МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

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

Курсовой проект
по курсу «Программирование графических процессоров»

Технологии CUDA и OpenMP

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

Москва, 2020

Условие

Цель работы: использование технологии openMP для создание фотореалистической визуализации. Рендеринг полузеркальных и полупрозрачных правильных геометрических тел. Получение эффекта бесконечности. Создание видеоролика.
Задание: 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.
Вариант: гексаэдр, октаэдр, додекаэдр.

Программное и аппаратное обеспечение

- GPU: Geforce 940MX
  - Compute capability : 5.0
  - Total Global Memory : 2147483648
  - Shared memory per block : 49152
  - Registers per block : 65536
  - Max threads per block : (1024, 1024, 64)
  - Max block : (2147483647, 65535, 65535)
  - Total constant memory : 65536
  - Multiprocessors count : 3
- CPU: Intel Core i5-6200U 2.30GHz
- RAM: 4GB
- Software: Windows 10, Visual Studio Code, nvcc

Метод решения

Обратная трассировка лучей — это метод создания фотореалистичных изображений, при котором мы исследуем взаимодействие отдельных лучей с поверхностями, причем отслеживание траектории лучей мы производим не от источника к экрану, а наоборот, что позволяет сохранить огромное число ресурсов и повысить производительность.
В общем виде алгоритм обратной трассировки выглядит следующим образом:

1. Для каждого пикселя камеры, выпускаем из него луч
2. Ищем пересечения луча с ближайшим к нему объектом. Если есть такое пересечение, переходим к пункту 3, иначе возвращаем цвет фона.
3. Для точки пересечения луча с объектом ищем его освещенность каждым источником света по некоторой локальной модели (например Фонга). Затем суммируем эти освещенности.
4. Если объект обладает отражающими свойствами, отслеживаем цвет отраженного луча и прибавляем его к итоговой освещенности, т.е процедура трассировки рекурсивно повторяется.
5. Если объект обладает преломляющими свойствами, отслеживаем цвет преломленного луча и прибавляем его к итоговой освещенности

Существуют различные усложнения алгоритма для его большей реалистичности и производительности, но в данной работе я реализовал простую модель.
Что касается модели освещения, то я использовал частный случай затенения по Фонгу, в общем случае формула вычисления освещенности выглядит следующим образом:

$$I = K_a I_a + K_d (n, l) + K_s (n, h)^p$$

где:

n — вектор нормали к поверхности
l — направление на источник света
h — направление на наблюдателя
K_a — коэффициент фонового освещения
K_s — коэффициент зеркального освещения
K_d — коэффициент диффузного освещения

В моем конкретном случае коффициенты K_s и K_a равны нулю, то есть я учитываю только диффузную составляющую, это немного портит картинку, когда объект находится близко к источнику освещения, но в целом изображение выглядит неплохо. Также я учитываю уменьшение интенсивности света с расстоянием, а итоговая освещенность от источника света обратно пропорциональна квадрату расстояния до этого источника).
Для вычисления отраженного луча используетя закон отражения, выглядящий следующим образом:

$$R_{refl} = R_{inc} - 2(N, R_{inc}) N$$

Для вычисления преломленного луча можно воспользоваться Законом Шелла, однако он зависит от коэффициентов преломления двух сред, поэтому чтобы не усложнять программу, было принято решения считать их равными единице для всех сред, а итоговая формула приняла следующий вид:

$$R_{refr} = R_{inc}$$

Описание программы

Для удобства организации кода, он поделен на несколько логически завершенных файлов:

- consts.h – файл с основными константами, позволяет быстро настраивать параметры генерации джижка.
- ray.h – файл с классом, представляющим луч (начало луча и его направление)
- camera.h – файл с классом, представляющим камеру. Можно задавать ее характеристики: угол обзора, соотношение сторон, положение, направление взгляда.
- vec_types.h – файл, определяющий векторные типы int2, int3, uchar4, float2, float3, а также векторные операции для float3.
- objects.h — файл, определяющий тела, которые могут быть отрисованы с помощью трассировки лучей. Представляет несколько абстрактных классов и их завершенных потомков.
    - Class HittableObject – абстрактный класс, реализующий интерфес всех отображаемых тел.

    - Class Sphere – класс, представляющий шар.
    - PlatonicSolid – абстрактный класс для платоновских тел, служит интерфейсом для их общих методов, наследуется от HittableObject. В его интерфейсе присутствуют виртулаьные функции n_triangles, n_edges и т.д.,
- ray_tracer.h – файл, содержащий класс RayTracer, с помощью которого происходит генерация картинки. который содержит версии движка рейтрейсинга для цпу и гпу. Определяет следующие методы:
    - void render – генерирует картинку с полученным миром и записывает ее в полученный буфер image.
    - float3 clamp_color(float3 color) — не позволяет цвету выйти за границы (1, 1, 1)
    - float3 reflect(float3 incident_dir, float3 normal) — возвращает отраженный луч для заданного входного луча и нормали поверхности
    - float3 refract(float3 incident_dir, float3 normal) — возвращает преломленный луч для заданного входного луча и нормали поверхности
    - int trace_intersect — проверяет, пересекается ли заданный луч с каким-либо объектом мира, и возвращает информацию о найденном пересечении в структуре HitRecord
    - float3 get_color — возвращает цвет заданного луча
    - void render_sequantially – генерирует картинку в режиме одного потока
    - void render_parallel — генерирует картинку параллельно с помощью технологии openMP

