Фролов В.А.

Методы решения проблемы глобальной освещенности на графических процессорах

Диссертация

Институт Прикладной Математики имени М.В. Келдыша Российской Академии Наук

На правах рукописи

Фролов Владимир Александрович

Методы решения проблемы глобальной освещенности на графических процессорах

05.13.11 – Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей

ДИССЕРТАЦИЯ
на соискание ученой степени
кандидата физико-математических наук

Научный руководитель
д. ф.-м. н., проф.
Галактионов Владимир Александрович

Москва – 2014

Содержание

Введение

Актуальность работы Проблема глобальной освещенности возникает при решении широкого круга задач проектирования, оптического моделирования, компьютерной анимации и виртуальной реальности. Освещенность принято разделять на локальную и глобальную. Локальной освещенностью точки поверхности называют освещенность, вызванную прямым попаданием света от его источника. Глобальной освещенностью точки поверхности называют освещенность, вызванную как прямым так и непрямым попаданием света в данную точку (т.е. в результате одного или более переотражений). Выделим три основных группы сфер применения алгоритмов расчета глобальной освещенности.

Первую группу сфер применения алгоритмов, вычисляющих глобальное освещение, составляет синтез реалистичных изображений:

1. Оценка внешнего вида проектируемых промышленных изделий. Прежде всего это моделирование внешнего вида переносных электронных устройств (телефоны, ноутбуки, планшеты), транспортных средств (автомобилей, мотоциклов и других транспортных средств, для которых важен внешний вид).

2. Архитектурный дизайн — моделирование экстерьеров, интерьеров помещений, салонов транспортных средств в целях оценки уровня освещенности и эстетичности соответствующего интерьера или экстерьера. Для архитектурных проектов особенно важно уметь демонстрировать, как будет выглядеть спроектированное здание или интерьер дома при различных условиях освещения.

3. Создание рекламных и демонстрационных презентаций. В данной сфере применение можно условно разделить на демонстрацию решения, находящегося в проекте, и дополненную реальность — встраивание виртуального 3D объекта в видео-последовательность или изображения, снятые с реального мира.

4. Киноиндустрия, мультипликация и индустрия компьютерных игр (созда-

ние карт освещенности). Скорость расчета освещения для этих областей имеет решающее значение, т.к. процесс работы художника является циклическим, и расчет освещения производится множество раз.

Обозначенные выше сферы применения требуют высокого уровня реализма, позволяющего получить изображения виртуальной сцены визуально неотличимые от фотографии реального мира. Синтез таких изображений был бы невозможен без вычисления глобальной освещенности.

Вторая группа сфер применения расчета глобальной освещенности связана с моделированием оптических эффектов для определенного ряда задач:

1. Расчет уровня освещенности приборов в кабине самолета, поезда или другого транспортного средства, а также офисных помещений в целях обеспечения необходимого уровня безопасности и комфорта, а также соответствия стандартам.

2. Моделирование осветительных приборов, панелей ЖК-дисплеев.

3. Расчет нежелательных слепящих или ухудшающих видимость эффектов, вызванных отражениями света от зеркальных поверхностей. Среди примеров таких эффектов можно отметить отражения светящихся элементов приборной панели в лобовом стекле автомобиля, отражения ярких источников света на экране монитора, отражения солнечного света от поверхностей автобилей и архитектурных сооружений.

Сферы применения из данной группы требуют высокоточного расчета освещения в виртуальной сцене и, как следствие, вычисления глобальной освещенности.

Третья группа приложений алгоритмов вычисления глобальной освещенности связана с расчетом переноса несветового излучения. Особенно важна задача расчета переноса нейтронов в ядерных реакторах. Алгоритмы глобальной освещенности могут быть использованы для расчета переноса несветового излучения, если поведение такого излучения для заданной модели в достаточной мере описывается законами геометрической оптики.

В настоящий момент практически все известные промышленные системы, использующие графические процессоры для расчета глобальной освещенности и реалистичного синтеза изображений, основаны на различных вариациях метода Монте-Карло, дающих несмещенную оценку решения на всем множестве точек изображения. Несмещенной оценкой называют такую оценку, которая при стремлении числа Монте-Карло выборок к бесконечности сходится к точному решению. Если оценка не обладает этим свойством, ее называют смещенной. Для методов с несмещенной оценкой характерна точность в пределе, однако, они обладают медленной сходимостью по своей природе, поскольку такие методы, как правило, вычисляют искомое значение цвета методом Монте-Карло независимо для каждого пиксела изображения. В связи с этим, существующие системы реалистичной визуализации и расчета освещенности, использующие центральные процессоры, но более интеллектуальные методы расчета во многих случаях имеют сравнимое или даже меньшее время построения изображения. Как следствие — системы расчета освещения на графических процессорах получили ограниченное распространение.

Такие алгоритмы как кэш освещенности и фотонные карты дают смещенную оценку (то есть не обладают точностью в пределе), но позволяют получить приемлемое изображение за гораздо меньшее время, поскольку они в значительной мере переиспользуют результаты вычислений. Однако их реализация на графических процессорах затруднена по ряду причин:

1. Переиспользование результатов вычислений уменьшает общий объем вычислений, однако, приводит к необходимости синхронизации данных между потоками и построения ускоряющих структур на графических процессорах.

2. Методы реалистичной визуализации со смещенной оценкой решения требуют особых способов контроля точности. Существующие подходы на основе смещенных методов или не Монте-Карло (не основанных на Монте-Карло трассировке лучей) алгоритмов на графических процессорах не удовлетворяют необходимым критериям точности и направлены на интерактивный синтез изображений для компьютерных игр.

В связи с этим, в настоящий момент не существует систем расчета освещенности (кроме системы Red Shift, находящейся в разработке), способных эффективно использовать методы со смещенной оценкой для расчета освещенности с высокой точностью и синтеза реалистичных изображений. Однако выигрыш от таких алгоритмов на центральном процессоре может быть весьма значительным (до 10 раз). По этой причине в настоящей работе было принято решение разработать систему расчета освещенности на графических процессорах на основе алгоритмов со смещенной оценкой, но обладающих высокой точностью. Достаточно высокой точностью будем считать такую точность, при которой человеческий глаз не заметит разницы между изображением, построенным эталонным методом с предельно большим временем расчета (методом Монте-Карло трассировки лучей) и изображением, построенным исследуемым методом. Предельно большим временем расчета будем называть такое время, при котором эталонный метод Монте-Карло не оставляет на изображении видимого для человеческого глаза шума (или оставляет незначительный шум, сливающийся с деталями изображения).

Цель диссертационной работы состоит в разработке эффективных методов решения проблемы глобальной освещенности на графических процессорах на основе алгоритмов, дающих смещенную оценку решения, и разработке системы расчета освещенности, реализующей эти методы. Под эффективностью метода понимаются следующие два требования:

1. Метод, реализуемый на графическом процессоре, должен требовать сравнимый объем вычислений и памяти по отношению к его однопоточному аналогу.

2. Производительность метода должна линейно масштабироваться с увеличением числа ядер и вычислительной мощности графического процессора.

Необходимым требованием к решению в целом является обеспечение контроля точности и возможность достижения такой точности, при которой человеческий глаз не заметит разницы между изображением, построенным эталонным методом с предельно большим временем расчета (методом Монте-Карло трассировки лучей), и изображением, построенным на основе разработанных методов.

Для достижения поставленной цели необходимо решить следующие **задачи**:

1. Исследовать существующие алгоритмы и методы вычисления глобальной освещенности и синтеза реалистичных изображений на центральных и графических процессорах.

2. Выявить основные проблемы, возникающие при адаптации наиболее эффективных алгоритмов вычисления глобальной освещенности со смещенной оценкой решения к архитектуре графических процессоров, и предложить методы их решения.

   а. Разработать алгоритм распределения работы для метода Монте-Карло на графических процессорах в применении к задаче вычисления глобальной освещенности на множестве пикселей изображения.

   б. Разработать параллельную реализацию алгоритма кэширования освещенности на графических процессорах, позволяющую значительно снизить время построения изображения при сравнимом с эталонным методом Монте-Карло качестве.

   в. Разработать метод построения ускоряющей структуры на графических процессорах, позволяющий эффективно реализовать алгоритм фотонных карт.

3. Создать систему расчета освещенности на графических процессорах с использованием технологии CUDA.

4. Интегрировать разработанную систему в среду трехмерного моделирования 3d Studio Max.

Научная новизна

Получены следующие новые результаты в области расчета глобальной освещенности на графических процессорах.

1. Предложен новый алгоритм распределения работы для эффективной реализации метода Монте-Карло на графических процессорах в приме-

нении к задаче вычисления глобальной освещенности и реалистичного синтеза изображений. Алгоритм позволяет эффективно распределить вычислительные ресурсы графического процессора на множестве пикселей изображения с неравномерной сложностью расчета. В отличие от появившихся позднее аналогов, алгоритм позволяет сохранять пространственную близость групп лучей, что повышает эффективность использования ресурсов графического процессора за счет снижения числа расходящихся по разным веткам групп потоков.

2. Предложена параллельная реализация алгоритма кэширования освещенности на графических процессорах, позволяющая снизить время построения изображения на порядок по сравнению с традиционным методом Монте-Карло при сравнимом качестве получаемого изображения. В отличие от существующих параллельных реализаций кэша освещенности или реализаций, частично использующих графические процессоры, в предложенном подходе впервые решена проблема одновременной вставки в кэш большого числа не дублирующих друг друга записей. За счет этого достигнута масштабируемость алгоритма и возможность легкой интеграции в существующие программные системы расчета освещения на графических процессорах.

3. Разработан метод построения структуры пространственного поиска на графических процессорах, позволяющий от 2 до 5 раз ускорить сбор освещенности в алгоритме фотонных карт по сравнению с существующими методами. Разработанный метод впервые позволил строить октодеревья со множественными ссылками над множеством объектов ненулевого размера полностью параллельно на графических процессорах. В отличие от других способов построения структуры пространственного поиска для фотонных карт, предложенный метод строит структуру, позволяющую выполнить процесс сбора освещенности в заданной точке со всех фотонов в единственном цикле сбора, не содержащем ветвлений.

Практическая значимость

Разработанный на основе созданных алгоритмов программный комплекс “Hydra Renderer” является основным продуктом компании “Рей Трейсинг Системс” (http://raytracing.ru). Установка и использование продукта возможна на настольных и мобильных компьютерах, оснащённых серийными графическими процессорами с поддержкой технологии CUDA. Комплекс используется для решения практических задач в промышленности и дизайне. Основные сферы применения:

1. Оценка внешнего вида и реалистичная визуализация проектов промышленных изделий.

2. Архитектурный дизайн — моделирование экстерьеров, интерьеров помещений, салонов транспортных средств для оценки уровня освещенности и эстетичности соответствующего интерьера или экстерьера.

3. Создание рекламных и демонстрационных презентаций.

4. Расчет нежелательных оптических эффектов при проектировании архитектурных и инжереных сооружений.

Разработанный программный комплекс был применен при оценке внешнего вида проекта здания для компании Аскон и расчета нежелательных отражений от зеркальных элементов конструкции этого здания (рис. 1).

Апробация работы Основные результаты диссертации докладывались на следующих конференциях и семинарах:

1. Международная Конференции по Компьютерной Графике и Зрению GraphiCon'2009 (Москва, факультет ВМК МГУ).

2. Международная Конференции по Компьютерной Графике и Зрению GraphiCon'2010 (Санкт-Петербургский государственный университет информационных технологий, механики и оптики).

3. Международная Конференции по Компьютерной Графике и Зрению GraphiCon'2012 (Москва, факультет ВМК МГУ).

4. Международная Конференции по Компьютерной Графике и Зрению GraphiCon'2013 (Институт автоматики и процессов управления ДВО РАН, Дальневосточный Федеральный Университет).

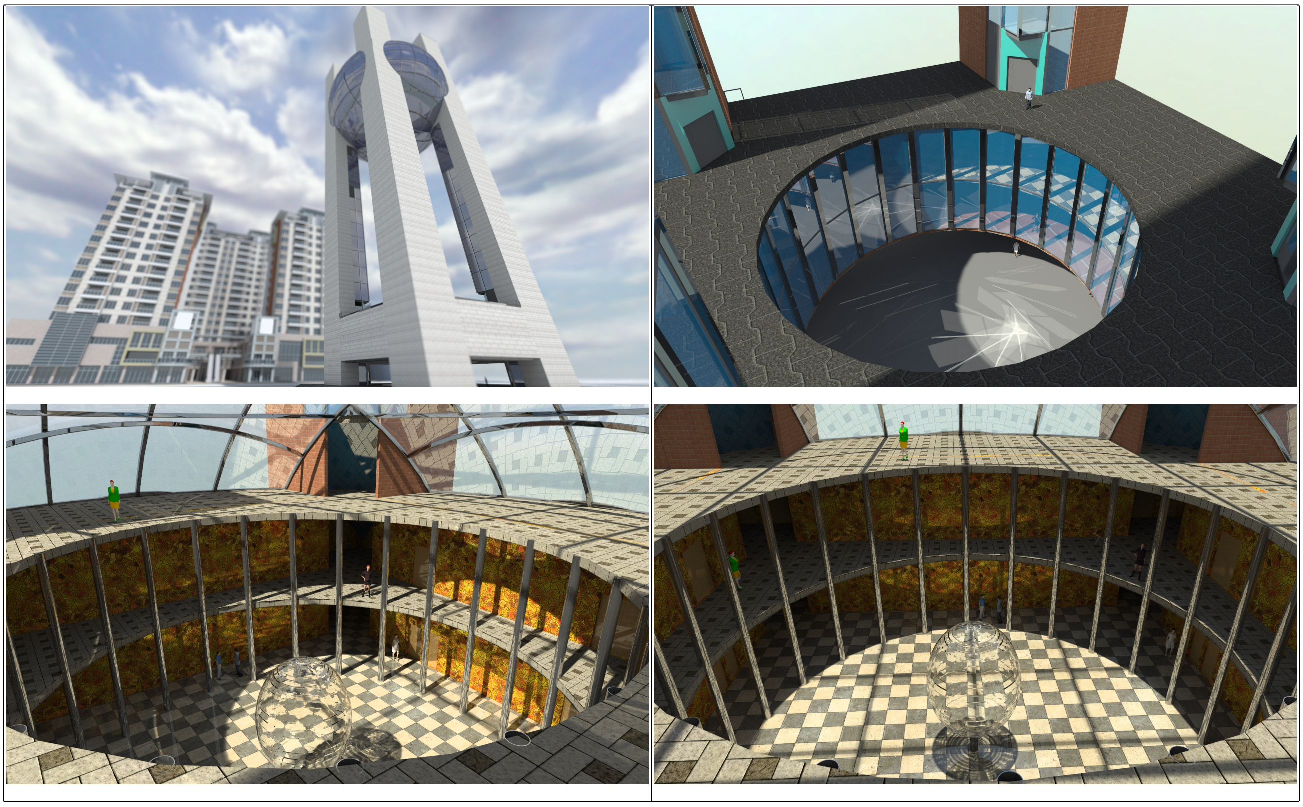


Рис. 1. Примеры применения разработанной системы для оценки внешнего вида небоскреба и расчета нежелательных отражений.

5. Семинар по компьютерной графике и машинному зрению Ю.М. Баяковского (Москва, факультет ВМК МГУ).

6. Семинар направления “Программирование” им. М. Р. Шура-Бура в ИПМ им. М. В. Келдыша.

7. Тринадцатый научно-практический семинар “Новые информационные технологии в автоматизированных системах”. Московский государственный институт электроники и математики (2010 год).

Публикации По результатам работы имеются 10 печатных работ [1–10], из них 2 публикации в рецензируемых журналах Перечня ВАК [2, 4], 1 публикация входит в библиографическую базу Web Of Science [2], 3 публикации входят в библиографическую базу Scopus [1–3].

Личный вклад автора Содержание диссертации и основные положения, выносимые на защиту, отражают персональный вклад автора в опубликованных работах. Подготовка к публикации полученных результатов проводилась совместно с соавторами, причем вклад диссертанта был определяющим. Все представленные в диссертации результаты получены лично автором.

Структура и объем диссертации Диссертация состоит из введения, обзора литературы, 4 глав, заключения, библиографии и приложения А. Общий объем диссертации 175 страниц, из них 146 страниц основного текста, включая 60 рисунков. Библиография включает 154 наименования. Приложение А включает 28 страниц и содержит данные сравнения разработанной системы с аналогами.

Глава 1

Введение в предметную область

Первая глава представляет из себя введение в предметную область и содержит основные определения и термины. В главе проводится обзор базовых алгоритмов решения проблемы глобальной освещенности.

1.1. Трассировка лучей

Центральным алгоритмом, вокруг которого построены практически все методы расчета освещенности, является трассировка лучей. Алгоритм трассировки лучей был впервые продемонстрирован Turner Whitted-ом в 1980-ом году [11]. Этот алгоритм часто называют обратной трассировкой лучей или ’Whitted-style’ трассировкой. Сам по себе алгоритм трассировки лучей позволяет получить изображения лишь с базовым уровнем реалистичности и рассчитывает такие эффекты как тени, отражения и преломления.

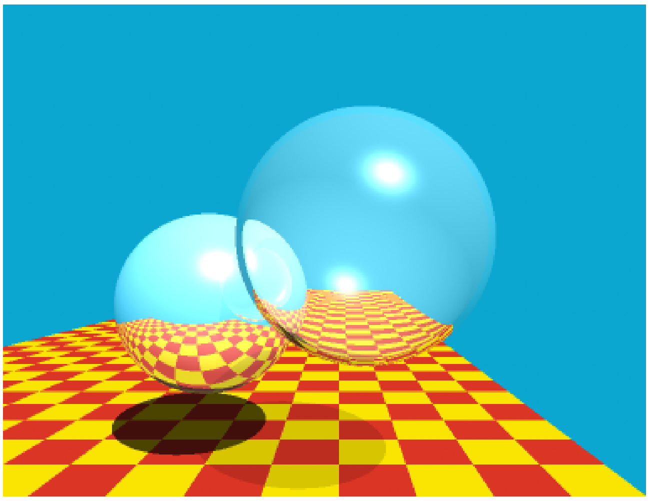


Рис. 1.1. Whitted-style трассировка лучей.

1.1.1. Поиск пересечений

Поиск пересечений луча с геометрическими объектами сводится к нахождению всех общих точек луча и объекта. Эта операция выполняется наиболее часто и почти всегда является узким местом в трассировке лучей. Объекты в компьютерной графике, как правило, представляются набором примитивов. Два наиболее важных вида примитовов — трегольник и прямоугольный параллелипипед. Треугольник важен, поскольку в современной компьютерной графике поверхности в подавляющем большинстве случаев задаются именно при помощи треугольников, а прямоугольный паралеллипипед необходим для реализации эффективного поиска пересечений на основе BVH деревьев [12]. Для вычисления пересечения луча и треугольника обычно используются алгоритмы, описанные в [13], [14]. Для вычисления пересечения луча и прямоугольного паралеллипипеда можно использовать алгоритм, описанный в [15].

1.1.2. Ускорение поиска пересечений

Как правило, существенное ускорение может быть достигнуто при использовании структур пространственного разбиения. Среди таких структур выделяют регулярные и иерархические сетки, kd-деревья, окто-деревья, BVH-деревья. В сочетании с ранним подразбиением [16], BVH дерево на практике показало себя с наилучшей стороны [17]. Кроме того, алгоритм построения BVH достаточно прост и занимает определенный объем памяти, пропорционально количеству входных примитивов.

1.2. Проблема глобальной освещенности

Один из способов получения фотореалистичных изображений — моделирование оптических процессов с целью точного вычисления освещенности каждой точки сцены. Как следствие, появляется необходимость в вычислении глобальной освещенности. Как обсуждалось во введении, освещение принято разделять на локальное и глобальное. Напомним, что локальным освещением называют освещение вызванное прямым попаданием света от источника на поверхность. Глобальным освещением называют освещение, вызванное прямым и непрямым попаданием света на поверхность. Непрямое попадание света, в свою очередь, вызвано переотражениями световой энергии от различных объектов трехмерной сцены и является трудоемким в плане вычислений.

Трассировка лучей, описанная выше, позволила бы рассчитать изображение корректно, если бы поверхности объектов реального мира были гладкими и отражали свет строго по закону 'угол падения равен углу отражения'. Как правило, из-за наличия микрорельефа поверхностей это не так, и свет, приходящий строго с одного направления, распределяется по полусфере вокруг нормали к поверхности в соответствии со свойствами материала.

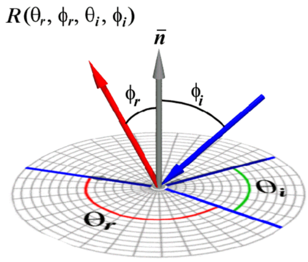


Рис. 1.2. Двунаправленная Функция Отражения.

Для моделирования свойств материала вводят Двунаправленную Функцию Отражения (ДФО) [18] (в целях упрощения изложения ограничимся только отражающими материалами). В английской терминологии обозначается как BRDF (Bidirectional Reflectance Distribution Function). ДФО связывает по некоторому закону интенсивность и угол падающего света с интенсивностью и углом отраженного поверхностью света. Зная ДФО материала, можно вычислить освещенность точки поверхности при помощи интеграла освещенности [19]:

$$I(\phi_r, \theta_r) = \iint_{\phi_i \theta_i} L(\phi_i, \theta_i) R(\phi_i, \theta_i, \phi_r, \theta_r) cos(n, l_{\phi_i, \theta_i}) \mathrm{d}\phi_i \mathrm{d}\theta_i \tag{1.1}$$

В уравнении 1.1 функция $L(\phi_i, \theta_i)$ задает яркость света, падающего с направления, задаваемого вектором l_{ϕ_i, θ_i}. $R(\phi_i, \theta_i, \phi_r, \theta_r)$ — ДФО. n — вектор нормали к поверхности. Вычисляемая интегрированием интенсивность освещения в точке $I(\phi_r, \theta_r)$ является не числом, а функцией. Другими словами, она дает значения интенсивности света, отражаемой поверхностью под разными углами. Таким образом, выполняя интегрирование по полусфере приходящего (падающего) в точку освещения L с учетом свойств поверхности R, получаем новую функцию распределения освещения в пространстве I, обусловленную свойствами отражения поверхности (материала) в некоторой точке x.

Если зафиксировать конкретный луч (например выпущенный для конкретного пиксела и виртуальной камеры), направление ϕ_r, θ_r было бы фиксированно и задавалось бы этим лучом.

Размерность интеграла освещенности (формула 1.1) зависит от количества учитываемых переотражений света, поскольку функция L в некоторой точке x будет зависеть от функции I в некотрой другой точке y. Из-за большой размерности интеграла для его вычисления обычно используют метод Монте-Карло, поскольку его скорость сходимости не зависит от размерности интеграла [20]. В применении к интегралу освещенности:

$$I(\phi_r, \theta_r) \approx \frac{1}{\pi} \frac{\sum_{i=1}^{N} L(\phi_i, \theta_i) R(\phi_i, \theta_i, \phi_r, \theta_r) cos(n, l_{\phi_i, \theta_i})}{N} \tag{1.2}$$

Таким образом, мы можем оценивать значения интеграла при помощи вычисления суммы достаточно большого числа значений подинтегральной функции [20]. Далее будут рассмотрены некоторые определения и понятия математической статистики.

Смещенные и несмещенные оценки и методы.

В данном разделе мы кратко рассмотрим понятие смещенных и несмещенных оценок в применении к вычислению интеграла освещенности, чтобы пояснить такие часто встречающиеся англо язычные термины как biased renderer и unbiased renderer.

Пусть $X_1..X_n$ случайная выборка из распределения, зависящего от параметра $\theta \in \Theta$

Статистикой называется любая измеримая функция от выборки θ_i, не зависящая от θ (важно, что значение статистики можно вычислить при каждой реализации выборки, не зная значения θ).

Точечной оценкой называется любая статистика, принимающая некоторые значения на области определения Θ.

Оценка называется несмещенной (unbiased), если ее математическое ожидание равно оцениваемому параметру. В противном случае оценка называется смещенной (biased).

Оценка называется состоятельной (consistent), если она сходится по вероятности (сходимость почти наверное) к оцениваемому параметру.

Будем называть метод или алгоритм вычисления интеграла освещенности несмещен
если он позволяет получить несмещенную оценку интеграла освещенности для индивидуальной точки или набора точек трехмерного пространства. В противном случае будем называть метод смещенным.

Будем называть метод вычисления интеграла освещенности состоятельным, если он позволяет получить состоятельную оценку интеграла освещенности для индивидуальной точки или набора точек трехмерного пространства.

Таким образом, если программное обеспечение для вычисления освещенности и фотореалистичной визуализации (англ. renderer) использует в том или ином виде метод Монте-Карло и при этом позволяет при бесконечно большом времени расчета получить абсолютно точное решение, говорят, что это несмещенный рендерер (unbiased renderer). К таким программам, например, можно отнести все виды трассировщиков путей. Если программа использует метод Монте-Карло, но при этом в пределе (при бесконечно большом времени расчета) не позволяет получить абсолютно точное решение, говорят, что такой рендерер является смещенным (например программы, использующие фотонные карты и/или кэш освещенности). К программам, не использующим метод Монте-Карло, понятия смещенный/несмещенный неприменимы. Например, некорректно использовать указанные термины по отношению к программе, использующей метод излучательности [21].

Смещенность алгоритма еще не означает, что он вычисляет интеграл с низкой точностью. На практике смещение (bias) проявляется в виде цветных пятен. Если смещение небольшое, пятна незаметны для глаза. Как правило, смещенные методы на начальном этапе работы (при небольшом числе выборок или небольшом времени работы) дают более приемлемую для человеческого глаза оценку, чем несмещенные. Однако в пределе при длительном расчете несмещенные методы обычно дают более качественное изображение чем смещенные методы за то же самое время.

1.3. Стохастическая трассировка лучей

Стохастическая трассировка лучей (также называемая Монте-Карло трассировкой) вычисляет значение интеграла освещенности (для фиксировнаного направления ϕ_r, θ_r) напрямую по формуле 1.2. Метод был предложен James-ом Kajiya в 1986 году [19].

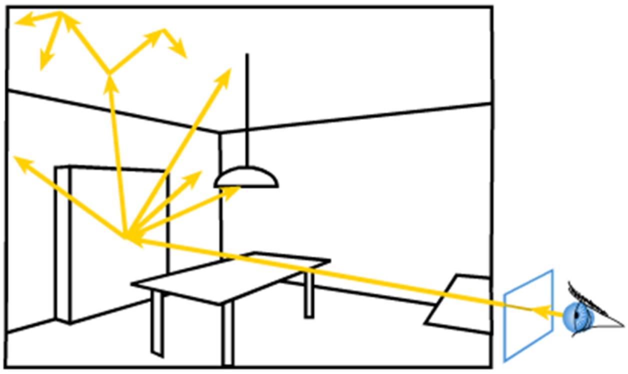


Рис. 1.3. Иллюстрация монте-карло трассировки лучей.

Из виртуальной камеры испускается луч, который трассируется в сцену. В точке пересечения с поверхностью выпускается некоторое число лучей по полусфере и для каждого луча процедура выполняется рекурсивно (рис. 1.3). Полученные значения яркости на каждом уровне рекурсии складываются с учетом формулы 1.2. В дальнейшем будем рассматривать модификацию стохастической трассировки лучей, при которой отраженный луч всегда один (рис 1.4). Такое упрощение немного замедляет сходимость метода, но позволяет упростить реализацию и при желании избавиться от рекурсии в реализации алгоритма. Скорость сходимости трассировки путей может быть существенным образом улучшена за счет изменения схемы генерирования Монте-Карло выборок — теневых лучей (введения явной стратегии создания выборок), выборки по значимости и многократной выборки по значимости [22].

1.3.1. Грамматика путей

Для классификации различных ситуаций, возникающих в процессе трассировки путей часто используют понятие грамматики путей [23]. Этот способ классифицирует пути при помощи строк и регулярных выражений. Каждый

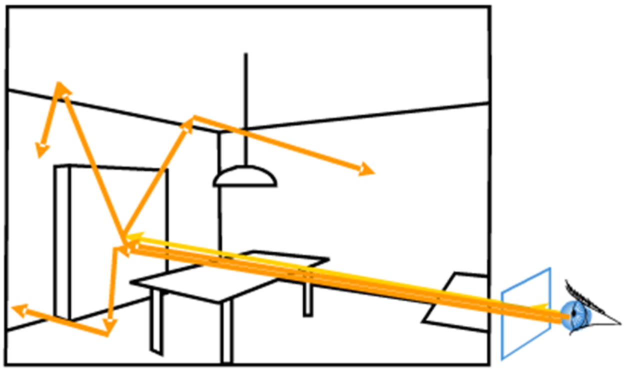


Рис. 1.4. Иллюстрация монте-карло трассировки путей.

символ в строке обозначает определенное событие, произошедшее с лучом (путём) в процессе трассировки.

1. S — (Specular); зеркальное отражение/преломление
2. D — (Diffuse); диффузное (ламбертовское) отражение
3. G — (Glossy); матовое отражение/преломление
4. V — (Volume); объемное рассеивание
5. L — (Light); источник света
6. E — (Eye); глаз
7. Символ или последовательность символов, отмеченная '+' встречается один или более раз.
8. Символ или последовательность символов, отмеченная '*' встречается ноль или более раз.

При этом под символами S и G понимается не только отражение, но и преломление света. Грамматика путей позволяет классифицировать видимые эффекты, однозначно ставя им в соответствие классы путей которые эти эффекты вызывают. Например, EDL — прямое освещение диффузной поверхности. $E(S|G)^{+}L$ — яркий блик. $EDSL$ — каустик (эффект, возникающий в результате отражения света от ярких зеркальных поверхностей (или преломления света через стекло — рис. 1.5) и последующего рассеивания света на диффузных поверхностях - например солнечный зайчик от зеркала или стакана с водой), видимый из камеры напрямую, обусловленный одним зеркальным переотражением света. EDS^+L — каустик, видимый из камеры напрямую, вызванный одним или более зеркальным переотражением. ES^+DS^+L — каустик, видимый через стекло или зеркало, вызванный одним или более зеркальным переотражениями. При помощи грамматики путей удобно классифицировать алгоритмы по типу освещения, которые они способны расчитывать. Например, Трассировка лучей Уиттеда, рассмотренная нами в самом начале, строит пути ES^*L. Обратная трассировка путей строит пути вида $E(S|D|G)^*L$.

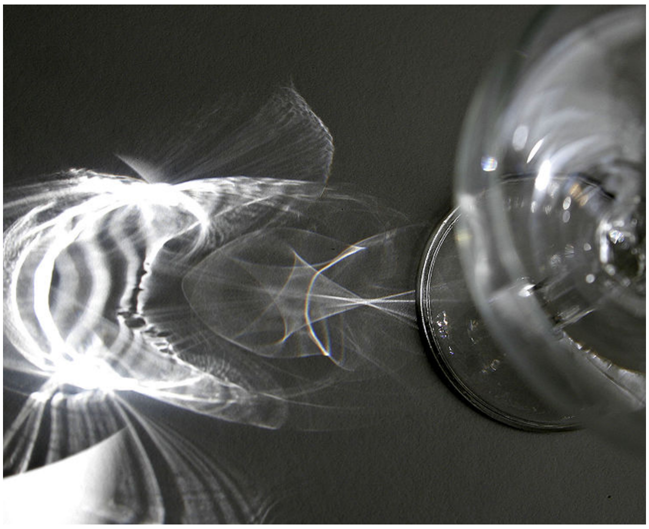


Рис. 1.5. Пример каустика.

1.4. Прямая трассировка (Light Tracing)

В противоположность обратной трассировке путей, рассмотренной выше, алгоритм прямой трассировки путей начинает свою работу из источника света и строит пути вида $L(S|D|G)^*E$. При каждом соударении с поверхностью создается аналог теневого луча, направленный в камеру. Если луч успешно доходит до камеры, значение яркости записывается в пиксель, соответствующий проекции начала луча на экранную плоскость.

Пути, создаваемые на источнике, должны быть сгенерированы в соответствии с распределением световой энергии для источника света (рис. 1.6).

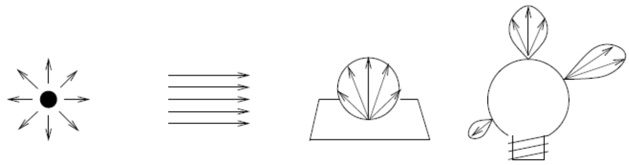


Рис. 1.6. Иллюстрация различных видов распределения световой энергии для различных источников света.

Существующий миф о крайне низкой скорости прямой трассировки путей (или лучей) далеко не всегда соответствует действительности. Такие эффекты как каустики чрезвычайно быстро могут быть рассчитаны именно при помощи прямой трассировки. В связи с этим, возникает вопрос о том, можно ли комбинировать прямую и обратную трассировки путей для того, чтобы исключить недостатки обоих алгоритмов.

Метод, реализующий эту идею, существует и называется 'усеченная двунаправленная трассировка путей' [24]. Идея этого метода в упрощенном представлении изложена ниже:

1. Будем производить обратную трассировку путей с теневыми лучами. При этом отключим расчет каустиков. Результат сохраняем в изображение с номером 1. На изображении 1 мы полчим картинку без каустиков.

2. Будем производить прямую трассировку из источника света, но на этот раз будем сохранять вклад только от тех путей, для которых первое переотражение было зеркальным. Результат сохраняем в изображение с номером 2. На изображении 2 мы получим только каустики.

3. Финальное изображение можно получить простым сложением изображений 1 и 2.

Усеченная двунаправленная трассировка путей дает почти корректный результат, поскольку изображения 1 и 2 будут нести в себе освещение, обусловленное различными типами переотражений. Слово 'почти' означает, что при помощи такого метода не удастся эффективно рассчитать каустики, не видимые камерой напрямую. То есть не удастся эффективно вычислять освещение, вызванное путями вида ES^+DS^+L. Время расчета таких эфектов будет сопоставимо со временем их расчета обыкновенной трассировкой путей.

Предлагаемая в работе [25] модификация метода усеченной двунаправленной трассировки путей на основе регуляризации (подход регуляризации будет рассмотрен во 2 главе) позволяет устранить недостаток метода и сделать его эффективным при рассчете путей содержащих S^+DS^+ фрагменты.

1.5. Двунаправленная трассировка путей

Идея объединить преимущества прямой и обратной трассировки привела к созданию алгоритма двунаправленной трассировки путей (Bidirectaional Path Tracing, BDPT) [22]. Основная идея двунаправленной трассировки путей заключается в том, чтобы соединять теневыми лучами не только источник с точками пути (как в обратной трассировке) или камеру с точками пути (как в прямой трассировке), но и различные точки путей друг с другом (рис. 1.7).

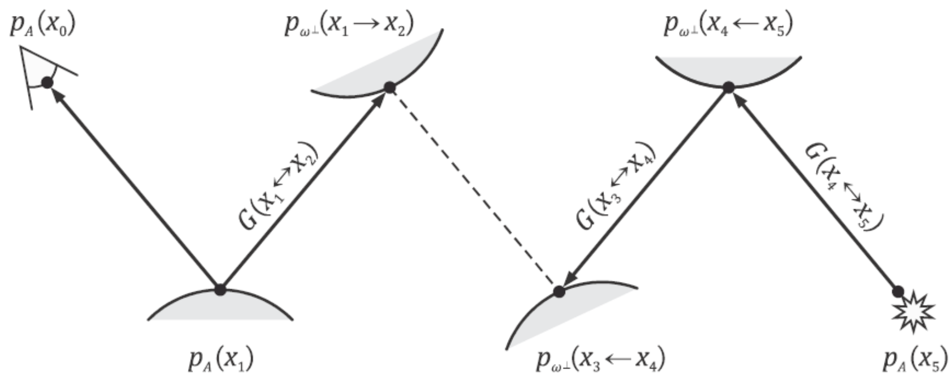


Рис. 1.7. Идея двунаправленной трассировки путей.

Использование многократной выборки по значимости в двунаправленной трассировке путей является ключевым моментом [22]. В противном случае, изображение получается сильно зашумленным, а скорость сходимости низкой. Однако даже при реализации многократной выборки по значимости, двунаправленная трассировка путей по прежнему не позволяет эффективно рассчитывать пути вида ES^+DS^+L (каустики, видимые через зеркало или стекло). Такие пути могут быть эффективно вычислены при помощи фотонных карт, переноса света Метрополиса и метода соединения вершин (vertex merging)[26].

1.6. Перенос света Метрополиса (MLT)

Трассировка путей обладает достаточно низкой скоростью на сценах со сложными условиями освещения. Сценами со сложными условиями освещения будем называть 2 вида сцен — сцены, на которых преобладает вторичное освещение, вызванное узкими и яркими световыми пятнами (рис. 1.8), и сцены, на которых $ES^{+}DS^{+}L$ пути вносят значительный вклад (рис. 1.9).

Рис. 1.8. Пример сцены со сложными условиями освещения.

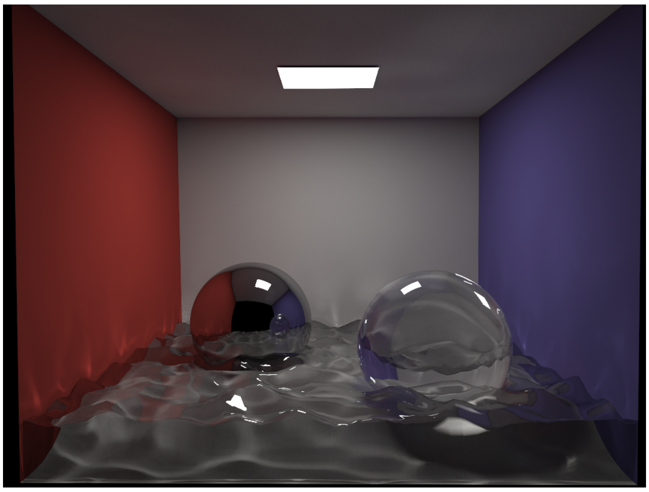


Рис. 1.9. Пример сцены с $ES^{+}DS^{+}L$ путями. Каустики под водой (рис. 1.8).

Проблема на таких сценах заключается в том, что в большинстве мест на сцене функция падающего освещения имеет один или более резких максимумов. Для того чтобы интеграл от такой функции вычислить с высокой точностью методом Монте-Карло с равномерным распределением случайной величины, нужно большое число выборок.

Для улучшения скорости сходимости на таких сценах Э. Вичем и Л. Гибсом был разработан алгоритм переноса света Метрополиса (Metropolis Light Transport, MLT) [27]. Идея этого метода аналогична применению выборки по значимости [20] к под-интегральному выражению 1.3, однако имеет совершенно иную природу и работает даже при полном отсутствии какой-либо априорной информации о функции распределения падающей освещенности.

$$L(\phi_i,\theta_i)R(\phi_i,\theta_i,\phi_r,\theta_r)cos(n,l_{\phi_i,\theta_i}) \tag{1.3}$$

Рассмотрим Метод Монте-Карло в общем виде (формула 1.4).

$$\int\limits_a^b f(x)dx \approx \frac{1}{N}\sum_{i=1}^{N}\frac{f(u_i)}{p(u_i)} \tag{1.4}$$

Если бы мы имели функцию распределения $p(x)$ пропорциональную $f(x)$, тогда для некоторой константы c можно записать $p(x) = cf(x)$. Подставив константу в выражение для интеграла 1.4, получим оценку для константы:

$$c = \frac{1}{\int_a^b f(x)dx} \tag{1.5}$$

Таким образом, мы имеем возможность оценивать константу c, взяв, например, выборки с равномерным распределение и применив метод монте-карло. Алгоритм Метрополиса отвечает за оставшуюся часть задачи — создание $p(x)$ пропорционально $f(x)$ [27], [28]. Важным свойством алгоритма Метрополиса является то, что он строит $p(x)$ даже если значения $f(x)$ могут быть вычислены только стохастически. Это именно та ситуация, которая присутствует в трассировке путей. Алгоритм переноса света Метрополиса строит распределение $p(x)$ при помощи механизма мутаций путей в трехмерном пространстве и вычисления специальным образом вклада путей на основе вероятности мутации одного пути в другой. Наиболее сложная часть в алгоритме переноса света Метрополиса – выбор хорошей стратегии мутации и вычисление вероятностей мутаций [27].

1.7. Фотонные карты

Рассмотренные ранее несмещенные методы на основе трассировки путей работают в терминах яркости. Каждый путь в них отвечал за перенос яркости от источника света к камере. Метод фотонных карт работает в терминах потока. Идея этого метода заключается в том, чтобы вычислить распределение световой энергии по сцене при помощи трассировки множества частиц, переносящих порцию световой энергии. После чего можно оценивать интеграл освещенности при помощи апроксимации дифференциального потока, выполняя на поверхности поиск ближайших фотонов (рис. 1.6) [29].

$$I(\phi_r, \theta_r) = \iint\limits_{\phi_i \theta_i} \frac{d^2\Phi}{dA} R(\phi_i, \theta_i, \phi_r, \theta_r) \tag{1.6}$$

$$I(\phi_r, \theta_r) \approx \sum_{i=1}^{K} R(\phi_i, \theta_i, \phi_r, \theta_r) \frac{\Delta\Phi(\phi_i, \theta_i)}{\Delta A} \tag{1.7}$$

$$= \frac{1}{\pi r^2} \sum_{i=1}^{K} R(\phi_i, \theta_i, \phi_r, \theta_r) \Delta\Phi(\phi_i, \theta_i) \tag{1.8}$$

В выражении 1.6 переменная r отвечает за радиус диска на поверхности, в пределах которого выполняется поиск ближайших фотонов. Метод фотонных карт, таким образом, состоит из 4 шагов:

1. Испускание фотонов из источников света. Фотоны испускаются из источника света в соответствии с распределением световой энергии для данного источника.

