МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Компьютерные науки и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»

**Курсовая работа**
**по курсу «Программирование графических процессоров»**

# Обратная трассировка лучей (Ray Tracing) на GPU.

Выполнил: Ф.М. Шавандрин
Группа: 8О-408Б
Преподаватель: А.Ю. Морозов

Москва, 2022

**Условие**

**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.** Камера выполняет облет сцены согласно определенным законам. В цилиндрических координатах *(r, φ, z)*, положение и точка направления камеры в момент времени *t* определяется следующим образом:

$$r_c(t) = r_c^0 + A_c^r sin(\omega_c^r \cdot t + p_c^r)$$

$$z_c(t) = z_c^0 + A_c^z sin(\omega_c^z \cdot t + p_c^z)$$

$$\varphi_c(t) = \varphi_c^0 + \omega_c^\varphi t$$

$$r_n(t) = r_n^0 + A_n^r sin(\omega_n^r \cdot t + p_n^r)$$

$$z_n(t) = z_n^0 + A_n^z sin(\omega_n^z \cdot t + p_n^z)$$

$$\varphi_n(t) = \varphi_n^0 + \omega_n^\varphi t$$

где

$$t \in [0, 2\pi]$$

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).

**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 (коэффициенты в цилиндрических координатах из формул выше).
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).
Программа должна поддерживать следующие ключи запуска:

**--cpu** Для расчетов используется только центральный процессор

**--gpu** Для расчетов задействуется видеокарта

**--default** В stdout выводится конфигурация входных данных (в формате описанном ранее), при которой получается наиболее красочный результат, после чего программа завершает свою работу.

**Выходные данные:** В процессе работы программа должна выводить в stdout статистику в формате: {номер кадра}\t{время на обработку кадра в миллисекундах}\t{общее количество лучей}\n.

**Цель работы:** Использование GPU для создание фотореалистической визуализации. Рендеринг полузеркальных и полупрозрачных правильных геометрических тел. Получение эффекта бесконечности. Создание анимации.

**Вариант 9.** На сцене должны располагаться три тела: Гексаэдр, Додекаэдр, Икосаэдр. Без рекурсии, без текстур, без отражений (без эффекта бесконечности), простые модели без отдельных ребер с источниками света, один источник света.

**Программное и аппаратное обеспечение**

**GPU:**

- Название NVIDIA GeForce GT 545
- Compute capability: 2.1
- Графическая память: 3150381056
- Разделяемая память: 49152
- Константная память: 32768
- Количество регистров на блок: 32
- Максимальное количество нитей: (1024, 1024, 64)
- Максимальное количество блоков: (65535, 65535, 65535)
- Количество мультипроцессоров: 3

**Сведения о системе:**

- Процессор: Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz
- ОЗУ: 15 ГБ
- HDD 500 ГБ

**Программное обеспечение:**

- OS: Ubuntu 16.04.6 LTS
- Текстовый редактор: Vim
- Компилятор: nvcc

**Метод решения**

Сцена и геометрические фигуры задаются в виде массива полигонов. Координаты вершин фигур и их полигонов высчитываются по математическим формулам.

Для реализации трассировки лучей будем искать первый полигон, которой он пересечет. Таким образом, цвет пересеченного полигона будет соответствовать цвету соответствующего пикселя на экране. Для нахождения этого первого полигона, который пересечет луч, будем использовать формулы из линейной алгебры:

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{dot\ (P, E1)} * \begin{bmatrix} dot\ (Q, E2) \\ dot\ (P, T) \\ dot\ (Q, D) \end{bmatrix}$$

$$E1 = v1 - v0$$

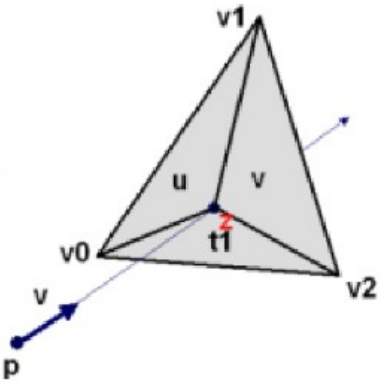
$$E2 = v2 - v0$$

$$T = p - v0$$

$$P = cross\ (D, E2)$$

$$Q = cross\ (T, E1)$$

$$D = v$$




Реализовал один источник света следующим образом: для каждого луча необходимо проверить, есть ли луч от источника света к первому полигону для этого луча. Если нет, то это значит, что мы находимся за объектом (в тени), следовательно, затемняем значение данного пикселя.

