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. О. Тояков
Группа: М8О-407Б-18
Преподаватели: К. Г. Крашенинников,
А. Ю. Морозов

Москва, 2022 г.

УСЛОВИЕ

Цель работы: Использование GPU для создание фотореалистической визуализации. Рендеринг полузеркальных и полупрозрачных правильных геометрических тел. Получение эффекта бесконечности. Создание анимации.
Вариант № 4: Тетраэдр, Октаэдр, Додекаэдр.

ПРОГРАММНОЕ И АППАРАТНОЕ ОБЕСПЕЧЕНИЕ

Device: GeForce MX250

Размер глобальной памяти: 3150381056

Размер константной памяти: 65536

Размер разделяемой памяти: 49152

Регистров на блок: 32768

Максимум потоков на блок: 1024

Количество мультипроцессоров: 3

OS: Linux Ubuntu 18.04

Редактор: VSCode

Компилятор: nvcc версии 11.4 (g++ версии 7.5.0)

МЕТОД РЕШЕНИЯ

Мы моделируем сцену из 4 объектов: пола и 3-ёх геометрических фигур, которые в свою очередь состоят из полигонов – треугольников. Необходимо отрендерить определённое количество кадров со сценой, используя обратную трассировку лучей, а затем готовые картинки объединить в анимацию. Чтобы применить ray tracing необходимо из каждого пикселя выпустить луч и вычислить его отражение, преломление и взаимодействие с объектами на сцене.

ОПИСАНИЕ ПРОГРАММЫ

void build_space() – построение сцены и отрисовка всех полигонов.

uchar4 ray() – рассчёт цвета заданного пикселя по траектории луча.

void ssaa() – алгоритм сглаживания для устранения зубчатости.

void render() – непосредственно рендер кадра с вычислением начальных значений сдвига камеры и вызовом ray tracing функции.

int main() – отвечает за переключение между режимами выполнения (CPU/GPU), ввод и вывод данных.

ИССЛЕДОВАТЕЛЬСКАЯ ЧАСТЬ

Так как я выполнял задание на оценку три (без рекурсии, отражений и источник света один), я приведу сравнение времени работы только на разных конфигурациях ядер. Конфигурации для ядра рендера и ядра фильтра ssaa одинаковы.

GPU:

Конфигурации ядер	Общее время работы	Среднее время на обработку одного кадра
<<<dim3(8, 8), dim3(8, 8)>>>	14707.1 ms	122.559 ms
<<<dim3(4, 16), dim3(4, 16)>>>	14419.2 ms	120.16 ms
<<<dim3(32, 32), dim3(32, 32)>>>	13863.3 ms	115.527 ms
<<<dim3(8, 32), dim3(8, 32)>>>	12725.8 ms	106.048 ms

CPU:

