МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»
Кафедра математической кибернетики и компьютерных наук

ИССЛЕДОВАНИЕ СПОСОБОВ ОПТИМИЗАЦИИ ТРАССИРОВКИ ЛУЧЕЙ

АВТОРЕФЕРАТ МАГИСТЕРСКОЙ РАБОТЫ

студента 2 курса 273 группы
направления 01.04.02 — Прикладная математика и информатика
факультета КНиИТ
Карелова Максима Дмитриевича

Научный руководитель
к. ф.-м. н., доцент _____ С. В. Миронов

Заведующий кафедрой
к. ф.-м. н., доцент _____ С. В. Миронов

Саратов 2019

ВВЕДЕНИЕ

Последние анонсы новых графических процессоров NVIDIA Turing, технологии RTX и Microsoft DirectX Ray Tracing вызвали новый интерес к трассировке лучей. Использование этих технологий значительно упрощает возможность написания приложений с использованием трассировки лучей.

Трассировщики лучей позволяют создавать фотореалистичные изображения трехмерных сцен, качественно отличающиеся от изображений, полученных методом растеризации. При трассировке лучей учитываются тени, отбрасываемые объектами сцены, отражающие и преломляющие поверхности, рассеивание световых лучей.

Целью настоящей магистерской работы является исследование возможных оптимизаций трассировки лучей с использованием доступных программных и аппаратных средств, их анализ и оценка.

В процессе выполнения исследования были поставлены следующие задачи:

1. Выполнить реализацию трассировки лучей стандартными программными средствами;
2. Рассмотреть возможности улучшения базового алгоритма с использованием OpenMP;
3. Реализовать трассировку лучей с использованием программных средств OpenGL;
4. Изучить Unified Memory фрейм буфер, в который пишет и читает CPU;
5. Написать C++ решения, запускаемые на CPU или GPU;
6. Проверить скорость работы режима точности в режиме double precision floating point;
7. Рассмотреть управление памятью в C++, выделение памяти на GPU и ее использование во время выполнения программы;
8. провести серию экспериментов для сравнения параметров работы решений на основе фрагментного, вычислительного шейдеров, OpenCL и CUDA;
9. Выработать рекомендации по выбору параметров для OpenCL и CUDA.

1 Описание области исследования

Задача реализации алгоритмов трассировки лучей имеет достаточно большую историю, новый импульс которой придало появление графических процессоров, способных выполнять вычисления общего назначения. Расширенные возможности программируемости графических процессоров и появление технологий NVIDIA CUDA и AMD Stream привели к пересмотру ряда принципов, заложенных в реализацию трассировки лучей для графического процессора.

Техника рендеринга с трассировкой лучей отличается высоким реализмом, по сравнению с растеризацией, так как она имитирует распространение лучей света очень похоже на то, как это происходит в реальности. Среди недостатков способа выделяют один, но очень важный — отрисовать все вышеописанное с вычислительной точки зрения в несколько раз сложнее. Низкая производительность на существующем «железе» — главный недостаток метода трассировки, который долгое время перечеркивал все его плюсы. Нахождение пересечения лучей с объектами сцены не ускоряется столь же легко, как сравнительно простые операции при растеризации треугольников, для которых много лет используются специальные 3D-ускорители, именно поэтому в графике реального времени до сих пор используется метод растеризации, который позволяет довольно быстро нарисовать картинку, хоть и несколько уступающую в качестве полноценной трассировке, но достаточно реалистичную при этом. В тоже время алгоритм трассировки лучей хорошо распараллеливается, и его можно выполнять самым простым технически методом — увеличением числа вычислительных ядер графического процессора. При этом обеспечен линейный рост производительности при трассировке. А если учесть явную недостаточность оптимизации как аппаратного, так и программного обеспечения для трассировки лучей на GPU сейчас, можно предположить потенциально быстрый рост возможностей по аппаратной трассировке лучей.

Трассировка лучей хоть и кажется довольно простым и элегантным методом, который можно реализовать буквально несколькими строками кода, но это будет совершенно неоптимизированный алгоритм, а высокопроизводительный код для трассировки лучей сделать крайне сложно. Если при растеризации алгоритм работает быстро, но приходится придумывать хитрые

методы для сложных визуальных эффектов, то трассировка лучей умеет отрисовывать их все изначально, но заставляет очень тщательно оптимизировать код для того, чтобы он исполнялся достаточно быстро для реального времени.

Есть множество методов для ускорения трассировки, самые производительные алгоритмы трассировки лучей обрабатывают лучи не по одному, а используют наборы лучей, что ускоряет процесс обработки лучей одинакового направления. Такие оптимизации отлично подходят для исполнения на современных SIMD-блоках CPU и на GPU, они эффективны для основных сонаправленных лучей и для теневых лучей, но все равно не подходят для лучей преломления и отражения. Поэтому приходится серьезно ограничивать количество лучей, рассчитываемых для каждого пикселя сцены, а повышенную «шумность» картинки убирать при помощи специальной фильтрации.

