Моделирование каустик на графическом процессоре

Денис Боголепов, Татьяна Удалова, Вадим Турлапов
Нижегородский государственный университет им. Н.И. Лобачевского
Факультет вычислительной математики и кибернетики, Нижний Новгород, Россия
bogdencmc@inbox.ru, tauname@rambler.ru, vadim.turlapov@cs.vmk.unn.ru

Аннотация

The goal of this work is adaptation of the photon mapping method for caustics modeling in real time. The hybrid algorithm executed both on CPU and GPU is proposed. Building of the photon map and ray tracing are performed on GPU. To improve performance preliminary sorting of the photon map on CPU is used. Performance estimation is presented.

Keywords: *Interactive Graphics, GPU Ray Tracing, GPU Photon Mapping, Caustics, GPGPU.*

1. ВВЕДЕНИЕ

Целью настоящей работы является адаптация *метода фотонных карт* [1] для исполнения на графическом процессоре в реальном масштабе времени, что позволит корректно моделировать отражающие и прозрачные объекты.

2. РЕАЛИЗАЦИЯ МЕТОДА ФОТОННЫХ КАРТ

2.1 Простейший подход к моделированию каустик

Для моделирования каустик на основе метода фотонных карт можно предложить следующий двухпроходный алгоритм. При первом проходе в соответствии с расположением и ориентацией источника света генерируются фотоны света, число которых соответствует числу текселей в текстуре фотонной карты `PhotonMapTexture`. В простейшем случае можно рассматривать *точечные* (испускают фотоны равномерно по всем направлениям) или *прямоугольные* (из произвольной точки поверхности испускают фотоны в произвольном направлении полупространства) источники света.

Рисунок 1. Примеры источников света

Для каждого фотона света прослеживается траектория его распространения и возможные взаимодействия с объектами сцены. Если фотон соударяется с *диффузным* объектом сцены, в качестве результата фрагментный шейдер возвращает *координаты* найденной точки соударения. Если объект обладает *отражающими* или *преломляющими* свойствами, отслеживаются дальнейшие взаимодействия фотона вдоль направления *отражения* или *преломления* соответственно. Наконец, если фотон покинул сцену без соударения с объектами, в качестве результата возвращаются заведомо *недопустимые* координаты. Интенсивность всех фотонов полагается одинаковой и задается параметром I_{photon}. Данный подход не обеспечивает моделирования *хроматических аберраций*. Однако это ограничение может быть снято путем генерации фотонов отдельно для каждого канала RGB и использования дополнительной текстуры для хранения интенсивностей.

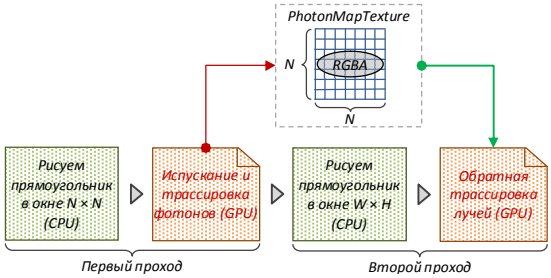

Рисунок 2. Простейшая реализация метода фотонных карт

При втором проходе выполняется традиционный алгоритм обратной трассировки лучей. При этом к вычисленной в точке соударения освещенности прибавляется интенсивность фотонов из некоторой окрестности, которые доступны посредством обращения к текстуре фотонной карты. В данной работе использовалась следующая формула для вычисления итоговой освещенности точки:

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

где p – координаты точки соударения, в которой вычисляется освещенность, а p_i – координаты фотона, записанного в i-ый тексель текстуры фотонной карты. Параметр $\varepsilon > 0$ задает радиус окрестности, с которой производится сбор фотонов. Значение данного параметра подбирается эмпирически для конкретной сцены.

