связь с ним на протяжении всего этого процесса имела решающее значение для приведения текста в его нынешнее состояние. Наконец, группа людей, которую Пэт помог собрать в Стэнфордской графической лаборатории, и созданная им открытая среда общения создали захватывающую, стимулирующую и плодотворную обстановку. Мэтт и Грег чувствуют себя там очень комфортно.

Мы в долгу перед многими студентами, которые использовали ранние варианты этой книги на курсах в Стэнфорде и Университете Вирджинии в период с 1999 по 2004 г. Эти студенты предоставили огромное количество отзывов о книге и `pbrt`. Особого упоминания заслуживают ассистенты преподавателей этих курсов: Тим Перселл, Майк Каммарано, Ян Бак и Рен Нг в Стэнфорде и Нолан Гуднайт в Вирджинии. Несколько студентов в этих классах дали особенно ценные отзывы и прислали отчеты об ошибках и исправления ошибок; мы особенно хотели бы поблагодарить Эвана Паркера и Фила Битти. Черновик рукописи этой книги использовался на занятиях, которые вели Билл Марк и Дон Фассел в Техасском университете в Остине и Рагху Мачираджу в Университете штата Огайо; их отзывы были бесценны, и мы благодарны им за предприимчивость при включении этой системы в свои курсы, даже когда она еще редактировалась и пересматривалась.

Мэтт Фарр хотел бы выразить признательность коллегам и сотрудникам по работе с рендерингом, которые стали отличным источником знаний и существенно повлияли

на его метод написания модулей рендеринга и его понимание нашей темы. Особая благодарность Крейгу Колбу, который заложил краеугольный камень в фундамент образования Мэтта в области компьютерной графики в виде бесплатно доступного исходного кода системы рейтрейсинга `rayshade`, и Эрику Вичу, который также не пожалел своего времени и опыта. Спасибо Дагу Шульту и Стэну Эйзенстату за основополагающие уроки математики и информатики в высшей школе и колледже соответственно и, что особенно важно, родителям Мэтта за образование, которому они способствовали, и постоянную поддержку на этом пути. Наконец, спасибо NVIDIA за поддержку подготовки как первого, так и последнего издания книги; в NVIDIA спасибо Нику Триантосу и Джаянту Колхе за их поддержку на заключительных этапах подготовки первого издания, также спасибо Аарону Лефону, Дэвиду Любке и Биллу Далли за их поддержку при работе над четвертым изданием.

Грег Хамфрис очень благодарен всем профессорам и ассистентам, которые терпели его, когда он был студентом Принстона. Многие люди поощряли его интерес к графике, особенно Майкл Коэн, Дэвид Добкин, Адам Финкельштейн, Майкл Кокс, Гордон Столл, Патрик Мин и Дэн Уоллах. Даг Кларк, Стив Лайон и Энди Вульф также руководили досужими независимыми исследованиями, ни разу не рассмеявшись над ними. Однажды на групповом собрании, посвященном годичному проекту робототехники, Стив Лайон пришел в ярость и завопил: «Хватит объяснять мне, почему это невозможно сделать, и придумай, как это сделать!» – импровизированный урок, который мне никогда не забыть. Эрик Ристад выгнал Грега с должности летнего научного сотрудника после его первого года обучения (еще до начала лета), сдав его на руки ничего не подозревающего Пэта Ханрахана и начав консультационные отношения, которые потом продлятся 10 лет и охватят оба побережья. Наконец, Дэйв Хэнсон научил Грега тому, что грамотное программирование – отличный способ работы и что компьютерное программирование может быть красивой и тонкой формой искусства.

Венцель Якоб был в восторге, когда первое издание `pbrt` пришло ему по почте во время учебы в бакалавриате в 2004 г. Излишне говорить, что это оказало большое и продолжительное влияние на его карьеру, поэтому Венцель хотел бы начать с благодарности своим соавторам за приглашение стать частью третьего и четвертого изданий этой книги. Венцель чрезвычайно обязан Стиву Маршнеру, который был консультантом его докторской диссертации и в течение пяти лет работы в Корнеллском университете. Стив привел его в мир исследований и остается постоянным источником вдохновения. Венцель также благодарен за руководство и стимулирующую исследовательскую среду, созданную другими членами графической группы, включая Кавита Бала, Дага Джеймса и Брюса Уолтера. Венцель провел замечательную постдокторскую работу с Ольгой Соркин-Хорнунг, которая познакомила его с процессингом геометрии. Мы глубоко признательны Ольге за участие Венцеля в третьем издании этой книги.

Мы особенно хотели бы поблагодарить рецензентов, прочитавших черновики целиком; все дали проницательные и конструктивные отзывы о рукописи на разных этапах ее проработки. За отзывы о первом и втором изданиях книги выражаем благодарность Яну Эшдауну, Перу Кристенсену, Дагу Эппсу, Дэну Голдману, Эрику Хейнсу, Эрику Рейнхарду, Питу Ширли, Питеру-Пайку Слоану, Грегу Уорду и множеству анонимных рецензентов. За второе издание выражаем благодарность Янне Контканен, Биллу Марку, Нельсону Максу и Эрику Табеллиону. За четвертое издание мы благодарны Томасу Мюллеру и Перу Кристенсену, которые предоставили подробные отзывы, которые значительно улучшили окончательную версию книги.

Многие специалисты любезно объясняли нам тонкости в своей работе и направляли к лучшим практикам. За первое и второе издания мы также благодарны Дону Митчеллу за его помощь в понимании некоторых деталей семплинга и реконструкции; Томасу Коллигу и Александру Келлеру за объяснение тонкостей низкодисперсного семплинга; Кристеру Эриксону, у которого было несколько предложений по улучшению нашей реализации *k-d*-деревьев; а также Кристофу Хери и Юджину д'Эону за помощь в тонкостях подповерхностного рассеяния.

В третьем издании мы особенно хотели бы поблагодарить Лео Грюншлосса за рецензирование нашей главы о семплировании; Александра Келлера за предложения по

темам для этой главы; Эрика Хайтца за обширную помощь в описании микрограней и рецензирование нашего текста по этой теме; Тиаго Изе за тщательный просмотр текста об ошибках округления чисел с плавающей запятой; Тома ван Бассела за сообщение о ряде ошибок в нашем коде BSSRDF; Ральфа Хабела за рецензирование нашего текста BSSRDF; Тошия Хачисука и Антона Капланяна за обширный обзор и комментарии к нашим главам о световом переносе.

В четвертом издании выражаем благодарность Алехандро Конти Эстевесу за рецензию на нашу трактовку семплинга со множеством источников света; Юджину д'Эону, Бэйли Миллеру и Яну Новаку за комментарии к главам, посвященным объемному рассеянию; Эрику Хайнсу, Симону Каллвейту, Мартину Стичу и Карстену Вахтеру за рецензирование главы о рендеринге на GPU; Карлу Ли за отзывы о ряде глав; Цзы-Мао Ли за обзор нашего обсуждения инверсного и дифференцируемого рендеринга; Фабрис Роузелл за отзыв о машинном обучении и рендеринге и Гурприту Сингху за комментарии к нашему обсуждению анализа Фурье интегрирования Монте-Карло. Мы также ценим обширные комментарии и предложения от Йеппе Ревала Фрисвада по поводу трактовки `pbrt`-моделей отражения в предыдущих изданиях.

За улучшения реализации `pbrt` в этом издании мы благодарим Пьера Моро за его усилия по отладке поддержки графического процессора `pbrt` в Windows, а также Джима Прайса, который не только обнаружил и исправил многочисленные ошибки в ранней версии исходного кода `pbrt`, но и внес лучшее представление хроматических объемных сред переноса, чем наша первоначальная реализация. Мы также очень признательны Андерсу Лэнгландсу и Луке Фасьоне из Weta Digital за предоставление реализации их системы *PhysLight*, которая была включена в класс `pbrt PixelSensor`, и реализации источника света.

Многие люди сообщали об ошибках в тексте предыдущих изданий или ошибках в `pbrt`. Мы особенно хотели бы поблагодарить Соломона Булоса, Стивена Ченни, Пера Кристенсена, Джона Дэнкса, Майка Дэя, Кевина Игана, Владимира Качуровского, Костю Смоленского, Ке Сю и Арека Зимни, чьи замечания были особенно полезны.

За предложения и отчеты об ошибках мы также хотели бы поблагодарить Рахита Агравала, Фредерика Акалина, Томаса де Бодта, Марка Болстада, Брайана Баджа, Джонатона Кая, Брайана Катандзаро, Цзы-Чие Чанга, Марка Колберта, Юньцзяна Дина, Тао Ду, Маркоса Фахардо, Шаохуа Фана, Луку Фашионе, Этьена Ферриера, Найджела Фишера, Йеппе Ревалла Фрисвада, Роберта Г. Графа, Асбьёрна Хайда, Стива Хилла, Вэй-Фэн Хуана, Джона «Спайк» Хьюза, Кита Джеффри, Грега Джонсона, Аарона Карпа, Эндрю Кенслера, Алана Кинга, Дональда Кнута, Мартина Крауса, Криса Кулла, Мурата Курта, Ларри Лая, Моргана Макгуайра, Крейга МакНотона, Дона Митчелла, Сваминатана Нараянана, Андерса Нильссона, Йенса Олссона, Винсента Пегораро, Шрината Равичандирана, Энди Селле, Себастьена Шпейерера, Нильса Тюрей, Эрика Вича, Инго Вальда, Зеджиана Вана, Сюн Вей, Вей-Вэй Сюй, Тициана Зельтнера и Матиаса Цвикера. Наконец, мы хотели бы поблагодарить разработчиков *LuxRender* и сообщество *LuxRender*, особенно Терренса Вергаувена, Жана-Филиппа Гримальди и Асбьёрна Хайда; было приятно видеть систему рендеринга, которую они построили на основе `pbrt`, мы узнали нечто новое, читая их исходный код реализаций новых алгоритмов рендеринга.

