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: Семин А. В.
Группа: 8О-406Б
Преподаватели: К.Г. Крашенинников,
А.Ю. Морозов

Москва, 2023

Условие.

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

Вариант 1. Тетраэдр, Гексаэдр, Октаэдр.

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 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 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 принимает на вход следующие параметры:

1. Количество кадров

2. Путь к выходным изображениям (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

Графический процессор (GeForce GTX 1650 Ti)

1. Количество потоковых процессоров: 1024
2. Частота ядра: 1350 МГц
3. Частота в режиме Boost: 1485 МГц
4. Количество транзисторов: 6,600 млн
5. Тип памяти: DDR6
6. Видеопамять: 4096 МБ
7. Частота памяти: 12000 МГц

Процессор AMD Ryzen 7 4800H

1. ядра: 8
2. потоки: 16
3. частота: *2.9 ГГц*
4. максимальная частота: *4.2 ГГц*
5. кэш 1 уровня: *64 КБ (на ядро)*
6. кэш 2 уровня *512 КБ (на ядро)*
7. кэш 3 уровня: *8 МБ (общий)*

16 ГБ ОЗУ и 512 ГБ SSD.
OS – Windows 11Домашняя, WSL, IDE – VS Code, Compiler - nvcc, g++.

Метод решения

Решение начинается с инициализации сцены и всех ее объектов. Они представлены в виде массива полигонов, где координаты вершин фигур и сами полигоны вычисляются в отдельных функциях на основе математических правил.

Трассировка лучей выполняется для каждого из лучей. Количество лучей равно произведению размерностей экрана. Каждый луч ищет первый пересекаемый им полигон. Цвет полигона определяет цвет соответствующего пикселя на экране. Поиск пересечения осуществляется с применением методов линейной алгебры.

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

$$E1 = v1 - v0$$

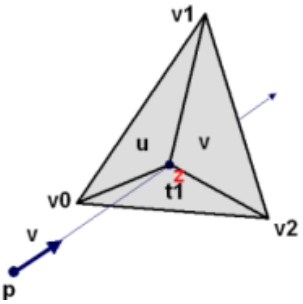
$$E2 = v2 - v0$$

$$T = p - v0$$

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

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

$$D = v$$



Реализована поддержка одного источника освещения. Каждый луч проверяет наличие луча от источника к первому пересекаемому им полигону. Соответственно, если луча нет, то пиксель находится в тени, и его значение затемняется.

Для сглаживания реализован алгоритм SSAA следующим образом: исходное изображение расширяется в несколько раз (для быстрой генерации кадров лучше использовать небольшое значение, например, 4), затем производится трассировка всех лучей для расширенного изображения. После происходит возврат к исходному размеру, усредняя значения пикселей на расширенном изображении.

Рендеринг изображений и сглаживание выполняются параллельно для нескольких пикселей при выполнении программы с использованием GPU.

Описание программы

Входные данные считываются из стандартного потока. На их основе инициализируются размеры изображения, параметры камеры, фигур,

сцены, источника света и сглаживания. При запуске программы можно задать различные ключи:
«--default» позволяет получить стандартный набор данных для ввода в программу;
«--gpu» (или его отсутствие) дает указание программе на выполнение вычислений с использованием мощностей GPU;
«--cpu» дает указание на использование только CPU мощностей (т. е. без GPU);
Для выполнения вычислений на GPU реализованы два ядра. Их сигнатуры:
__global__ void kernel_render(uchar4 *data, Polygon *polygons, double3 cam_pos, double3 cam_view, int w, int h, double angle, double3 lpos, uchar4 lcol)

__global__ void kernel_ssaa(uchar4 *data, uchar4 *data_out, int w, int h, int sqrtSamples)

Они выполняют рендеринг и сглаживание изображения соответственно. Также реализованы их аналоги для проверки работоспособности программы без использования мощностей видеокарты.