В качестве сглаживания использовал алгоритм SSAA(Super Sampling Anti-Aliasing), который заключается в том, что мы выполняем трассировку лучей для изображения большего разрешения, затем для изображения исходного разрешения вычисляется значение каждого пикселя как среднее значение соседних пикселей большего изображения.

Программа параллельно выполняет рендеринг и сглаживание изображения для нескольких пикселей.

**Описание программы**

Реализовал классы *polygon* для определения полигона и *vector* для определения вектора в трехмерном пространстве, а также функции для вычисления полигонов заданных вариантом геометрических фигур, математических операций над векторами, печати входных данных для наиболее красочного ответа и обратной трассировки лучей с учетом освещения.

Также реализовал функции для рендеринга изображения и сглаживания SSAA на CPU и GPU для сравнения результатов работы.

**Результаты**

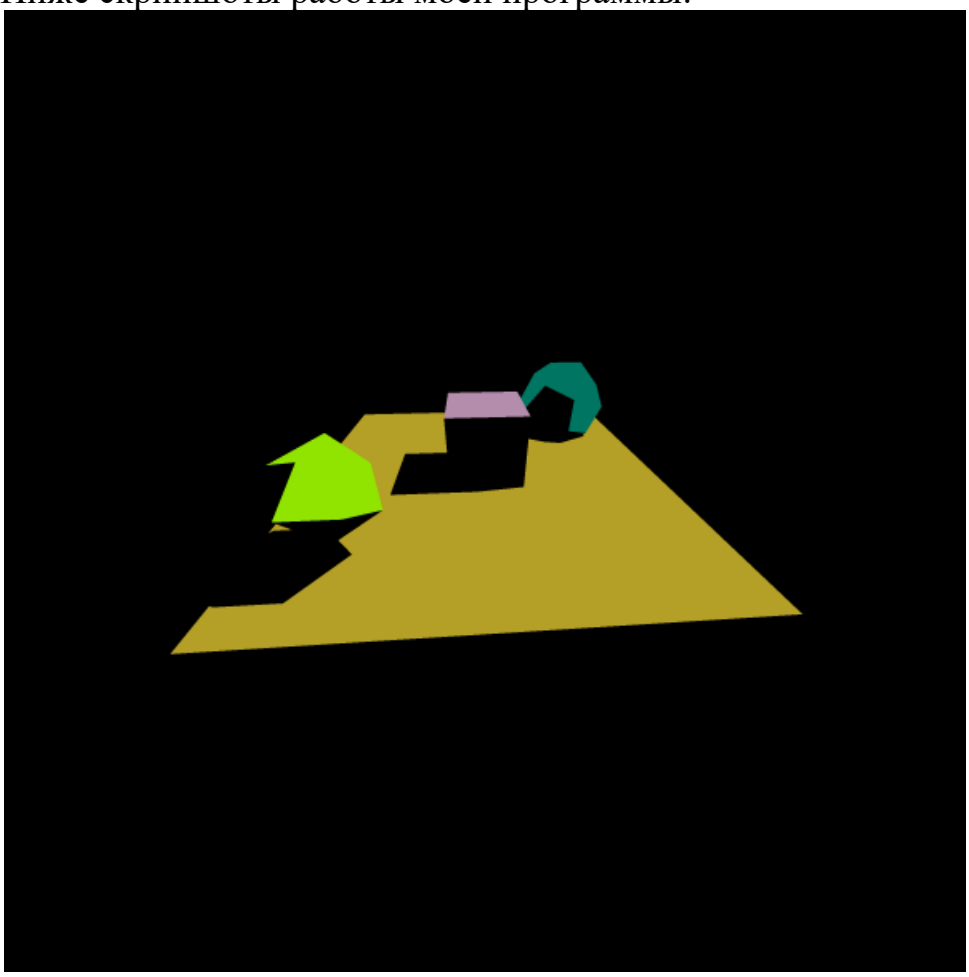
Для тестирования программы будем замерять время рендеринга одного кадра на CPU и GPU для разного количества лучей (будем менять разрешение изображения).