2.2 Развитие подхода

Описанный подход имеет низкую эффективность, поскольку во время второго прохода вычисление освещенности каждой точки соударения требует полного перебора всех элементов текстуры фотонной карты. Эффективным способом повышения производительности является предварительная *сортировка* данной текстуры, что позволит фрагментному шейдеру обратной трассировки использовать алгоритм бинарного поиска для определения фотона с *минимальными* и *максимальными* координатами в ε-окрестности точки соударения. Если N_1 и N_2 – индексы фотонов с минимальными и максимальными координатами соответственно, то формула для вычисления итоговой освещенности точки соударения перепишется следующим образом:

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

Таким образом, суммирование производится не по всем элементам текстуры фотонной карты, а лишь по некоторому

промежутку текстурных координат. Модифицированная схема визуализации выглядит следующим образом:

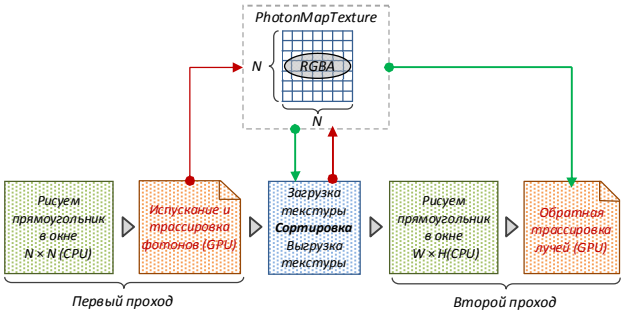

Рисунок 3. Улучшенная реализация метода фотонных карт

Указанная сортировка может быть выполнена как на *центральном*, так и на *графическом* процессоре. Сортировка на графическом процессоре является весьма затратной: обработка текстуры из миллиона элементов требует выполнения сотен проходов рендеринга [3]. С другой стороны, использование центрального процессора требует загрузки текстуры в системную память и последующей выгрузки в память графического ускорителя. Данные операции нагружают системную шину и могут служить лимитирующим фактором для производительности. Тем не менее, при использовании не слишком больших фотонных карт (до миллиона элементов) данный подход может оказаться вполне эффективным и используется в данной работе. Сортировка проводилась с помощью стандартной библиотечной функции *qsort* языка C.

3. ОЦЕНКА ПРОИЗВОДИТЕЛЬНОСТИ

Оценка производительности рассмотренных алгоритмов проводилась на примере визуализации водной поверхности.

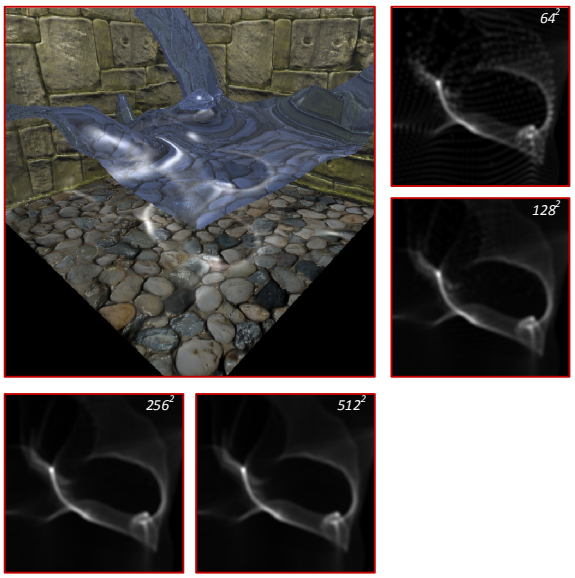

Рисунок 3. Изображения каустик для фотонных карт различного размера (*64 × 64 – 512 × 512*)

На основе физических расчетов формируется неявно заданная функция трех переменных $\varphi(x,y,z)$, нулевое значение уровня которой соответствует водной поверхности.

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 ε 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*

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.

5. ЛИТЕРАТУРА

[1] Henrik Wann Jensen. "*Realistic Image Synthesis Using Photon Mapping*". AK Peters. Ltd., Massachusetts. 2001.

[2] Tin-Tin Yu, John Lowther, Ching-Kuang Shene. "*Photon mapping made easy*". ACM SIGCSE Bulletin, v.37 n.1, 2005.

[3] Timothy J. Purcell1, 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] Jag Mohan Singh, P. J. Narayanan, "*Real-Time Ray-Tracing of Implicit Surfaces on the GPU*". IEEE Transactions on Visualization and Computer Graphics, 2009.