

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Компьютерные науки и прикладная математика»
Кафедра №806 «Вычислительная математика и программирование»

**Курсовой работа
по курсу «Параллельная обработка данных»**

Обратная трассировка лучей (Ray Tracing) на GPU

Выполнил: К.М. Воронов
Группа: 8О-407Б
Преподаватели: А.Ю. Морозов

Москва, 2022

Условие

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

Сцена. 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. Границы тел обладают зеркальным и прозрачным эффектом. За счет многократного переотражения лучей внутри тела, возникает эффект бесконечности.

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 (<http://www.ray-tracing.ru/>) 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 gnu и сри (т.е. дополнительно нужно реализовать алгоритм без использования CUDA).

Программное и аппаратное обеспечение

GPU:

- Название NVIDIA GeForce RTX 3050
- Compute capability: 8.6
- Графическая память: 4100456448
- Разделяемая память: 49152
- Константная память: 65536
- Количество регистров на блок: 65536
- Максимальное количество нитей: (1024, 1024, 64)
- Максимальное количество блоков: (2147483647, 65535, 65535)
- Количество мультипроцессоров: 16

Сведения о системе:

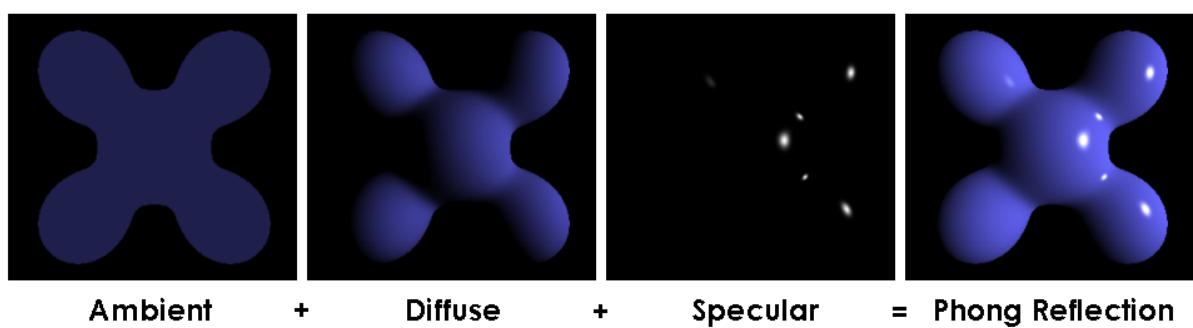
- Процессор: Intel Core i5-11400H 2.70GHz
- ОЗУ: 32 ГБ
- SSD 1ТБ

Программное обеспечение:

- OS: Linux Mint 21
- Текстовый редактор: Sublime text
- Компилятор: nvcc

Метод решения

На GPU реализован алгоритм обратной трассировки лучей с использованием развернутой рекурсии. Лучи из первого массива преломляются, отражаются и записываются во второй массив. После этого происходит сжатие второго массива до размеров первого, путем удаления пустых мест. Если места не хватает, размеры массивов увеличиваются вдвое. На CPU реализована обычная рекурсия. В качестве модели освещения используется затенение по Фонгу, которое состоит из трех частей:



Фоновое освещение (ambient) - это постоянная в каждой точке величина надбавки к освещению. Вычисляется фоновая составляющая освещения как:

$$I_a = k_a \cdot i_a, \text{ где}$$

I_a – фоновая составляющая освещенности в точке,

k_a – свойство материала воспринимать фоновое освещение,

i_a – мощность фонового освещения.

Рассеянный свет (diffuse) при попадании на поверхность рассеивается равномерно во все стороны. При расчете такого освещения учитывается только ориентация поверхности (нормаль) и направление на источник света. Рассеянная составляющая рассчитывается по закону косинусов (закон Ламберта):

$$I_d = k_d \cdot i_l \cdot \cos(\vec{L}, \vec{N}) = k_d \cdot i_l \cdot (\vec{L} \cdot \vec{N}), \text{ где}$$

I_d – рассеянная составляющая освещенности в точке,

k_d – свойство материала воспринимать рассеянное освещение,

i_l – интенсивность точечного источника,

L – направление из точки на источник света,

N – вектор нормали в точке.

Зеркальный свет (specular) при попадании на поверхность подчиняется следующему закону: “Падающий и отраженный лучи лежат в одной плоскости с нормалью к отражающей поверхности в точке падения, и эта нормаль делит угол между лучами на две равные части”. Т.о. отраженная составляющая освещенности в точке зависит от того, насколько близки направления на наблюдателя и отраженного луча. Это можно выразить следующей формулой:

$$I_s = i_l \cdot k_s \cdot \cos^p(\vec{R}, \vec{S})$$

I_s - спектральная составляющая освещения в точке

p - степень, аппроксимирующая пространственное распределение света.

R - отражённый луч

S - вектор наблюдения

Для определения первого полигона, который пересекает луч используются барицентрические координаты:

- $u := u/S, v := v/S, t1 := t1/S$
- $t1 = 1 - u - v$

И, соответственно, расчет происходит следующим образом:

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

$$E1 = v1 - v0$$

$$E2 = v2 - v0$$

$$T = p - v0$$

$$P = \text{cross}(D, E2)$$

$$Q = \text{cross}(T, E1)$$

$$D = v$$

Описание программы

Материалы, источники света, шарики в фигурах и полигоны хранятся в соответствующих структурах.

GPU:

Для обработки лучей используется структура ray, первый массив rays и второй rays2. Реализованы следующие ядра:

1. kernel_begin_rays - заполняет начальные лучи, обнуляет значения картинки
2. clear_rays2 - очищает вектор лучей
3. count_rays2 - считает количество лучей в массиве
4. kernel_rays - основная часть работы, расчет освещения, генерация лучей
5. kernel_compact - сжатие массива лучей
6. kernel_ssaa - реализация SSAA

CPU:

Расчет цветов картинки запускается в функции render, рекурсия реализована в функции raytr. Реализация SSAA находится в функции main.

Для расчета освещения, пересечения с полигоном, определения цвета пола по текстуре реализованы функции phongShade, intersection и textColor, которые работают как на CPU, так и на GPU.

