Моделирование каустик в реальном времени на графическом процессоре с помощью OpenCL

Дмитрий Сопин1, Денис Боголепов1, Максим Захаров2, Олег Блохин1

1Нижегородский государственный университет им. Лобачевского

2Нижегородский государственный технический университет им. Алексеева

sopindm@gmail.com, denisbogol@gmail.com, maxim.zakharov@inbox.ru, blohin.o.d@gmail.com

**Аннотация**

The goal of this work is adaptation of the photon mapping method for caustics modeling in real time. The fast algorithm capable of running entirely on GPU and utilizing both OpenGL and OpenCL is proposed. Building of the photon map and ray tracing are performed using OpenGL shaders. To improve rendering we use grid-based acceleration structure that is constructed directly on the GPU using OpenCL. Performance estimation is presented.

***Keywords****: Interactive Rendering, GPU Ray Tracing, GPU Photon Mapping, Caustics, GPGPU, OpenGL, OpenCL.*

# ВВЕДЕНИЕ

Эффекты *глобального освещения* играют важную роль в задачах синтеза реалистичных изображений. К базовым эффектам можно отнести *тени*, многократные *отражения* и *преломления*. Для их моделирования в большинстве случаев применяется алгоритм *трассировки лучей* [1]. Расширенные эффекты, связанные с *диффузным отражением* света и *каустиками*, требуют более сложных алгоритмов. К числу наиболее распространенных относится метод *фотонных карт*, который обеспечивает практическое и вычислительно эффективное решение для задач глобального освещения [2]. Метод фотонных карт состоит из *двух* проходов трассировки лучей. На первом проходе выполняется прямая трассировка, в процессе которой из источников света испускаются фотоны, а информация об их взаимодействиях с объектами сцены записывается в специальную структуру данных – фотонную карту. На втором проходе выполняется обратная трассировка, при этом в процессе расчета освещенности точки соударения используется информация о ближайших фотонах из ее окрестности, доступная посредством обращения к фотонной карте. Алгоритмы глобального освещения и метод фотонных карт в частности традиционно полагаются на сложные программные реализации и статическую визуализацию. Тем не менее, появление производительной программируемой графической аппаратуры позволило сделать большой шаг в этом направлении.

Одна из первых успешных реализаций метода фотонных карт на графическом процессоре описана в работе [3]. Авторы отказались от использования традиционного -дерева для ускорения доступа к фотонной карте. Вместо этого, в работе используется регулярная сетка, и предлагаются два способа распределения фотонов по вокселям. Кроме того, авторами был предложенметод поиска ближайших фотонов сетки для заданной точки соударения. Данный метод оказался весьма ресурсоемким и занимал порядка 90% от времени визуализации. Безусловно, используемая на тот момент аппаратура класса NVIDIA GeForce FX 5900 Ultra не могла обеспечить интерактивной визуализации даже для самых простых сцен. Например, сцена с одним прозрачным объектом в окне 512 × 512 точек для 65 × 103 фотонов обрабатывалась более одной минуты. Тем не менее, данный подход характеризуется высокой трудоемкостью и вряд ли обеспечит реальное время для сцен средней сложности даже на современном оборудовании. Хотя производительность уступала существующим реализациям для центрального процессора, работа была крайне интересной и послужила основой для дальнейших улучшений. К их числу относятся работы [4]-[5], в которых авторы исследовали различные ускоряющие структуры, включая -деревья и иерархию ограничивающих объемов. Однако применение современного оборудования не позволило достичь реального времени даже на простых тестовых сценах. Например, в работе [5] на простых сценах в окне 512 × 512 точек для 128 × 103 фотонов получена производительность порядка 4-6 *к*/*с* на графическом процессоре NVIDIA GeForce GTX 280.

Целью настоящей работы является реализация *упрощенного* метода фотонных карт для исполнения на графическом процессоре в *реальном* масштабе времени. В данной работе показано, каким образом можно дополнить трассировку лучей для корректного моделирования отражающих и прозрачных объектов, которые вызывают каустики – области с резко возрастающей интенсивностью светового поля. Основой программной реализации служат инструменты OpenGL и OpenCL, которые являются стандартами в области графики и гетерогенных вычислений и обеспечивают поддержку оборудования различных производителей.

