МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Информационные технологии и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»

Курсовая работа
по курсу «Параллельная обработка данных»

Обратная трассировка лучей (Ray Tracing) на GPU

Выполнил: А.В. Синявский
Группа: 8О-408Б
Преподаватели: К.Г. Крашенинников,
А.Ю. Морозов

Москва, 2021

Условие

Цель работы.

Использование GPU для создания фотореалистичной визуализации. Рендеринг полузеркальных и полупрозрачных правильных геометрических тел. Получение эффекта бесконечности. Создание анимации.

Задание.

Сцена - прямоугольная текстурированная поверхность (пол), над которой расположены три платоновых тела. Сверху находятся несколько источников света. На каждом ребре многогранника располагается определенное количество точечных источников света. Грани тел обладают зеркальным и прозрачным эффектом. За счет многократного переотражения лучей внутри тела, возникает эффект бесконечности.

Камера выполняет облет сцены согласно определенным законам.

Требуется реализовать алгоритм обратной трассировки лучей с использованием технологии CUDA. Выполнить покадровый рендеринг сцены. Для устранения эффекта «зубчатости», выполнить сглаживание (например с помощью алгоритма SSAA). Полученный набор кадров склеить в анимацию любым доступным программным обеспечением. Подобрать параметры сцены, камеры и освещения таким образом, чтобы получить наиболее красочный результат. Провести сравнение производительности gpu и cpu (т.е. дополнительно нужно реализовать алгоритм без использования CUDA).

Вариант.

Вариант №10. Октаэдр, Додекаэдр, Икосаэдр

Входные данные.

Программа должна принимать на вход следующие параметры:

1. Количество кадров.
2. Путь к выходным изображениям. В строке содержится спецификатор %d, на место которого должен подставляться номер кадра. Формат изображений соответствует формату описанному в лабораторной работе 2.
3. Разрешение кадра и угол обзора в градусах по горизонтали.
4. Параметры движения камеры
5. Параметры тел: центр тела, цвет (нормированный), радиус (подразумевается радиус сферы в которую можно было бы вписать тело), коэффициент отражения, коэффициент прозрачности, количество точечных источников света на ребре.
6. Параметры пола: четыре точки, путь к текстуре, оттенок цвета и коэффициент отражения.
7. Количество (не более четырех) и параметры источников света: положение и цвет.
8. Максимальная глубина рекурсии и квадратный корень из количества лучей на один пиксель (для SSAA).

Программное и аппаратное обеспечение

Nvidia GeForce GTX 660

Compute capability: 3.0
Графическая память: 2048MB
Регистров на блок: 65536
Нитей на блок: 1024
Мультипроцессоров: 5
Всего ядер: 960

Intel(R) Core(TM) i5-3570 CPU @ 3.40GHz

Тактовая частота: 3.4 GHz
Кэш-память: 6 MB

Оперативная память

Объём: 8 GB

Жёсткий диск

Объём: 2 TB

Программное обеспечение

OS: Windows 10
IDE: Visual Studio Code
CUDA: v10.2 nvcc

Метод решения

Для каждого пикселя изображения будем рассчитывать луч, направленный от камеры в глубь сцены. При обнаружении пересечения луча с полигоном будем, во-первых, рассчитывать «теневой луч», направленный от места пересечения к источнику света, который будет определять освещённость пикселя, во-вторых, рекурсивно будем рассчитывать до 2-х дополнительных лучей – преломлённый (прошедший сквозь полупрозрачный полигон) и отражённый. Глубина рекурсии жёстко ограничена небольшим значением, чтобы нивелировать проблему дивергенции потоков на GPU. Итоговый цвет будет складываться из цвета полигона (текстуры если это пол), отражённого и преломлённого лучей с коэффициентами, соответствующими коэффициентам отражения и прозрачности у объектов.