2. Трассировка фотонов. Трассировка фотонов происходит похожим на прямую трассировку путей образом за исключением одного важного отличия – механизма русской рулетки (этот механизм можно также применять и в трассировке путей [24], но исторически он появился именно

в методе фотонных карт) [29]. За счет использования русской рулетки повышается точность решения, покольку в местах с высокой освещенностью будет сохраняться больше фотонов, чем в местах с низкой освещенностью; в неосвещенных областях фотоны будут отсутствовать. Во время трассировки фотоны сохраняются на всех диффузных (ламбертовых) поверхностях, которые они ударяют.

3. Построение фотонной карты. Стадия построения фотонной карты необходима для того, чтобы эффективно выполнять сбор освещенности.

4. Сбор освещенности. После того как фотонная карта построена, наступает стадия сбора освещенности. Из виртуальной камеры испускаются лучи и выполняется обратная или стохастическая трассировка лучей (или трассировка путей). В местах соударений луча и поверхности производится сбор освещенности (формула 1.9, рисунок 1.10).

$$I(\phi_r, \theta_r) \approx \frac{1}{\pi r^2} \sum_{i=1}^{K} R(\phi_i, \theta_i, \phi_r, \theta_r) \Delta\Phi(\phi_i, \theta_i) \tag{1.9}$$

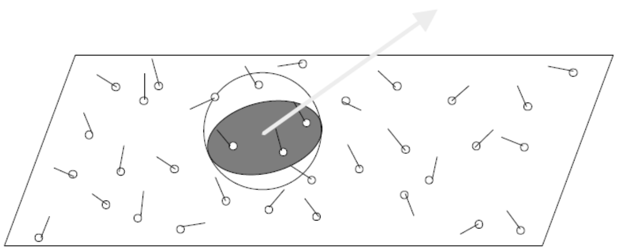


Рис. 1.10. Иллюстрация к формуле 1.9. Сбор освещенности или интегрирование фотонной карты. В данной формуле r — радиус диска, полученного при проекции сферы на поверхность. πr^2 — площадь диска.

Существует 2 основные стратегии сбора освещенности:

1. Сбор k-ближаших фотонов (динамический радиус сбора), в соответствии с формулой 1.9.

2. Сбор всех фотонов в заданном радиусе (фиксированный радиус сбора). Данная стратегия упрощает процесс сбора и дает корректный результат в пределе (при стремлении общего числа фотонов к бесконечности), однако не обеспечивает постоянное значение K в формуле 1.9.

1.7.1. Финальный сбор

Один из артефактов, возникающих при реализации сбора освещенности — темные края поверхностей(рис 1.11). Данный артефакт возникает по причине того, что на границах поверхностей освещение собирается только с половины диска. При этом площадь, вычисляемая как πr^2, для целого диска остается прежней. Один из способов, позволяющих убрать темные края заключается в том, чтобы найти реальную площадь элемента поверхности, на котором происходил сбор при помощи вычисления площади пересечение сферы сбора и всех треугольников с нормалью, близкой к нормали в точке сбора (то есть вычислить честную проекцию сферы на поверхность). Второй метод называется Финальный Сбор (Final Gathering, FG). Вместо непосредственного сбора освещенности с фотонов, в методе финального сбора из заданной точки испускается некоторое число лучей по полусфере и освещенность собирается уже в тех местах, куда попали лучи. При использовании финального сбора сами фотонные карты, таким образом, задействуются только для вычисления апроксимации вторичной освещенности. Помимо избавления от темных краев, финальный сбор значительно повышает точность решения (рис. 1.11).

Карты светимости (radiosity maps). Процесс поиска ближайших фотонов является, как правило, узким местом финального сбора. Поскольку в данном случае при помощи самих фотонных карт достаточно получить лишь грубую оценку освещенности, существуют различные методы ускорения финального сбора, использующие этот факт. К таким методам относятся карты светимости [30] или октанные текстуры [31]. Их смысл заключается в том, чтобы предрасчитать светимость в точках поверхности [30] (или в объеме [31]) и запомнить ее в виде карты светимости — двумерной или трехмерной (возможно разряженной) текстуры. При попадании луча финального сбора в определенную точку поверхности поиск ближайших фотонов не выполняется. Вместо этого производится быстрая выборка значения из карты светимости.

Рис. 1.11. Сбор освещенности с 10M фотонов (слева). Финальный сбор с 1M фотонов (справа).

1.7.2. Прогрессивные фотонные карты

Для фотонных карт в классической реализации серьезной проблемой является объем потребляемой памяти. Для того чтобы получить изображение высокого качества, может потребоваться до нескольких миллиардов фотонов. Хранить все фотоны в памяти в этом случае не представляется разумным, поскольку даже если хранить фотоны в соответствии с компактным представлением в 16 байт, один миллиард фотонов займет $16 * 10^9 \approx 15Gb$.

Для того чтобы иметь возможноть обрабатывать произвольное число фотонов, имея фиксированный объем памяти, был разработан метод прогрессивных фотонных карт [32]. Основная идея метода прогрессивных фотонных карт заключается в том, чтобы повторять все 4 шага (испускание фотонов из источника света, трассировка, построение фотонной карты и сбор освещенности) несколько раз для порций фотонов фиксированного размера (например,по 1 миллиону фотонов). При этом, при обработке каждой следующей порции фотонов радиус сбора и аккумулируемый поток уменьшаются в соответствии с формулой 1.10. Такой метод называется Прогрессивными фотонными картами (Progressive Photon Mapping, PPM [32]). Если выбрана стратегия с фиксированым радиусом сбора, то для каждой следующей порции фотонов радиус сбора уменьшается в соответствии с соотношением 1.11. Такой алгоритм называется Стохастическими Прогрессивными Фотонными Картами (Stochastic

Progressive Photon Mapping, SPPM [33], [34]). Параметр α варьируется в пределах от 0 до 1. Рекомендуемое значение — больше или равно 2/3.

$$\frac{r_{i+1}^2}{r_i^2} = \frac{N_i + \alpha M_i}{N_i + M_i} \tag{1.10}$$

$$\frac{r_{i+1}^2}{r_i^2} = \frac{i + \alpha}{i + 1} \tag{1.11}$$

В соотношении 1.10 N_i — число уже собранных фотонов. M_i — число вновь добавленных фотонов (на очередном проходе).

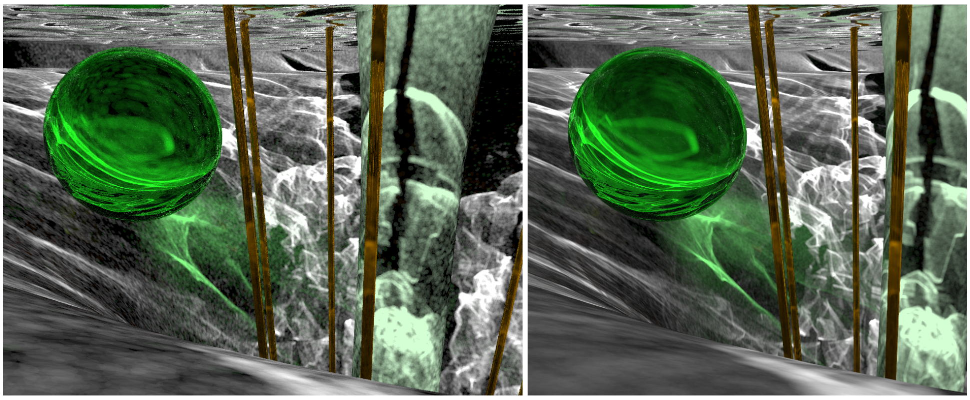


Рис. 1.12. Стохастические прогрессивные фотонные карты. 1M фотонов слева. 10M (10 проходов) — справа.

Метод стохастических прогрессивных карт значительно проще в реализации, чем обыкновенные прогрессивные фотонные карты, поскольку он не фиксирует точки сбора (что необходимо делать в методе прогрессивных фотонных карт)[33], [34]. Причем, в [33] было показано, что стохастические прогрессивные фотонные карты обладают лучшей сходимостью. Результаты для различных порций фотонов могут быть скомбинированы простым усреднением, как и в обыкновенной трассировке путей.

1.7.3. Некоторые важные оптимизации

Для достижения приемлемого результата на практике при использовании фотонных карт прибегают к таким оптимизациям как:

1. Три разные фотонные карты. Глобальная, каустическая и объемная. Данный прием значительно повышает точность решения [29].

2. Определение видимости поверхностей. За счет того, что на невидимых поверхностях фотоны не сохраняются, можно хранить в памяти только те фотоны, которые непосредственно участвуют в построении изображения. Один из методов пометки видимых поверхностей — на основе частиц видимости [35].

3. Использование карт проекций (projection maps) для увеличения эффективности трассировки фотонов [29].

1.7.4. Резюме по фотонным картам

Фотонные карты наряду с методом соединения вершин [26] являются одним из самых эффективных алгоритмов для вычисления путей вида S^+DS^+ (каустики, видимые через стекло или зеркало). Такие пути, как правило, значительно хуже вычисляются трассировкой путей. В сочетании с финальным сбором и картами светимости, фотонные карты также являются эффективным способом вычисления диффузного вторичного освещения.

1.8. Кэш Освещенности (Irradiance Cache)

Кэш Освещенности (Iradiance Cache, IC) – это не способ вычисления интеграла освещенности. Это лишь способ ускорения вычисления этого интеграла на множестве точек. Алгоритм был впервые представлен в работе [36]. Основная идея заключается в том, что вторичное освещение разделяется на две компоненты – низкочастотную (диффузную) и высокочастотную (отражающую). Низкочастотная компонента на изображении и в трехмерном пространстве меняется плавно, поэтому ее можно вычислить каким-либо из методов лишь в очень небольшом числе точек, а в остальных — интерполировать [37]. Рисунок 1.13 демонстрируют пример использования кэша освещенности.

Высокочастотная компонента обычно обусловлена резкими максимумами BRDF, поэтому она может быть вычислена сэмплированием с помощью относительно небольшого числа (что конечно не всегда так) лучей только в максимумах BRDF (importance sampling).

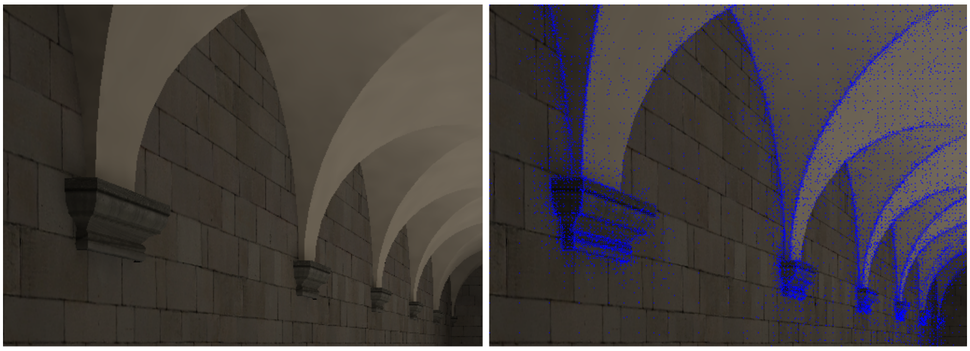


Рис. 1.13. Освещение, вычисленное при помощи кэша освещенности (слева). Визуализированные точки кэша (справа).

Различают кэш освещенности в трехмерном пространстве сцены (world space) и в экранной плоскости (screen space). Реализация кэша в экранной плоскости проще и эффективнее. Интерполяция также производится в пространстве экрана. Однако, кэш освещенности в пространстве экрана может быть использован по очевидным причинам только для первично видимых поверхностей.
Как правило, кэш освещености вычисляется на лету. Каждый раз при вычислении вторичного диффузного освещения делается попытка выборки из кэша. Если значение может быть выбрано так, что ошибка меньше допустимой величины, оно получается как результат интерполяции ближайших точек кэша. Если ошибка больше допустимой величины, значение честно вычисляется с помощью трассировки лучей по всем направлениям (или финального сбора из

фотонной карты) и полученная точка сохраняется в КЭШе [37].

Исходные параметры: p - точка на поверхности, n - нормаль к поверхности в точке p

Результат: Значение освещенности

```
if InterpolationIsPossibe(p,n) then
    return IrradianceCacheLookUp(p,n) ;
else
    I ← EvalIntegral(p, n) ;
    R ← CalcValidityRadius(p, n);
    InsertRecordInCache(I, R, p, n) ;
    return I;
end
```

Алгоритм 1: Алгоритм кэширования освещенности

Алгоритм кэширования вторичного освещения содержит 2 узких места — расчет собственно вторичного освещения в точках кэша и финальный алгоритм интерполяции освещения во всех остальных точках.

В целом, кэширование вторичного освещения — весьма эффективная техника ускорения вычисления глобально освещения, позволяющая снизитиь количество необходимых вычислений на 1-2 порядка (для больших разрешений). Допустим имеется изображение размером 1024x1024 пиксела. Если вычислять вторичное освещение в каждой точке, то потребуется суммарно протрассировать порядка 10^9 лучей только для вычисления вторичной освещенности. При использовании кэша освещенности количество точек, в которых необходимо вычислить вторичную освещенность, сокращается с миллиона до нескольких тысяч. Соответственно, число лучей падает с 10^9 до 10^6 - 10^7.

Однако, кэширование освещенности имеет определенные недостатки:

1. Кэш освещенности хорошо работает только на сценах с преимущественно гладкими поверхносями. На сценах с большим количеством геометрических деталей (например карты нормалей для имитации микрорельефа) кэш освещености не даст преимущества и может даже замедлить процесс построения изображения.

2. Кэш освещенности достаточно сложно сделать параллельным.

3. Кэш освещенности дает мерцание при анимации. Если стохастическая трассировка путей дает шум, который эффективно удаляется при помощи размытия в движении, то пятна вызванные кэшем освещенности удалить гораздо труднее.

4. Кэш освещенности все же снижает точность решения и является смещенным методом. Чем ниже точность, тем больше выигрыш в скорости, но тем более заметны артефакты.

1.9. Выводы к первой главе

Таким образом, методы расчета освещения и синтеза реалистичных изображений опираются преимущественно на Монте-Карло интегрирование при помощи трассировки лучей. При этом существует несколько базовых методов решения проблемы глобальной освещенности. Разнообразие методов обуславливается тем, что особенности освещения и природа визуальных эффектов в различных условиях могут быть использованы для значительного ускорения расчета. Так кэш освещенности использует гладкость вторичного диффузного освещения а фотонные карты используют локализованность каустиков в пространстве.

Глава 2

Современные методы вычисления глобальной освещенности

В данной главе проводится обзор существующих алгоритмов (исключая алгоритмы, рассмотренные в предыдущей главе) и программных решений как на центральных, так и на массивно-параллельных процессорах, рассматриваются наиболее передовые методы решения проблемы глобальной освещенности.

2.1. Алгоритмы на центральном процессоре

2.1.1. Излучательность

Метод излучательности (Radiosity) в компьютерной графике был впервые представлен в работе [21]. Идея алгоритма Излучательности заключается в том, чтобы разбить поверхность сцены на множество маленьких кусочков — патчей; после чего производится расчет переноса энергии между отдельными патчами.
Алгоритм вводит следующие ограничения на входную сцену:

1. Все поверхности имеют ДФО Ламберта (отражают свет равномерно во все стороны).

2. Сцена замкнута, т.е. суммарная энергия в системе должна сохраняться.

На первом этапе алгоритма все поверхности сцены делятся на патчи. Патч – это элементарная единица поверхности. Дискретизация поверхности на патчи позволяет заменить интеграл в уравнении освещенности на конечную сумму интегралов специального вида. Каждый такой интеграл, называемый форм-фактором, задает взаимное влияние двух отдельных патчей (т.е. сколько энергии переходит от одного патча к другому). Если мы знаем значение каждого форм-фактора (для каждой пары патчей), то процесс синтеза изображений сводится к решению системы линейных алгебраических уравнений. Основная трудность в алгоритме – расчет форм-факторов [38].

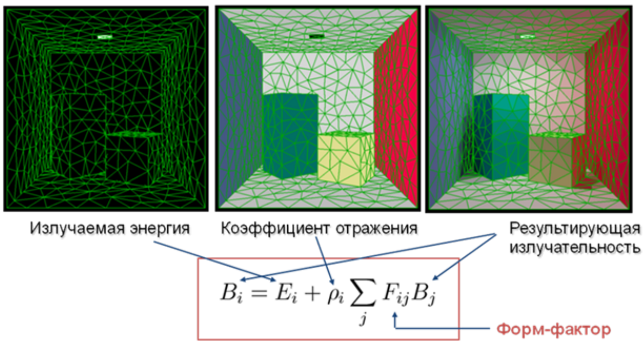


Рис. 2.1. Иллюстрация алгоритма излучательности и получаемой системы уравнений [38].

Форм фактор, в свою очередь, можно расчитать для каждой пары патчей при помощи Аналогии Нуссельта [39] зная только геометрию сцены.
Основными недостатками метода излучательности являются:

1. Поддержка только Ламбретовских материалов.

2. Артефакты на изображении (т.е. низкая точность) в местах перехода между патчами.

3. Большой объем потребляемой памяти и вычислений при увеличении точности и необходимости расчета резких переходов свет-тень даже при адаптивном разбиении (разный размер патчей в зависимости от того, насколько быстро на поверхности происходит смена света и тени).

Тем не менее излучательность достаточно часто используется для вычисления вторичного диффузного освещения, когда высокая точность не требуется. Например, в известной компьютерной игре Battlefield 3 для предрасчета вторичного освещения [40].

Мгновенная Излучательность и VPL

Основная идея метода мгновенной излучительности (Instant Radiosity; рис. 2.2) заключается в том, чтобы апроксимировать вторичное освещение в сцене при помощи некоторого числа виртуальных точечных источников света (VPL, Virtual Point Light) [41]. При этом сами виртуальные источники могут расставляться по сцене различными способами. Освещение от точечных источников света может быть вычислено при помощи трассировки лучей или метода карт теней. Однако, заслуживает внимания вопрос эффективной реализации при большом числе источников. Одно из возможных решений основывается на построении специального дерева для источников света [42].

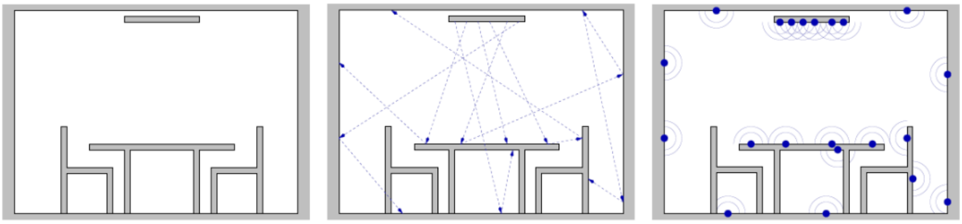


Рис. 2.2. Иллюстрация метода мгновенной излучательности.

Мгновенная излучательность породила целое направление исследования решения проблемы глобальной освещенности, которое называется 'Методом Многих Источников' (Many Light Method) [43]. Идея вычислять освещение при помощи множества источников вторичного освещения применяется в основном, если предпочтение отдается скорости, но не точности решения [43]. Здесь следует обратить внимание на работы [42], [44], [45], которые значительно улучшают точность решения проблемы глобальной освещенности при помощи виртуальных источников.

2.1.2. Подход на основе облаков точек

Данный подход (называется Point Based Global Illumination или PBGI) используется в программном решении Render Man компании Pixar [46]. Он хорошо зарекомендовал себя в производстве анимационным компьютерных фильмов (т.е. мультфильмов), поскольку позволяет достаточно быстро получить свободное от шума (что особенно важно для анимации) приближение диффузного глобального освещения.

На первом этапе алгоритм преобразует освещенные поверхности в облака точек (правильнее будет сказать элементов поверхности), сохраняющих позицию, нормаль, радиус (задающий площадь элемента поверхности) и цвет (значение освещенности). Точки называются англоязычным термином 'сюрфел'(surfel), происходящем от слово-сочетания SURFace-ELement.

После того как множество сюрфелов получено, над ними строится окто-дерево. Для каждого листа окто-дерева вычисляется излучаемая освещенность сле-

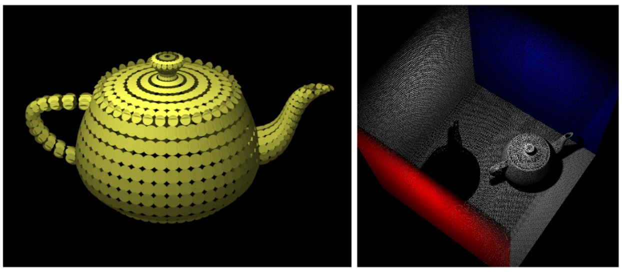


Рис. 2.3. Представление освещенной поверхности в виде множества сюрфелов [46].

дующим образом:

1. Если сюрфелы в узле мало отличаются друг от друга по нормали, вычисляется один сюрфел с усредненной позицией, нормалью, площадью и цветом.

2. Если сюрфелы сильно различаются по нормали, значение излучаемой освещенности сохраняется в представлении, основанном на сферических гармониках [47]. Причем оказывается, что коэффициенты гармоник для узла могут быть вычеслены как сумма коэффициентов отдельных сюрфелов.

Описанный выше метод применяется иерархически, строя более высокие уровни окто-дерева. При этом, интересным фактом является то, что в родительских узлах коэффициенты гармоник могут быть вычислены точно так же простым суммирование коэффициентов дочерних узлов [46].
Глобальная освещенность затем вычисляется путем рекурсивного обхода окто-дерева и растеризации отдельных сюрфелов или кластеров сюрфелов. Для того чтобы определить можно ли использовать текущий уровень дерева (представляющий целый кластер точек), предлагается контроль точности на основе телесного размера кластера (площадь проекции узла на провехность, деленная на квадрат расстояния).
Для того чтобы обеспечить учет видимости сюрфелов (т.е. обеспечить расчет вторичных теней убедившись, что сюрфелы, закрытые друими сюрфелами не вносят вклад в получаемое значение освещенности), используется растеризация геометрического приближения сцены на куб вокруг точки, в которой вычисляется освещенность (будем называть ее целевой точкой).

Для поддержки множественных диффузных отскоков света в [46] предлагается повторять алгоритм в течении нескольких проходов. Учитывая это, метод PBGI можно считать в некотором роде аналогом иерархической излучательности [48]. Интересно отметить, что в сочетании с собственно алгоритмом PBGI в работе [46] использовался модифицированный кэш освещенности для ускорения вычисления глобального освещения и отмечается его высокая эффективность.

2.1.3. Методы на основе Монте-Карло интегрирования

В предыдущей главе были детально рассмотрены методы на основе Монте-Карло интегрирования и метод фотонных карт. Чтобы избежать повторения, в данной главе они не описываются. Однако, необходимо рассмотреть работы, предлагающие значительные улучшения по сравнению с оригинальными методами а также работы основанные на комбинированных методах.

Методы на основе MLT (Metropolis Light Transport)

Kelemen style MLT Оригинальный алгоритм MLT, разработанный Э. Вичем и Л. Гибасом [27] и обсуждаемый в первой главе, использует мутации в мировом пространстве или, иначе говоря, мутации в пространстве путей. Причем, одной общей стратегии мутации для всех видов путей не существует. Такая стратегия строится путем комбинирования большого числа различных стратегий для отдельных случаев (мутации в пространстве линз, мутации каустик, мутации на источнике света и.т.д.). Это значительно осложняет реализацию, поскольку:

1. Усложняется процесс вычисления вероятностей перехода — функций $T(X|Y)$ и $T(Y|X)$. Если вероятности будут вычислены неправильно, нарушится условие, гарантирущее сходимость к правильному решению ('детальный баланс').

2. Параметры алгоритма мутации необходимо настраивать в зависимости от входных данных.

В работе [49] предложен простой и стабильный (независящий от входных данных) метод MLT, основанный на мутациях путей в ’первичном пространстве выборок’ — единичном кубе, в котором генерируются случайные числа для метода Монте-Карло. При этом, в работе проводятся некоторые выкладки, позволяющие значительно упростить вычисление вклада текущей выборки (текущего пути) в изображение, которое фактически удалось свести к 2 простым формулам [49]. Причем, алгоритм [49] может хорошо работать на темных участках (где оригинальный MLT работает, как правило, хуже чем обыкновенная Монте-Карло трассировка путей поскольку отводит на темные области меньшее число выборок) [49].

Energy Redistribution Path Tracing (ERPT) Алгоритм ERPT [50] является гибридом MLT и идеи фильтрации, о которой пойдет речь далее. В отличие от MLT, ERPT использует кототкие последовательности мутаций и перераспределяет энергию путей на плоскости изображения. В отличие от фильтров и кэша освещенности ERPT позволяет получить несмещенное решение. Идея, которая лежит в основе алгоритма ERPT, называется ’поток энергии’ (energy flow) и основывается на том факте, что интегралы освещенности для соседних пикселей, как правило, кореллируют.

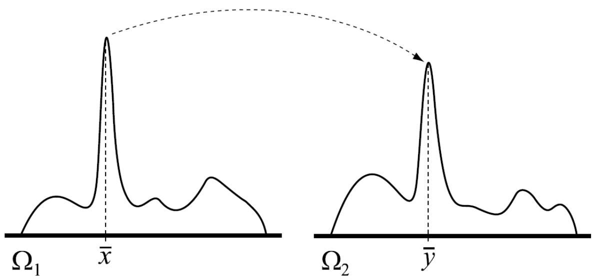


Рис. 2.4. Иллюстрация идеи ’потока энергии’ в алгоритме ERPT. Поток энергии связывает коррелилующие интегралы и устанавливает перенос энергии между ними.

Предположим, что мы имеем два кореллирующих интеграла I_1 и I_2 на соответствующих областях определения их под-интегральных функций Ω_1 и Ω_2 (рис. 2.4). Далее предположим, что в процессе интегрирования I_1 в Ω_1 была найдена точка $\overline{x}$ с высокой значимостью. Поскольку I_1 и I_2 коррелируют, высока вероятность найти на области определения Ω_2 недалеко от точки $\overline{x}$ точку

$\overline{y}$, которая также будет иметь высокую значимость. В действительности эта же самая идея лежит в основе MLT, однако, в алгоритме ERPT ослабляются ограничения на реализацию потока энергии (рис. 2.5). Вместо детального баланса потока энергий используется так называемый общий баланс. Общий баланс потока энергий требует, чтобы суммарная входящая и выходящая энергии для точки были равны. Тогда как детальный баланс является более строгим ограничением и требует, чтобы для каждой пары точек энергия, перетекаемая между ними, сохранялась [50].

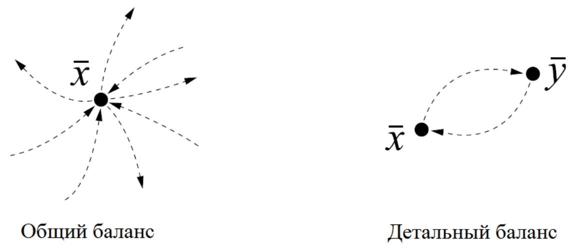


Рис. 2.5. Общий (general balance) и детальный (detailed balance) баланс потока энергий.

ERPT применят так называемый фильтр сбалансированного потока энергий (balanced energy flow filters) – набор правил, при помощи которых поток энергии может быть реализован таким образом, чтобы условие общего баланса сохранялось. Сравнения, предоставленные в работе [50], позволяют сделать вывод о том, что методы MLT и ERPT – приблизительно одинаковы по эффективности.

Gradient Domain MLT В работе [51] было предложено расширение оригинального алгоритма MLT, позволяющее улучшить сходимость метода. Алгоритм сначала вычисляет вертикальную и горизонтальную разницу (градиенты) изображения, а также грубое приближение самого изображения при помощи обыкновенного алгоритма MLT, после чего производится реконструкция финального изображения решая уравнение Пуассона.

Метод соединения вершин (Vertex Connection Merging, VCM)

В работах [26], [52] с целью улучшения алгоритма двунаправленной трассировки путей на путях вида SDS был предложен метод соединения вершин

(Vertex Connection Merging, VCM). Идея основывается на том, чтобы соединять пути на диффузной поверхности даже если они попали не строго в 1 точку, а находятся друг от друга на расстоянии некоторого небольшого радиуса r(рис 2.6). Данный метод может рассматриваться как расширение финального сбора, при котором вместо фотонов хранятся пути от источника света, а вместо сбора непосредственно освещенности применяется полностью аналогичное BDPT соединения вершин пути сбора и путей из источника, закончившихся на той же поверхности в некотром радиусе от точки сбора.

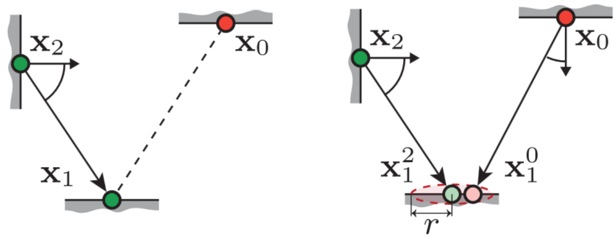


Рис. 2.6. Иллюстрация алгоритма VCM. Соединение пути в классическом BDPT изображено слева. Соединение в VCM — справа.

Регуляризация в пространстве путей

В работе [53] предложена похожая на метод соединения вершин идея о том, что можно соединять точки различных путей для переотражений вида SDS даже если при зеркальном отражение (S) попадание в точку где произошло диффузное отражение (D) неточно (рис 2.7). Однако, данный метод отличается тем, что использует преобразование S отражения в D отражение и может быть применен ко всем видам Монте-Карло интеграторов, а не только к BDPT, как метод соединения вершин.

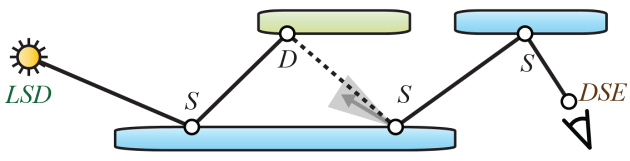


Рис. 2.7. Иллюстрация процесса регуляризации в пространстве путей.

В работе [53] отмечается, что угол регуляризации (помеченный на рисунке 2.7 серым треугольником) может быть вычислен из параметра радиуса сбора в методе VCM (то есть сам способ соединения похож на способ в методе

VCM). Однако, в отличие от VCM, в данном методе будут соединяться не все пути, а только так называемые ’несэмплируемые’ пути. То есть те пути, которые не могут быть получены без внесения смещения методами Монте-Карло (в любом виде — PT, BDPT, MLT, ERPT и.т.д.). Это позволяет уменьшить смещение (bias) в получаемом решении. Помимо вышеперечисленных отличий, важным результатом метода [53] по сравнению с предыдущими работами является возможность получения изображения точечных (или бесконечно малых) источников света при многократных зеркальных переотражениях.

Следует обратить внимание на работу [25], предлагающую эффективную реализацию усеченной двунаправленной трассировки путей с применением регуляризации на графических процессорах.

2.1.4. Методы на основе фотонных карт

Метод фотонных карт, рассмотренный в первой главе, имеет весьма обширный простор для улучшений. Одним из наиболее практичных методов, позволяющих решить большинство проблем оригинального алгоритма фотонных карт является алгоритм Стохастических Прогрессивных Фотонных Карт (SPPM), рассмотренный в 1 главе. Однако, заслуживают рассмотрения методы, дополнительно улучшающие фотонные карты сами по себе, а также альтернативные SPPM решения и комбинированные методы.

Оценка плотности (сбор освещенности)

Алгоритм фотонных карт для решения проблемы глобальной освещенности породил подзадачу ’оценки плотности фотонов’ (Photon Density Estimation) [34]. Для решения этой задачи применяются различные подходы.
В работе [54] был предложен метод, позволяющий значительно снизить смещение (bias) в получаемом решении и сделать изображение каустик более четкими. Идея метода состоит в том, чтобы хранить дополнительно на каждый фотон так называемый ’дифференциал фотона’ (photon differential) по аналогии с дифференциалами лучей [55] (рис. 2.8). Дифференциалы фотонов позволяют апроксимировать одним лучом целый пучок фотонов. Именно этот апроксимируемый пучок фотонов и называется дифференциалом фотона в ра-

боте [54].

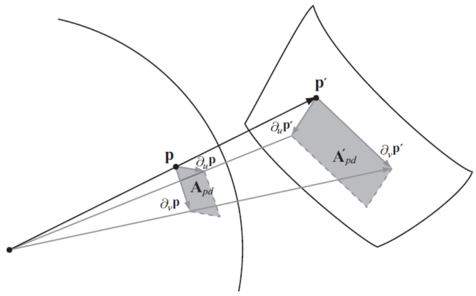


Рис. 2.8. Трассировка луча и его дифференциальных векторов от точки p к точке p’.

Трассировка таких пучков технически не отличается от обыкновенной трассировки фотонов. При взаимодействии с поверхностью дифференцыалы фотонов пересчитываются аналогично диффернциалам лучей в работе [55]. Дифференциалы используются далее во время сбора для повышения точности. Обыкновенные фотоны вносят вклад в интеграл освещенности на поверхности в виде круга. Дифференциалы фотонов вносят вклад в виде эллипса, делая, таким образом, фильтрацию анизотропной. Метод дифференциалов фотонов требуют хранения дополнительно 24 байт на фотон и увеличивает объем вычислений, выполняемых при сборе за счет умножения матрицы 3x3 на вектор разницы между позицией фотона и точкой сбора. Тем не менее, повышение точности решения значительное, что позволяет считать метод практичным.

В работе [56] предолжен подход, называемый ’прогрессивной релаксацией фотонов’ (progressive photon relaxation). Ключевым моментом метода является использование фиксированного объма памяти и фиксированного числа сохраняемых фотонов, называемых в работе ’устойчивыми фотонами’ (persistent photons). Метод основан на построении диаграмм Вороного с последующей регуляризацией ячеек диаграммы на основе информации о вновь поступающих, временных (transient), фотонах. В среднем по эффективности метод не на много превосходит стохастические прогрессивные фотонные карты и является достаточно сложным в реализации. Преимущество метода релаксации заключается том, что он работает в мировом пространстве и позволяет получить решение высокой точности, не производя оценку освещенности на каждом проходе (что требуется, например, в методах PPM и SPPM). Можно сказать, что метод позволяет построить карту освещенности высокой степени детализации и качества. В числе плюсов метода также следует отметить построение структуры с высокой степенью регулярности, что может положительно сказаться на производительности сбора освещенности при реализации на графических процессорах.

В работе [57] для ускорения финального сбора был предложен подход, называемый обратными фотонными картами (reverse photon mapping). Основная идея метода заключается в том, чтобы вместо фотонов запоминать лучи сбора в kd-дереве специального вида. После чего производить трассировку фотонов, накапливая результат для лучей сбора. Серьезным недостатком метода является большой объем потребляемой памяти, поскольку число лучей сбора, как правило, чрезвычайно велико.

Комбинированные методы на основе фотонных карт

Идея разделения освещенности на независимые компоненты, по-видимому, впервые была высказана в виде алгоритма кэша освещенности [36]. В дальнейшем многие исследователи приходили к тому, что вычислять отдельные компоненты освещенности можно более эффективно за счет априорных знаний о природе компоненты и применении соответствующего алгоритма [58], [33], [59]. В работе [60] аппробирована идея комбинирования метода Монте-Карло и прогрессивных фотонных карт с целью использования преимущества обоих методов. Освещение разделялось на 2 части: освещение, вызванное каустиками (пути вида $E(S|G)^{+}DL$ или $E(S|G)^{+}D(S|G)^{+}L$), и все остальное освещение. Первую часть освещения предлагается вычислять при помощи метода стохастических прогрессивных фотонных карт, а вторую при помощи стандартной Монте-Карло трассировки путей. Поскольку множества путей из первой и второй части не имеют пересечений, а их объединение составляет все множество путей, метод приводит к корректному результату. В работе [61] предложен улучшенный способ комбинирования результата при помощи взвешивания на основе многократной выборки по значимости.

В работе [62] был предложен иной способ комбинирования фотонных карт и Монте-Карло трассировки. Вместо того чтобы использовать фотонные карты для вычисления непосредственно освещенности, оценка освещенности на основе фотонной карты использовалась для вычисления плотности вероятности падающего освещения. После чего к методу Монте-Карло применялась выборка по значимости на основе ДФО и посчитанной плотности вероятности падающего освещения. Этот метод является альтернативой MLT, когда падающее освещение неравномерно в высокой степени.

2.1.5. Методы на основе кэша освещенности

Помимо исследований методов вычисления интеграла освещенности, значительное число работ направлено в сторону ускорения вычисления интеграла освещенности на множестве точек. Идея кэширования освещенности получила множество направлений развития.

В работах J.Krivanek-a [63], [64], [37] предложено улучшение контроля ошибки экстраполяции, многопроходный адаптивный алгоритм, который, при добавлении новой точки в кэш, сравнивает относительное отклонение освещенности в соседних точках, что уменьшает количество артефактов (но приводит к большому числу дополнительных вычислений) и расширение метода кэширования освещенности на умеренно-зеркальные ДФО. Падающая освещенность сохранялась при помощи сферических гармоник [47]. Значительным преимуществом кэша падающей освещенности над простым кэшем освещенности является способность ускорять вычисления на поверхностях с высокой геометрической детализацией на основе микро-релефа.

В [65] описывается использование процедуры репроекции вместе с кэшем освещенности. Идея репроекции состоит в том, что точки пересечения лучей и поверхностей сцены, которые были вычислены ранее и записаны в кэш, проецируются на полусферу в новой точке поверхности, в которой необходимо вычислить освещение. Недостатком подхода, описываемого в [65], является наличие 'пустот' на полусфее вокруг точки, на которую выполняется репроекция. Репроекция получила дальнейшее развитие в работе [66], где устранены основные недостатки кэша падающей освещенности на основе репроекции за счет специального алгоритма расслоения репроецируемых точек (это позволяет корректно реализовать экстраполяцию освещения от нескольких записей кэша с учетом закрытия одних объектов другими) и перетрассирвок в местах, где были обнаружены пустоты, и представлена высокоэффективная параллельная реализация на основе SSE инструкций центрального процессора.

В работе [66] также высказывается идея о том, что можно использовать кэш падающей освещенности для оценки плотности вероятности падающего освещения, чтобы ускорить сходимость метода Монте-Карло, если необходимо получать несмещенное решение.

В работе [67] предложено разделять падающеее освещения на дальнее и ближнее с тем, чтобы сохранять только дальнее освещение в кэше падающей освещенности. Поскольку освещение от дальних объектов меняется, как правило, значительно более плавно чем освещение от ближних, сохранение в кэше только 'дальней' освещенности позволяет значительно снизить число записей кэша. Ближнее освещение в работе [67] вычислялось при помощи апроксимации, основанной на методе излучательности без учета функции видимости. Это значительно ускоряет вычисление ближнего освещение, однако, приводит к некорректному результату. Тем не менее, развитие идеи разделения падающего освещения на ближнее и дальнее с корректным вычислением ближнего освещения, освещенное в [66], можно считать перспективным направлением.

2.1.6. Методы на основе фильтрации

Идея кэширования освещенности, обсуждаемая ранее, говорит о том некоторую функцию освещенности можно вычислить на небольшом подмножестве точек трехмерного пространства, а в остальных точках интерполировать. Скорость возрастает, таким образом, за счет пере-использования информации и уменьшения объема вычислений.

Фильтрация (рис. 2.9) ускоряет расчет освещения за счет того же эффекта — переиспользования информации. Однако, вместо того чтобы вычислять значение функции в нескольких точках с высокой точностью, этот метод принимает на вход Монте-Карло выборки. А именно: результирующую яркость путей с некоторой дополнительной информацией — позицией первого пересечения, нормалью к поверхности в этой точке, текстурными координатам, координатами на линзе (если моделируется эффект глубины резкости) и случайными параметрами самого пути. Таким образом, строится многомерное пространство, где каждое измерение является некоторым параметром пути.

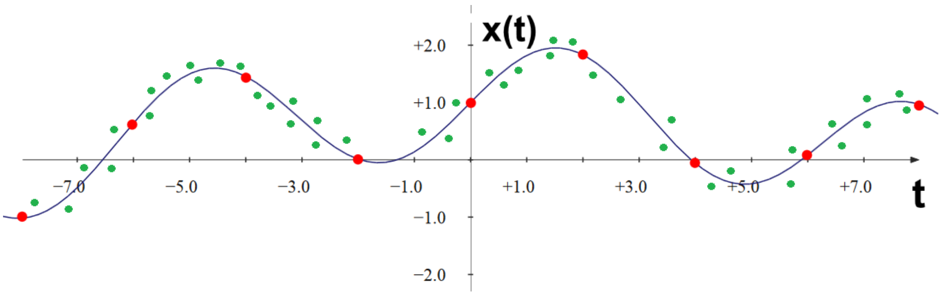


Рис. 2.9. Интерполяция и фильтрация. Красные точки обозначают места, в которых значения функции $x(t)$ были вычислены точно. Зеленые точки обозначают значения точечных выборок в методе Монте-Карло при оценки функции $x(t)$ на множестве точек оси t. Применив к ним операцию фильтрации, можно восстановить с некоторой точностью искомую функцию $x(t)$.

В этом пространстве затем производится фильтрация. Одно из наиболее качественных решений было получено в работе [68], где фильтрация производится с учетом зависимости между случайными величинами в методе трассировки лучей Монте-Карло и значениями в полученной выборке.

Основной проблемой подходов на основе многомерной фильтрации является высокая алгоритмическая сложность многомерных фильтров. Однако, следует отметить работу [69], где предложен подход с линейной сложностью в зависимости от числа измерений и количества пикселей изображения и, с учетом применения графических процессоров, достигнута скорость обработки поступающих сэмплов (выборок) порядка десятков кадров в секунду в разрешении 10 мега-пикселей.

