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: Р.С. Лисин
Группа: 8О-406Б
Преподаватели: К.Г. Крашенинников,
А.Ю. Морозов

Москва, 2023

Условие

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

Задание.

Сцена. Прямоугольная текстурированная поверхность (пол), над которой расположены три платоновых тела. Сверху находится один источник света.

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).

Вариант 2. Тетраэдр, Гексаэдр, Додекаэдр.:
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 процессе работы программа должна выводить в stdout статистику в формате: {номер кадра}\t{время на обработку кадра в миллисекундах}\t{общее количество лучей}\n и записывать результат по кадру в файлы.

Программное и аппаратное обеспечение

В качестве графического процессора использую видеокарту NVIDIA Tesla T4.

```
Compute capability : 7.5
Name : Tesla T4
Total Global Memory : 15835398144
Shared memory per block : 49152
Registers per block : 65536
Warp size : 32
Max threads per block : (1024, 1024, 64)
Max block : (2147483647, 65535, 65535)
Total constant memory : 65536
Multiprocessors count : 40
```

В качестве процессора использую Intel(R) Xeon(R) CPU @ 2.20GHz.

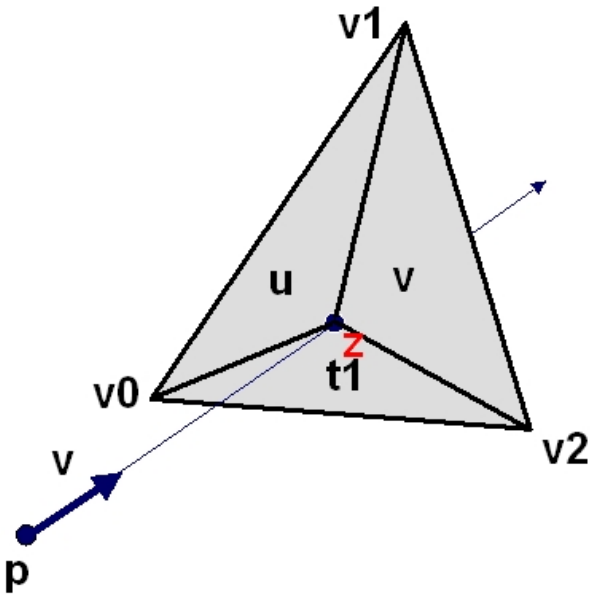
```
Architecture: x86_64
CPU op-mode(s): 32-bit, 64-bit
Address sizes: 46 bits physical, 48 bits virtual
Byte Order: Little Endian
CPU(s): 2 On-line CPU(s) list: 0,1
Vendor ID: GenuineIntel
Model name: Intel(R) Xeon(R) CPU @ 2.20GHz
CPU family: 6
Model: 79
Thread(s) per core: 2 Core(s) per socket: 1 Socket(s): 1 Stepping: 0
BogoMIPS: 4399.99
```

Оперативная память - 12.7 GB. Диск - 107.7 GB.

ОС - Ubuntu 22.04.3 LTS. В качестве редактора кода использовался Jupyter Notebook в Google Colab. Компилятор - nvcc 12.2.

Метод решения

Чтобы выполнить трассировку лучей, нужно для каждого луча найти первый треугольник, которой он пересечёт. Цвет треугольника будет соответствовать цвету соответствующего пикселя на экране. Поиск пересечения осуществляется с помощью следующих формул:



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

$$E1 = v1 - v0$$

$$E2 = v2 - v0$$

$$T = p - v0$$

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

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

$$D = v$$

Источник: http://www.ray-tracing.ru/articles213.html.

Для освещения я использую один источник света. Проверяю для каждого треугольника, есть ли от источника света луч к нему в первую очередь, то есть без других треугольников на пути. Если нет, тогда треугольник находится в тени.

Для сглаживания использую алгоритм SSAA. Он заключается в рендеринге сцены с повышенным разрешением, с последующим сэмплированием и смешением результата в меньшее количество пикселей. В качестве паттерна сэмплирования обычно используются ближайшие соседние пиксели (по сути, окружающий пиксель квадрат), а математика смешивания заключается в простом арифметическом усреднении сэмплов. Источник: https://habr.com/ru/articles/558552/.

Описание программы

Использую структуры данных vec3 для хранения координат точек в трёхмерном пространстве и trig для того, чтобы задать треугольник. Сцена и координаты тел задаются соответственно в функциях make_floor, make_tetrahedron, make_hexahedron,

