МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

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

Курсовая работа
по курсу «Программирование графических процессоров»

Обратная трассировка лучей (Ray Tracing). Технологии MPI, CUDA и OpenMP.

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

Москва, 2022 г.

УСЛОВИЕ

Цель работы: Совместное использование технологии MPI, технологии CUDA и технологии OpenMP для создание фотореалистической визуализации. 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

Метод решения полностью идентичен методу для КП по ПОД, за исключением того, что все переменные мы передаём всем процессам технологии MPI и работа происходит на кластере. Также цикл рендера в реализации для CPU исполняется параллельно с помощью технологии OpenMP.

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() – параллельный рендер кадра с помощью технологии OpenMP с вычислением начальных значений сдвига камеры и вызовом ray tracing функции.

int main() – отвечает за переключение между режимами выполнения (CPU/GPU), рассылку данных процессам с помощью MPI_BCAST(), 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. Кластер включает в себя 2 процесса MPI.

GPU:

Конфигурации ядер	Общее время работы	Среднее время на обработку одного кадра
<<<dim3(8, 8), dim3(8, 8)>>>	14289.9 ms	119.082 ms
<<<dim3(4, 16), dim3(4, 16)>>>	14025.7 ms	116.880 ms
<<<dim3(32, 32), dim3(32, 32)>>>	13642.6 ms	113.688 ms
<<<dim3(8, 32), dim3(8, 32)>>>	12217.8 ms	102.134 ms

CPU:

Общее время работы: 1436060 ms

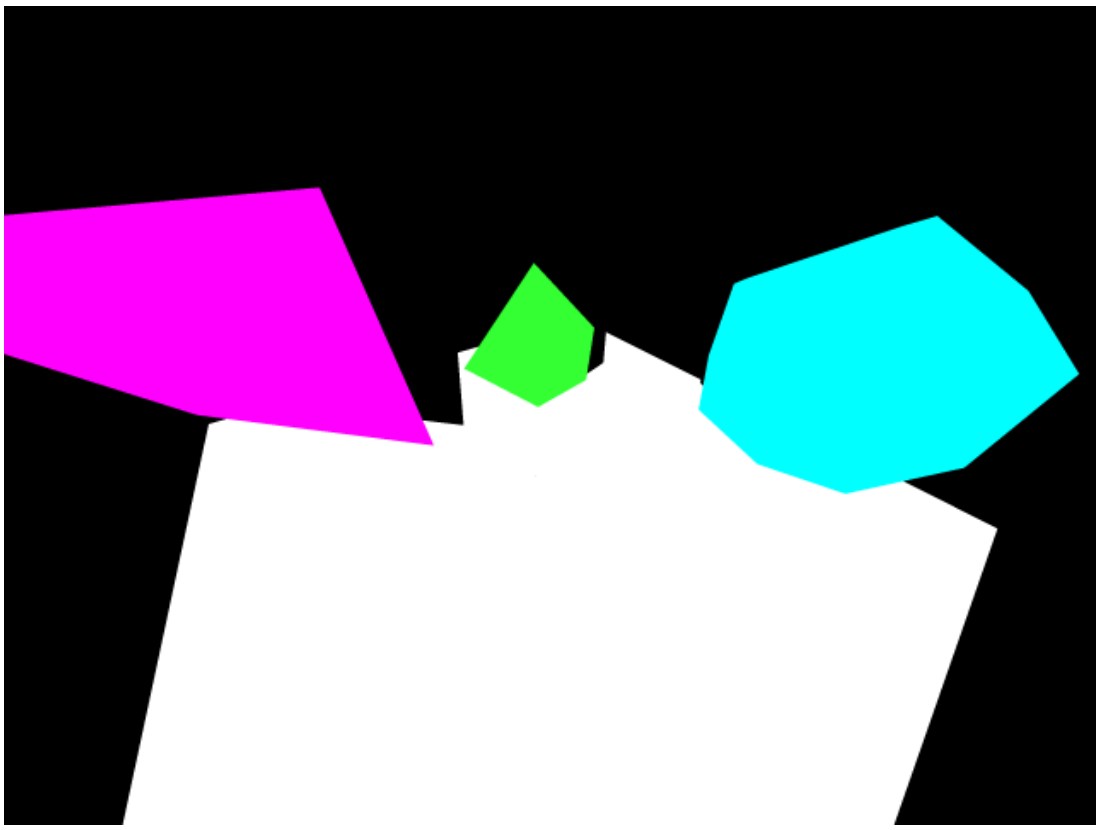
Среднее время на обработку одного кадра: 11967.2 ms

