Некоторый опыт применения Nvidia OptiX

В. Дебелов
Лаборатория численного анализа и машинной графики
Институт вычислительной математики и математической геофизики СО РАН,
Новосибирск, Россия
debelov@oapmg.sscc.ru

Аннотация

В данном сообщении рассматривается причина выбора OptiX в противовес CUDA в качестве инструмента для научно-исследовательской разработки при использовании Nvidia GPU.

На примере разработки простейшего рендера для сцен, состоящих из полупрозрачных оптически изотропных объектов, рассматриваются побудительные мотивы к использованию OptiX и основные привлекательные характеристики OptiX. Сообщается также о неприятности, которая может случиться во время работы программы.

Ключевые слова: *трассировка лучей, спектральный рендеринг, Nvidia GPU, CUDA, OptiX.*

1. ВВЕДЕНИЕ

Достаточно много научно-исследовательских программ разрабатывается для проведения численных экспериментов во время отладки и получения различных характеристик разрабатываемого алгоритма. Во многих случаях алгоритм последовательный, но для качественной проверки требуется получить некоторую целостную картину, просчитывая его для ряда связанных наборов параметров. В нашей области знаний алгоритм работает для серии пикселей.

Простой прямолинейный подход – просчитать алгоритм для всех пикселей последовательно – приводит к длительному затягиванию времени проведения экспериментов, особенно при наличии рекурсии. Часто применяется MPI, но для реального получения выгоды во времени счета требуется наличие подручного кластера. В последнее время стало модно программировать параллельные приложения, используя технологию CUDA, а значит:

- Надо изучить технологию. Знать архитектуру ГПУ. Очень времяемкий процесс. Или приглашать программиста, которому надо будет объяснить суть алгоритма в некотором объеме.
- Имеющийся код на C/C++ необходимо значительно переделать.
- По окончании экспериментов код на CUDA становится ненужным. Также, скорее всего, этот код не подойдет для GPU других производителей.

В принципе, подход напоминает переход с языка высокого уровня на более низкий уровень, что, как правило, ведет к уменьшению производительности труда исследователя, хотя может дать значительный выигрыш в производительности программы.

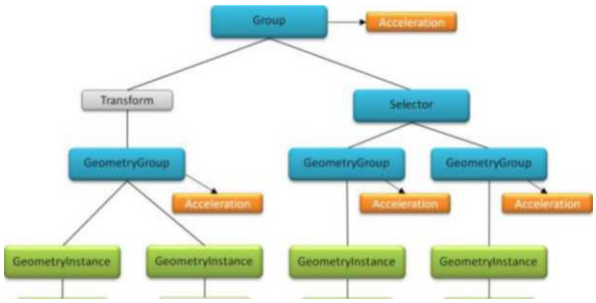
В последнее время (с 2008 г.) Nvidia популяризирует технологию OptiX, построенную над CUDA для создания параллельных программ для приложений, основанных на трассировке лучей [2]:

- Хотя OptiX построена над CUDA, программист не обязан знать CUDA. Однако для знатоков CUDA, OpenGl и DirectX возможно применение их знаний для построения более эффективных программ.
- Программист должен немного перестроить свои модули на C или C++ при переходе на OptiX. В основном это касается косметических изменений при передаче параметров и организации основного цикла.
- Очень полезная черта: исходные коды переносимы между Windows и Linux платформами.
- Большой набор примеров приложений в исходных кодах, например, алгоритмы Виттеда, Кука.
- Для Windows обеспечивается автоматизация компиляции и сборки программ в рамках Visual Studio 2010. Для Visual Studio 2012 создан wizard приложений в среде OptiX.

Отрицательной стороной OptiX на современном этапе является ориентация на float, а не на double. Использование последних потребует определенных усилий.

В данной работе не приводится описание OptiX в какой-либо сокращенной форме, а просто показано, как применение механизма ускорения сказывается на времени работы программы.

