Моделирование каустик в реальном времени

Денис Боголепов, Дмитрий Сопин, Вадим Турлапов
Нижегородский государственный университет им. Лобачевского
Факультет вычислительной математики и кибернетики, Нижний Новгород, Россия
{denisbogol, sopindm, vadim.turlapov}@gmail.com

АННОТАЦИЯ

Целью данной работы является адаптация метода фотонных карт для моделирования каустик в реальном времени. В работе предлагается быстрый алгоритм, который целиком выполняется на ГПУ и реализован на основе инструментов OpenGL и OpenCL. Для построения фотонной карты и визуализации сцены используются шейдеры OpenGL. Для увеличения производительности визуализации разработана основанная на вокселях ускоряющая структура, которая конструируется на ГПУ средствами OpenCL. Приводятся оценки производительности.

Ключевые слова: *Интерактивная визуализация на ГПУ, трассировка лучей, метод фотонных карт, моделирование каустик, GPGPU, OpenGL, GLSL, OpenCL.*

1. 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]. 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 направлении интерактивного применения данных методов.

Одна из первых успешных реализаций метода фотонных карт на графическом процессоре описана в работе [3]. Авторы отказались от использования традиционного *kd*-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 предложен метод поиска *k* 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 × 10^3 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 *kd*-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 × 10^3 фотонов получена производительность порядка 4–6 к/с на ГПУ 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.

2. МОДЕЛИРОВАНИЕ КАУСТИК

2.1 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.

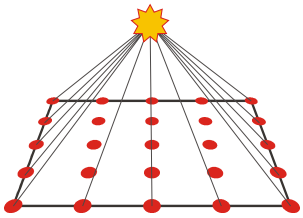

Рис. 1. Генерация фотонов для регулярных точек на прямоугольной площадке

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

N, 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 направлениях отражения или преломления соответственно. Для фотонов, покинувших сцену без взаимодействия с объектами, записываются *недопустимые* координаты. Интенсивность всех фотонов считается одинаковой и равной параметру I_{photon}. Подход может быть обобщен для учета уменьшения интенсивности фотона при соударении с *неидеальными* отражающими или прозрачными объектами (необходимо сохранять не только координаты точки соударения, но и интенсивность на момент данного соударения) или *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 освещенности прибавляется интенсивность фотонов из некоторой окрестности, доступных посредством обращения к сформированной фотонной карте. В данной работе принимается, что на освещенность точки влияют *все* фотоны из ее окрестности вместо k ближайших, что в значительной степени упрощает процедуру 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:

$$I = I_{ray\ tracing} + I_{photon} \times \sum_{i=0}^{N} \max\left\{0, 1 - \frac{\|p - p_i\|}{\varepsilon}\right\} \qquad (1)$$

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

Согласно работе [2] сбор всех фотонов из фиксированной окрестности вместо k ближайших может привести к худшей оценке освещенности в областях с малой плотностью фотонов или размытому результату в областях с высокой плотностью. Тем не менее, за счет применения излагаемых далее быстрых алгоритмов выигрыш производительности оказывается весьма существенным и оправдывает данный подход. Кроме того, в качестве альтернативной весовой функции можно использовать *интерполяцию Эрмита* вида $3t^2–2t^3$, t – параметр в диапазоне $[0, 1]$, отдавая больший приоритет близким фотонам и меньший – более отдаленным (в сравнении с линейным весом). 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:

$$I = I_{ray\ tracing} + I_{photon} \times \sum_{i=0}^{N} \max\left\{0, 1 - \mathrm{smoothstep}\left(0, \varepsilon, \|p - p_i\|\right)\right\} \qquad (2)$$

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

2.2 Сортировка фотонов

Описанный подход неэффективен, поскольку предполагает перебор *всех* элементов фотонной карты. Для повышения производительности можно использовать предварительную сортировку фононной карты по координатам фотонов в лексикографическом порядке. Данная сортировка позволяет применять алгоритм бинарного поиска со сложностью $O(\log N)$ для определения индексов фотонов с *минимальной* и *максимальной* координатой в кубической ε-окрестности точки соударения. В результате вычисления на втором проходе сокращаются до перебора фотонов из *непрерывного* сегмента карты, заключенного между данными индексами:

$$I = I_{ray\ tracing} + I_{photon} \times \sum_{i=N_1}^{N_2} \max\left\{0, 1 - \frac{\|p - p_i\|}{\varepsilon}\right\} \qquad (3)$$

Здесь N_1 и N_2 – индексы фотонов с минимальными и максимальными координатами соответственно. Сортировка может быть выполнена средствами ЦПУ или ГПУ. В первом случае целесообразно использовать *быструю* сортировку со сложностью $O(N \log N)$, во втором – *поразрядную* [7] или *битоническую* [8] сортировку с трудоемкостью $O(N)$ и $O(N \log^2 N)$ соответственно.