Особая благодарность Мартину Престону и Стефу Брюнингу из Framestore за помощь в использовании кадра из фильма «*Гравитация*» (изображение предоставлено Warner Bros. и Framestore), а также Weta Digital за помощь с кадром из фильма «*Алита: Боевой ангел*» (© 2018 Twentieth Century Fox Film Corporation, все права защищены).

Производство

За выпуск первого издания мы хотели бы поблагодарить нашего редактора Тима Кокса за его готовность взяться за этот весьма смелый проект, а также за его руководство проектом и терпение на протяжении всего процесса. Мы очень благодарны Элизабет Беллер (менеджер проекта), которая для нашей книги вышла далеко за рамки своего служебного долга; ее способность держать этот сложный проект под контролем и контролировать график его выполнения была замечательной, и мы особенно благодарны ей за ощу-

тимое влияние, которое она оказала на качество конечного результата. Спасибо также Рику Кэмпу (помощнику редактора) за его неоценимый вклад. Пол Анагностопулос и Джеки Скарлотт из Windfall Software разработали композицию книги; мы высоко ценим их способность использовать наш файл в формате грамотного программирования и превращать его в высококачественный конечный результат, а также жонглировать несколькими необычными типами индексации, о которых мы просили. Спасибо Кену ДеллаПента (редактор) и Дженнифер Макклейн (корректор), а также Максу Спектору из Chen Design (дизайнер текста и обложки) и Стиву Рату (индексатор).

За второе издание мы хотели бы поблагодарить Грега Чалсона, который уговорил нас расширить и обновить книгу; Грег также позаботился о том, чтобы Пол Анагностопулос из Windfall Software снова занялся композицией книги. Мы хотели бы еще раз поблагодарить Пола за его усилия по работе во всей сложности при производстве этой книги. Наконец, мы также хотели бы поблагодарить Тодда Грина, Пола Готтерера и Хизер Шерер из Elsevier.

За третье издание мы хотели бы поблагодарить Тодда Грина, который курировал этот этап, и Эми Инверницци, которая держала поезд на рельсах на протяжении всего процесса. Мы были рады, что Пол Анагностопулос из Windfall Software уже в третий раз участвует в этом проекте; его усилия сыграли решающую роль в высокой производственной ценности книги, которая так важна для нас.

В четвертом издании мы перешли в MIT Press; большое спасибо Элизабет Суэйзи за ее энтузиазм, за то, что она способствовала нам в этом, за руководство в производственном процессе и за то, что Пол Анагностопулос снова взялся за композицию книги. Мы искренне благодарим Пола за то, что он вернулся к нам для работы над еще одним изданием, а также большое спасибо Мэри-Эллен Оливер за ее превосходную работу по редактированию и корректуре.

Сцены, модели и данные

Многие люди и организации бесплатно предоставили сцены и модели для использования в этой книге и в дистрибутиве `pbrt`. Их щедрость была неоценима, она помогла нам создать интересные примеры рендеринга изображений на протяжении всей книги.

Мы очень благодарны Гильермо М. Леалу Ллагуно из Evoluciòn Visual, www.evvisual.com, который смоделировал и визуализировал культовую сцену «*Сан-Мигель*», что была изображена на обложке второго издания и до сих пор используется на многочисленных рисунках в книге. Мы также особенно хотели бы поблагодарить Марко Дабровича (www.3lhd.com) и Миховила Одака из RNA Studios (www.rna.hr), которые предоставили множество моделей и сцен, использованных в более ранних изданиях книги, включая атриум дворца Спонца, Шибеникский собор и модель автомобиля Audi TT, которые можно увидеть на рис. 16.1 этого издания.

Мы искренне благодарим Яна-Вальтера Шлипа, Бурака Кахрамана и Тимма Даппера из Laubwerk (www.laubwerk.com) за создание «*Сельского ландшафта*», который был на обложке предыдущего издания книги и используется во многих рисунках в этом издании.

Большое спасибо Анджело Ферретти из Lucydreams (www.lucydreams.it) за лицензию на сцены «*Акварель*» и «*Крокен*», которые использованы для замечательной обложки данного издания, материал для многочисленных рисунков и пару сложных сцен, которые используют возможности `pbrt`.

Джим Прайс любезно предоставил несколько сцен с интересными протяженными средами; они значительно улучшили уровень подачи этой темы. Также спасибо Beeple за предоставление разрешающей лицензии на сцены «*День Ноль*» и «*Прозрачные машины*», и Мартину Любичу за модель короны Австрийской империи. Наконец, мы выражаем нашу глубочайшую благодарность Walt Disney Animation Studios за то, что она сделала доступной сцену «*Остров Моана*», сложную для производства, а также предоставила детальную объемную модель облака.

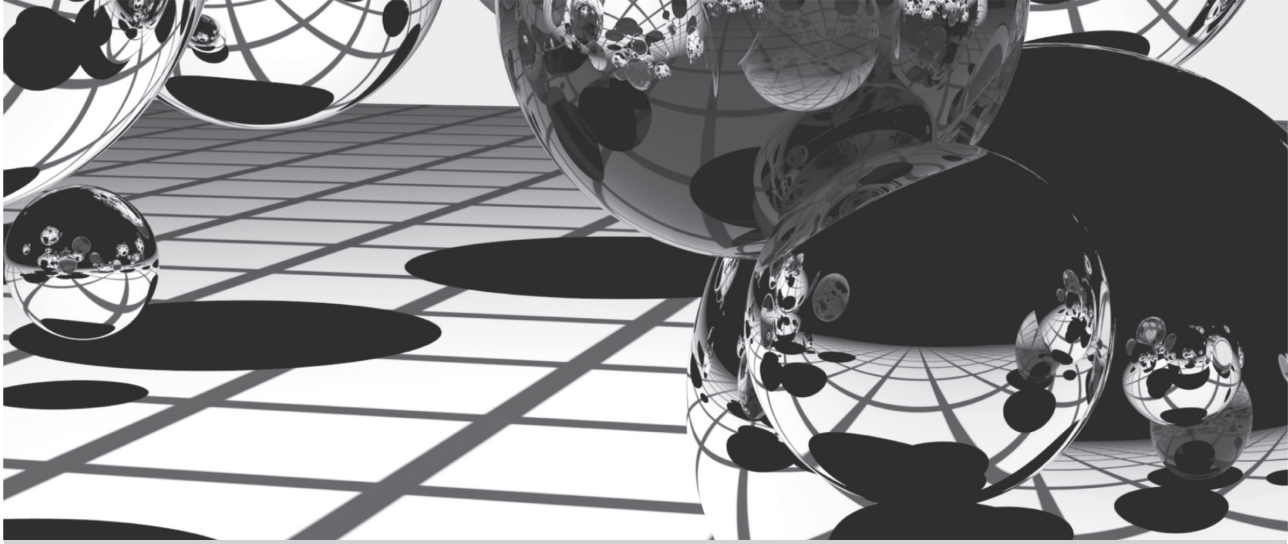
Модели кролика, Будды и дракона предоставлены репозиторием сканирования Стэнфордской лаборатории компьютерной графики. Модель «*Киллеру*» включена с разрешения Фила Денча и Мартина Резарда (3D-сканирование и цифровое представле-

ние – Headus, дизайн и скульптура из глины – Rezard). Отсканированная модель дракона, использованная в главе 9, предоставлена Кристианом Шуллером, и мы благодарим Ясутоши Мори за материальную сферу и модель спортивного автомобиля. Модель головы, используемая для иллюстрации подповерхностного рассеяния, была предоставлена Infinite Realities, Inc. по лицензии Creative Commons Attribution 3.0. Спасибо также «Обезьяне-тирану»[3] за модель автомобиля BMW M6 и «Wig42» за сцену со столом для завтрака; обе были размещены на blendswap.com, также под лицензией Creative Commons Attribution 3.0.

Мы использовали многочисленные карты окружения с веб-сайта *PolyHaven* (polyhaven.com) для HDR-освещения различных сцен; все они доступны по лицензии Creative Commons CC0. Спасибо Сергею Майбороде и Грегу Заалу, карты окружения которых мы применяли.

Марк Элленс предоставил спектральные данные различных источников света; данные спектральных измерений RGB для различных дисплеев предоставлены Томом Лианзой из X-Rite. Мы также благодарим Дэнни Паскаля (www.babelcolor.com) за то, что он позволил нам включить его измерения спектральной отражательной способности цветовой карты. Спасибо Михаилу Полянскому за данные индекса преломления на refractiveindex.info, а также Андерсу Лэнглендсу, Луке Фасьоне и Weta Digital за данные чувствительности световоспринимающего датчика (матрицы) камеры, которые включены в `pbrt`.