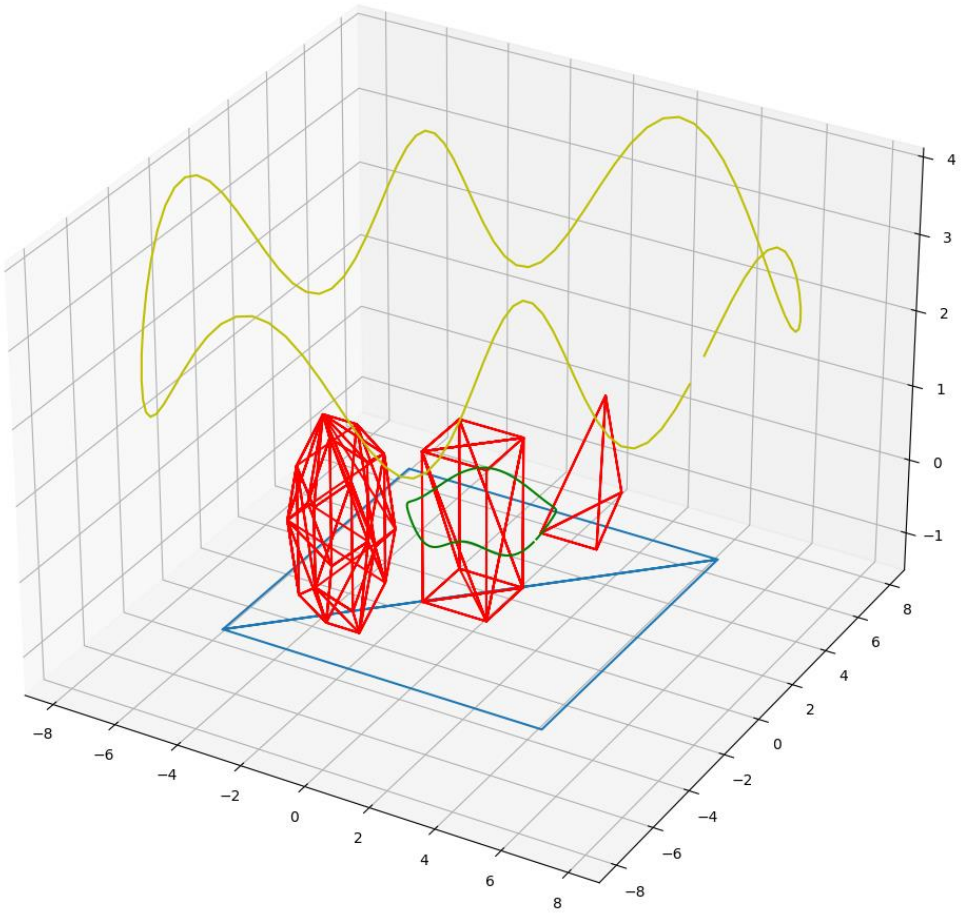
make_dodecahedron. В функции ray выполняется обратная трассировка луча. В ядре kernel_render происходит рендеринг изображения на GPU. В ядре kernel_ssaa выполняется сглаживание с помощью алгоритма SSAA, потому что это самый простой алгоритм для этого. Также всё это реализовано и для CPU.

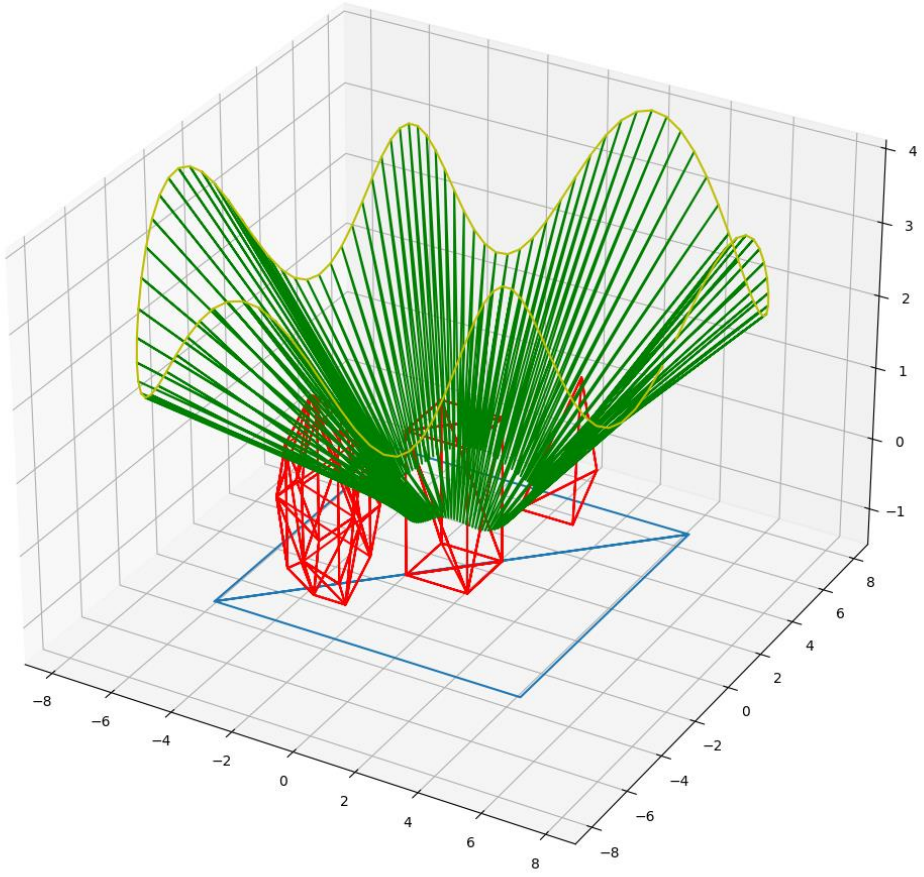
Исследовательская часть и результаты

Наиболее красочный, по моему мнению, результат получается на таких данных.