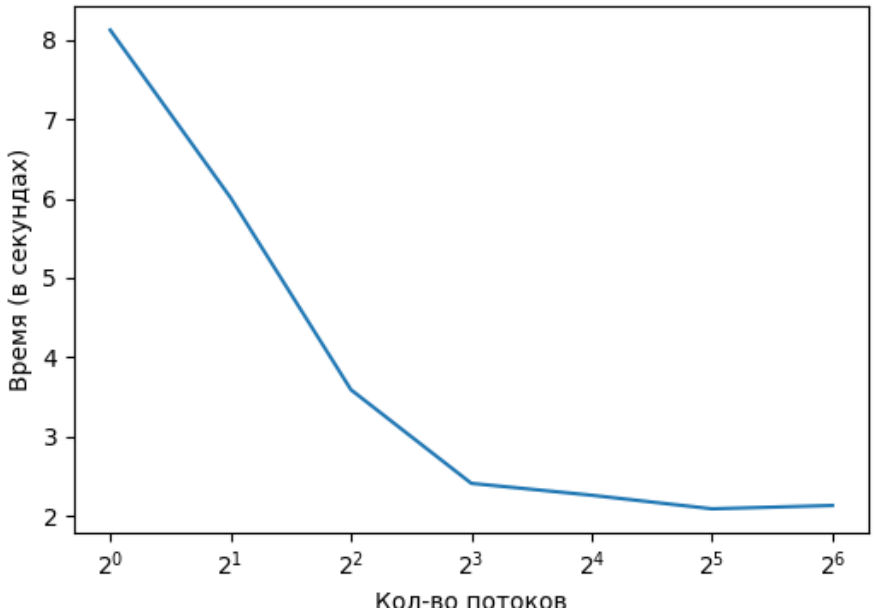
Исследовательская часть и результаты

Проведем сравнение производительности однопоточной и параллельной версии программ. Расспараллелить оригинальный алгоритм можно двумя способами:

1. генерация одного кадра (все пиксели внутри одного кадра независимы друг от друга)
2. генерация последовательности кадров (в случае, когда закон движения задан явно, мы можем параллельно генерировать кадры для разных моментов времени)

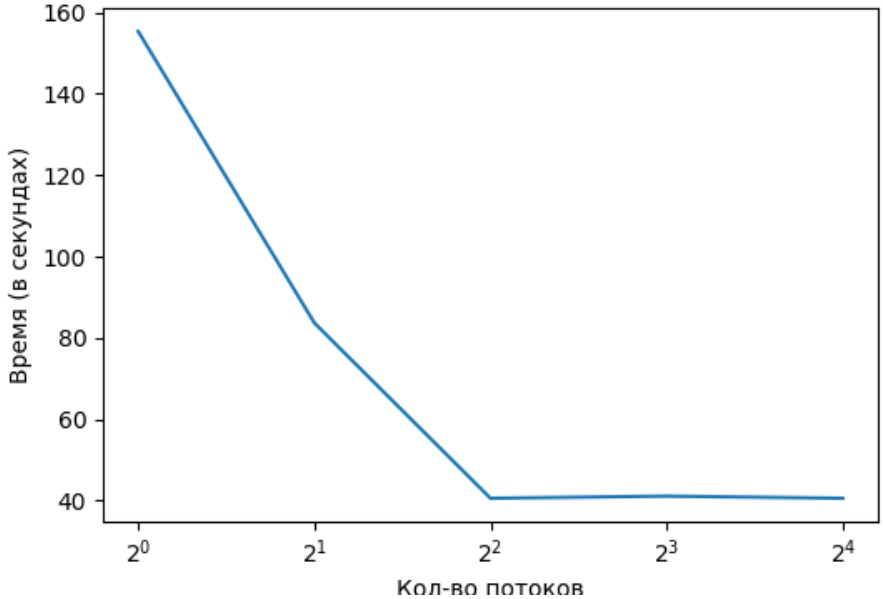
1) Распараллеливание генерации одного кадра на примере простой сцены (куб с двумя источниками освещения на каждом ребре и пол с текстурой)

Количество потоков	время (в секундах)
1	8.122
2	6.01
4	3.591
8	2.412
16	2.262
32	2.091
64	2.133