Описание программы

Программа содержит множество примитивных функций, реализующих различные операции над векторами. Опишем наиболее значимые элементы программы

1. Get_from_hex
   Функция, принимающая на вход ссылку на текстуру, некоторые параметры текстуры и пола, а также координаты пересечения луча с полом в плоскости XY. Возвращает цвет текстуры в месте падения луча

2. Octahedron, dodecahedron, icosahedron
   Функции, генерирующие вершины и полигоны трёх платоновых тел

3. Build_space
   Функция, генерирующая полигоны пола и вызывающая предыдущие три, чтобы обобщить задание сцены вызовом одной функции.

4. ray
   Рекурсивная функция, рассчитывающая цвет пикселя при помощи трассировки лучей. Принцип работы функции подробнее описан выше, в методе решения задачи.

5. render
   Функция, генерирующая изображение сцены при заданном положении, направлении и угле обзора камеры. Рассчитывает для каждого пикселя направление первичного луча, и «выпускает» этот луч вызовом функции ray. Результирующий цвет пикселя сохраняется в соотв. Ячейку массива.

6. Gpu_render
   Функция аналогична предыдущей, но выполняется параллельно на GPU

7. ssaa
   Алгоритм сглаживания изображения, по сути считает среднее значения для множества пикселей, требует размер входного изображения больше, чем выходное в n раз.

8. Gpu_ssaa
   Функция аналогична предыдущей, но выполняется параллельно на GPU

9. main
   Основная функция, отвечающая за параметризацию всего, что можно параметризировать в программе, за выделение и очистку памяти, за запуск функции render для каждого кадра, собственно за расчет положения и направления камеры в каждом кадре, и наконец за сохранения каждого кадра в отдельный файл.

Результаты

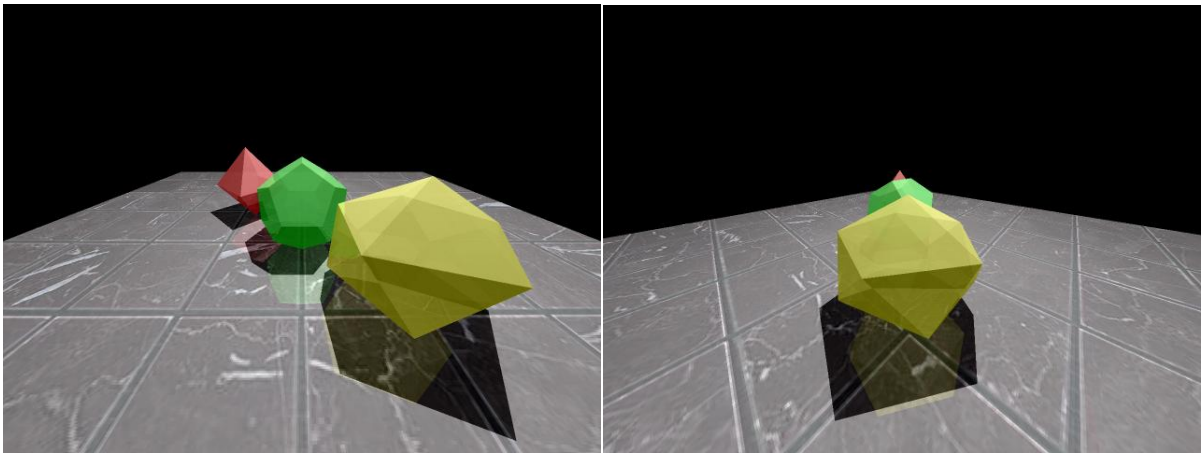
Конфигурация программы для тестирования:

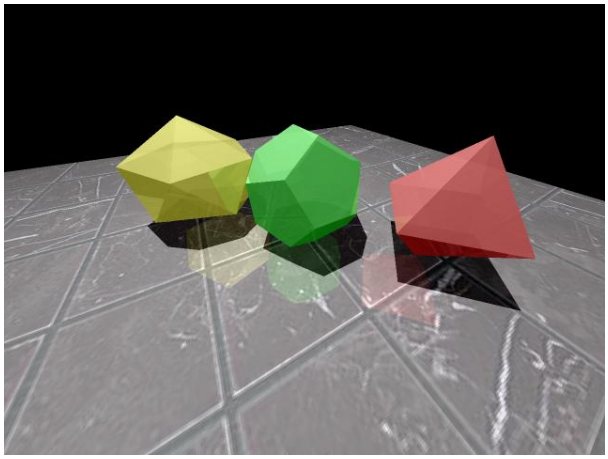
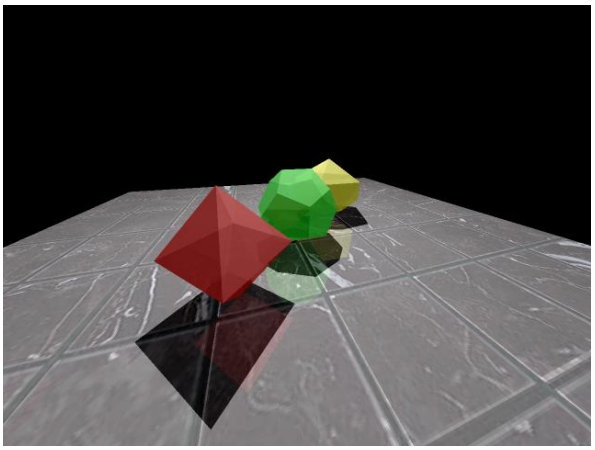
```
126
res/%d.data
640 480 120.000000
7.000000 5.000000 0.000000      2.000000 0.000000       2.000000 6.000000 1.000000      0.000000 0.000000
2.000000 2.000000 0.000000      0.000000 0.000000       1.000000 4.000000 1.000000      0.000000 0.000000
-3.000000 -3.000000 2.000000    0.500000 0.000000 0.000000      2.000000        0.250000        0.250000        0
0.000000 0.000000 2.000000      0.000000 0.500000 0.000000      2.000000        0.250000        0.250000        0
3.000000 3.000000 2.000000      0.500000 0.500000 0.000000      2.000000        0.250000        0.250000        0
1
0.000000 2.500000 15.000000     0.500000 0.500000 0.500000
5 2
```

Замеры времени.

Номер кадра	CPU	GPU
0	118766.210938ms	338.925537ms
1	113233.031250ms	333.602081ms
2	-	326.072693ms
3	-	315.488434ms

Визуализация результатов.



Выводы

Проделав работу, я познакомился с алгоритмом трассировки лучей, узнал для чего он применяется, и каковы его основные сильные стороны (кинематограф, 3д мультипликация, построение фотореалистичных либо просто красочных изображений). При реализации алгоритма я столкнулся с рядом проблем (вызовов), с которыми мне пришлось бороться. Во-первых, нужно было решить, как организовать рекурсию. Несмотря на то, что вариант «на 4» должен был быть без рекурсии, я всё же оставил примитивную рекурсию со счётчиком, чтобы иметь возможность достаточно легко изменять её глубину в случае проблем с производительностью, или же просто из любопытства. Во-вторых, задание точек для платоновых тел оказалось нетривиальной задачей. Точки для первых двух тел я нашёл в сети, а вот для третьего пришлось поломать голову над подобием школьной задачки по геометрии. В-третьих, наложение текстур при помощи текстурной памяти у меня не пошло, так как в моей реализации основная функция трассировки ray может быть вызвана как на CPU, так и на GPU, а в таких функциях работа с текстурной памятью в принципе запрещена стандартом. Даже если я как программист уверен, что обращения к текстурной памяти с CPU не будет, подобный код просто не скомпилируется. Возможных решений проблемы я нашёл 2: либо создавать отдельно аналоги функции ray для CPU и GPU, либо отказаться от использования текстурной памяти в пользу глобальной. Первый вариант намного лучше с точки зрения производительности, однако я выбрал второй, так как он требует в разы меньше усилий для реализации, и не перегружает и без того раздутый почти на 700 строк код.

Работа сначала показалась мне пугающе большой, но в итоге мною было затрачено 3 полных дня, хотя я думал, что потрачу не меньше недели. Работа мне понравилась, особенно приятно то, что на выходе получается действительно красивое изображение, которое можно ещё и изменять на свой вкус.