[3] Псевдоним автора. – *Прим. ред.*



ГЛАВА ПЕРВАЯ

01 ВВЕДЕНИЕ

Рендеринг – это процесс создания изображения из описания 3D-сцены. Очевидно, что это комплексная задача, и есть много способов приблизиться к ее решению. *Основанные на законах физики* методы пытаются имитировать реальность; то есть они используют законы физики для моделирования взаимодействия света и материи. Хотя физический метод может показаться наиболее очевидным методом к рендерингу, он получил широкое распространение на практике только за последние 15 лет или около того.

В этой книге описывается `pbrt` – физически корректная система рендеринга, основанная на алгоритме рейтрейсинга, или трассировки лучей. Он способен отображать реалистичные изображения сложных сцен, таких как показанная на рис. 1.1. За исключением нескольких исключений в этой главе, которые отмечены их внешним видом, все изображения в этой книге визуализируются с помощью `pbrt`.

В большинстве книг по компьютерной графике представлены алгоритмы и теория, иногда в сочетании с фрагментами кода. Напротив, в этой книге теория сочетается с полной реализацией полнофункциональной системы рендеринга. Кроме того, полный исходный код системы доступен по лицензии с открытым исходным кодом, а полный текст этой книги находится в свободном доступе в интернете по адресу pbr-book.org/4ed с 1 ноября 2023 г. Дополнительную информацию, включая пример сцены и добавочную информацию о `pbrt`, можно найти на веб-сайте pbrt.org.

1.1. ГРАМОТНОЕ ПРОГРАММИРОВАНИЕ

Создавая систему набора текста TEX, Дональд Кнут разработал новую методологию программирования, основанную на простой, но революционной идее. Цитируем Кнута: «давайте изменим наше традиционное отношение к построению программ: вместо того чтобы воображать, что наша главная задача – инструктировать компьютер, что делать, давайте лучше сосредоточимся на объяснении людям того, что мы хотим получить от компьютера, что он должен сделать для нас. Он назвал эту методологию *грамотным программированием* (или *литературным программированием*). Эта книга (включая главу, которую вы сейчас читаете) представляет собой длинную «грамотную программу». Это означает, что в ходе чтения данной книги вы прочтете полную реализацию системы рендеринга `pbrt`, а не только ее высокоуровневое описание.

Грамотные программы пишутся на метаязыке, который смешивает язык форматирования документов (например, TEX или HTML) и язык программирования (например,

C++). Программу обрабатывают две отдельные системы: «weaver», которая преобразует грамотную программу в документ, пригодный для набора, и «tangler», создающая исходный код, пригодный для компиляции. Используемая в этой книге система грамотного программирования – наша собственная разработка, но на нее сильно повлияла система `noweb` Нормана Рэмси.

Рис. 1.1. Сцена, визуализированная с помощью `pbrt`. Сцена «*Крокен*» обладает сложной геометрией, материалами и световым переносом. Хорошая обработка всех этих эффектов в системе рендеринга делает возможным рендеринг фотореалистичных изображений, подобных представленному. Эту сцену и многие другие можно скачать с сайта `pbrt` (*сцена предоставлена Анджело Ферретти*)

Грамотный метаязык программирования обладает двумя важными свойствами. Во-первых, это возможность смешивать литературный текст с исходным кодом. Эта функция ставит описание программы в один ряд с ее исходным кодом, поощряя тщательную разработку и документирование. Во-вторых, язык дает механизмы для представления программного кода читателю в порядке, полностью отличающемся от ввода компилятора. Таким образом, программа может быть описана логическим образом. Каждый именованный блок кода называется *фрагментом*, и каждый фрагмент может ссылаться на другие фрагменты по имени.

В качестве простого примера рассмотрим функцию `InitGlobals()`, которая отвечает за инициализацию всех глобальных переменных программы[1]:

```
void InitGlobals() {
    nMarbles = 25.7;
    shoeSize = 13;
    dielectric = true;
}
```

Несмотря на лаконичность, эту функцию трудно понять без контекста. Почему, например, переменная `nMarbles` может принимать значения с плавающей запятой? Просто взглянув на код, нужно было бы просмотреть всю программу, чтобы увидеть, где объявлена каждая переменная и как она используется, чтобы понять ее назначение и значения ее допустимых значений. Хотя такая структура системы удобна для компилятора, читатель-человек предпочел бы видеть код инициализации для каждой переменной, представленный отдельно, рядом с кодом, который объявляет и использует переменную.

В грамотной программе вместо этого можно написать `InitGlobals()` следующим образом:

⟨*Function Definitions*⟩ ≡
```
void InitGlobals() {
    ⟨Initialize Global Variables⟩
}
```

Это определяет фрагмент, названный ⟨*Function Definitions*⟩, который содержит определение функции `InitGlobals()`. Сама функция `InitGlobals()` ссылается на другой фрагмент ⟨*Initialize Global Variables*⟩. Поскольку фрагмент инициализации еще не определен, мы ничего не знаем об этой функции, кроме того что она предположительно будет содержать значения глобальных переменных.

Само наличие имени фрагмента – правильный уровень абстракции на данный момент, поскольку переменные еще не объявлены. Когда мы введем глобальную переменную `shoesSize` где-нибудь позже в программе, мы сможем написать

⟨*Initialize Global Variables*⟩ ≡
```
shoeSize = 13;
```

Здесь мы начали определять содержимое ⟨*Initialize Global Variables*⟩. Когда грамотная программа запутается в исходном коде для компиляции, грамотная система программирования подставит код `shoesSize = 13;` внутри определения функции `InitGlobals()`.

Далее по тексту мы можем определить еще одну глобальную переменную, `dielectric`, и можем добавить ее инициализацию к фрагменту:

⟨*Initialize Global Variables*⟩ +≡
```
dielectric = true;
```

Символ +≡ после имени фрагмента показывает, что мы добавили к ранее определенному фрагменту.

При запутывании (tangling) эти три фрагмента превращаются в код

```
void InitGlobals() {
    // Initialize Global Variables
    shoeSize = 13;
    dielectric = true;
}
```

[1] Пример кода в этом разделе носит исключительно иллюстративный характер и не является частью самой `pbrt`. – *Прим. авт.*

Таким образом, мы можем разложить сложные функции на логически обособленные части, что значительно упростит их понимание. Например, мы можем написать сложную функцию в виде набора фрагментов:

```
⟨Function Definitions⟩ +≡
   void complexFunc(int x, int y, double *values) {
       ⟨Check validity of arguments⟩
       if (x < y){
           ⟨Swap x and y⟩
       }
       ⟨Do precomputation before loop⟩
       ⟨Loop through and update values array⟩
   }
```

Опять же, содержимое каждого фрагмента расширяется в строке `complexFunc()` для компиляции. В документе мы можем представить каждый фрагмент и его реализацию по очереди. Это разделение позволяет нам представлять код по нескольку строк за один раз, что упрощает его понимание. Еще одним преимуществом данного стиля программирования является то, что путем разделения функции на логические фрагменты, каждый из которых имеет одну и четко очерченную цель, каждый из них может быть написан, проверен или прочитан независимо. В общем, мы постараемся, чтобы каждый фрагмент был не длиннее 10 строк.

В общем смысле грамотная система программирования – это всего лишь расширенный пакет подстановки макросов, настроенный на задачу перегруппировки исходного кода программы. Это может показаться тривиальным изменением, но на самом деле грамотное программирование сильно отличается от других способов структурирования программных систем.

1.2. ФОТОРЕАЛИСТИЧЕСКИЙ РЕНДЕРИНГ И АЛГОРИТМ РЕЙТРЕЙСИНГА

Целью фотореалистичного рендеринга является создание изображения трехмерной сцены, *не отличимого* от фотографии той же сцены. Прежде чем мы опишем процесс рендеринга, важно понять, что в данном контексте слово «неотличимый» является неточным, поскольку оно ссылается на человека-наблюдателя, а разные наблюдатели могут воспринимать одно и то же изображение по-разному. Хотя в этой книге мы рассмотрим некоторые вопросы восприятия, учет точных характеристик конкретного наблюдателя – трудная и не до конца решенная проблема. По большей части мы будем удовлетворены точным моделированием физики света и его взаимодействия с материей, полагаясь на наше понимание технологии отображения, чтобы представить зрителю изображение в лучшем возможном виде.

Ввиду этого целенаправленного акцента на реалистичности моделирования распространения света кажется благоразумным спросить: *а что такое свет*? Восприятие окружающего мира через посредство света играет центральную роль в самом нашем существовании, и поэтому этот простой вопрос занимал умы знаменитых философов и физиков с незапамятных времен. Древнеиндийская философская школа Вайшешика (V–VI вв. до н.э.) рассматривала свет как совокупность мелких частиц, движущихся по лучам с большой скоростью. В V в. до н. э. греческий философ Эмпедокл постулировал, что божественный огонь выходит из человеческих глаз и в сочетании со световыми лучами солнца дает зрение. Между XVIII и XIX веками ученые, такие как Исаак Ньютон, Томас Янг и Огюстен-Жан Френель, поддерживали противоречивые теории, моделирующие свет как следствие распространения волн или частиц. В тот же период времени Андре-Мари Ампер, Жозеф-Луи Лагранж, Карл Фридрих Гаус и Майкл Фарадей исследовали отношения между электричеством и магнетизмом, кульминацией которых стало внезапное и драматическое объединение Джеймсом Клерком Максвеллом всей суммы знаний в единую теорию, которая теперь известна как *электромагнетизм*.