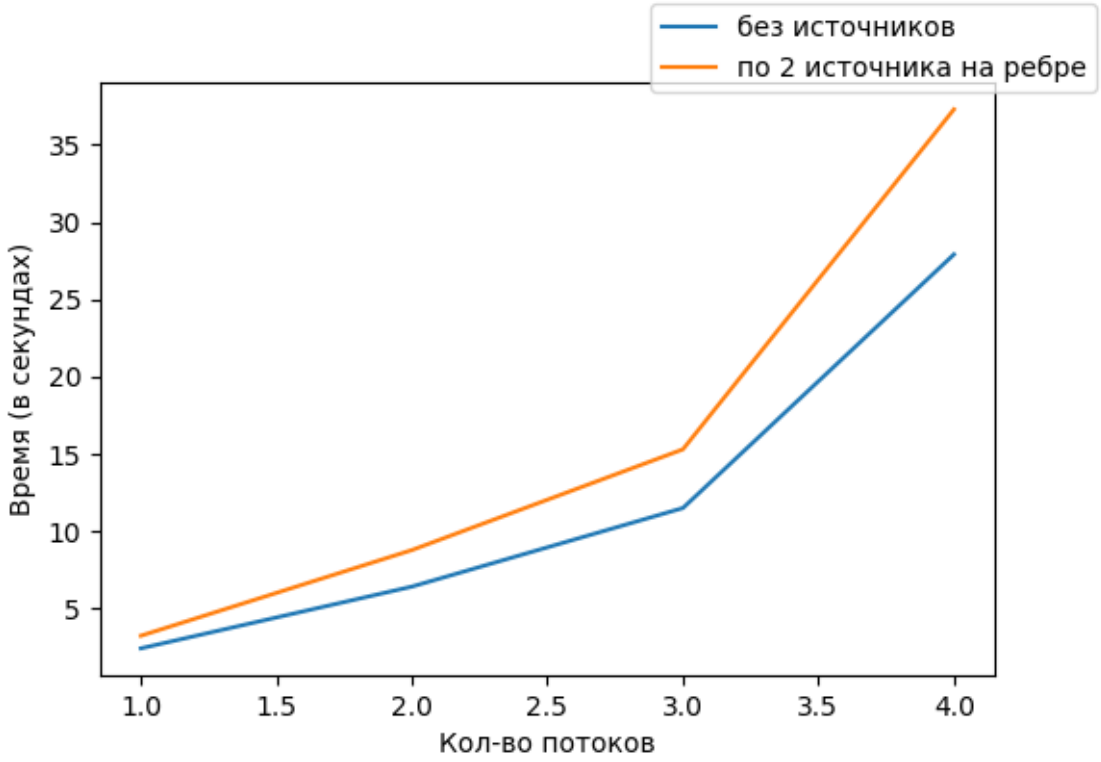
2) Распараллеливание генерации последовательности кадров при вращении камеры вокруг простой сцены (куб с двумя источниками освещения на каждом ребре и пол с текстурой)

Количество потоков	время (в секундах)
1	155.348
2	83.687
4	40.574
8	41.117
16	40.574
Последовательное распареллеливание с 8 потоками на один кадр	45.317

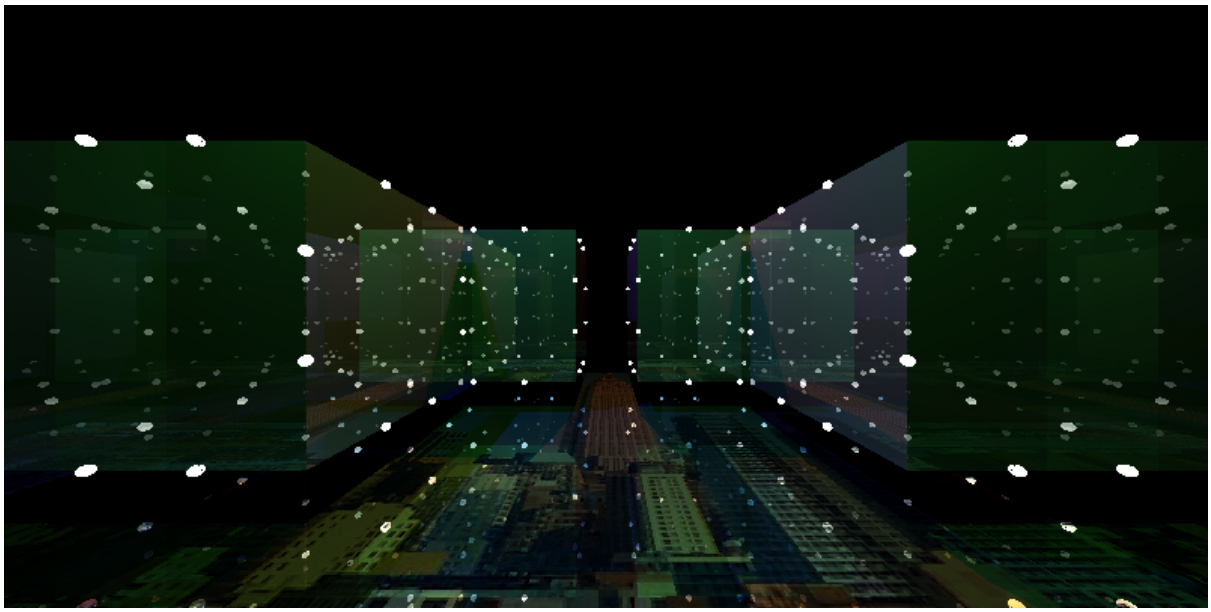


Производительность при генерации простой сцены с 8 потоками в зависимости от количества объектов:

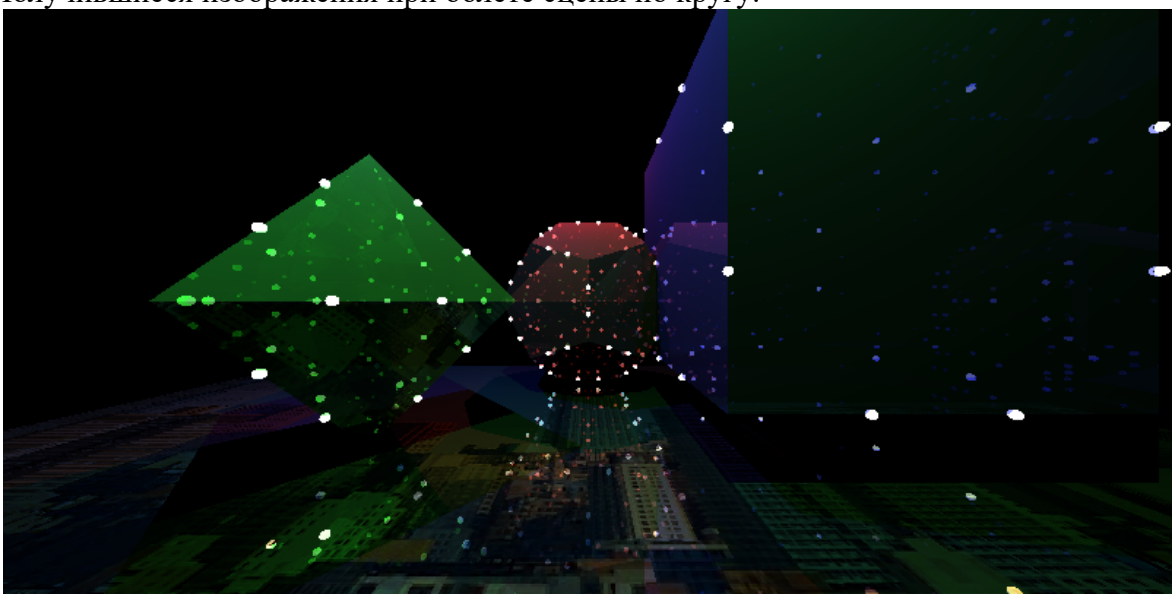
	Без источников на ребрах	По 2 источника на ребре
1 куб	2.446	3.259
2 куба	6.435	8.797
3 куба	11.513	15.306
4 куба	27.905	37.28