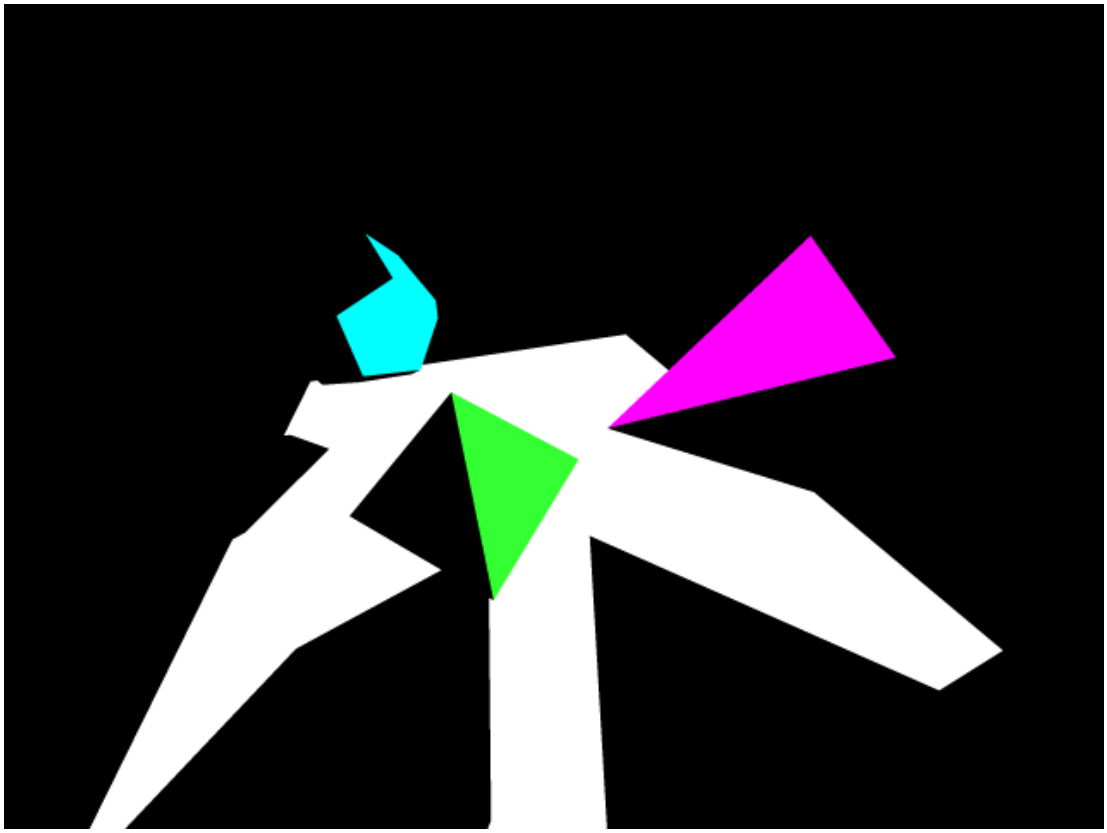
Таким образом ускорение $S = T1 / Tn = 1436060 / 12217.8 \approx 117$