С этой точки зрения свет является волновым явлением: движение электрически заряженных частиц, таких как электроны, в нити накаливания лампочки вызывает возмущение в виде *электрического поля*, которое распространяется от источника. Колебания напряженности электрического поля вызывают вторичные колебания напряженности *магнитного поля*, которые, в свою очередь, вызывают колебания электрического поля, и т. д. Взаимодействие этих двух полей приводит к самораспространяющейся волне, которая может распространяться на чрезвычайно большие расстояния, даже на миллионы световых лет, – на ясном ночном небе видны очень далекие звезды. В начале XX в. работы Макса Планка, Макса Борна, Эрвина Шредингера и Вернера Гейзенберга привели к еще одному качественному сдвигу в нашем понимании: на микроскопическом уровне элементарные свойства, такие как энергия и импульс, квантуются, что означает, что они могут существать только как целое число, кратное базовой сумме, известной как *квант*. В случае электромагнитных колебаний этот квант называется *фотоном*. В этом смысле наше физическое понимание совершило полный круг: как только мы обращаемся к очень малым масштабам, свет снова проявляет поведение, подобное частице, которое сосуществует с его общей волновой природой[2].

Как наша цель – симуляция света для создания реалистичных изображений – вписывается во все это? Столкнувшись с этой горой все более изощренных объяснений, возникает фундаментальный вопрос: как далеко мы должны забраться на эту гору, чтобы достичь фотореализма? К нашему большому счастью, ответ оказывается «совсем недалеко». Длина волн, составляющих видимый свет, чрезвычайно мала, их период равен всего лишь нескольким сотням нанометров. Сложность поведения света в этих малых масштабах имеет место, но она не имеет большого значения при моделировании объектов в масштабе, скажем, сантиметров или метров. Это отличная новость, потому что детальное моделирование волн размером более нескольких микрометров нецелесообразно: компьютерная графика не существовала бы в ее нынешнем виде, если бы такой уровень детализации был необходим для рендеринга изображений. Вместо этого мы будем в основном иметь дело с уравнениями, разработанными между XVI и началом XIX в., которые моделируют свет как частицы, движущиеся вдоль лучей. Это приводит к более эффективному вычислительному методу, основанному на главной операции, известной как *рейтрейсинг* (ray tracing, или «трассировка лучей»).

Рейтрейсинг – это концептуально простой алгоритм; он основан на отслеживании пути луча света через сцену, когда он взаимодействует с объектами в окружающей среде, поглощается ими и отражается от них. Хотя существует много способов написать трассировщик лучей, все такие системы имитируют как минимум следующие объекты и явления:

- *камеры*. Модель камеры определяет, как и откуда просматривается сцена, в том числе то, как изображение сцены записывается на матрицу камеры. Многие системы рендеринга генерируют лучи обзора не от источника света[3], а от камеры, которые затем прослеживаются в сцене, чтобы определить, какие объекты видны в каждом направлении;
- *пересечение луча и объекта*. Мы должны быть в состоянии точно сказать, где данный луч пересекается с геометрическим объектом сцены («натыкается» на него). Кроме того, нам необходимо определить некоторые свойства объекта в точке пересечения, такие как нормаль к поверхности или ее материал. Большинство рейтрейсеров также имеют средства для проверки пересечения луча с несколькими объектами, принимая во внимание ближайшее пересечение с объектом по направлению распространения луча;
- *источники света*. Без освещения сцены не было бы никакого смысла в ее рендеринге. Рейтрейсер должен моделировать распределение света по всей сцене, включая не только расположение самих источников света, но и то, как излучаемая ими энергия распространяется в пространстве;

---

[2] В физике этот феномен называют «корпускулярно-волновой дуализм». – *Прим. ред.*

[3] В дальнейшем для краткости наряду с термином «источник света» будет применяться только «свет», где очевидно, что речь идет об источнике, а не о луче. – *Прим. ред.*

- *видимость*. Чтобы узнать, доставляет ли данный источник света энергию к точке на поверхности, мы должны знать, существует ли непрерывный путь от этой точки до источника света. К счастью, на этот вопрос легко ответить в трассировщике лучей, поскольку мы можем просто построить луч от поверхности к источнику света, найти ближайшее пересечение луча и объекта и сравнить расстояние пересечения с расстоянием до источника света;
- *рассеяние света на поверхностях*. Каждый объект должен содержать описание своего внешнего вида, включая информацию о том, как свет взаимодействует с поверхностью объекта, а также о характере переизлучаемого (или рассеянного) света. Модели поверхностного рассеяния определяются таким образом, чтобы они могли имитировать параметры взаимодействия со светом различных поверхностей;
- *непрямое распространение света*. Поскольку свет может попасть на поверхность после отражения от других поверхностей или прохождения сквозь них, необходимо проследить дополнительные лучи, чтобы учесть этот эффект;
- *распространение лучей*. Нам нужно знать, что происходит со светом, движущимся вдоль луча, когда он проходит сквозь пространство. Если мы рассчитываем сцену в вакууме, световая энергия вдоль луча остается постоянной. Хотя настоящий вакуум необычен для Земли, он является разумным приближением для многих сред. Доступны более сложные модели для трассировки лучей сквозь туман, дым, атмосферу Земли и т. д.

В данном разделе мы кратко обсудили каждую из этих задач моделирования. В следующем разделе мы покажем высокоуровневый интерфейс `pbrt` для базовых компонент моделирования и представим простой алгоритм рендеринга, который случайным образом выбирает пути света через сцену для создания изображений.

1.2.1. Камеры и пленка

Практически каждый пользовался фотоаппаратом и знаком с его основными функциями: вы заявляете о своем желании зафиксировать изображение мира (обычно нажатием кнопки или касанием экрана), и изображение экспонируется на пленку или на электронную матрицу камеры[4].

Одним из самых простых устройств для фотосъемки является *камера-обскура*. Камера-обскура представляет собой светонепроницаемый бокс с крошечным отверстием на одной из стенок (рис. 1.2). Когда отверстие открыто, свет входит и падает на лист фотобумаги, прикрепленный к задней поверхности бокса. Несмотря на свою простоту, этот тип камеры все еще используется сегодня, в основном в художественных целях. Длительное время выдержки необходимо, чтобы на пленку попало достаточно света для формирования изображения.

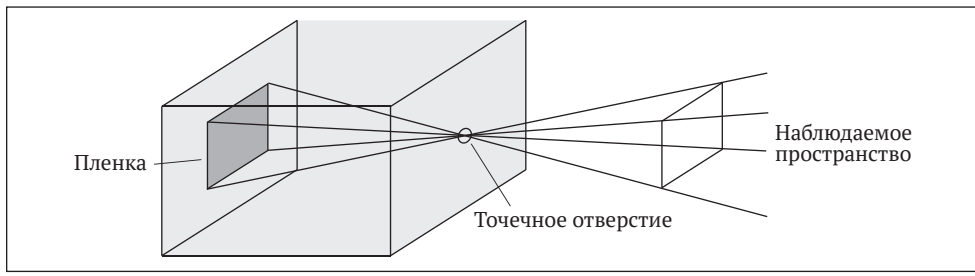


Рис. 1.2. Камера-обскура. Поле зрения определяется проекцией пленки через точечное отверстие

[4] Хотя цифровые матрицы в настоящее время более распространены, чем физическая пленка, мы будем использовать слово «пленка» для обозначения обеих в тех случаях, когда можно использовать любую из них. – *Прим. авт.*

Хотя большинство камер существенно сложнее, чем камера-обскура, она представляет собой удобную отправную точку для моделирования. Наиболее важной функцией камеры является оценка части сцены, которая будет зафиксирована на пленку. На рис. 1.2 мы можем видеть, как соединение отверстия с краями пленки создает двойную пирамиду, уходящую вглубь сцены. Объекты, не находящиеся внутри этой пирамиды, не могут быть отображены на пленке. Поскольку настоящие камеры отображают более сложную форму, чем пирамида, мы будем называть область пространства, которая потенциально может быть отображена на пленке, *полем зрения*.

Еще один способ представить камеру-обскуру – поместить плоскость пленки перед отверстием, но на том же расстоянии (рис. 1.3). Обратите внимание, что соединение отверстия с пленкой определяет точно такое же поле зрения, как и раньше. Конечно, это непрактичный способ построить реальную камеру, но для целей моделирования это удобное приближение. Когда плоскость пленки (или изображения) находится перед отверстием, отверстие часто называют *глазом*.