Исследовательская часть и результаты

Графики

По х - конфигурация, по у - время в мс

Параметры сцены:

150

res/%d.data

640 480

6 5 1.5707963267948966 0 2 0 2.0053522829578814 1.0026761414789407 0 0

3 0 4.71238898038469 0 0 0 0 1.0026761414789407 0 0

-2 0 2 0.1 0.6 0 1 0.7 1 5

0 3 3 0.5 0.5 0.5 1 0.5 1 2

3 -3 1 0.7 0 0 1.5 0.7 1 4

-5 -5 0 5 -5 0 -5 5 0 5 5 0

text.data

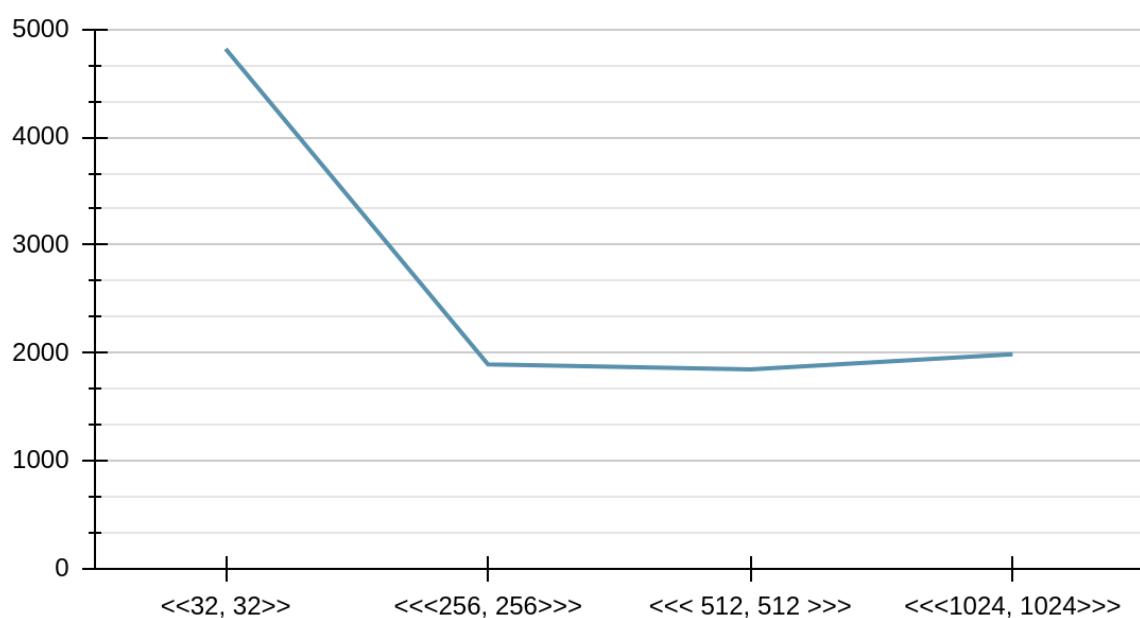
1 1 1 0

1

0 0 14 1 1 1

5 1

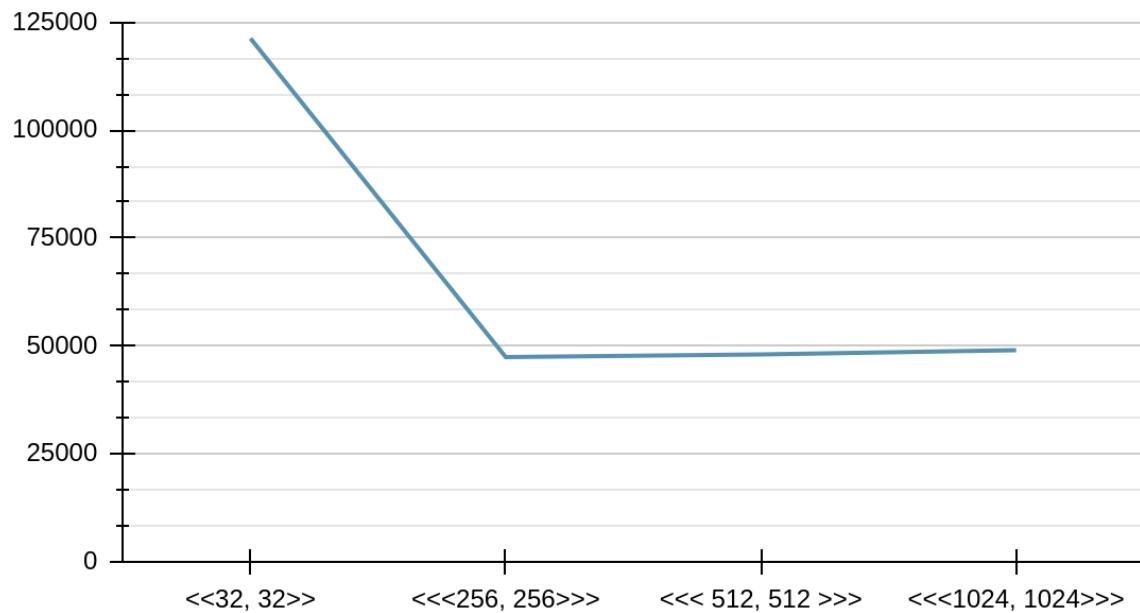
GPU



Параметры сцены:

Такие же как выше, но с разрешением 1920x1080 и сглаживанием 2x.

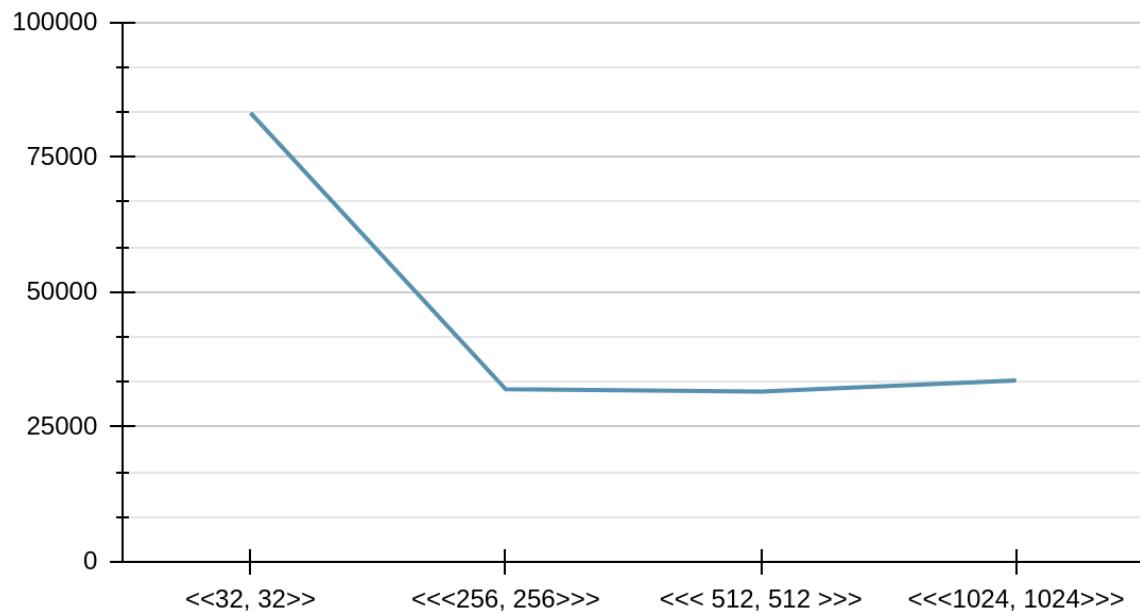
GPU



Параметры сцены:

Как 1, но с разрешением 640x480, сглаживанием 4x, двумя источниками света

GPU

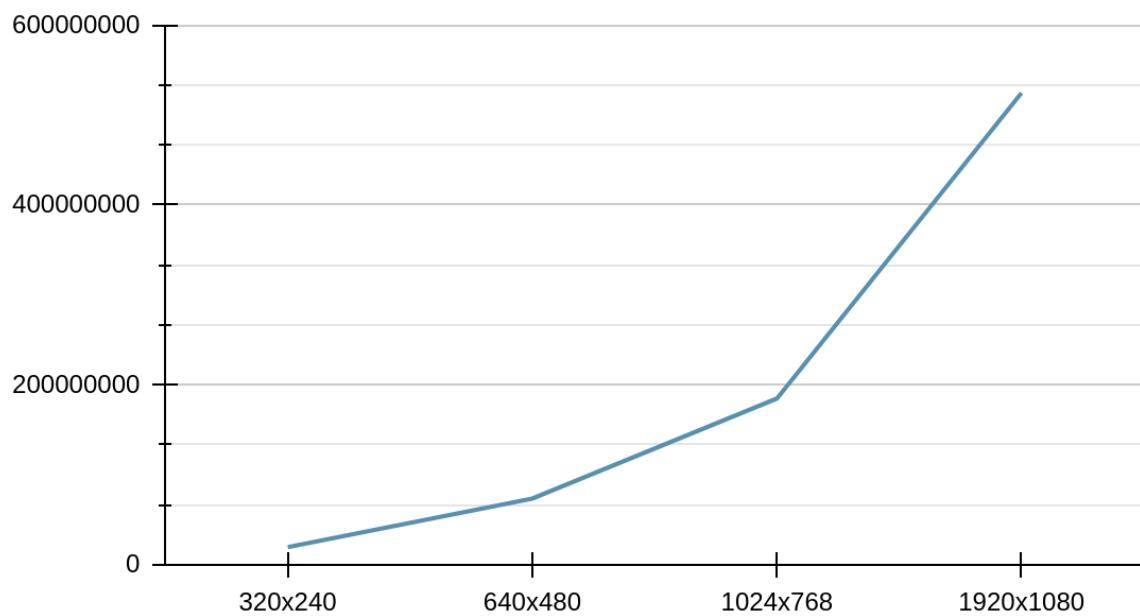


CPU:

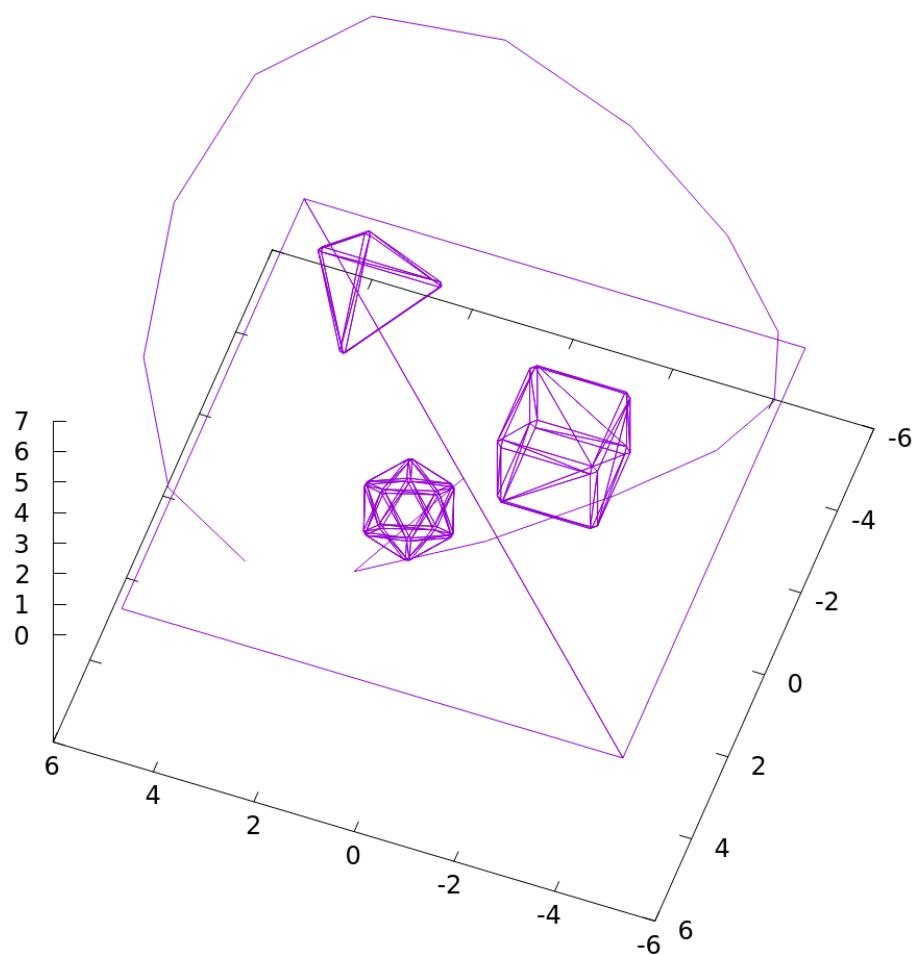
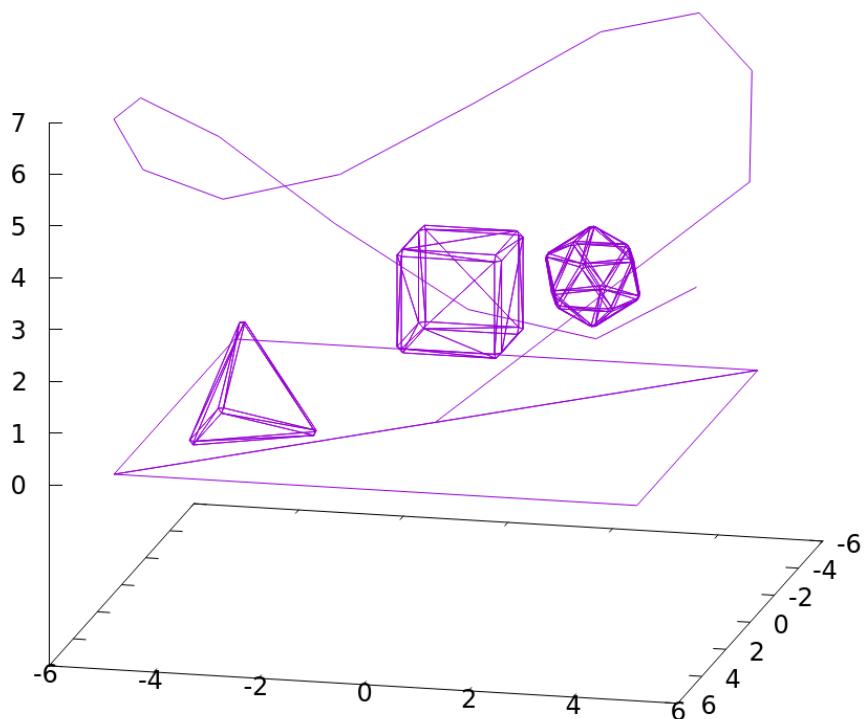
По х - разрешение, по у - время в мс