Результаты

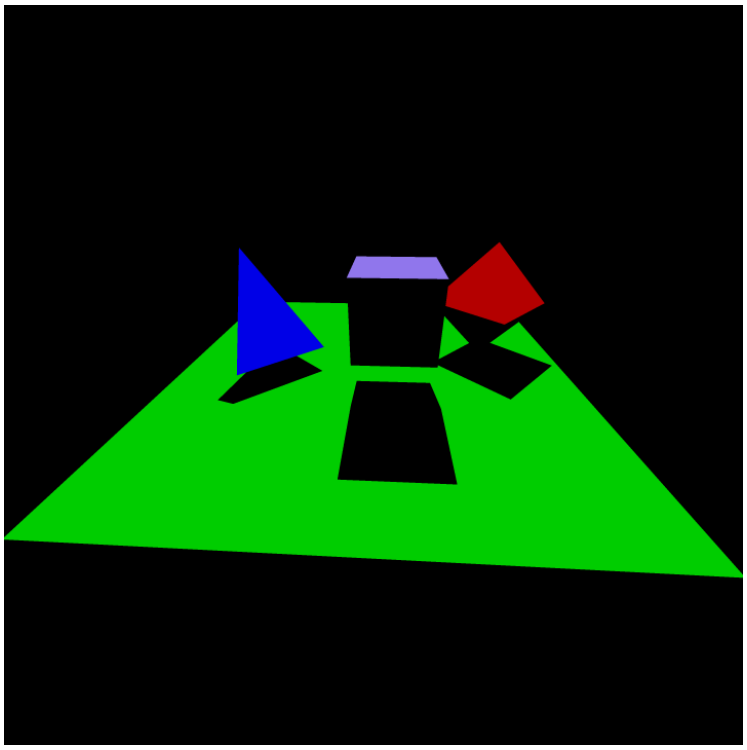
Сравнение происходит для одинаковых входных параметров за исключением размера изображения. Подсчет времени происходит только на этапах рендеринга и выполнения сглаживания. В качестве времени используется примерное среднее значение на протяжении создании изображений.

Разрешение изображения	Рендеринг кадра на GPU, *мс*	Рендеринг кадра на CPU, *мс*
1000	1.8	18
10000	7	178
100000	50	1818
1000000	490	18765
10000000	4869	195589

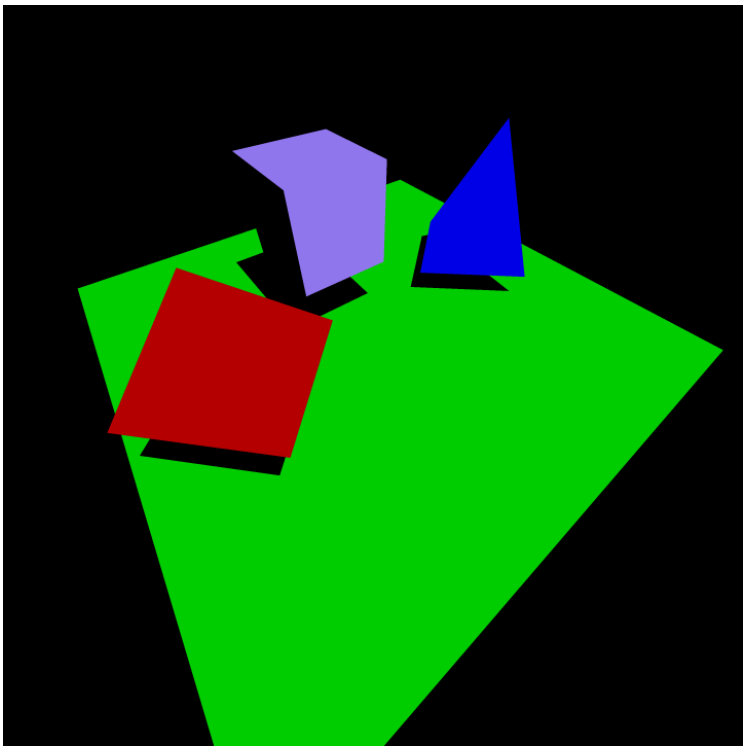
На полученных результатах видим, что использование мощности видеокарты позволяет получить выигрыш по времени в десятки-сотни раз даже на небольших картинках.