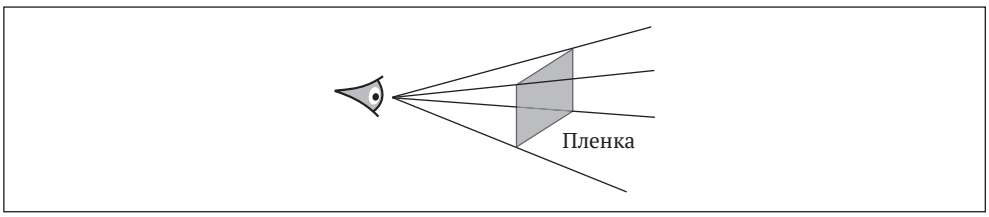


Рис. 1.3. Когда мы симулируем камеру-обскуру, то помещаем пленку перед отверстием в плоскости изображения, и отверстие переименовывается в *глаз*

Теперь мы переходим к главному вопросу рендеринга: какой цвет записывает камера в каждой точке изображения? Ответ на этот вопрос частично определяется тем, какая часть сцены видна в данный момент. Если мы вспомним оригинальную камеру-обскуру, станет ясно, что только лучи света, которые проходят вдоль вектора между обскурой и точкой на пленке, могут внести свой вклад в это положение пленки. В нашей смоделированной камере с плоскостью пленки перед глазом нас интересует количество света, проходящего от точки изображения к глазу.

Таким образом, важной задачей симулятора камеры является получение точки на изображении и создание *лучей*, идущий вдоль которых свет создаст изображение. Поскольку луч включает в себя исходную точку и вектор направления, эта задача для модели камеры-обскуры особенно проста; на рис. 1.3 у нас есть точечное отверстие в качестве начала координат и вектор от точечного отверстия к плоскости изображения, что есть направление луча. Для более сложных моделей камер с несколькими линзами расчет луча, соответствующего заданной точке изображения, может быть более сложным.

Свет, входящий в камеру по направлению луча, обычно несет различное количество энергии на разных длинах волн. Зрительная система человека интерпретирует эту совокупность как цвет. Большинство матриц камер записывают измерения интенсивностей трех длин волн света, соответствующих красному, зеленому и синему цветам отдельно, чего достаточно, чтобы реконструировать визуальный образ сцены для человека-наблюдателя. В разделе 4.6 цвет обсуждается более подробно. Поэтому камеры в `pbrt` также предоставляют симуляцию пленки, которая одновременно сохраняет изображение и моделирует реакцию пленочного сенсора на падающий свет.

Камера `pbrt` и симуляция пленки подробно описаны в главе 5. Процесс преобразования точек изображения в лучи, инкапсулированный в модуле камеры, и симуляция пленки, отвечающая за определение реакции сенсора на свет, позволяют остальной части системы рендеринга сосредоточиться на оценке освещения вдоль пути распространения этих лучей.

1.2.2. Пересечения луч–объект

Каждый раз, когда камера генерирует луч, первая задача визуализатора – определить, какой объект, если таковой имеется, этот луч пересекает первым и где происходит пересечение. Эта точка пересечения является видимой точкой вдоль луча, и нам нужно смоделировать взаимодействие света с объектом в этой точке. Чтобы найти пересечение, мы должны протестировать луч на пересечение со всеми объектами сцены и выбрать тот, который луч пересекает первым. Имея луч r, мы сначала запишем его в *параметрической форме*:

$$r(t) = \mathrm{o} + t\mathbf{d},$$

где o – начало луча (origin), $\mathbf{d}$ – вектор его направления, t – параметр, допустимый диапазон которого равен $[0, \infty)$. Мы можем получить точку вдоль луча, указав ее параметрическое значение t и оценив приведенное выше уравнение.

Часто найти пересечение между лучом r и поверхностью, определяемой неявной функцией $F(x, y, z) = 0$, бывает легко. Сначала мы подставляем уравнение луча в неявное уравнение, задавая новую функцию, единственным параметром которой является t. Затем мы решаем эту функцию для t и подставляем наименьший положительный корень в уравнение луча, чтобы найти нужную точку. Например, неявное уравнение сферы с центром в начале координат и радиусом r имеет вид

$$x^2 + y^2 + z^2 - r^2 = 0.$$

Подставляя уравнение луча, имеем

$$(\mathrm{o}_x + t\mathbf{d}_x)^2 + (\mathrm{o}_y + t\mathbf{d}_y)^2 + (\mathrm{o}_z + t\mathbf{d}_z)^2 - r^2 = 0,$$

где нижние индексы обозначают соответствующую компоненту точки или вектора. Для данного луча и данной сферы все значения, кроме t, известны, что дает нам легкорешаемое квадратное уравнение относительно t. Если действительных корней нет, луч проходит мимо сферы; если есть корни, наименьший положительный дает точку пересечения.

Одной точки пересечения недостаточно для остальной части трассировщика лучей; ему нужно знать определенные свойства поверхности в точке. Во-первых, необходимо определить представление материала в точке и передать его на более поздние этапы алгоритма рейтрейсинга.

Рис. 1.4. Сцена острова Моана, визуализированная `pbrt`. Эта модель из художественного фильма демонстрирует исключительную сложность сцен, отрисованных для фильмов (Walt Disney Animation Studios, 2018). В ней представлено более 146 млн уникальных треугольников, хотя истинная геометрическая сложность сцены достигает десятков миллиардов треугольников из-за широкого использования экземплярирования объектов[5] (*сцена предоставлена анимационной студией Уолта Диснея*)

[5] Экземплярирование – это когда мы берем объект или группу объектов и ссылаемся на них несколько раз. – *Прим. ред.*

Во-вторых, потребуется дополнительная геометрическая информация о точке пересечения, чтобы присвоить точке цвет. Например, всегда требуется нормаль к поверхности **n**. Хотя многие рейтрейсеры работают только с **n**, более сложные системы рендеринга, такие как `pbrt`, требуют еще больше информации, такой как различные частные производные положения и нормали к поверхности относительно локальной параметризации поверхности.

Конечно, большинство сцен содержат несколько объектов. Метод грубой силы будет состоять в том, чтобы по очереди проверять луч на каждом объекте, выбирая минимальное положительное значение t всех пересечений, дабы найти ближайшее пересечение. Этот метод хоть и правильный, но очень медленный даже для сцен небольшой сложности. Лучшим методом является включение *структуры ускорения*, которая быстро отбрасывает целые группы объектов в процессе определения пересечений лучей. Эта способность быстро отбрасывать ненужные объекты означает, что рейтрейсинг часто выполняется за время $O(m \log n)$, где m – количество пикселей в изображении, а n – количество объектов в сцене[6]. Построение самой структуры ускорения займет не менее $O(n)$ времени. Благодаря эффективности структур ускорения можно визуализировать очень сложные сцены, такие как показанная на рис. 1.4, за разумное время.

Геометрический интерфейс `pbrt` и его реализации для различных форм описаны в главе 6, а интерфейс и реализации ускорения показаны в главе 7.

1.2.3. Распределение света

Этап пересечения луча и объекта дает нам точку для шейдинга и некоторую информацию о локальной геометрии в этой точке. Напомним, что наша конечная цель – найти количество света, покидающего эту точку в направлении камеры. Для этого нам нужно знать, сколько света попадает в эту точку. Это включает в себя как *геометрическое*, так и *радиометрическое* распределение света в сцене. Для очень простых источников света (например, точечных источников света) геометрическое распределение света – это просто вопрос знания положения источниках света. Однако в реальном мире точечных источников света не существует, поэтому физически обоснованное освещение часто основано на *протяженных* источников света. Это означает, что источник света представляет из себя некую геометрическую форму, излучающую свет со своей поверхности[7]. Однако в этом разделе мы рассмотрим точечные источники света, чтобы проиллюстрировать компоненты распределения света; более тщательное обсуждение измерения и распределения света является темой глав 4 и 12.

Нам часто нужно знать световой поток, падающий на дифференциал площади, окружающий точку пересечения `p` (рис. 1.5). Предположим, что точечный источник света имеет некоторую связанную с ним мощность Φ и что он излучает свет одинаково во всех направлениях. Это означает, что мощность на единицу площади сферы, окружающей источник света, составляет $\Phi/(4\pi)$. Эти измерения будут объяснены и формализованы в разделе 4.1.

Если мы рассмотрим две такие сферы (рис. 1.6), то станет ясно, что мощность на единицу площади в точке на большей сфере должна быть меньше мощности в точке на

---

[6] Хотя логарифмическая сложность трассировки лучей часто считается одной из ее сильных сторон, обычно эта сложность верна только в среднем. В литературе по вычислительной геометрии опубликован ряд алгоритмов рейтрейсинга с гарантированным логарифмическим временем работы, но эти алгоритмы работают только для определенных типов сцен и требуют очень больших затрат на предварительную обработку и хранение. Симей-Калос и Мартон дают ссылки на соответствующую литературу (Szirmay-Kalos and Marton, 1998). На практике алгоритмы пересечения лучей, представленные в этой книге, являются сублинейными, но без затратной предварительной обработки и использования большого объема памяти всегда можно построить сцены с наихудшим случаем, в которых рейтрейсинг выполняется за время $O(mn)$. Одним из утешений является то, что сцены, представляющие реалистичную среду, обычно не демонстрируют такого наихудшего поведения. – *Прим. авт.*

[7] И не обязательно только с поверхности, он может иметь определенную, возможно неоднородную, прозрачность и внутреннее распределение яркости по объему. – *Прим. ред.*

меньшей сфере, поскольку та же самая суммарная мощность распределяется по большей площади. В частности, мощность на единицу площади, поступающая в точку на сфере радиусом r, пропорциональна $1/r^2$.