# МОДЕЛИРОВАНИЕ КАУСТИК НА ГПУ

## Простейший подход

Для моделирования каустик может использоваться простой двухпроходный алгоритм. На *первом* проходе в соответствии с расположением и ориентацией источников света генерируются фотоны, число которых определяется заранее. В простейшем случае можно рассматривать *точечные* (испускают фотоны равномерно по всем направлениям) или *прямоугольные* (испускают фотоны в произвольном направлении полупространства) источники света.

**Рисунок 1**. Генерация фотонов для регулярных  
точек на прямоугольной площадке

В данной работе использовался точеный источник, однако для выделения значимых областей сцены выбирались только направления, проходящие через заданную прямоугольную площадку. Вместо использования генератора случайных чисел площадка покрывалась равномерной сеткой, для каждого узла которой генерировался фотон. При таком подходе размерность сетки определяет общее число фотонов, а фотонная карта может быть легко представлена в виде текстуры, каждый тексель которой соответствует точке регулярной сетки.

Для каждого фотона прослеживается траектория его движения и взаимодействия с объектами сцены. При соударении с *диффузным* объектом в фотонную карту записываются координаты найденной точки соударения. При соударении с объектами, обладающими *отражающими* или *прозрачными* свойствами, отслеживаются дальнейшие взаимодействия в направлении отражения или преломления соответственно. Для фотонов, покинувших сцену без взаимодействия с объектами, записываются недопустимые координаты. Интенсивность всех фотонов при этом считается одинаковой и равной параметру . Подход может быть обобщен для учета падения интенсивности фотона при соударении с *неидеальными* отражающими или прозрачными объектами (для этого необходимо сохранять не только координаты точки соударения, но и интенсивность на момент данного соударения) или для моделирования *хроматических аберраций* (необходимо генерировать фотоны отдельно для каждой длины волны).

При *втором* проходе используется традиционный алгоритм трассировки лучей. При этом к вычисленной в точке соударения освещённости прибавляется интенсивность фотонов из некоторой окрестности, которые доступны посредством обращения к сформированной фотонной карте. В данной работе принимается, что на освещенность точки влияют *все* фотоны из ее окрестности вместо ближайших, что в значительной степени упрощает этап сбора. Кроме того, принимается тривиальное соображение, что влияние фотона уменьшается с увеличением расстояния до точки соударения. В простейшем случае можно воспользоваться *линейной* весовой функцией, что приводит к следующей формуле:

|  |  |
| --- | --- |
|  | **(1)** |

**Здесь – координаты обрабатываемой точки соударения, – координаты -ого элемента фотонной карты. Параметр – радиус окрестности сбора фотонов, который** подбирается **эмпирически для каждой конкретной сцены.**

**Сбор всех фотонов из фиксированной окрестности** вместо ближайших приводит к худшей оценке освещенности в областях с малой плотностью фотонов или размытому результату в областях с высокой плотностью. В качестве альтернативной весовой функции можно использовать интерполяцию Эрмита вида , отдавая больший приоритет близким фотонам и меньший – более отдаленным (в сравнении с линейной функцией). Данный вид интерполяции реализован в ГПУ аппаратно и доступен через встроенную функцию smoothstep языка шейдеров GLSL [6]. С использованием данной функции формула (1) принимает вид:

|  |  |
| --- | --- |
|  | **(2)** |

**Применение модифицированной формулы (2) позволяет придать каустикам дополнительную резкость.**

## Сортировка фотонов

Описанный подход малоэффективен, поскольку для расчета освещённости в каждой точке соударения требует перебора *всех* элементов фотонной карты. Для повышения производительности можно выполнить предварительную сортировку фононной карты по координатам фотонов в лексикографическом порядке. При этом вычисления на втором проходе сокращаются до перебора всех фотонов из *непрерывного* сегмента карты, заключенного между фотонами с *минимальной* и *максимальной* координатой в кубической **-**окрестности точки соударения:

|  |  |
| --- | --- |
|  | **(3)** |

Здесь и – индексы фотонов с минимальными и максимальными координатами соответственно. Сортировка может быть выполнена как на центральном, так и на графическом процессоре. В первом случае целесообразно использовать метод быстрой сортировки со сложностью , во втором – ***поразрядную* [7] или *битоническую* [8] сортировку с трудоемкостью и соответственно. В данной работе использовалась реализация битонической сортировки для ГПУ.**

Описанный подход позволяет значительно повысить производительность визуализации, однако также не достаточно эффективен: сбор фотонов осуществляется не из **-**окрестности точки соударения, а из **-**полосы (множество точек вида ). Далее будет рассмотрен алгоритм, позволяющий сократить перебор до некоторой окрестности точки соударения.