Примеры изображений, полученных в результате выполнения программы:

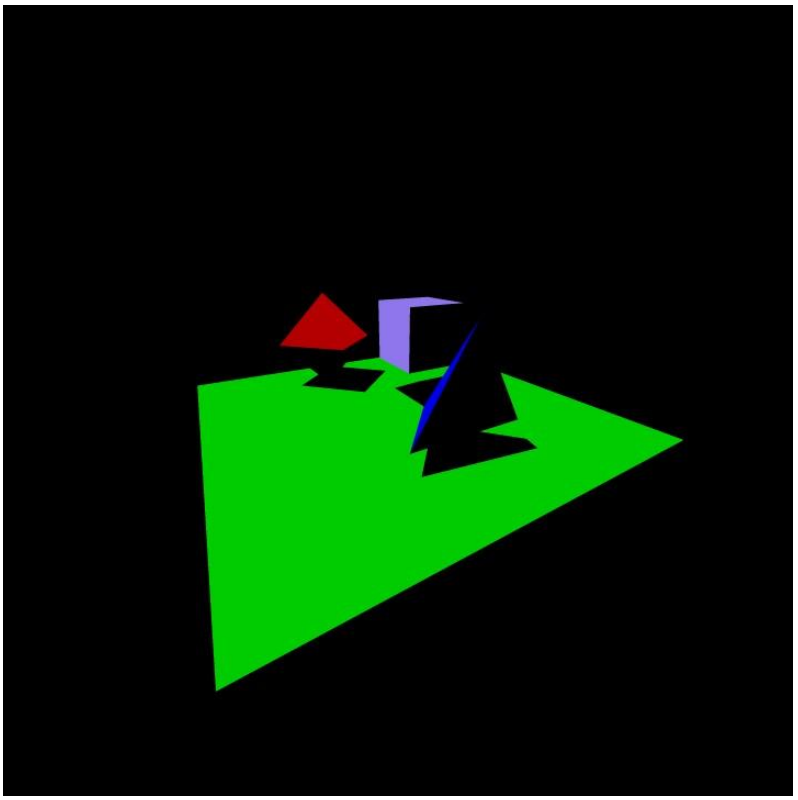
1 кадр



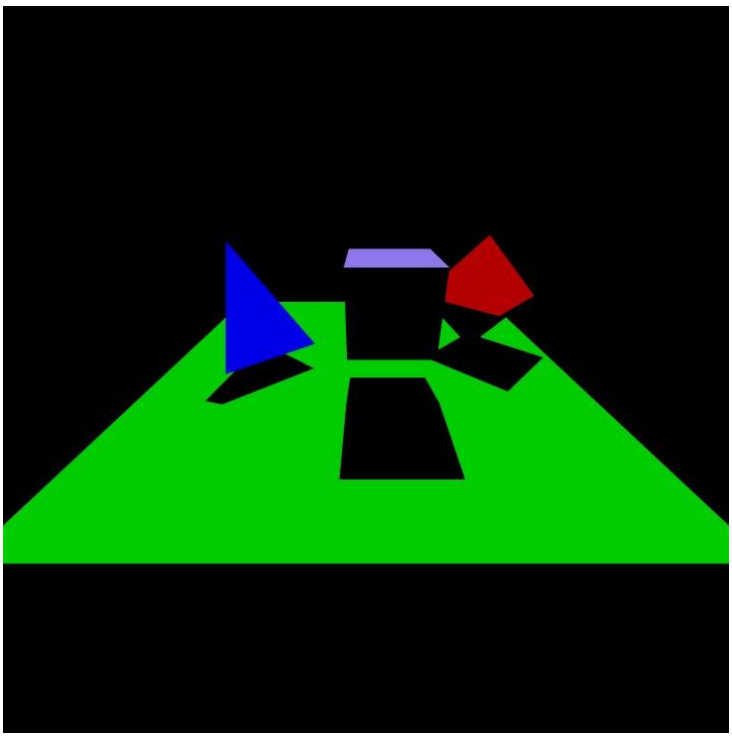
50 кадр



84 кадр



126 кадр



Входные данные:

```
126
output/img_%d.data
720 720 100
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
0 -3 0.3 0 0 1        1.7
1 0.5 1      0.3 0.7      0.8     1
-1 3 0.5    1 0 0      1.5
-5 -5 -1    -5.0 5.0 -1.0    5.0 5.0 -1.0    5.0 -5.0
-1.0  0.0 1.0 0.0
-10.0 0.0 20.0     0.3 0.2 0.1
4
```

Выводы

В ходе выполнения курсовой работы я реализовал стандартный алгоритм обратной трассировки лучей как с использованием параллельных вычислений на GPU, так и линейных на CPU. Данный алгоритм широко распространен в наши дни в сфере мультимедии. На основе полученных результатов мы можем точно убедиться, что использование параллельных вычислений на GPU значительно (в десятки и сотни раз) ускоряет выполнение программы, ведь именно для таких задач видеокарты и были разработаны.