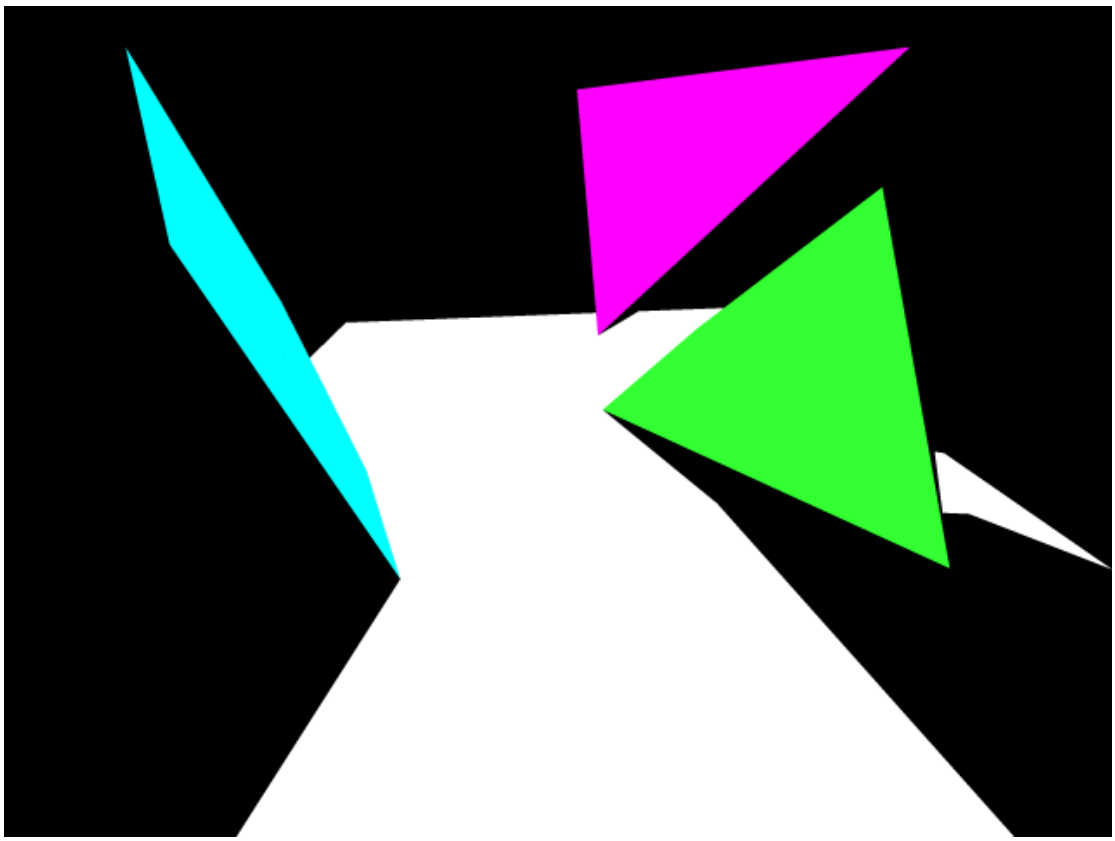
Коэффициент распараллеливания $P = S / N = 117 / 3 \approx 39$