## Воксельная сортировка

**Разделим пространство, в котором расположена сцена, на *воксели* со стороной (радиус окрестности для сбора фотонов). Тогда каждому фотону кроме его координат можно сопоставить координаты вокселя, в котором он находится. Для этого можно использовать следующую формулу:**

|  |  |
| --- | --- |
| **, ,** | **(4)** |

**В случае сцены конечного размера (как это обычно и бывает в практических задачах) данная формула имеет вид:**

|  |  |
| --- | --- |
| **, ,** | **(5)** |

**Здесь через , , обозначена *минимальная* точка ограничивающего параллелепипеда сцены. Таким образом, в случае конечной сцены координаты вокселей принимают ограниченные положительные значения. Данное замечание будет использовано в дальнейшем.**

*0*

*2*

*3*

*4*

*5*

*5*

*P1X*

*P1Y*

*P1Z*

*P2X*

*P2Y*

*P2Z*

*P3X*

*P3Y*

*P3Z*

*P4X*

*P4Y*

*P4Z*

*P5X*

*P5Y*

*P5Z*

*P6X*

*P6Y*

*P6Z*

*P7X*

*P7Y*

*P7Z*

*P8X*

*P8Y*

*P8Z*

...

*P0X*

*P0Y*

*P0Z*

*Массив фотонов*

...

*Массив вокселей*

**Рисунок 2**. Ускоряющая структура для  
доступа к фотонной карте

**После формирования фотонной карты описанным выше способом все фотоны сортируются в лексикографическом порядке по координатамсоответствующихвокселей. При таком способе сортировки фотоны, принадлежащие одному вокселю, считаются равными.**

**Для определения фотонов, содержащихся в -окрестности заданной точки, используется следующее соображение: указанные фотоны принадлежат вокселям, координаты которых отличаются не более чем на 1 от координат вокселя для заданной точки (в противном случае разность координат составляет не менее длины вокселя, которая равна ). То есть для вычисления освещённости в конкретной точке достаточно произвести суммирование по всем фотонам из 27 вокселей. Конечная формула будет выглядеть следующим образом:**

**,**

**где – координаты вокселя, соответствующего рассматриваемой точки, и – индексы соответственно первого и последнего фотона, лежащего в вокселе с координатами , ,  (могут быть найдены линейным поиском по фотонной карте или предложенным далее способом).**

## Ускорение поиска

**Множество вокселей, на которые разбита сцена, очевидно, является дискретным и, в случае конечных размеров сцены, конечным. Поэтому возможно вычислить индексы первого и последнего фотонов в каждом вокселе один раз вместо многократного вычисления для каждой точки. В нашей работе величина бралась порядка 1% от размеров сцены, что порождает множество вокселей размером 100 × 100 × 100. Таким образом создание ускоряющей структуры, содержащей индекс первого и последнего фотона в каждом вокселе (или невозможные значения в случае их отсутствия), реально. Для её заполнения можно использовать следующий параллельный алгоритм.**

**Для каждого элемента отсортированной карты фотонов проверяется, является ли он первым элементом в своём вокселе (т.е. не лежит ли предыдущий в другом). В случае положительного результата номер элемента записывается в ускоряющую структуру в качестве номера первого элемента. Аналогично поступаем с последним элементом. Т.к. в каждом вокселе существуют единственные первый и последний элемент, ни каких коллизий при параллельном выполнении не возникнет.**

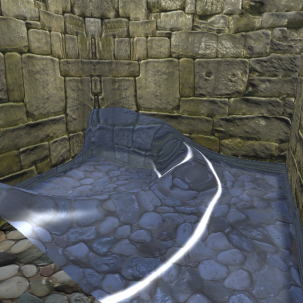
## Результат

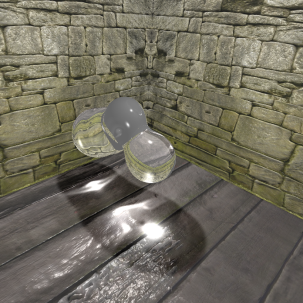
**В результате получаем следующий четырёхпроходный алгоритм:**