2. СТРУКТУРА СЦЕНЫ OPTIX



Geometry **Geometry** **Geometry** | **Geometry**

Рис. 1. Граф сцены OptiX

На рис. 1 приведен пример графа сцены,

здесь:

- Group – вся сцена или ее подграф.
- Transform, Selector – способ включения подграфа в данный узел – через преобразование координат или как выбор из альтернативы.
- GeometryGroup – геометрия некоторого, возможно, осмысленного объекта сцены.
- Geometry – геометрия, набор однотипных геометрических примитивов. Может быть только один примитив.
- GeometryInstance – геометрия, специфицированная одним или несколькими материалами.
- Acceleration – способ организации геометрии подграфа, предназначенный для ускорения операции пересечения

луча с геометрией, заданной данным подграфом на основе габаритных боксов сыновних узлов данного узла.

Структура описания сцены вполне естественная, поэтому здесь переход на OptiX не вызывает трудности. Явно ни один из примитивов не является в OptiX стандартным. Для каждого используемого примитива программист должен предоставить две процедуры: 1) вычисление габаритов примитива; 2) вычисление точки (точек) пересечения произвольного луча с примитивом. Здесь надо отметить, что богатый набор примерных приложений, поставляемых с OptiX, содержит большое число требуемых процедур, запрограммированных с высоким качеством.

Еще один очень полезный механизм предлагается в OptiX – это *Callable* программы, которые, по сути, C/C++ функции, вызываемые в модулях, работающих на GPU. Получается, что нет ощутимых ограничений на стиль программирования.

3. НЕМНОГО О ТЕСТОВОМ ПРИЛОЖЕНИИ

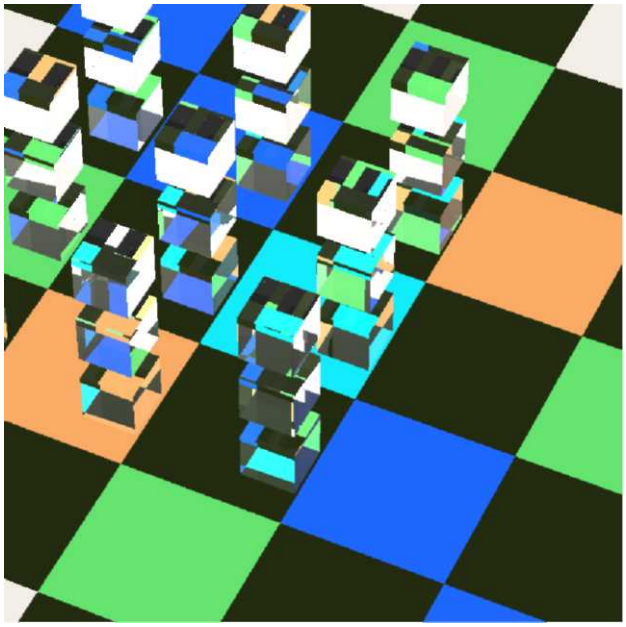


Рис. 2. Общий вид сцены

Описываемое экспериментальное приложение программировалось для отладки расчета взаимодействия луча поляризованного света с границей двух оптически изотропных сред, например, с границей воздуха или воды с кристаллом алмаза или стеклом. Алгоритм расчета взят из работы [1] и вычислительно является очень трудоемким. Тестовая сцена (см. рис. 1) включает в себя 27 прозрачных кубов и излучающего квадратного многоцветного источника света. Воображаемая охватывающая сцену сфера излучает освещенность 40% стандартного источника CIE D65. Как и в работе [1], применяется обратная рекурсивная трассировка лучей и спектральный рендеринг: для 81 длины волны в видимом диапазоне от 380 до 780 нм. Все кубы идентичны, для их простоты коэффициент преломления положен равным 1.2 и не зависит от длины волны.

