МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Информационные технологии и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

**Курсовая работа**

**по курсу «Параллельная обработка данных»**

**Обратная трассировка лучей (Ray Tracing) на GPU**

Выполнил: Г.Н. Хренов

Группа: 8О-407Б

Преподаватели: К.Г. Крашенинников,

А.Ю. Морозов

Москва, 2021

**Условие**

1. Цель работы: использование GPU для создания фотореалистической визуализации. Рендеринг полузеркальных и полупрозрачных правильных геометрических тел. Получение эффекта бесконечности. Создание анимации.
2. Вариант 10. Октаэдр, Додекаэдр, Икосаэдр.

**Программное и аппаратное обеспечение**

GPU name: NVIDIA GeForce RTX 2060

compute capability 7:5

totalGlobalMem: 6442450944

sharedMemPerBlock: 49152

totalConstMem: 65536

regsPerBlock: 65536

maxThreadsDim: 1024 1024 64

maxGridSize: 2147483647 65535 65535

multiProcessorCount: 30

CPU name: AMD Ryzen 7 3750H with Radeon Vega Mobile Gfx

MaxClockSpeed: 2300

NumberOfCourse: 4

RAM: 8

SSD: 256, HDD: 1024

OS: Windows10

Compiler: nvcc

**Метод решения**

Сцена состоит из 4 объектов – 3 фигуры и пол. Каждый объект строится из полигонов(треугольников), при этом важно указывать вершины полигонов в определенном порядке, чтобы сохранить направления нормалей. Освещение состоит из трех компонент: фоновое, диффузное и рассеянное, каждая компонента считается отдельно, а затем складываются. Для реализации отражения и преломления из точки пересечения луча и полигона необходимо заново запустить луч по физическим законам.

**Описание программы**

int Closest\_trig(const vec3 &pos, const vec3 &dir, float &ts\_min, trig\* trigs) { - нахождение ближайшего по направлению камеры полигона.

vec3 reflect(const vec3 &I, const vec3 &N) { - отражение, рассчитывается по углу падения и нормали

vec3 refract(const vec3 &I, const vec3 &N, const float eta\_t, const float eta\_i=1.0) – преломление, рассчитывается по углу падения, нормали и коэффициентам преломления сред

uchar4 get\_texture\_clr(uchar4 \*floor, float x, float y, int w, int floor\_w) { - наложение текстуры по пол, отображение точки текстуры на точку на полу

void build\_space(trig\* trigs, const vec3 &oct\_c, …) { - построение сцены, отрисовка всех полигонов пола, 3 фигур, а также ребер

uchar4 ray(vec3 pos, vec3 dir, int depth…) – непосредственно расчет траектории заданного луча, в зависимости от параметров, возвращает цвет заданного пикселя.

