МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Компьютерные науки и прикладная математика»

Курсовой проект
по курсу «Технологии параллельного программирования»

Обратная трассировка лучей (Ray Tracing) на GPU

Выполнил: И.В. Сектименко
Группа: М8О-410Б-22
Преподаватель: А.Ю. Морозов

Москва, 2025

Условие

Описание задачи:

1. Цель работы, общая постановка задачи.
   Использование GPU для создания фотореалистичной визуализации. Рендеринг правильных геометрических тел без рекурсии, без текстур, без отражений, с одним источником света. Создание анимации.
2. Вариант задания №3.
   На сцене должны располагаться 3 тела: тетраэдр, гексаэдр, икосаэдр.

Программное и аппаратное обеспечение

Характеристики:

- графического процессора:

compute capability:	7.5
графическая память:	15828320256
разделяемая память на блок:	49152
константная память:	65536
количество регистров на блок:	65536
максимальное количество блоков:	(2147483647, 65535, 65535)
максимальное количество нитей на блок:	(1024, 1024, 64)
количество мультипроцессоров:	40

- процессора: Intel(R) Xeon(R) CPU @ 2.00GHz;
- оперативной памяти: 12Гб общей памяти;
- жесткого диска: 256 Гб общей памяти.

Во время выполнения работы использовалась IDE Google Colab. В нее встроена ОС Ubuntu 22.04.4 LTS с видеокартой NVIDIA-SMI 550.54.15.

Метод решения

Суть обратной трассировки лучей заключается в том, чтобы «выпустить» из камеры к каждому пикселю выходного изображения луч и посчитать цвет этого пикселя. При этом для каждого выпущенного луча необходимо определить, в какие фигуры (грани фигур, состоящие из треугольников), он попал и покрасить пиксель в цвет ближайшей к камере фигуры.

Любая точка p в плоскости треугольника может быть записана в барицентрических координатах:

$$p(u, v) = a + ue_1 + ve_2,$$

где $e_1 = b - a$,

$e_2 = c - a$,

a, b, c – точки треугольника.

Луч представляется, как:

$$r(t) = o + td,$$

где o – начало луча,

d – направление луча.

Тогда точку пересечения луча можно найти, решив следующую систему уравнений:

$$o + td = a + ue_1 + ve_2.$$

Если $u < 0$ или $u > 1$, то точка вне треугольника.

Если $v < 0$ или $u + v > 1$, то точка вне треугольника.

Если $t < \varepsilon$, то точка позади камеры.

Помимо этого, необходимо определить световую составляющую в данной точке. Свет определяется по модели Фонга: свет в точке – сумма окружающего (ambient) и рассеянного (diffuse) света.

Окружающий свет вычисляется, как цвет объекта, помноженный на маленькую константу. Так моделируется то, что даже в тени объект не черный, а имеет свой пусть и очень затемненный цвет.

Диффузная составляющая вычисляется следующим образом:

$$I_{diffuse} = k * (N * L) * I_{light},$$

где k – диффузный коэффициент отражения,

N – нормализованная нормаль к поверхности в точке,

L – нормализованный вектор от точки к источнику света,

I_{light} – интенсивность света от источника.

Так же в моей программе реализованы отражения. Вычисляется отраженный луч, находится цвет пикселя, куда попал отраженный луч, и этот цвет «примешивается» к текущему.

Описание программы

Программа написана в одном файле без использования классов, то есть в стиле функционального программирования.

Есть функция, которая создает сцену: в ней создаются прописанные в коде фигуры с учетом передаваемых параметров центров фигур и их масштабов.

После этого начинается функция, которая «запускает» лучи по всем пикселям выходного изображения. В зависимости от ключей, с которыми запускалась программа, программа запустить либо GPU версию, либо CPU версию функции. Все остальные функции: функция поиска пересечений луча с фигурами, функция вычисления цвета пикселя и др. – могут работать как на GPU, так и на CPU.

Также функции рендеринга изображения вычисляют, сколько всего лучей было выпущено, включая лучи из камеры и отраженные. На GPU для этого применяются атомарные операции.