Общее время работы: 2616500 ms

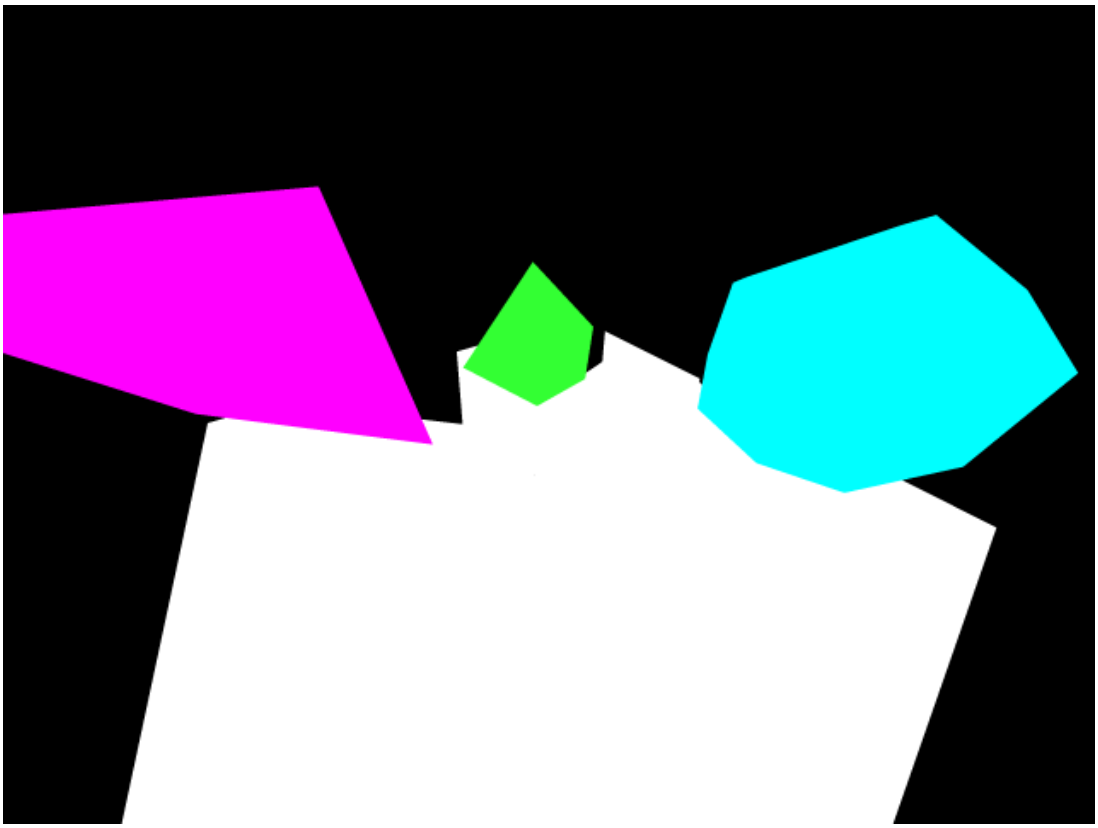
Среднее время на обработку одного кадра: 22013.8 ms

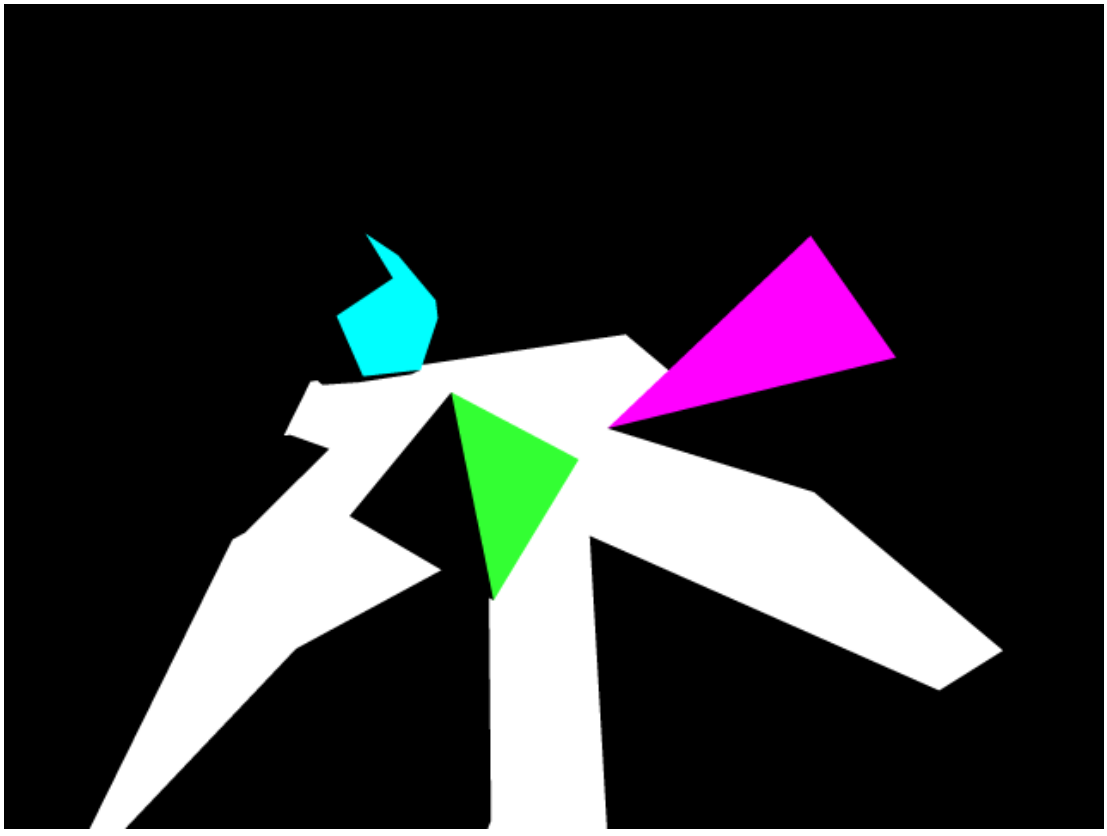
Таким образом ускорение $S = T1 / Tn = 2616500 / 14707.1 \approx 178$

