ИНТЕРАКТИВНЫЙ РЕНДЕРИНГ НА GPU: ВЫЧИСЛЕНИЕ ФУНКЦИИ ВИДИМОСТИ

С.С. Свистунов

Тульский госуниверситет

Тени в компьютерной графике придают реалистичность. Для получения мягких теней необходимо вычислять значения функции видимости, что является затратной операцией. Предлагаются способы расчета функции видимости при помощи GPU с использованием OpenGL и буферов глубины, а также основанные на технологии NVIDIA CUDA.

Введение

Изучается задача интерактивного рендеринга трехмерной сцены, состоящей из модели и удаленного окружающего освещения L. Рассматривается случай первичного освещения, определяемого уравнением освещенности Kajiya [1]

$$B_p(v)=\int_{S^2} L(x)V_p(x)\rho_p(x,v)d\mu(x). \tag{1}$$

Здесь $B_p(v)$ – яркость отраженного освещения из точки модели p в направлении $v\in S^2$; $V_p(x)$– функция видимости; $\rho_p(x,v)$ – BRDF с учтенным ламбертовским множителем $(n_p x)_+$, где n_p – нормаль к поверхности модели в точке p.

Одной из самых затратных операций является вычисление значений функции видимости $V_p(x)$. Это бинарная функция, равная нулю, если луч, выпущенный из точки p в направлении x, пересекает модель, и единице – иначе. Для непрозрачной модели $V_p(x)=0$, если $n_p x<0$.

Учет функции видимости в уравнении (1) дает затенения модели, что приводит к большей реалистичности изображения. Легко посчитать аналитическое выражение функции видимости для шара. Все остальные случаи сложны, поэтому требуются качественные аппроксимации функции видимости. В работе [2] предлагается способ быстрого приближенного вычисления функции видимости в случае, когда модель покрывается шариками. Однако он имеет свои недостатки, в частности, непросто получать подобные покрытия (особенно в интерактивном режиме). Поэтому несомненный интерес представляет быстрый подсчет значений функции видимости для дискретных направлений прямо для полигональной модели (или, возможно, ее низкополигональной аппроксимации).

1. Расчет функции видимости

Основной вклад в затенение модели в точке p дает нулевой сферический коэффициент Фурье функции видимости, тесно связанный с моделью затенения Ambient Occlusion (AO) [3].

В методе AO (также как в интегральной теореме о среднем значении) полагается

$$B_p(v)\approx A_p\int_{S^2} L(x)\rho_p(x,v)d\mu(x), \tag{2}$$

где константа A_p также называется АО в точке p. Отметим, что приближенное вычисление интеграла в формуле (2) уже заметно проще и есть разные способы его аппроксимации, в частности, метод PRT [4]. Нами разработан более общий, быстрый и удобный метод сферических дизайнов SDPRT [5, 6].

Есть несколько способов определения АО. В диффузном случае для ламбертовской поверхности, когда $\rho(x,v)=(n_p x)_+$, константа A_p выбирается из условия точности формулы (2) для константного освещения. В этом случае

$$A_p = 4\int_{S^2} V_p(x)(n_p x)_+ \, d\mu(x).$$

Таким образом, задача нахождения АО сводится к приближенному вычислению интеграла по сфере (на самом деле полусфере из-за множителя $(n_p x)_+$) от негладкой численно определяемой функции. Подобные интегралы вычисляются при помощи кубатурных формул Монте-Карло или, что предпочтительнее, сферических дизайнов [5, 6]. В обоих случаях приближенно имеем

$$A_p = \frac{4}{N}\sum_{k=1}^{N} \lambda_k V_p(x_k)(n_p x_k)_+ , \quad (3)$$

где λ_k, x_k – соответственно веса и узлы выбранной кубатурной формулы. Для метода Монте-Карло при использовании стандартных генераторов случайных чисел на сфере обычно используется значение $N = 10^4 - 10^5$. Метод дизайнов при схожей точности позволяет использовать кубатуры с $N = 10^2 - 10^3$, что является их несомненным преимуществом.

В итоге приходим к задаче, типичной для рейтрейсинга: в каждой точке модели необходимо найти пересечения пучка лучей с полигонами модели. Прямое решение при помощи метода грубой силы требует $O(V \cdot N \cdot F)$ итераций, где V – число вершин, F – число полигонов, поскольку для вычисления значения $V_p(x_k)$ требуется найти пересечение луча x_k из точки p с моделью, что приводит к необходимости перебора всех полигонов модели. Его выполнение на CPU для моделей с сотнями тысяч вершин и полигонов занимает часы.