Исследовательская часть и результаты

На рисунках 1 показан график зависимости времени (в мс) обработки разных кадров (2560x1920 и 5120x3840) от количества нитей.

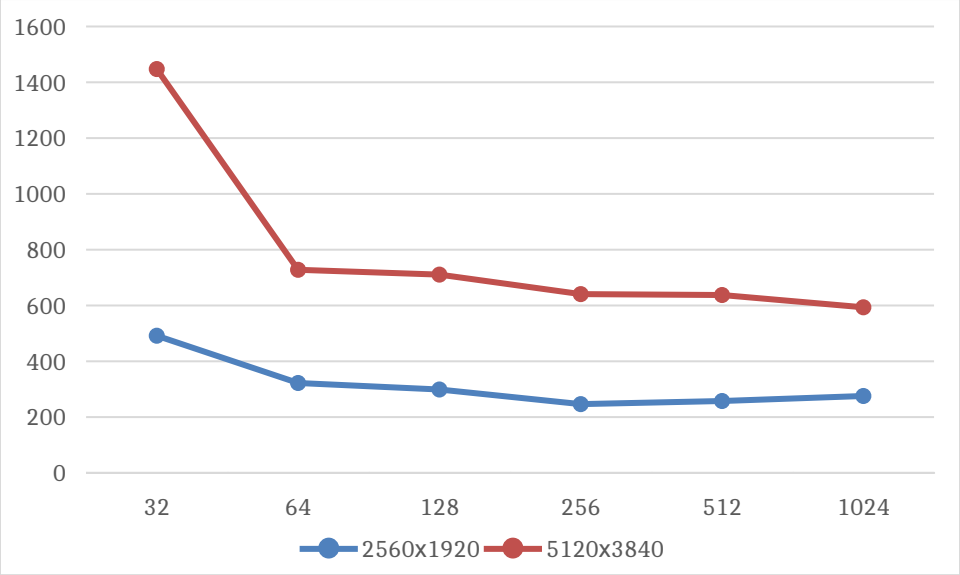

Рисунок 1 – Зависимость времени (в мс) обработки кадра от количества нитей

Результаты замеров производительности работы алгоритма на CPU в зависимости от размера кадра представлены в таблице ниже.

	2560x1920	5120x3840
CPU	75,552 с	306,429 с

Ниже привести входные данные, на которых получается наиболее красочный результат:

1
res/%d.data
5120 3840 120
4.0 1.0 0.0 0.0 0.0 0.0 0.0 0.1 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
2 0 2 0.5 0.0 0.5
-2.5 0.5 1.5 1 0.0 0.5
0 0 2 1 0.0 0.5
1
0 0 5

На рисунках 2-6 показано несколько трехмерных графиков, содержащих все полигоны сцены, траекторию облета камеры и траекторию направления камеры.