4. ЭКСПЕРИМЕНТЫ

Итак, для отладки последовательного алгоритма, предназначенного для вычисления одного пересечения луча с границей двух сред, пишем последовательный рендер-П для расчета изображения сцены. Второй рендер-О программируем с применением OptiX. За день добиваемся того, чтобы все обрабатывающие модули имели идентичный код, за исключением способа передачи параметров. Никакого знания об GPU, кроме того, что он разработан Nvidia и имеет сколько-то CUDA-ядер и его *compute capability* SM_NM (например, SM_21, SM_50). Последнее надо указать в параметрах сборки приложения.

Основная платформа для экспериментов: Intel® Core i7-3630QM @ 2.40GHz, Win 9.1, x84, VS2010, CUDA 5.5, OptiX 3.5.1. GPU1: GeForce GT 650M, **384** CUDA Cores. GPU2: GeForce GTX 560 Ti, **448** CUDA Cores. GPU2 стоит в десктопе, который сам в сравнениях не используется.

Расчет рендером-П сцены рис. 2 (768x768 пикселей) занял:

Глубина трассировки	Время
6	204
8	298

При этом рендер-О показывает:

Глубина тр-ки	Время: GPU1	GPU2
6	23	10
8	47	22

В CUDA есть понятие варпов (warp) из 32-х ядер CUDA. Мы для себя просто считаем, что GPU1 – это Np1 = 384/32 = 12, а GPU2 – это Np2 = 448/32 = 14. Таким образом, мое видение: GPU1 – это кластер с общей памятью и 12-ю «виртуальными» процессорами, работающими почти независимо. Часть из них занимает под менеджмент сам OptiX, а на остальных параллельно выполняется трассировка лучей.

Приведенные выше времена получены программой без всяких установок, ускоряющих работу программы на GPU, т.е. режим *NoAccel* (без структур ускорения).

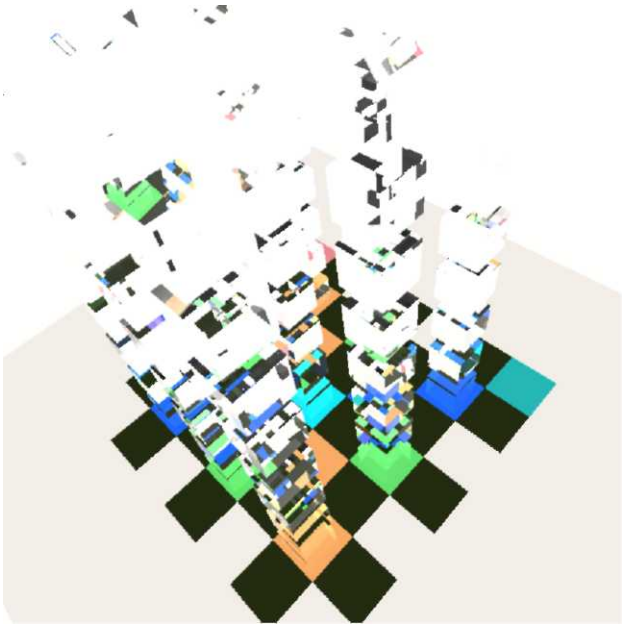


Рис. 3. Усложненная сцена

OptiX предлагает с десяток различных алгоритмов ускорения операции пересечения луча. Здесь не будут рассмотрены все они. Рассмотрим некоторые и то, как они влияют на время решения задачи. Каждый программист должен посвятить определенное время и подобрать подходящий алгоритм ускорения для его задачи, его геометрии.

Усложним сцену. Вместо трех этажей кубов сделаем девять, т.е. 81 куб, как на рис. 3. Все кубы организованы как единая Geometry, состоящая из 81 примитива типа куб. При создании GeometryInstance назначаем ту или иную стратегию ускорения: *NoAccell* – без ускорения; *Lbvh, Tbvh* – алгоритмы OptiX.

Получены следующие времена счета (г.т. – глубина трассировки):

Г.т.	Алгоритм	GPU1	GPU2
6	NoAccel	114	29
8	NoAccel	279	80
10	NoAccel	635	Q

Из приведенных цифр ясно, что на GPU2 под собственно трассировку выделено больше виртуальных процессоров, чем на GPU1. Очевидно, что менеджмент OptiX и там, и там занимает одно число процессоров. Символом Q отмечена ситуация, рассмотренная ниже.