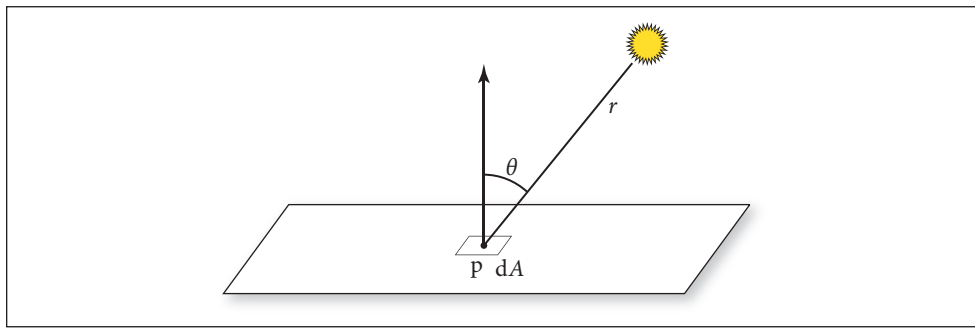


Рис. 1.5. Геометрическая схема для определения величины светового потока (мощности света) на единицу площади, поступающего в точку p от точечного источника света. Расстояние от точки до источника света обозначается как r

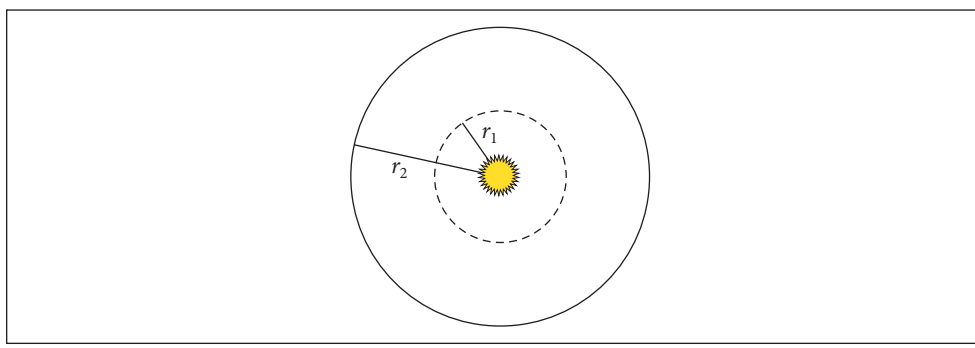


Рис. 1.6. Поскольку точечный источник света излучает свет одинаково во всех направлениях, световой поток распределяется по площади всех сфер с центром в источнике света равномерно

Кроме того, можно показать, что если крошечный участок поверхности dA наклонен на угол θ с вектором от точки поверхности к свету и количество энергии, выделяемой на dA, пропорционально cosθ, с учетом всего этого дифференциальная мощность на площадь dE (*дифференциальная облученность*) равна

$$\mathrm{d}E = \frac{\Phi \cos\theta}{4\pi r^2}.$$

Читатели, уже знакомые с основами освещения в компьютерной графике, заметят два знакомых закона, заложенных в этом уравнении: косинусное уменьшение освещенности в зависимости от наклона поверхности, упомянутое выше, и квадратичное затухание света с расстоянием (r^{-2}).

Сцены с несколькими источниками света легко обрабатываются, поскольку суммирование освещения *линейно*: вклад каждого источника света можно рассчитать отдельно и суммировать для получения общего вклада[8]. Следствием линейности света явля-

[8] Принцип суперпозиции. – *Прим. ред.*

ется то, что сложные алгоритмы могут применяться для случайной схемы освещения только от некоторых источников света в каждой точке сцены, для которой делается шейдинг; это тема раздела 12.6. На рис. 1.7 показана сцена с тысячами источников света, визуализированных таким образом.

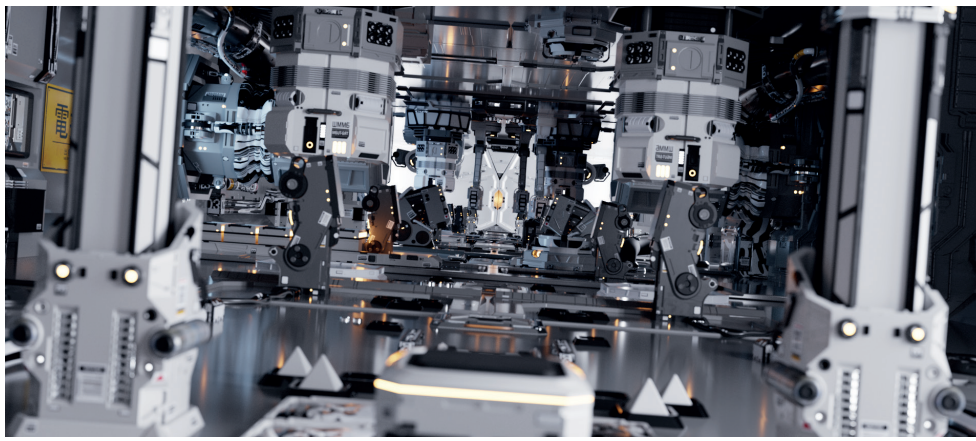


Рис. 1.7. Сцена с тысячами источников света. В этой сцене слишком много источников света, чтобы учитывать их все в каждой точке, где вычисляется отраженный свет. Тем не менее его можно эффективно визуализировать, используя стохастический семплинг источников света (*сцена предоставлена Beeple*)

1.2.4. Видимость

Распределение освещения, описанное в предыдущем разделе, игнорирует один очень важный компонент: *тени*. Каждый источник света способствует освещению точки только в том случае, если на пути от точки до положения источника света нет препятствий в виде других объектов (рис. 1.8).

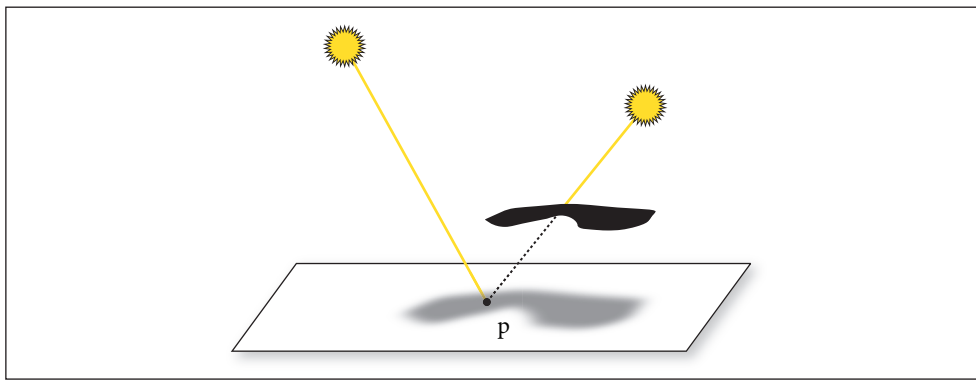


Рис. 1.8. Источник света излучает энергию на поверхность только в том случае, если источник не перекрыт, если смотреть с точки регистрации (камеры). Источник света слева освещает точку p, а источник света справа – нет

К счастью, в рейтрейсере легко определить, виден ли источник света из затеняемой точки. Мы просто строим новый луч, начало которого находится в точке поверхности,

а направление указывает на источник света. Эти особые лучи называются *теневыми лучами*. Если мы проследим этот луч через окружающую среду, мы можем проверить, найдены ли какие-либо пересечения между источником луча и источником света, сравнив параметрическое значение t любых найденных пересечений с параметрическим значением t вдоль луча с положения источника света. Если между источником света и поверхностью нет блокирующего объекта, вклад источника света учитывается.

1.2.5. Поверхности, рассеивающие свет

Теперь мы можем вычислить два вида информации, жизненно важные для правильного шейдинга (шейдинга) точки: ее местоположение и падающее освещение. Сейчас нам нужно определить, каким образом падающее освещение *рассеивается* на поверхности. В частности, нас интересует количество световой энергии, рассеянной назад вдоль луча, который мы первоначально проследили, чтобы найти точку пересечения, так как этот луч ведет к камере (рис. 1.9).

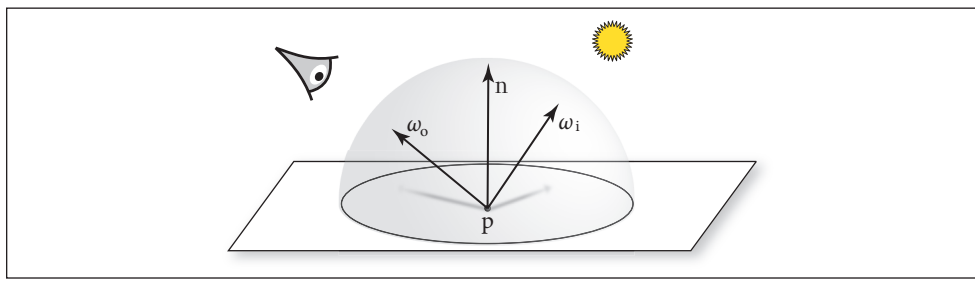


Рис. 1.9. Геометрия поверхностного рассеяния. Падающий свет, идущий по направлению ω_i, взаимодействует с поверхностью в точке p и рассеивается обратно к камере по направлению ω_o. Количество света, рассеянного по направлению к камере, определяется произведением энергии падающего света на BRDF

Каждый объект в сцене предполагает *материал*, который является описанием свойств внешнего вида объекта в каждой точке его поверхности. Это описание дается *двунаправленной функцией распределения отражательной способности* (BRDF). Эта функция говорит нам, сколько энергии отражается от входящего направления ω_i в исходящее направление ω_o. Мы запишем BRDF в точке p как $f_r(p, \omega_o, \omega_i)$. По соглашению, направления ω являются единичными векторами.