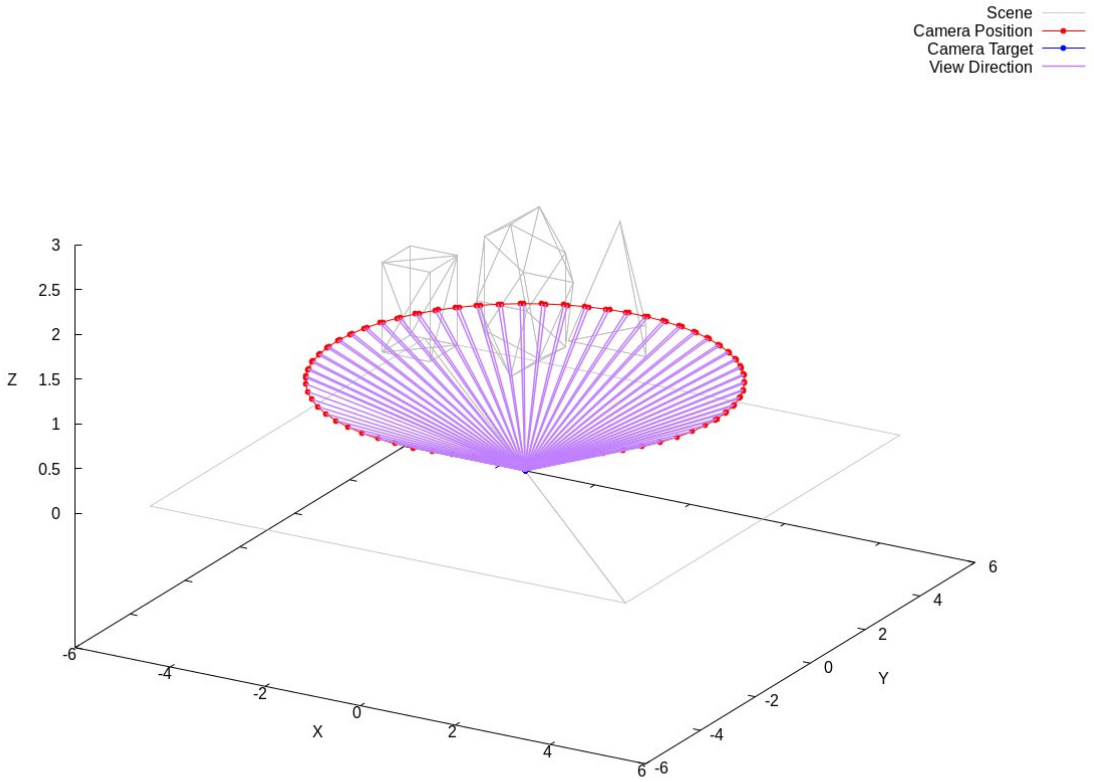

Рисунок 2 – Трехмерный график сцены с траекториями облета и направления камеры

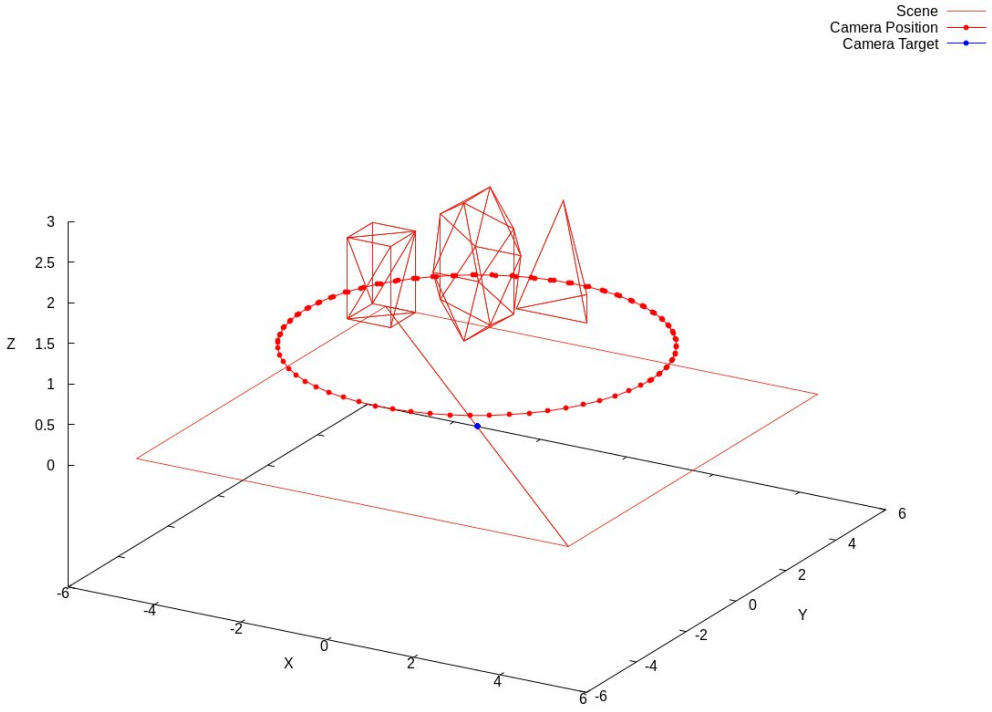

Рисунок 3 – Трехмерный график сцены с траекторией облета камеры