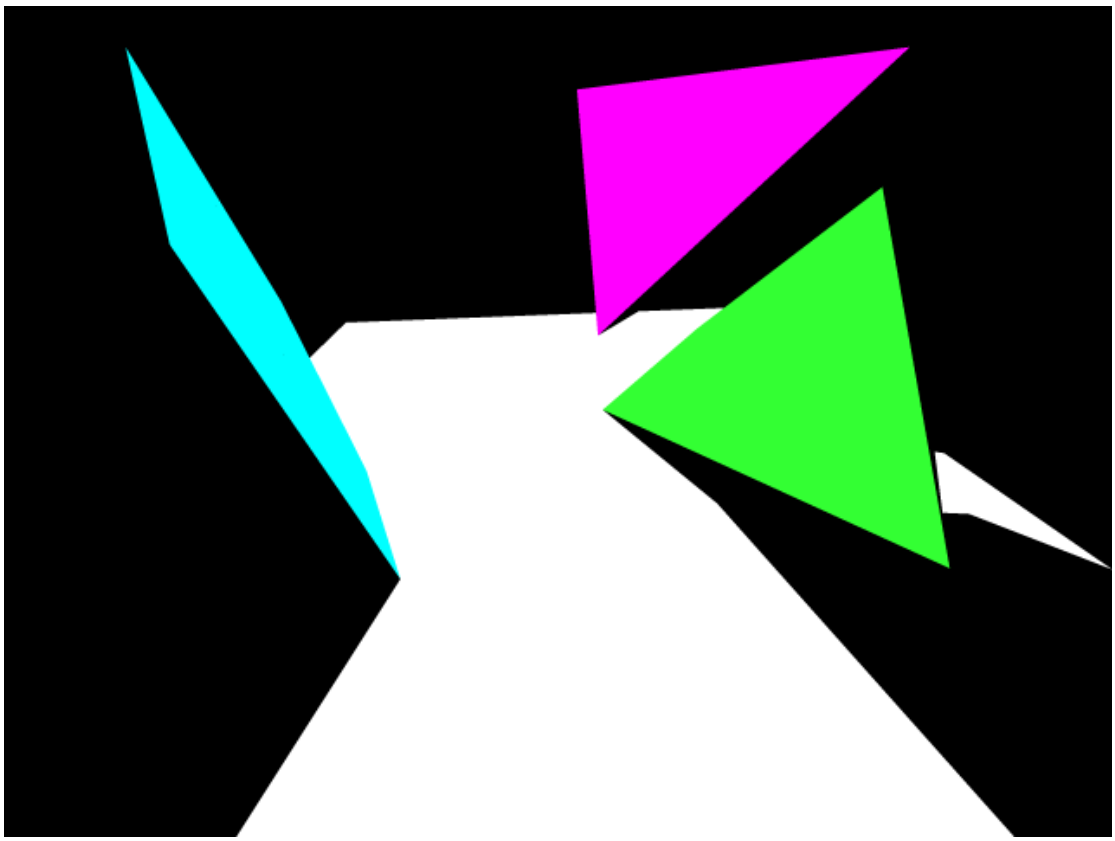
Коэффициент распараллеливания $P = S / N = 178 / 3 \approx 60$