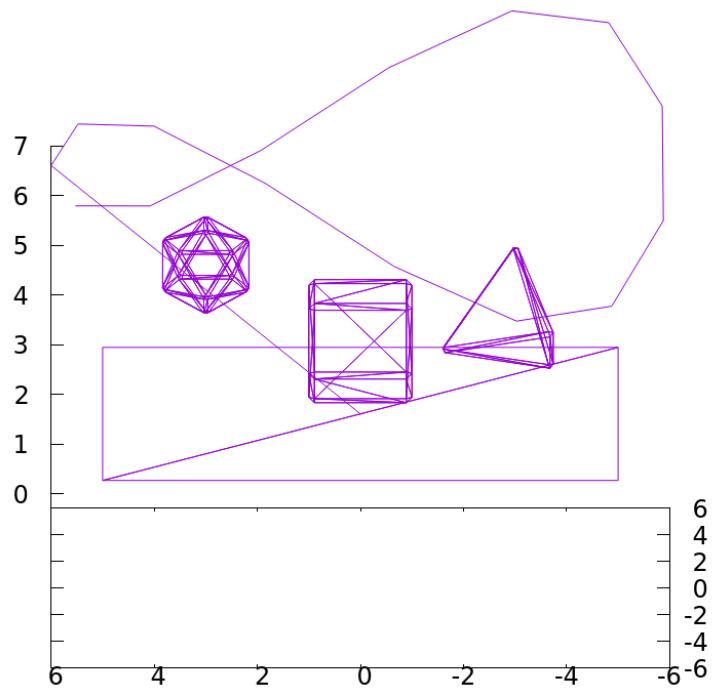
Взяты первые параметры сцены

CPU



Расположение фигур на сцене





Результаты работы

Параметры сцены

150

res/%d.data

1920 1080

6 5 1.5707963267948966 0 2 0 2.0053522829578814 1.0026761414789407 0 0

3 0 4.71238898038469 0 0 0 0 1.0026761414789407 0 0