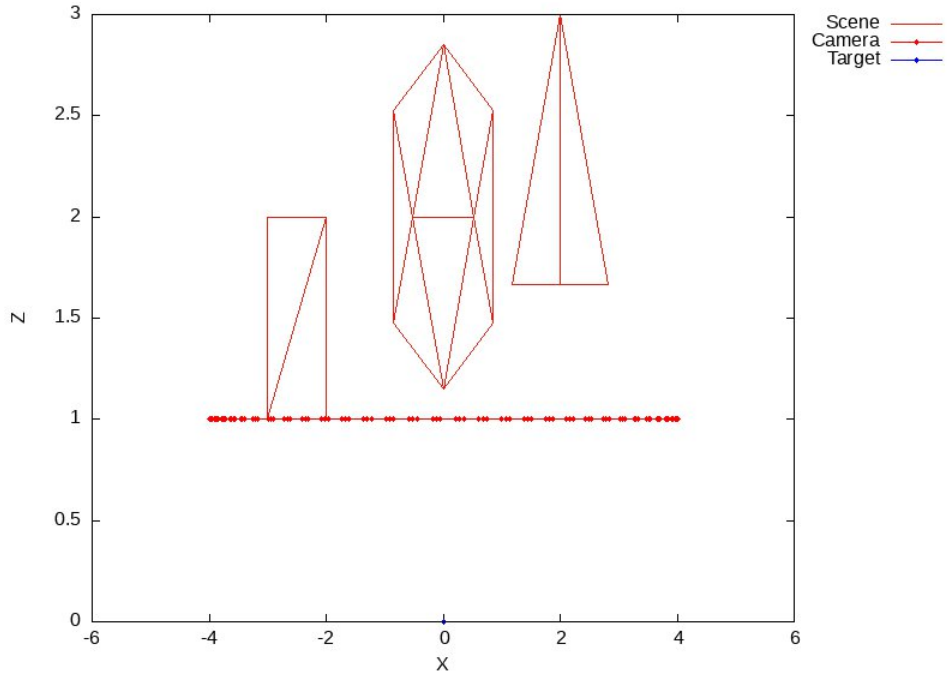

Рисунок 4 – Трехмерный график сцены с траекторией облета камеры в плоскости XZ

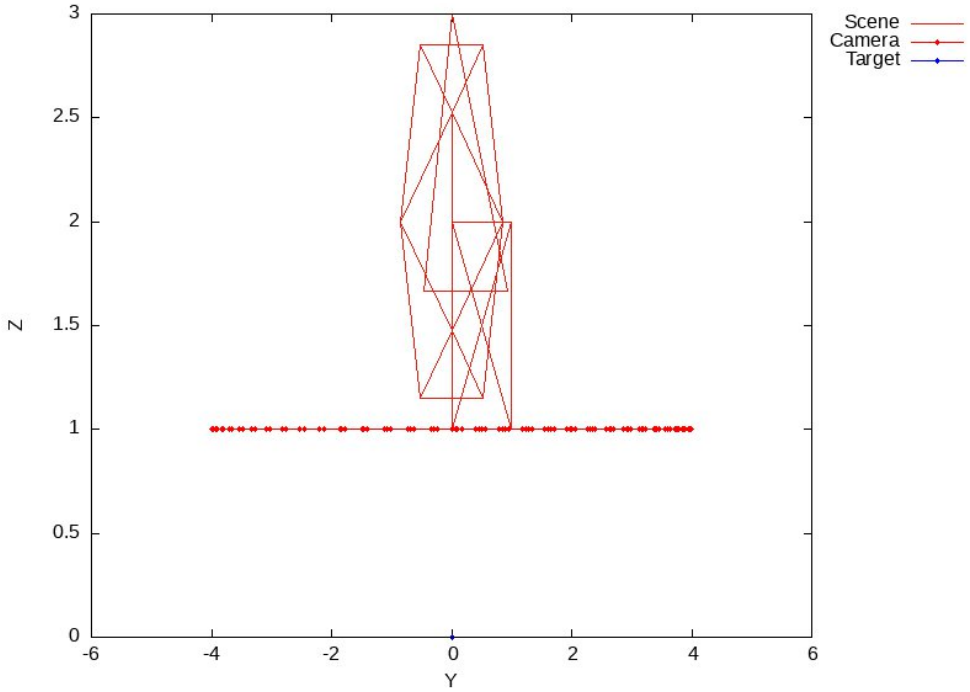

Рисунок 5 – Трехмерный график сцены с траекторией облета камеры в плоскости YZ