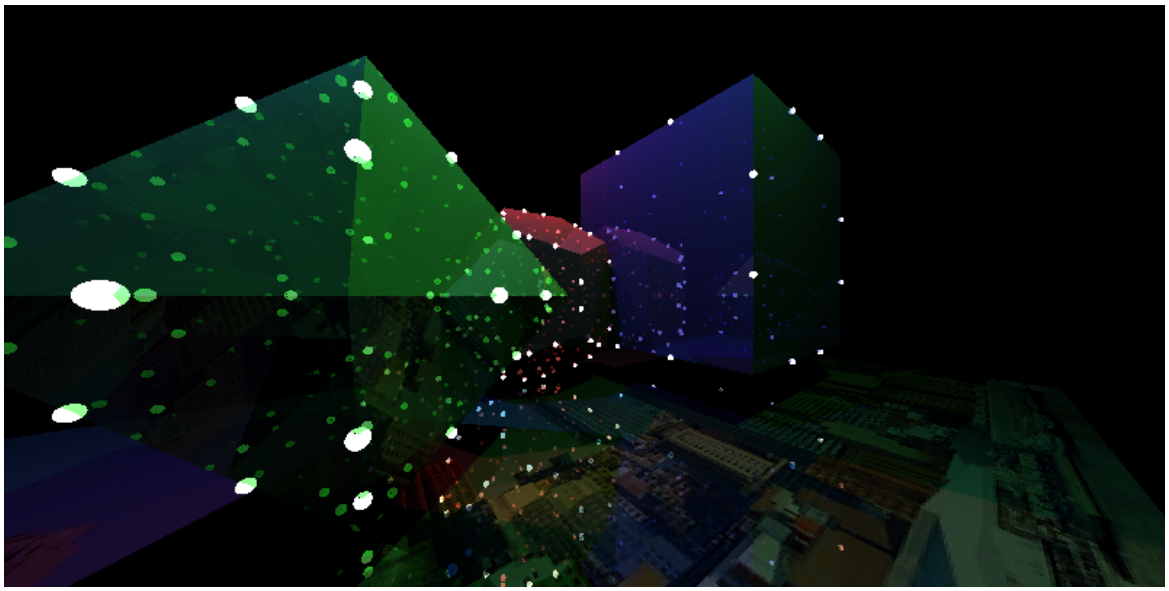


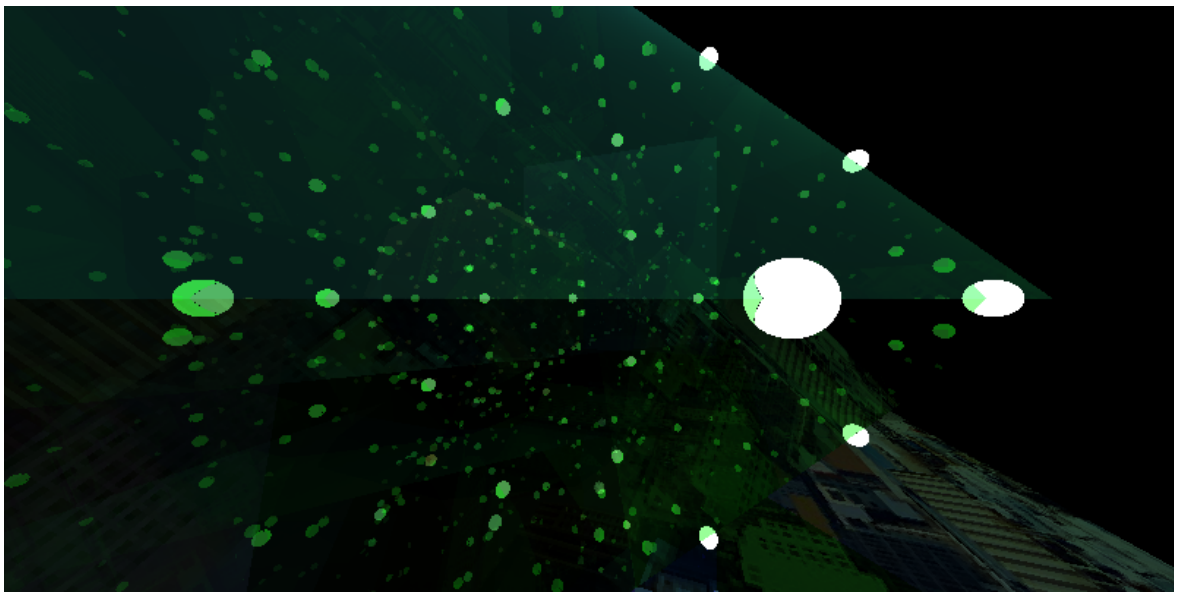
Сцена 4 кубами:

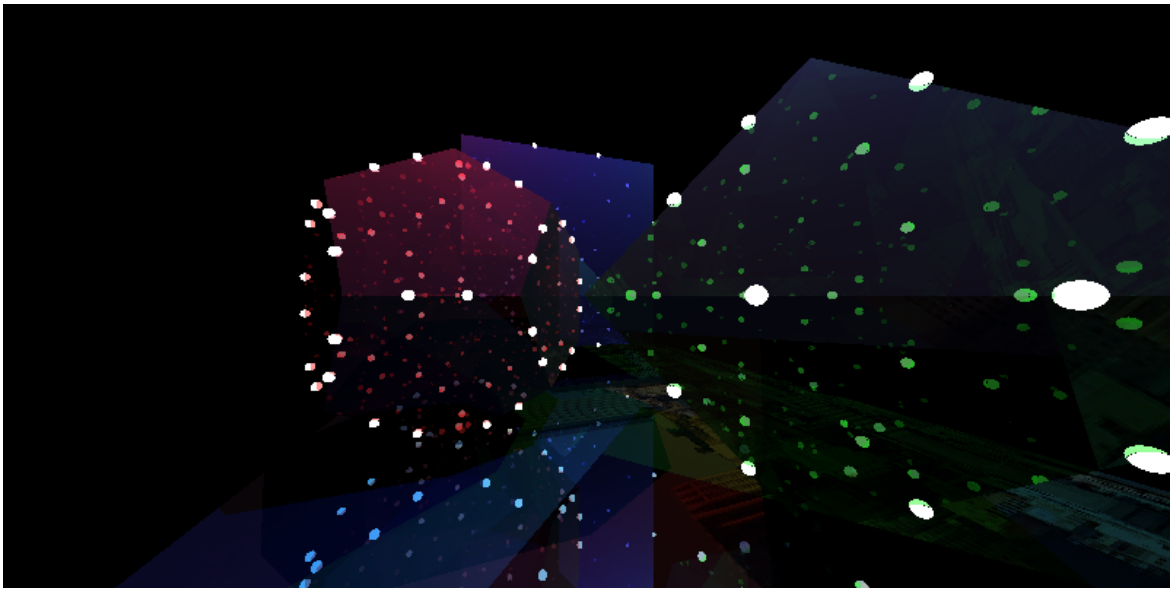


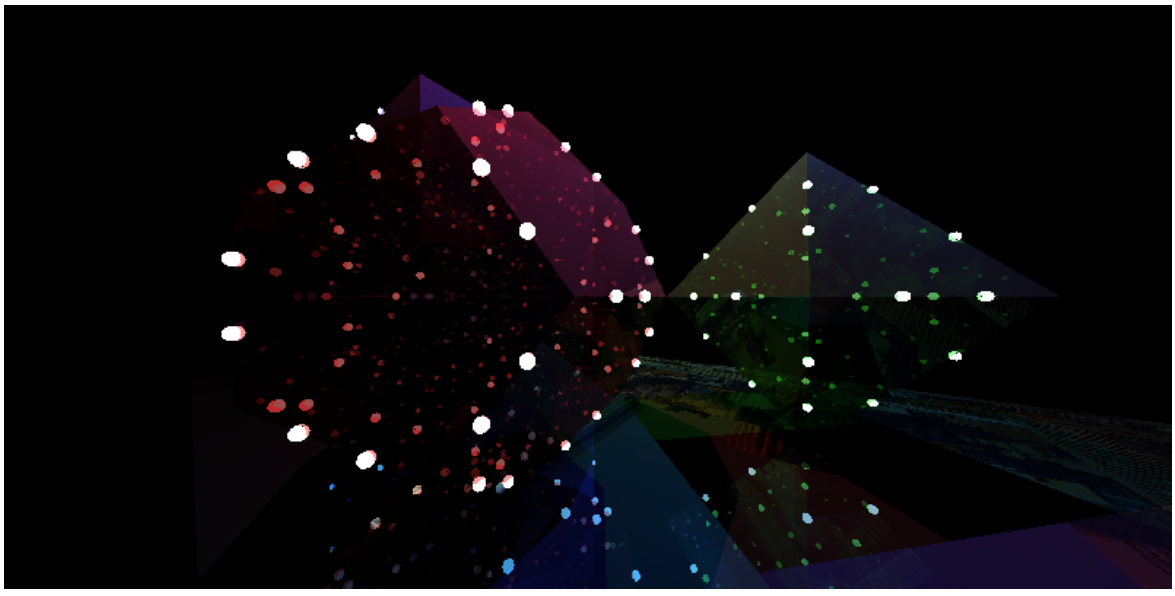
Получившиеся изображения при облете сцены по кругу:

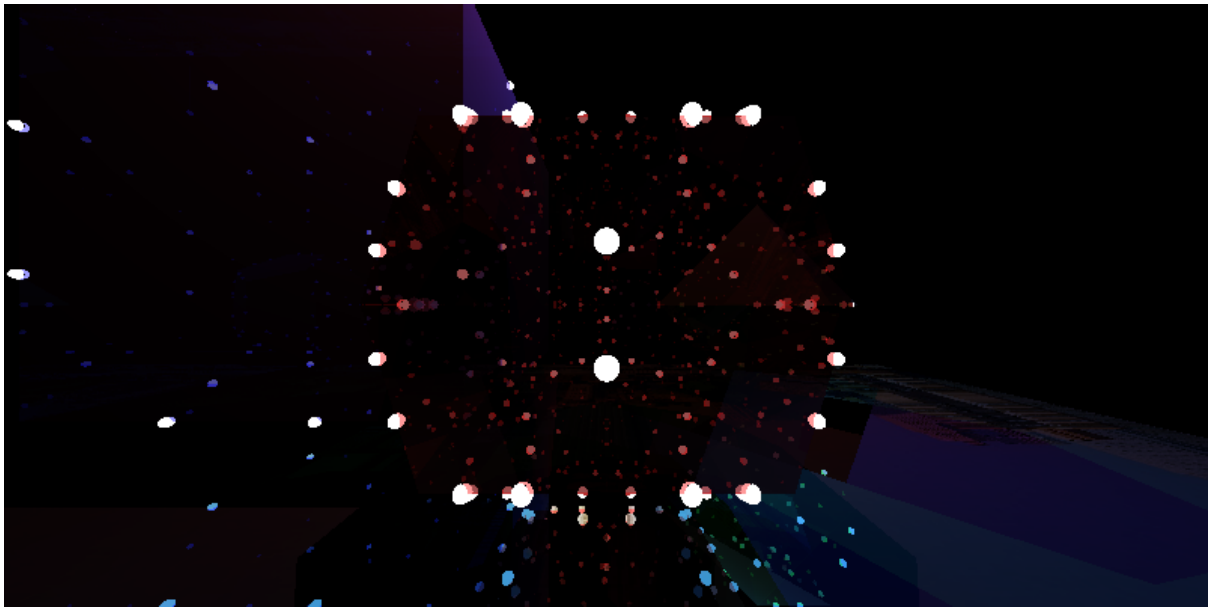


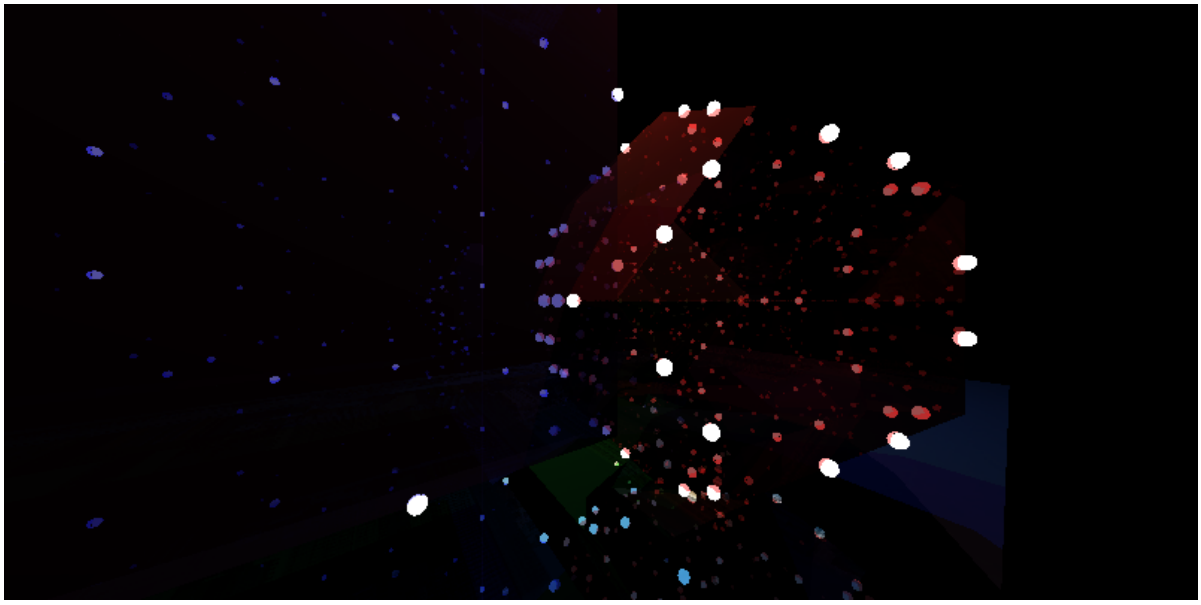


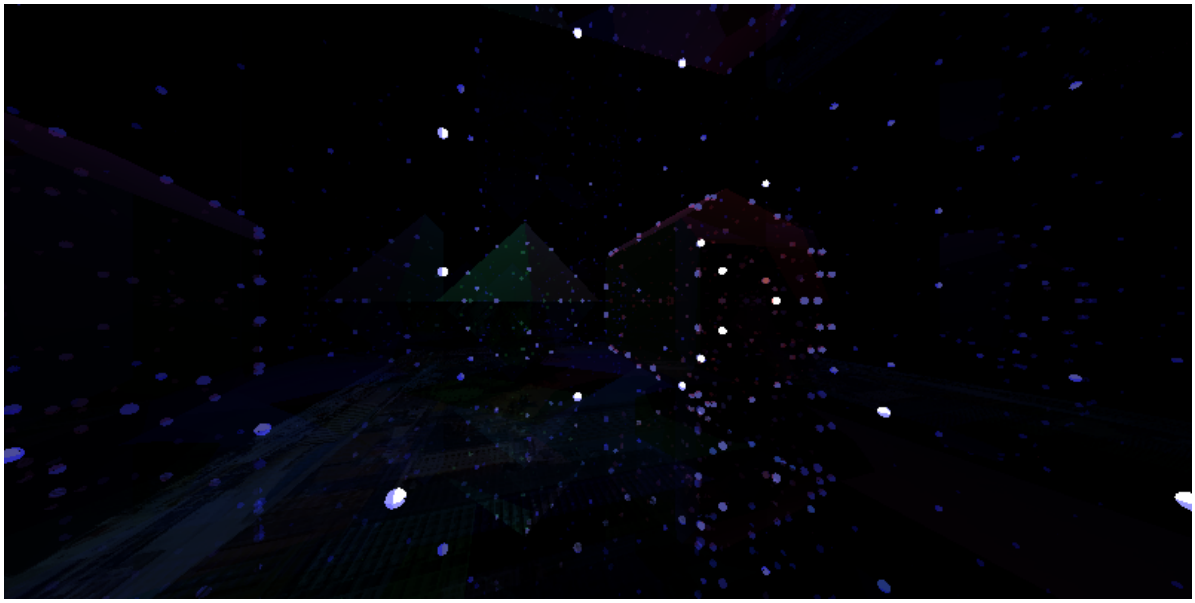


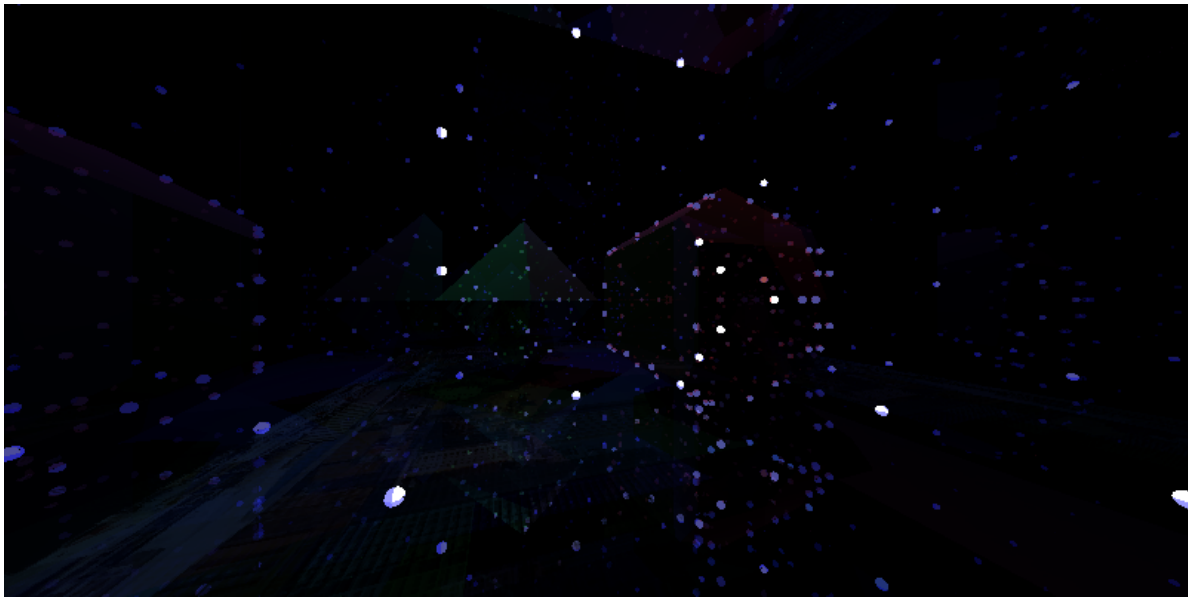


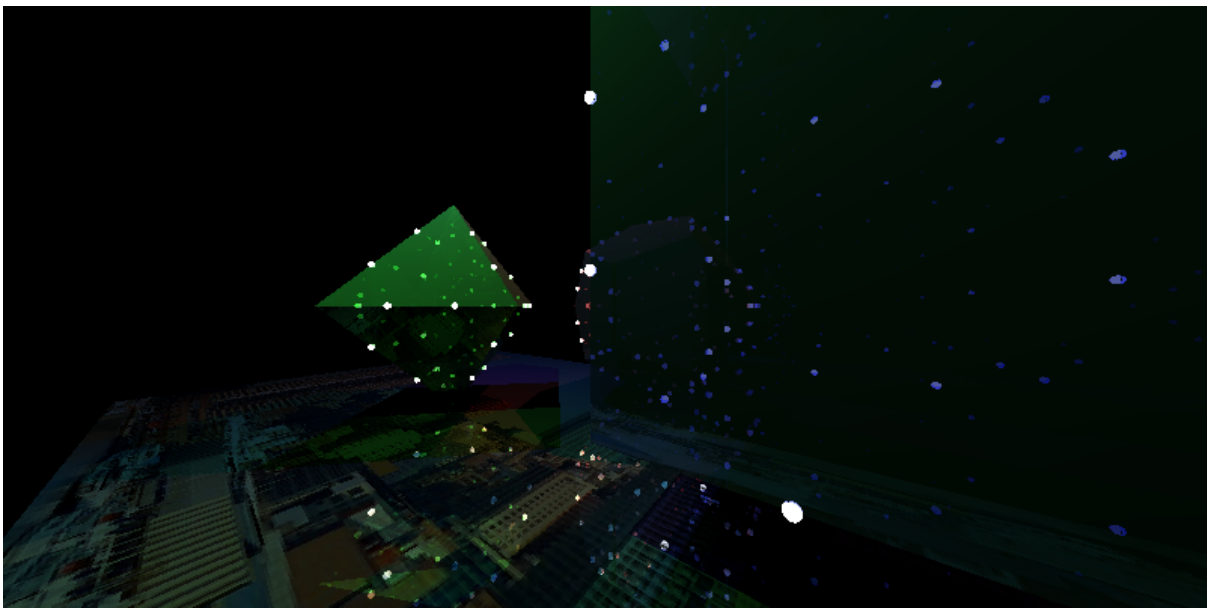












Суммарно на рендеринг такой сцену ушло приблизительно 35 минут.

Выводы

Алгоритм обратной трассировки лучей при правильной реализации позволяет добиться таких результатов, которые сложно достичь другими техниками рендеринга. Единственным его недостатком и, причем существенным, является огромная прожорливость, он потребляет значительное число вычислительных мощностей. Именно поэтому применяется он при создании продуктов, где время не является критическим фактором, например при создании фильмов/мультфильмов. С другой стороны он легко поддается распараллеливанию, например пиксели внутри одного кадра не зависят друг от друга, а значит могут вычисляться одновременно, тоже можно сказать и про последовательность кадров, при рендеринге анимации, кадры хоть и зависят друг от друга, но нам необязательно вычислять их последовательно. Еще одним плюсом алгоритма обратной трассировки лучей является логические отсечение невидимых поверхностей и наличие перспективы. Что касается сложности программирования, то базовый движок реализовать достаточно просто, но с уличением требований к производительности и качеству картинки и растет сложность, как мне кажется, растет геометрически.

Литература

1. www.scratchpixel.com
2. raytracing.github.io/books/RayTracingInOneWeekend