Легко обобщить понятие BRDF на проходящий свет (получение BTDF) или на общее рассеяние света, поступающего с любой стороны поверхности. Функция, описывающая общее рассеяние, называется *функцией распределения двунаправленного рассеяния* (BSDF). Система `pbrt` поддерживает множество моделей BSDF; они описаны в главе 9. Еще более сложной является *двунаправленная функция распределения отражения от рассеивающей поверхности* (BSSRDF), которая моделирует свет, выходящий из поверхности в другой точке, чем он входит. Это необходимо для воспроизведения полупрозрачных материалов, таких как молоко, мрамор или кожа. BSSRDF описан в разделе 4.3.2. На рис. 1.10 показано изображение, полученное с помощью `pbrt` на основе модели головы человека, где рассеяние от кожи моделируется с помощью BSSRDF.

1.2.6. Непрямое пропускание света

В оригинальной статье Тернера Уиттеда о рейтрейсинге (1980) подчеркивалась его *рекурсивная* природа, которая была ключом, позволившим включить непрямое зеркальное отражение и пропускание в визуализированные изображения. Например, если луч от камеры попадает на блестящий объект, такой как зеркало, мы можем отразить луч относительно нормали к поверхности в точке пересечения и рекурсивно вызвать про-

цедуру трассировки лучей, чтобы найти свет, попадающий в точку на зеркале, делая свой вклад в исходный луч камеры. Этот же метод можно использовать для трассировки лучей, прошедших сквозь прозрачные объекты. Многие ранние примеры рейтрейсинга демонстрировали зеркала и стеклянные шары (рис. 1.11), потому что эти типы эффектов было трудно передать с помощью других методов рендеринга.

Рис. 1.10. Голова человека с рассеянием, смоделированным с помощью BSSRDF. Точное моделирование подповерхностного светового переноса взамен предположения, что свет выходит из поверхности в той же точке, в которой он вошел, значительно повышает реалистичность визуализируемого изображения (*модель предоставлена Infinite Realities, Inc.*)

Как правило, количество света, попадающего в камеру из точки на объекте, определяется суммой света, излучаемого объектом (если он сам является источником света), и количеством отраженного света. Эта идея формализована *уравнением светового переноса* (также часто известным как *уравнение рендеринга*), которое измеряет свет по отношению к яркости, радиометрической единице, которая будет определена в разделе 4.1. Он говорит, что исходящая яркость $L_o(p, \omega_o)$ из точки p в направлении ω_o – излучаемая яркость в этой точке в этом направлении, $L_e(p, \omega_o)$, плюс падающая яркость со всех направлений на сфере $\mathcal{S}^2$ вокруг p, определяемая BSDF $f_r(p, \omega_o, \omega_i)$ и косинусным членом:

$$L_o(p, \omega_o) = L_e(p, \omega_o) + \int_{\mathcal{S}^2} f(p, \omega_o, \omega_i) L_i(p, \omega_i) |\cos \theta_i| d\omega_i. \quad (1.1)$$

Мы покажем более полный вывод этого уравнения в разделах 4.3.1 и 13.1.1. Аналитическое решение данного интеграла невозможно, за исключением самых простых сцен, поэтому мы должны либо сделать упрощающие предположения, либо использовать численные методы интегрирования.

Алгоритм рейтрейсинга Уиттеда упрощает этот интеграл, игнорируя входящий свет с большинства направлений и оценивая $L_i(p, \omega_i)$ только для направлений на источники света и для направлений идеального отражения и преломления. Другими словами, он превращает интеграл в сумму по небольшому числу направлений. В разделе 1.3.6 мы увидим, что простой рандомный семплинг уравнения 1.1 может создавать реалистичные изображения, включающие как сложное освещение, так и сложные эффекты

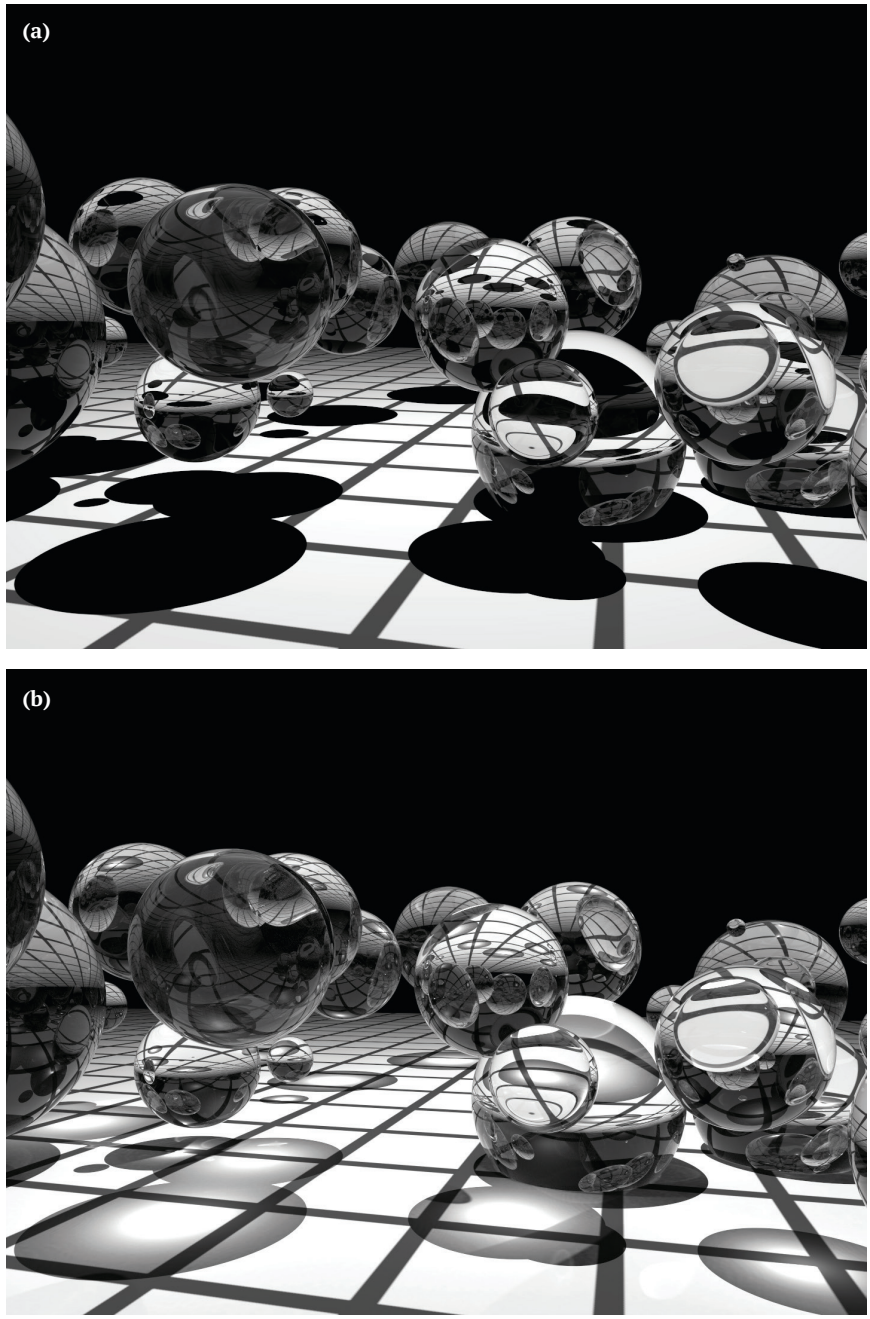


Рис. 1.11. Прототип сцены для рейтрейсинга ранних лет его разработки. Обратите внимание на использование зеркальных и стеклянных объектов, что подчеркивает способность алгоритма обрабатывать такие поверхности: (a) рендеринг с использованием оригинального алгоритма рейтрейсинга Уиттеда 1980 года и (b) рендеринг с использованием *стохастического прогрессивного фотонного маппинга* (SPPM), современного усовершенствованного алгоритма светового переноса. SPPM может точно имитировать фокусировку света, проходящего через сферы

поверхностного рассеяния. В оставшейся части книги мы покажем, как использование более сложных алгоритмов рандомного семплинга значительно повышает эффективность этого общего метода.

1.2.7. Распространение луча

До сих пор дискуссия предполагала, что лучи проходят через вакуум. Например, при описании распределения света от точечного источника мы предполагали, что мощность света равномерно распределяется по поверхности сферы с центром в источнике света, не уменьшаясь по пути. Присутствие *передающих сред* (сред распространения), таких как дым, туман или пыль, может сделать это предположение неверным. Эти эффекты важно смоделировать – с помощью передающих сред можно описать широкий класс интересных явлений. На рис. 1.12 показан взрыв, визуализированный с помощью `pbrt`. Почти на все сцены под открытым небом, хоть и не так драматично, существенное влияние оказывают передающие среды. Например, атмосфера Земли заставляет объекты, которые находятся дальше, казаться менее насыщенными по цвету.