2.2. Выводы по методам вычисления глобальной освещенности

Несмотря на значительное число существующих методов комьютерной графики, сфокусированных на решении проблемы глобальной освещенности, область продолжает активно исследоваться. Об этом свидетельствует анализ объема публикаций, предлагающих новые методы за 2009-2014 годы. Среди причин, способствующих неугасающему интересу исследователей к проблеме глобально освещенности, следует отметить:

1. Высокую актуальность проблемы и широкий спектр применения ее ре-

шений. От индустрии развлечений до точной визуализации промышленных изделий, оптического моделирования и расчета переноса нейтронов в реакторах.

2. Высокую вычислительную сложность алгоритмов и большое время расчета. Ни одно современное исследование не обходится без анализа эффективности предложенного метода.

3. Специфическими требованими области (где-то требуется точноть, где-то скорость, где-то простота реализации) и, как следствие, высокой специализацией многих методов.

4. Достаточно высокую сложность алгоритмов в плане реализации и поддержки коллективом программистов и стремление исследователей сделать алгоритмы проще, а решения — практичнее.

Среди практичных подходов, обладающих высокой точностью, следует отметить Монте-Карло трассировку, усеченную двунаправленную трассировку, Kelemen-Style MLT, подходы на основе регуляризации, соединения вершин, фотонных карт, кэша освещенности, а также комбинированные подходы. Поскольку одним из важнейших факторов является производительность, следует обратить особое внимание на работы по решению проблемы глобальной освещенности на параллельных и массивно-параллельных системах. Это особенно важно в связи с ростом популярности массивно-параллельных процессоров.

2.3. Алгоритмы на массивно-параллельных процессорах

Особенности массивно-параллельных процессоров и массивного паралеллизма самого по себе делают во многих случаях невозможным применение алгоритма, разработанного для центрального процессора один в один. Более того, даже если это возможно, в целях лучшей производительности алгоритм приходится сильно модифицировать. Метод Монте-Карло трассировки путей является примером такого алгоритма. Несмотря на то, что каждый отдельный путь можно обрабатывать независимо, для достижения максимальной (или хотя бы приемлемой) эффективности необходимо произвести ряд модификаций алгоритма (рассмотрены ниже). В связи с этим, исследования в области решения проблемы глобальной освещенности на массивно-параллельных системах можно условно поделить на 2 части:

1. Исследования о том, как эффективно реализовать некоторый известный алгоритм на GPU, если принципиальные трудности по его реализации отсутствуют, однако, по каким-либо причинам, оцениваемая производительность не достигается. Будем условно называть данную группу исследований 'исследованиями производительности алгоритмов на GPU'.

2. Исследования о том, как реализовать на массино-параллельной системе алгоритм, изначально спроектированный для однопроцессорной системы. Такой алгоритм в силу своей природы не может быть напрямую реализован на массивно-параллельной системе и требует переработки в самих основах. Будем называть эту группу 'исследованиями по распараллеливанию алгоритмов на GPU' или 'исследованиями по созданию высоко-параллельных алгоритмов'.

Настоящая работа в большей степени относится ко второй группе исследований. Однако, вопрос эффективной реализации важен и исследования производительности с учетом архитектурных особенностей GPU также затрагиваются.

2.3.1. Трассировка лучей на GPU

Ранние работы по реализации трассировки лучей на GPU исследовали проблему эффективной реализации без-стекового поиска в ускоряющих структурах в условиях ограниченных возможностей графических API DirectX 9 и OpenGL2 и их аппаратных реализаций [70], [71], [72].
Можно легко ошибиться в предположении, что с появлением новых технологий и GPU с более широкими возможностями и ресурсами задача реализации трассировки лучей на современных GPU стала тривиальной, поскольку достаточно лишь назначить каждому потоку свой луч и портировать существующий

CPU код. В действительности это не так. Портированный ’как есть’ код будет обладать чрезвычайно низкой эффективностью вследствие особенностей выполнения кода на GPU [5]. Для создание эффективного кода необходимо применять такие подходы как ’убер-ядро’ и ’разделенное ядро’ [5].

Конвейер трассировки лучей

Проблемы подхода убер-ядра обсуждаются позднее в [24] и [73]. В работах [5] и [24] предлагается использовать подход ’разделенного ядра’, формируя и настраивая конвейер трассировки лучей за счет замещения и/или вставки ядер на различных этапах конвейера. Этот способ предполагает реализацию различных частей в отдельных ядрах и организацию взаимодействия между ними путем сохранения данных в глобальной памяти. Работа [73] развивает данный подход на основе очередей на выполнение работы (каждая очередь запускает на выполнение только одно конкретное ядро), сортировки потоков по номеру ядра и механизма создания работы на GPU.

2.3.2. Монте-Карло трассировка путей на GPU

Несмотря на тот факт, что Монте-Карло трассировка достаточно легко может быть реализована на GPU, ее эффективная реализация вызывает ряд определенных проблем, связанных с природой графических процессоров.

Контроль глубины трассировки. Различные потоки заканчивают трассировку на различной глубине. Это приводит к тому, что через некоторое количество переотражений внутри группы потоков warp [74] лишь небольшое число потоков остаются активными; тем не менее вся группа в этом случае продолжает выполняться на мультипроцессоре.

В работе [75] представлено простое и эффективное решение данной проблемы, называемое русской рулеткой на warp — ’per warp russian roulette’. Основная идея данного метода заключается в том, чтобы применять метод русской рулетки не к каждому потоку отдельно (как было рассмотрено в 1 главе), а сразу ко всей группе потоков warp. Решение о терминировании, таким образом, принимается сразу для всей группы, что исключает возникновение групп warp с малым числом активных потоков при терменировании лучей посредством русской рулетки.

Для решения проблемы неоднородной загрузки GPU из-за различной глубины терминирования потоков, в работах [75] и [24] исследовался метод повышения эффективности использования GPU, называемый регенерацией путей. Сообщается об ускорении от 20 до 100% для случаев, требующих учет большого числа переотражений. Метод был впервые предложен в [76] и иследован на стандартной трассировке путей, двунаправленной трассировке путей и алгоритме MLT. В работе [76] сообщается об ускорении от 20 до 50%. Метод работает следующим образом:

1. Для трассируемых лучей выделяется буффер, вмещающий в себя данные ровно для N лучей.
2. При каждом переотражении производится операция уплотнения с тем чтобы отсечь мертвые потоки.
3. В освободившееся место добавляются новые лучи с тем чтобы загрузить GPU работой.

Несмотря на указанные преимущества в скорости, метод регенерации путей не лишен недостатоков. Во-первых, он вынуждает применять уплотнение, которое занимает определенное время. Во-вторых, он нарушает когерентность лучей в буфере что может привести к падению производительности. Наконец, метод требует значительной модификации архитектуры и кода самого трассировщика путей.

Дивергентные потоки. По сравнению с трассировкой лучей, трассировка путей обладает значительно большей дивергентностью потоков, поскольку при диффузном отражении лучи расходятся по полусфере в различные стороны. Это приводит к неэффективному использованию мультипроцессоров и, как следствие, низкой производительности.

В работе [77] описывается подход, называемый когерентной трассировкой путей. Идея данного подхода заключается в том, чтобы использовать одно состояние случайного генератора для всех лучей (в группе warp [74], блоке или вообще всех на экране).

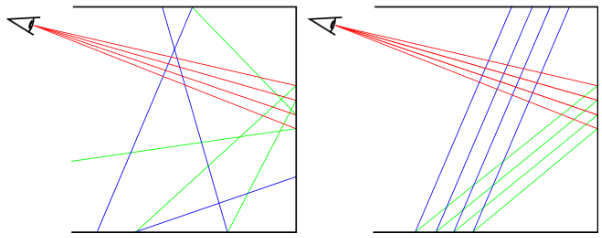


Рис. 2.10. Стандартная трассировка путей (слева). Когерентная трассировка путей (справа). [77].

Данный подход является эффективным но, меняет характер артефактов, превращая шум в более заметные для глаза артефакты с резкими границами. Для того чтобы снизить заметность границ, в [77] применяется случайное перемешивание к схеме нумерования пикселей.

Более сложные методы на основе Монте-Карло трассировки путей на GPU

Усеченная двунаправленная трассировка. В работе [24] предлагается алгоритм усеченной двунаправленной трассировки путей. Данный алгоритм фактически комбинирует обратную (Path Tracing) и прямую (Light Tracing) Монте-Карло трассировку в пространстве изображения. Иными словами, результирующее изображение может быть получено путем простого сложения (или сложения с весами на основе многократной выборки по значимости) изображений, получаемых прямой и обратной Монте-Карло трассировкой. Данный алгоритм не требует значительных модификаций по отношению к простой реализации трассировки путей на GPU и является в высокой степени практичным решением.

Двунаправленная трассировка путей на GPU. Основной сложностью в реализации двунаправленной трассировки путей на GPU является корректная реализация многократной выборки по значимости. В классической реализации двунаправленной трассировки для корректного вычисления весов MIS необходимо сохранять в памяти информацию о всех точках пересечения для прямого и обратного путей. Это создает высокие расходы памяти и вынуждает ограничивать глубину трассировки, поскольку GPU должен обрабатывать одновременно большое число путей — и для всех этих путей придется сохранять информацию о точках пересечения с поверхностью, нормалях и материалах.

В работе [78] предложен подход, называемый 'Рекурсивная многократная выборка по значимости' (Recursive Multiple Importance Sampling). Этот подход позволяет решить проблему высоких затрат памяти. Вместо того, чтобы соединять каждую вершину пути из источника и камеры, в работе [78] на каждую вершину пути из источника строился полный путь из камеры. Благодаря этому, на каждом шаге трассировки пути из камеры производится только одно соединение и необходимость в хранении всех вершин пути отпадает. Аналогичное решение, называемое 'Light Vertex Cache', используется и в работе [79].

MLT и ERPT на GPU. В работе [78] исследовался вопрос эффективной реализации алгоритма ERPT на GPU. Основная проблема реализации алгоритмов, близких по своей идеи к MLT, заключается в необходимости хранить случайные числа на протяжении всего пути для каждого пиксела. В [78] говорится, что поскольку в ERPT в отличии от MLT все лучи в группе имеют одинаковую глубину, в ERPT можно реализовать эффективную схему динамического выделения памяти при помощи атомарных операций, за счет чего автору [78] удалось снизить необходимый объем памяти на 60%. Тем не менее, в условиях ограниченной памяти GPU, это решение всё ещё нельзя назвать приемлимым, поскольку нетрудно посчитать, что в высоких разрешениях это приведет к значительным затратам памяти (более 50% доступной памяти) GPU.

Соединение вершин (Vertex Connection Merging) на GPU. Наряду с алгоритмом соединения вершин (VCM) в работе [79] были исследованы также алгоритмы двунаправленной трассировки путей и алгоритмы на основе фотонных карт на GPU и проведено сравнение алгоритмов. Основная проблема, на которую приходится обращать внимание при реализации алгоритма соединения вершин, — та же проблема высоких трат памяти что и в двунаправленной трассировке лучей. Предложенные в работах [78] и [79] подходы для двунаправленной трассировки лучей работают и для метода соединения вершин.

2.3.3. Параллельный Кэш освещенности

Реализация параллельного кэша освещенности даже на центральном процессоре сопряжена с рядом трудностей.

1. Первая проблема (проблема синхронизации) связана с обеспечением целосности данных кэша — каждый поток должен видеть один и тот же экземпляр данных кэша.

2. Вторая проблема (последовательная природа) связана с тем, что алгоритм вычисляет расположение записей кэша и их радиусы валидности на основании эвристических методов, опираясь на то, что точки вставляются в кэш последовательно, одна за одной [37].

Один из простых способов решения проблемы синхронизации, используемый в [66] заключается в том, чтобы не синхронизировать данные кэша между потоками вообще. Первый недостаток этого подхода состоит в том, что на изображении могут появиться видимые границы между областями, которые были использованы разными потоками т.к. каждый поток будет иметь доступ к своему уникальному подмножеству записей кэша — и, как следствие, в одной той же точке трехмерного пространства для разных потоков результат интерполяции будет различен. Второй недостаток заключается в том, что при увеличении числа потоков будет возрастать объём избыточных вычислений, поскольку каждый поток будет вычислять полностью независимый набор записей, а их перекрытие никак не контролируется.

Следующий способ решения проблемы синхронизации — использование разделяемой структуры данных в том или ином виде. Здесь возможны различные варианты — синхронизация при вставке единичной записи [80], синхронизация на основе слияния групп [81] и более сложные подходы — [82], [83].

В работе [82] предложен подход, использующий пространственную когерентность кэша освещенности для реализации алгоритма на кластерных системах. Все узлы кластера разбивались на 2 подмножества — узлы, занимающиеся вычислением записей кэша, и узлы, отвечающие за построение изображения в целом. За счет использования такого разделения и высокой пространственной когерентности операций чтения из кэша, авторам [82] удалось сохранять частоту обновления кэша на высоком уровне без существенных потерь в производительности.

Работы [80], [81] и [82] используют для синхронизации критические секции. Корректная реализация с использованием критических секций блокирует как запись в кэш, так и чтение из него. Это негативно сказывается на масштабируемости реализации при увеличении числа параллельно исполняющихся потоков [83]. В работе [83] предлагается 'неблокирующий' подход к построению разделяемой структуры кэша освещенности на основе атомарных инструкций современных CPU.

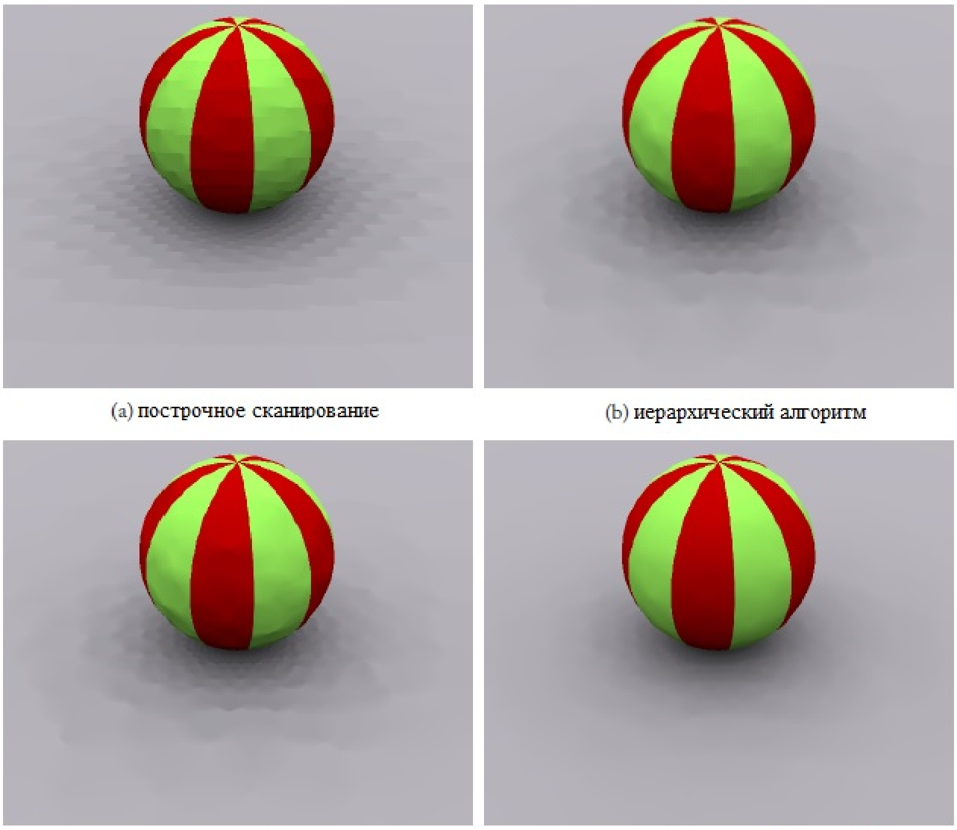
Наконец, еще один способ решения проблемы синхронизации заключается в том, чтобы отделить процесс построения кэша освещенности от процесса его использования и разделить, таким образом, процесс построения изображения на 2 стадии:

1. Стадия построения кэша освещенности. На данной стадиии необходимо покрыть всё пространство сцены, куда попадают лучи из камеры, записями из кэша. При этом, на данной стадии целосность кэша не важна, поскольку результат интерполяции не испольуется для вычисления финального изображения.

2. Финальная стадия построения изображения. На ней новые записи в кэш освещенности не добавляются. Таким образом, кэш остается консистентным.

Идея такого подхода обсуждается в [64], [37] и [84]. Причем, в указанных работах двухстадийный алгоритм используется не для параллельной реализации, а для повышения точности интерполяции (рисунок 2.11). Ранее была рассмотрена ситуация, при которой разные потоки, имея различные подмножества записей кэша в одной и той же точке пространства, будут давать различные результаты интерполяции.

В действительности эта проблема возникает даже при однопоточной реализации кэша освещенности, поскольку в некоторой точке пространства X в



(a) построчное сканирование (b) иерархический алгоритм

(c) двухпроходной алгоритм (d) двухпроходной алгоритм с размытием

Рис. 2.11. Сравнение различных подходов построения изображения с применением кэша освещенности из работы [37]. Под 'размытием' (случай d) в данном случае понимается небольшое увеличение радиусов валидности для всех записей (в 1.6 раза).

разные моменты времени при интерполяции может быть затронуто различное число записей [37]. Это возможно, так как стандартный алгоритм не может гарантировать, что в процессе построения изображения вблизи точки X не появятся новые записи, которые станут участвовать в интерполяции при следующих выборках из кэша в точке X.

Поскольку разделение процесса построения изображения на 2 стадии позволяет повысить точность, будем рассматривать данный подход как наиболее практичный способ решения проблемы синхронизации, т.к. повышение точности важно при реализации кэша освещенности. При этом, что касается проблемы последовательной природы кэша освещенности, здесь следует выделить 2 основных момента.

Первый момент — алгоритм создания самого множества записей кэша не предполагает одновременную вставку многих точек в кэш. Причем, здесь существуют 2 проблемы:

1. Проблема остановки алгоритма построения кэша. Сам процесс расстановки записей реализует ленивую схему вычислений. Оригинальный алгоритм (1) вычисляет новые записи кэша по запросу. Поэтому в нем проблема остановки не существовала. При необходимости новая запись всегда может быть вычислена. В двух-проходной схеме это не так. Процесс создания кэша освещенности должен уметь останавливаться. Реализация из [64] использует последователный алгоритм (называемый 'Full convergence for a set of shading points'). На первом проходе алгоритм запоминает в kd-дереве все точки, в которых потребуется осуществляь выборку из кэша (shading points). При вставке новой записи в кэш освещенности, из дерева удаляются точки (shading points), покрытые областью, на которую распространяется влияние вставляемой записи. Существенным недостатком данного подхода является необходимость хранить в памяти все точки 'shading points'. При использовании стохастической трассировки лучей или трассировки путей хранить все точки, в которых возможна выборка их кэша не представляетя возможным, т.к. число эти точек стремится к бесконечности. В [37] предлагается использовать упрощение рассмотренного выше алгоритма, помечая пикселы изображения, для которых не встретилось высоких значений оценки ошибки интерполяции. Когда все пикселы помечены, можно остановить алгоритм. Хотя параллельная реализация обоих решений теоретически возможна, эта проблема требует исследования, поскольку здесь возникает все та же необходимость синхронизации структур данных, хранящих shading points или маркеры пикселей.

2. Проблема выбора оптимального расположения записей кэша. В рассмотренных ранее подходах при увеличении числа одновременно вставляемых в кэш записей неизбежно падение эффективности решения в целом. Это связано с тем, что при большом числе одновременно вставляемых точек (записей), некоторое число запросов на вычисление освещенности могут возникнуть в близко расположенных областях пространства. Такие запросы являются избыточными, поскольку в этих областях про-

странства последовательный алгоритм обошелся бы всего одной записью.

Второй момент — алгоритм вычисления радиусов валидности. Эвристика Neighbour Clamping [37], значительно повышающая точность решения на основе Кэша освещености, предполагает последовательную вставку записей в кэш. То же верно и об алгоритме контроля плотности, описываемом в разделе 5.1 работы [64]. Отсутствие учета соседства точек при параллельной вставке большого числа независимых записей приведет к тому, что радиусы валидности записей будут слишком большими. Как будет показано в главе 3 — это, в свою очередь, ведет к высокой степени перекрытия записей и медленной интерполяции.

Кэш освещенности на GPU

Впервые GPU был использован при реализации кэша освещенности в работе [85] и в дальнейшем обсуждается в [37]. Рассматриваемый в этих работах подход позволяет заменить стандартный алгоритм интерполяции на растеризацию сплатов (дисков). Это позволяет избежать поиска в дереве на GPU при финальной визуализации сцены, что было важно для ранних GPU, не имеющих поддержку циклов. В работах [85] и [37] также предлагается использовать растеризацию на GPU при расчете освещенности в точках кэша. Метод, обсуждаемый в этих работах, имеет следующие ограничения:

1. Интерполяция при помощи растеризации сплатов позволяет визуализировать вторичное освещение только на первично видимых поверхностях. Хотя существуют методы визуализации отражений на основе растеризации (и в этом случае, растеризация сплатов также применима), такие методы работают для специальных случаев и/или визуализируют отражения не совсем корректно [86]. Поэтому при точной визуализации и/или визуализации на основе трассировки лучей данный метод неприменим.

2. При использовании растеризации для вычисления непрямого освещения значения освещенности, сохраняемые в кэше, несут информацию только

о первом переотражении. Такое решение неприемлимо, если необходимо учитывать большее число переотражений света. Хотя этот недостаток можно побороть при использовани фотонных карт в сочетании с картами светимости (iradiance maps), эффективность такого подхода может оказаться низкой, поскольку алгоритм растеризации для каждой точки вынужден полностью пропускать всю геометрию по графическому конвейеру и плохо подходит для вычисления освещенности на множестве точек.

2.3.4. Фотонные карты на GPU

Фотонные карты на GPU – довольно популярная тема для исследований. Одна из основных трудностей при реализации фотонных карт – эффективное построение ускоряющих структур для последующего поиска ближайших фотонов.

В работе [87] алгоритм фотонных карт был впервые реализован на GPU. Аппаратные ограничения GPU того времени не позволяли производить запись в произвольную область памяти, что создавало целый ряд проблем при реализации ускоряющих структур на GPU. В [87] предложено 2 метода построения ускоряющих структур на основе регулярной сетки. Первый метод основан на сортировке фотонов по номеру воксела (voxel), в который он попадает. После чего при помощи бинарного поиска находился первый фотон в отсортированном массиве фотонов и формировался список индексов. Второй метод использует аппаратные возможности на основе буфера трафарета и является более быстрым, но позволяет сохранять ограниченное число фотонов на каждый воксел. Недостатком обоих методов из работы [87] является высокий расход памяти.

В работе [88] был предложен эффективный алгоритм построения kd-деревьев с учетом эвристики VVH [89] и алгоритм поиска k-ближайших фотонов. Алгоритм из работы [88] изначально ориентирован на построения kd-дерева над множеством полигонов. Он использует 2 различных подхода в зависимости от количества примитивов в узле и хранит данные в динамических списках на GPU. Алгоритм [88] сложен в реализации и, кроме того, расходует больше памяти, чем его CPU аналог. Производительность поиска k-ближайших фотонов при этом низкая вследствие итеративного поиска оптимального радиуса сбора.

В [5] в целях упрощения алгоритма построения и повышения его эффективности был предложен подход на основе предрассчитанного kd-дерева и сортировки ндексов фотонов на CPU при помощи механизма быстрых списков. Алгоритм построения результирующей структуры имеет линейную сложность. Среди недостатков данного подхода следует отметить необходимость априорного знания о геометрии сцены (на основе которого предрассчитывается kd-дерево) и копирование данных между CPU и GPU.

В [90] для ускорения сбора освещености было использовано BVH-дерево и фиксированный радиус сбора. Среди алгоритмов построения BVH дерева на GPU достаточно эффективны подходы из работ [91, 92]. Использование BVH дерева в целом является неплохим решением, и основным недостатком этого и других подходов, использующих классические деревья, является рекурсивный алгоритм обхода дерева при сборе освещенности. Причины падения производительности при использовании такого обхода — высокая степень дивергентности потоков на GPU и использование локальной памяти.

В работе [93] предложен эффективный подход построения ускоряющей структуры на основе лексикографической сортировки координат фотонов. После окончания трассировки все фотоны сортировались в лексикографическом порядке по координатам соответствующих вокселей. При таком выборе оператора сравнения все фотоны, принадлежащие одному вокселю, являются эквивалентными и занимают непрерывный сегмент в отсортированной фотонной карте. Над отсортированным массивом строилась воксельная ускоряющая структура на основе 3D текстур. Данный метод похож на алгоритм из работы [87] и его основным недостатком является высокий расход памяти.

В работе [94] предложен метод пространственных хэш-таблиц на основе ’хэширования кукушки’. Авторы не исследовали метод в применении к алгоритму фотонных карт и в работе отмечается, что для различных применений могут потребоваться различные хэш-функции.

Авторы [95] использовали хэш-таблицы для реализации фотонных карт, причем в целях исключения коллизий во время поиска дополнительно производили сортировку пар (ключ, значение) по ключу. При этом поиск ближайших фотонов в точке (x,y,z) производился в ячейке hash(x,y,z) и в 27 соседних ячейках. В работе [95] отмечается, что для глобальной фотонной карты по сравнению с kd-деревом из работы [88] этот метод не дает ускорения сбора с ростом искомого числа ближайших фотонов (т.е. с ростом радиуса сбора). Это может быть объяснено потерей производительности на ветвлениях из-за наличия вложенного цикла, что необходимо для поиска в 27 соседних ячейках. При большом радиусе сбора и, как следствие, значительном времени итерирования фотонов в листьях, время обхода узлов дерева становится относительно малым. Однако, если в лист попадает только часть потоков группы warp (назовем эту часть активной), остальные потоки будут вынуждены ждать до тех пор, пока лист не будет пройден активной частью группы warp. И эта проблема одинаково наблюдается как для классических деревьев, так и для хэш-таблиц из работы [95].

В работе [96] предложен иной подход на основе пространственных хэш-таблиц. При возникновении коллизии авторы предлагают стохастически сохранять только 1 фотон на элемент таблицы. Данный подход замедляет трассировку фотонов (в 3-4 раза в соответствии со средним числом коллизий), что может быть нецелесообразно, т.к. при сложных условиях освещения трассировка фотонов будет занимать много больше времени, чем построение ускоряющих структур при помощи любого из известных методов. Позже, в работе [97] на основе метода из [94] был реализован алгоритм фотонных карт и было продемонстрировано, что решение на основе метода из работы [96] работает медленее, чем решение на основе метода из работы [94].

Среди практичных подходов следует упомянуть работы [98], [85], [99], авторы которых использовали растеризацию сфер с включенным альфа-смешиванием. Это позволяет реализовать сбор освещенности на GPU вообще без построения каких-либо укоряющих структур. Данный подход может быть использован только совместно с растеризацией, но неприменим в трассировке лучей.

Хотя подходы на основе окто-дереьвев используются при реализации фотонных карт достаточно часто, публикаций по даной тематике относительно немного. Причина этого заключается в том, что значительное число работ по построению окто-деревьев на GPU было опубликовано в смежных областях. Одна из работ, использующих окто-деревья на GPU для фотонных карт —

[100]. Причем, в работе [100] обсуждается эффективная реализация поиска k-ближайших фотонов над окто-деревьями. Сравнения, сделанные в данной работе, показывают примерно одинаковое качество при использовании сбора освещенности с фиксированным радиусом и использовании сбора с k-ближайших фотонов. По этой причине далее будем считать, что можно использовать фиксированный радиус сбора и рассмотрим некоторые работы, обсуждающие возможность построения окто-деревьев на GPU.

Построение окто-деревьев на GPU. Алгоритмы построения окто-деревьев на GPU используют факт взаимосвязанности окто-дерева и трехмерной Z-кривой, которая получается в результате вычисления Z-индекса (или кода Мортона) [101]. Зафиксируем некоторую глубину дерева 'k'. Пусть у нас имеется такой набор точек в трехмерном пространстве, что в каждый узел построенного в будущем окто-дерева попадет не более чем 1 точка. Если вычислить для точек коды Мортона и затем отсортировать точки по этим кодам, мы получим массив, эквивалентный окто-дереву, где в каждом листе хранится ровно по одному элементу (одной точке). Выполняя далее поиск в трехмерном пространстве в произвольной точке (x,y,z), мы вычисляем ее код Мортона, и применив бинарный поиск для полученного кода Мортона (ZInxed(x,y,z)), находим листовой узел окто-дерева, в который попала точка (x,y,z).
В работе [102] было впервые замечено соответствие между Z-кривой и окто-деревом и использовано свойство отбрасывания младших битов Z-индекса для получения идентификаторов старших узлов. Окто-дерево в [102] представлялось набором массивов L1..Lk, размером 8i элементов для каждого уровня глубиной i (т.е. несуществующие узлы хранились в памяти на регулярной сетке). Вопрос хранения каких-либо элементов данных в окто-дереве в работе [102] не рассматривается. Таким образом, построенное окто-дерево не может быть использовано непосредственно для пространственного поиска элементов, а может только отвечать на вопрос имеются ли искомые элементы в некотором объеме или нет.
В работе [103] окто-дерево было использовано для построения поверхности из набора точек при помощи алгоритма марширующих кубов. Алгоритм построения использовал серию из параллельных операций уплотнения и сортировки.

В отличие от работы [102], Zhou и др. сохраняют только существенные узлы в памяти, а также сохраняют списки точек на всех уровнях дерева. При этом, в [103] было замечено, что имея отсортированный массив точек по их Z-индексу в узле достаточно сохранять лишь 2 смещения, обозначающих начало и конец последовательности точек для данного узла в отсортированном массиве. Причем, при переходе от уровня k на уровень k-1 это свойство сохраняется и точки пересортировывать не нужно. Достаточно пересчитать указатели соответствующие началу и концу последовательности.

Если в рассмотренных ранее работах только отдельные уровни дерева строились параллельно, то в работе [92] был предложен метод, использующий массивный параллелизм GPU при построении всего дерева. Основная идея алгоритма заключается в том, чтобы пронумеровать все узлы дерева при помощи индексов таким образом, чтобы индекс родительского узла всегда являлся префиксом по отношению к индексам его дочерних узлов, если рассматривать индексы как битовые строки. Тогда, отсортированный массив индексов будет эквивалентен дереву. Следует отметить, что это свойство префиксов кодов Мортона было описано ранее в работе [102].

В работе [104] впервые высказана идея построения ускоряющей структуры на GPU при помощи сортировки индексов ссылок (а не индексов центров объектов). Размер ячейки в [104] был в 2 раза больше размера самого большого объекта, что гарантирует возникновения не более 8 ссылок на каждый объект. Каждая ссылка, таким образом, может быть записана в заранее отведенную для нее ячейку памяти а общий размер массива ссылок — 8*N где N — число объектов. Такое ограничение размера ячейки может негативно сказаться на скорости поиска при разных размерах объектов (кэш освещенности является одним из таких случаев), поскольку метод вынужденно выбирает размер ячейки в соответствии с самым большим объектом.

2.3.5. Приближенные методы на GPU

Существует достаточно большое число работ, вычисляющих глобальное освещение приближенно. Сюда можно отнести такие методы как Light Propagation Volumes (LPV) [105], трассировка конусов над воксельными моделями [106], метод ’Radiance Hits’ [107], методы на основе мгновенной излучательности

[108] и апроксимацию вторичного освещения при помощи предрассчитанных сферических гармоник [109]. Данные методы обладают хорошими временными характеристиками. Многие из них широко используются в современных компьютерных играх. Однако, в данной работе они не рассматриваются детально, поскольку их точность при этих временных характеристиках чрезвычайно низкая. Если же говорить о тех из них, которые могли бы обеспечить необходимую точность в пределе — [105] и [106], то при такой точность они работают не быстрее, чем обыкновенная Монте-Карло трассировка путей (а в случае метода [105] потребуется еще и невероятно большой объем памяти).

2.4. Программные решения на центральных процессорах

В настоящий момент число программных решений, позволяющих в той или иной мере решать проблему глобальной освещенности, достаточно большое. Будем называть такие программные решения рендер-системами. Среди наиболее известных рендер-систем следует отметить:

1. Mental Ray, Pixar RenderMan, Mantra (Houdini), Arnold, Indigo, Maxwell, VRay, Corona, KeyShot, Thea Render, Lightworks'Artisan, ZEANY, Lumion, Clarisse, Twinmotion, DirectViz, keyshot, modo, PhotoView 360, fryrender, SWAP, Turtle, Brazil, final render, Cheetah3D, kerkethea, EIAS9, Team Render (CINEMA 4D).

2. Specter, Inspirer, Inspirer 2, lumicept, Tonatiuh, SolTRACE, rtt, dialux, lighttools, speos, ASAP, tracepro, zemax, DiamCalc.

3. PBRT, Mitsuba, LuxRender, POV-Ray, Sunflow, Yafaray, Pixie, Aqsis, Radiance.

Разделение на 3 группы, представленное выше сделано следующим образом: к первой группе относятся системы, ориентированные на реалистичный синтез изображений; ко второй группе относятся системы, предназначенные для физически точного расчета освещения; к третьей группе относятся системы, ориентированные в большей степени на реалистичный синтез изображений, но используемые в основном в обучающих и исследовательских целях. Для систем из 3 группы важна доступность и расширяемость исходных кодов.

Такое большое разнообразие существующих программных решений обусловлено в первую очередь чрезвычайно широким спектром применения рендер-систем и, в некоторых случаях, высокой специализацией. Например система DiamCalc [110] ориентирована на реалистичную визуализацию драгоценных камней, а библиотека LIAC [111] на расчет взаимодействия света с изотропными и анизотропными кристаллами. В действительности, даже рендер-системы общего назначения имеют свою специализацию. Например, системы RenderMan [112], Mantra (Houdini) [113], Arnold [114] ориентированы в первую очередь на создание кино-эффектов. В таких системах важна поддержка огромного спектра эффектов и, что гораздо более важно, возможность тонкой настройки всех этапов визуализации. Причем, настройка подразумевает не просто установку предопределенных параметров, а, как правило, создание специализированного производственного процесса, направленного на визуализацию определенного материала в кино. В качестве примера можно привести генерацию и визализацию лесного массива, волос и меха в мультфильме CABBA [115], моделирование катастрофы поезда в фильме Метро [116], визуализация боевых действией в фильме Сталинград [117] и.т.д. При этом временные затраты на создание самих моделей и анимации настолько велики, что время расчета конечных кадров фильма, хотя и важно, не всегда имеет решающее значения.

Другие системы (Например Mental Ray [118], VRay [119], Corona [120]) в большей степени ориентированы на работу индивидуальных дизайнеров, работающих над архитектурными или рекламными проектами в небольших дизайнерских студиях. В таких системах чрезмерно широкие возможности и настройки не только не нужны, но и создают дополнительные трудности пользователям, которым необходимо уметь разбираться иногда не только в функциональных возможностях системы, но и в параметрах используемых алгоримов. Для этих систем важен относительно простой и быстрый процесс проектирования и визуализации. Причем скорость является критичным параметром, поскольку время визуализации может занимать значительную часть рабочего времени дизайнера. Более того, необходимо уметь демонстрировать потенциальному заказчику интерьер и/или экстерьер в различных условиях освещения, а также различные варианты самого интерьера. По этой причине достаточно востребованной функциональностью рендер-систем является 'переосвещение' (relighting) [121] - возможность менять условия освещения в реальном времени на основе предрассчитанной информации. Пользователи таких систем, как правило, профессиональные дизайнеры и художники, имеющие навыки в работе со сложными пакетами трехмерного моделирования.

Такие системы как KeyShot [122] еще в большей степени ориентированы на быстрое создание рекламных роликов и презентаций. Основная цель — создание рекламной презентации за несколька минут, имея на руках сырую модель, экспортированную из САПР пакета. В таких системах важна не только скорость визуализации, но и скорость создания самой модели. Не имея специальных навыков (как в случае предыдущей группы систем), пользователь должен иметь возможность за небольшое число действий достигать конечного результата.

Наконец, наличие самых разнообразных специализированных программных решений (например программа для создания вируальных моделей мебели — PRO100 [123]) и существование разнообразных САПР-систем, создает спрос на интегрированные в производственный процесс систем визуализации.

Вторая причина большого разнообразия рендер-систем — низкая производительность существующих систем и, как следствие, конкуренция систем, интегрированных в одни и те же пакеты трехмерного моделирования. Здесь следует отметить чрезвычайно популярный пакет 3D Studio Max [124] компании Autodesk, который используется во многих областях промышленности и бизнеса и служит полем для соперничества самых различных программных решений.

Что касается систем, ориентированных на высокоточное моделирование, такие системы, как правило, обладают в высокой степени спецификой решаемой задачи. Среди таких систем необходимо отметить программные и программно-аппаратные комплексы, разработанные в Институте Прикладной Математики имени М.В. Келдыша РАН (ИПМ РАН) — Specter, Inspirer, Inspirer 2, lumicept [125]. Данные комплексы прочно закрепились в современной промышленности и обеспечивают интеграцию с популярной САПР-системой CATIA.

Обозначенные выше системы позволяют проектировать сложные оптические системы и решают такие задачи как ([126], [127]):

1. Физически-аккуратный расчет освещенности и построение фотореалистичных изображений сцен, содержащих оптически сложные материалы при различных условиях искусственного и естественного [128] освещения [129].

2. Моделирование осветительных приборов со сложными свойствами распределения света, сложных оптических систем и устройств, включая системы, содержащие новейшие микроструктурные рассеивающие материалы, применяемые в современном проектировании оптических светопроводящих систем [130], [131].

3. Измерение оптических свойств плоских образцов материалов [132].

4. Моделирование сложных автомобильных красок, красящих покрытий с высокой концентрацией пигмента [133], тканей [134].

5. Расчет карт освещенности и последующая интерактивная визуализация в режиме навигации сцен [135].

6. Моделирования освещенности и синтез реалистичных изображений через Интернет [136].

7. Автоматизацию задания параметров сцены с целью упрощения процесса работы пользователей [127].

Как было уже не раз отмечено, одним из краеугольных камней реалистичной визуализации и расчета освещенности является производительность. Опыт разработки высоко производительных вычислительных систем в последние годы приводит к ряду важных результатов, среди которых необходимо обратить особое внимание на специализированные программно-аппаратные модели (такие как массивно-паралльные системы на основе GPU). На сегодняшний день графические процессоры обладают значительно большей производительностью (и, более того, значительно большим потенциалом) именно благодаря специальной программной-аппаратной модели. Эта модель отличается от классических программ для централных процессоров тем, что в ней, так называемый паралеллизм данных описывается явно на уровне алгоритмов.

Большинство рендер-систем, закрепившихся в промышленности на сегодняшний день, используют центральные процессоры. Эти системы были спроектировны в 80-ых и 90-ых годах прошлого века и в лучшем случае предполагали паралеллизм на основе кластерных систем. Причем, паралеллизм не закладывался в алгоритмы изначально, а реализовывался постфактум, в буквальном смысле 'как получится'. Показательным примером является рендер-система Radiance [80], использующая алгоритм кэширования освещенности и работы [81, 82], пытающиеся сделать кэш освещенности параллельным, не меняя оригинального алгоритма и архитектуру системы.
Однако, такой путь развития имеет свой предел и по этой причине в настоящее время активно развиваются новые системы (использующие как CPU так и GPU), в которых паралеллизм заложен изначально на уровене алгоритмов.

2.5. Программные решения на графических процессорах

Среди существующих рендер-систем реалистичной визуализации, использующих GPU, следует отметить такие проекты как Arion, Octane, VRay RT, LuxRender GPU, IRay, OptiX (API), OpenRL (API), Gelato (проект закрыт), RenderAnts, Render Bro (проект закрыт), Centi Leo, Cycles Render, RTRays, RedShift, Hydra.
Последняя в списке система (Hydra) была создана в результате данной диссертационной работы. Она будет детально описана в 4 главе.

2.5.1. Гибридные системы (CPU+GPU)

Гибридные рендер-системы используют графические процессоры лишь частично, для реализации отдельных алгоритмов на GPU (например для ускорения пересечение луча и сцены). К таким системам относятся Arion, LuxRender, IRay. Гибридная архитектура имеет преимущество в легкости поддержки исходного кода и расширяемости. Однако её производительность ограничена передачей данных между CPU и GPU и производительностью CPU кода [78] (рисунок 2.12). В [78] также сообщается, что с увеличением числа ядер, производительность гибриной системы не растет линейно и по прежнему продолжает оставаться ограниченной (рисунок 2.13). Это объясняется тем, что подсистема памяти (и без того являющаяся узким местом во многих CPU трассировщиках) дополнительно нагружается передачей данных 'батчей' (больших групп данных для лучей) [78].

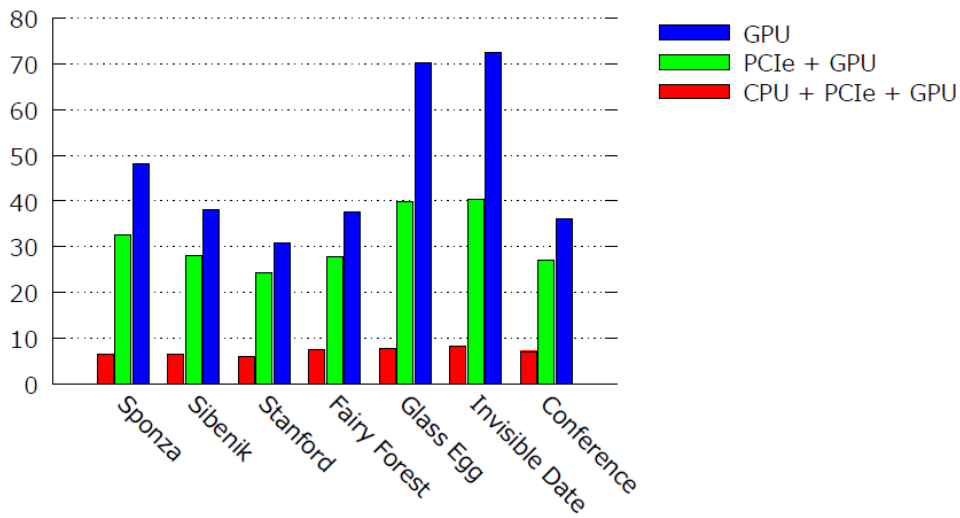


Рис. 2.12. Производительность гибридного трассировщика лучей ([78]). Столбцы GPU (синий) — производительность чистого GPU трассировщика. Столбцы PCIe+GPU(зеленый) — производительность GPU трассировщика, ограниченная пропускной способностью шины PCI. Столбцы PCIe+GPU+CPU(красный) — производительность гибридного решения.