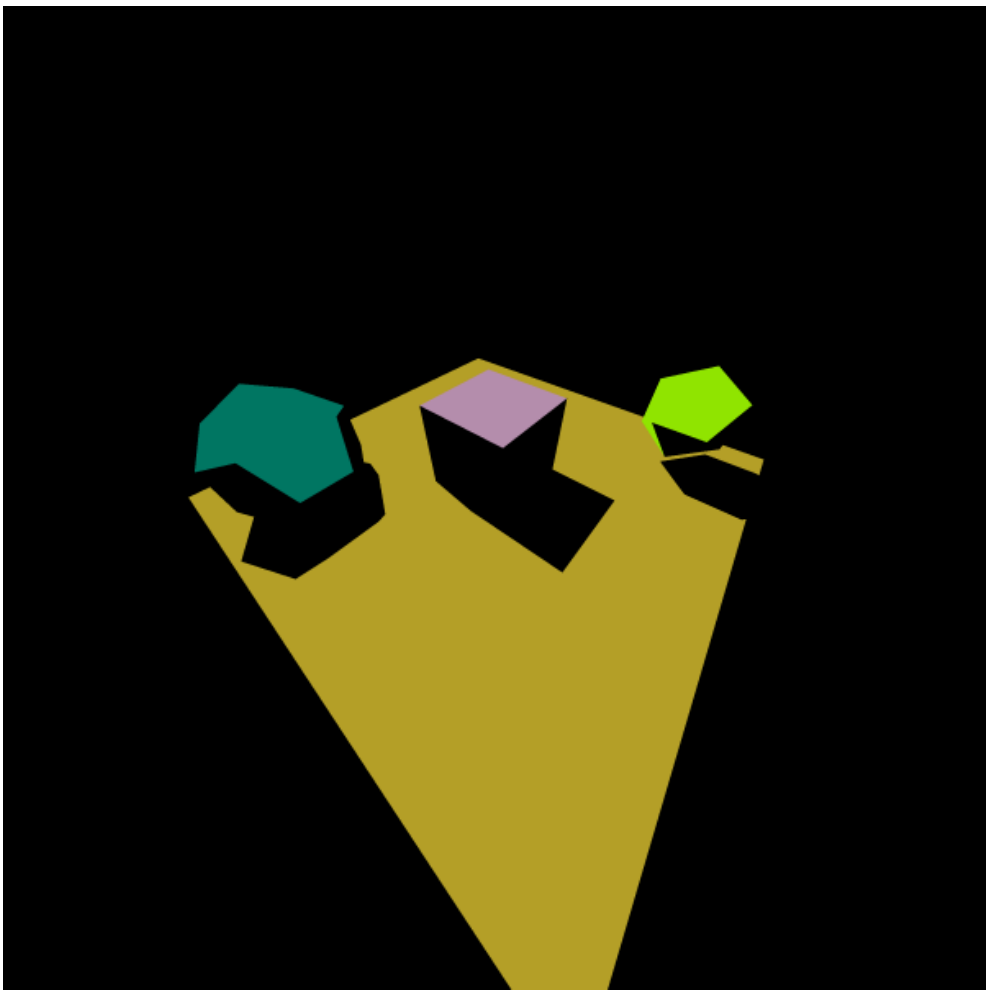
| Количество лучей | Рендеринг 1 кадра на GPU, мс | Рендеринг 1 кадра на CPU, мс |
|---|---|---|
| 1600 | 1.05536 | 12.35180 |
| 16000 | 4.24182 | 121.74000 |
| 160000 | 26.36080 | 669.63300 |
| 1600000 | 300.36400 | 7343.04000 |
| 16000000 | 2454.72000 | 59777.50000 |

Как видно из таблицы, рендеринг на GPU выполняется в десятки раз быстрее, чем на CPU.

**Результат работы программы**

Ниже скриншоты работы моей программы:





**Выводы**

В данной курсовой работе познакомился с алгоритмом обратной трассировкой лучей, который до сих пор успешно применяется при разработки игр или мультфильмов и попробовал самостоятельно его реализовать.

Также поближе познакомился с алгоритмом сглаживания SSAA, который несмотря на появление на сегодняшний день большого количества других типов сглаживаний, до сих пор является самым качественным методом, устраняющим искажения.

На примере своей курсовой работы, а именно при сравнении времени рендеринга кадров на видеокарте и процессоре, в очередной раз убедился, что видеокарты гораздо лучше использовать для подобного рода задач.