1. **Генерация фотонной карты. Данный этап имеет сложность , где – число генерируемых фотонов.**
2. **Воксельная сортировка фотонной карты. Сложность определяется используемым алгоритмом сортировки. Возможны следующие варианты: быстрая сортировка (*quick sort*) имеет сложность , поразрядная сортировка (*radix sort*) имеет сложность , битонная сортировка (*bitonic sort*) имеет сложность . Последние два варианта могут быть успешно реализованы на графическом процессоре.**
3. **Заполнение ускоряющей структуры фотонной карты. Данный этап имеет линейную сложность .**
4. **Трассировка лучей с учётом фотонной карты. Имеет сложность , где – число трассируемых лучей, – максимальное количество фотонов в -окрестности точки (в общем случае может быть принято за константу).**

**Таким образом, мы получили алгоритм реализации метода фотонных карт, имеющий сложность , где – число генерируемых фотонов, – число трассируемых лучей.**

# ОЦЕНКА ПРОИЗВОДИТЕЛЬНОСТИ

Для проведения эксперимента использовался графический ускоритель AMD Radeon 4850 и центральный процессор AMD Phenom II X3 710. Размер окна визуализации составлял 512 × 512 точек, значение параметра выбиралось равным ~1,5% от размера сцены.

В следующей таблице представлены результаты замеров производительности.

**Таблица 1**. Результаты замеров производительности

|  |  |  |
| --- | --- | --- |
| *Размер фотонной карты* | *Время загрузки и выгрузки текстуры (мс)* | *Время сортировки фотонной карты (мс)* |
| *64* × *64* | *1* | *1* |
| *128* × *128* | *3* | *5* |
| *256* × *256* | *7* | *22* |
| *Размер фотонной карты* | *Простейший подход (к/с) для глубины трассировки 1/2/3/4* | *Улучшенный подход (к/с) для глубины трассировки 1/2/3/4* |
| *64* × *64* | *9/6/5/4* | *52/40/32/26* |
| *128* × *128* | *2/1/0/0* | *24/18/15/12* |
| *256* × *256* | *0/0/0/0* | *15/11/8/4* |

# ЗАКЛЮЧЕНИЕ

Предложен упрощенный вариант метода фотонных карт для исполнения на графическом процессоре в реальном масштабе времени. Данный метод позволяет корректно моделировать кустики – области с резко возрастающей интенсивностью светового поля, возникающие при взаимодействии света с прозрачными или отражающими объектами сцены. Для повышения эффективности визуализации предложенный алгоритм выполняет предварительную сортировку фотонной карты на центральном процессоре. Данный подход обеспечивает приемлемую скорость работы при использовании фотонных карт размера 256 × 256 элементов.

# **ЛИТЕРАТУРА**

1. Holly Rushmeier, David Banks, Peter Shirley. “*A Basic Guide to Global Illumination*”. SIGGRAPH 98 Course 5.
2. Henrik Wann Jensen*.* “***Realistic Image Synthesis Using* *Photon Mapping*”***.* AK Peters. Ltd., Massachusetts. 2001.
3. Timothy J. Purcell, Craig Donner, Mike Cammarano, Henrik Wann Jensen and Pat Hanrahan. “*Photon mapping on programmable graphics hardware*”*.* Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference, 2003.
4. Martin Fleisz. “*Photon Mapping on the GPU*”. Master's thesis, School of Informatics, University of Edinburgh, 2009.
5. B. Fabianowski and J. Dingliana. “*Interactive Global Photon Mapping*”. Eurographics Symposium on Rendering, Volume 28 (2009), Number 4.
6. John Kessenich, Dave Baldwin, Randi Rost. “*The OpenGL Shading Language*” (Revision 8, 7 September 2006).

<http://www.opengl.org/registry/doc/GLSLangSpec.Full.1.20.8.pdf>

1. “*Radix sort*” (From Wikipedia, the free encyclopedia).

<http://en.wikipedia.org/wiki/Radix_sort>

1. “*Bitonic sorter*” (From Wikipedia, the free encyclopedia).

<http://en.wikipedia.org/wiki/Bitonic_sorter>