Данный подход значительно повышает производительность визуализации, однако также не вполне эффективен: сбор фотонов осуществляется не из ε-окрестности точки соударения, а из ε-полосы вида $\|p_x - p_{i_x}\| \leqslant \varepsilon$. Далее будет рассмотрен алгоритм, позволяющий сократить перебор до некоторой окрестности точки соударения. Данный алгоритм развивает подход, предложенный в работе [3], обеспечивая более эффективное построение и использование ускоряющей структуры.

2.3 Воксельная ускоряющая структура

Разделим ограничивающий параллелепипед сцены на *воксели* со стороной ε (радиус окрестности для сбора фотонов). Тогда каждому фотону с координатами (x_p, y_p, z_p) можно сопоставить координаты вокселя, в котором он находится:

$$x_v = \left\lfloor \frac{x_p - x_{min}}{\varepsilon} \right\rfloor, \; y_v = \left\lfloor \frac{y_p - y_{min}}{\varepsilon} \right\rfloor, \; z_v = \left\lfloor \frac{z_p - z_{min}}{\varepsilon} \right\rfloor \qquad (4)$$

Здесь через $(x_{min}, y_{min}, z_{min})$ обозначена *минимальная* точка ограничивающего параллелепипеда сцены.

После формирования фотонной карты описанным выше способом все фотоны сортируются в лексикографическом порядке по координатам соответствующих *вокселей*. При таком способе сортировки фотоны, принадлежащие одному вокселю, будут эквивалентны и займут *непрерывный* сегмент в отсортированной фотонной карте.

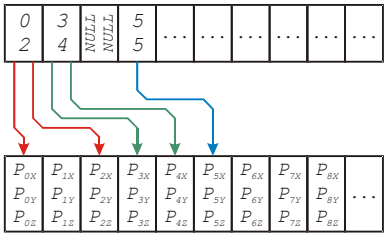

Рис. 2. Воксельная ускоряющая структура

Для отсортированной фотонной карты возможно создание воксельной ускоряющей структуры. Для этого необходимо

сформировать массив двумерных векторов, каждый элемент которого соответствует одному вокселю. В первую компоненту вектора записывается индекс первого фотона в соответствующем вокселе, во вторую – индекс последнего фотона. Для представления данного массива на ГПУ удобно использовать трехмерную текстуру.

Использование ускоряющей структуры на этапе визуализации основано на тривиальном соображении: все фотоны в ε-окрестности точки соударения принадлежат вокселям, координаты которых отличаются не более чем на 1 от координат вокселя для данной точки (в противном случае разность координат составляет не менее длины вокселя, которая равна ε). В итоге для вычисления освещенности в конкретной точке необходимо произвести суммирование по всем фотонам из $3^3 = 27$ вокселей. Окончательная формула будет выглядеть следующим образом:

$$I = I_{ray\ tracing} + I_{photon} \times \sum_{i=x_v-1}^{x_v+1} \sum_{j=y_v-1}^{y_v+1} \sum_{k=z_v-1}^{z_v+1} \sum_{l=N_1^{ijk}}^{N_2^{ijk}} \max\left\{0, 1 - \frac{\|p - p_l\|}{\varepsilon}\right\} \quad (5)$$

Здесь (x_v, y_v, z_v) – координаты вокселя, которому принадлежит рассматриваемая точка, N_1^{ijk} и N_2^{ijk} – индексы соответственно первого и последнего фотона в вокселе (x_v, y_v, z_v). В большинстве случаев значимыми окажутся лишь несколько вокселей из окрестности, в то время как пустые воксели будут быстро отброшены.

2.4 Формирование массива вокселей

Для типичной сцены значение параметра ε лежит в диапазоне 0,5 – 1,5% от размера сцены (в качестве которого примем наибольшую сторону ограничивающего параллелепипеда). Такой выбор параметра ε порождает множество вокселей размером не более 256^3 элементов, что делает возможным создание и хранение данной ускоряющей структуры на ГПУ. Кроме того, в ряде случаев можно ограничиться *двумерным* массивом вокселей для моделирования каустик на плоской поверхности (например, на полу и стенах комнаты), что сокращает размер массива на два порядка. Для вычисления индекса первого и последнего фотона в каждом вокселе предлагается следующий параллельный алгоритм.