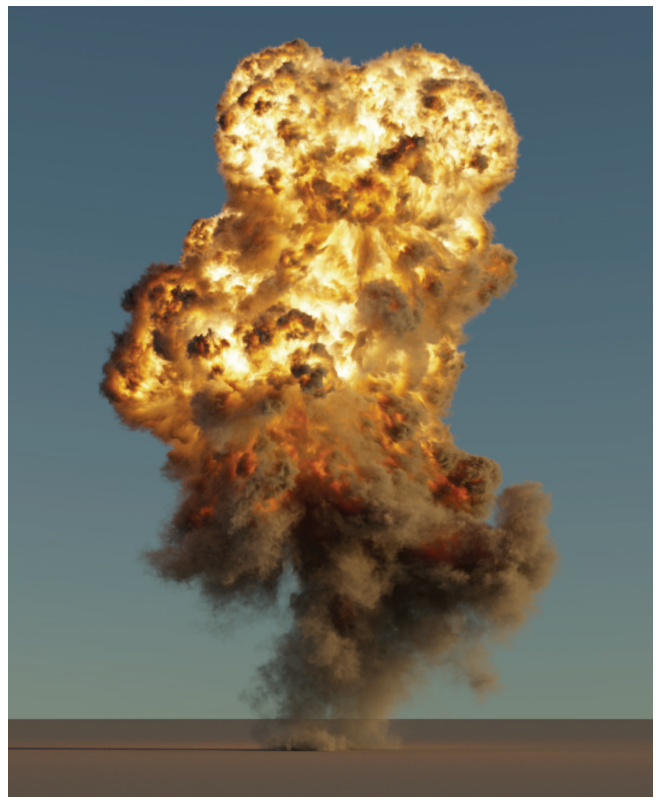


Рис. 1.12. Взрыв, смоделированный с учетом свойств, передающих среды света. Поскольку метод `pbrt` способен имитировать излучение, рассеяние и поглощение света в подробных моделях передающих сред, он способен отображать изображения, такие как это (*сцена предоставлена Джимом Прайсом*)

Есть два способа, которыми передающая среда может влиять на свет, распространяющийся вдоль луча. Во-первых, среда может *гасить* (или *ослаблять*) свет, поглощая его либо рассеивая в другом направлении. Мы можем учесть этот эффект, вычислив *коэффициент пропускания* T_r между началом луча и точкой его пересечения с объектом.

Коэффициент пропускания говорит нам, какая часть света, рассеянного в точке пересечения, возвращается к началу луча.

Передающая среда также может добавлять свет вдоль луча. Это может произойти либо в том случае, если среда излучает свет (как в случае с пламенем), либо если среда рассеивает свет с других направлений обратно по лучу. Мы можем найти эту величину путем численной оценки *уравнения объемного светового переноса*, точно так же, как мы оценивали уравнение светового переноса, чтобы найти количество света, отраженного от поверхности. Оставим описание передающих сред и объемного рендеринга до глав 11 и 14.

1.3. ОБЗОР СИСТЕМЫ pbrt

Система `pbrt` структурирована с использованием стандартных объектно ориентированных методов: для каждого из ряда фундаментальных типов система определяет интерфейс, которому должны соответствовать реализации этого типа. Например, `pbrt` требует реализации определенной формы, представляющей геометрию сцены, для предоставления набора методов, включая один, возвращающий ограничивающую форму бокс, и другой, проверяющий пересечение с заданным лучом. В свою очередь, большая часть системы может быть реализована исключительно с точки зрения этих интерфейсов, например код, проверяющий наличие перекрывающих объектов между источником света и затеняемой точкой, вызывает методы пересечения форм без необходимости учитывать, какие конкретные типы форм присутствуют в сцене.

Существует всего 14 основных базовых типов, которые приведены в табл. 1.1. Добавить новую реализацию одного из этих типов в систему несложно; реализация должна предоставлять необходимые методы, она должна быть скомпилирована и связана с исполняемым файлом, а процедуры создания объекта сцены должны быть изменены для создания экземпляров объекта в соответствии с требованиями системы, поскольку файл описания сцены анализируется. В разделе 4 расширение системы обсуждается более подробно.

Таблица 1.1. Основные типы интерфейсов. Большая часть `pbrt` реализована в виде 14 главных базовых типов, перечисленных здесь. Реализацию каждого из них можно легко добавить в систему для расширения ее функциональности

Базовый тип	Файлы-источники	Раздел
`Spectrum`	`base/Spectrum.h, util/Spectrum.{h,cpp}`	4.5
`Camera`	`base/camera.h, cameras.{h,cpp}`	5.1
`Shape`	`base/shape.h, shapes.{h,cpp}`	6.1
`Primitive`	`cpu/{primitive,accelerators}.{h,cpp}`	7.1
`Sampler`	`base/Sampler.h, Samplers.{h,cpp}`	8.3
`Filter`	`base/Filter.h, Filters.{h,cpp}`	8.8.1
`BxDF`	`base/bxdf.h, bxdfs.{h,cpp}`	9.1.2
`Material`	`base/material.h, materials.{h,cpp}`	10.5
`FloatTexture, SpectrumTexture`	`base/texture.h, textures.{h,cpp}`	10.3
`Medium`	`base/medium.h, media.{h,cpp}`	11.4
`Light`	`base/Light.h, Lights.{h,cpp}`	12.1
`LightSampler`	`base/LightSampler.h, LightSamplers.{h,cpp}`	12.6
`Integrator`	`Integrator cpu/Integrators.{h,cpp}`	1.3.3

Обычной практикой в C++ было бы указывать интерфейсы для каждого из этих типов, используя абстрактные базовые классы, которые определяют чистые виртуальные

функции, и получать реализации, наследуемые от этих базовых классов, реализующие требуемые виртуальные функции. В свою очередь, компилятор позаботится о создании кода, вызывающего соответствующий метод, для данного указателя на любой объект типа базового класса. Этот метод использовался в трех предыдущих версиях pbrt, но добавление поддержки рендеринга на графических процессорах (GPU) в этой версии мотивировало более портативный метод, основанный на *диспетчеризации тегов*, где каждой конкретной реализации типа назначается уникальное целое число, определяющее его тип во время выполнения (дополнительную информацию по этой теме см. в разделе 1.5.7). Все полиморфные типы, реализованные таким образом в pbrt, определены в файлах заголовков в каталоге base/.

Эта версия pbrt может работать на графических процессорах, поддерживающих C++17, и предоставляет API-интерфейсы для проверки пересечения лучей[9]. Мы тщательно спроектировали систему таким образом, чтобы почти вся реализация pbrt работала как на CPU, так и на GPU, как это представлено в главах со 2 по 12. Поэтому мы обычно мало говорим о CPU по сравнению с GPU в большинстве следующих разделов.

Основные различия между путями рендеринга CPU и GPU в pbrt заключаются в их потоке данных и в том, как они распараллелены, – по сути, в том, как части связаны друг с другом. Как базовый алгоритм рендеринга, описанный далее в этой главе, так и алгоритмы светового переноса, описанные в главах 13 и 14, доступны только на CPU. Конвейер рендеринга на GPU обсуждается в главе 15, хотя он также может работать на CPU (однако не так эффективно, как ориентированные на CPU алгоритмы светового переноса).

Хотя система pbrt может хорошо отображать многие сцены с их текущей реализацией, она часто расширяется студентами, исследователями и разработчиками. В этом разделе есть ряд примечательных изображений, сделанных в результате этих усилий. Рисунки 1.13, 1.14 и 1.15 были созданы студентами на курсе рендеринга, где последний проект класса должен был расширить pbrt новыми функциями для рендеринга изображения, которое она не могла визуализировать раньше. Эти изображения являются одними из лучших, что они сделали.

1.3.1. Этапы выполнения

Метод pbrt можно концептуально разделить на три этапа выполнения. Во-первых, система анализирует файл описания сцены, предоставленный пользователем. Описание сцены представляет собой текстовый файл, в котором указываются геометрические формы, из которых состоит сцена, свойства их материалов, источники света, которые их освещают, расположение виртуальной камеры в пространстве сцены и параметры для всех отдельных алгоритмов, используемых в системе. Формат файла сцены представлен на веб-сайте pbrt: pbrt.org.

Результатом фазы парсинга (парсинга) является экземпляр класса BasicScene, в котором хранится спецификация сцены, но в форме, еще не пригодной для рендеринга. На втором этапе выполнения pbrt создает объекты, соответствующие сцене; например, если задана перспективная проекция, именно на этом этапе создается объект PerspectiveCamera, соответствующий заданным параметрам вида с точки зрения камеры. Предыдущие версии pbrt смешивали эти первые две фазы, но в этой версии мы разделили их, потому что пути рендеринга CPU и GPU различаются в некоторых способах представления сцены в памяти.

На третьем этапе выполняется основной цикл рендеринга. На этот этап pbrt обычно тратит большую часть времени выполнения рендеринга, и большая часть этой книги посвящена коду, который выполняется на данном этапе. Для управления рендерингом pbrt реализует *интегратор*, названный так потому, что его основной задачей является вычисление интеграла в уравнении 1.1.

---

[9] На момент написания эти возможности были доступны только на оборудовании NVIDIA, но в будущем будет несложно перенести pbrt на другие архитектуры, которые смогут это обеспечить. – *Прим. авт.*