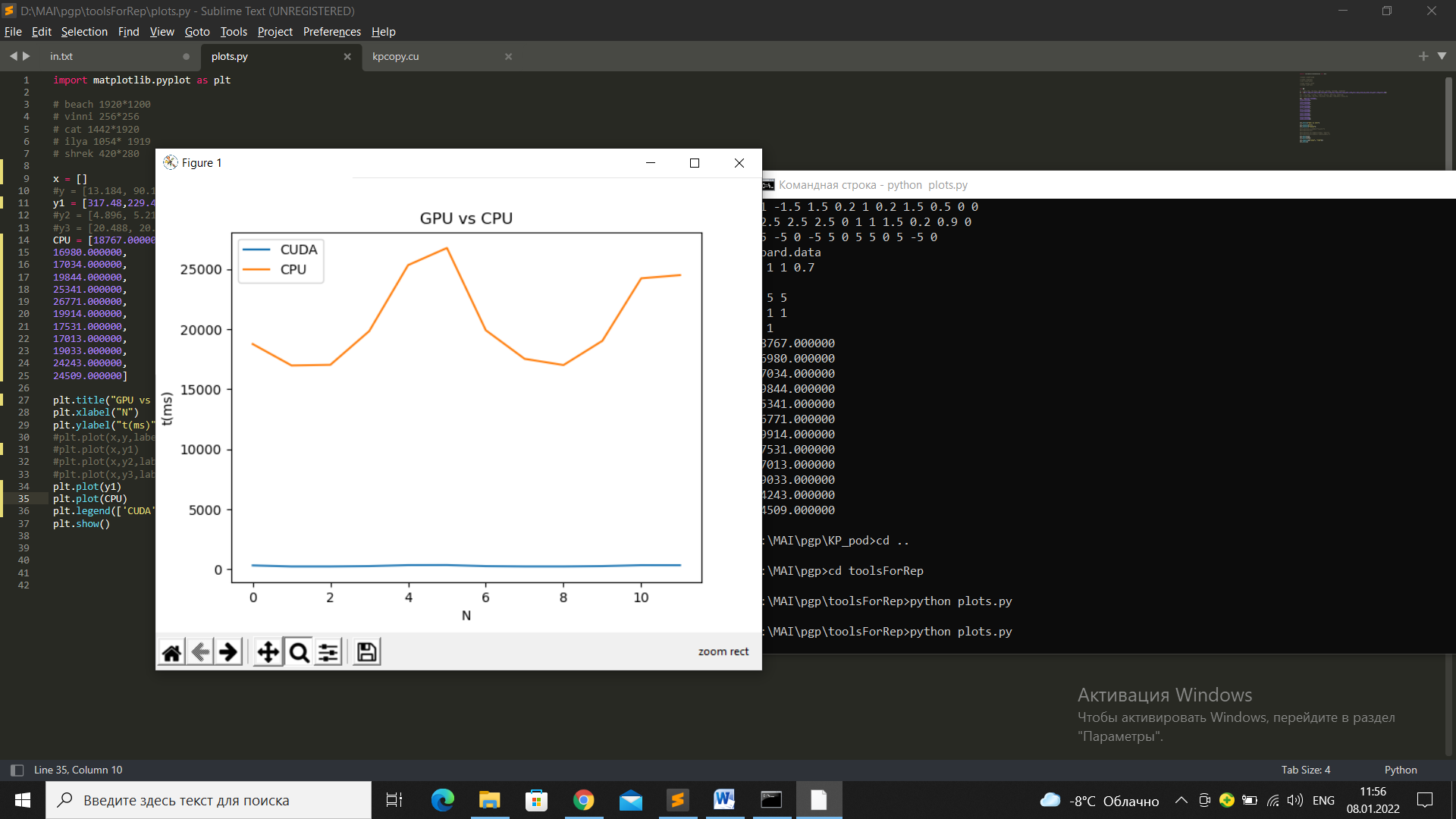
\_\_global\_\_ void kernel\_ssaa(uchar4\* data, uchar4\* out, int w, int h, int sqrt\_ray\_num) { - алгоритм ssaa для сглаживания, устранения эффекта зубчатости.

**Исследовательская часть**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Глубина рекурсии\конфигурация | dim3(8, 8), dim3(8, 8) | dim3(4, 16), dim3(4, 16) | dim3(16, 16), dim3(16, 16) | dim3(8, 32), dim3(8, 32) |
| 0 | 443.3ms | 446.3ms | 354.7ms | 307.8ms |