Для каждого элемента *отсортированной* фотонной карты выполняется проверка, является ли данный элемент *первым* в соответствующем вокселе (не лежит ли предыдущий элемент в *другом* вокселе). В случае положительного результата номер элемента записывается в соответствующий воксель в качестве индекса первого элемента. Аналогичная процедура выполняется для определения индекса последнего элемента вокселя. Поскольку в каждом вокселе существует единственный первый и последний элемент, не возникнет никаких коллизий при параллельном выполнении указанных операций. Данный алгоритм эффективно выполняется на ГПУ и имеет линейную сложность $O(N)$, где N – число генерируемых фотонов.

2.5 Схема результирующего алгоритма

В результате получаем следующий четырехпроходный алгоритм:

- Генерация фотонной карты с помощью *прямой* трассировки лучей. На ГПУ фотонная карта представлена двумерной текстурой размера $P_X \times P_Y$, каждый тексель которой соответствует ровно одному выпущенному фотону. Общее число фотонов $N = P_X \times P_Y$. Для реализации данного этапа использовался шейдерный язык GLSL.
- "Воксельная" сортировка фотонной карты на ГПУ. Эффективными алгоритмами являются битоническая сортировка со сложностью $O(N \log^2 N)$ и поразрядная сортировка со сложностью $O(N)$. В данной работе реализована битоническая сортировка на базе OpenCL.
- Формирование массива вокселей для отсортированной фотонной карты на ГПУ. Предложенный алгоритм решает данную задачу за линейное время $O(N)$. Для реализации данного этапа использовался OpenCL.
- Визуализация сцены с помощью *обратной* трассировки лучей. Для обработки каустик выполняется сбор фотонов из окрестности точки соударения предложенным выше способом. Наряду с первым проходом для реализации выбран шейдерный язык GLSL. Оба этапа используют общие шейдеры с минимальными изменениями.

Предлагаемый алгоритм обеспечивает эффективную визуализацию каустик, выполняя подготовку необходимой ускоряющей структуры за *линейное* время $O(N)$ при использовании поразрядной сортировки.

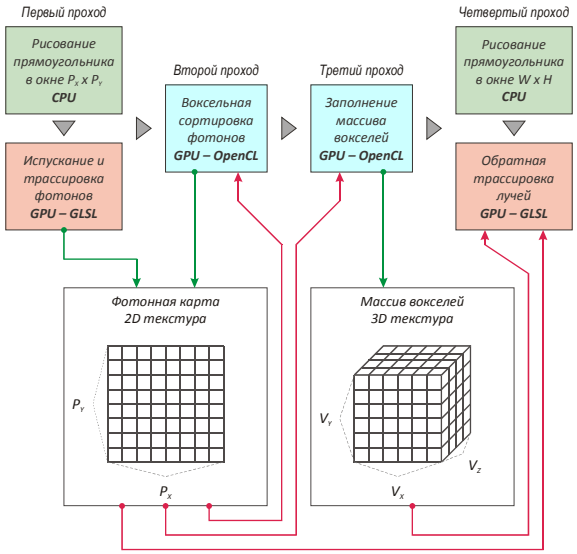

Рис. 3. Схема работы результирующего алгоритма

Необходимо сделать несколько замечаний относительно программной реализации. Текущая версия стандарта OpenCL не позволяет ядру выполнять чтение и запись в одну и ту же текстуру, что необходимо на этапе воксельной сортировки. Для обхода данного ограничения текстура с фотонной картой копируется в буфер (простой одномерный массив) перед этапом сортировки, содержимое которого вновь копируется в текстуру после окончания сортировки. На этапе формирования массива вокселей запись первого и последнего фотона осуществляется в различные компоненты текселя. Тем не менее, стандарт не позволяет обновлять *отдельные* компоненты текселя: записи подлежит сразу весь вектор цвета. Для обхода ограничения данный этап разбивается на два прохода: при первом проходе определяются индексы первых фотонов, а при втором – последних.

3. ОЦЕНКА ПРОИЗВОДИТЕЛЬНОСТИ

Оценка производительности рассмотренных алгоритмов проводилась на примере визуализации неявно заданной поверхности – "*Metaballs*" или "*Blobs*" [9]. Поверхность визуализировалась непосредственным образом без предварительной триангуляции. При расчете учитывался эффект полного внутреннего отражения при глубине трассировки 5.