В рамках данной работы были исследованы существующие программные и программно-аппаратные решения, а так же проведены эксперименты собственных решений на базе описанных технологий. Среди программных решений с использованием CPU и GPU рассмотрены OpenGL, OpenCL и CUDA.

OpenGL — открытая графическая библиотека, которая является одним из самых популярных прикладных программных интерфейсов для разработки приложений в области двумерной и трехмерной графики. На сегодняшний день графическая система OpenGL поддерживается большинством производителей аппаратных и программных платформ. Эта система доступна тем, кто работает в среде Microsoft Windows, пользователям компьютеров Apple, Unix-платформ, PlayStation. Свободно распространяемые коды системы Mesa можно компилировать в большинстве операционных систем, в том числе в Linux.

OpenCL — фреймворк для написания компьютерных программ, связанных с параллельными вычислениями на различных графических и центральных процессорах, а также FPGA. В OpenCL входят язык программирования, который основан на стандарте языка программирования Си, и интерфейс программирования приложений. OpenCL обеспечивает параллелизм на уровне инструкций и на уровне данных и является осуществлением техники GPGPU. OpenCL является полностью открытым стандартом, его использование не облагается лицензионными отчислениями.

CUDA — программно-аппаратная архитектура параллельных вычислений, которая позволяет существенно увеличить вычислительную производительность благодаря использованию графических процессоров фирмы Nvidia. CUDA SDK позволяет программистам реализовывать на специальных упрощённых диалектах языков программирования C, C++ и Фортран алгоритмы, выполнимые на графических и тензорных процессорах Nvidia. Архитектура CUDA даёт разработчику возможность по своему усмотрению организовывать доступ к набору инструкций графического или тензорного ускорителя и управлять его памятью. Функции, ускоренные при помощи CUDA, можно вызывать из различных языков, таких как Python, MATLAB и других.

В главе 2 рассмотрена реализация с использованием OpenGL без аппаратного ускорения и проведена оценка z-буфера. Для исследования оптимизаций с использование графического процессора были реализованы эксперименты с использованием OpenCL, CUDA, фрагментного и вычислительного шейдеров, которые более подробно описаны в главе 3.

Метод обладает значительным списком достоинств, некоторыми из которых не обладают стандартные методы растеризации:

— возможность рендеринга гладких объектов без аппроксимации их полигональными поверхностями;
— вычислительная сложность метода слабо зависит от сложности сцены;
— высокая алгоритмическая распараллеливаемость вычислений — можно параллельно и независимо трассировать два и более лучей, разделять участки для трассирования на разных узлах кластера и т.д;
— отсечение невидимых поверхностей, перспектива и корректное изменения поля зрения являются логическим следствием алгоритма.

В свою очередь серьёзным недостатком метода обратного трассирования является производительность. Метод растеризации и сканирования строк использует когерентность данных, чтобы распределить вычисления между пикселями. В то время как метод трассирования лучей каждый раз начинает процесс определения цвета пикселя заново, рассматривая каждый луч наблюдения в отдельности. Впрочем, это разделение влечёт появление некоторых других преимуществ, таких как возможность трассировать больше лучей, чем предполагалось для устранения контурных неровностей в определённых местах модели. Также это регулирует отражение лучей и эффекты прелом-

ления, и в целом — степень фотореалистичности изображения.

В данной проблеме существует различные готовые программные реализации среди открытого и проприетарного программного обеспечения.

Среди проприетарных выделяют:

— Arnold
— Brazil R/S
— BusyRay (плагин к 3DS MAX)
— finalRender
— Fryrender
— Gelato
— Holomatix Rendition (интерактивный рейтрейсер)
— Indigo Renderer
— Kerkythea
— Mantra (как часть пакета Houdini)
— Maxwell Render
— mental ray
— RenderMan (PhotoRealistic или PRMan)
— V-Ray
— bCAD
— SolidWorks

Основные программные решения с открытым исходным кодом:

— BRL-CAD
— Cycles (Blender)
— LuxRender
— Sunflow
— YafaRay
— POV-Ray

2 Постановка задачи эксперимента

В данной работе проведено сравнение производительности volume ray casting с использованием фрагментного шейдера, вычислительного шейдера, OpenCL и CUDA. В отличие от других исследований, в данной работе сравнивается более двух параллельных реализаций приведения лучей и учитывается использование вычислительного шейдера.