Согласно (3) алгоритм расчета сводится к следующим действиям:

1) для каждой вершины p выпустить пучок из N лучей;

2) для каждого луча x_k определить пересечение с моделью и, как следствие, вычислить значение функции видимости $V_p(x_k)$;

3) просуммировать результат с учетом весов и ламбертовского множителя $(n_p x_k)_+$.

Естественно, для ускорения расчетов в первую очередь необходимо воспользоваться ускоряющими структурами, например, BVH-деревьями или пространственным хешированием. Тогда в лучшем случае получаем цикл из $O(V \cdot N)$ итераций при условии, что построение вспомогательных структур выполняется за $O(F)$ итераций и они заполняются равномерно.

Однако оказалось, что гораздо большее ускорение можно получить при использовании возможности современных GPU по параллельным вычислениям: технологии NVIDIA CUDA и OpenCL. Некоторые из использованных нами идей приведены в [7].

2. Способы ускорения расчетов с использование GPU

Первая идея состоит в том, что вычисления функции видимости в разных точках модели не зависят друг от друга и, значит, их можно распараллелить по вершинам мо-

дели, что для метода грубой силы в лучшем случае приводит к циклу из $O(N \cdot F)$ итераций. Вторая идея использует то, что современные видеокарты имеют возможность в заданной точке модели быстро получать буфер глубины по всем пространственным направлениям. Третья идея заключается в том, что пространственное хеширование также можно очень быстро выполнять в параллельном режиме.

Кратко приведем особенности реализаций.

2.1. Использование буферов глубины

Для ускорения расчетов воспользуемся возможностью GPU сохранять буфер глубины. Представим, что в каждой вершине модели расположено шесть камер, ориентированных по нормали и направленных по соответствующим координатным осям. После этого каждая камера «снимает» (рендерит) буфер глубины. Таким образом получаем кубическую карту глубины в данной точке модели. Важно, что эти операции на GPU выполняются очень быстро.

Теперь для поиска пересечений луча с моделью достаточно определить необходимую сторону карты окружения и с помощью простых матричных преобразований вычислить пиксель, отвечающий лучу. После этого по цвету пикселя в текстуре буфера глубины легко определить, было ли пересечение или нет (если цвет не черный, значит пересечение есть). Это процесс схож с известным методом построения теней от точечных источников света – теневые карты (Shadow mapping) [8].

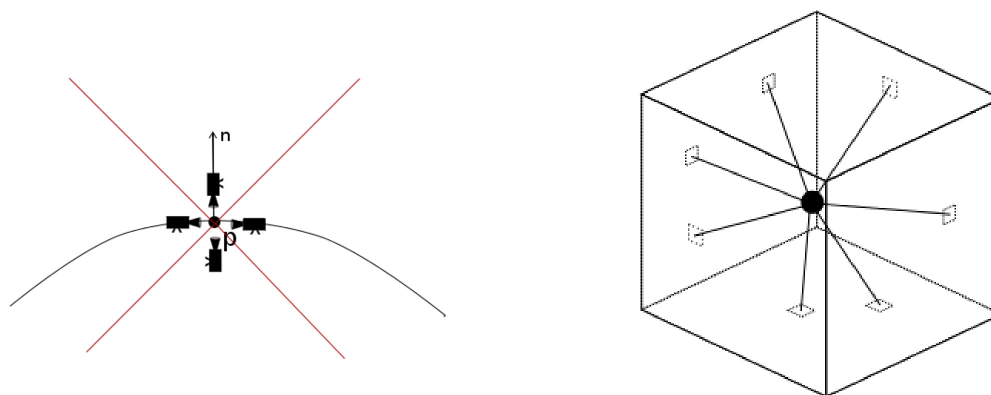

Рис. 1. Организация расчетов с использованием буфера глубины: расположение камер для получения карты окружения (слева); выбор значений из карты окружения (справа)

Такой подход позволяет избежать вложенного цикла по полигонам при поиске пересечений луча с моделью и заменить его созданием карты глубины, что делается очень быстро с помощью стандартных средств OpenGL/DirectX.

2.2. Использование NVIDIA CUDA

Для выполнения расчетов с использованием технологии NVIDIA CUDA применялись различные способы распараллеливания вычислений: по вершинам модели (ядро с одномерной топологией), по вершинам и лучам (ядро с двумерной топологией), по полигонам и лучам (ядро с двумерной топологией). Наиболее удобным и результативным подходом оказалось распараллеливание по вершинам и лучам.

При реализации возникла проблема ограничения максимального времени выполнения одного запуска ядра (около одной секунды). Для обхода этого ограничения вычисления разбивались на несколько этапов: обработка по несколько лучей, обработка частей модели.