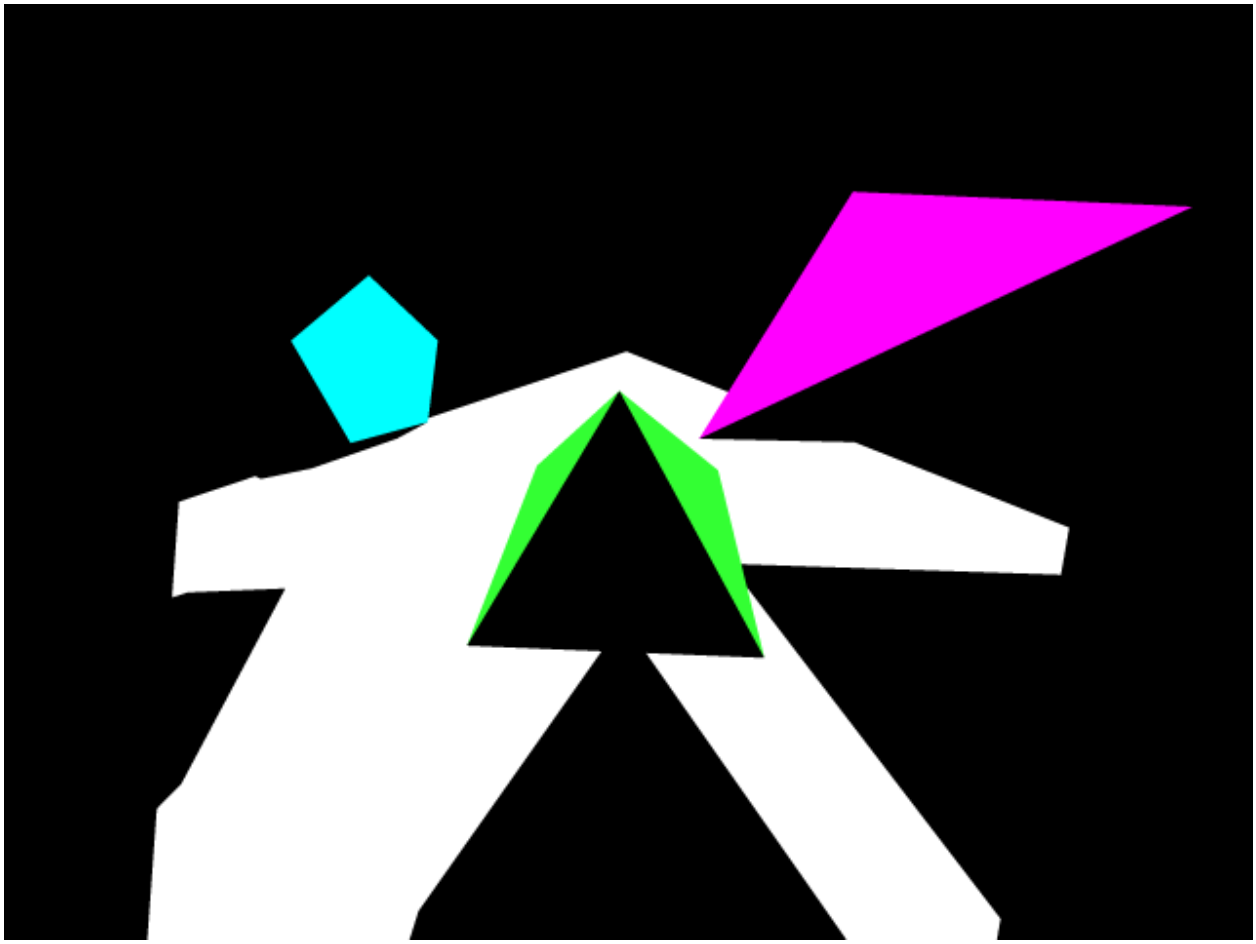
РЕЗУЛЬТАТЫ

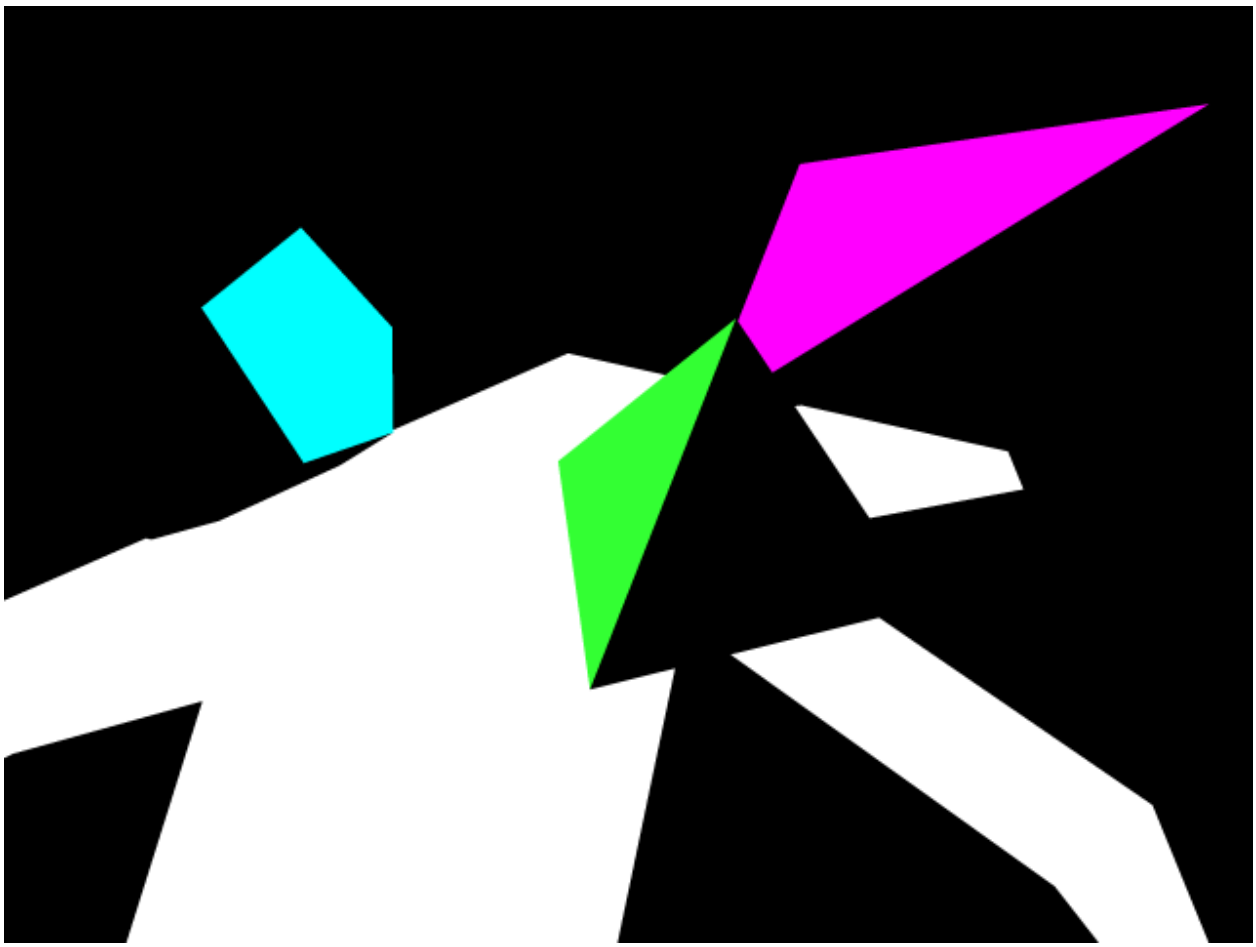
Скриншоты отрендеренных сцен:











ВЫВОД

Мне уже приходилось совместно использовать технологии MPI, CUDA и OpenMP в лабораторных работах, однако по достоинству оценить увеличение производительности тогда не получилось, так как алгоритм был не самый тяжеловесный, чего не скажешь про ray tracing. Если сравнить результаты работы программы на CPU, то версия распараллеленная на OpenMP дала ощутимый выигрыш по времени почти в 2 раза. Технология MPI же не показала особого преимущества, но я думаю это потому, что запускалось всего 2 процесса. Таким образом ускорение и коэффициент распараллеливания заметно уменьшились, т. к. время работы на GPU почти не изменилось, а на CPU существенно уменьшилось.

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