| 1 | 626.6ms | 625.6ms | 482.9ms | 471.6ms |
| 2 | 1101.4ms | 1140ms | 616.4ms | 587.2ms |

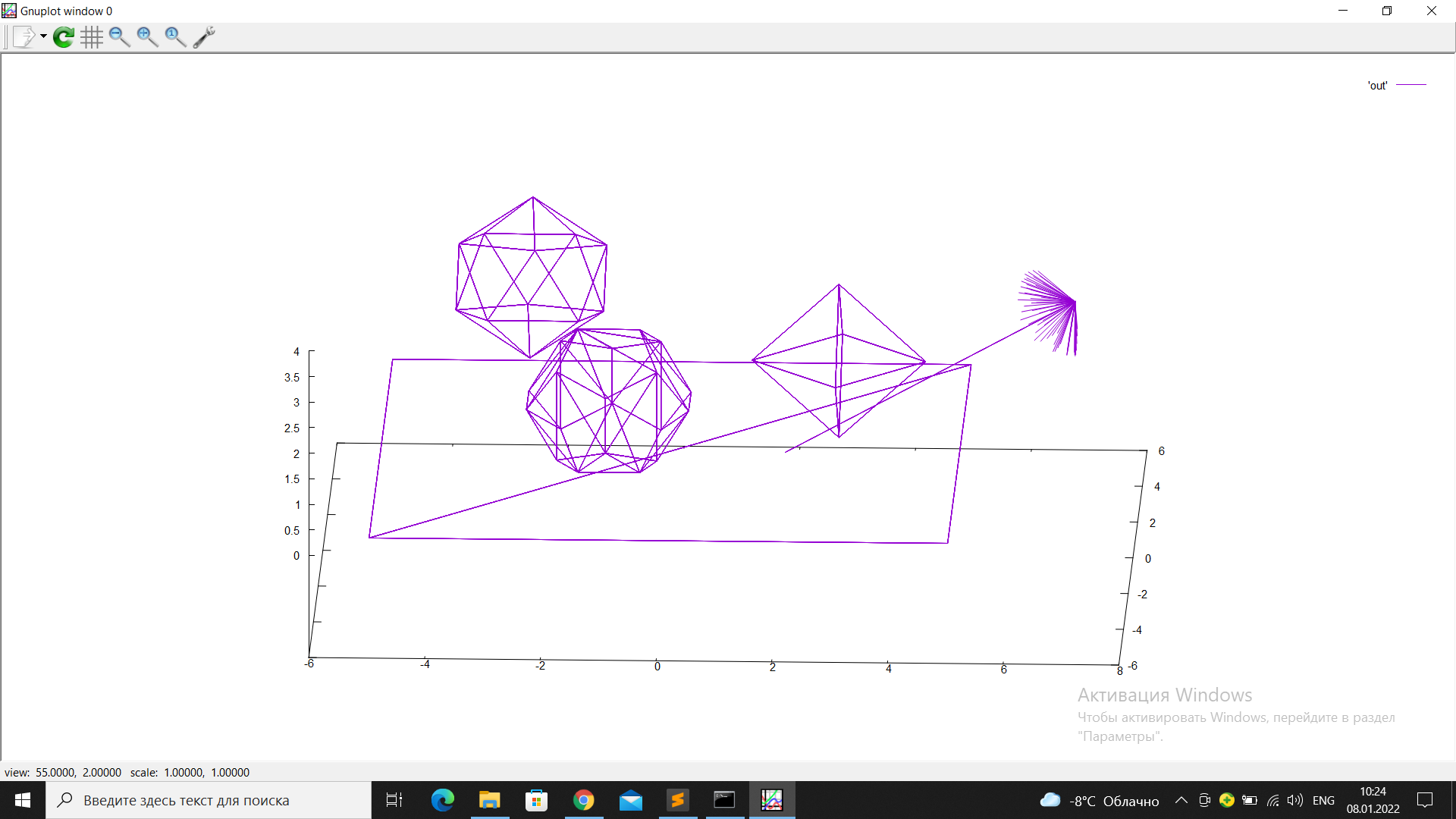
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Источники света\конфигурация | dim3(8, 8), dim3(8, 8) | dim3(4, 16), dim3(4, 16) | dim3(16, 16), dim3(16, 16) | dim3(8, 32), dim3(8, 32) |
| 2 | 501.8ms | 499.9ms | 382.9ms | 378.9ms |
| 3 | 548.2ms | 568.8ms | 436.2ms | 428.6ms |
| 4 | 618.1ms | 643.9ms | 482.6ms | 477.7ms |