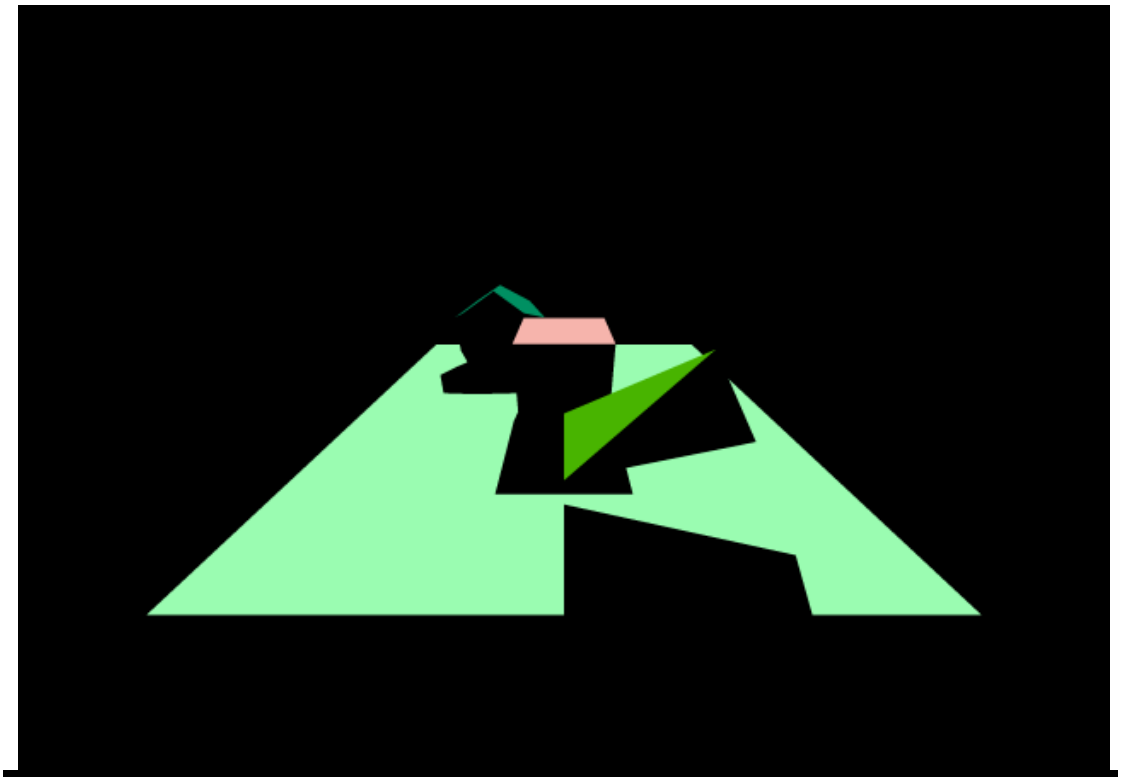
```
120
res/%d.data
640 480 120
7.0 3.0 0.0 2.0 1.0 2.0 6.0 1.0 0.0 0.0
2.0 0.0 0.0 0.5 0.1 1.0 4.0 1.0 0.0 0.0
3.0 2.0 0.6 0.3 0.75 0.0 1.0
0.0 0.0 0.0 0.6 0.25 0.55 1.75
-3.0 -2.0 0.0 0.0 0.8 0.7 1.5
-5.0 -5.0 -1.0 -5.0 5.0 -1.0 5.0 5.0 -1.0 5.0 -5.0 -1.0 1.0 0.9 0.35
-10.0 0.0 12.0 0.4 0.3 0.1
4
```

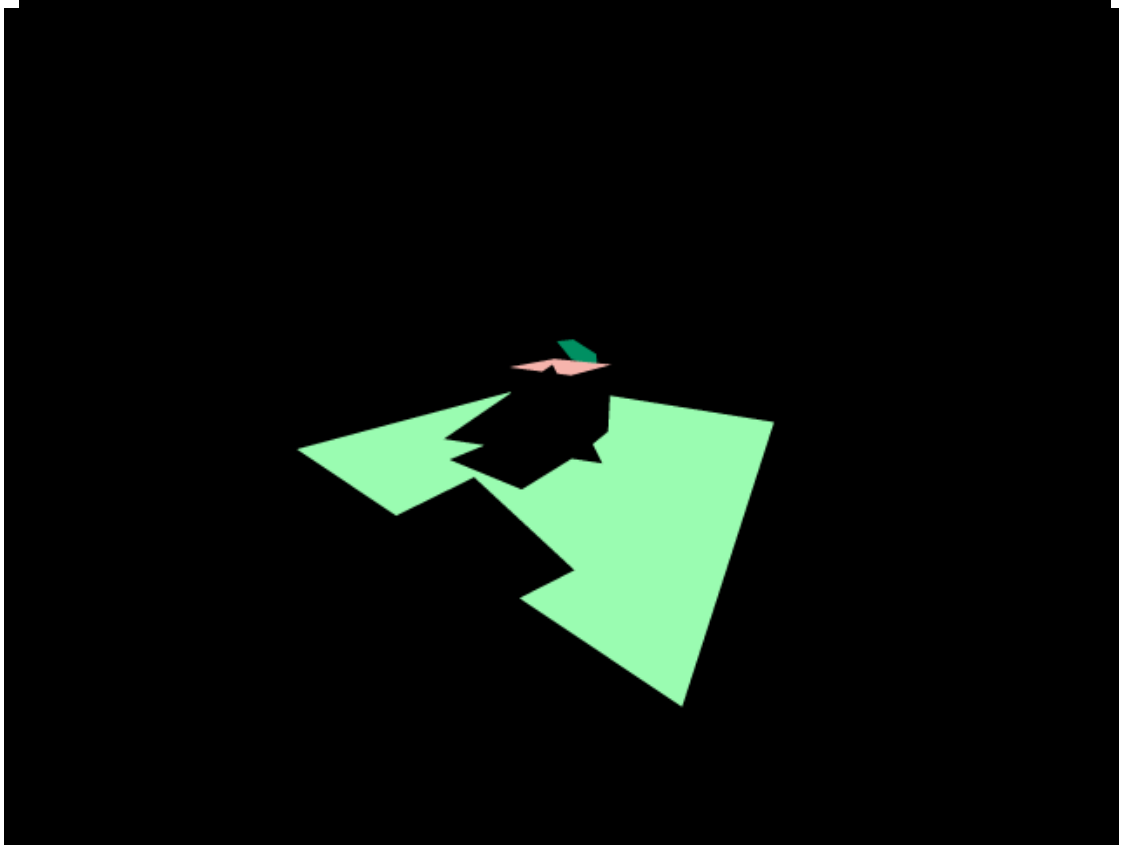
Трёхмерные графики.