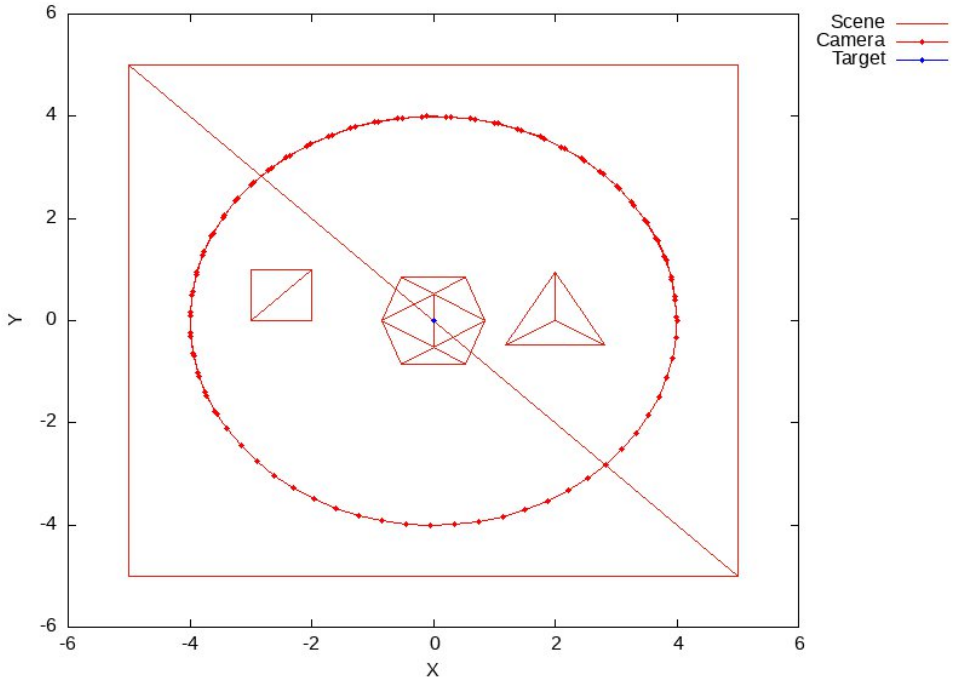

Рисунок 6 – Трехмерный график сцены с траекторией облета камеры в плоскости XY

На рисунках 7-10 показаны результаты работы программы с разных ракурсов.

