МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

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

Курсовая работа
по курсу «Параллельная обработка данных»

Обратная трассировка лучей (Ray Tracing) на GPU.

Выполнил: И. П. Моисеенков
Группа: М8О-408Б-19
Преподаватель: А.Ю. Морозов

Москва, 2022

Условие

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

Вариант 2. Тетраэдр, гексаэдр, додекаэдр.

Сцена. Прямоугольная поверхность (пол), над которой расположены три платоновых тела. Сверху находятся источники света.

Камера. Камера выполняет облет сцены согласно следующим законам. В цилиндрических координатах положение и направление камеры определяется как:

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

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

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

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

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

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

Требуется реализовать алгоритм обратной трассировки лучей с использованием CUDA. Выполнить покадровый рендеринг сцены. Для устранения эффекта «зубчатости» выполнить сглаживание (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 принимает на вход следующие параметры:

1. Количество кадров
2. Путь к выходным изображением (строка со спецификатором %d)
3. Разрешение экрана и угол обзора в градусах по горизонтали
4. Параметры движения камеры (коэффициенты из формул выше)
5. Параметры трех тел: центр тела, цвет тела (нормированный), радиус (подразумевается радиус описанной сферы)
6. Параметры пола: четыре точки, оттенок цвета
7. Параметры источника света: положение и цвет (нормированный)

8. Квадратный корень из количества лучей на один пиксель для SSAA.

Выходные данные. В соответствующие файлы нужно записать полученные картинки в бинарном виде (как было в лабораторных работах).

Программное и аппаратное обеспечение

В качестве графического процессора использую видеокарту Nvidia GeForce GT 545, установленную на сервере преподавателя.

```
Compute capability      : 2.1
Name                    : GeForce GT 545
Total Global Memory     : 3150381056
Shared memory per block : 49152
Registers per block     : 32768
Warp size               : 32
Max threads per block   : (1024, 1024, 64)
Max block   : (65535, 65535, 65535)
Total constant memory   : 65536
Multiprocessors count   : 3
```

В качестве редактора кода использовался Visual Studio Code.

Метод решения

Сцена. Сцена и все объекты на ней задаются как массив полигонов. Координаты вершин фигур и сами полигоны высчитывались по математическим формулам для всех фигур.

Трассировка лучей. Для каждого луча (а их количество равняется произведению размерностей экрана) мы ищем первый полигон («треугольник»), которой он пересечет. Цвет этого полигона и будет соответствовать цвету соответствующего пикселя на экране. Поиск пересечения осуществляется с помощью механизмов линейной алгебры:

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

$$E1 = v1 - v0$$

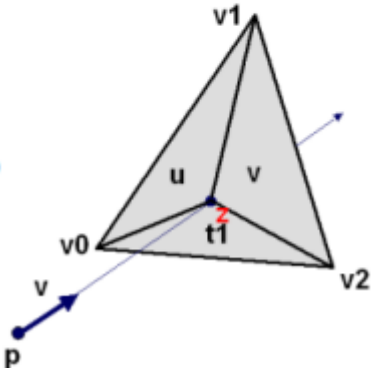
$$E2 = v2 - v0$$

$$T = p - v0$$

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

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

$$D = v$$



Освещение. В своей работе я реализовал поддержку только одного источника освещения. Чтобы учесть его, мы для каждого луча проверяем, есть ли луч от источника к первому полигону для этого луча. Если нет, то мы находимся в тени, следовательно затемняем значение данного пикселя.

Сглаживание. Я использовал алгоритм сглаживания SSAA. Этот алгоритм заключается в том, что мы расширяем исходную картинку в несколько раз (например, в 4) и выполняем трассировку всех лучей расширенной картинки. Далее возвращаемся к исходному размеру, считая что значение пикселя есть среднее значение соседних пикселей в расширенной картинке.

Параллельные вычисления. Рендеринг изображений и сглаживание можно выполнять параллельно для нескольких пикселей. Кажется, что можно дополнительно распараллелить еще и саму трассировку лучей - параллельно рассматривать несколько полигонов сразу. Но реализация такого параллельного рей трейсинга была бы очень сложной.

Описание программы

В данной работе я реализовал два ядра для вычислений на видеокарте - параллельный рендеринг и параллельное сглаживание. Для сравнения производительности я реализовал те же функции на cpu.