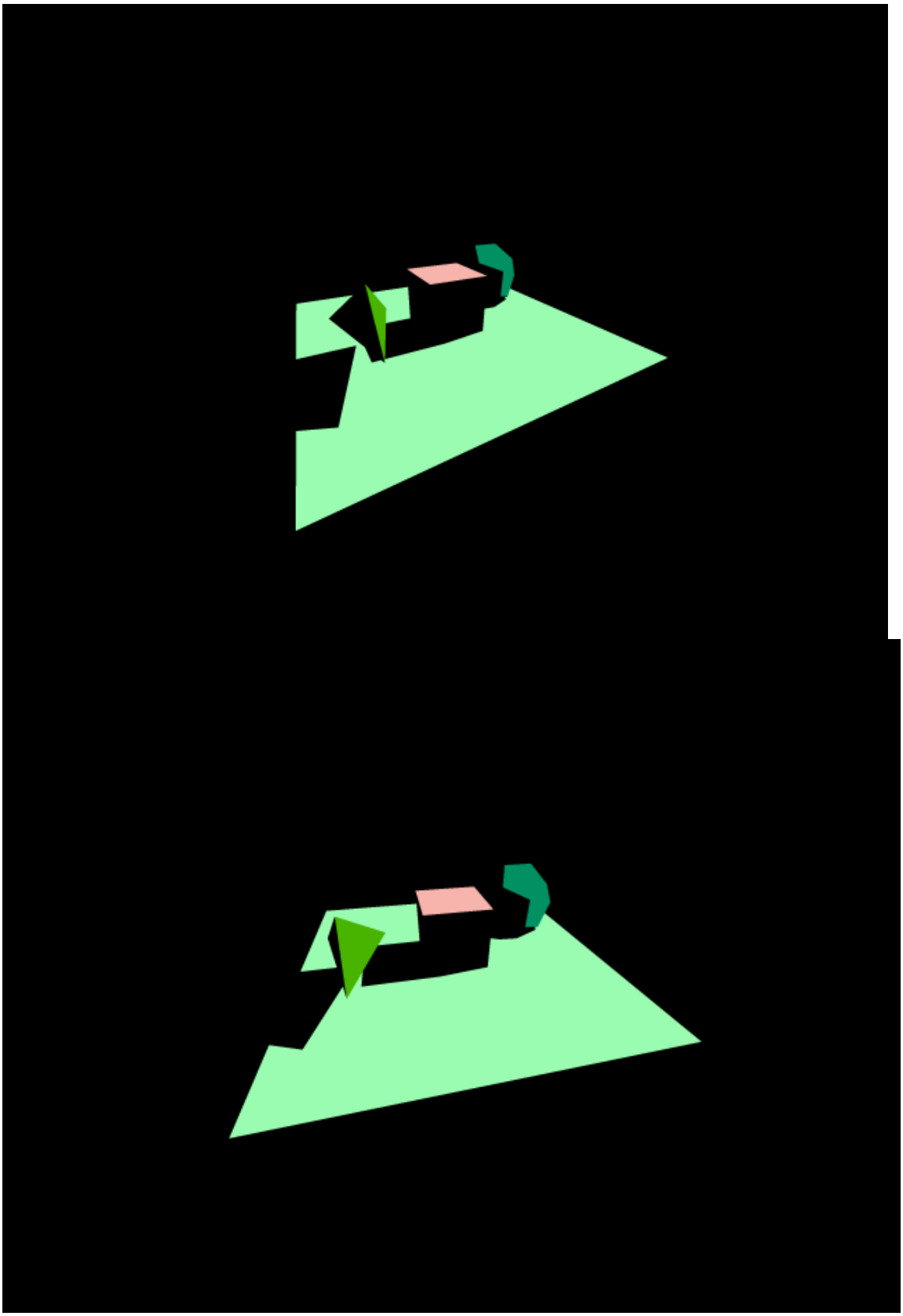

4
3
2
1
0
−1
−8
−6
−4
−2
0
2
4
6
8
−8
−6
−4
−2
0
2
4