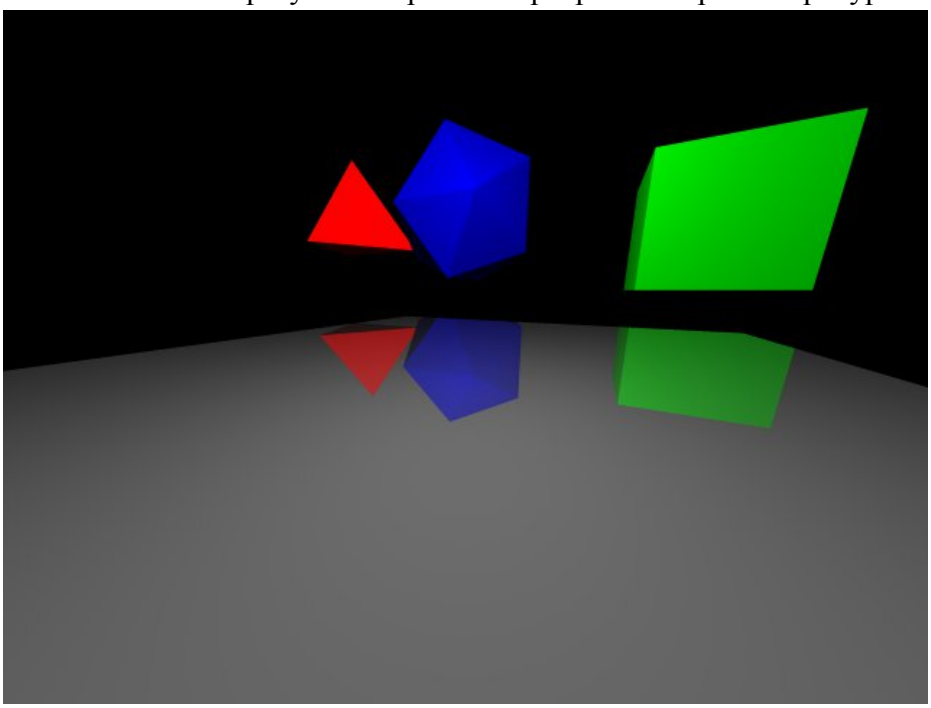

Рисунок 7 – Отрендеренная сцена, вид «спереди»

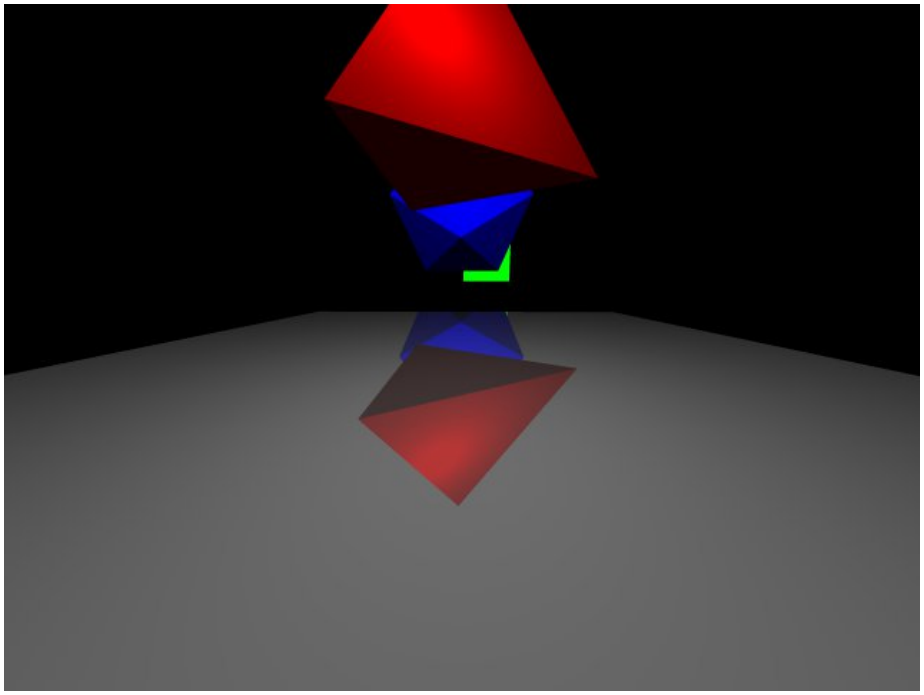

Рисунок 8 – Отрендеренная сцена, вид «сбоку» (левый бок)