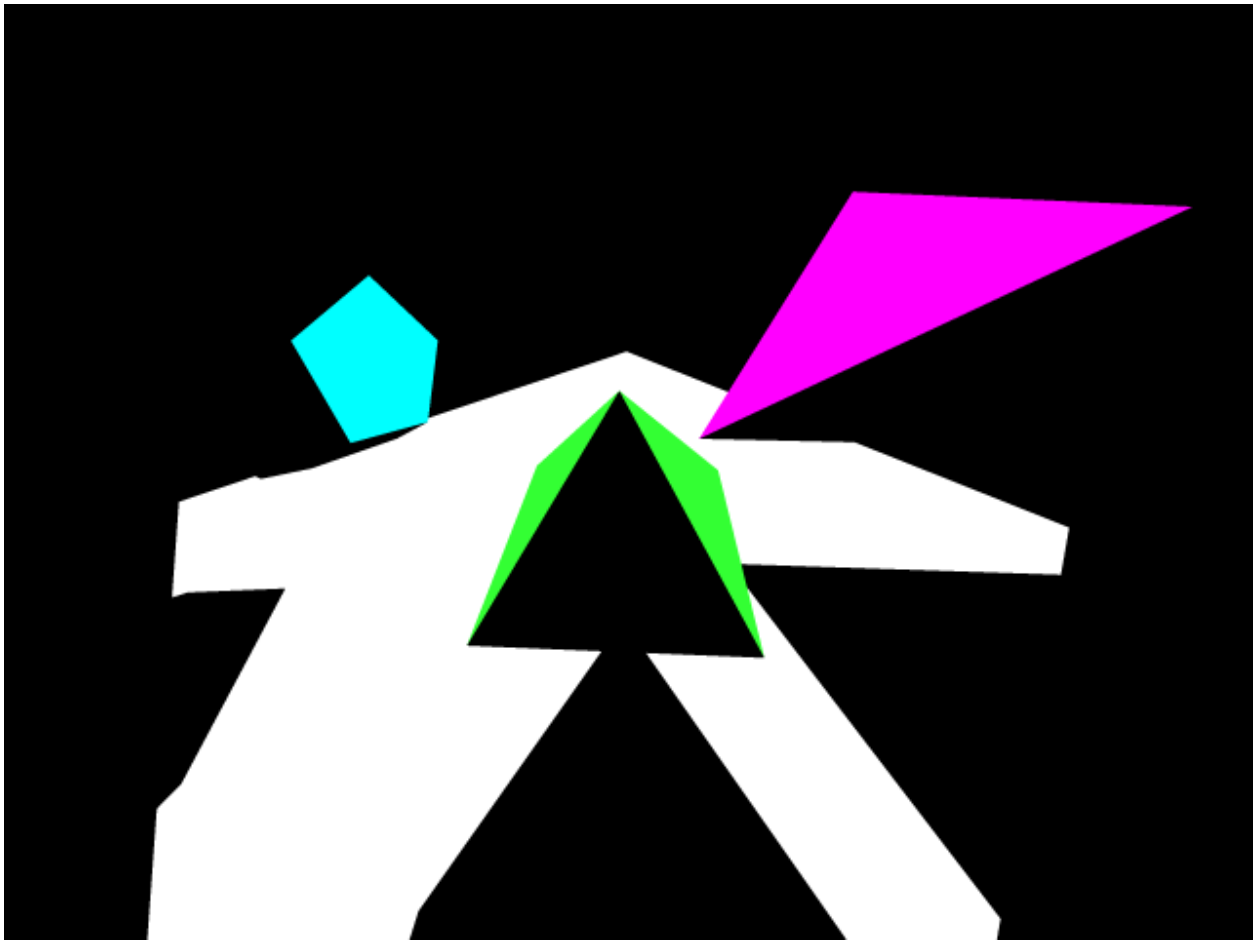
РЕЗУЛЬТАТЫ

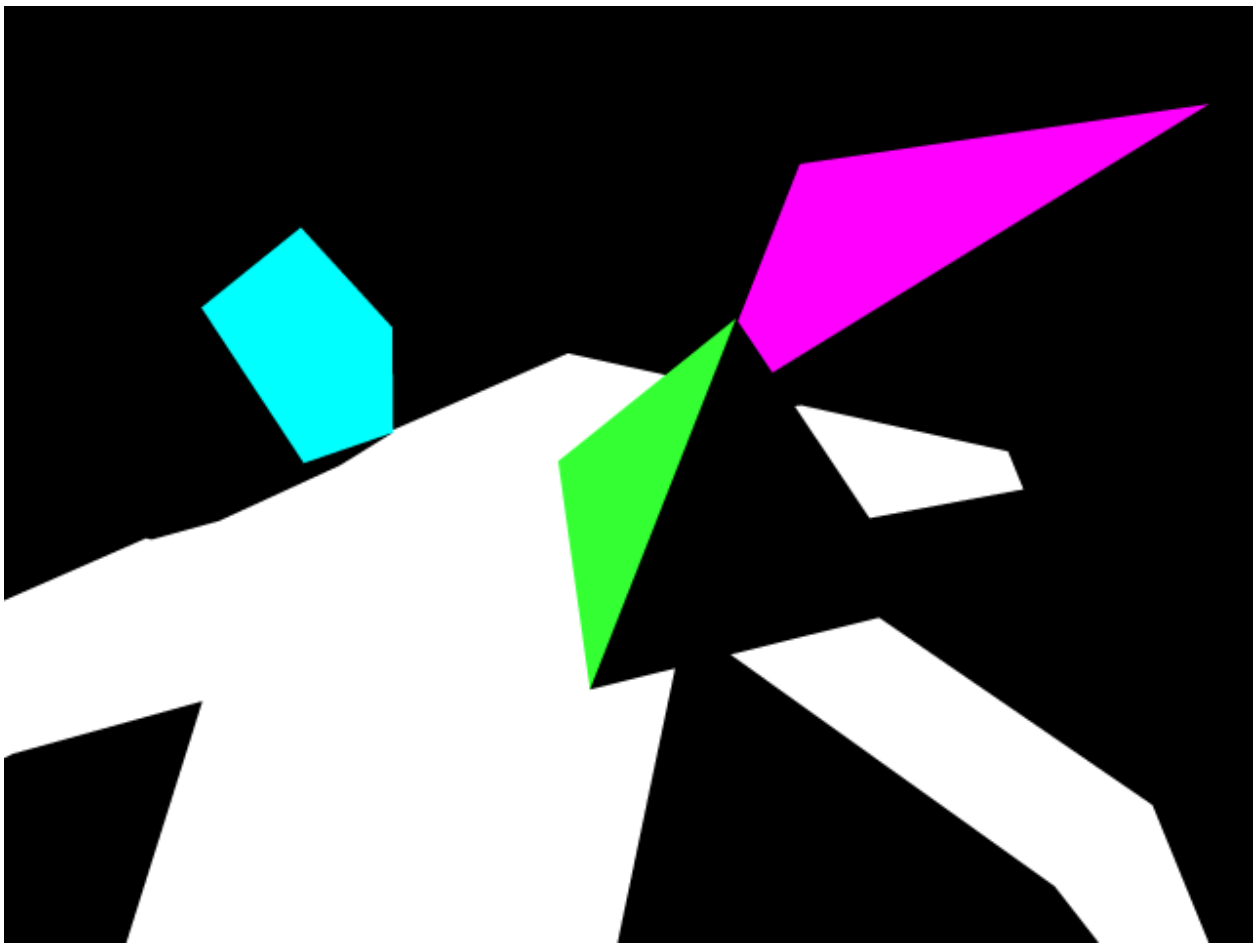
Скриншоты отрендеренных сцен:











ВЫВОД

Существует 2 основных принципа работы создания изображений на компьютере: растеризация и трассировка лучей. После выполнения курсовой работы я познакомился со вторым. Этот алгоритм вычислительно тяжёлый и хорошо параллелится, что делает его идеальным для разработки на CUDA. Было непросто добиться корректной работы программы, так как было сложно задать правильный порядок нормалей и, соответственно, вычислять преломление и отражение, учитывая все физические законы. Также были какие-то проблемы с альфа-каналом и картинка получалась просто чёрной, что было исправлено. Я доволен результатом, однако он далёк от идеала. Было бы здорово добавить источники света на рёбра фигур, высчитать внутреннее отражение лучей и наложить текстуры на полигоны. Также можно использовать текстурную память, которая предназначена как раз для

реализации подобных алгоритмов. В итоге мы получаем огромный выигрыш по времени на GPU, что в прикладных задачах очень важно, т. к. на одном компьютере современные фильмы или игры рендерились бы по несколько лет.

СПИСОК ЛИТЕРАТРУРЫ

1. Ray tracing http://www.ray-tracing.ru/
2. Трассировщик лучей с нуля https://habr.com/ru/post/436790/
3. Tracing in one weekend https://raytracing.github.io/books/RayTracingInOneWeekend.html