В программе имеются функции для:

- определения полигонов заданных геометрических тел
- обратной трассировки луча (с учетом освещения)
- рендеринга изображения (на cpu и на gpu)
- сглаживания (на cpu и на gpu)
- вспомогательных математических операций над векторами
- получением входных данных для наиболее красочного ответа

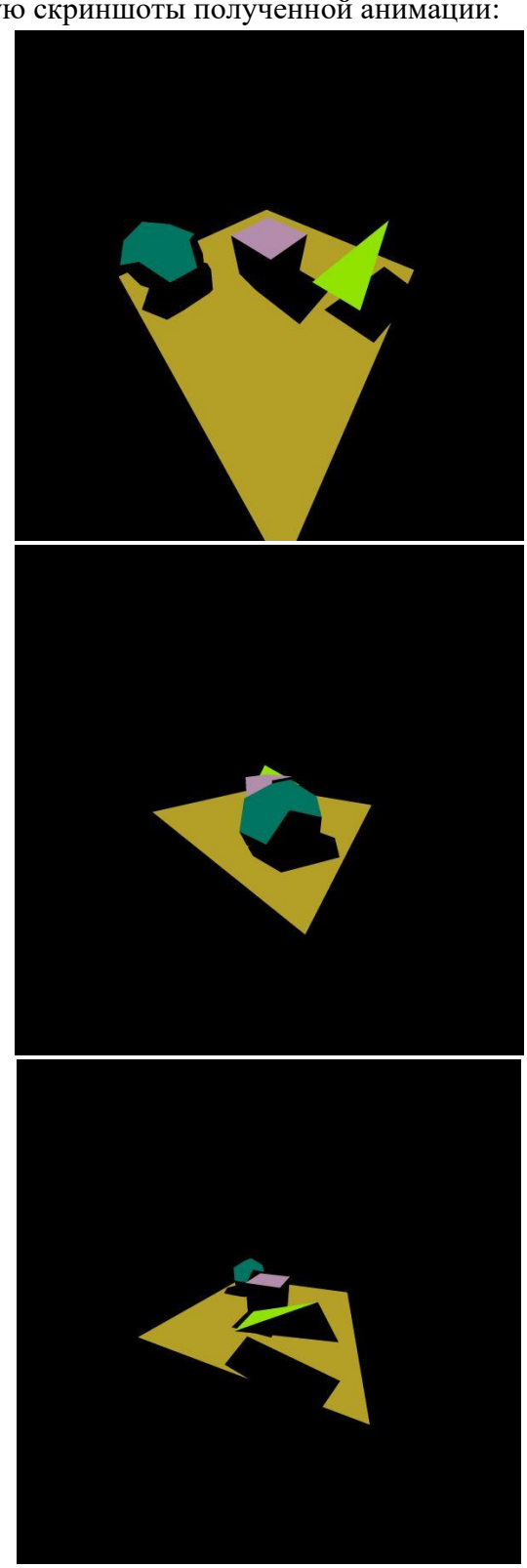
Исследовательская часть и результаты

Сравним время рендеринга одного кадра с использованием графического процессора и без него. Будем рассматривать одну и ту же сцену, изменять будем количество лучей (т.е. размерность изображения). Замерим время рендеринга одного кадра. В сравнении я не буду учитывать время, затраченное на копирование данных на/с видеокарты.

Количество лучей (пикселей в картинке)	Рендеринг 1 кадра на GPU, мс	Рендеринг 1 кадра на CPU, мс
1600	1.0	10.7
16000	3.0	68.2
160000	21.4	471.8
1600000	237.5	5565.5
16000000	1922.3	46568.0

При использовании видеокарт мы выигрываем по времени в десятки раз! Даже на небольших картинках. Это еще раз демонстрирует всю прелесть видеокарт и их возможностей.

Продемонстрирую скриншоты полученной анимации:



Данный результат получается при следующих входных данных:

```
100
res/%d.data
600 600 120

7.0 3.0 0.0     2.0 1.0     2.0 6.0 1.0     0.0 0.0
2.0 0.0 0.0     0.5 0.1     1.0 4.0 1.0     0.0 0.0
```

```
3.0 3.0 0.5    0.3 0.55 0.0     1.0
0.0 0.0 0.0     0.5 0.25 0.55     1.75
-3.0 -3.0 0.0     0.0 0.7 0.7     1.5

-5.0 -5.0 -1.0     -5.0 5.0 -1.0    5.0 5.0 -1.0    5.0 -5.0 -1.0   1.0 0.9
0.25

-10.0 0.0 15.0     0.3 0.2 0.1
4
```

Выводы

В курсовой работе я разбирался с алгоритмом обратной трассировки лучей и попробовал реализовать его самостоятельно с использованием параллельных вычислений на графическом процессоре.

Этот алгоритм очень широко используется при создании мультимедийного контента (игр, мультфильмов итд). Хотя в его основе лежит довольно простая математика.

Я убедился, что использование gpu во много раз ускоряет рендеринг изображений. Для таких задач видеокарты становятся просто необходимыми.

Сама работа была достаточно объемной, но интересной. К сожалению, из-за нехватки времени мне не удалось реализовать все заданные условия и получить красивую картинку с эффектом бесконечности. Но тем не менее мне понравился полученный мною результат.