Рассмотрим времена, полученные на GPU2 для разных глубин трассировки и разных алгоритмов ускорения.

	Алг-м/Г.т.	6	8	10	12
GPU1	NoAccel	116	275	635	-
GPU1	Lbvh	113	277	645	-
GPU2	NoAccel	29	80	222	643
GPU2	Lbvh	35	89	212	552

По этим цифрам можно сделать вывод, что для такой простой сцены выигрыш от применения алгоритмов ускорения начинает проявляться только при больших глубинах трассировки, т.е. когда число операций пересечения луча со сценой превысит некоторую величину. На GPU1 этого порога мы не стали дожидаться. Сравнение GPU1 и GPU2 показывает, что скорость расчета приложения OptiX зависит от числа ядер CUDA на GPU.

Теперь рассмотрим ту же сцену, но каждый куб опишем при помощи 12-ти треугольников, т.е. создадим треугольную сетку из 972 треугольников. Число примитивов увеличилось в 12 раз.

	Алг-м/Г.т.	6	8	10
GPU1	NoAccel	666	1623	-
GPU1	Lbvh	99	246	582
GPU1	Tbvh	99	247	585
GPU2	NoAccel	433	Q	-
GPU2	Lbvh	37	121	271
GPU2	Tbvh	41	121	273

Судя по цифрам, оба алгоритма ускорения работают примерно одинаково. Скорее всего, это связано с тем, что рассматривается очень простая ситуация – статическая сцена с одинаковыми по размерам примитивами.

5. Q – ПРОБЛЕМЫ ВИДЕОДРАЙВЕРА

В приведенных выше таблицах символ Q информирует о том, что задача снялась аварийно. Windows сообщает об этом как "Видеодрайвер перестал отвечать и был успешно восстановлен" ("Display driver has stopped working and has recovered"). Поиск по сети показал, что аналогичные сообщения возникают и в игровых программах при работе в Windows. Имя механизму – Timeout Detection and Recovery (TDR). В сети предлагается ряд способов обхода этого механизма. В моих экспериментах не все случаи удалось просчитать. Возможно возникновение ситуации Q связано с большим объемом вычислений в алгоритме из работы [1] в каждом узле дерева трассировки.

6. ЗАКЛЮЧЕНИЕ

В данном докладе сделана попытка ознакомить с некоторыми экспериментами по применению OptiX для разработки приложения, базирующегося на трассировке лучей (путей).

- OptiX оказался удобным инструментом для приложений, базирующихся на трассировке лучей.
- Появляется возможность превратить свой десктоп или ноутбук в небольшой кластер.
- Поскольку вполне достаточное подмножество API OPtiX (без прямого использования DirectX и CUDA) ориентировано только на создание определенного типа приложений, то можно ожидать, что кто-нибудь реализует OptiX на других типах GPU.
- При возникновении непонятных ситуаций и для получения достаточно оперативных консультаций полезен форум для общения с пользователями и разработчиками OptiX [3].

7. БЛАГОДАРНОСТИ

Работа была выполнена при частичной поддержке гранта РФФИ № 12-07-00386 а.

8. ССЫЛКИ

[1] Debelov V.A., Kozlov D.S. A Local Model of Light Interaction with Transparent Crystalline Media, IEEE Transactions on Visualization and Computer Graphics, 2013. - Vol. 19, No. 8. – P. 1274 - 1287.

[2] Parker S.G. OptiX: A General Purpose Ray Tracing Engine. Siggraph, 2010.

[3] Форум Nvidia OptiX: https://devtalk.nvidia.com/default/board/90/optix/1 /

ОБ АВТОРАХ

Дебелов Виктор – доктор технических наук, ведущий научный сотрудник лаборатории численного анализа и машинной графики Института вычислительной математики и математической геофизики СО РАН.

E-mail: debelov@oapmg.sscc.ru.