2.5.2. Монте-Карло интеграторы на GPU

Как обсуждалось ранее, в настоящее время все известные промышленные системы (кроме системы RedShift [137], находящейся в разработке), использующие графические процессоры для расчета глобальной освещенности и реалистичного синтеза изображений основаны на различных вариациях метода Монте-Карло. Плюсом данного подхода является возможность получения точного решения в пределе при достаточно-большом числе Монте-Карло выборок. Однако, использование метода Монте-Карло в чистом виде негативно сказывается на времени расчета изображения. В результате, несмотря на то, что современные GPU при одинаковой стоимости способны до 10 раз опережать CPU на задаче трассировки лучей [17], время построения изображения в современных рендер-ситемах на CPU (при примерно одинаковом качестве)

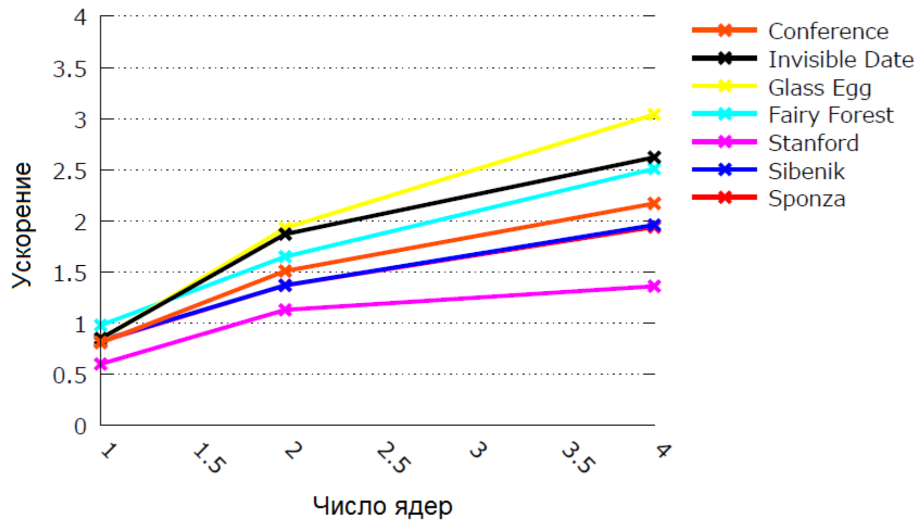


Рис. 2.13. Масштабируемость гибридной системы ([78]).

сравнимо или даже меньше времени построения изображения при помощи систем на основе GPU. Причина этого заключается в том, что некоторые методы, рассмотренные в 1 и 2 главах, позволяют при сравнимом качестве получаемого изображения значительно сократить объем производимых вычислений. За счет этого, CPU решения в ряде случаев продолжают выигрывать у GPU решений.

Среди рендер-систем, работающих полностью на GPU, следует отметить разработку Нижегородского Государственного Университета имени Лобачевского — систему RTRays [24], [138]. Помимо обычной Монте-Карло трассировки путей данная система реализует собственный оригинальный алгоритм расчета освещенности — усеченную двунаправленной трассировку путей [24].

Этот алгоритм позволяет рассчитывать значительно более широкий спектр эффектов нежели при помощи обыкновенной Монте-Карло трассировки. Из других положительных сторон следует отметить расширямость системы, возможность ее использования в обучающем процессе и быстрый алгоритм посроения BVH дерева, реализованный полностью на GPU.

Стоит отметить и другую отечественную разработку — систему Centi Leo [139, 140]. Как и в системе RTRays, в Centi Leo построение BVH дерева реализовано полностью на GPU. Однако, Centi Leo — единственная из существующих на сегодня систем интегрирования освещенности, способная эфективно работать с объёмами данных, которые не помещаются в память GPU.

2.5.3. Конструкторы

Такие системы как OptiX [141] и OpenRL [142] не являются законченными рендер-системами, а представляют из себя специализированные API, предназначенные для создания рендер-систем на их основе. Причем, OptiX предназначен для разработки систем на основе трассировки лучей для GPU производимых компанией Nvidia, а OpenRL имеет в настоящий момент специализированную аппаратную реализацию от компании Caustic Graphics [142]. Хотя специализированный чип Caustic Graphics имеет определенные преимущества, по соотношению цена/производительность в настоящий момент он проигрывает решению от Nvidia (OptiX). Из недостатков OptiX следует отметить серьезное падение производительности при реализации сложных алгоритмов. Это падение связано с тем, что в системе OptiX весь пользовательский код собирается компилятором в одно большое мега-ядро, которое работает по принципу конечного автомата. Такой подход позволяет достичь значительной гибкости, однако, он негативно влияет на производительность решения (в 2-3 раза медленнее чем подход на основе множества небольших ядер) [73].

2.6. Выводы ко второй главе и выбор направления исследования

Хотя при реализации трассировки лучей и Монте-Карло трассировки путей на GPU возникают определеные технические трудности, эта задача не встречает фундаментальных препятствий. Каждый путь/луч может обрабатываться независимо в отдельном потоке; причем, потоки не используют результаты вычислений друг друга, поэтому синхронизация и разделение данных между ними не требуется. Смещенные алгоритмы, с другой стороны, нетривиальны при параллельной реализации и требуют применения специальных подходов (например подход из работы [83]). Что касается эффективной реализации таких подходов на массивно-параллельных системах, в настоящий момент эта область еще не исследована в достаточной для промышленного применения мере. Существующие программные системы на графических процессорах не используют смещенные методы решения проблемы глобальной освещенности и по этой причине во многих случаях значительно проигрывают CPU системам, в которых такие методы реализованы. Например, такие новейшие GPU рендер-системы как Octane и IRay в лучшем случае обеспечивают паритет со старыми CPU системами со смещенным решением (VRay и Mental Ray), — и значительно проигрывают новейшей CPU рендер-системе Corona (приложение А). Поэтому, несмотря на наличие большого числа программных решений и серьезную базу исследований, проблема глобальной освещенности не перестает быть актуальной.

Работы последних 5 лет (2009-2014) предлагают новые эффективные методы решения проблемы глобальной освещенности — такие как стохастические прогрессивные фотонные карты, методы на основе кэша освещенности и фильтрации, методы на основе переноса света Метрополиса, метод соединения вершин и гибридные методы. Помимо новых методов, ключевыми посылками необходимости создания нового программного обеспечения именно сейчас является развитие массивно-параллельных систем (GPU) в последние 10 лет. Появившаяся в 2007 году технология CUDA способствовала реализации все более сложных алгоритмов на GPU, и создание массивно-параллельных алгоритмов решения проблемы глобальной освещенности является приоритетной темой современных исследований в области компьютерной графики.

По указанным выше причинам настоящая диссертационная работа направлена на разработку эффективных смещенных методов решения проблемы глобальной освещенности на GPU и создания соответствующего программного обеспечения.

Глава 3

Предложенные методы

3.1. Использумый подход в общем

Один из наиболее практичных подходов к решению проблемы глобальной освещенности заключается в том, чтобы разделить освещенность на несколько компонент, каждую из которых можно эффективно вычислять отдельно — определенным алгоритмом. Идея разделение освещенности на несколько компонент и вычисление отдельных компонент различными алгоритмами широко используется [33], [58], [59], [60]. По аналогии с работой [59] проведем разделение освещенности на 3 компоненты:

1. Прямая освещенность — $E(S|G|D)L$ и освещенность, вызванная только зеркальными или матовыми отражениями — $E(S|G)^+L$. Данная компонента эффективно вычисляется при помощи Монте-Карло трассировки путей.

2. Вторичная диффузная освещенность — $E(S|G)^*DD(S|G|D)^*L$. Данная компонента может быть эффективно вычислена при помощи кэширования освещенности в сочетании с финальным сбором [37]. Сюда также следует отнести более сложный случай $E(S|G)^+D(S|G)^+D(S|G)^*L$

3. Каустики — $E(S|G)^+D(S|G)^*L$. Данная компонента может быть эффективно вычислена при помощи метода стохастических прогрессивных фотонных карт [33].

В данном подходе отсутствует объемная составляющая V, ответственная за подповерхностное и объемное рассеяние света [143]. Известно, что эффект подповерхностного рассеивания может быть эффективно вычислено при помощи фотонных карт [144], а объемное рассеивание — при помощи стохастических прогрессивных фотонных карт [145]. В целях ограничения сложности в данной работе компонента V не рассматривается. Тем не менее, рассмотренный выше подход может быть обобщен на случай наличия объемных эффектов. Среди недостатков комбинированного подхода, использующего фотонные карты и трассировку путей совместно, следует отметить усложненную процедуру контроля ошибки по сравнению с чистым методом Монте-Карло. Метод контроля ошибки в этом случае требует оценки не только уровня шума но и смещения. Один из эффективных методов контроля ошибки описан в работе [146].

Параллельная реализация описанного выше подхода на многоядерных CPU встречает ряд проблем (связанных в основном с параллельной реализацией кэша освещенности и фотонных карт), описанных во второй главе. Реализация данной идеи на массивно-параллельных системах встречает ряд новых проблем, на исследование которых направлена настоящая работа.

3.2. Проблема неравномерного распределения работы

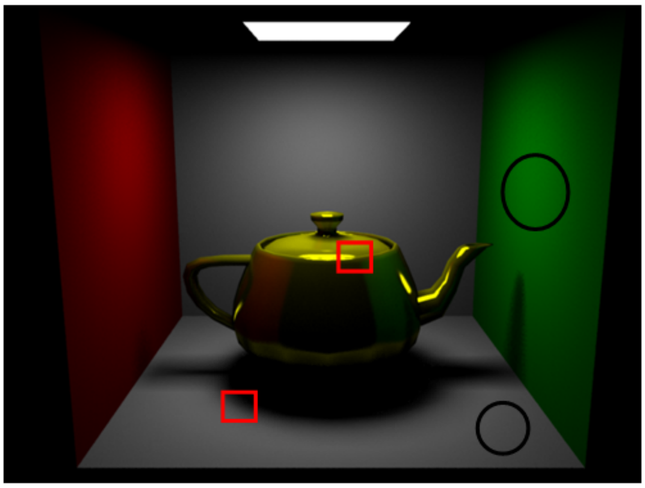


Рис. 3.1. Иллюстрация участков изображения различной сложности. Красные прямоугольники обозначают сложные для расчета участки, зеленые круги — относительно простые.

Неравномерное распределение работы по пикселам изображения — одна из первых проблем при реализации трассировки путей на GPU. В действительности эта проблема встречается и в обыкновенной реализации метода Монте-Карло, однако, при применении комбинированного подхода она становится более существенной. На рисунке 3.1 изображен пример сцены без вторичной диффузной освещённости. Черными кругами отмечены области, в которых, чтобы вычислить интеграл освещённости с достаточно высокой точностью достаточно всего несколько Монте-Карло выборок (для каждого пикела изображения). Красными квадратами выделены наиболее тяжёлые для расчета участки изображении (содержащие мягкие тени и отражения ярких источников света на матовых поверхностях). Для достижения того же уровня точности, в этих областях может потребоваться несколько сотен Монте-Карло выборок на пиксел.

При реализации трассировки путей на центральном процессоре это не создает дополнительных проблем, т.к. каждый пиксел изображения можно обрабатывать последовательно — до тех пор пока ошибка, оцениваемая через стандартное отклонение, не будет меньше желаемой величины.

При реализации на графических процессорах неравномерное распределение объема работы по изображению создает серьезные трудности, поскольку модель вычислений на GPU (и классическая реализация трассировки путей) предусматривает обработку всего изображения целиком фиксированное число раз. Было бы крайне неэффективно вычислять 1000 Монте-Карло выборок для каждого пиксела изображения, если всего лишь некоторые из них требуют 1000 выборок.

3.2.1. Предложенный подход

Для того чтобы сделать распределение работы адаптивным, экран разбивается на блоки размером 8x8 или 16x16 пикселей. Внутри каждого блока пиксели нумеруются при помощи кривой Мортона (рис. 3.2) для индексации всех данных, уникальных для каждого пиксела. Такая нумерация убирает разрывы адреса при переходе к следующей строчке (в отличие от простой двухмерной индексации), что позволяет использовать объединение запросов к памяти (coalescing) и сохранить пространственную близость лучей с близкими идентификаторами потоков. А это, в свою очередь, уменьшает число расхождений по ветвлениям в SIMD группах warp [74] и повышает эффективность.

1. В начале работы алгоритма все блоки добавляются в список активных блоков active_list.

2. Выделим на GPU память для локальных данных лучей, достаточную для того чтобы параллельно обрабатывать TMAX блоков.

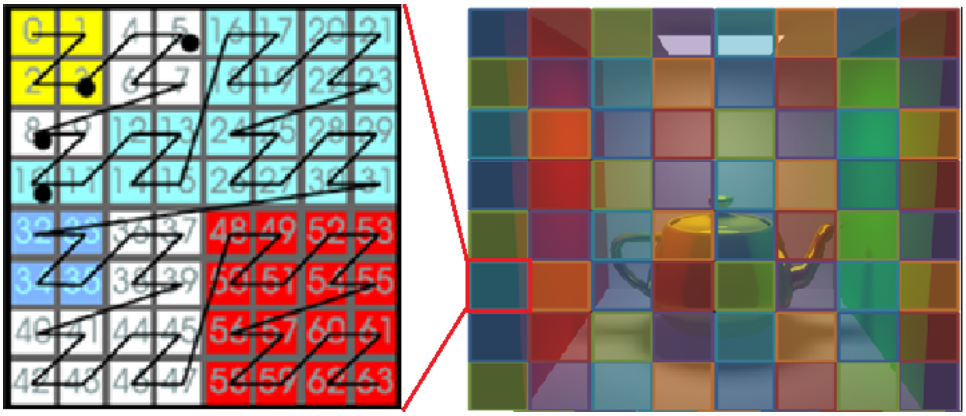


Рис. 3.2. Иллюстрация кривой мортона внутри блока (слева) и обработка изображения блоками (справа).

3. Извлечем из списка active_list ровно $TMAX$ блоков и обработаем их (рассчитаем некоторое небольшое число Монте-Карло выборок). Результат сохраним в соответствующие блокам места на изображении.

4. Для каждого блока оцениваем по некоторой метрике можно ли считать его завершённым. Если блок завершён, он удаляется из списка. Если нет – добавляется обратно в список active_list.

5. Если размер списка active_list меньше чем TMAX, удваиваем число путей на пиксел, трассируемых за одну итерацию.

6. Повторяем алгоритм начиная с шага 3 до тех пор пока список active_list не пуст.

Удваивание числа путей, трассируемых за 1 итерацию на пиксел позволяет создать достаточный объем работы для GPU даже при малом числе активных блоков. Таким образом, блоки, соответствующие наиболее сложным участкам изображения, будут оставаться активными наиболее длительное время, и все вычислительные ресурсы GPU в конечном счёте сосредоточатся на них (рис. 3.3).

При больших разрешениях (например 4000x4000) предложенный подход позволяет значительно экономить память, поскольку объем выделяемой памяти под локальные данные лучей не зависит от разрешения и определяется фиксированным числом $TMAX$. Само число $TMAX$ должно быть до-

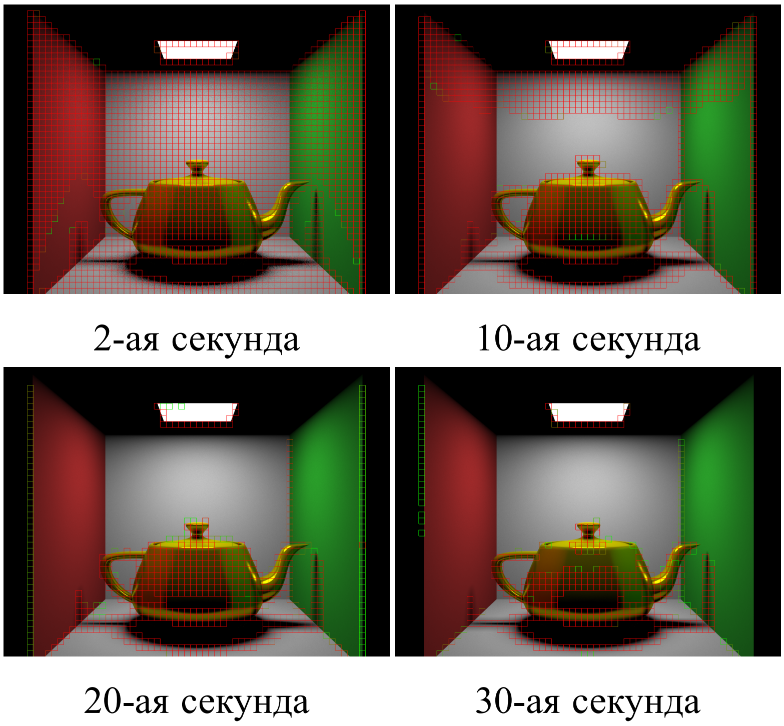


Рис. 3.3. Иллюстрация работы адаптивной трассировки путей в течение определенного времени. Красные квадраты обозначают блоки 16x16, находящиеся в процессе расчета. Зеленые — блоки, также находящиеся в процессе расчета, но готовые в скором времени завершится.

статочно большим для того, чтобы загрузить GPU работой при трассировке $TMAX * 16^2$ лучей.

```
type Tile is record
  index    : integer;
  error    : float;
  counter  : integer;
end record;
```

Listing 3.1. Структура данных блока

При этом сами списки реализованы на CPU, а массив таких структур размером $TMAX$ перемещается между центральным и графическим процессорами на каждом проходе алгоритма. Однако, так как элементов массива в 256 раз меньше, чем число потоков (блоки 16x16), такое копирование не является узким местом и занимает менее 0.5% от времени построения изображения. Поле 'index' содержит смещение для группы из 256 лучей в памяти GPU. Оно используется во время выборки нужных лучей и сохранения результата в соответвующее блоку место на изображении. Поле 'error' — некоторую оценку ошибки для всего блока, вычисляемую на GPU, но используемую на CPU для того чтобы определить, можно ли удалить блок из списка активных. Поле 'counter' обозначает число раз, которое был обработан блок. Оно используется для корректного учета вклада Монте-Карло выборок.

Ошибка error вычисляется на основании следующего критерия: для каждого пиксела изображения сохраняется сумма значений Монте-Карло выборок и сумма квадратов этих значений. Это позволяет оценивать для каждого пикселя блока дисперсию и стандартное отклонение.

3.2.2. Сравнение

Распределение работы (по блокам или даже попиксельное) для задачи Монте-Карло трассировки путей могло бы быть реализовано достаточно эффективно и просто при наличии в CUDA буфера трафарета ('stencil buffer'). Буфер трафарета позволяет использовать аппаратный механизм распределения работы на GPU в графических API (OpenGL, DirectX), когда некоторые его пикселы не активны. Тем не менее, следует отметить, что по сравнению с предложенным алгоритмом, распределение работы на основе буфера трафарета обладает следующими недостатками:

1. Такой алгоритм позволит отбрасывать пустые области, но не позволит сосредоточить ресурсы GPU на нескольких небольших, но сложных/шумных участках изображения.

2. Реализация при помощи буфера трафарета будет тратить значительно больше памяти, т.к. для локальных данных лучей в этом случае необходимо выделять буфер размером с изображение.

3. В технологиях вычислений общего назначения CUDA и OpenCL буфер трафарета не доступен.

При отсутствии буфера трафарета реализация попиксельного распределения возможна при помощи операции параллельного уплотнения. На рисунке 3.4 представлено сравнение предложенного алгоритма с разработанным в Нижегородском университете им. Лобачевского попиксельным распределением работы, реализованным в системе RTRays [24].

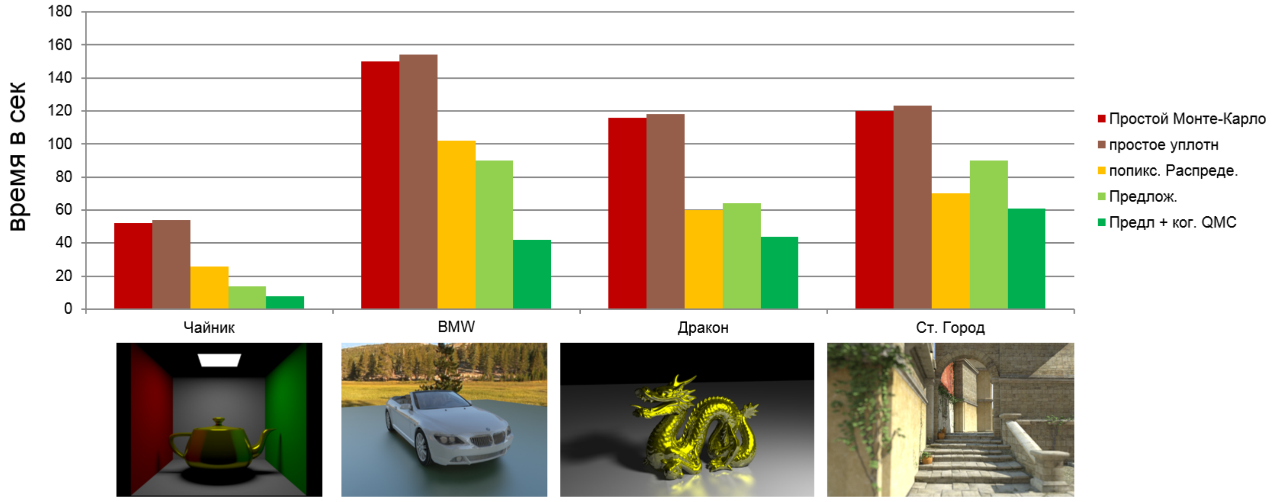


Рис. 3.4. Сравнение предложенного подхода (столбики 'предлож' и 'предлож + ког. QMC') с простой реализацией метода Монте-Карло ('простой Монте-Карло'), реализацией с применением уплотнения на каждом шаге трассировки ('простое уплотн') и алгоритмом, разработанным в Нижегородском университете им. Лобачевского [24] ('попикс. распределение'). Сравнение производилось на видеокарте GTX680. Приемлемая относительная ошибка была выставлена в 5%, а количество Монте-Карло выборок для каждого пикселя ограничено сверху числом 1024.

Колонка 'Предлож' (рис. 3.4) — время построение изображения при помощи простого метода Монте-Карло, а колонка 'Предлож + ког. QMC' — время построения изображения при помощи предложенного метода в сочетании с когерентным квази Монте-Карло [77]. Когерентный Квази Монте-Карло — это такой вид Монте-Карло трассировки путей, в котором для всех пикселей используется одно и то же состояние квази-случайного генератора. Это позволяет сохранять пространственную близость между лучами с близким идентификатором потоков, что в свою очередь увеличивает производительность из-за снижения числа расхождений групп warp на ветвлениях [77]. Следует отметить, что в отличие от попиксельного распределения работы, предложенный метод позволяет сохранять изначальную пространственную близость лучей и, таким образом, использовать преимущество когерентной трассировки лучей.

Выводы

Таким образом, предложен алгоритм распределения работы при реализации метода Монте-Карло на графических процессорах в применении к задаче вычисления глобальной освещенности и реалистичного синтеза изображений. Алгоритм позволяет эффективно (рис. 3.4) распределить вычислительные ресурсы графического процессора на множестве пикселей изображения с неравномерной сложностью расчета. В отличие от разработанного позднее попиксельного распределения работы [24] алгоритм позволяет сохранять пространственную близость групп лучей, что повышает эффективность использования ресурсов графического процессора за счет снижения числа расходящихся по разным веткам групп потоков (рис. 3.4). Следует отметить, что предложенный алгоритм не использует атомарные инструкции и операцию уплотнения на GPU, за счет чего может быть реализован на старом оборудовании без существенных затруднений, в отличие от попиксельного распределения.

3.3. Проблема параллельного кэширования освещенности

При реализации параллельного построения кэша освещенности, приходится решать ряд проблем. Среди основных трудностей следует выделить:

1. Проблему конечности построения кэша освещенности. Данная проблема возникает при использовании двухэтапного алгоритма создания кэша освещенности. На первом этапе необходимо построить кэш, а на втором только осуществлять выборку, но не добавлять в кэш новые точки. Как обсуждалось во 2-ой главе, такая схема позволет решить проблемы номер 3 и 5 (см. ниже), и повышает точность решения даже для однопоточной реализации.

2. Проблему эффективного расположения точек (записей). Иначе говоря, необходимо определить куда помещать записи кэша.

3. Проблему зависимости интерполяции (вернее радиусов валилности записей кэша и, как следствие, результатов интерполяции) от порядка добавления записей в кэш освещенности (рисунок 2.11) [37]. Данную проблему можно считать решенной при использовании двух-этапного алгоритма создания кэша освещенности.

4. Проблему вычисления радиусов валидности [64].

5. Проблему синхронизации данных между потоками. Данную проблему можно считать решенной при использовании двух-этапного алгоритма создания кэша освещенности.

Во 2 главе были рассмотрены существующие способы решения этих проблем и обсуждались их недостатки. Напомним, что подходы для многоядерных систем, рассмотреные во 2 главе, в основном нацелены на решение проблемы синхронизации данных на многоядерных и кластерных системах и используют подход ’ленивых вычислений’, автоматически решающий проблемы конечности и расположения записей кэша (как и оригинальный, однопоточный алгоритм). Как обсуждалось ранее, такой подход ограничен в масштабируемости, поскольку алгоритмы создания самого множества записей кэша не предполагает одновременного добавления многих точек в кэш. Проблема вычисления радиусов валидности и проблема зависимости радиусов от порядка добавления записей в кэш также обсуждалась в главе 2 и решалась на основе двух-этапного алгоритма — [64], [37] и [84], но вопрос эффективной одновременной генерации множества точек кэша освещенности остается в этих работах нерешенным, а проблема конечности построения кэша имеет описанные во 2 главе существенные недостатки.

При реализации кэширования освещенности на массивно-параллельных системах проблема эффективного одновременного добавления и генерации большого числа записей становится существенной. При этом необходимо уделить особое внимание проблеме конечности алгоритма, проблеме расположения записей и проблеме вычисления радиусов валидности.

3.3.1. Предложенный подход

Будем использовать двухэтапную схему построения кэша освещенности. Такой подход сразу позволяет решить проблемы номер 3 (зависимость результата от порядка добавления) и номер 5 (синхронизации данных), поскольку после окончания этапа построения новые записи в кэш не добавляются. Первый этап построения является многопроходным (15-20 проходов); каждый проход является двух-стадийным — стадия в экранном и мировом пространствах (алгоритм 2, рис. 3.5, 3.6). При этом, каждый проход алгоритма состоит из тройки операций — предсказание, вычисление, оценка. Оценка на текущем проходе используется для предсказания на следующем. На самом первом проходе использьется оценка на основе геометрических разрывов на изображении.

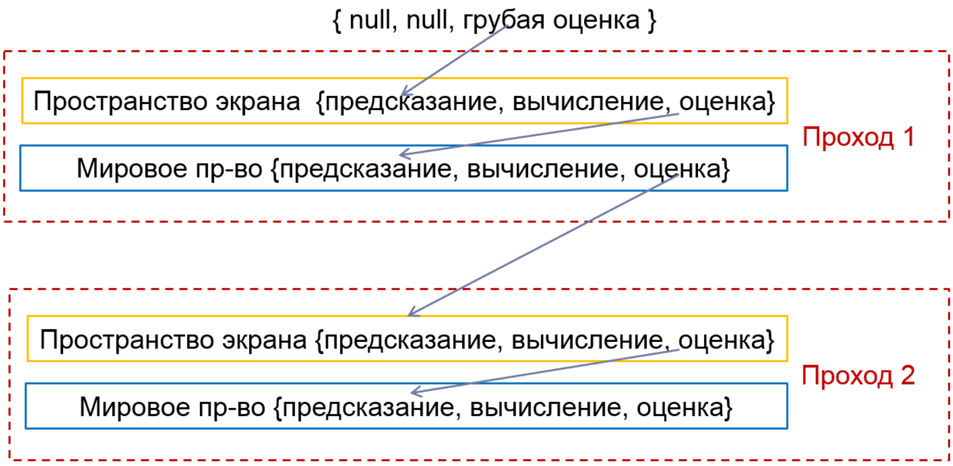


Рис. 3.5. Логическая схема работы предложенного алгоритма построения кэша освещенности. ’Грубая оценка’ вычисляется из изображения геометрических разрывов.

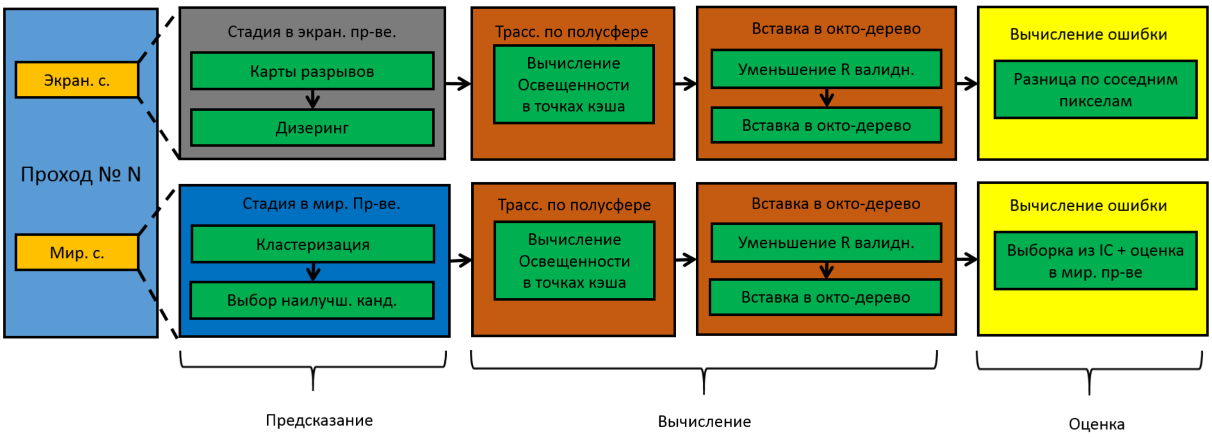


Рис. 3.6. Схема добавления новых записей в кэш освещенности на N-ом прохоже. ’Экран. с’ обозначает стадию в экранном пространстве. ’Мир. с’ — стадию в мировом пространстве.

Рассмотрим алгоритм 2. В самом начале его работы, о вторичном освещении на изображении ничего не известно. Однако, хорошо известен факт, что кэш освещенности располагает записи с низкой плотностью на гладких поверхностях и увеличивает плотность в областях с высокой геометрической детализацией и сильным изменением нормалей. Зная геометрию сцены, мы можем вычислить геометрические разрывы на изображении (функция $RenderGeomDiscontinutyImage$ вычисляет геометрическую разницу между соседними пикселами) и на основании этих разрывов сгенерировать первое приближение для кэша освещенности. То есть, в отличие от классического алгоритма кэширования освещенности мы идем с обратной стороны — сначала предсказываем, где записи потребуются, а затем добавляем их в кэш большими порциями в места, где они скорее всего нужны.

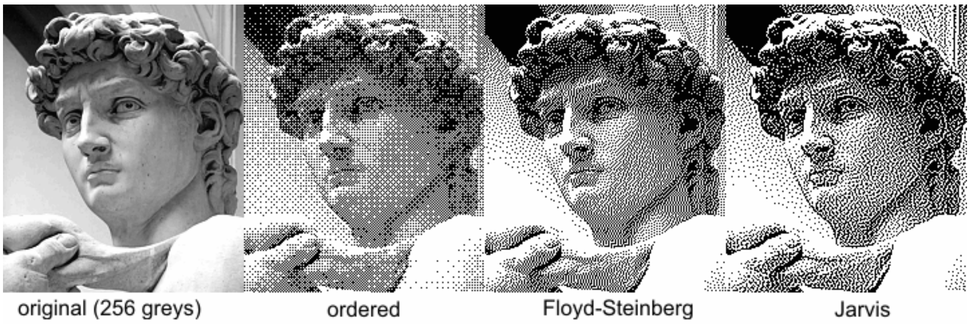


Рис. 3.7. Иллюстрация работы различных алгоритмов дизеринга, применяемых к чернобелому изображению с условием бинарной выходной палитры.

Механизм предсказания работает при помощи операции дизеринга изображения [147]. Дизеринг представляет собой операцию понижения палитры изображения с добавлением случайного шума таким образом, чтобы сохранить информацию о деталях, которая была бы утерена в случае простого квантования. В нашем случае входное изображение — монохромное изображение геометрических разрывов, а выходная палитра бинарная (рисунок 3.7). В результате операции дизеринга имеем изображение геометрических разрывов, представленное множеством черных и белых точек. Теперь, если белые точки будут представлять записи кэша освещенности — получим распределение записей кэша, соответствующее изначальному требованию — больше записей в областях с большей геометрической разницей и детализацией и меньше точек на гладких протяженных поверхностях. Далее, на последующих проходах алгоритма механимз предсказания работает по той же схеме, однако, в качестве входных данных уже используется не геометрическая разница, а разница между соседними пикселами по вторичной освещенности (функция $RenderIrradDiscontinutyImage$).

Выше была рассмотрена стадия, работающая в экранном пространстве. Однако, для достижения изображений высокого качества этой стадии недостаточно, поскольку в конечное изоражение вносят вклад участки сцены, не видимые камерой напрямую (отражения в зеркале или преломления через стекло). Для таких участков сцены вводится стадия, работающая в мировом пространстве.

Исходные параметры: scene - сцена

Результат: Множество записей кэша освещенности

// первый проход на основе геометрических разрывов

$geomImage \leftarrow RenderGeomDiscontinutyImage(scene)$;

$candidates \leftarrow Dithering(geomImage)$;

$ic.insert(candidates)$;

// многопроходный алгоритм построения кэша освещенности

$i \leftarrow 0$;

$N \leftarrow 20$;

while *candidates.size() > THRESHOLD and i < N* **do**

// стадия в экранном пространстве

$irradImage \leftarrow RenderIrradDiscontinutyImage(scene, ic)$;

$candidates \leftarrow Dithering(irradImage)$;

$ic.insert(candidates\ ic.records)$;

// стадия в мировом пространстве

$candidates \leftarrow SelectIfInterpErrorIsLarge()$;

$candidates \leftarrow SortWithZCurve(candidates)$;

$groups \leftarrow GroupRecords(candidates)$;

$candidates \leftarrow SeveralBestOfGroup(groups)$;

$ic.insert(candidates)$;

$i \leftarrow i + 1$;

end

Алгоритм 2: Предложенный алгоритм построения кэша освещенности

Стадия, работающая в мировом пространстве, более близка к классическому алгоритму, но имеет существенные отличия. Сначала в сцену трассируются пути для всех пикселей изображений и генерируются всевозможные запросы на вычисление точек кэша освещенности. При помощи параллельной операции добавления элементов в буфер (append) все такие запросы добавляются в буфер, расположенный в памяти GPU. Далее, запросы сортирутся на GPU по их Коду Мортона и копируются в память CPU, где в дальнейшем производится их кластеризация и выбор наилучших кандидатов ($SeveralBestOfGroup$). Кластеризация и выбор наилучших кандидатов нужны поскольку изначальное число запросов на вычисление освещенности избыточно.

Можно было бы снизить число запросов, просто уменьшив количество трассиуемых в сцену лучей, но это снизит качество решения, т.к. повышается вероятность пропустить некоторые участки сцены. Подход на основе кластеризации дает более точное решение, т.к. позволяет исследовать больше пространства и выбирать среди кандидатов тех, которые дают наилучше покрытие для группы точек.

Описанный алгоритм повторяется до тех пор, пока стадия в мировом пространстве генерирует достаточно большое число записей. Как только это число падает ниже определенного порога, или по достижению ограничения итераций, установленного пользователем алгоритм завершается.

Что касается реализации алгоритмов дизеринга и кластеризации, следует полагать, что здесь применимы различные подходы. В целях достижения высокой скорости, простоты реализации и задействования ресурсов GPU, были разработаны следующие алгоритмы.

Описание используемого алгоритма дизеринга

Для каждого пиксела изображения вычисляется разница с соседними пикселами — геометрическа разница если необходимо сгенерировать изображение геометрических разрывов и разница во вторичной освещенности (получаемой как результат интерполяции из существующего множетва точек кэша освещенности) если необходимо создать изобажение разницы вторичной освещенности. Для полученного изображения строится последовательность мип-уровней. Мип-уровни необходимы чтобы учесть детали в различном масштабе (рисунок 3.8).

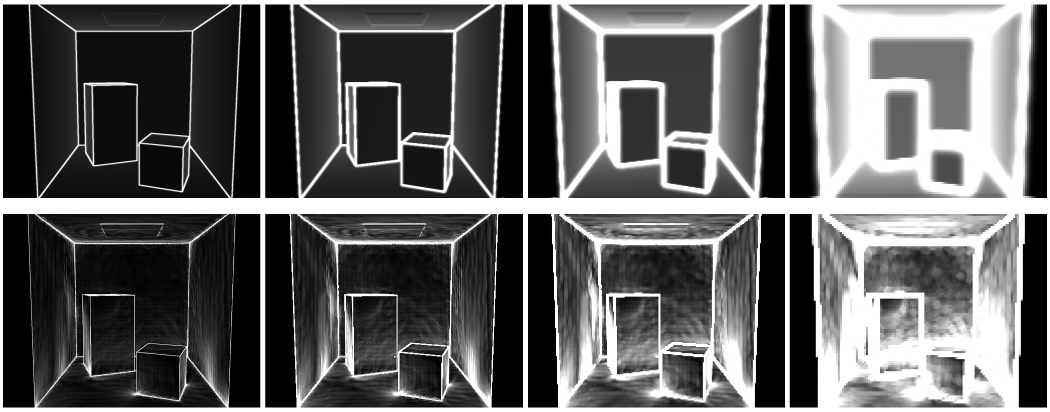


Рис. 3.8. Последовательность мип-уровней изображения геометрических разрывов (верхний ряд) и последовательность мип-уровней изображения разрывов освещенности (нижний ряд). Все изображения отмасштабированны в исходное разрешение. Дизеринг был применен ко всем мип-уровням изображений разрывов, чтобы учесть детали на изображении в различном масштабе.

Исходные параметры: Текстура discontinuty
Результат: Точки кэша в буфере queryBuffer

// in parallel for each screen pixel (x,y)
$d0 \leftarrow tex2DLod(discontinuty, x, y, 0)$;
$d1 \leftarrow tex2DLod(discontinuty, x, y, 1)$;
$d2 \leftarrow tex2DLod(discontinuty, x, y, 2)$;
...
$deterministicSelect \leftarrow F1(d0, d1, d2, ..., x, y)$;
$stochasticSelect \leftarrow F2(d0, d1, d2, ..., x, y)$;
if *deterministicSelect or stochasticSelect* **then**
    $queryBuffer.append(thisPixel)$;
end

Алгоритм 3: Дизеринг текстуры с мип-уровнями.

Далее для каждого пиксела, на основании алгоритма 3 принимается решение о том, генерировать для данного пиксела запись кэша освещенности или нет. Если решение принимается положительное, точка добавляется в буфер запросов при помощи операции параллельного добавления элементов в буффер ($queryBuffer.append(thisPixel)$). Следует полагать что функции F1 и F2 (алгоритм 3) допускают различную реализацию. В данной работе для F1 был использован детерминистический выбор на основании нескольких (разных для различных мип-уровней) фиксированных порогов для F1 и стохастический выбор на основании некоторого множителя для F2. Алгоритм дизеринга реализован полностью на GPU.

Описание используемого алгоритма кластеризации и выбора наилучших кандидатов

После создания всех запросов на вычисление освещенности такие запросы сортируются по их коду мортона (берется от позиции) и копируюся в память CPU. С этого момента в памяти CPU и GPU хранится идентичный массив точек-кандидатов. Далее на стороне CPU построение кластера начинается с нулевого элемента (самого меньшего по коду мортона). Кластер растет путем добавления в него следующих в массиве кандидатов до достижения некотрого порога относительного размера ограничивающего бокса кластера (относительно большого бокса, включащего всех кандидатов), после чего со следующего кандидата начинается образование нового кластера (рис. 3.9). Данный алгоритм обладает тем преимуществом, что не меняет порядок точек в массиве, а получающиеся кластеры представляют из себя пары чисел - начало и конец интервала в массиве кандидатов. Конечная его цель — группировка пространственно близко-расположенных кандидатов в единый кластер. Следует полагать, что здесь могут быть использованы различные альтернативные подходы.

После формирования кластеров, для каждого из них необходимо выбрать некоторый небольшой набор кандидатов, способных покрыть весь кластер. Для определения факта перекрытия использовались формулы интерполяции в кэше освещенности из работы [30]. Если в точке A можно интерполировать освещение с точки B, точка A считается перекрытой точкой B. Для каждого кластера случайным образом выбирается точка (кандидат), после чего из кластера целиком исключаются все точки (другие кандидаты), которые оказываются перекрыты данным кандидатом. Алгоритм повторяется до тех пор пока кластер не пуст.