-2 0 2 0.1 0.6 0 1 0.7 1 5

0 3 3 0.5 0.5 0.5 1 0.5 1 2

3 -3 1 0.7 0 0 1.5 0.7 1 4

-5 -5 0 5 -5 0 -5 5 0 5 5 0

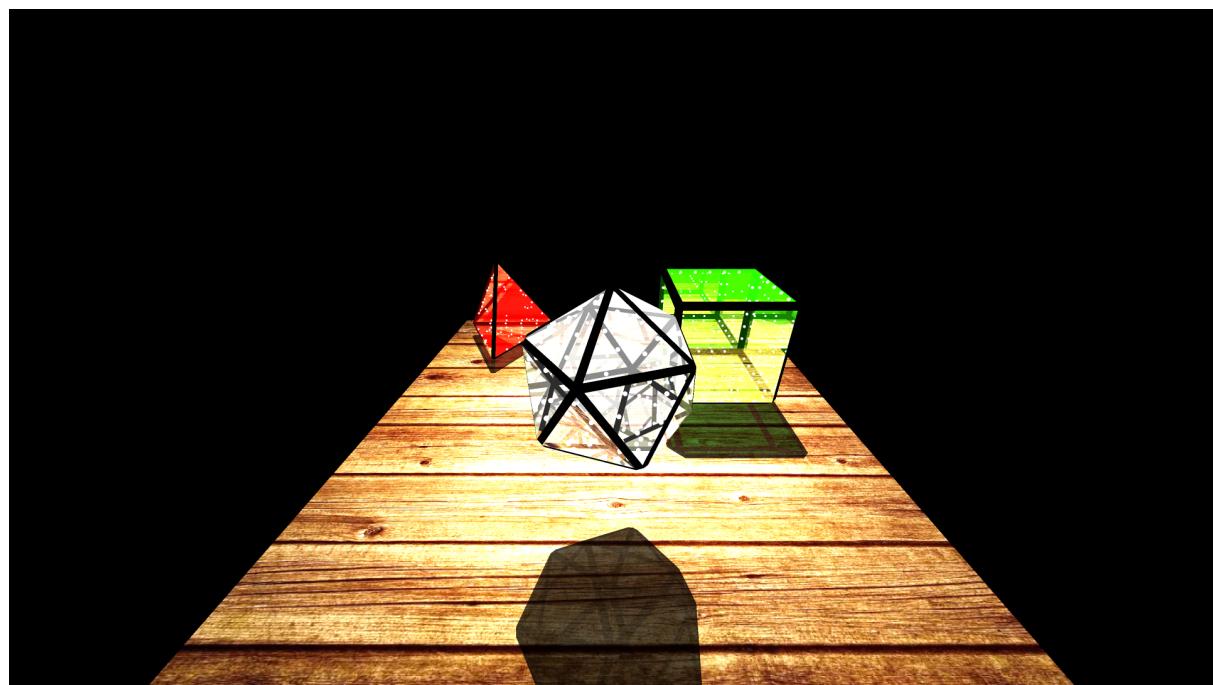
text.data

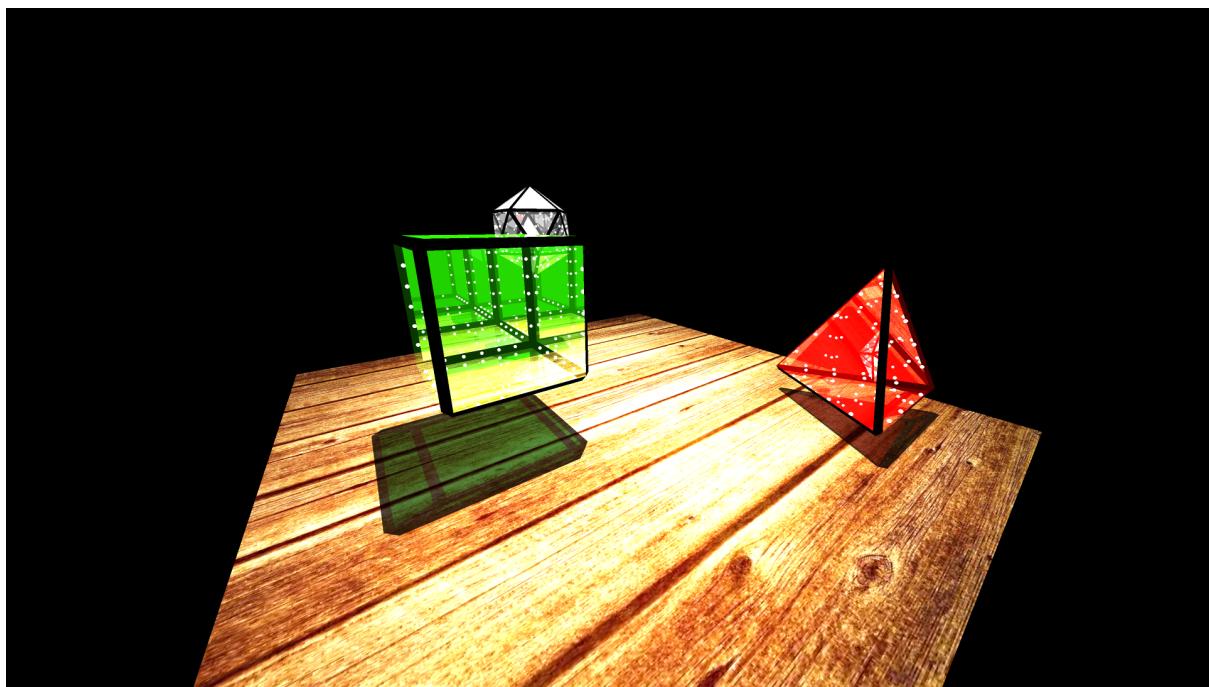
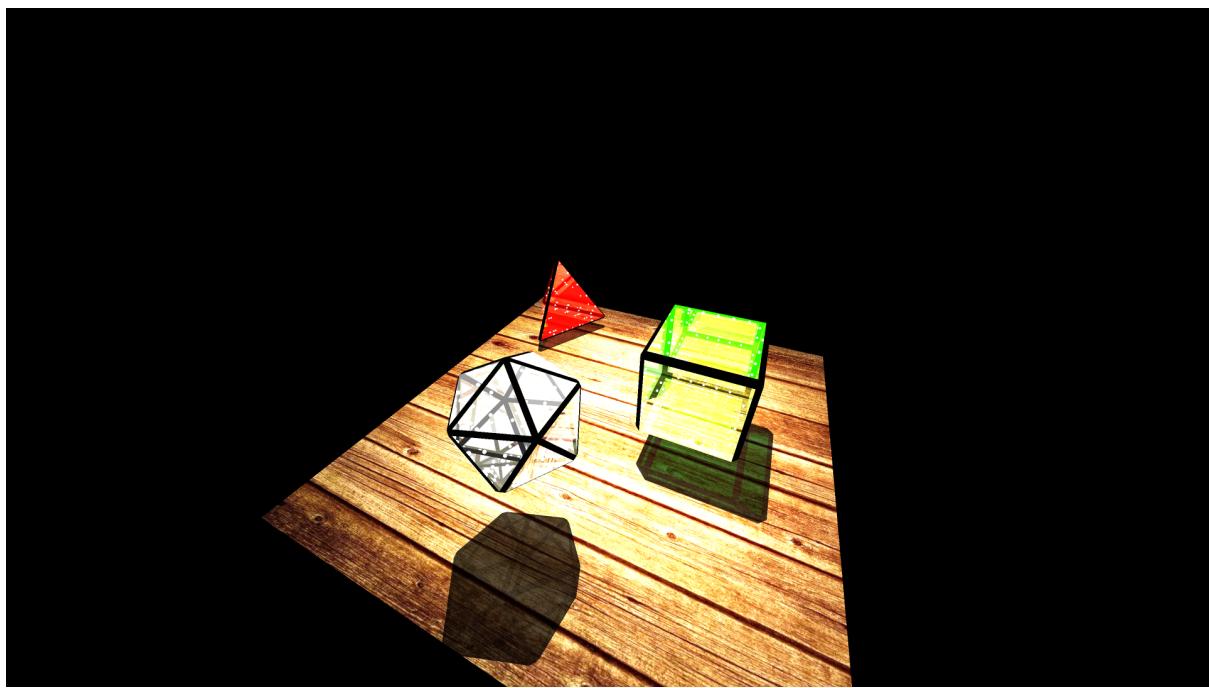
1 1 1 0