6
8

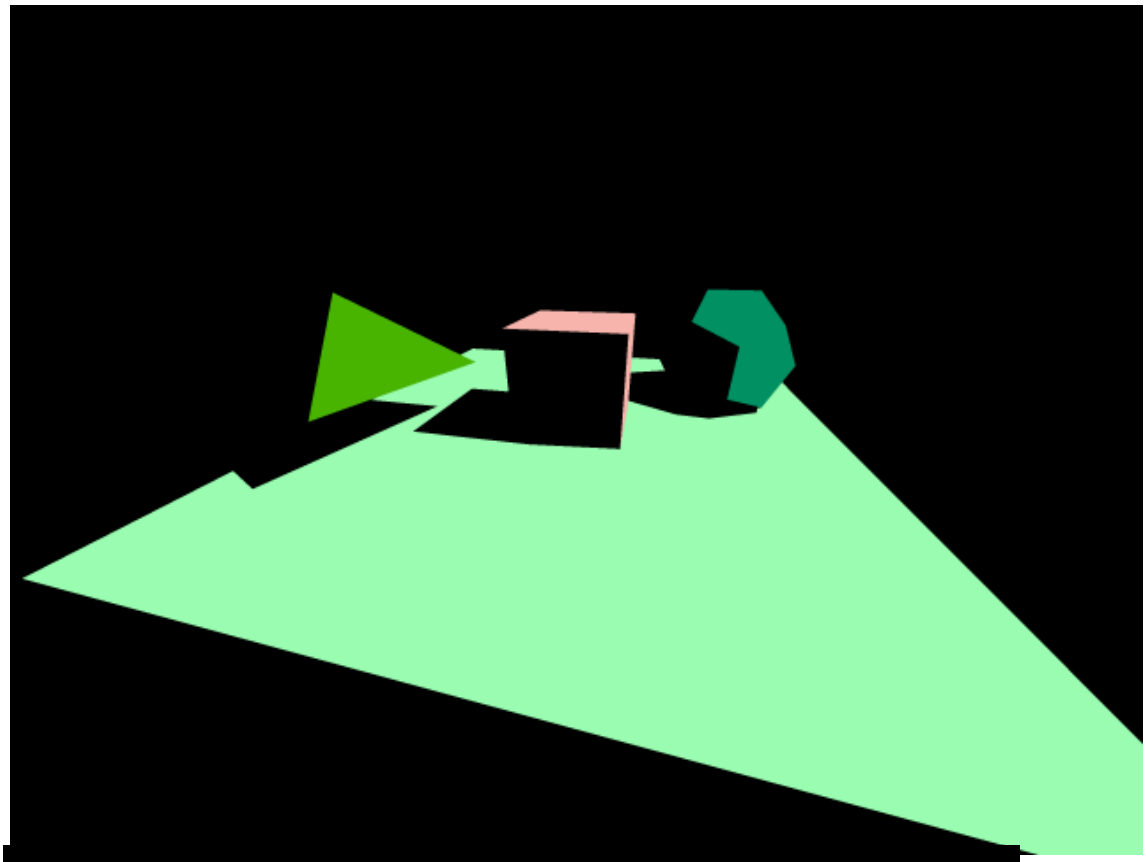


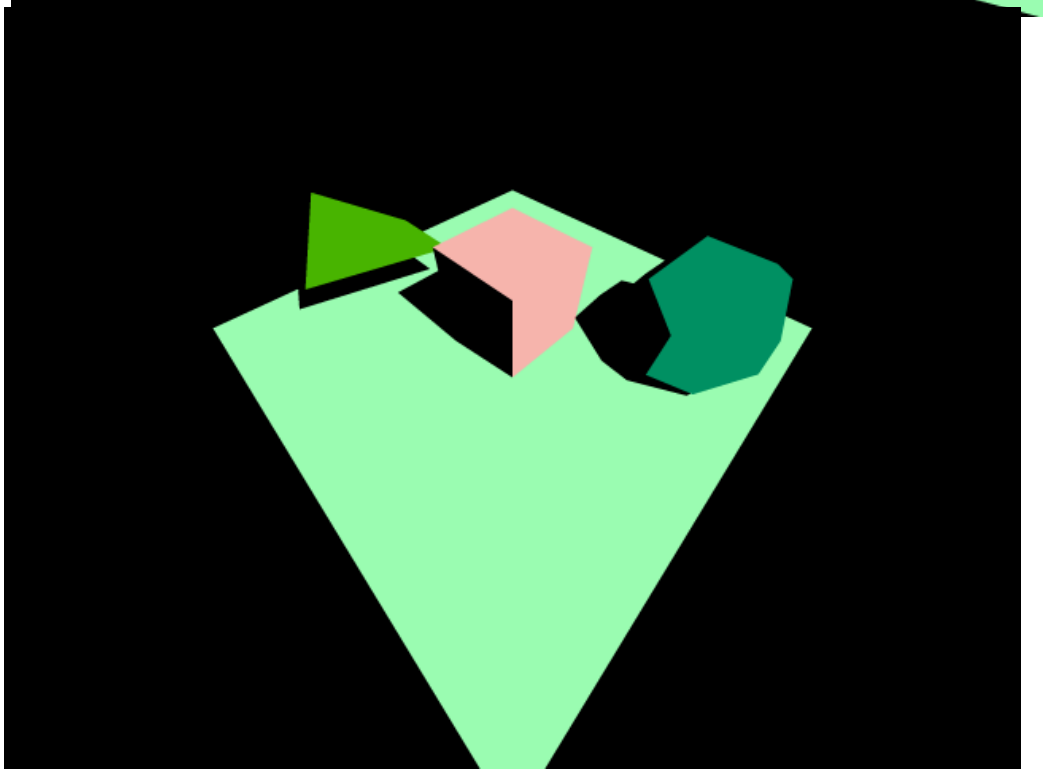
Кадры.