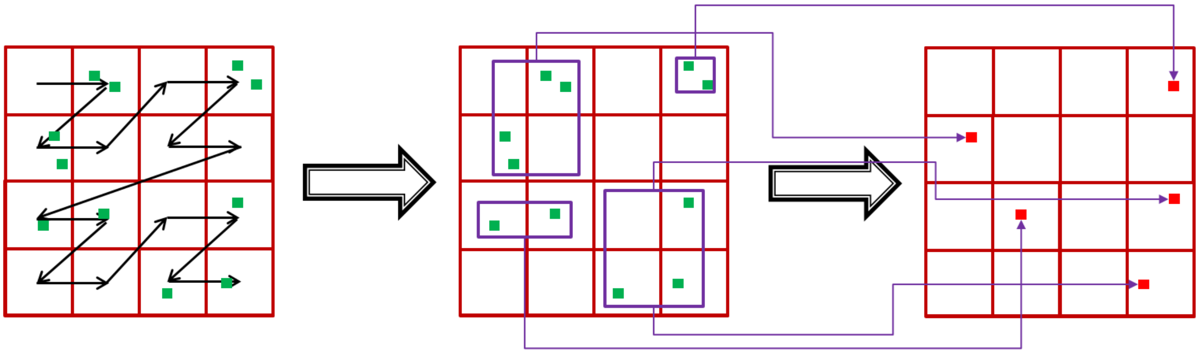


Рис. 3.9. Процесс кластеризации и выбора наилучшего кандидата. Маленькие зеленые квадратики обозначают кандидатов. Фиолетовые прямоугольники обозначают ограничивающие боксы кластеров. Маленькие красные квадратики — отобранные наилучшие кандидаты, которые станут в будущем записями кэша освещенности. Один кластер может генерировать более одного кандидата, в случае если весь кластер целиком не может быть перекрыт (следуя алгоритму интерполяции в кэше освещенности) одним кандидатом.

Решение проблемы радиусов валидности

При одновременном добавлении большого числа записей в кэш освещенности возникает новая проблема — чрезмерно большое перекрытие записей. Происходит это по той причине, что добавляемые параллельно записи (особенно характерно для стадии в пространстве экрана) ничего не знают друг о друге.

В однопоточной реализации, основанной на схеме ленивых вычислений такой проблемы не возникает, поскольку последовательное добавление в сочетании с рядом эвристик обеспечивает корректное вычисление радиусов валидности — т.е. новая запись при добавлении имеет информацию обо все существующих записях и фунции вторичной освещенности, аппроксимируемой на основе существующих записей кэша. При одновременном добавлении большого числа новых записей это не так и перекрытие записей получается слишком большим, что приводит к низкой скорости интерполяции и увеличению времени построения изображения на этапе финального рендеринга.

Алгоритм, разработанный для решения этой проблемы, работает следующим образом: Для каждой записи находятся ближайшие 4-7 соседей в различных направлениях (рис. 3.10). Последнее означает, что при поиске соседей

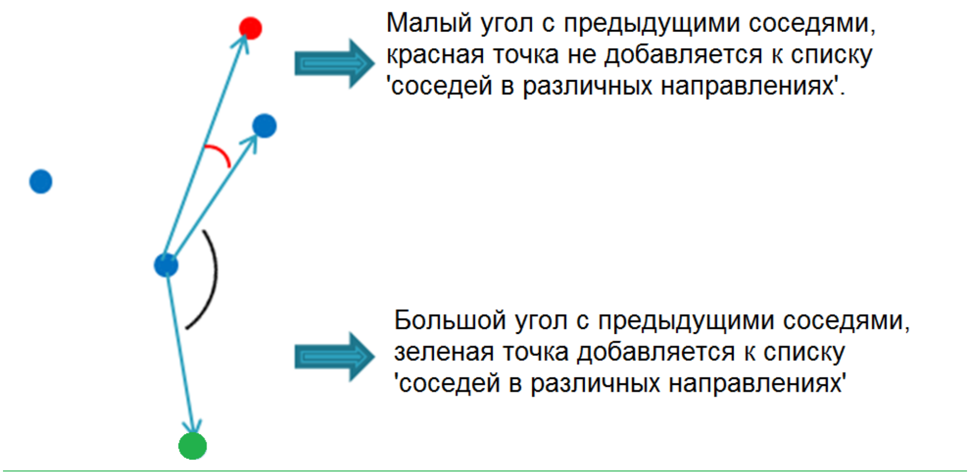


Рис. 3.10. Поиск ближайших 'соседей в различных направлениях'.

точки X новым соседом считаются только такие точки Y, для которых вектор XY имеет достаточно большой угол со всеми векторами XZ, где Z — найденный ранее найденный сосед. Реализованный алгоритм позволил ускорить финальную интерполяцию от 2 до 5 раз без ухудшения ее качества (рисунок 3.11).

Следует отметить, что для указанной проблемы хорошей альтернативой может являться алгоритм триангуляции Делоне [148]. Недостаток использования триангуляции Делоне заключается в необходимости перехода в тангенциальное пространство и работу в тангенциальной плоскости в 2 измерениях. Такой переход не всегда является тривиальным и может привести к появлению 'швов' для некоторых типов геометрии.

Вычисление освещенности в точках кэша

Освещенность в точках кэша вычисляется по прогрессвиной схеме — аналогично разработанному ранее алгоритму для изображения. Все записи помещаются в список активных записей. Каждая запись генирирует случайные лучи, число которых является элементом последовательности — 1024, 4096, 16384, 65536, и.т.д. Причем, такие лучи заранее объединены в когерентные (пространственно близкие) группы, что увеличивает скорость трассировки (рис. 3.12).

Рис. 3.11. Сравнение выборки из кэша освещенности для 1 миллиона операций интерполяции.

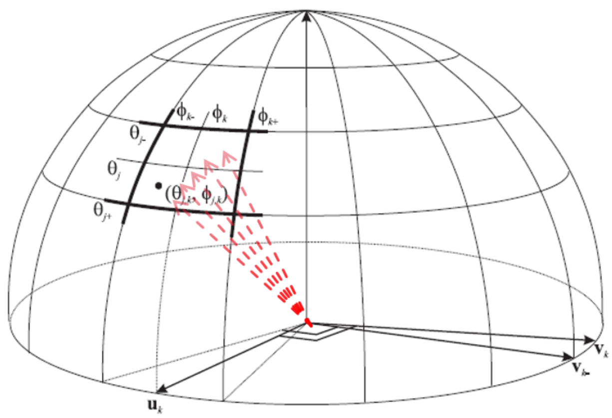


Рис. 3.12. Повышение когенетности групп лучей для вычисления освещенности в записях кэша освещенности реализавно путем разбиения полусферы на сектора.

В начале работы алгоритма все записи используют 1024 луча. Если для какой-либо записи оцениваемое стандартное отклонение получается больше установленного пользователем порога, на следующем шаге берется следующее из последовательности число (4096 лучей). На каждой итерации из списка активных записей извлекается такое число записей, лучи которых могут быть обработаны параллельно. Записи, стандартное отлконение которых находится ниже установленного порога, исключаются из списка активных.

Для вычисления значения освещенности используются как Монте-Карло трасировка путей, так и финальный сбор — при наличии глобальной фотонной карты. Финальный сбор повышает точность решения и позволяет практически всегда трассировать когерентные (пространственно близкие) группы лучей во время вычисления освещенности.

Интерполяция и структуры данных

Для реализации выборки из кэша использовалась структура, называемая октодеревом со множественными ссылками [37]. Вопрос построения такой структуры на GPU будет рассматриваться в следующем разделе.

3.3.2. Сравнение

Провести адекватное сравнение полученных результатов с аналогичными методами достаточно сложно, поскольку в современных работах по ускорению расчета вторичной освещенности редко указываются численные ошибки (а основной упор делается на визуально приемлемый результат). Без указания численных величин ошибок практически невозможно сделать вывод о точности полученного решения. А скорость визуализации напрямую связана с точностью. В некоторых известных работах ([105], [106], [107], [108], [109]) достигнута интерактивная скорость визуализации (несколько кадров в секунду), однако отличия от точного решения видны на глаз даже в низких разрешениях. С другой стороны, в системах высокоточного расчета освещения (и работах имеющих дело с методами Монте-Карло) реализованы в основном несмещенные методы, которые работают над одним кадром часами, но для них известно число выборок на пиксел и в некоторых случаях относительная погрешность интеграла. По этой причине в данной работе предоставлены сравнения предложенного решения на основе кэша освещенности на GPU и чистого метода Монте-Карло на GPU с указанием времени построения изображения и полученных численных значений ошибок (таблицы 3.1, 3.2; рисунок 3.14).

Сцена	Ош^2 Кэш (MSE)	Ош^2 М.К. (MSE)
Чайник	2.1	3.4
Sponza1	2.1	6.0
Sponza2	3.6	13.5
C-Sponza	2.5	6.3
Ренессанс	2.9	9.3
Класс	4.3	11.0

Таблица 3.1. Численные ошибки для сравнения на рисунке 3.14. Колонка 'Ош^2 Кэш' отображают разницу предложенного метода с эталоном. Колонка 'Ош^2 М.К. (MSE)' — разницу между эталоном и результатом, полученным при помощи метода Монте-Карло за то же самое время. Ошибки посчитаны для всего изображения (а не только для выделенных на рисунке 3.14 фрагментов).

Сцена	Прим	Луч/с	Зап.	П.Кэш	Ф.Р	Σ	Эталон	Уск	Ош^2
Чайник	26K	82M/8.1M	17K	9.0с	130с	139с	510с	3.6	2.3/2.1
Sponza1	66K	62M/5.4M	213K	157с	20с	177с	1554с	8.8	1.3/2.1
Sponza2	66K	64M/5.4M	98K	109с	11с	120с	1695с	14.1	1.6/3.6
C-Sponza	262K	30M/2.8M	442K	501с	40с	541с	3347с	6.2	2.9/2.5
Ренессанс	872K	64M/4.3M	306K	167с	10с	177с	1830с	10.3	1.7/2.9
Класс	1M	45M/4.1M	302K	285с	20с	305с	2001с	6.6	2.1/4.3

Таблица 3.2. Сравнение разработанного алгоритма с методом Монте-Карло на наборе тестовых сцен и сопутствующая статистика. Колонка 'Сцена' идентифицирует используемую в целях проведения сравнения сцену. Колонка 'Прим' — число примитивов в этой сцене. Колонка Луч/с отображает число лучей в секунду и число выборок в секунду, выполняемых в среднем трассировщиком лучей в формате: лучей/выборок. Колонка 'Зап.' показывает число созданных записей кэша освещенности. Колонка 'П.Кэш' — время построения кэша освещенности. Колонка 'Ф.Р' — время финального рендеринга. Колонка 'Σ' — суммарное время построение изображения при помощи предложенного метода. Колонка 'Эталон' — время построения изображения при помощи эталонного метода Монте-Карло с оговоренными ограничениями (5% ошибка и не более 1024 выборок на пиксел). Колонка 'Уск' — ускорение, достигаемое предложенным методом по сравнению с эталоном. Колонка 'Ош^2' — квадратичную разницу между предложенным методом и эталонов в широком динамическом диапазоне (hdr) и обыкновенном (png), посчитанную при помощи программы 'The Compressonator' в формате 'hdr/png'. Все замеры были произведены на GPU GTX560 в разрешении 1920x1200.

На рисунке 3.13 представлена увеличенная в 16 раз разница между изображениями, полученными разработанным методом и методом Монте-Карло с 5% точностью и предельным ограничением в 1024 выборки на пиксел. На рисунке 3.14 представлено увеличенное сравнение изображений, полученных разработанным методом и методом Монте-Карло за одно и то же время. В таблице 3.1 представлены численные значения ошибок, соответствущие рисунку 3.14. В таблице 3.2 статистика для тестовых сцен. Порядок тестовых сцен в таблице 3.2 тот же, что и для скриншотов на рисунке 3.13.

Выводы

Таким образом, предложена параллельная реализация алгоритма кэширования освещенности на графических процессорах, позволяющая снизить вре-

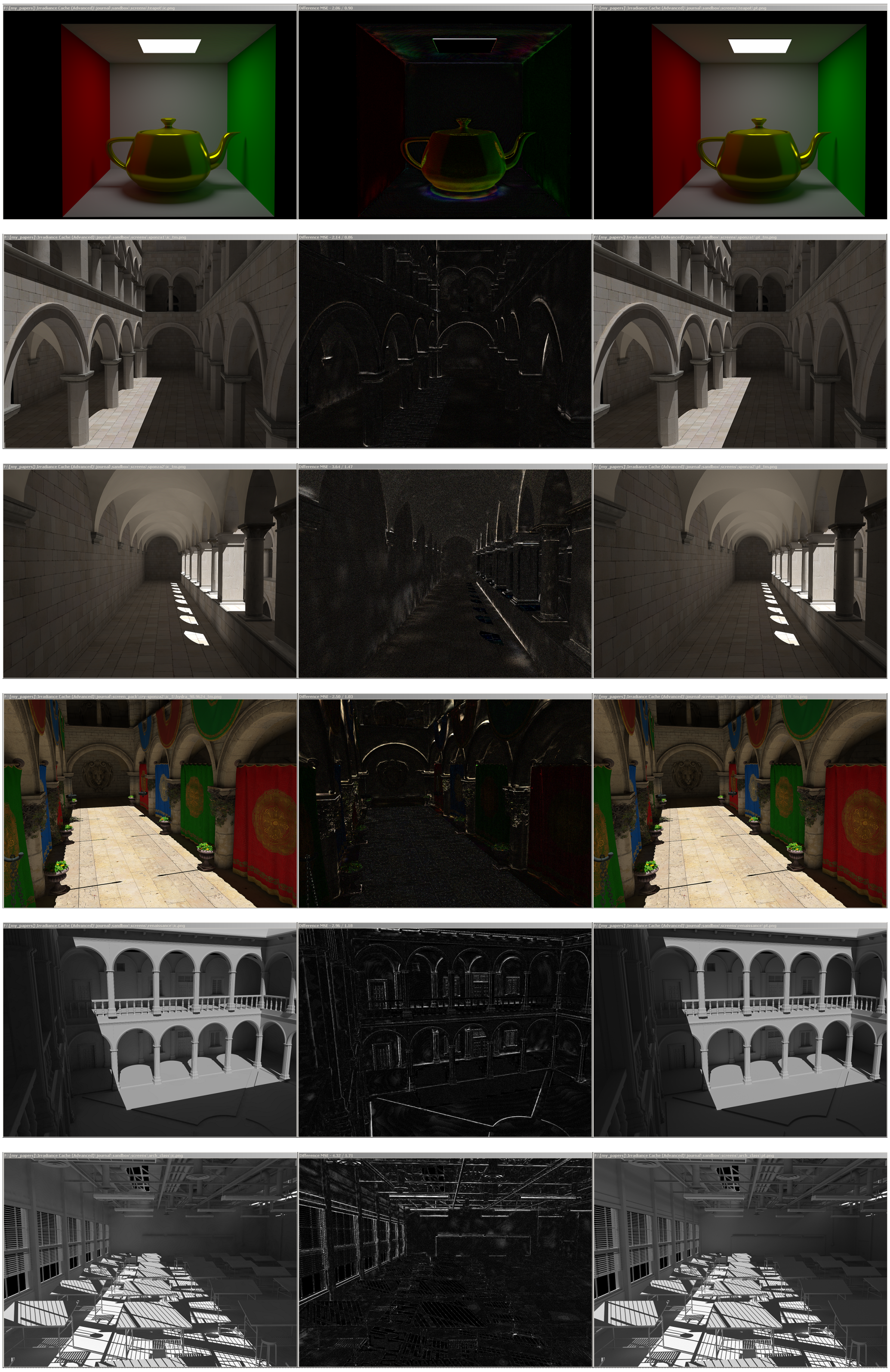


Рис. 3.13. Кэш освещенности (слева), метода Монте-Карло (справа), разница, увеличенная в 16 раз (посередине). Разница получена при помощи программы 'The Compressonator'.

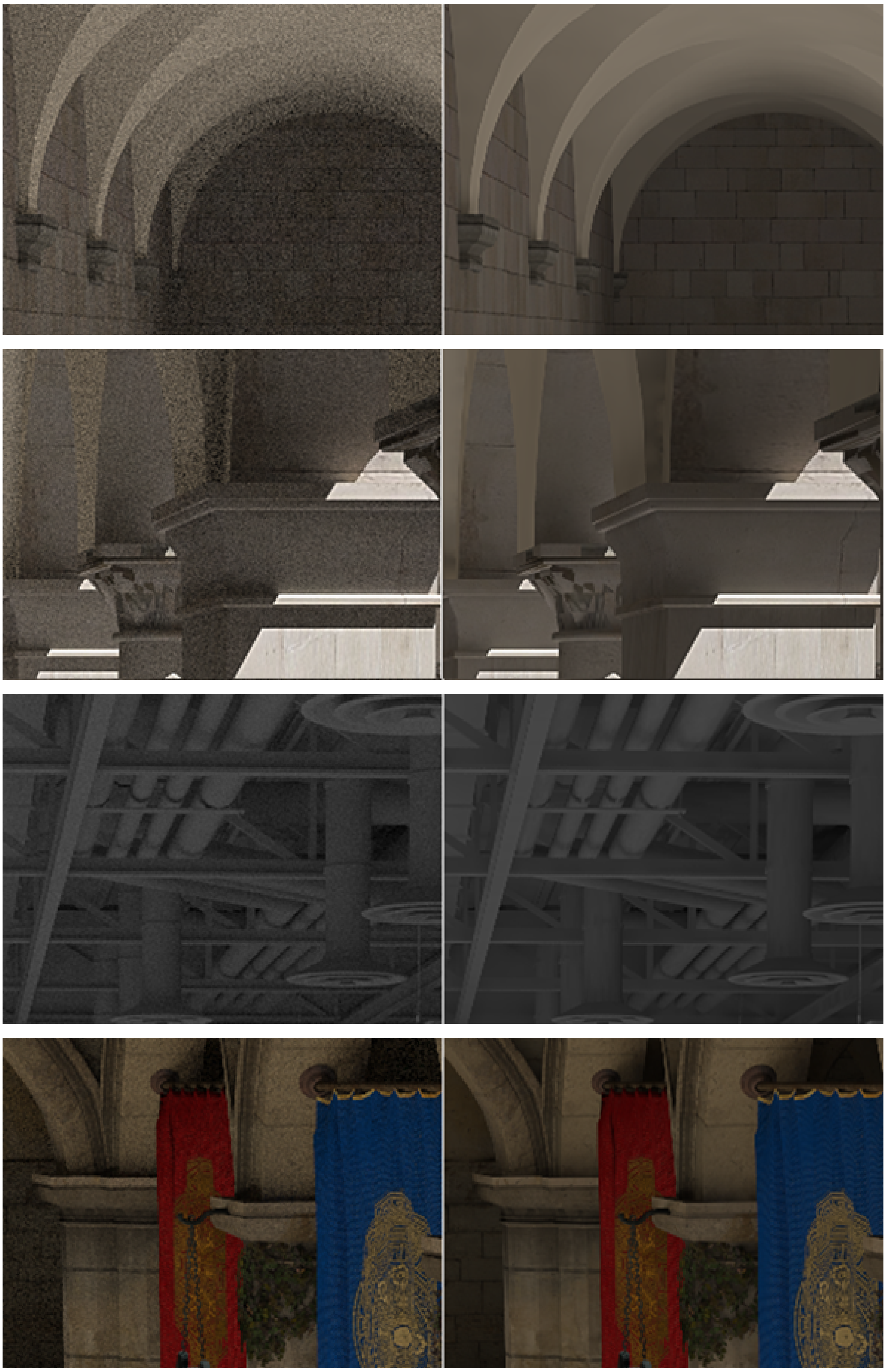


Рис. 3.14. Сравнение в увеличенном мастабе Монте-Карло трассировки путей (слева) и Кэша Освещенности (справа) для изображений, полученных за одинаковое время. Численные ошибки представлены в таблице 3.1.

мя построения изображения на порядок по сравнению с традиционным методом Монте-Карло при сравнимом качестве получаемого изображения (рис. 3.13). При этом, за то же самое время алгоритм позволяет получить в среднем в 2-3 раза более точное решение чем чистый метод Монте-Карло (рис. 3.14, таблица 3.1). Это соответствует временному выигрышу в 4-9 раз при фиксированной точности исходя из скорости сходимости метода Монте-Карло и подтверждает выигрыш во времени на порядок. Такой выигрыш коррелирует с ускорением при реализации кэша освещенности на CPU [149]. В отличие от существующих параллельных реализаций кэша освещенности или реализаций, частично использующих графические процессоры, в предложенном подходе впервые решена проблема одновременном добавлении в кэш большого числа недублирующих друг друга записей. За счет этого достигнута масштабируемость алгоритма и возможность интеграции в существующие системы расчета освещения на графических процессорах без существенного изменения архитектуры системы.

3.4. Проблема реализации фотонных карт и проблема построения окто-дерева со множественными ссылками на GPU

Как отмечалось во 2 главе, реализация фотонных карт на GPU является довольно популярной темой исследований. Связано это прежде всего с высокой сложностью задачи построения пространственной структуры на GPU — для реализации быстрого поиска ближайших фотонов (k ближайших, или всех фотонов в заданном радиусе). Основные недостатки существующих решений — низкая скорость сбора освещенности и трудоемкость реализации алгоритма построения структуры пространственного поиска на GPU. Особенно важно отметить, что скорость сбора становится критичной при расчете изображений в больших разрешениях. Немаловажным фактором является и сложность такого алгоритма, поскольку в промышленной системе вопрос поддержки стоит не на последнем месте. Следует отметить, что окто-дерево со множественными ссылками, используемое для реализации выборки из кэша освещенности, может быть использовано в процедуре сбора освещенности в алгоритме фотонных карт. Причем алгоритм сбора в этом случае получается нерекурсивным, а наиболее ресурсоемкое место заключено всего в одном цикле, не содержащем ветвлений. Эффективный алгоритм построения окто-дерева со множественными ссылками на GPU позволит использовать единую структуру данных для эффективной реализации как кэша освещенности так и фотонных карт и унифицировать реализацию обоих алгоритмов. Последнее упрощает сопровождение исходных кодов.

Предложенный подход

Предложенный метод позиционируется как метод построения ускоряющей структуры более эффективной по скорости сбора освещенности на GPU, чем существующие методы на основе деревьев, хэш таблиц и регулярных сеток и при этом не замедляющей трассировку фотонов (как метод из работы [96]). Чтобы избежать рекурсивного обхода и собрать ближайшие фотоны в заданном радиусе, было использовано окто-дерево с множественными ссылками. Принимая во внимании то, что радиус сбора фиксирован, можно рассматривать фотоны как сферы с радиусом, равным радиусу сбора освещенности. И строить окто-дерево, содержащее в качестве объектов не точки, а сферы. Этот известный факт используется при построении окто-деревьев с множественными ссылками для кэша освещенности [37].

3.4.1. Построение дерева

В начале работы алгоритм выбирает некоторый фиксированный уровень дерева k, для которого определяется размер его листьев. Уровень k выбирается так, чтобы размер листового узла был меньше или равен среднему размеру объектов. Важно отметить, что в предложенном алгоритме данный выбор не является строго обязательным, а всего лишь позволяет достичь хорошего баланса между скоростью построения дерева, скоростью поиска и объемом занимаемой памяти.

Далее все трехмерное пространство разбивается регулярной сеткой на воксели так, чтобы каждый воксел являлся листом будущего окто-дерева. Однако не все листья будут существовать в действительности, а только те, для которых будут иметься ссылки на объекты (т.е. непустые воксели). Используя тот факт, что в окто-дереве со множественными ссылками каждый лист ссылается на все объекты, которые имеют с ним пересечение, можно сразу сгенерировать массив всех ссылок, не строя пока что само дерево, а вычислив для каждого объекта множество ссылок на него (рис.1).

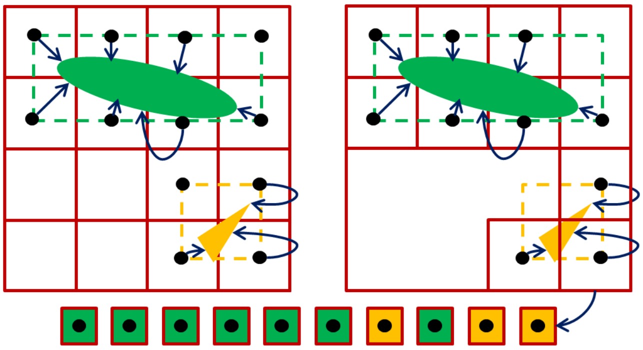


Рис. 3.15. Иллюстрация работы алгоритма. Зеленые элементы внизу представляют собой ссылки на овальный объект, оранжевые — на треугольный.

Оказывается, что если отсортировать массив всех ссылок (всех ссылок для всех объектов) по Z-индексу узлов, из которых они указывают на объект (рис. 3.15), мы уже получим практически готовое окто-дерево. Ниже представлен предлагаемый алгоритм (алгоритм 4):

Исходные параметры: objects

Результат: nodes, refs

$refs \leftarrow Append(Voxelize(objects))$; `// 1`

$refs.sort(compareByZindex)$; `// 2`

$nodes \leftarrow Append(Unique(refs))$; `// 3`

$nodes.sort(compareByiIndex)$; `// 4`

$return(nodes, refs)$;

Алгоритм 4: Алгоритм построения окто-дерева с множественными ссылками.

Шаг номер 1 – генерация множества всех ссылок, описанная ранее. Все ссылки добавляются в общий буфер при помощи операции параллельного добавления элементов в буфер. Такая операция может быть реализована через атомарные операции в CUDA или, что более эффективно, через функциональность графических API 'Transform Feedback' (OpenGL) и 'Stream Out' (DirectX). Использование параллельной операции добавления элементов в буфер позволяет предложенному алгоритму, в отличие от работы [104], строить дерево с произвольными размерами объектов и листьев и снимает ограничение работы [104] на минимальный размер листового узла.

Шаг номер 3 представляет из себя выделение множества узлов из множества ссылок и является прямым аналогом шага 4 'remove the duplicates using parallel compaction' из работы [92]. Операция уплотнения (compaction) была заменена на последовательность из операций (append, sort), чтобы не выделять дополнительный массив, размер которого равен количеству ссылок. Это позволяет экономить память. Ниже представлены типы данных, используемые алгоритмом (листинг 1).

```
type Reference is record
  zindex     : int32; -- код Мортона
  dataIndex  : int32; -- инд. объекта
end record;

type Node is record
  zindex    : int32; -- код Мортона
  refStart  : int32; -- инд.перв.ссылк.
  refEnd    : int32; -- инд.посл.ссылк.
end record;
```

Listing 3.2. Структуры данных

Реализация предложенного метода использует способ хранения данных SOA (Structure Of Arrays), благодаря чему имеем возможность освобождать неиспользуемые массивы после построения дерева. Так мы поступили с массивом для хранения кода Мортона ссылок и в результирующей структуре ссылка представлена, таким образом, одним 32 битовым числом. Следует дополнительно отметить, что предложенный подход позволяет хранить всё дерево в 2 массивах (nodes и refs), причем нелистовые узлы в памяти не хранятся совсем (nodes сохраняет только листовые узлы).

3.4.2. Сбор фотонов

Алгоритм сбора ближайших фотонов не использует стек и является прямым аналогом выборки из кэша освещенности на основе окто-деревьев с множественными ссылками [37]. Его основное преимущество заключается в том, что после спуска от корня к листу дерева искомые фотоны итерируются единственным циклом, перебирающим список всех ссылок и не содержащим ветвлений.

Сравнение

В зависимости от радиуса сбора (и как следствие различного числа попадающих в сферу фотонов) предложенный алгоритм (назовем его 'фотонные карты на основе окто-деревьев со множественными ссылками') ускоряет сбор освещенности в 2-5 раз по сравнению с классическим окто-деревом (рис. 3.16, таблица 3.5). В отличие от [95] предложенный алгоритм продолжает значительно выигрывать у дерева с одиночными ссылками при увеличении числа собираемых фотонов. Поскольку в работе [97] не было представлено анализа скорости сбора освещенности, проведем аналогию с работой [95] и сделаем заключение, что предложенный подход более эффективен на GPU из-за наличия только одного цикла итерирования фотонов в отличие от [95], [93] и [97], где необходим еще 1 вложенный цикл (а, как обсуждалось в разделе 2.3.4 'Фотонные карты на GPU', это приводит к потере производительности на ветвлениях).

Сравнение проводилось с обычным окто-деревом из работы [92]. Так как известно, что алгоритм [92] ограничен производительностью сортировки, за наилучшее время построения обычного окто-дерева по методу [92] имеем право взять время сортировки 1 миллиона точек. Полученное время построения (таблица 3.5) ожидаемо, так как в рассматриваемом нами типе окто-дерева на 1 фотон может сгенерироваться от 10 до 20 ссылок; поэтому время построения окто-дерева с множественными ссылками в 10-20 раз больше времени построения обычного окто-дерева на GPU на основе подхода [92]. Однако, можно считать, что данное замедление не имеет значимого эффекта на конечное время формирования изображения, поскольку дерево строится 1 раз после этапа трассировки фотонов. Причем трассировка фотонов, даже с учетом оптимизированной реализации трассировки лучей, сравнимой по производительности с [17], занимает в 5-10 раз больше времени, чем построение дерева (таблицы 3.3, 3.5, колонка ’т.ф.’).

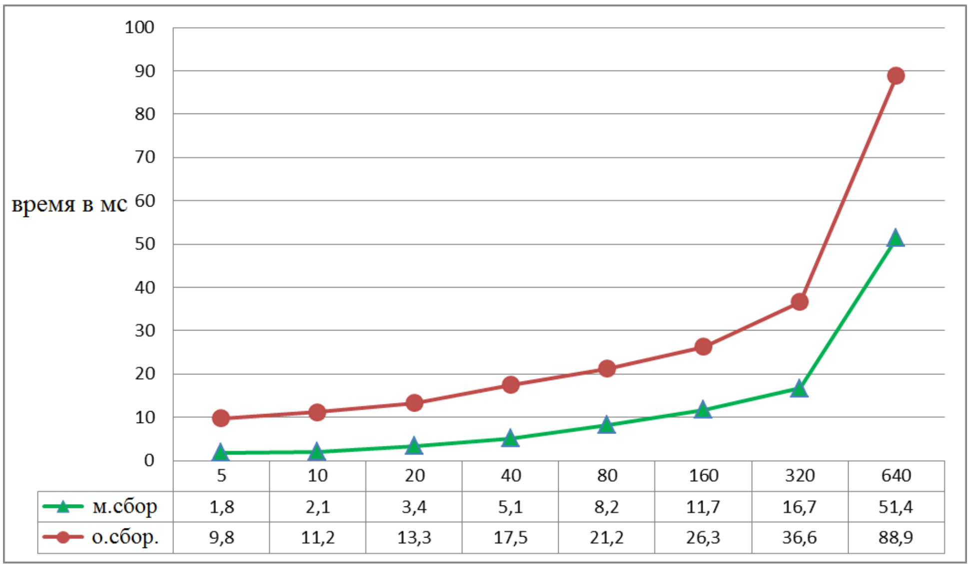


Рис. 3.16. Зависимость скорости сбора от плотности фотонов (среднем числе фотонов в узле — по оси x). Линия ’о.сбор’ показывает скорость сбора при использовании обыкновенного окто-дерева. Линия ’м.сбор’ — скорость сбора при использовании предложенного метода на основе окто-дерева со множественными ссылками. Замеры сделаны для 1 миллиона первичных лучей в разрешении 1024x1024 на GTX680.

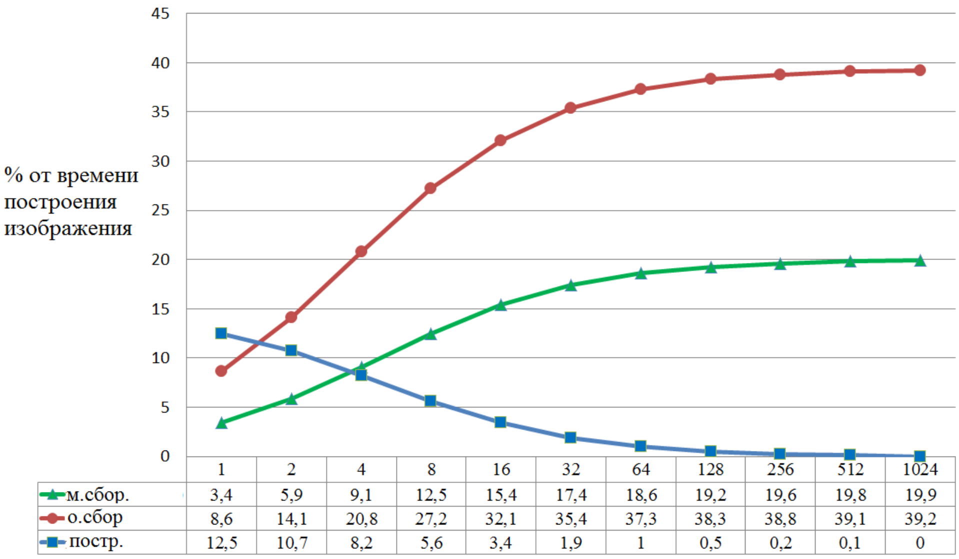


Рис. 3.17. Статистика потраченного времени в процентах. Для сцены cornell box с водой и каустической фотонной карты в зависимости от числа лучей сбора (по оси x в миллионах лучей) на 1 миллион фотонов. Линия 'о.сбор' отвечает за время сбора при использовании обыкновенного окто-дерева. Линия 'м.сбор' — время сбора при использовании предложенного метода на основе окто-дерева с множественными ссылками. Линия 'постр' — время построения окто-дерева со множественными ссылками.

номер шага	имя шага	глоб.	kауст.
0	сорт. фотонов	7.6%	15%
1	добавл. ссылок	21%	11%
2	сорт. ссылок	27%	21%
3	добавление узлов	20%	5.2%
4	сортировка узлов	4.8%	12%
-	остальное*	20%	35%

Таблица 3.3. Статистика потраченного времени для различных этапов построения окто-дерева для обыкновенной (или глобальной) фотонной карты (глоб.) и каустической (кауст.). Глобальная фотонная карта, как правило, имеет значительно больший радиус сбора. Для улучшения работы кэша GPU фотоны дополнительно сортировались перед построением дерева по их Z-индексу (шаг 0, сорт. фотонов). Такая сортировка в среднем дает от 15% до 20% прироста скорости на этапе сбора. Колонка 'остальное*' включает время, потраченное на выделение памяти, накладные расходы на вызовы функций CUDA, копирование констант на GPU и время, потраченное на вычисление Z-индексов фотонов.

сцена	алгоритм	трас. фотон.	построение	трас. лучей.	сбор	сумм.
sponza	IC+FG	0.6%	0.2%	72%	27%	25c
class	FG	0.5%	0.2%	76%	23%	62c
museum	IC+FG	0.5%	0.1%	83%	16%	94c
cornel1	SPPM	50%	5%	28%	17%	4.0c
cornel2	SPPM	59%	11%	22%	8%	1.7c
ring	SPPM	82%	6.6%	8.4%	3%	3.3c
water	SPPM	78%	6.3%	6.7%	9.0%	4.0c

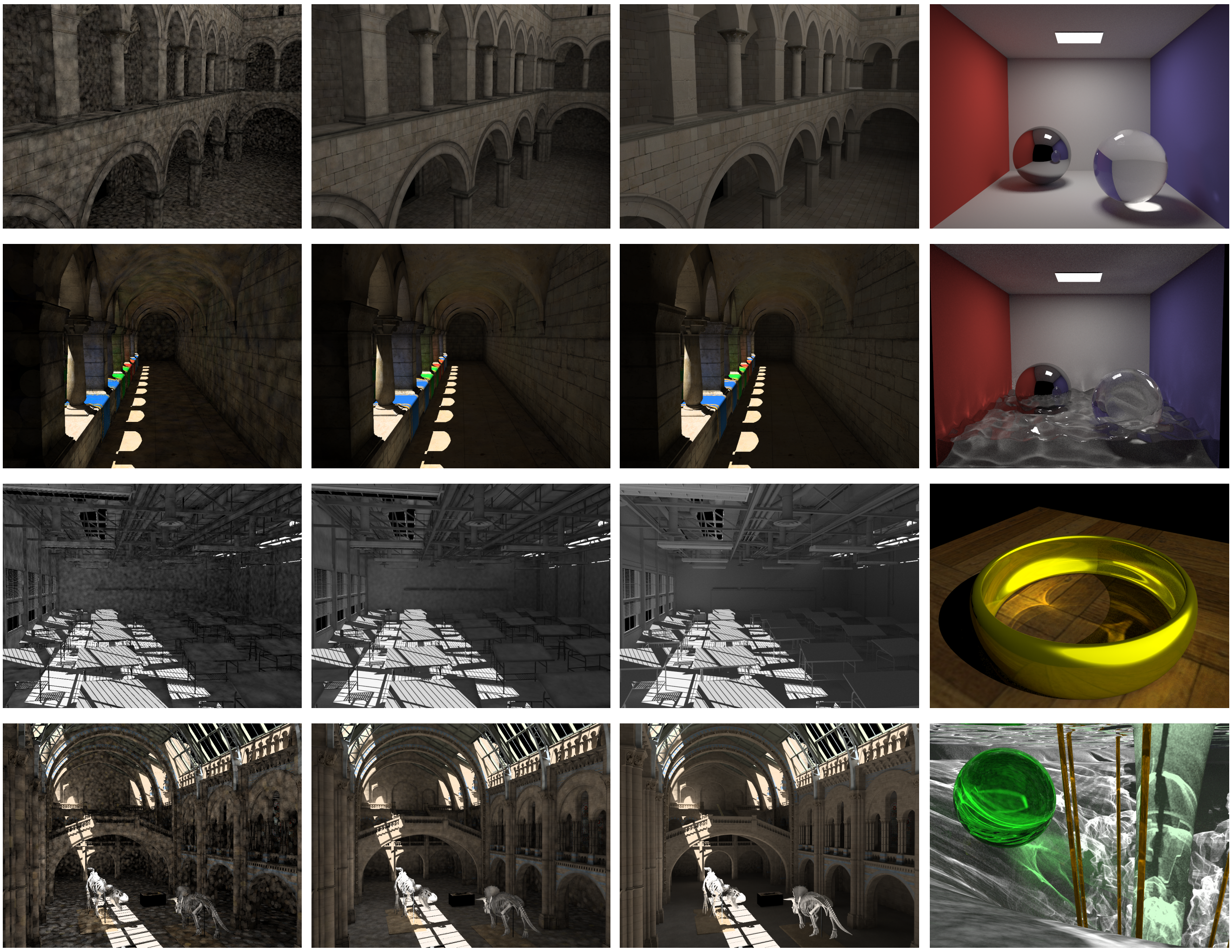
Таблица 3.4. Статистика потраченного времени для различных этапов построения при синтезе изображения. Колонка 'алгоритм' идентифицирует использованный метод построения изображения. FG обозначает использование финального сбора в чистом виде. FG+IC — использование кэша освещенности, где финальный сбор задействовался при вычислении освещенности в точках кэша. SPPM — стохастические прогрессивные фотонные карты. Колонка 'трас. фотон.' показывает время, затраченное на трассировку фотонов. Колонка 'построение' отвечает за время построения дерева. Колонка 'трас. лучей.' показывает время трассировки лучей сбора. Колонка 'сбор' — время, затраченное на сбор освещенности. Колонка 'сумм' — суммарное время рендеринга (для SPPM за 10 итераций). Для сцен с каустиками алгоритм SPPM был использован только для вычисления самих каустик. На сценах cornel1 и cornel2 перетрассировку фотонов выполнялась с пониженной частотой. Это позволяет распределять вычислительные ресурсы на остальные компоненты освещенности (не каустические).

Оценка применимости предложенного подхода

При использовании гибридного алгоритма построения изображения (на основе фотонных карт и трассировки путей) время построения изображения может быть оценено по следующей формуле:

$$T = N * (P + B) + M * (G + R)$$

- N — суммарное число обработанных фотонов.
- P — условное время трассировки одного фотона.
- B — условное время построения дерева в перессчете на 1 фотон.
- M — число лучей сбора.



1M фотонов | 10M фотонов | Финальный Сбор | 10M кауст. фотонов

Рис. 3.18. Сцены и изображения, соответствующие таблице 3.5.

сцена	треуг.	ссылок	лист	пам.	т.ф.	т.ф.(о)	п.(mr)	п.(sr)	с.(mr)	с.(sr)
sponza	66K	10M	418K	115MB	155мс	22мс	46мс	3.0мс	3.1мс	14мс
cry-sponza	262K	7.7M	88K	88MB	410мс	41мс	40мс	3.1мс	3.6мс	16мс
class	653K	9.9M	1.1M	110MB	160мс	23мс	48мс	3.1мс	7.7мс	30мс
museum	1.4M	11.2M	327K	128MB	470мс	26мс	52мс	3.2мс	4.4мс	16мс
cornel1	12	2.3M	74K	26MB	200мс	20мс	20мс	3.0мс	3.0мс	6.1мс
cornel2	19K	2.3M	99K	26MB	100мс	37мс	18мс	3.0мс	3.2мс	7.1мс
ring	18K	2.0M	57K	23MB	174мс	27мс	24мс	3.0мс	11мс	29мс
water	98K	2.0M	15K	22MB	310мс	35мс	25мс	3.1мс	36мс	78мс

Таблица 3.5. статистика для 1 миллиона фотонов и 1 миллиона путей сбора (с учетом переотражений). Колонка 'треуг.' показывает число треугольнико в сцене. Колонка 'ссылок' — число созданных ссылок на 1 миллион фотонов. Колонка 'лист' — число полученных при этом листьев. Колонка 'пам.' — максимальный объем памяти, требуемый алгоритмом для построения дерева (с учетом дополнительной памяти выделяемой thrust::sort). Колонка 'т.ф.' время трассировки фотонов до момента накопления в буфере 1 миллиона. Колонка 'т.ф.(о)' показывает теоретически-оптимальное время трассировки фотонов, как если бы все выпущенные из источника света фотоны сохранялись в фотонной карте. Это время вычислялось путем деления 'т.ф.' на отоношение числа выпущенных фотонов к числу акумулированных (1 миллион). Колонка 'п.(mr)' — время построения окто-дерева с множественными ссылками. Колонка 'п.(sr)' — время построения окто-дерева с одиночными ссылками. Колонка 'с.(mr)' — время сбора для 1 миллиона путей при использовании окто-дерева со множественными ссылками. Колонка 'с.(sr)' — время сбора для 1 миллиона путей при использовании окто-дерева с одиночными ссылками.