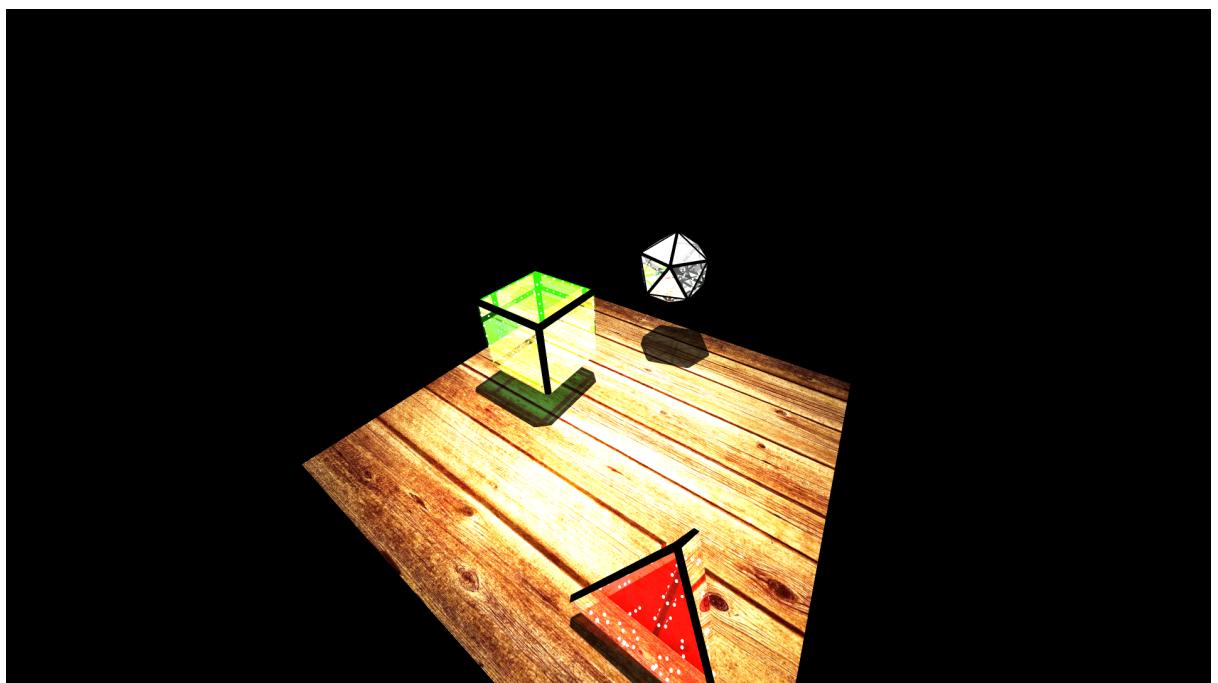
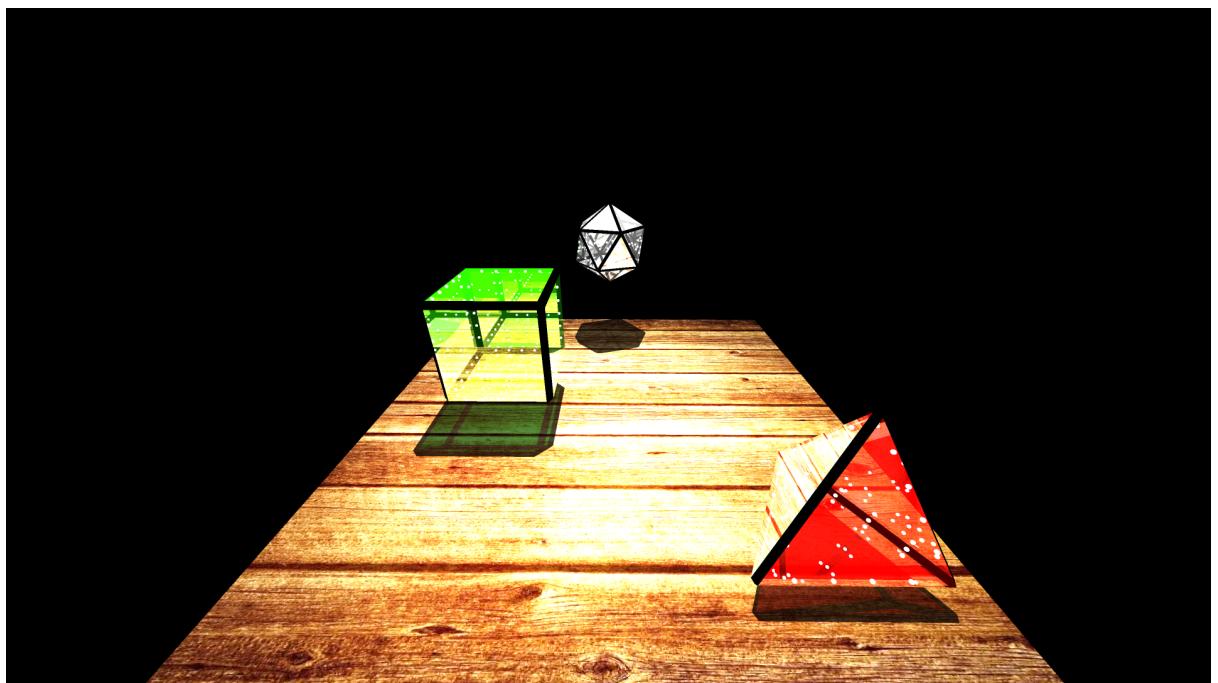
1