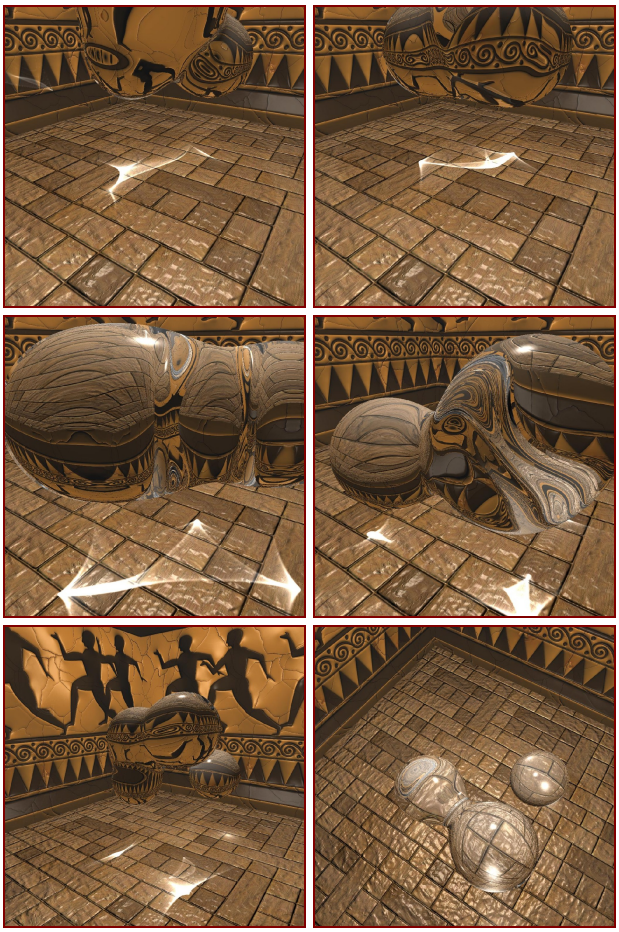

Рисунок 4. Пример визуализации "*Metaballs*"
(число фотонов $N = 256 \times 256$, глубина трассировки 5)

Для проведения эксперимента использовались графические ускорители AMD Radeon 5870 (R800) и NVIDIA Quadro FX 5600 (G80). Карта AMD работала под управлением драйвера Catalyst 10.4 и ATI Stream SDK 2.1, карта NVIDIA – под управлением драйвера ForceWare 197.13 и CUDA Toolkit 3.0. Для тестовой сцены ограничивающий параллелепипед имеет размеры $10 \times 10 \times 10$, а значение параметра $\varepsilon = 0{,}04 = 0{,}4\%$ от размера сцены. Визуализация выполнялась в разрешении 1920×1080 (Full HD). В следующих таблицах представлены результаты замера производительности.

Результаты экспериментов позволяют сделать вывод о высокой эффективности предложенного метода. Воксельная ускоряющая структура более чем в четыре раза быстрее метода на основе лексикографической сортировки и обеспечивает качественную визуализацию каустик в реальном времени с разрешением Full HD.

Таблица 1. Лексикографическая сортировка фотонов

Число фотонов	Лексикографическая сортировка (мс)		Производительность визуализации (к/с)	
	R5870	Q5600	R5870	Q5600
$128^2 = 16K$	10,2	12,5	20	7,5
$256^2 = 65K$	14,5	24,2	8	3,2
$512^2 = 262K$	28,5	60,1	3	1,0

Таблица 2. Воксельная ускоряющая структура

Число фотонов	Воксельная сортировка и заполнение массива вокселей (мс)		Производительность визуализации (к/с)	
	R5870	Q5600	R5870	Q5600
$128^2 = 16K$	10,5	16,8	49,5	19,5
$256^2 = 65K$	17,5	30,3	35,8	14,5
$512^2 = 262K$	39,4	98,5	18,2	6,5
$1024^2 = 1M$	140	381	5,2	1,8

4. ЗАКЛЮЧЕНИЕ

Предложен упрощенный вариант метода фотонных карт для выполнения на ГПУ в реальном масштабе времени. Данный метод позволяет корректно моделировать каустики, возникающие при взаимодействии света с прозрачными или отражающими объектами. Эффективность визуализации достигается за счет воксельной ускоряющей структуры, которая формируется непосредственно на ГПУ средствами OpenCL. Формирование данной структуры включает в себя "воксельную" сортировку фотонов и определение для каждого вокселя индекса первого и последнего фотона. Данные задачи могут быть решены с помощью алгоритмов со сложностью $O(N)$. Предложенный подход обеспечивает высокую скорость работы при использовании фотонных карт вплоть до 1 миллиона элементов с разрешением Full HD.

5. 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

[7] "*Radix sort*" (From Wikipedia, the free encyclopedia).
http://en.wikipedia.org/wiki/Radix_sort

[8] "*Bitonic sorter*" (From Wikipedia, the free encyclopedia).
http://en.wikipedia.org/wiki/Bitonic_sorter

[9] Ryan Geiss. "*Metaballs (Blobs)*" (3/10/2000).
http://www.geisswerks.com/ryan/BLOBS/blobs.html