- G — условное время единичной операции сбора освещенности.
- R — условное время трассировки 1 луча сбора.

Графики на рисунке 3.17 отображают зависимость влияния компоненты G на время построения изображения при использовании предложенного метода и окто-дерева с одиночными ссылками. Предложенный алгоритм увеличивает компоненту B, но снижает компоненту G. Поэтому его использование оправдано если $B * N < G * M$. Таким образом, в низких разрешениях и/или при небольшом числе лучей сбора (относительно числа трассируемых фотонов) предложенный подход может замедлить процесс построения изображения в среднем не более чем на 5-10% (рисунок 3.17, начало графика; таблица 3.3, строчки с SPPM). В больших разрешениях и при увеличении числа лучей сбора (большое количество отражающих поверхностей или финальный сбор) метод уменьшает время построения изображения в среднем в 1.5 - 2 раза (рисунок 3.17, конец графика; таблица 3.3, первые 3 строчки).

Выводы

Таким образом, разработан метод построения ускоряющей структуры на графических процессорах, позволяющий от 2 до 5 раз ускорить сбор освещенности в алгоритме фотонных карт по сравнению с существующими методами. Разработанный метод впервые позволил строить окто-деревья со множественными ссылками над множеством объектов ненулевого размера полностью параллельно на графических процессорах. В отличие от других способов построения структуры пространственного поиска для фотонных карт, предложенный метод строит такую структуру, которая позволяет выполнить процесс сбора освещенности в заданной точке со всех фотонов в единственном цикле сбора, не содержащем ветвлений.

Глава 4

Программное решение

4.1. Структура программного обеспечения

4.1.1. Архитектура и организация CPU кода

Построенный на основе описанных в предыдущих главах алгоритмических решениях программный комплекс Hydra Renderer работает под управлением операционных систем семейства Windows и состоит из нескольких частей:

1. Основная программа (или сервер), непосредственно выполняющая расчет освещения — hydra.exe.

2. Ядро системы (в котором реализованы алгоримы) в виде DLL библиотеки (RTE.dll или RTE64.dll), загружаемое в адресное пространство hydra.exe.

3. Внешний графический интерфейс 'hydra_gui.exe' для демонстрации возможности работы программы без 3ds max.

4. HydraRender.dlr — плагин-рендер в виде DLL библиотеки, загружаемый в адресное пространство 3D Studio Max.

5. HydraMaterial.dlt — плагин-редактор материалов в виде DLL библиотеки, загружаемый в адресное пространство 3D Studio Max.

6. Прокси-сервер — hydra_server.exe.

Управление программой может вестись в 2 режимах. В первом режиме пользователь работает внутри 3D Studio Max и управляет программой посредством плагина-рендера (рисунок 4.2). После инициирования процесса расчета освещения (кнопка 'Render') плагин отсылает основной программе (будем называть её сервером) данные в виде 'Пакета-запроса' (рисунок 4.1). Такой пакет содержит геометрию в специальном формате (geom.vsgf), информацию о материалах и источниках в виде xml файла (profile.xml), файлы текстур и иные необходимые ресурсы в архиве (textures.zip), и xml файл с настройками алгоритмов (settings.xml). Во время расчета освещения сервер отсылает плагину с некоторой периодичностью полученное изображение, а также информацию о прогрессе расчета освещения посредством сообщений.

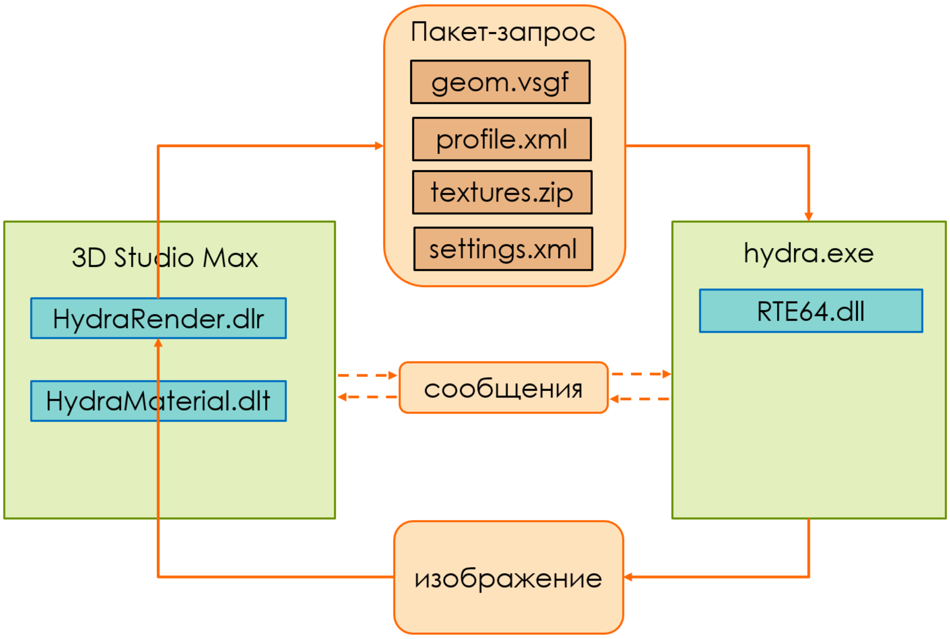


Рис. 4.1. Схема взаимодействия отдельных составляющих программного комплекса ’Hydra Renderer’ при работе в 3D Studio Max 2014.

Между плагином и сервером существует 2 механизма общения — через разделяемую память и по сети. В первом случае плагин и сервер находятся на одной машине и описанное выше взаимодействие осуществляется через файлы в разделяемой и внешней (дисковой) памяти. При этом плагин управляет процессом-сервером (hydra.exe). Например, при отмене расчета плагин завершает процесс-сервер. Во втором случае взаимодействие осуществляется по протоколу TCP/IP с прокси-сервером (hydra_server.exe), который должен быть запущен на машине, где планируется проводить расчет. При получении пакета-запроса прокси сервер запускает расчетные процессы hydra.exe и выступает только в роли посредника между плагином и сервером. Таким образом, прокси-сервер имитирует для расчетных процессов схему, показанную на рисунке 4.1, избавляя основную расчетную программу от лишних деталей, связанных с сетью. Во втором режиме (при отсутствии 3D Studio Max на машине пользователя) управление осуществляется посредством внешнего графического интерфейса, а сцена загружается из файла в формате COLLADA (файлы с расширением ’.dae’).

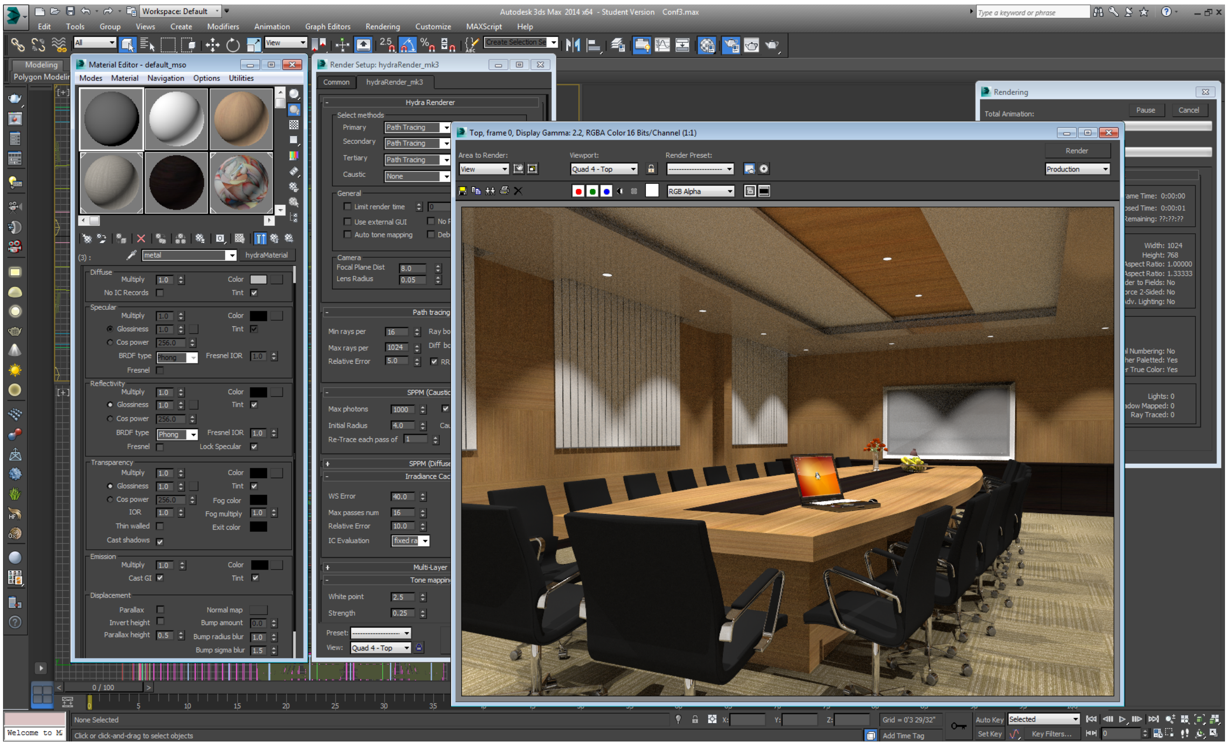


Рис. 4.2. Скриншот работы Hydra Renderer в 3D Studio Max.

4.1.2. Архитектура и организация GPU кода

По аналогии с работой [24] для реализации системы был использован конвейер трассировки лучей (рис. 4.3). По центру рисунка 4.3 расположены строго обязательные данные и ядра (присутствующие в конвейере всегда), справа — опциональные данные и ядра, появляющиеся в конвейере при задействовании кэша освещенности или фотонных карт. Работа конвейера осуществляется следующим образом:

1. Конвейер начинает свою работу с генерации лучей из камеры (’Генерация лучей (кам)’) или из точек-записей кэша освещенности, где необходимо вычислить освещенность (’Генерация лучей (кэш)’). Если производится трассировка фотонов, на этой стадии работают различные ядра для различных источников света, генерирующие соответствующее источнику света распределение фотонов.

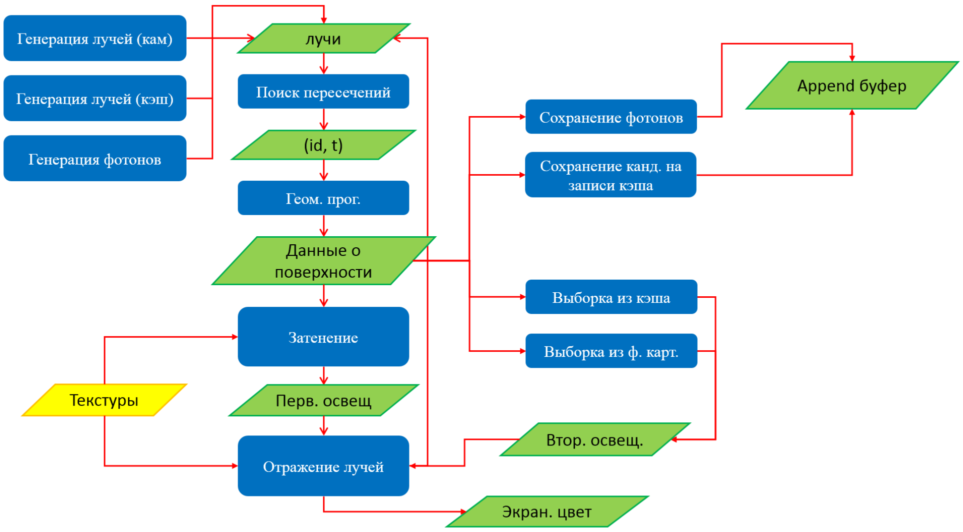


Рис. 4.3. Конвейер выполнения ядер на GPU. Синие прямоугольники со сглаженными углами отображают ядра. Зеленые ромбы — данные.

2. Сгенерированные лучи передаются далее в ядро, занимающееся поиском пересечений ('Поиск пересечений'). Это ядро производит буфер из 2 элементов — идентификатора ближайшего примитива, который был пересечен (id) и расстояния от начала луча до точки пересечения (t).

3. Полученный буфер передается в ядро 'Геом. Прог', вычисляющее все необходимые на следующих этапах данные о пересечении луча и поверхности (такие как позиция, нормаль, текстурные координаты, индекс материала и.т.д.).

4. В ядре 'Геом. Прог' также реализован алгоритм 'Parallax Occlusion Mapping' [150], имитирующий микро-рельеф на поверхностях при наличие карт нормалей и смещений.

5. Эти данные используются далее различными ядрами. Ядро 'Затенение' выполняет генерацию теневых лучей (с отбраковкой источников света, заведомо не вносящих вклад в освещение конкретной точки), производит трассировку теневых лучей и вычисляет прямое освещение. В результате ядро 'Затенение' сохраняет первичное освещение в буфере 'Перв. освещ.', который затем используется для вычисления результрующего вклада в изображение в ядре 'Отражение лучей'.

6. Ядро 'Выборка из кэша' реализует выборку из кэша освещенности при

его использовании, а ядро ’Выборка из ф.карты’ реализует сбор освещенности из фотонной карты. Оба ядра сохраняют свой результат в буфере ’Втор. освещ’, который затем также используется в ядре ’Отражение лучей’. Такая схема позволяет сократить количество ядер, осуществляющих доступ к текстурам материалов, что упрощает реализацию ядер ’Выборка из кэша’ и ’Выборка из ф.карты’ и делает их боле эффективными за счет экономии регистров GPU.

7. Ядра ’Затенение’ и ’Отражение лучей’ осуществляют доступ к текстурам материалов, реализуют многократную выборку по значимости и полностью отвечают за реализацию свойств материалов.

8. Ядра ’Сохранение фотонов’ и ’Сохранение канд. на записи кэша’ реализуют параллельную операцию добавления в буфер соответствующих ядру элементов при помощи атомарных операций при трассировке фотонов и поиске новых кандидатов для записей кэша в мировом пространстве.

4.2. Функциональные возможности

Разработанное программное обеспечение поддерживает следующую основную функциональность 3D Studio Max по отношению к визуализации:

1. Встроенные стандартные материалы (Standard) [151].

2. Собственные материалы рендера (hydraMaterial), позволяющие добиться высокой реалистичности при моделировании металлов и стекол.

3. Все типы стандартных источников.

4. Частичная поддержка фотометрических источников.

   а. Форма источника в виде точки, сферы, прямоугольника и диска.

   б. Стандартые распределение интенсивности – равномерное, диффузное и прожекторное.

    в. Источник типа 'Небесный Портал' (mr Sky Portal), значительно повышащий эффективность расчета дневного освещения в помещениях с окнами.

5. Поддержка стандартных камер и вывод изображения в окно рендера 3D Studio Max.

6. Все типы текстур включая Bump, NormalBump, Opacity [151] и процедурные текстуры любой сложности.

Среди основных функциональных возможностей расчетного ядра системы следует выделить:

1. Способность работать с большими объемами текстур (объемы, не помещающиеся в память GPU).

2. Возможность гибкого расчета освещения при помощи комбинаций 3 алгоритмов (кэш освещенности, фотонные карты и Монте-Карло трассировка путей).

3. Эффективный расчет прямого освещения, включая блики от площадных источников, — за счет реализации многократной выборки по значимости [22].

4. Реализацию карт смещения (Displacement) при помощи алгоритмов Normal Mapping и Parallax Occlusion Mapping.

5. Возможность моделирования эффекта глубины резкости (DOF).

6. Реализацию операции тонирования изображения (Tone Mapping) [152].

4.2.1. Описание основных режимов расчета освещения

В разработанной системе реализована гибкая схема задания алгоритмов для вычисления отдельных компонент освещения (рисунок 4.4). Разрешается настраивать следующие компоненты:

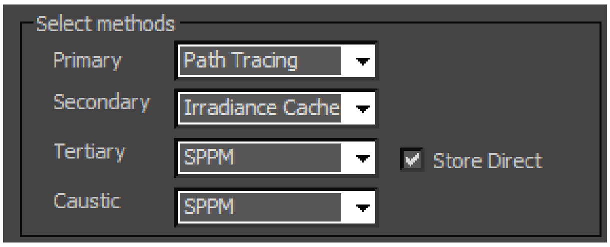


Рис. 4.4. Управление способом расчета освещения в системе Hydra. При данных настройках первичное освещение, преломление и отражение вычисляются при помощи метода Монте-Карло. Вторичное при помощи кэша освещенности и финального сбора. Каустическое освещение вычисляется при помощи алгоритма стохастических прогрессивных фотонных карт.

- Primary — первичное освещение. Рекомендуемое значение - 'Path Tracing' соответствует монте-карло трассировки путей. Следует отметить также что Монте-Карло будет всегда использоваться для вычисления отражений и преломлений, даже если значение селектора 'Primary Solver' не равно 'Path Tracing'.

- Secondary — вторичное диффузное освещение. Возможные значения - Path Tracing, Irradiance Cache, SPPM.

- Tertiary — третичное диффузное освещение. Возможные значения - Path Tracing, Irradiance Cache, SPPM.

- Caustic — Метод вычисления каустиков. Возможные значения — SPPM или Монте-Карло.

Опция 'Store Direct Light' дополнительно позволяет контролировать точность вычисления третичного освещения. Снятие этой опции увеличивает точность и фактически откладывает апроксимацию (из фотонной карты, карты светимости или кэша освещенности на 3-ем переотражении) еще на 1 переотражение:

- Если опция отмечена: в точках попадания вторичных лучей всё освещение считается при помощи сбора освещенности из глобальной фотонной карты, карты светимости или кэша освещенности.

- Если опция не отмечена: в точках попадания вторичных лучей первичное освещение рассчитывается при помощи трассировки лучей, а при помощи сбора освещенности вычисляется только вторичное освещение. Этот режим рекомендуется использовать в комбинации с кэшем освещенности для вторичного освещения, поскольку он значительно снижает число необходимых фотонов, которые во время рассчета кэша освещенности не могут обновляться.

Адаптивная трассировка путей, описанная в 1 части 3 главы, используется всегда и позволяет эффективно вычислять оставшиеся компоненты освещенности (первичное освещение, отражения и преломления) при использовании других методов для вычисления вторичного, третичного освещения и каустиков. Кэш освещенности, описанный во 2 части 3 главы, может быть использован для вычисления вторичного или третичного освещения (в последнем случае такой метод имеет известное название — рекурсивный кэш освещенности [37]). Стохастические прогрессивные фотонные карты (3 часть главы 3) используются при вычислении первичного, вторичного, третичного или каустического освещения при выборе 'SPPM'.

4.3. Практическое применение

Разработанная система была применена при демонстрации основного продукта компании 'Рей Трейсинг Системс' (рисунки 4.5, 4.6), при проектировании небоскреба (рисунки 4.9, 4.10, 4.11, 4.12) в системе 'Компас-3D' компании 'Аскон' (модели небоскреба были экспортированы в 3ds Max), при проектировании интерьера 'South Cafe' в г. Ялта (рисунки 4.7, 4.8)).

Рис. 4.5. Демонстрация возможностей рендер-системы Hydra Renderer компанией 'Рей Трейсинг Системс'. 1024x600, GTX760, 60 минут (Финальный Сбор); 2.8M треугольников.

Рис. 4.6. Демонстрация возможностей рендер-системы Hydra Renderer компанией 'Рей Трейсинг Системс'. 1024x608, GTX760, 60 минут (Финальный Сбор); 3.1M треугольников.

Рис. 4.7. Моделирование освещения интерьера South Cafe. 8.2M треугольников. 1024x768. GTX560. Время расчета 40 минут для всех изображений. Данная сцена характеризуется сложными условиями освещения. Светильники у стен были смоделированы таким образом, что всё фактическое освещение от них полностью отраженное.

Вариант проекта Фотография

Рис. 4.8. Вариант проекта и реальная фотография построенного помещения.

Рис. 4.9. Моделирование внешнего вида небоскреба. 1920x1200, GTX560, 5 минут (Монте-Карло); 2.2M треугольников. Моделируется эффект глубины резкости.

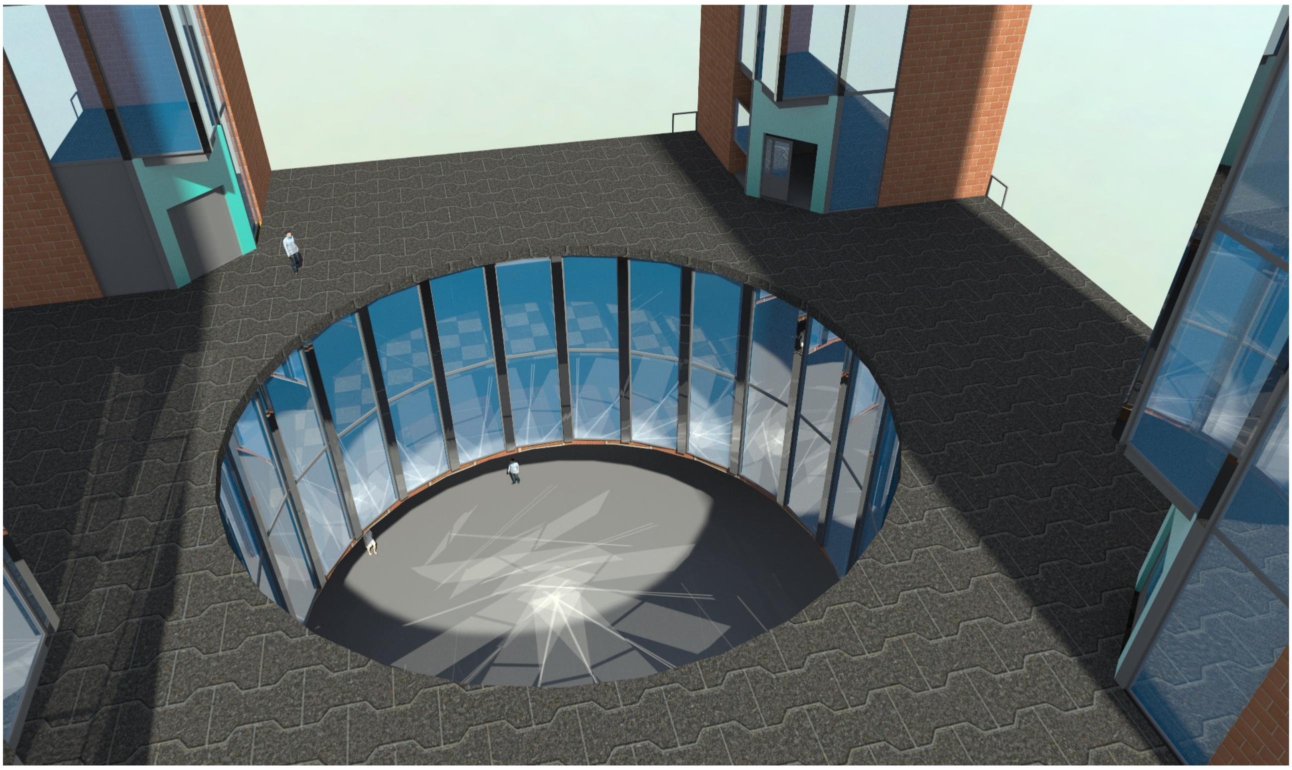


Рис. 4.10. Моделирование нежелательных отражений от зеркальных элементов конструкции. 1920x1200, GTX560, 5 минут (Монте-Карло + SPPM); 2.2M треугольников.

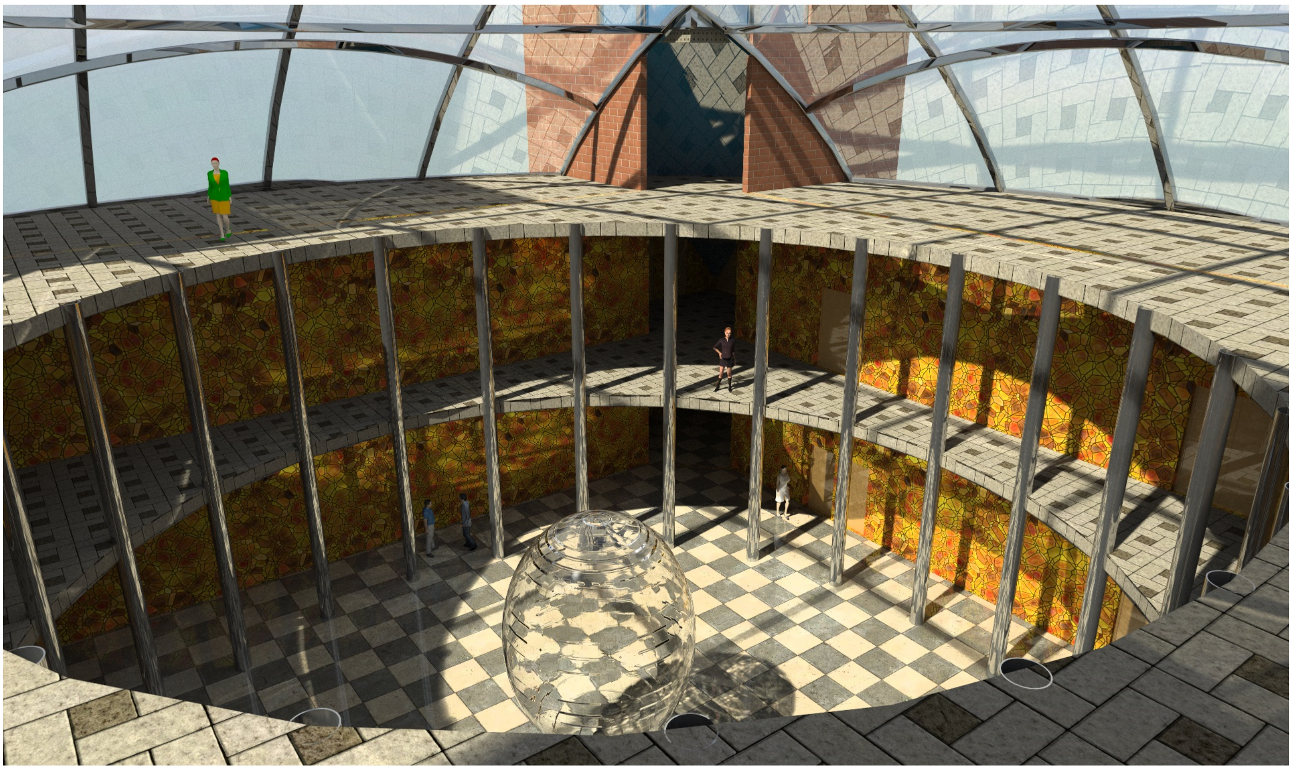


Рис. 4.11. Моделирование освещения внутри небоскреба. 1920x1200, GTX560, 5 минут (Монте-Карло + Кэш освещенности + SPPM); 2.2M треугольников.

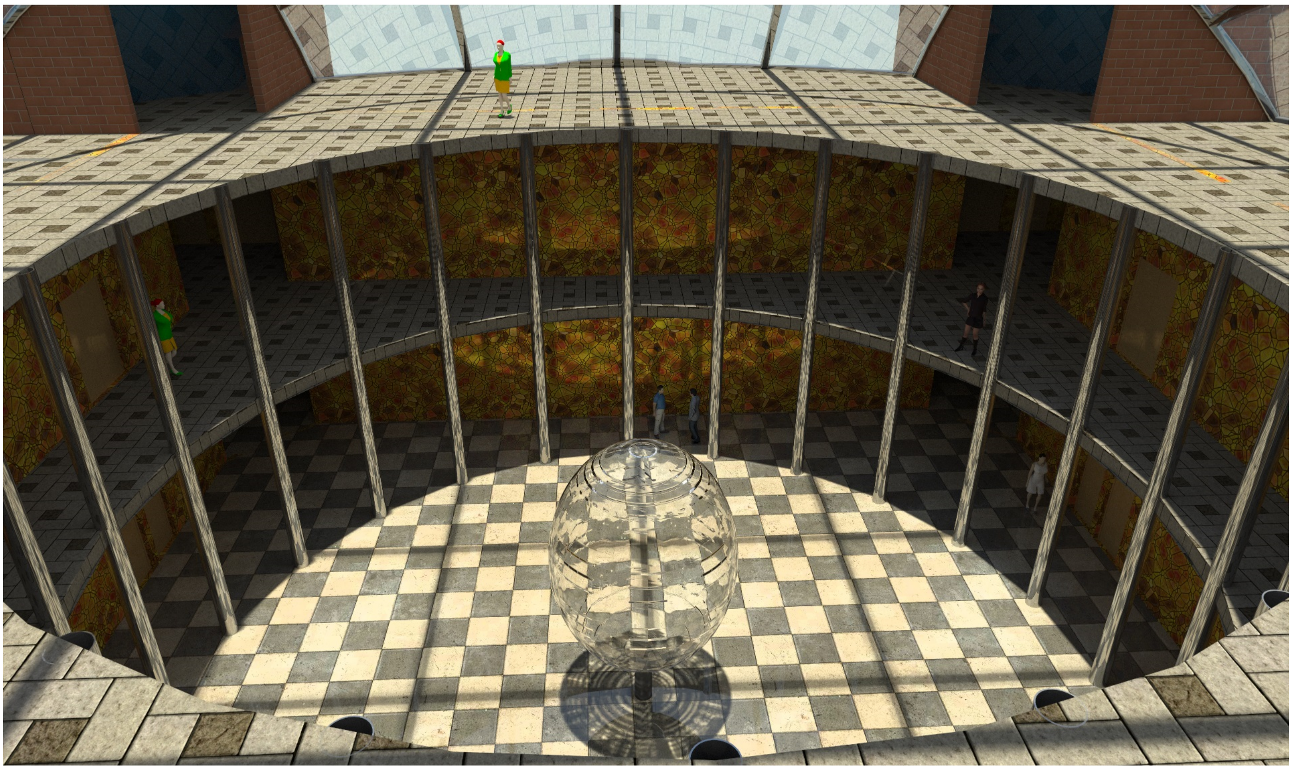


Рис. 4.12. Моделирование освещения внутри небоскреба. 1920x1200, GTX560, 5 минут (Монте-Карло + Кэш освещенности + SPPM); 2.2M треугольников.

4.4. Сравнение

Сравнение производилось на машине с процессором Inter Core i7-3770 (3.4 Ghz), 16-ю GB оперативной памяти, видеокартой GTX680 (2 GB видеопамяти) и 64 битной ОС Windows 7.

4.4.1. Сравнение по скорости трассировки лучей

На рисунках 4.13, 4.14 представлено сравнение разработанной системы по скорости трассировки лучей на двух сценах. За счет реализации раннего подразбиения примитивов ('Early Split Clipping') [16] при построении BVH дерева реализованная система на современных видеокартах с объемом видеопамяти около 1 Gb показывает хорошие результаты на сценах до 5-6 миллионов примитивов, занимая второе место после самой быстрой в мире реализации трассировки лучей [17] (рисунок 4.13). При большем размере сцены в целях экономии памяти раннее подразбиение производится с меньшей величиной, в результате чего эффективность трассировки падает (рисунок 4.14).

На рисунках 4.13, 4.14 для каждой системы представлено 3 различных колонки — скорость трассировки первичных, вторичных и третичных лучей. Первичные лучи — это лучи, выпущенные из камеры. Вторичные — это лучи после одного диффузного переотражения. Третичные — после двух диффузных переотражений. Вследствие особенностей выполнения кода на GPU, вторичные и третичные лучи порождают значительное количество расходящихся потоков, что значительно замедляет скорость трассировки некогерентных вторичных и третичных лучей по сравнению с первичными.

4.4.2. Сравнение по времени построения изображения

Сравнение рендер-систем, использующих алгоритмы со смещенным решением, не является тривиальной задачей. Различный характер артефактов и отсутствие возможности прямого управления временем расчета во многих случаях не позволяет сделать однозначного вывода в пользу той или иной системы.

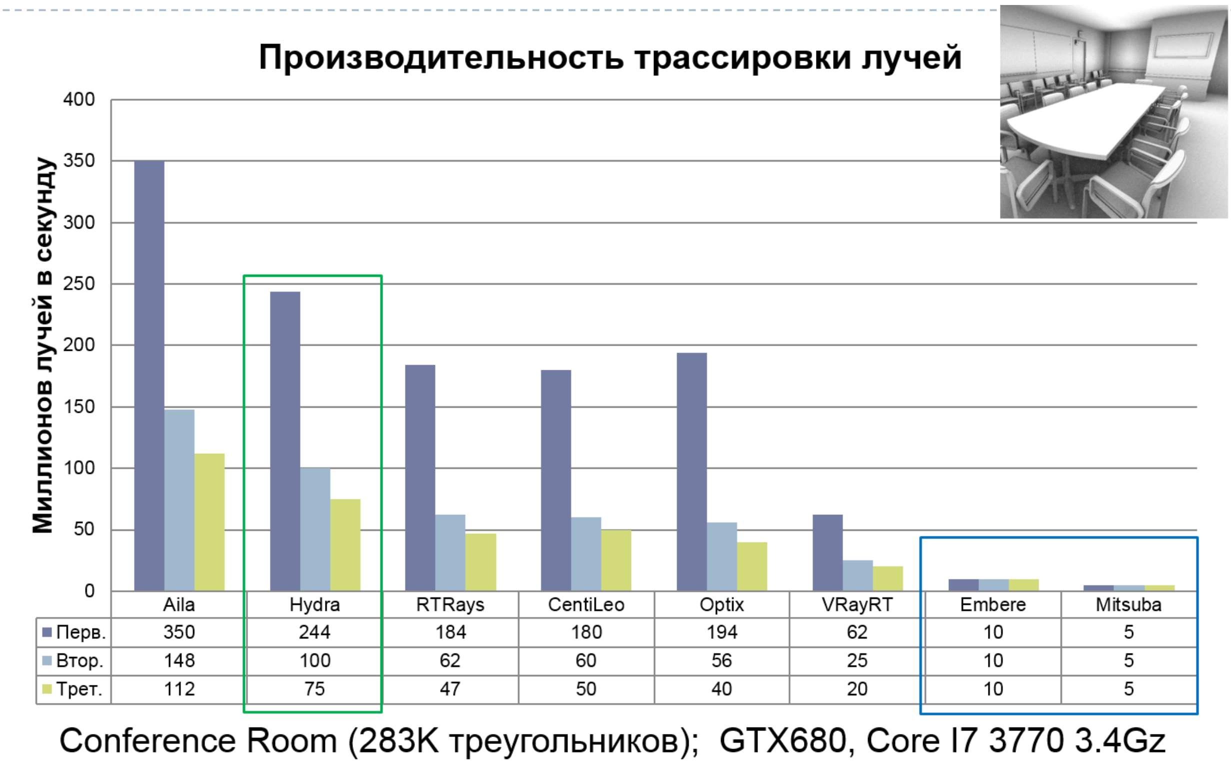


Рис. 4.13. Сравнение разработанной системы ’Hydra’ с другими системами по скорости трассироки лучей на сцене ’Conrefence Room’ (283 тыс. треугольников). Разработанная система выделена зеленым прямоугольником. Системы, использующие центральный процессор — синим прямоугольником.

4.4.3. Методика сравнения

Для оценки точности была использована квадратичная ошибка, посчитанная при помощи программы ’The Compressonator’ (MSE, Mean Square Error) [153]. Далее везде значения MSE были посчитаны между конкретными изображениями и эталоном для данной системы и данной сцены:

1. Для каждой системы и каждого сценария рассчитывался свой эталон при помощи метода Монте-Карло с предельно-большим временем расчета, с которым впоследствии производилось сравнение. Это позволяет исключить различия реализации отдельных незначительных моментов и сосредоточиться на сравнении скорости интегрирования.

2. Для всех систем производилось сравнение двух типов. Сначала фиксировалась небольшая ошибка (насколько это возможно) и измерялось время синтеза изображения. Так сравнивалось время синтеза качественного

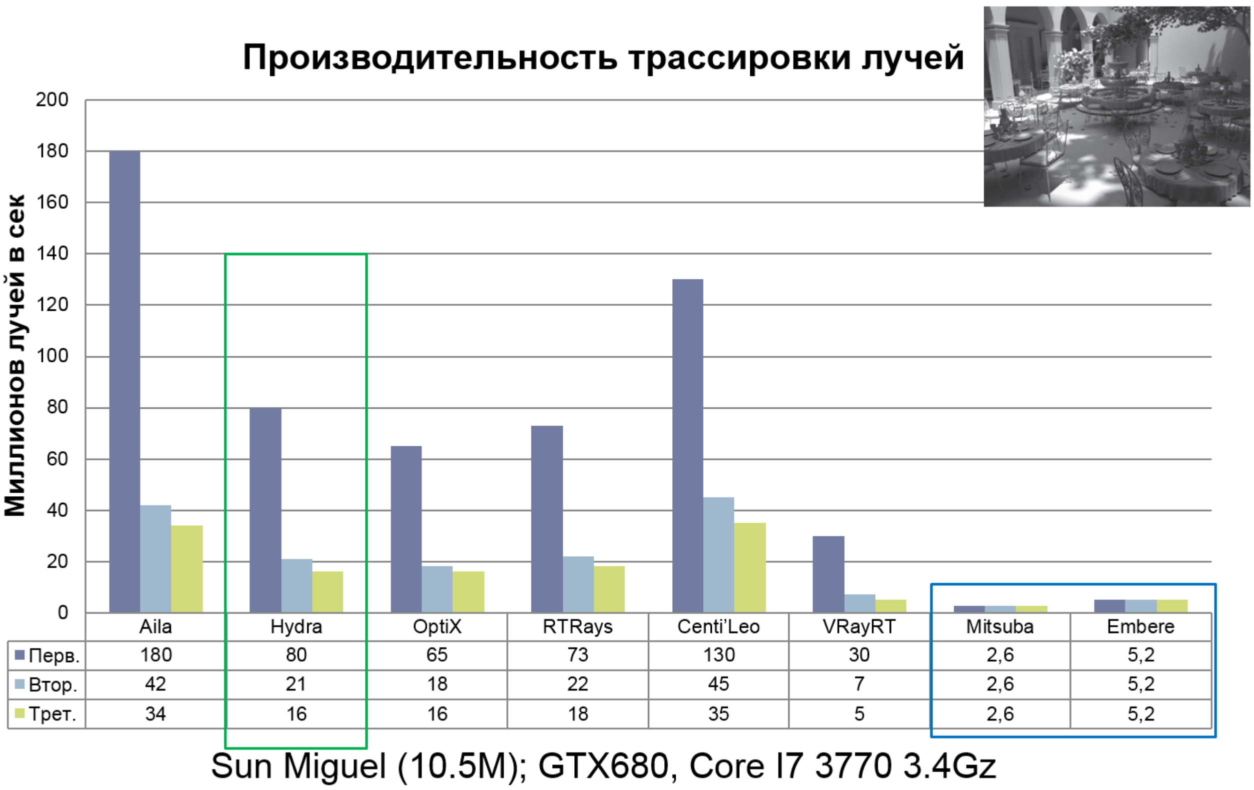


Рис. 4.14. Сравнение разработанной системы 'Hydra' с другими системами по скорости трассироки лучей на сцене 'San Miguel' (10.5) миллионов треугольников. Разработанная система выделена зеленым прямоугольником. Системы, использующие центральный процессор — синим прямоугольником.

изображения. Затем фиксировалось время синтеза изображения (обычно небольшое, в пределах 1 минуты) и измерялась ошибка. Так сравнивалось качество полученных изображений в условиях фиксированного времени. В качестве результата брался средний абсолютный индекс производительности (см. далее) по обоим тестам.

Абсолютный и относительный индекс производительности

Для того чтобы оценить производительность рендер-систем на различных сценах и сопоставить их друг с другом, введем абсолютный индекс производительности (формула 4.1).

$$P_I = \frac{1}{MSE^2 * t} \tag{4.1}$$

Где MSE — квадратичная ошибка (Mean Square Error), вычисляемая при помощи программы 'The Compressonator', а t — время в секундах. Если сцена, условия освещения и оборудование фиксированны, отношение индексов производительности для двух систем будет адекватно отражать отношение производительности этих систем. Однако, зависимость сравнения от оборудования и абсолютные значения индекса снижают наглядность сравнения. По этой причине введем относительный индекс производительности (формула 4.2).

$$P_{I(Rel)} = 100 * \frac{P_I}{MAX(P_{I(VRay)}, P_{I(MentalRay)}, P_{I(IRay)}, ...)} \tag{4.2}$$

Относительный индекс производительности будет равен 100 баллам для системы, которая является на данной сцене самой быстрой. Остальные системы получат баллы пропорционально тому, насколько они медленее. Таким образом, относительный индекс производительности не зависит от сложности сцены и его можно усреднить по всем тестовым сценам, оценив насколько система проявила себя в комплексе.

4.4.4. Описание рендер-систем

Mental Ray

Mental Ray – один из пионеров среди систем рендеринга со смещенным решением [118]. Со временем система не потеряла популярность и продолжает активно развиваться. Используется преимущественно в мультипликации. Ключевой особенностью системы является высокая степень гибкости, позволяющая дизайнеру настроить расчет в точности так как нужно. Ключевыми алгоритмами расчета являются кэш освещенности в сочетании с финальным сбором используя фотонные карты [118]. Также может использовать фотонные карты для расчета каустиков. Алгоритм интерполяции освещенности интересен тем, что использует не строго ближайшие точки как в классической реализации кэша освещенности, а фильтрует освещение, учитывая многие точки.