Объемный рендеринг (volume rendering) — это метод рендеринга трехмерного скалярного поля путем проецирования его выборок без необходимости промежуточной реконструкции данных. Этот метод моделирует взаимодействие между светом и участвующей средой (объемом) и моделируется с помощью уравнения 1.

$$C = \int_0^D c(x(\lambda))\tau(x(\lambda))e^{\int_0^\lambda \tau(x(\lambda'))d\lambda'}d\lambda \tag{1}$$

Значения $c(x(\lambda))$ и $\tau(x(\lambda))$ представляют цвет и поглощение образца на расстоянии λ, а $x(\lambda)$ представляет параметризацию луча. Есть несколько способов оценить уравнение рендеринга [**?**]. Как правило, луч квантуется в одинаково разнесенных выборках, и передающая функция применяется к каждой выборке для выполнения составной операции между классифицированными выборками. Приведение лучей оценивает непосредственно уравнение рендеринга объема путем приведения луча для каждого пикселя в конечном изображении с началом координат в положении глаза, как это может быть изображено на рисунке 1.

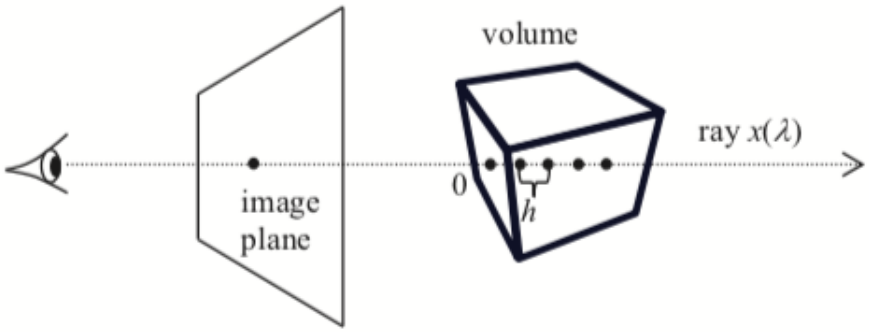

Рисунок 1 – Расчет цвета пикселя изображения по вкладу цвета и непрозрачности образцов эквидистантных лучей

Уравнение визуализации объема может быть приблизительно дискре-

тизировано, как показано в уравнении 2, где αi и ci — непрозрачность и цвет i-го образца соответственно.

$$C \approx \sum_{i=0}^{n-1} \prod_{j=0}^{i-1} (1 - \alpha_j)\alpha_i c_i \tag{2}$$

Поскольку метод объемного луча напрямую оценивает уравнение визуализации объема, он обычно дает лучшие визуальные результаты, чем другие подходы. Кроме того, поскольку расчет независим для каждого луча, процесс может быть легко распараллелен.

3 Оценка полученных результатов

В данной работе представлено сравнение производительности реализаций рендеринга томов с использованием фрагментного шейдера, вычислительного шейдера, OpenCL и CUDA. Реализации были протестированы с тремя наборами данных с различными разрешениями томов, от меньших томов (менее 40 МБ), до промежуточных томов (около 150 МБ), до больших томов (около 600 МБ). Каждый том был протестирован с различными передаточными функциями и размерами блоков для реализаций GPGPU (от 4 до 128 потоков на блок). Как и ожидалось, производительность наших реализаций лучевого литья уменьшается с большими объемами и с полупрозрачной передаточной функцией.

Для тестов заключаем, что реализация вычислительного шейдера представляет собой лучший вариант для реализации базового объемного лучевого кастера в целом. Вычислительный шейдер показывает лучшие результаты почти во всех тестируемых случаях, будучи в 1,03-2,9 раза быстрее, чем фрагментный шейдер, в 1,5-7,1 раза быстрее, чем OpenCL, и от 1,01 до 1,3 раза быстрее, чем CUDA. Следующий лучший вариант между фрагментным шейдером и CUDA. CUDA показывает лучшие результаты, чем фрагментный шейдер, почти во всех случаях больших и средних объемов, с улучшением скорости от 1,1 до 2,4. Принимая во внимание, что в меньших объемах нет глобального победителя между фрагментным шейдером и CUDA. Наконец, OpenCL имеет худшую производительность почти во всех случаях, что от 1,01 до 7,1 раза медленнее, чем у любого другого API. Единственное исключение — для томов с самым высоким разрешением, где OpenCL до 1,4 раза быстрее, чем фрагментный шейдер.

Кроме того, был проведен дополнительный тест для измерения накладных расходов на расчет диффузного освещения в каждом тестируемом объеме. Добавление подсветки делает время выполнения рендеринга в 2,8 раза медленнее, чем при обычном рендеринге, так как в нем больше вычислений и больше выборок текстур. Приращение времени выполнения для фрагментного шейдера составляет до 2,8, для вычислительного шейдера — до 1,7, для OpenCL — до 2,3, а для CUDA — до 1,7. Обратим внимание, что влияние вычислений освещения меньше для вычислительных шейдеров и CUDA, что согласуется с результатами без освещения, где они были также самыми быст-

рыми подходами.