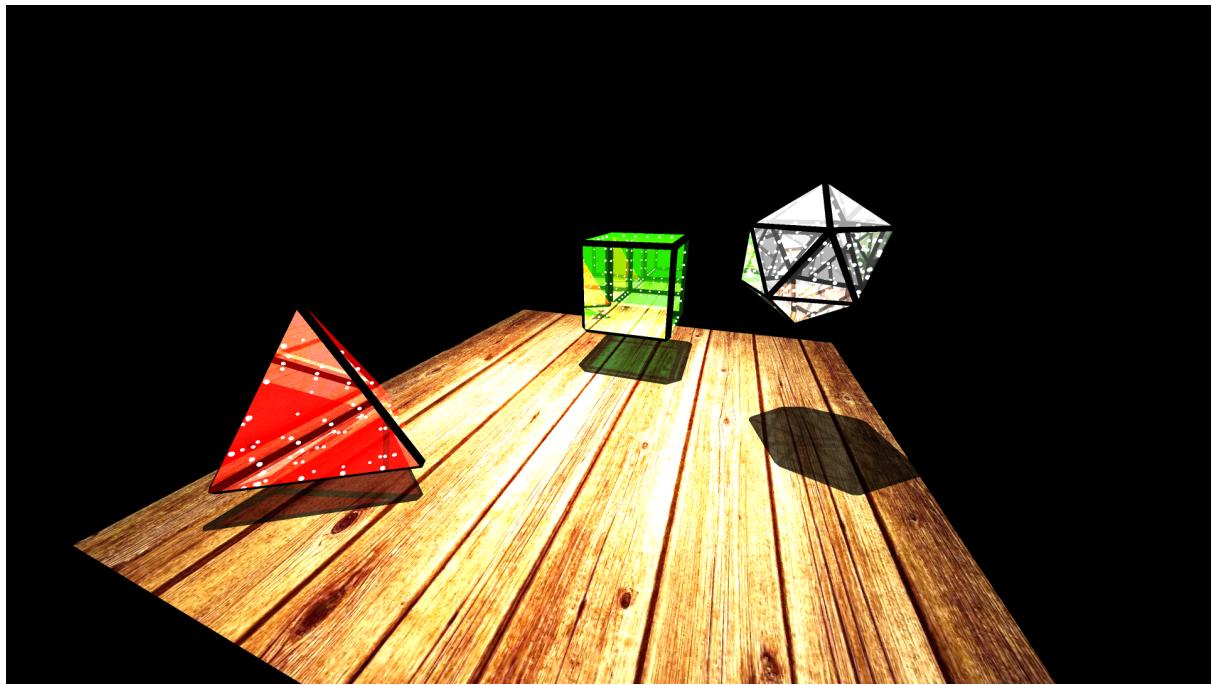
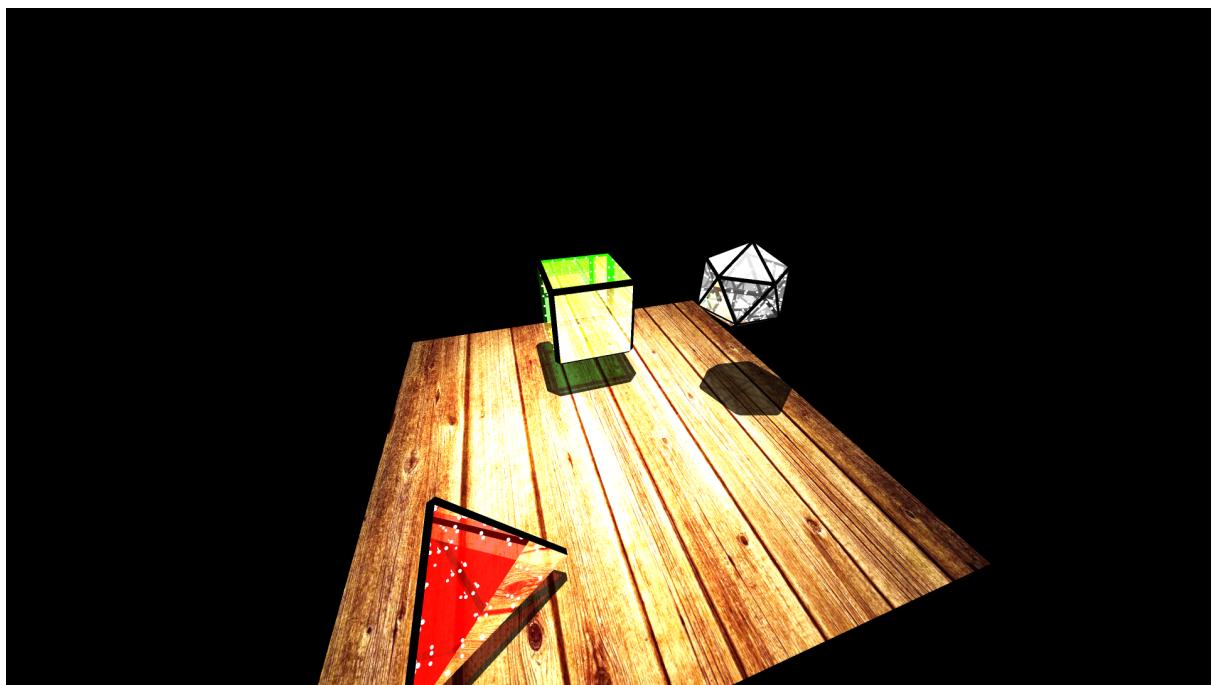
0 0 14 1 1 1