VRay

VRay является одой из наиболее популярных рендер-систем в архитектуре. VRay – CPU рендер со смещенным решением. Ключевыми алгоритмами являются кэш освещенности (называемый в VRay термином ’Irradiance Maps’) и карты светимости (называемые в VRay термином ’Light Cache’). Каустики в VRay могут рассчитываться при помощи фотонных карт. Для сравнения была использована новейшая версия VRay 3.0.

IRay

IRay является гибридной CPU/GPU рендер-системой с несмещенным решением, частично совместимой с Mental Ray. IRay использует GPU для ускорения трассировки лучей [154]. Исходя из документации, в IRay реализован один из вариантов алгоритма переноса света Метрополиса.

VRayRT

Частично совместимая с VRay GPU рендер-система с несмещенным решением. В системе реализована только простая Монте-Карло трассировка путей. Эффективный расчет каустик невозможен.

Octane

Система Octane – одна из первых рендер-систем с несмещенным решением, реализованная полностью на GPU. В Octane для ускорения расчета каустик реализован один из вариантов переноса света Метрополиса, называемый в системе Octane ’PMC’.

Corona

Созданная в 2012 году CPU рендер-система со смещенным и несмещенным решениями [120]. В отличие от VRay и Mental Ray, Corona использует кэш освещенности только для третьего переотражения, выполняя, таким образом, попиксельно финальный сбор. Отличительной чертой системы Corona является алгоритм Vertex Connection Merging (VCM), значительно ускоряющий расчет каустик. Кроме этого, в системе Corona реализована двунаправленная трассировка лучей (BDPT) и двунаправленные прогрессивные фотонные карты (BDPM).

Hydra

Эта система получена в результате настоящей диссертационной работы.

4.4.5. Выводы

На основе проведенного сравнения (приложение А) можно заключить, что разработанная система демонстрирует наибольший выигрыш (4-5 раз) при вычислении вторичной освещенности и каустиков (рис. 4.15), что подтверждает высокую эффективность разработанных методов по отдельности. Разработанная система значительно превосходит существующие аналоги по производительности и в среднем (рис. 4.16), что доказывает эффективность разработанных методов в комплексе, — в условиях, приближенных к реальному использованию системы.

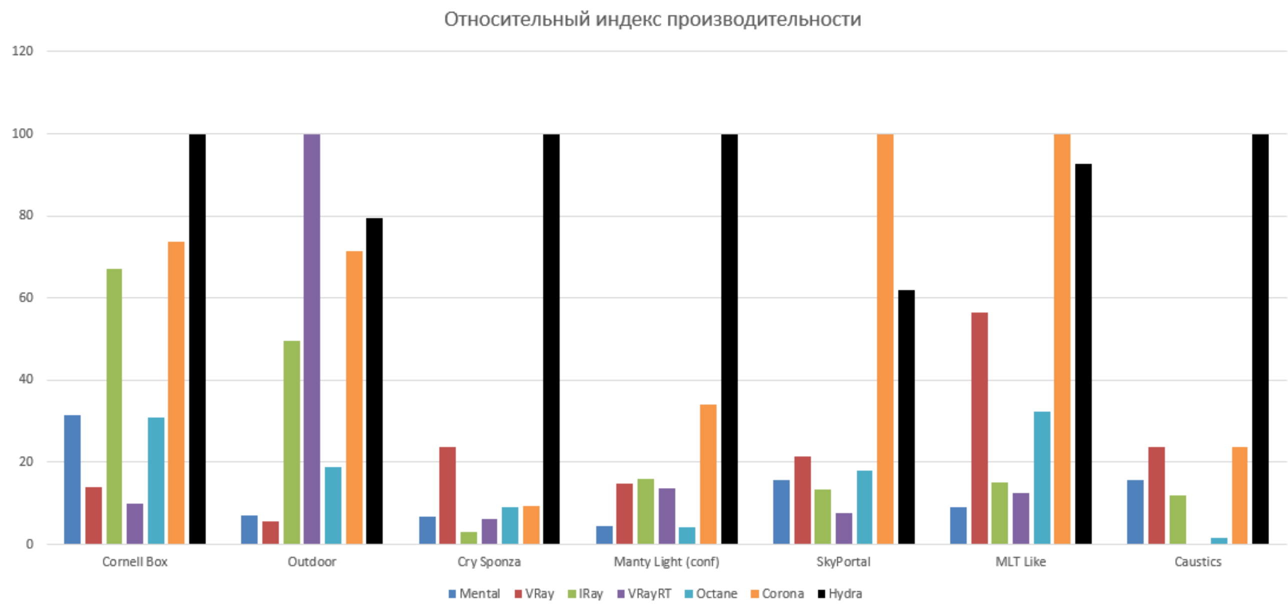


Рис. 4.15. Сравнение рендер-систем по относительному индексу производительности (на различных сценах).

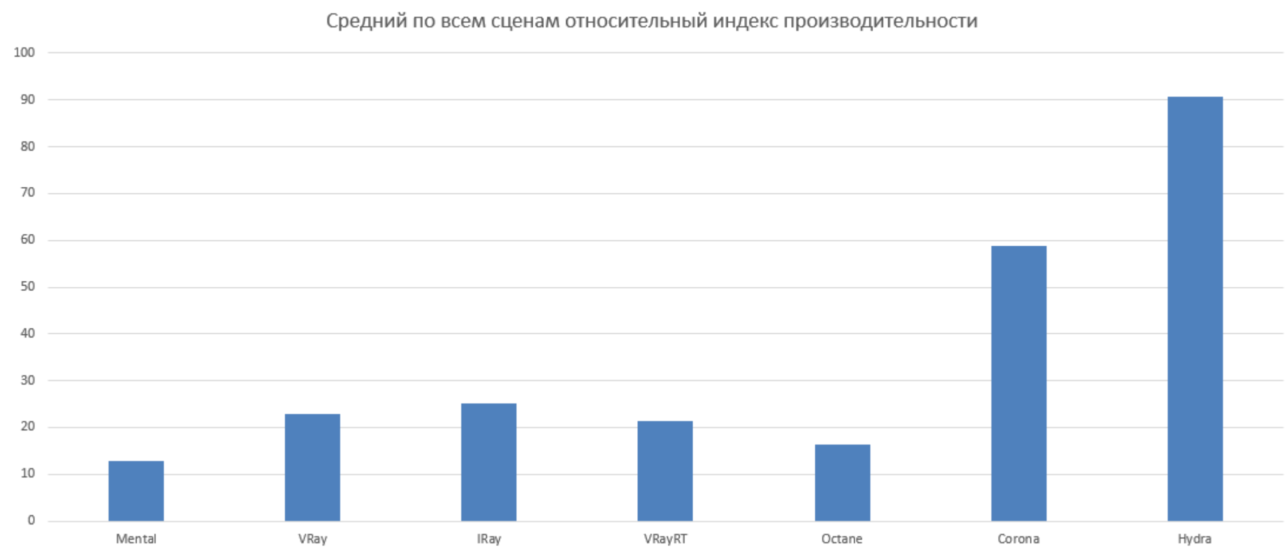


Рис. 4.16. Сравнение рендер-систем по относительному индексу производительности (среднее по всем сценам).

Заключение

Основными результатами работы являются следующие:

1. Разработаны новые эффективные параллельные алгоритмы вычисления глобальной освещенности на графических процессорах на основе алгоритмов со смещенной оценкой решения.

   а. Разработан алгоритм распределения работы для эффективной реализации метода Монте-Карло на графических процессорах в применении к задаче вычисления глобальной освещенности и фотореалистичного синтеза изображений.

   б. Предложена параллельная реализация алгоритма кэширования освещенности на графических процессорах, позволяющая снизить время построения изображения на порядок по сравнению с традиционным методом Монте-Карло при сравнимом качестве получаемого изображения.

   в. Разработан метод построения ускоряющей структуры на графических процессорах, позволяющий от 2 до 5 раз ускорить сбор освещенности в алгоритме фотонных карт по сравнению с существующими методами.

2. На основе разработанных алгоритмов создана система расчета освещенности на графических процессорах, в несколько раз превосходящая по скорости существующие аналоги. Разработанная система интегрирована в среду трехмерного моделирования 3d Studio Max 2013 и 2014 и находит применение при решении практических задач в науке и промышленности.

Литература

1. Frolov Vladimir, Kharlamov Alexander, Ignatenko Alexei. Biased Global Illumination via Irradiance Caching and Adaptive Path Tracing on GPUs // Proceedings of GraphiCon'2010 international conference on computer graphics and vision. St.Petersburg, 2010. P. 49–56. URL: http://www.graphicon.ru/proceedings/2010/conference/EN/Se2/43.pdf.

2. Фролов В. А., Харламов А. А., Игнатенко А. В. Смещённое решение интегрального уравнения светопереноса на графических процессорах при помощи трассировки путей и кэша освещенности // ПРОГРАММИРОВАНИЕ. 2011. Т. 37, № 5. С. 47–60. English translation: V.A. Frolov, A.A. Kharlamov, A.V. Ignatenko. Biased solution of integral illumination equation via irradiance caching and path tracing on GPUs // Programming and Computer Software. 2011. vol 31. P 255-259. URL: http://dx.doi.org/10.1134/S0361768811050021.

3. Frolov Vladimir, Vostryakov Konstantin, Kharlamov Alexander, Galaktionov Vladimir. Implementing Irradiance Cache in a GPU Realistic Renderer // Transactions on Computational Science XIX. Lecture Notes in Computer Science. 2013. Vol. 7870, no. 1. P. 17–32. URL: http://dx.doi.org/10.1007/978-3-642-39759-2_2.

4. Фролов В.А., Харламов А.А., Галактионов В.А., Востряков К.А. Окто-деревья со множественными ссылками в применении к реализации фотонных карт и кэша освещенности на GPU. // ПРОГРАММИРОВАНИЕ. 2014. Т.40. №4., С. 64-73. English translation: Frolov V.A. Galaktionov V.A. Vostryakov K.A., Kharlamov A.A. Multiple reference octrees for a GPU photon mapping and irradiance caching // Programming and Computer Software. 2014. Vol. 40, no 4. P. 208–214. URL: http://dx.doi.org/10.1134/S0361768814040033.

5. Фролов В. А., Игнатенко А. В. Интерактивная трассировка лучей и фотонные карты на GPU. // Труды 22-й Международной конферен-

ции по компьютерной графике и Зрению Графикон’2009. Москва: Московский государственный университет им. М.В.Ломоносова, 2009. С. 255–262. URL: http://www.graphicon.ru/proceedings/2009/conference/se12/97/97_Paper.pdf.

6. Фролов В. А., Игнатенко А. В. Фотореалистичная визуализация с помощью трассировки путей на графических процессорах // Новые информационные технологии в автоматизированных системах. Материалы 13-ого научно-практического семинара. М.: МГИЭМ. 2010. С. 149–150.

7. Груздев А.А., Фролов А.В., Игнатенко А.В. Ускорение расчёта вторичного освещения с помощью фильтрации в пространстве экрана и уточнения на основе информации о близлежащей геометрии. // Труды 25-й Международной конференции по компьютерной графике и зрению Графикон’2012. Москва: Московский государственный университет им. М.В.Ломоносова, 2012. С. 269–271.

8. Gruzdev Alexei, Frolov Vladimir, Vostryakov Konstantin, Ignatenko Alexei. Multidimetial Filtering in application to Progressive Video Rendering // Proceedings of 23-th International Conference on Computer Graphics and Vision GraphiCon’2013. Vladivostok: Far Eastern Federal University, 2013. P. 75–78. URL: http://2013.graphicon.ru/files/2013/u8/Graphicon2013_proceedings.pdf.

9. Боресков А.В., Харламов А.А., Марковский Н.Д., Микушин Д.Н., Мортиков Е.В., Мыльцев А.А., Сахарных Н.А., Фролов В.А. Параллельные вычисления на GPU. Архитектура и программная модель CUDA. Москва: МГУ, 2012. 336 с. ISBN: 978-5-211-06340-2.

10. Фролов В.А., Галактионов В.А., Трофимов М.А. Сравнительный анализ современных рендер-систем для 3ds Max // Труды 24-й Международной конференции по компьютерной графике и зрению Графикон’2014. Южный Федеральный Университет, Ростов-на-Дону, Россия, 2014. С. 43–46. URL: http://2014.graphicon.sfedu.ru/res/Trudi_GraphiCon_2014.pdf.

11. Whitted Turner. An improved illumination model for shaded display // Commun. ACM. 1980. — jun. Vol. 23, no. 6. P. 343–349. URL: http://doi.acm.org/10.1145/358876.358882.

12. Rubin Steven M., Whitted Turner. A 3-dimensional Representation for Fast Rendering of Complex Scenes // SIGGRAPH Comput. Graph. 1980. — jul. Vol. 14, no. 3. P. 110–116. URL: http://doi.acm.org/10.1145/965105.807479.

13. Möller Tomas, Trumbore Ben. Fast, Minimum Storage Ray-triangle Intersection // J. Graph. Tools. 1997. — oct. Vol. 2, no. 1. P. 21–28. URL: http://dx.doi.org/10.1080/10867651.1997.10487468.

14. Woop Sven. A Ray Tracing Hardware Architecture for Dynamic Scenes: Tech. rep.: Saarland University, 2004.

15. Williams Amy, Barrus Steve, Keith R., Shirley Morley Peter. An efficient and robust ray-box intersection algorithm // Journal of Graphics Tools. 2003. Vol. 10. P. 54.

16. Ernst Manfred, Greiner Gunther. Early Split Clipping for Bounding Volume Hierarchies // Proceedings of the 2007 IEEE Symposium on Interactive Ray Tracing. RT '07. Washington, DC, USA: IEEE Computer Society, 2007. P. 73–78. URL: http://dx.doi.org/10.1109/RT.2007.4342593.

17. Aila Timo, Laine Samuli. Understanding the efficiency of ray traversal on GPUs // Proceedings of the Conference on High Performance Graphics 2009. HPG '09. New York, NY, USA: ACM, 2009. P. 145–149. URL: http://doi.acm.org/10.1145/1572769.1572792.

18. Nicodemus Fred E. Directional Reflectance and Emissivity of an Opaque Surface // Applied Optics. 1965. — July. Vol. 4, no. 7. P. 767–773.

19. Kajiya James T. The rendering equation // SIGGRAPH Comput. Graph. 1986. — aug. Vol. 20, no. 4. P. 143–150. URL: http://doi.acm.org/10.1145/15886.15902.

20. Соболь И. М. Численные Методы Монте-Карло. 1 Издание. М.: Наука., 1973.

21. Goral Cindy M., Torrance Kenneth E., Greenberg Donald P., Battaile Bennett. Modeling the interaction of light between diffuse surfaces // Proceedings of the 11th annual conference on Computer graphics and interactive techniques. SIGGRAPH '84. New York, NY, USA: ACM, 1984. P. 213–222. URL: http://doi.acm.org/10.1145/800031.808601.

22. Veach Eric. Robust monte carlo methods for light transport simulation: Ph. D. thesis. Stanford, CA, USA: Stanford University, 1998. AAI9837162.

23. Heckbert Paul S. Adaptive radiosity textures for bidirectional ray tracing // SIGGRAPH Comput. Graph. 1990. — sep. Vol. 24, no. 4. P. 145–154. URL: http://doi.acm.org/10.1145/97880.97895.

24. Боголепов Денис. Методы глобального освещения для интерактивного синтеза изображений сложных сцен на графических процессорах: Кандидатская диссертация. НГГУ им. Лобачевского, Нижний Новгород, Россия: Нижний Новгород, 2013.

25. Боголепов Д.К., Турлапов В.Е., Ульянов Д.Я. Об одной реализации трассировки путей для графического процессора. НГГУ им. Лобачевского, Нижний Новгород, Россия: Н. Новгород: Изд-во Нижегор. гос. ун-та, 2013. С. 42–46.

26. Georgiev Iliyan, Křivánek Jaroslav, Slusallek Philipp. Bidirectional light transport with vertex merging // SIGGRAPH Asia 2011 Sketches. SA '11. New York, NY, USA: ACM, 2011. P. 27:1–27:2. URL: http://doi.acm.org/10.1145/2077378.2077412.

27. Veach Eric, Guibas Leonidas J. Metropolis light transport // Proceedings of the 24th annual conference on Computer graphics and interactive techniques. SIGGRAPH '97. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1997. P. 65–76. URL: http://dx.doi.org/10.1145/258734.258775.

28. Cline David, Egbert Parris. A Practical Introduction to Metropolis Light Transport: Tech. rep.: Brigham Young University, 2005.

29. Jensen Henrik Wann, Christensen Per. High quality rendering using ray tracing and photon mapping // ACM SIGGRAPH 2007 courses. SIGGRAPH '07. New York, NY, USA: ACM, 2007. URL: http://doi.acm.org/10.1145/1281500.1281593.

30. Tabellion Eric, Lamorlette Arnauld. An approximate global illumination system for computer generated films // ACM SIGGRAPH 2004 Papers. SIGGRAPH '04. New York, NY, USA: ACM, 2004. P. 469–476. URL: http://doi.acm.org/10.1145/1186562.1015748.

31. Востряков Константин. Глобальное освещение с помощью октанных текстур // Материалы 16-ой международной конференции Графикон'2006. GraphiCon'06. Новосибирск, Россия: Графикон, 2006.

32. Hachisuka Toshiya, Ogaki Shinji, Jensen Henrik Wann. Progressive photon mapping // ACM Trans. Graph. 2008. — dec. Vol. 27, no. 5. P. 130:1–130:8. URL: http://doi.acm.org/10.1145/1409060.1409083.

33. Hachisuka Toshiya, Jensen Henrik Wann. Stochastic progressive photon mapping // ACM Trans. Graph. 2009. — dec. Vol. 28, no. 5. P. 141:1–141:8. URL: http://doi.acm.org/10.1145/1618452.1618487.

34. Hachisuka Toshiya, Jarosz Wojciech, Bouchard Guillaume et al. State of the art in photon density estimation // ACM SIGGRAPH 2012 Courses. SIGGRAPH '12. New York, NY, USA: ACM, 2012. P. 6:1–6:469. URL: http://doi.acm.org/10.1145/2343483.2343489.

35. Suykens Frank, Willems Yves D. Density Control for Photon Maps // Proceedings of the Eurographics Workshop on Rendering Techniques 2000. London, UK, UK: Springer-Verlag, 2000. P. 23–34. URL: http://dl.acm.org/citation.cfm?id=647652.732120.

36. Ward Gregory J., Rubinstein Francis M., Clear Robert D. A ray tracing solution for diffuse interreflection // SIGGRAPH Comput. Graph. 1988. — jun.

Vol. 22, no. 4. P. 85–92. URL: http://doi.acm.org/10.1145/378456.378490.

37. Krivanek Jaroslav, Gautron Pascal. Practical Global Illumination with Irradiance Caching (Synthesis Lectures in Computer Graphics and Animation). Morgan and Claypool Publishers, 2009. ISBN: 1598296442, 978-1598296440.

38. Лебедев Андрей. История развития алгоритмов глобального освещения // Компьютерная графика и мультимедиа; сетевой журнал. 2011. Т. 1, № 9. 23 с. URL: http://cgm.computergraphics.ru/issues/issue19/globalillum.

39. Cohen Michael F., Wallace John, Hanrahan Pat. Radiosity and realistic image synthesis. San Diego, CA, USA: Academic Press Professional, Inc., 1993. ISBN: 0-12-178270-0.

40. Kenny Magnusson. Lighting you up in Battlefield 3. 2013. URL: http://dice.se/publications/lighting-you-up-in-battlefield-3/.

41. Keller Alexander. Instant radiosity // Proceedings of the 24th annual conference on Computer graphics and interactive techniques. SIGGRAPH '97. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1997. P. 49–56. URL: http://dx.doi.org/10.1145/258734.258769.

42. Walter Bruce, Fernandez Sebastian, Arbree Adam et al. Lightcuts: a scalable approach to illumination // ACM SIGGRAPH 2005 Papers. SIGGRAPH '05. New York, NY, USA: ACM, 2005. P. 1098–1107. URL: http://doi.acm.org/10.1145/1186822.1073318.

43. Křivánek Jaroslav, Hašan Miloš, Arbree Adam et al. Optimizing realistic rendering with many-light methods // ACM SIGGRAPH 2012 Courses. SIGGRAPH '12. New York, NY, USA: ACM, 2012. P. 7:1–7:217. URL: http://doi.acm.org/10.1145/2343483.2343490.

44. Walter Bruce, Khungurn Pramook, Bala Kavita. Bidirectional lightcuts // ACM Trans. Graph. 2012. — jul. Vol. 31, no. 4. P. 59:1–59:11. URL: http://doi.acm.org/10.1145/2185520.2185555.

45. Ou Jiawei, Pellacini Fabio. LightSlice: matrix slice sampling for the many-lights problem // Proceedings of the 2011 SIGGRAPH Asia Conference. SA '11. New York, NY, USA: ACM, 2011. P. 179:1–179:8. URL: http://doi.acm.org/10.1145/2024156.2024213.

46. Christensen Per H. Point Based Global Illumination // SIGGRAPH 2010 Course: Global Illumination Across Industries. SIGGRAPH '10. New York, NY, USA: ACM, 2010.

47. Green Robin. Spherical Harmonic Lighting: The Gritty Details. 2003.

48. Hanrahan Pat, Salzman David, Aupperle Larry. A rapid hierarchical radiosity algorithm // SIGGRAPH Comput. Graph. 1991. — jul. Vol. 25, no. 4. P. 197–206. URL: http://doi.acm.org/10.1145/127719.122740.

49. Kelemen Csaba, Szirmay-Kalos Laszlo, Antal Gyorgy, Csonka Ferenc. A Simple and Robust Mutation Strategy for the Metropolis Light Transport Algorithm // Computer Graphics Forum. Vol. 23. 2002. P. 531–540.

50. Cline David, Talbot Justin, Egbert Parris. Energy redistribution path tracing // ACM Trans. Graph. 2005. — jul. Vol. 24, no. 3. P. 1186–1195. URL: http://doi.acm.org/10.1145/1073204.1073330.

51. Lehtinen Jaakko, Karras Tero, Laine Samuli et al. Gradient-domain metropolis light transport // ACM Trans. Graph. 2013. — jul. Vol. 32, no. 4. P. 95:1–95:12. URL: http://doi.acm.org/10.1145/2461912.2461943.

52. Georgiev Iliyan, Křivánek Jaroslav, Davidovič Tomáš, Slusallek Philipp. Light transport simulation with vertex connection and merging // ACM Trans. Graph. 2012. — nov. Vol. 31, no. 6. P. 192:1–192:10. URL: http://doi.acm.org/10.1145/2366145.2366211.

53. Kaplanyan Anton S., Dachsbacher Carsten. Path Space Regularization for Holistic and Robust Light Transport // Computer Graphics Forum (Proc. of Eurographics 2013). 2013. Vol. 32, no. 2.

54. Schjøth Lars, Frisvad Jeppe Revall, Erleben Kenny, Sporring Jon. Photon differentials // Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia. GRAPHITE '07. New York, NY, USA: ACM, 2007. P. 179–186. URL: http://doi.acm.org/10.1145/1321261.1321293.

55. Igehy Homan. Tracing ray differentials // Proceedings of the 26th annual conference on Computer graphics and interactive techniques. SIGGRAPH '99. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1999. P. 179–186. URL: http://dx.doi.org/10.1145/311535.311555.

56. Spencer Ben, Jones Mark W. Progressive photon relaxation // ACM Trans. Graph. 2013. Vol. 32, no. 1. P. 7:1–7:11.

57. Havran Vlastimil, Herzog Robert, Seidel Hans-Peter. Fast Final Gathering via Reverse Photon Mapping // Computer Graphics Forum (Proceedings of Eurographics 2005). 2005. — August. Vol. 24, no. 3. P. 323–333.

58. Bekaert Philippe, Slusallek Philipp, Cools Ronald et al. A custom designed density estimation method for light transport: Research Report MPI-I-2003-4-004. Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany: Max-Planck-Institut für Informatik, 2003. — September.

59. Vorba Jiri, Krivánek Jaroslav. Bidirectional Photon Mapping // CESCG 2011. 2011.

60. Doidge Ian C., Jones Mark W., Mora Benjamin. Mixing Monte Carlo and progressive rendering for improved global illumination // Vis. Comput. 2012. — jun. Vol. 28, no. 6-8. P. 603–612. URL: http://dx.doi.org/10.1007/s00371-012-0703-2.

61. Hachisuka Toshiya, Pantaleoni Jacopo, Jensen Henrik Wann. A path space extension for robust light transport simulation // ACM Trans. Graph. 2012. —

nov. Vol. 31, no. 6. P. 191:1–191:10. URL: http://doi.acm.org/10.1145/2366145.2366210.

62. Jensen Henrik Wann. Importance Driven Path Tracing using the Photon Map // in Eurographics Rendering Workshop. Springer-Verlag, 1995. P. 326–335.

63. Křivánek Jaroslav. Radiance Caching for Global Illumination Computation on Glossy Surfaces: Ph.d. thesis / Université de Rennes 1 and Czech Technical University in Prague. 2005. — December. URL: http://www.cgg.cvut.cz/~havran/dissertation/index.htm.

64. Křivánek Jaroslav, Bouatouch Kadi, Pattanaik Sumanta N., Zara Jiri. Making Radiance and Irradiance Caching Practical: Adaptive Caching and Neighbor Clamping // Rendering Techniques 2006, Eurographics Symposium on Rendering. Nicosia, Cyprus: Eurographics Association, 2006. — June. P. 127–138.

65. Jensen Henrik Wann, Christensen Per. High quality rendering using ray tracing and photon mapping // SIGGRAPH 2002 Cource 43. New York, NY, USA: Association for Computing Machinery, Aug. 2002, ACM SIGGRAPH, 2002. URL: http://www.cs.princeton.edu/courses/archive/fall02/cs526/papers/course43sig02.pdf.

66. Востряков Константин. Высокочастотный кэш излучения // Материалы 16-ой международной конференции Графикон'2009. GraphiCon'09. Москва, Россия: Графикон, 2009.

67. Arikan Okan, Forsyth David A., O'Brien James F. Fast and Detailed Approximate Global Illumination by Irradiance Decomposition // Proceedings of ACM SIGGRAPH 2005. ACM Press, 2005. — july. URL: http://graphics.cs.berkeley.edu/papers/Arikan-FAD-2005-07/.

68. Sen Pradeep, Darabi Soheil. On filtering the noise from the random parameters in Monte Carlo rendering // ACM Trans. Graph. 2012. — jun. Vol. 31, no. 3. P. 18:1–18:15. URL: http://doi.acm.org/10.1145/2167076.2167083.

69. Gastal Eduardo S. L., Oliveira Manuel M. Adaptive Manifolds for Real-Time High-Dimensional Filtering // ACM TOG. 2012. Vol. 31, no. 4. P. 33:1–33:13. Proceedings of SIGGRAPH 2012.

70. Foley Tim, Sugerman Jeremy. KD-tree acceleration structures for a GPU raytracer // Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware. HWWS '05. New York, NY, USA: ACM, 2005. P. 15–22. URL: http://doi.acm.org/10.1145/1071866.1071869.

71. Horn Daniel Reiter, Sugerman Jeremy, Houston Mike, Hanrahan Pat. Interactive k-d tree GPU raytracing // Proceedings of the 2007 symposium on Interactive 3D graphics and games. I3D '07. New York, NY, USA: ACM, 2007. P. 167–174. URL: http://doi.acm.org/10.1145/1230100.1230129.

72. Günther Johannes, Popov Stefan, Seidel Hans-Peter, Slusallek Philipp. Realtime Ray Tracing on GPU with BVH-based Packet Traversal // Proceedings of the IEEE/Eurographics Symposium on Interactive Ray Tracing 2007. 2007. — sep. P. 113–118.

73. Laine Samuli, Karras Tero, Aila Timo. Megakernels Considered Harmful: Wavefront Path Tracing on GPUs // Proceedings of High-Performance Graphics 2013. 2013.

74. NVIDIA Corporation. NVIDIA CUDA C Programming Guide, 2013. — July.

75. Novak Jan. Global Illumination Methods on GPU with CUDA. Master Thesis: Ph. D. thesis. Czech Technical University, Prague: Czech Technical University, Prague, 2009.

76. van Antwerpen Dietger. Improving SIMD efficiency for parallel Monte Carlo light transport on the GPU // Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics. HPG '11. New York, NY, USA: ACM, 2011. P. 41–50. URL: http://doi.acm.org/10.1145/2018323.2018330.

77. Sadeghi Iman, Chen Bin, Jensen Henrik Wann. Coherent Path Tracing // Journal of Graphics, GPU, and Game Tools. 2009. Vol. 14, no. 2. P. 33–43.

78. Antwerpen Dietger Van. Unbiased Physically Based Rendering on the GPU: Ph. D. thesis / Delft University of Technology. Delft, The Netherlands, 2010.

79. Davidovič Tomáš, Křivánek Jaroslav, Hašan Miloš, Slusallek Philipp. Progressive Light Transport Simulation on the GPU: Survey and Improvements // ACM Trans. Graph. 2014. P. XXX:1–XXXX:18.

80. Ward Gregory J. The RADIANCE lighting simulation and rendering system // Proceedings of the 21st annual conference on Computer graphics and interactive techniques. SIGGRAPH '94. New York, NY, USA: ACM, 1994. P. 459–472. URL: http://doi.acm.org/10.1145/192161.192286.

81. Koholka Roland, Mayer Heinz, Goller Alois. MPI-parallelized Radiance on SGI CoW and SMP. // ACPC / Ed. by Peter Zinterhof, Marian Vajtersic, Andreas Uhl. Vol. 1557 of Lecture Notes in Computer Science. Springer, 1999. P. 549–558.

82. Debattista Kurt, Santos Luis Paulo, Chalmers Alan. Accelerating the Irradiance Cache through Parallel Component-Based Rendering . // EGPGV / Ed. by Alan Heirich, Bruno Raffin, Luis Paulo Peixoto dos Santos. Eurographics Association, 2006. P. 27–34.

83. Dubla Piotr, Debattista Kurt, Santos Luis Paulo, Chalmers Alan. Wait-Free Shared-Memory Irradiance Cache. // EGPGV / Ed. by Kurt Debattista, Daniel Weiskopf, Joao Comba. Eurographics Association, 2009. P. 57–64.

84. Pharr Matt, Humphreys Greg. Physically Based Rendering, Second Edition: From Theory To Implementation. 2nd edition. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2010. ISBN: 0123750792, 9780123750792.

85. Gautron Pascal, Krivanek Jaroslav, Bouatouch Kadi, Pattanaik Sumanta N. Radiance Cache Splatting: A GPU-Friendly Global Illumination Algorithm. //

Rendering Techniques / Ed. by Oliver Deussen, Alexander Keller, Kavita Bala et al. Eurographics Association, 2005. P. 55–64.

86. Nvidia. Cube Map OpenGL Tutorial. 1999. URL: http://www.nvidia.com/object/cube_map_ogl_tutorial.html.

87. Purcell Timothy J., Donner Craig, Cammarano Mike et al. Photon Mapping on Programmable Graphics Hardware // Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware. Eurographics Association, 2003. P. 41–50.

88. Zhou Kun, Hou Qiming, Wang Rui, Guo Baining. Real-time KD-tree construction on graphics hardware // ACM Trans. Graph. 2008. — dec. Vol. 27, no. 5. P. 126:1–126:11. URL: http://doi.acm.org/10.1145/1409060.1409079.

89. Wald Ingo, Gunther Johannes, Slusallek Philipp. Balancing Considered Harmful – Faster Photon Mapping using the Voxel Volume Heuristic // Computer Graphics Forum. 2004. Vol. 22, no. 3. P. 595–603. Proceedings of Eurographics.

90. Fabianowski B., Dingliana J. Interactive Global Photon Mapping // Computer Graphics Forum. 2009. Vol. 28, no. 4. P. 1151–1159.

91. Garanzha Kirill, Pantaleoni Jacopo, McAllister David. Simpler and Faster HLBVH with Work Queues // Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics. HPG '11. New York, NY, USA: ACM, 2011. P. 59–64. URL: http://doi.acm.org/10.1145/2018323.2018333.

92. Karras Tero. Maximizing Parallelism in the Construction of BVHs, Octrees, and k-d Trees. 2012. P. 33–37.

93. Боголепов Д.К., Турлапов В.Е. Моделирование каустик в реальном времени на основе комбинированных возможностей OpenCL и шейдеров // Вестник Нижегородского университета им. Н.И. Лобачевского. 2011. Т. 2, № 3. С. 180–186.

94. Alcantara Dan A., Sharf Andrei, Abbasinejad Fatemeh et al. Real-time parallel hashing on the GPU // ACM Trans. Graph. 2009. — dec. Vol. 28, no. 5. P. 154:1–154:9. URL: http://doi.acm.org/10.1145/1618452.1618500.

95. Fleisz Martin. Photon Mapping on the GPU // Master of Science. 2009. Vol. School of Informatics, University of Edinburgh, no. 1. P. 1–60.

96. Hachisuka Toshiya, Jensen Henrik Wann. Parallel progressive photon mapping on GPUs // ACM SIGGRAPH ASIA 2010 Sketches. SA '10. New York, NY, USA: ACM, 2010. P. 54:1–54:1. URL: http://doi.acm.org/10.1145/1899950.1900004.

97. Carlberg Kristofer. Stochastic Progressive Photon Mapping Using Parallel Hashing // Master Thesis. 2011. Vol. Lund University, no. 1. P. 1–51.

98. McGuire Morgan, Luebke David. Hardware-Accelerated Global Illumination by Image Space Photon Mapping // Proceedings of the 2009 ACM SIGGRAPH/EuroGraphics conference on High Performance Graphics. New York, NY, USA: ACM, 2009. — August. URL: http://graphics.cs.williams.edu/papers/PhotonHPG09/.

99. Mara Michael, McGuire Morgan, Luebke David. Toward Practical Real-Time Photon Mapping: Efficient GPU Density Estimation // Interactive 3D Graphics and Games 2013. 2013. — March. URL: http://graphics.cs.williams.edu/papers/PhotonI3D13/.

100. Li Shengren, Simons Lance C., Pakaravoor Jagaseesh Bhaskar et al. kANN on the GPU with Shifted Sorting // Proceedings of High Performance Graphics 2012 / Ed. by Carsten Dachsbacher, Jacob Munkberg, Jacopo Pantaleoni; High Performance Graphics 2012. The Eurographics Association 2012, 2012.

101. Morton G.M. A Computer Oriented Geodetic Data Base and a New Technique in File Sequencing. International Business Machines Company, 1966. URL: http://books.google.ru/books?id=9FFdHAAACAAJ.

102. Ajmera Prekshu, Goradia Rhushabh, Chandran Sharat, Aluru Srinivas. Fast, parallel, GPU-based construction of space filling curves and octrees // Proceedings of the 2008 symposium on Interactive 3D graphics and games. I3D '08. New York, NY, USA: ACM, 2008. P. 10:1–10:1. URL: http://doi.acm.org/10.1145/1342250.1357022.

103. Zhou Kun, Gong Minmin, Huang Xin, Guo Baining. Data-Parallel Octrees for Surface Reconstruction // IEEE Transactions on Visualization and Computer Graphics. 2011. — may. Vol. 17, no. 5. P. 669–681. URL: http://dx.doi.org/10.1109/TVCG.2010.75.

104. Grand Scott Le. Broad-Phase Collision Detection with CUDA // GPU Gems 3 / Ed. by Hubert Nguyen. Addison-Wesley, 2008. P. 697–721.

105. Kaplanyan Anton, Dachsbacher Carsten. Cascaded light propagation volumes for real-time indirect illumination // Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games. I3D '10. New York, NY, USA: ACM, 2010. P. 99–107. URL: http://doi.acm.org/10.1145/1730804.1730821.

106. Crassin Cyril, Neyret Fabrice, Sainz Miguel et al. Interactive Indirect Illumination Using Voxel Cone Tracing // Computer Graphics Forum (Proceedings of Pacific Graphics 2011). 2011. — sep. Vol. 30, no. 7. URL: http://maverick.inria.fr/Publications/2011/CNSGE11b.

107. Papaioannou Georgios. Real-Time Diffuse Global Illumination Using Radiance Hints. // High Performance Graphics / Ed. by Carsten Dachsbacher, William Mark, Jacopo Pantaleoni. Eurographics Association, 2011. P. 15–24.

108. Hašan Miloš, Pellacini Fabio, Bala Kavita. Matrix Row-Column Sampling for the Many-Light Problem // ACM SIGGRAPH 2007 papers. New York, NY, USA: ACM, 2007.

109. Green Robin. Spherical Harmonic Lighting: The Gritty Details // Archives of the Game Developers Conference. 2003. — mar. URL: http://www.research.scea.com/gdc2003/spherical-harmonic-lighting.pdf.

110. OctoNus. DiamCalc. 2014. URL: http://www.octonus.ru/oct/products/3dcalc/standard/.

111. Козлов Дмитрий. Модель взаимодействия света с прозрачными кристаллами для фотореалистического рендеринга: Кандидатская диссертация. Новосибирский национальный исследовательский государственный университет, Новосибирск, Россия: Новосибирск, 2014.

112. PIXAR. Render Man. 2013. URL: http://renderman.pixar.com/view/renderman.

113. Software Side Effects. Mantra. 2013. URL: http://www.sidefx.com/index.php.

114. Angle Solid. Arnold. 2013. URL: https://www.solidangle.com/arnold/.

115. Production Glukoza. Анимационный проект 'Савва' от студии Glukoza Production. 2014. URL: http://www.render.ru/books/show_book.php?book_id=845.

116. Маркалова Надежда. Создание VFX для фильма Метро: огонь, вода и бетонные тюбинги. 2014. URL: http://www.render.ru/books/show_book.php?book_id=1780.

117. Яхин Арман. Интервью с Арманом Яхиным - супервайзером визуальных эффектов фильма 'Сталинград'. 2014. URL: http://www.render.ru/books/show_book.php?book_id=2090.

118. NVIDIA. Mental Ray. 2014. URL: http://www.nvidia.ru/object/nvidia-mental-ray-ru.html.

119. Group Chaos. VRay. 2014. URL: http://www.chaosgroup.com/en/2/vray.html.

120. Karlik Ondra, Hotovy Adam, Krivanek Jaroslav et al. Corona Render. 2014. URL: http://corona-renderer.com/.

121. Costa António Cardoso, Sousa António Augusto, Ferreira Fernando Nunes. Lighting Design: A Goal Based Approach Using Optimisation // Proceedings of the 10th Eurographics Conference on Rendering. EGWR'99. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 1999. P. 317–328. URL: http://dx.doi.org/10.2312/EGWR/EGWR99/317-328.

122. Luxion Inc. KeyShot. 2014. URL: http://www.luxion.com/hypershot/.

123. Software Ecru. ПО проектирования мебели PRO100. 2014. URL: http://ru.pro100.eu/pro100.

124. Autodesk. ПО трехмерного моделирования Autodesk 3ds Max. 2014. URL: http://www.autodesk.ru/products/autodesk-3ds-max/overview.

125. Волобой А.Г., Галактионов. В.А. Компьютерная графика как эффективный инструмент развития современных технологий // Труды 23-й Международной конференции по компьютерной графике и зрению Графикон'2013. Дальневосточный Федеральный Университет: Институт автоматики и процессов управления ДВО РАН, 16-20 сентября 2013. С. 186–190.

126. Галактионов Владимир. Программные технологии синтеза реалистичных изображений: докторская диссертация. Институт прикладной математики им. Келдыша, Москва, Россия. 2006.

127. Волобой Алексей. Программные технологии автоматизации построения реалистичных изображений: докторская диссертация. Институт прикладной математики им. Келдыша, Москва, Россия. 2012.

128. Valiev Ildar, Voloboy Alexey, Galaktionov Vladimir. Improved Model of IBL Sunlight Simulation // Proceedings of the 24th Spring Conference on Computer Graphics. SCCG '08. New York, NY, USA: ACM, 2010. P. 27–32.

129. Adinetz Andrew, Barladian Boris, Galaktionov Vladimir et al. Physically Accurate Rendering with Coherent Ray Tracing. 2006.

130. Zhdanov Dmitry, Ershov Sergey, Pozdnyakov Sergey etal. Simulation of color shift in fluorescent LED cap // Optical Review. 2013. Vol. 20, no 2. P. 132–136.

131. Zhdanov Dmitry, Garbul Alexey, Mayorov Vadim etal. Automatic design of illumination systems // Optical Review. 2013. Vol. 20, no 2. P. 155–159.

132. Волобой Алексей, Галактионов Владимир, Ершов Сергей и др. Аппаратно-программный комплекс для измерения светорассеивающих свойств поверхностей // Информационные технологии и вычислительные системы. 2006. № 4.

133. Волобой Алексей, Ершов Сергей, Клышинский Эдуард, Поздняов С. Моделирование распространения света в тонком красящем слое с высокой концентрацией частиц // Труды 20-ой международной конференции по компьютерной графике и зрению ГРАФИКОН-2010. Санкт-Петербург, Россия: 2010. С. 155–162.

134. Voloboy A.G., Galaktionov V.A., Lobalzo N.A. Simulation and rendering algorithms for optically complex materials by the example of fabric // Programming and Computer Software. 2010. Vol. 36, no 4. P. 237–246.

135. Барладян Борис, Волобой Алексей, Шапиро Лев. Оптимизация представления карт освещенности и яркости для их интерактивной визуализации // Труды 19-ой международной конференции по компьютерной графике и зрению ГРАФИКОН-2009. Москва, Россия: 2009. С. 267–270.

136. Барладян Борис, Волобой Алексей, Вьюкова Н. и др. Моделирование освещенности и синтез фотореалистичных изображений с использованием Интернет технологий // Программирование. 2005. Т. 5. С. 3–18.