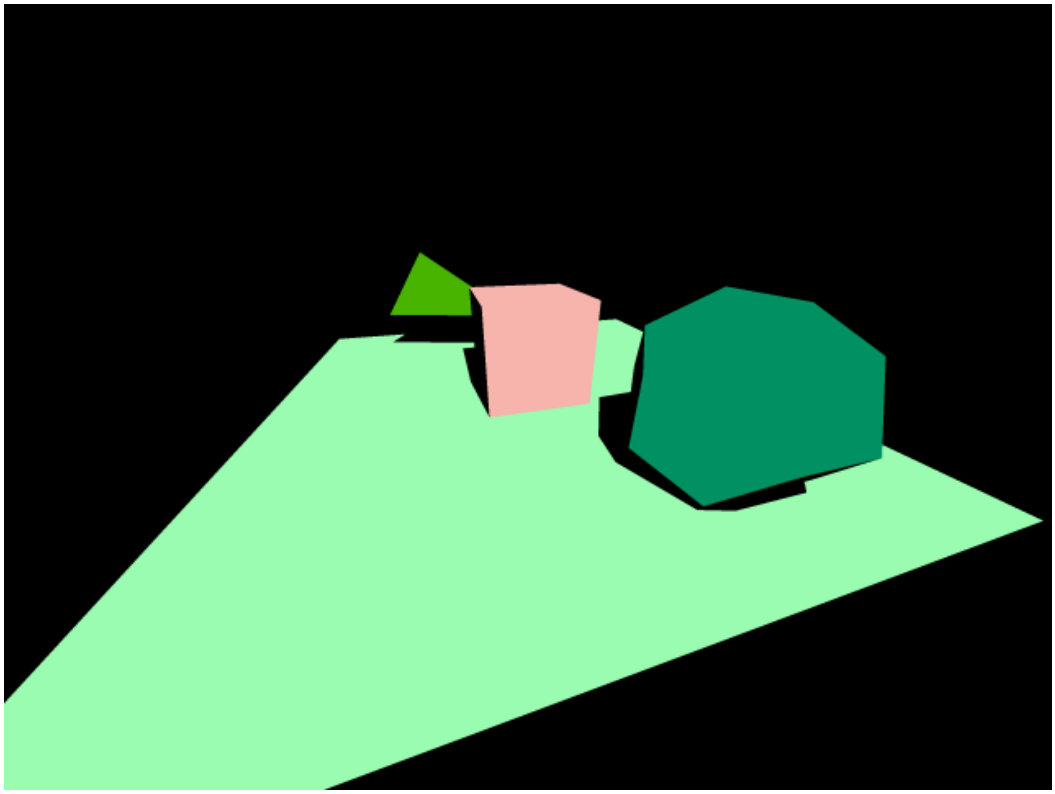


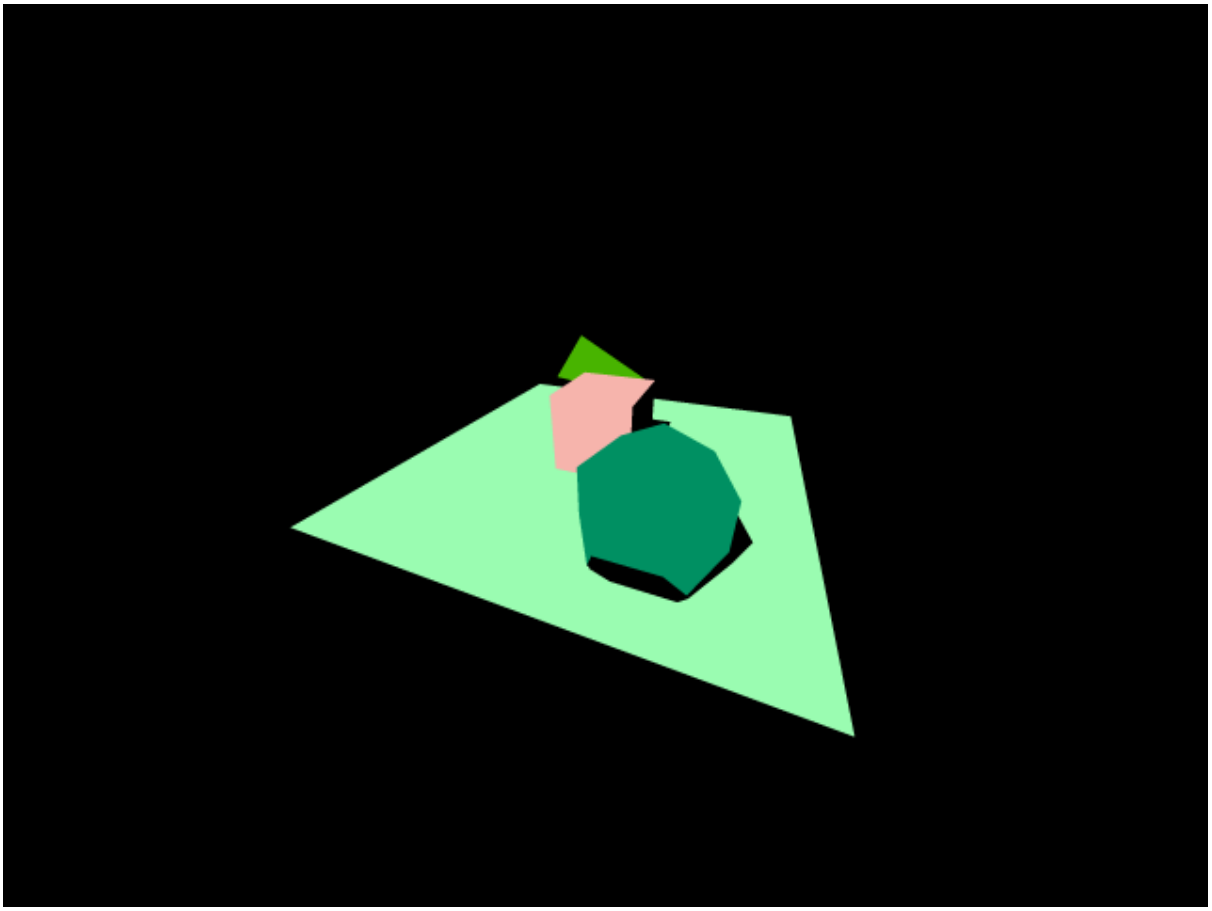


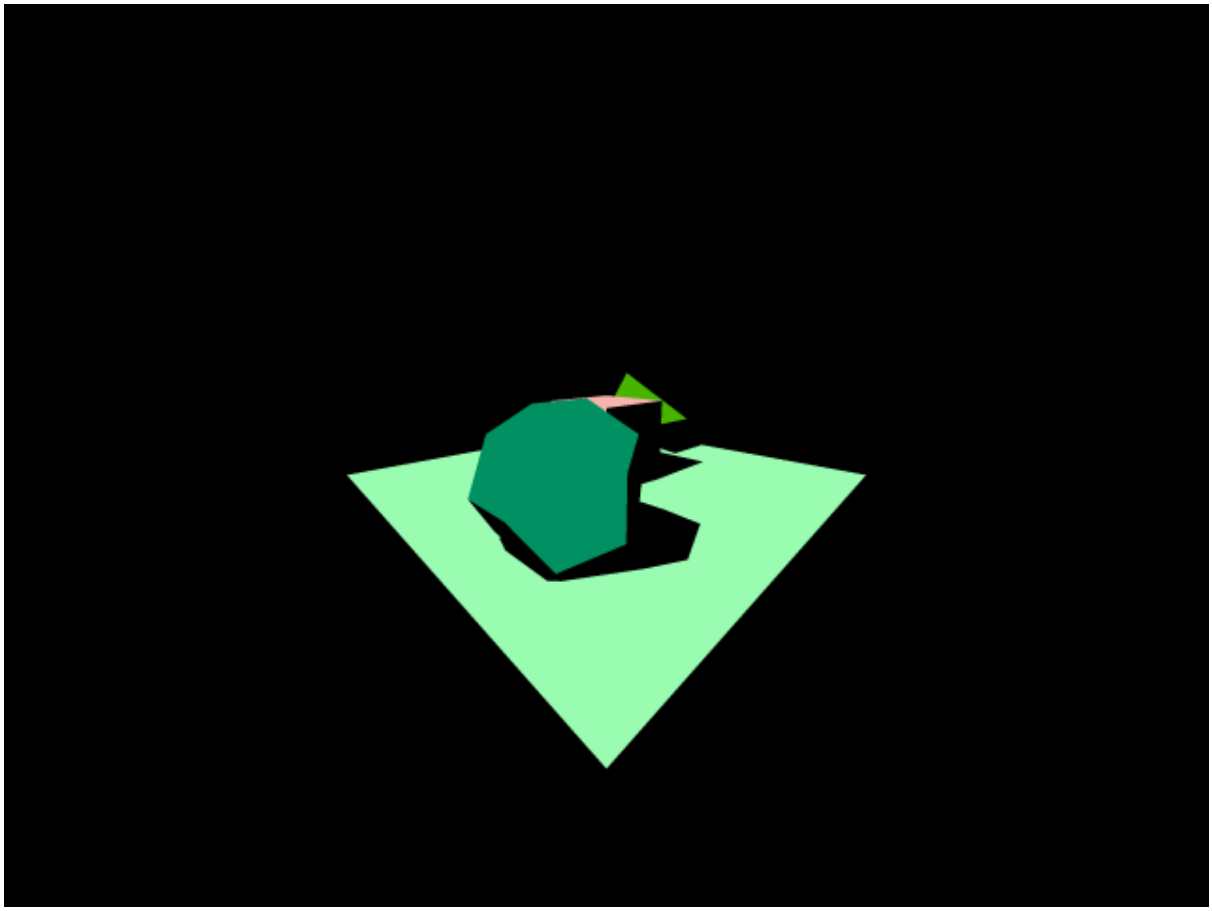


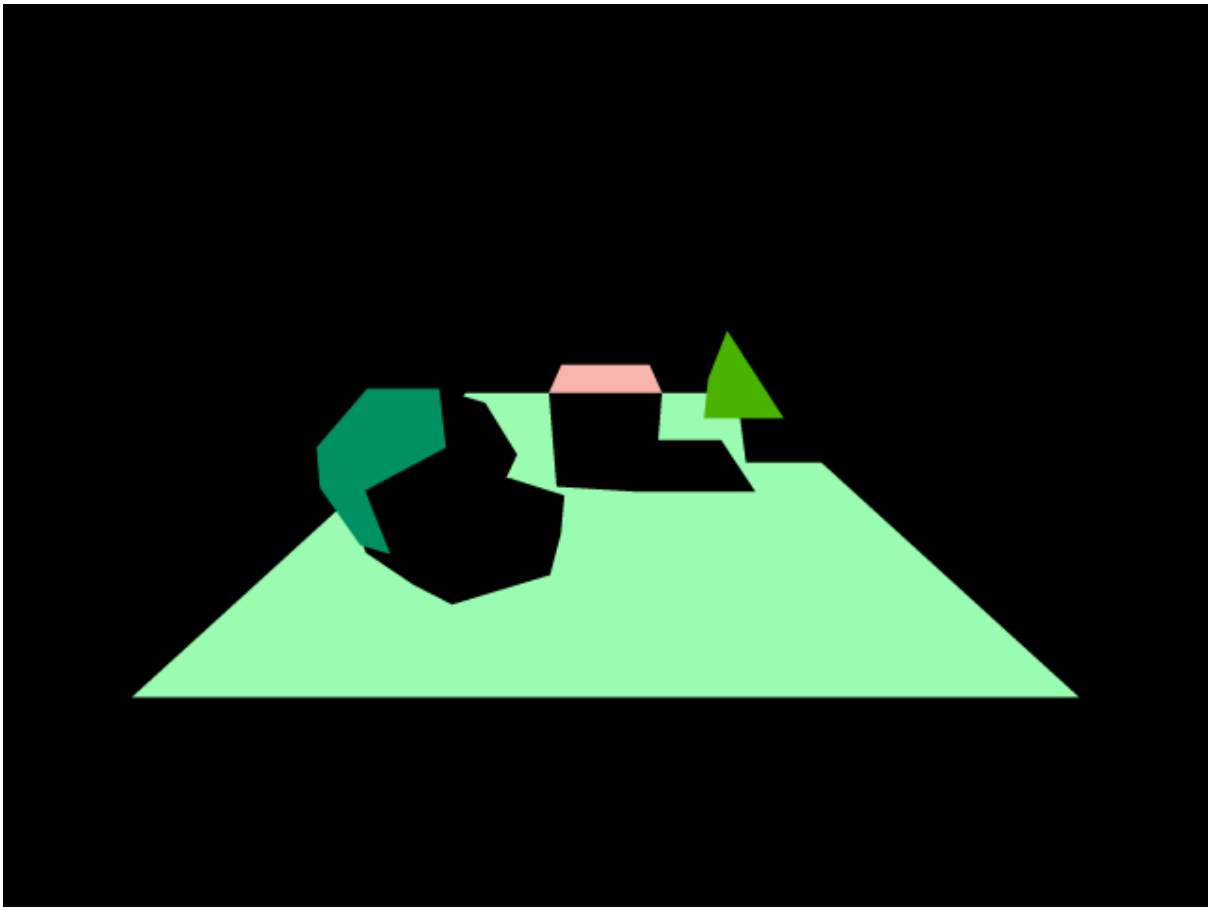




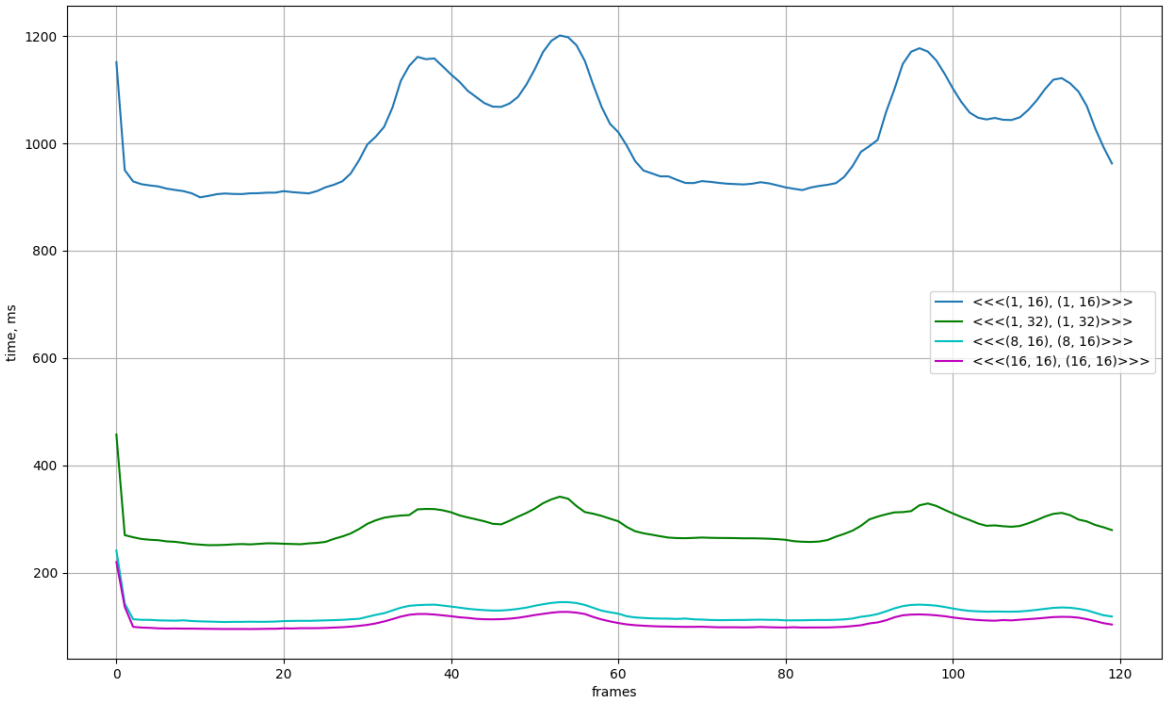


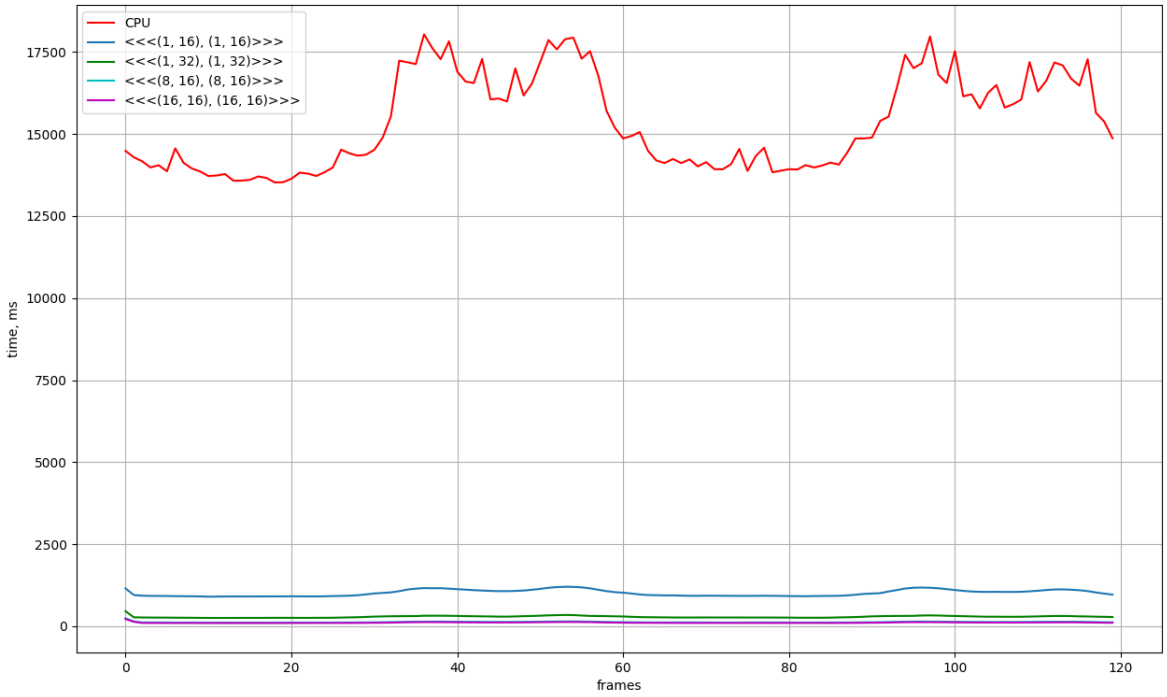






Графики замеров времени построения одного кадра с различными конфигурациями ядра и на CPU.


time, ms
frames
<<<(1, 16), (1, 16)>>>
<<<(1, 32), (1, 32)>>>
<<<(8, 16), (8, 16)>>>
<<<(16, 16), (16, 16)>>>



Алгоритм на CPU справляется сильно медленнее чем на GPU. Это безусловно связано с тем, что в данном случае распараллеливание в разы ускоряет работу алгоритма.

Ссылка на анимацию: https://drive.google.com/file/d/1U4EpCmX7hBG1L-SrZBytg-sfpqNMRZxZ/view

Выводы

Алгоритм, реализованный в данной работе, можно применить для создания качественных реалистичных изображений. Такие изображения нужны, к примеру, для создания игр. Алгоритм не сложный, основан на математике, но сама работа получилась достаточно объёмной. Реализация алгоритма на GPU значительно ускоряет работу программы по сравнению с CPU. Качество изображений получается хорошим.

Литература

1) Лекции по ПГП/ПОД
2) Трассировка лучей http://ray-tracing.ru/
3) SSAA https://habr.com/ru/articles/558552/