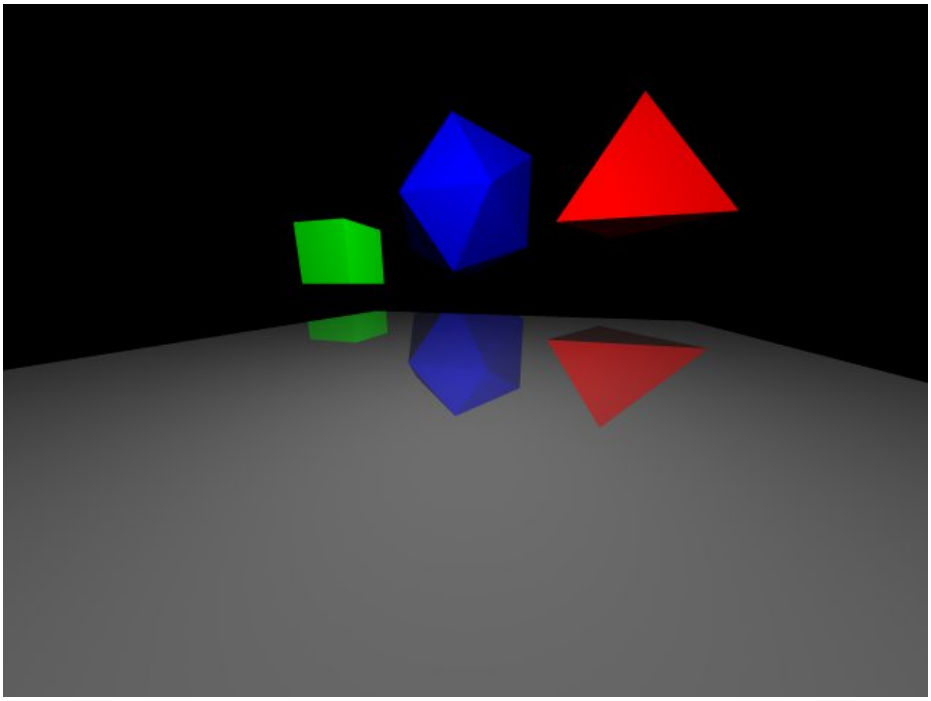

Рисунок 9 – Отрендеренная сцена, вид «сзади»

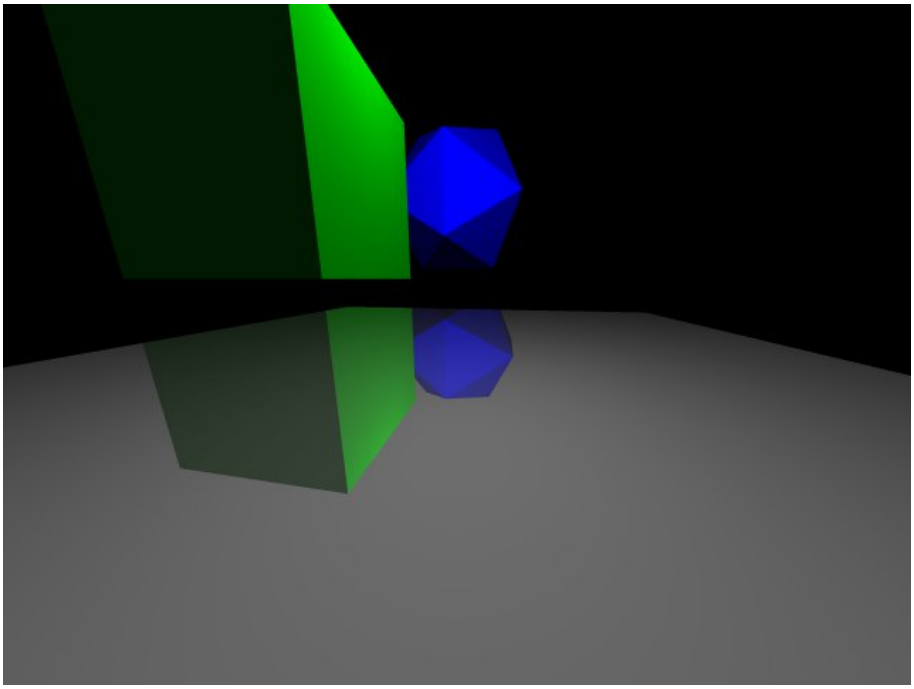

Рисунок 10 – Отрендеренная сцена, вид «сбоку» (правый бок)

Выводы

Обратный рейтрейсинг применяется для создания ярких спецэффектов и анимации, красивой картинки в играх. С помощью реализации законов отражения и преломления можно создать картинку, максимально приближенную к реальности.

Основная сложность программирования данного алгоритма заключается в знании линейной алгебры и законов физики, а запрограммировать готовый алгоритм не так уж сложно.

Обработка даже одного кадра анимации на GPU в несколько десятков раз быстрее, чем на CPU.

Литература

1. https://ray-tracing.ru/ – Трассировка лучей.
2. https://habr.com/ru/articles/441862/?ysclid=mjq2cr0vo434057541 – Затенение по Фонгу.