Стоит уделять особое внимание работе с различными типами памяти (особенно на бюджетных видеокартах). Самый простой, но наименее эффективный способ реализации – это трансляция алгоритма однопроцессорной версии и использование глобальной памяти видеокарты. Такой подход дает ускорение в расчетах, но простая замена глобальной памяти на текстурную позволяет в разы улучшить производительность.

Предпринимались попытки использования различных типов памяти видеокарты – от глобальной и текстурной до реализации управляемого кэша в разделяемой памяти и побитового хранения результатов вычислений. При использовании разделяемой памяти в ней сохранялась информация о полигонах, что позволило (как и положено) снизить количество чтений из глобальной памяти и ускорить расчет.

Однако наибольшей производительности удалось достичь при использовании разделяемой памяти и битового формата хранения значений функции видимости.

3. Результаты

Тестовая сцена состояла примерно из пяти тысяч полигонов и вершин. Для нее вычисление функции видимости (AO) заняло приблизительно 0.8 сек (расчеты велись на бюджетной видеокарте Nvidia GTX 260):

Метод	Время расчета AO
CPU	более 1 часа
Буферы глубины	около 11 сек
NVIDIA CUDA	около 0.8 сек

По ссылке http://goo.gl/AYNk7 можно скачать видео, иллюстрирующее процесс визуализации.

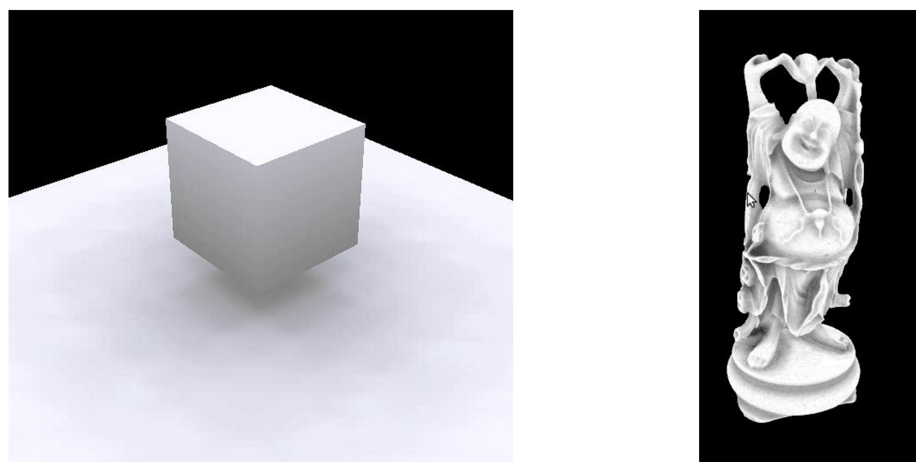

Рис. 2. Результат расчета функции видимости: расчет на тестовой сцене (слева); расчет на модели buddha [9] (справа)

Литература

1. Kajiya J.T. The rendering equation, Computer Graphics (Proceedings of SIGGRAPH 1986), 1986, 20 (4), 143–150.
2. Zhong Ren, Rui Wang, John Snyder, Kun Zhou, Xinguo Liu, Bo Sun, Peter-Pike Sloan, Hujun Bao, Qunsheng Peng and Baining Guo. Real-time Soft Shadows in Dynamic

Scenes using Spherical Harmonic Exponentiation // ACM Transaction on Graphics. Proceedings of SIGGRAPH . 2006. V. 25, N 3. P. 977–986.
3. A real time self ambient occlusion method from Nvidia’s GPU Gems 2 book. http://download.nvidia.com/developer/GPU_Gems_2/GPU_Gems2_ch14.pdf
4. Green R. Spherical harmonic lighting: The gritty details // In Game Developers' Conference. 2003.
5. Свистунов С.С. Интерактивный рендеринг при помощи сферических дизайнов (SDPRT) для низкочастотного окружающего освещения и BRDF Фонга // Изв. ТулГУ. Сер. Естественные науки. 2011. Вып. 1. С. 188–199.
6. Горбачев Д.В., Иванов В.И., Странковский С.А. Моделирование освещения в интерактивной графике при помощи сферических дизайнов // Изв. ТулГУ. Сер. Естественные науки. 2007. Т. 1, Вып. 1. С. 17–36.
7. Боресков А.В., Харламов А.А. Основы работы с технологией CUDA. М.: ДМК-Пресс, 2010.
8. Gary King, Shadow Mapping Algorithms – [ftp://download.nvidia.com/developer/presentations/2004/GPU_Jackpot/Shadow_Mapping.pdf].
9. The Stanford 3D Scanning Repository – [http://graphics.stanford.edu/data/3Dscanrep/].