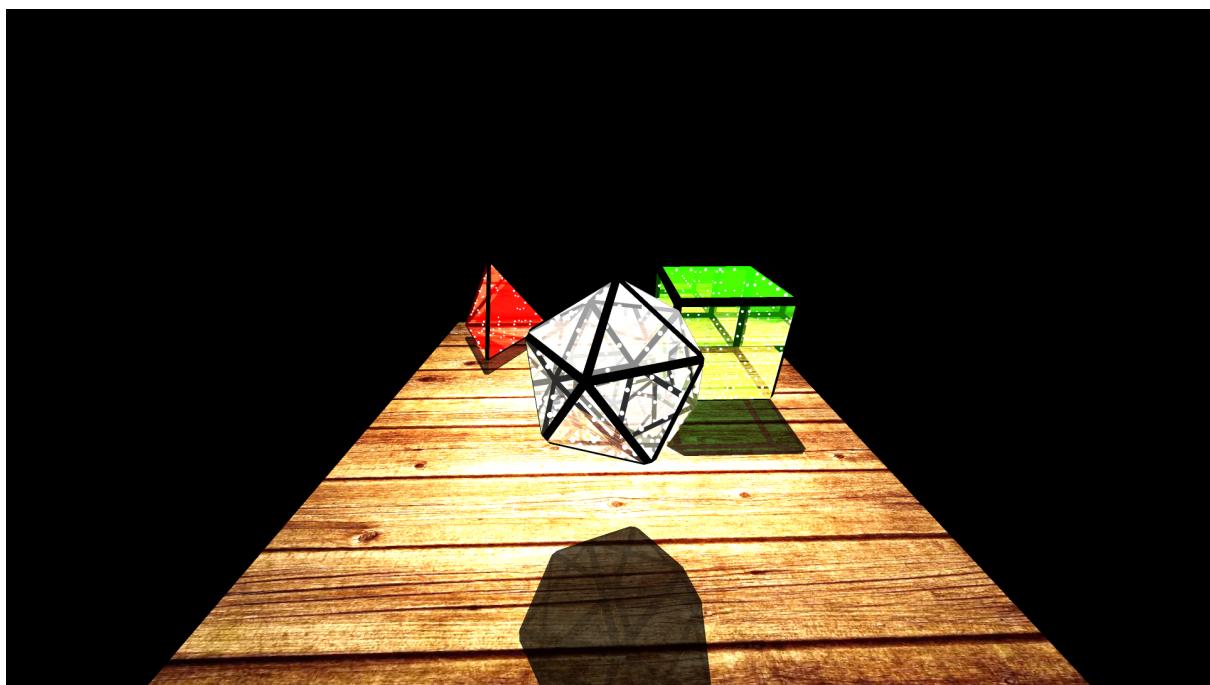
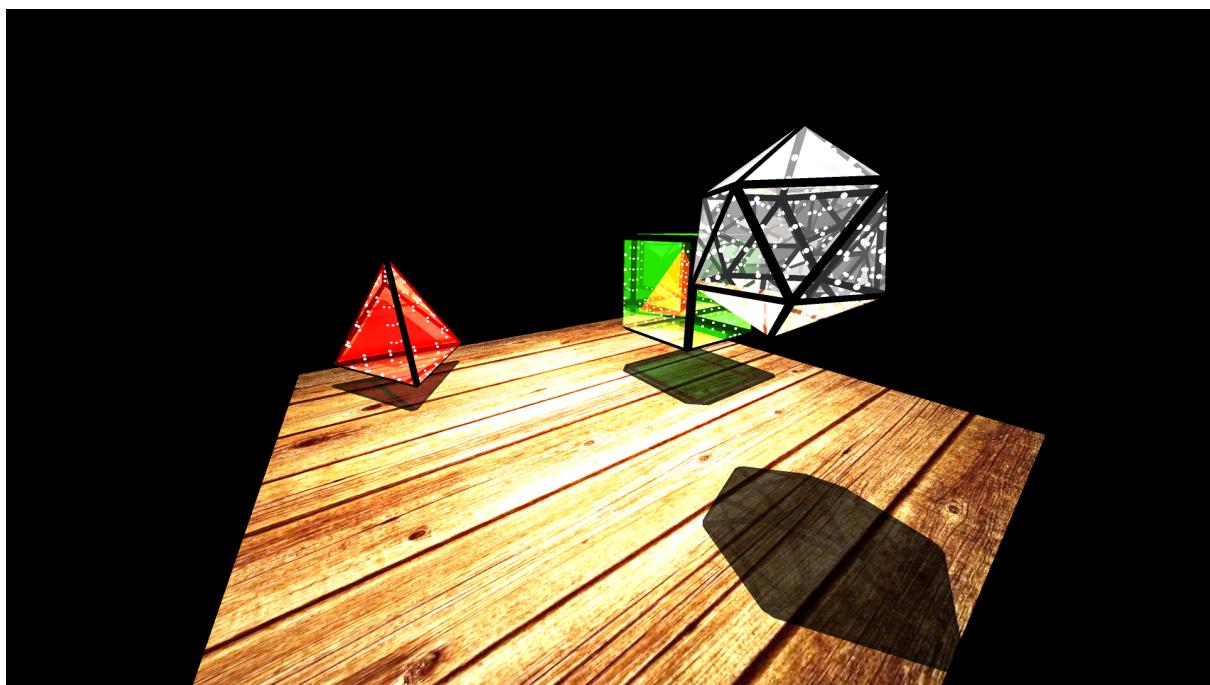
5 2

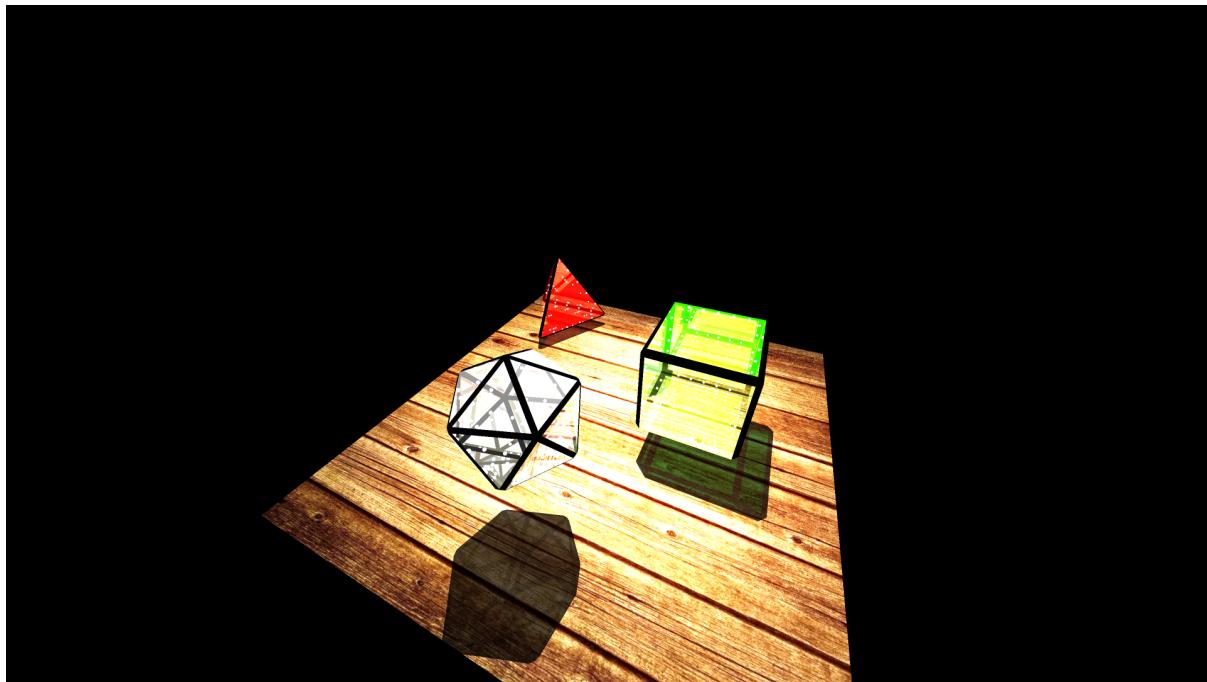












Выводы

Данный алгоритм делает картинку очень реалистичной и красивой, но он требует больших ресурсных затрат, так как лучей очень много, и каждый из них надо обрабатывать. Во время выполнения я столкнулся с тем, что atomicAdd не перегружен для unsigned char и double, что привело к тому, что мне пришлось корректировать архитектуру кода. Также были проблемы с синхронизацией потоков при сжатии массивов и отображении координат пола на текстуру. В целом курсовой проект был очень занимателен в плане результатов, и очень тяжёлым в плане кода. Реализация алгоритма на видеокарте ускорило вычисление картинок в несколько раз, что еще раз подтверждает удобность использования GPU.

Литература

1. Алгоритм обратной трассировки лучей URL: <http://www.ray-tracing.ru/>
2. Лекции Морозова А.Ю., Московский авиационный институт
3. Лекции Морозова А.В., Московский авиационный институт