137. Redshift Rendering Technologies Inc. RedShift. 2014. URL: https://www.redshift3d.com/products/redshift/.

138. Боголепов Д.К., Д.П Сопин, Ульянов Д.Я. Интерактивное моделирование глобального освещения на GPU для анимированных гетерогенных сцен. 2014. URL: http://graph-unn.ru/rus/projects/rtds.html.

139. Гаранжа Кирилл. CentiLeo. 2014. URL: http://www.centileo.com.

140. Гаранжа Кирилл. Интерактивный синтез реалистичных изображений больших 3D сцен с применением графических процессоров: Кандидатская диссертация. ИПМ имени М.В. Келдыша, Москва, Россия: Москва, 2014.

141. NVIDIA. NVIDIA OptiX ray tracing engine. 2014. URL: https://developer.nvidia.com/optix.

142. CausticGraphics. OpenRL. 2011. URL: https://caustic.com/dev_intro.php.

143. Jensen Henrik Wann, Marschner Stephen R., Levoy Marc, Hanrahan Pat. A Practical Model for Subsurface Light Transport // Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques. SIGGRAPH '01. New York, NY, USA: ACM, 2001. P. 511–518. URL: http://doi.acm.org/10.1145/383259.383319.

144. Jensen Henrik Wann, Christensen Per H. Efficient Simulation of Light Transport in Scenes with Participating Media Using Photon Maps // Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques. SIGGRAPH '98. New York, NY, USA: ACM, 1998. P. 311–320. URL: http://doi.acm.org/10.1145/280814.280925.

145. Hachisuka Toshiya, Jarosz Wojciech, Bouchard Guillaume et al. State of the Art in Photon Density Estimation // ACM SIGGRAPH 2012 Courses. SIGGRAPH '12. New York, NY, USA: ACM, 2012. — jul. P. 6:1–6:469. URL: http://doi.acm.org/10.1145/2343483.2343489.

146. Hachisuka Toshiya, Jarosz Wojciech, Jensen Henrik Wann. A Progressive Error Estimation Framework for Photon Density Estimation // ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH Asia 2010). 2010. — dec. Vol. 29, no. 6. P. 144:1?144:12.

147. Ostromoukhov Victor, Hersch Roger D. Multi-color and Artistic Dithering // Proceedings of the 26th Annual Conference on Computer Graph-

ics and Interactive Techniques. SIGGRAPH ’99. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1999. P. 425–432. URL: http://dx.doi.org/10.1145/311535.311605.

148. Скворцов А. В. Триангуляция Делоне и ее применение. 1 изд. М.: Томск: Изд-во Томск. ун-та, 2002. С. 128.

149. Востряков Константин. Когерентные алгоритмы синтеза реалистичных изображений: Кандидатская диссертация. ИПМ имени М.В. Келдыша, Москва, Россия: Москва, 2009.

150. Татарчук Наталья. Parallax Occlusion Mapping для изображения детальных поверхностей // Доклад на конференции разработчиков игр (КРИ). ATI Research, Inc, 2006.

151. Autodesk. Autodesk Media and Entertainment 2014 SDK Documentation. 2014. URL: http://usa.autodesk.com/adsk/servlet/item?siteID=123112&id=23015025.

152. Reinhard Erik, Stark Michael, Shirley Peter, Ferwerda James. Photographic Tone Reproduction for Digital Images // ACM Trans. Graph. 2002. — jul. Vol. 21, no. 3. P. 267–276. URL: http://doi.acm.org/10.1145/566654.566575.

153. AMD. The Compressonator. 2014. URL: http://developer.amd.com/tools-and-sdks/archive/legacy-cpu-gpu-tools/the-compressonator/.

154. Nvidia. Nvidia IRAY FAQ. 2014. URL: http://area.autodesk.com/blogs/shane/the_iray_faq.

Приложение А
Сравнение

Сравнения изображений производились при помощи программы ’The Compressonator’ [153]. Представленная в таблицах разница MSE соответствует значению квадратичной ошибки в указанной программе [153]. Все измерения производились на машине с процессором Inter Core i7-3770 (3.4 Ghz), 16-ю GB оперативной памяти, видеокартой GTX680 (2 GB видеопамяти) и 64 битной ОС Windows 7.

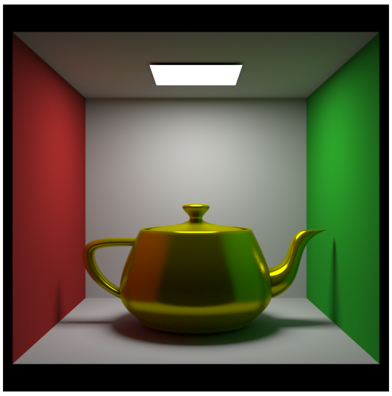
А.1. Teapot Cornell Box

Несмотря на свою простоту в данном сценарии присутствуют практически все основные эффекты трехмерной компьютерной графики: шумное первичное освещение и мягкие тени, зеркальные блики от источника освещения, отраженные каустики. Являясь геометрически простой, сцена в некоторой степени амортизирует стандартные потери производительности GPU трассировщиков лучей на ветвлениях, а CPU трассировщиков лучей на кэш промахах.

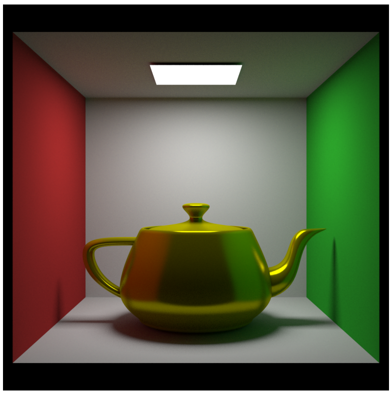
А.1.1. Монте-Карло трассировка, без каустиков

Mental Ray**, 400 сек.
MSE = 2.5

IRay*, 360 сек.
MSE = 2.3



VRay RT3, 50 сек.
MSE = 2.5



VRay3, 600 сек
MSE = 2.5

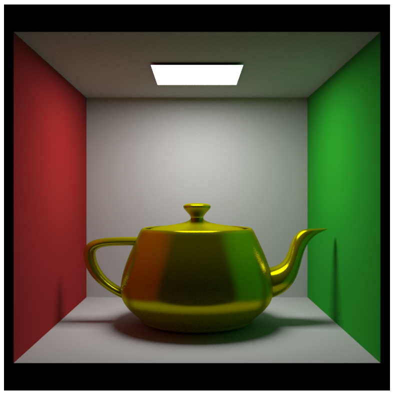
Corona*, 300 сек
MSE = 2.5

Hydra, 60 сек.
MSE = 2.5

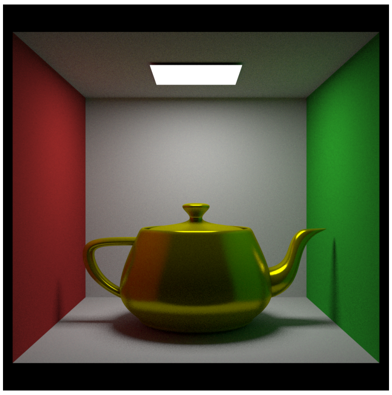
Рис. А.1. Сравнение на сцене ’Cornell Box’. MSE $\approx$ 2.5. Каустики отключены в тех системах, где предусмотрена такая возможность (системы где такая возможность не предусмотрена помечены ’*’). Пояснение к ’**’: Mental Ray не считает каустики от источника типа ’Sky Portal’, но у источника отличается распределение энергии, вследствии чего форма освещения на картинке не совпадает.

Mental Ray, 40 сек.
MSE = 11.34

IRay*, 40 сек.
MSE = 5.9



VRay RT3, 40 сек.
MSE = 3.12



VRay3, 40 сек
MSE = 6.6

Corona*, 40 сек
MSE = 7.5

Hydra, 40 сек.
MSE = 3.7

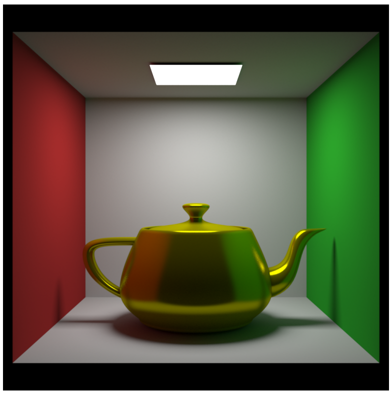
Рис. А.2. Сравнение на сцене 'Cornell Box' для фиксированного времени (40 секунд). Расчет выполнен при помощи трассировки путей. Каустики отключены в тех системах, где предусмотрена такая возможность (системы где такая возможность не предусмотрена помечены '*'). Пояснение к "**": Mental Ray не считает каустики от источника типа 'Sky Portal', но у такого источника отличается распределение энергии, вследствии чего форма освещения на картинке не совпадает.

A.1.2. Монте-Карло трассировка, с каустиками (разл. методами)

Каустики вычислялись различными методами. Для IRay и Corona это Монте-Карло трассировка. Для VRay3, Mental Ray и Hydra это фотонные карты. Выбор метода основан на возможностей системы (например в IRay и Corona не реализован алгоритм фотонных карт, а VRay наоборот может вычислять каустики только фотонными картами) и на основе наилучших настроек для нее. В системе Corona для вычисления каустиков был использован алгоритм Vertex Connection Merging или VCM [26].

Mental Ray, 1150 сек.
MSE = 2.5

IRay, 360 сек.
MSE = 2.3



VRay RT3*, 240 сек.
MSE = 2.5

VRay3, 900 сек
MSE = 2.5

Corona (VCM), 180 сек
MSE = 2.5

Hydra, 120 сек.
MSE = 2.5

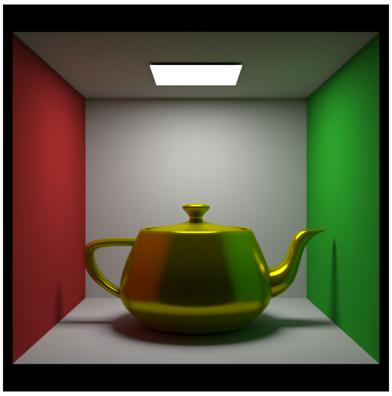
Рис. A.3. Сравнение на сцене 'Cornell Box'. MSE $\approx$ 2.5 Каустики включены. * - система VRayRT не способна рассчитывать каустики. Расчет выполнен при помощи трассировки путей и фотонных карт для каустиков (для систем Mental Ray, VRay, Hydra).

Mental Ray, 40 сек.

MSE = 10.8

IRay, 40 сек.

MSE = 6.0



VRay RT3*, 40 сек.

MSE = 10.6*

VRay3, 40 сек

MSE = 17.3

Corona (VCM), 40 сек

MSE = 6.7

Hydra, 40 сек.

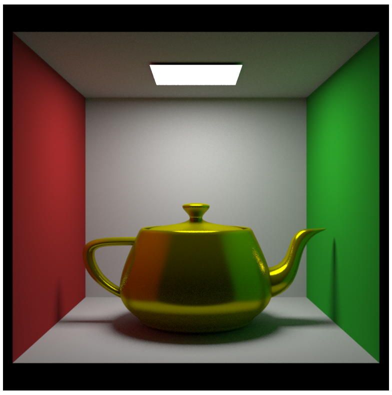
MSE = 4.3

Рис. А.4. Сравнение на сцене 'Cornell Box' для фиксированного времени (40 сек). * - система VRayRT не способна рассчитывать каустики. Отсутствие каустика засчитано за ошибку. Расчет выполнен при помощи трассировки путей. Каустики включены.

A.1.3. Кэш освещенности, без каустиков

Mental Ray, 90 сек.
MSE = 3.9

IRay *, *** 20 сек.
MSE = 8.0



VRay RT3***, 20 сек.
MSE = 4.8

VRay3, 73 сек
MSE = 8.5

Corona*, 20 сек
MSE = 9.22

Hydra, 20 сек.
MSE = 3.0

Рис. A.5. Сравнение на сцене ’Cornell Box’. MSE $\approx$ 4.0. Расчет выполнен при помощи кэша освещенности и трассировки путей. Каустики отключены в тех системах, где предусмотрена такая возможность (системы где такая возможность не предусмотрена помечены ’*’). ** - Mental Ray не считает каустики от источника типа ’Sky Portal’, но у источника отличается распределение энергии, вследствии чего форма освещения на картинке не совпадает. *** - в системе не реализован метод кэширования освещенности или аналогичные методы, вследствии чего использовалась Монте-Карло трассировка.

A.1.4. Кэш освещенности, каустики

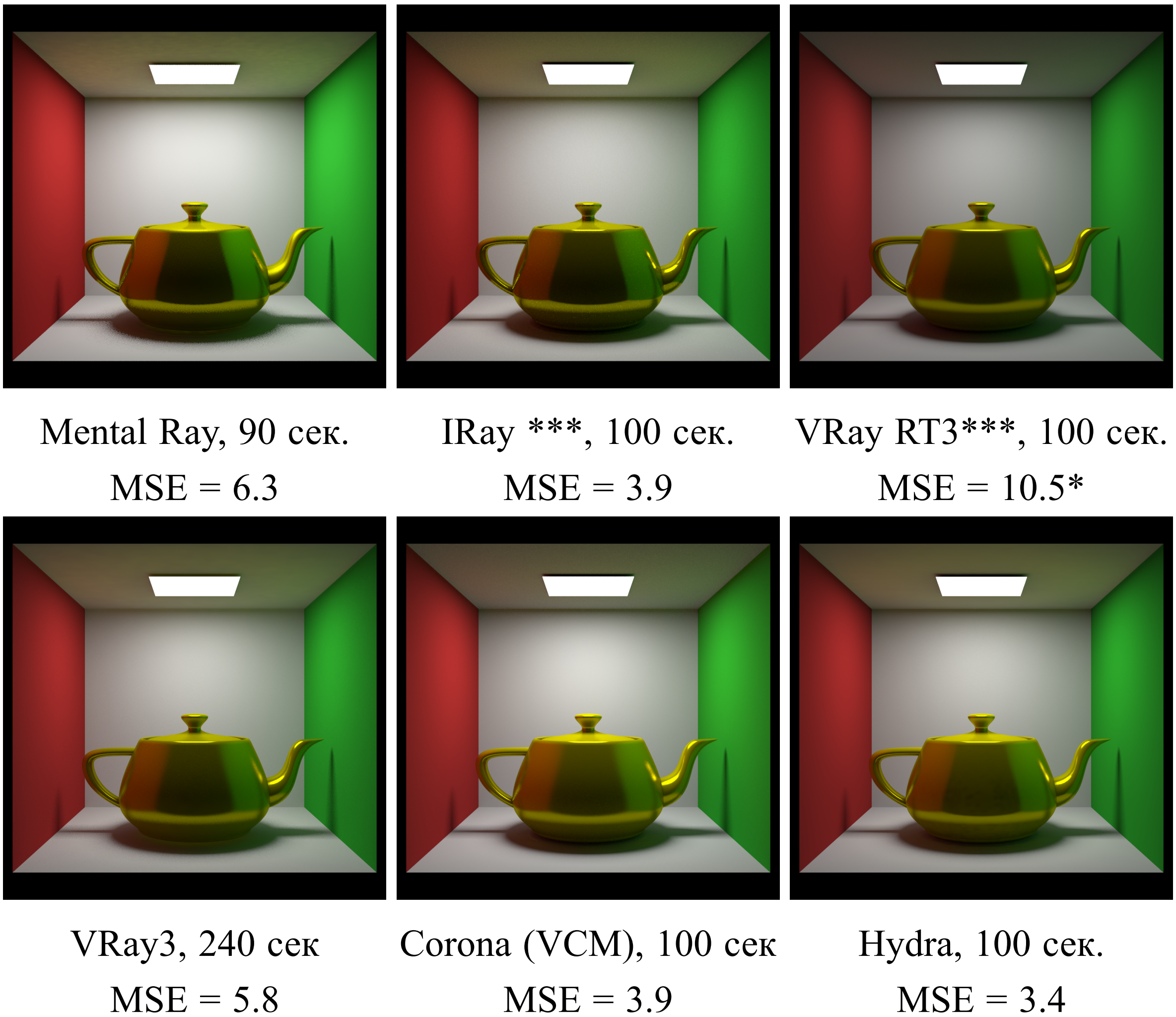


Рис. А.6. Сравнение на сцене 'Cornell Box'. Время рендера $\approx$ 100 секунд. Расчет выполнен при помощи кэша освещенности и трассировки путей. Каустики Включены. * - система VRayRT не способна рассчитывать каустики. Отсутствие каустика засчитано за ошибку. *** - в системе не реализован метод кэширования освещенности или аналогичные методы, вследствии чего использовалась Монте-Карло трассировка.

A.2. Открытая сцена ('Outdoor')

Данная сцена, являясь геометрически сложной (из-за травы), тем не менее, с точки зрения освещения достаточно проста - однородная панорама и один относительно неяркий источник освещения. Время синтеза изображения на такой сцене должно в основном быть обусловлено скоростью трассировки лучей и наличием/отсутствием специализированного сэмлера панорамм.

Mental Ray, 360 сек.
MSE = 3.3

IRay*, 120 сек.
MSE = 2.5

VRay RT3, 60 сек.
MSE = 2.5

VRay3, 240 сек
MSE = 5.5

Corona, 60 сек
MSE = 2.7

Hydra, 60 сек.
MSE = 2.7

Рис. А.7. Сравнение на сцене 'Outdoor'. MSE $\approx$ 2.5.

Mental Ray, 145 сек.
MSE = 7.7

IRay, 120 сек.
MSE = 2.52

VRay RT3, 120 сек.
MSE = 1.76

VRay3, 127 сек
MSE = 7.0

Corona, 120 сек
MSE = 2.3

Hydra, 120 сек.
MSE = 2.0

Рис. А.8. Сравнение на сцене ’Outdoor’. Время рендера $\approx$ 2 минуты.

А.3. Crytek Sponza

Оригинальная сцена ’Crytek Sponza’ содержит текстуры в различных каналах материалов. Для того чтобы форма освещения и возникающие артефакты были лучше заметны (как для человеческого глаза так и для численного сравнения), текстуры были удалены. В данном сценарии практически всё видимое освещение вторичное и силен вклад от второго и третьего диффузных переотражений, поэтому финальный сбор должен давать значительное ускорение. Данный сценарий наиболее точно отражает скорость расчета вторичного освещения. Сценарий нельзя назвать ’сложным для расчета’ (hard sampling), поскольку яркость солнца относительно невелика и солнцем освещена значительная часть поверхности сцены.

А.3.1. Кэш освещенности + FG (где возможно)

Mental Ray, 960 сек. MSE = 3.3 | IRay, 1400 сек* MSE = 3.8 | VRay RT3, 600 сек* MSE = 3.8

VRay3, 180 сек MSE = 3.8 | Corona, 600 сек MSE = 3.8 | Hydra, 60 сек. MSE = 3.8

Рис. А.9. Сравнение на сцене ’Crytek Sponza’. 1024x1024. MSE $\approx$ 4.0. Метод VCM для системы Corona не улучшил результат. * - в системе не реализован метод кэширования освещенности или аналогичные методы, вследствии чего использовалась Монте-Карло трассировка.



Mental Ray, 60 сек. MSE = 16.4 | IRay, 60 сек* MSE = 27.7 | VRay RT3, 60 сек* MSE = 21.6



VRay3, 60 сек MSE = 7.3 | Corona, 60 сек MSE = 12.6 | Hydra, 60 сек. MSE = 3.78

Рис. A.10. Сравнение на сцене 'Crytek Sponza'. 1024x1024. Время расчета $\approx$ 60 секунд. Метод VCM для системы Corona не улучшил результат. * - в системе не реализован метод кэширования освещенности или аналогичные методы, вследствии чего использовалась Монте-Карло трассировка.

A.4. Конференц-зал

Сложность расчета освещения в данной сцене заключается в большом числе источников освещения. Под потолком находится 20 источников прожекторного типа. Тем не менее, каждый из источников светит на относительно небольшой участок сцены, поэтому, если в рендер-системе реализована эффективная схема сэмплирования источников, в каждой конкретной точке сцены большинство источников должны быть эффективно отброшены или учитываться относительно нечасто.

A.4.1. Монте-Карло трассировка

Mental Ray, 3000 сек.
MSE = 4.8

IRay, 600 сек
MSE = 4.8

VRay RT3, 600 сек
MSE = 5.0

VRay3, 1000 сек
MSE = 4.8

Corona, 180 сек
MSE = 5.0

Hydra, 100 сек.
MSE = 4.8

Рис. А.11. Сравнение на сцене 'Конференц-зал'. 1024x1024. MSE $\approx$ 5.0.

Mental Ray, 90 сек.
MSE = 24.2

IRay, 60 сек
MSE = 16.0

VRay RT3, 60 сек
MSE = 18.0

VRay3, 60 сек
MSE = 30.0

Corona, 60 сек
MSE = 9.52

Hydra, 60 сек.
MSE = 6.3

Рис. А.12. Сравнение на сцене 'Конференц-зал'. 1024x1024. Время рендера ≈ 60 сек.

A.4.2. Кэш освещенности

Mental Ray, 540 сек.
MSE = 7.6

IRay, 600 сек*
MSE = 4.8

VRay RT3, 600 сек*
MSE = 4.5

VRay3, 300 сек
MSE = 6.3

Corona, 180 сек**
MSE = 5.0

Hydra, 60 сек.
MSE = 4.5

Рис. A.13. Сравнение на сцене 'Конференц-зал'. 1024x1024. MSE ≈ 5.0. ** - кэш освещенности в короне используется для апроксимации светимости (как карта светимости). * - в системе не реализован метод кэширования освещенности или аналогичные методы, вследствии чего использовалась Монте-Карло трассировка.

Mental Ray, 120 сек.
MSE = 18.9

IRay, 60 сек*
MSE = 16.0

VRay RT3, 60 сек*
MSE = 18.0

VRay3, 120 сек
MSE = 8.5

Corona, 60 сек**
MSE = 15.1

Hydra, 60 сек.
MSE = 4.5

Рис. А.14. Сравнение на сцене 'Конференц-зал'. 1024x1024. Время рендера $\approx$ 60 сек. ** - кэш освещенности в короне используется для апроксимации светимости (как карта светимости). * - в системе не реализован метод кэширования освещенности или аналогичные методы, вследствии чего использовалась Монте-Карло трассировка.

А.5. Освещение ’Небесными Порталами’ (Sky Portals)

Данная сцена демонстрирует достаточно типичный сценарий расчета дневного освещения в помещении. При таком сценарии напротив окон ставятся источники света, имитирующие свет от внешнего окружения, проникающий через окно. Такой источник называется ’Небесным Порталом’ (Sky Portal). При правильной реализации ’Небесный Портал’ является полностью корректным решением. Такой источник света является средством расчета освещения от окружения при помощи явной стратегии (стратегии сэмплирования источников света). Иными словами, ’Небесный Портал’ является не самостоятельным источник света, а всего лишь подсказкой для Монте-Карло трассировки лучей, позволяющей вычислять освещение от окружения более эффективно в тех случаях когда изнутри помещения видима относительно-небольшая часть окружения. Тем не менее, при использовании небесных порталов расчет первичного освещения в определенной степени осложняется по 2 причинам:

1. Небесные порталы могут иметь значительные размеры, в результате чего замедляется расчет мягких теней.

2. Число источников этого типа может быть достаточно большим (5-10), что еще более осложняет вычисление первичного освещения.

На данной сцене были использованы различные комбинации методов для различных систем, наилучшим образом показавшие себя. Например, интересно отметить, что поскольку первичное освещение досточно шумное, метод финального сбора, использующий апроксимацию первичного освещения из фотонной карты в системе Hydra на данной сцене работает в 5 раз быстрее стандартной Монте-Карло трассировки. Интересно дополнительно отметить, что для системы Corona карта светимости и финальный сбор дают выигрыш всего лишь в 2 раза. Учитывая время рендера, это можно объяснить более эффективной схемой сэмплирования источников типа ’Небесный портал’, что позволяет выиграть время на любой глубине Монте-Карло трассировки пути. Также подобная разница может быть вызвана падением производительности трассировки лучей на GPU в системе Hydra на глубоких уровнях Монте-Карло трассировки.

Таким Образом:

- Для системы Hydra был использован финальный сбор без кэша освещенности (т.е. попиксельный финальный сбор).
- Для системы Mental Ray был использован метод финального сбора в сочетании с кэшем освещенности.
- Для VRay - карты светимости (называемые в VRay термином 'Light Cache') в сочетании с кэшем освещенности.
- Для Corona - кэш освещенности для 3 переотражения и выше (как карта светимости), попиксельный финальный сбор для 2 переотражения. Такой метод аналогичен попиксельному финальному сбору, который был использован в системе Hydra.
- Для VRayRT и IRay - Монте-Карло трассировка, поскольку другие методы в них не реализованы.

Mental Ray, 330 сек.
MSE = 5.5

IRay, 480 сек
MSE = 5.3

VRay RT3, 900 сек
MSE = 5.0

VRay3, 240 сек
MSE = 5.8

Corona, 80 сек.
MSE = 4.8

Hydra, 120 сек.
MSE = 5.0

Рис. А.15. Сравнение для источников типа ’Sky-Portal’. 1024x1024. MSE $\approx$ 5.0.

Mental Ray, 110 сек.
MSE = 11.6

IRay, 120 сек
MSE = 11.1

VRay RT3, 120 сек
MSE = 15.1

VRay3, 180 сек
MSE = 7.3

Corona, 120 сек.
MSE = 4.0

Hydra, 120 сек.
MSE = 5.0

Рис. А.16. Сравнение для источников типа ’Sky-Portal’. 1024x1024. Время рендера $\approx$ 2 минуты.

А.6. Тест на MLT

В данном сценарии небольшой участок сцены освещается исключительно ярким направленным источником света, имитирующим солнце. Вторичное освещение, вызванное таким освещением является трудным для расчета. Фотонные карты в сочетании с финальным сбором (как и карты светимости) амортизируют увеличение времени расчета только за счет ускорения вычисления компоненты от третьего и более переотражений. Однако, вычисление компоненты от второго переотражения света данным методом не ускоряется (в системе Hydra финальный сбор дал ускорение в 3 раза по сравнению с обыкновенной Монте-Карло трассировкой). С другой стороны, алгоритм переноса света Метрополиса (MLT) и аналогичные алгоритмы при условии реализации их в той или иной системе должны давать на данной сцене значительное ускорение по сравнению с традиционным методом Монте-Карло и финальным сбором. Интересно отметить, что алгоритм соединения вершин в системе Corona, аналогичный двунаправленной трассировке путей на данном сценарии не дал выигрыша в скорости, а напротив, в 10 раз проиграл стандартному режиму с финальным сбором. Это можно объяснить тем фактом, что большое число световых путей сохраняет вершины на невидимых участках сцены - на крыше и наружных стенах здания. Такие вершины вынуждают трассировщик соединять множество путей заведомо не вносящих вклад в изображение, что приводит к избыточным вычислениям. Существующие методы решени данной проблемы в системе Corona по-видимому не реализованы. Что касается системы IRay, то по-видимому корректно алгоритм MLT в ней не реализован, т.к. при его явном включении (architectural sampler) время расчета не уменьшилось.

Mental Ray, 900 сек.
MSE = 15.1

IRay, 3600 сек
MSE = 5.0

VRay RT3, 4800 сек
MSE = 4.8

VRay3, 1000 сек
MSE = 5.5

Corona, 600 сек.
MSE = 5.0

Hydra, 720 сек.
MSE = 5.0

Рис. А.17. Сравнение в условиях сложного диффузного освещения. 1024x1024. MSE $\approx$ 5.0.

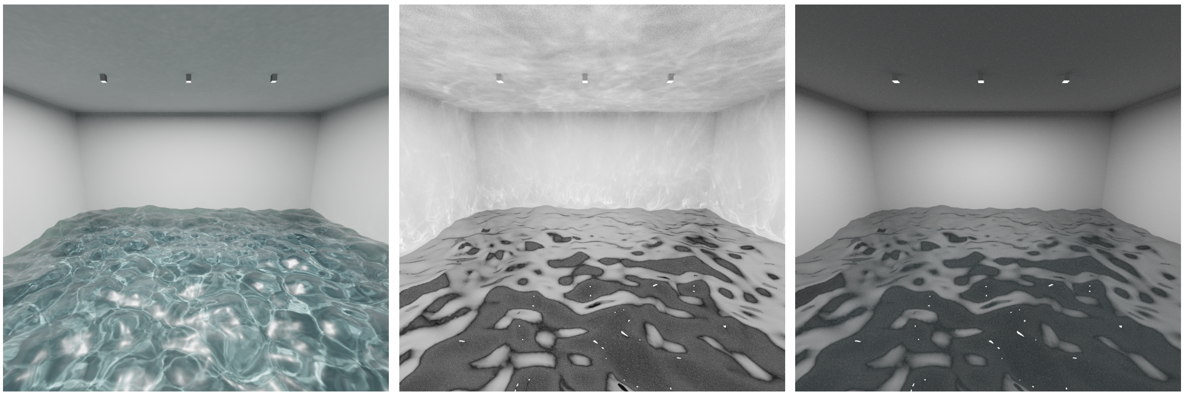
Mental Ray, 300 сек. MSE = 25.0 | IRay, 300 сек MSE = 24.2 | VRay RT3, 300 сек MSE = 26.0

VRay3, 480 сек MSE = 8.3 | Corona, 300 сек MSE = 8.6 | Hydra, 300 сек MSE = 8.3

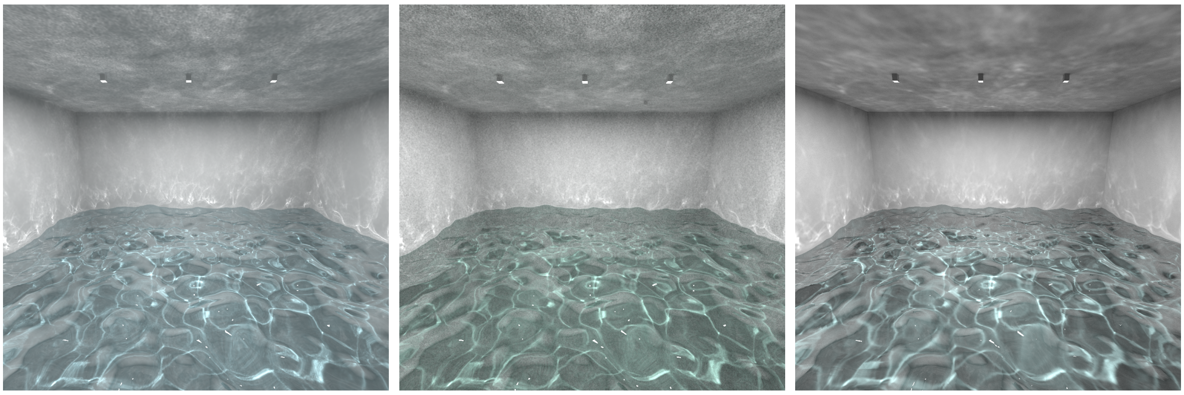
Рис. А.18. Сравнение в условиях сложного диффузного освещения. 1024x1024. Время рендера $\approx$ 5 минут.

А.7. Каустики

В данной сцене присутствуют отраженные каустики и подводные каустики SDS типа (Specular Diffuse Specular), являющиеся сложными для расчета (hard sampling). Свойства поверхности воды были настроены таким образом, чтобы создавать как подводные так и надводные каустики. К сожалению, для систем Octane и Mental Ray не удалось добиться совпадения яркости отраженных каустик с целевым (по выше описанному плану) изображением. Сделаем послабление для этих систем и будем считать, что сложность всех изображений приблизительно одинакова. Для системы IRay напротив, пришлось сделать поверхность воды зеркальной, поскольку IRay не может эффективно вычислять каустики SDS типа. Это подтверждается наличием импульсного шума на поверхности воды, который представляет из себя каустики, видимые в отражениях. В графиках сравнения индекс производительности системы IRay был разделен на 2 в качестве штрафа за невозможность вычисления корректного изображения. Система VRayRT, не способная к эффективному вычислению каустиков получила 0 баллов на данной сцене.



Mental Ray, 180 сек. | IRay, 60 сек | VRay RT3, 60 сек
MSE = 8.0 | MSE = 11.6 | MSE = 100.0



VRay3, 120 сек | Corona, 60 сек | Hydra, 60 сек
MSE = 7.8 | MSE = 10.5 | MSE = 5.6

Рис. А.19. Сравнение на каустиках. 1024x1024. Время рендера $\approx$ 1 минута.

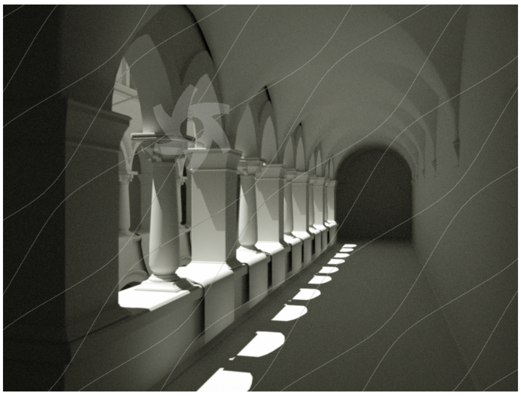
А.8. Сравнение с Octane

Из-за наличия ограничения на максимальное разрешение и водяных знаков в демо-версии системы Octane, сравнение с ней проводилось отдельно. Результаты (время) были позже отмасштабированы исходя из соотношений разрешений 800x600 и 1024x1024 и добавлены в общие графики сравнения. На рисунке A.21 в скобках указано отмасштабированное время.

Интересно отметить, что метод PMC в системе Octane на сценарии Cry Sponza не дал ускорения по сравнению с традиционной Монте-Карло трассировкой путей. На сценарии ’MLT Like’ PMC также не дал ускорения, а напротив был в 3.3 раза медленне стандартной Монте-Карло трассировки. Однако на каустиках метод PMC дает значительный выигрыш по сравнению с традиционной Монте-Карло трассировкой путей.

Cornell Box (PMC), 60 сек (130 сек)
MSE = 5.3

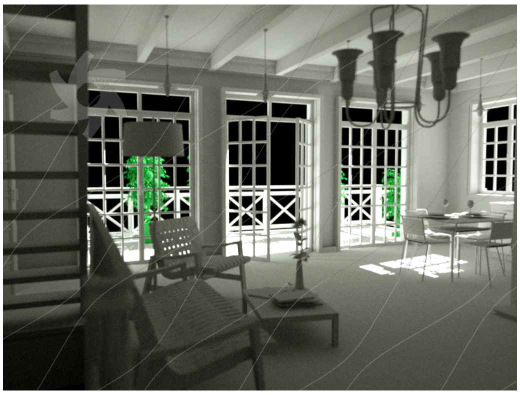
Outdoor, 120 сек (260 сек)
MSE = 2.8



Cry Sponza, 600 сек (1300 сек)
MSE = 5.3

Conference, 600 сек (1300 сек)
MSE = 5.45

Sky Portal, 300 сек (650 сек)
MSE = 4.4



MLT Like, 600 сек (1300 сек)
MSE = 6.6

Рис. А.20. Синтезированные в системе Octane изображения. MSE $\approx$ 4.0. 800x600. В скобках указано время для разрешения 1024x1024.

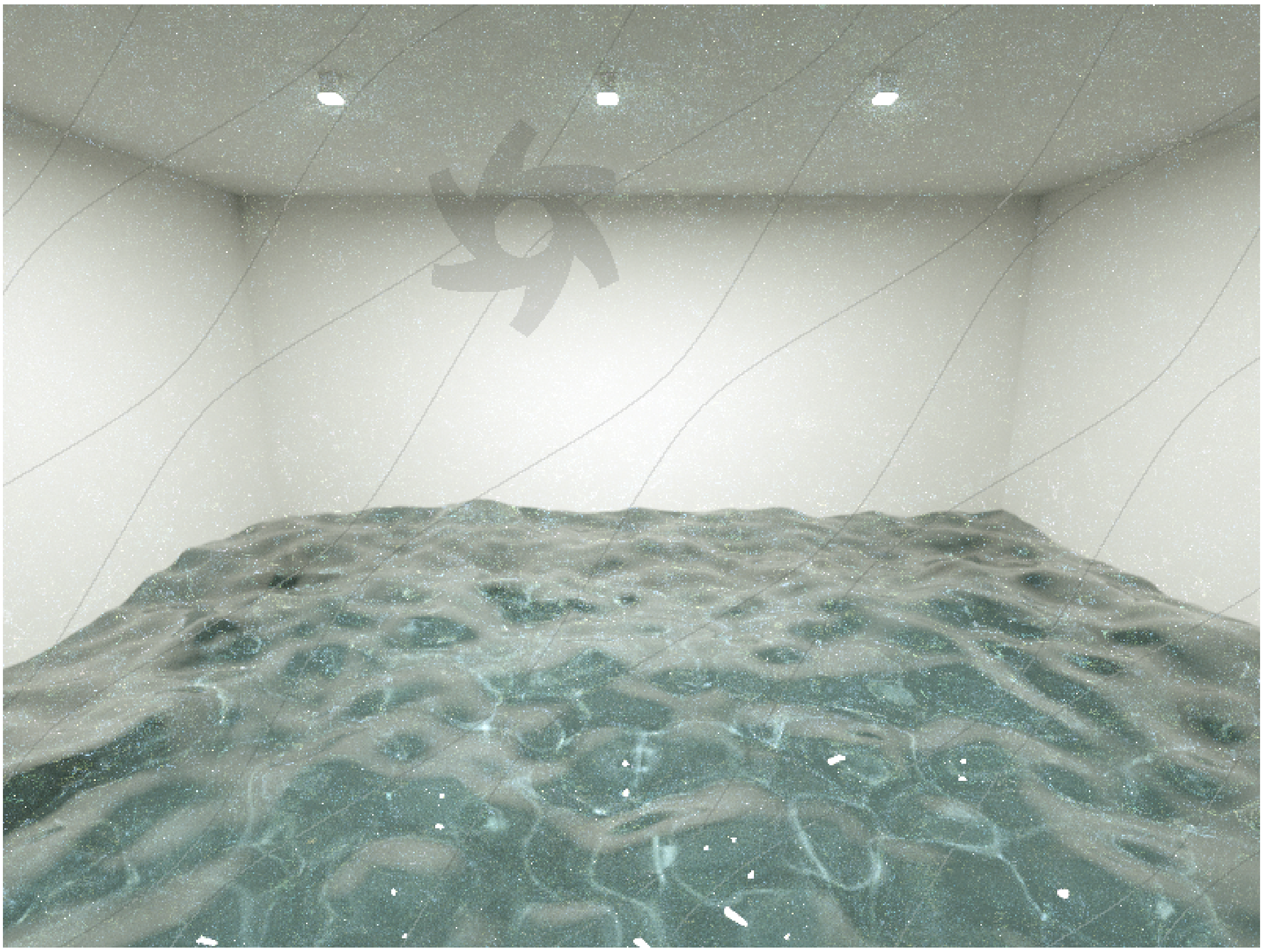


Рис. А.21. Синтезированные в системе Octane изображение с каустиками. MSE $\approx$ 15.0. Время рендера - 5 минут (11 минут). В скобках указано время для разрешения 1024x1024.

А.9. Анализ производительности

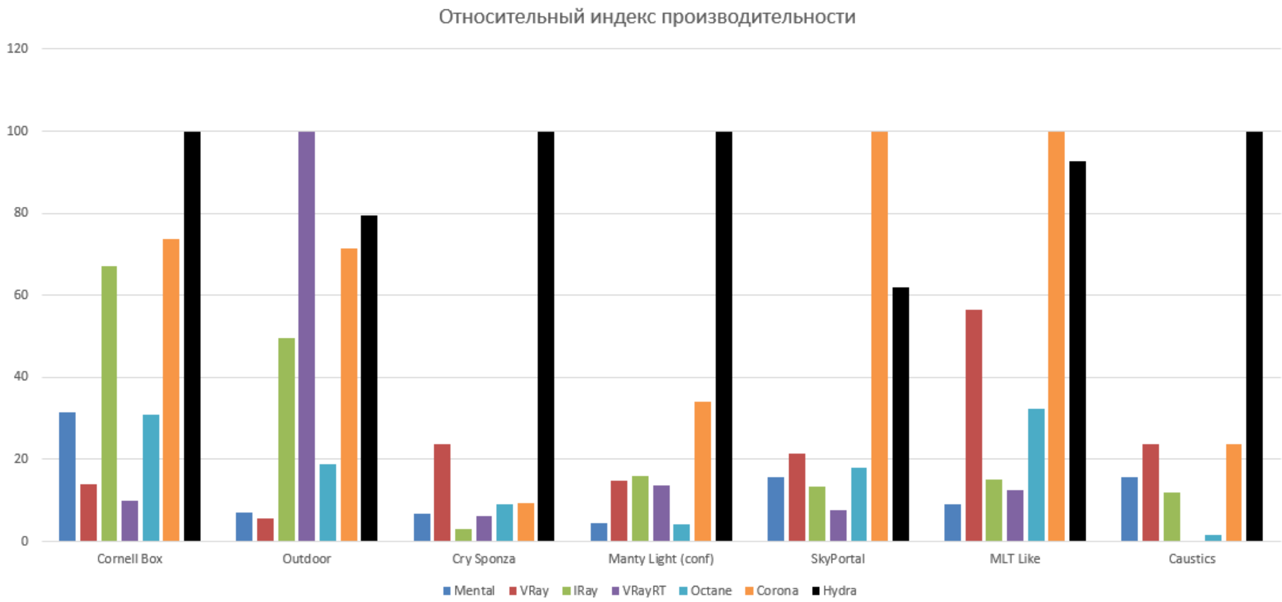


Рис. А.22. Сравнение рендер-систем по относительному индексу производительности (на различных сценах).

Рис. А.23. Сравнение рендер-систем по относительному индексу производительности (среднее по всем сценам).