Мы также обнаружили, что не существует глобального победителя с точки зрения времени выполнения между методами пересечения луча/объема (растеризация и луч/блок). Тем не менее, пересечение между лучом и блоком обеспечивает лучшую производительность для большинства случаев с вычислительным шейдером и OpenCL, в то время как тест пересечения растра представляет лучшую производительность для большинства случаев с CUDA. Таким образом, наилучшие общие результаты нашего теста были получены при комбинировании вычислительного шейдера с тестом пересечения луча/коробки.

В проведенных тестах обнаружено, что лучший размер блока зависит от размера тома. Для наборов данных большего объема наилучшим выбором является конфигурация, например, 4×4, 8×4 и 4×2. Стоит обратить внимание на конфигурации блоков 4×4 и 4×2, так как количество потоков меньше, чем размер основы CUDA (32 потока). Теоретически, при этих конфигурациях половина или более половины потоков основы будут простаивать, поэтому будет большой объем неиспользованной вычислительной мощности, которая должна быть неэффективной. Хотя размер тома становится больше, конфигурация с большим количеством потоков на блоки дает лучшие результаты. Например, 8×8 или 16×8 для томов размером до 150 МБ. Конфигурации блоков с соотношением $1 : 1$ или $1 : 2$ дают лучшие результаты, чем конфигурации с прямоугольными соотношениями сторон, что доказывает, что конфигурации с соотношением $1 : 1$ или $1 : 2$ обеспечивают больший объединенный доступ к текстуре тома и выполнение потока с меньшим расхождением. В конфигурациях блоков с прямоугольными пропорциями, поскольку потоки извлекают данные из удаленных областей тома, которые могут представлять разные материалы, возможно, что большое количество потоков выполняет раннее завершение луча, которое заканчивается большим числом незанятых потоков ожидание других потоков в том же блоке, чтобы закончить выполнение. Кроме того, для больших объемов лучшим вариантом является меньшее количество потоков на блок (например, 16 или 32), в то время как для меньших объемов лучшим вариантом является большее количество потоков на блок (например, 64 или 128). Это может быть связано с производительностью кеша, поскольку блоки больших размеров требуют одновременного доступа

к большим областям тома. Для небольших томов требуемые вокселы могут находиться в кеше графического процессора, тогда как в больших томах это сложнее для менеджера кэширования. Та же проблема производительности кэша может быть замечена в конфигурациях блоков с меньшим количеством потоков, чем размер деформации для больших томов. При больших томах каждый поток будет извлекать удаленные области томов, генерируя ошибки в кеше, что может привести к последовательному разрешению выборок. По этой причине может быть полезным даже количество потоков, меньших, чем размер деформации, поскольку для деформации можно будет избежать значительного пропуска кэша.

ЗАКЛЮЧЕНИЕ

На основе проделанных практических экспериментов были разработаны следующие рекомендации для программирования при использовании технологий с аппаратным ускорением вычислений:

— Математические вычисления над массивами лучше выполнять на графическом процессоре, так как они зачастую требуют мало памяти и в основном опираются на частоту процессора, с которой выполняется;

— В зависимости от количество памяти которой требуется для операций, которые перенаправляются на видеочип можно прогнозировать какое примерное количество памяти для оптимального выбора количества блоков и потоков, которые будут заняты;

— Желательно снизить количество использование констант в коде, выполняющемся на графическом процессоре, в следствии того, что константы находятся на отдельном чипе памяти на видеокарте и требует больше времени для каждого вызова, что может сильно сказаться на времени выполнения кода на видеочипе.

В рамках данной выпускной квалификационной работы была достигнута цель — исследовать возможные оптимизации трассировки лучей с использованием доступных программных и аппаратных средств, их анализа и оценки.

По завершению работы исследования были выполнены поставленные задачи:

1. Выполнена реализация трассировки лучей стандартными программными средствами;
2. Рассмотрены возможности улучшения базового алгоритма с использованием OpenMP;
3. Реализована трассировку лучей с использованием программных средств OpenGL;
4. Изучен Unified Memory фрейм буфер, в который пишет и читает CPU;
5. Написаны решения, запускаемые на CPU или GPU;
6. Проверена скорость работы режима точности в режиме double precision floating point;
7. Рассмотрено управление памятью в C++, выделение памяти на GPU и ее использование во время выполнения программы;

8. Проведена серия экспериментов для сравнения параметров работы решений на основе фрагментного, вычислительного шейдеров, OpenCL и CUDA;
9. Выработаны рекомендации по выбору параметров для OpenCL и CUDA.