Покадровый рендринг сцены

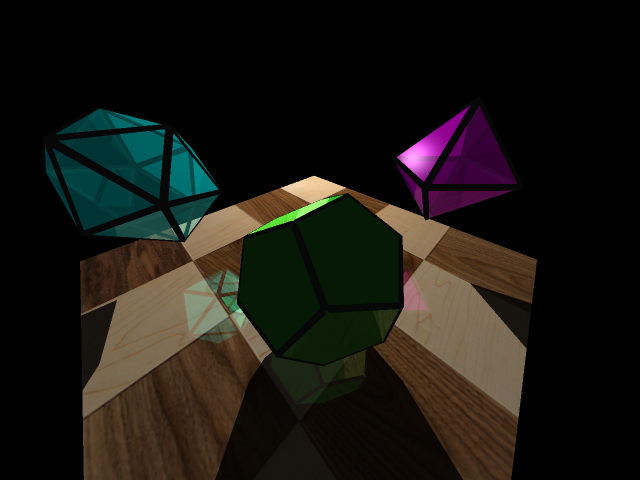


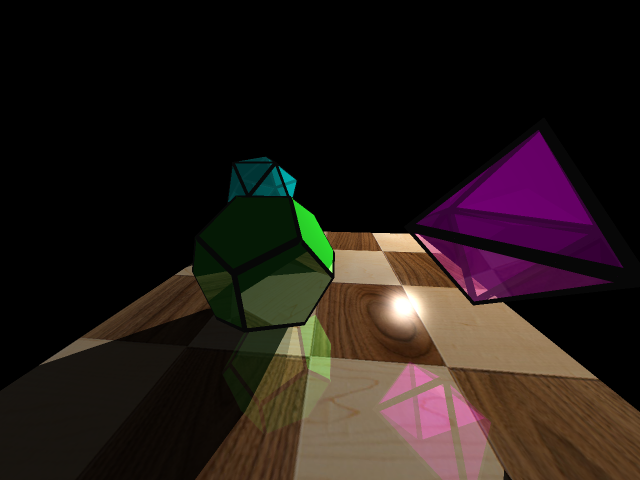
**Результаты**

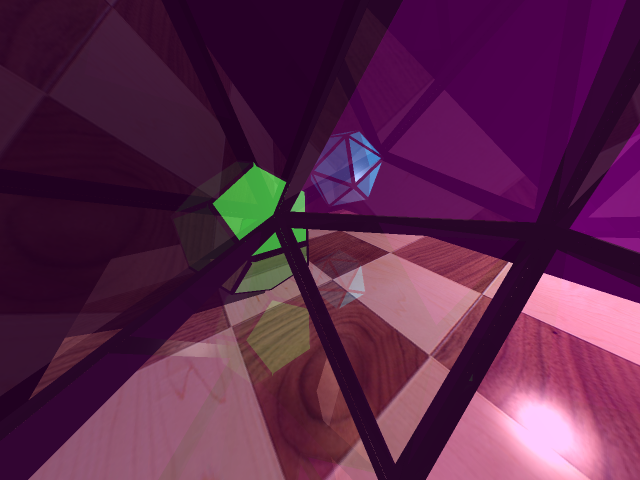
Отрисовка сцены

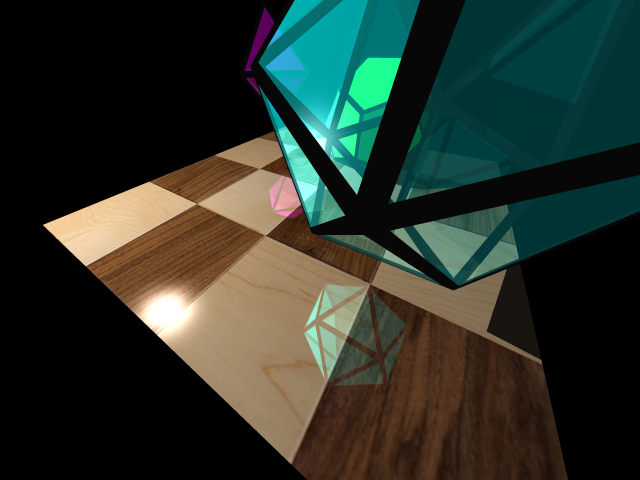


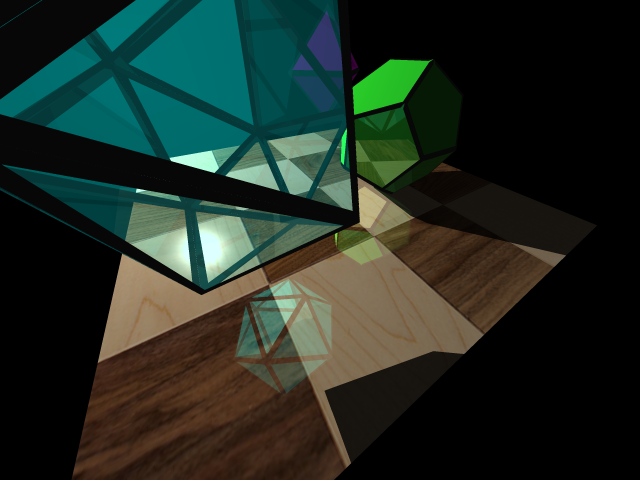
Скриншоты

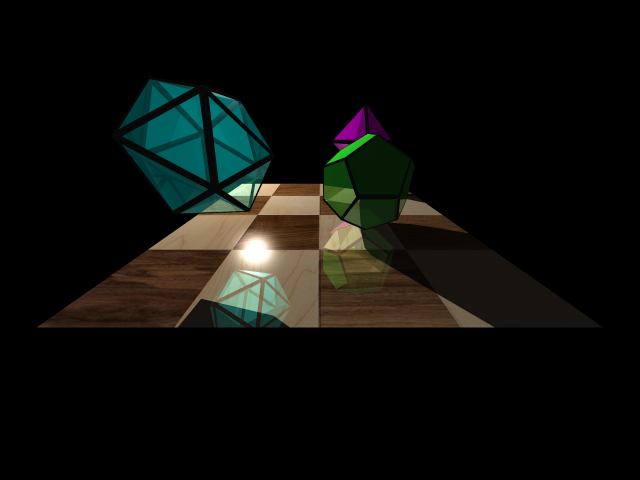


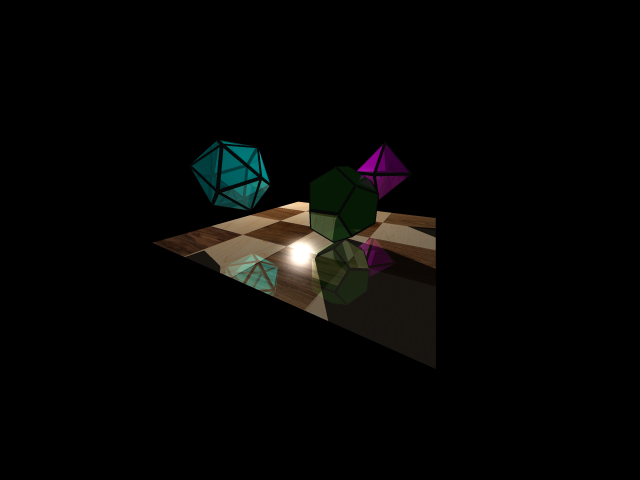


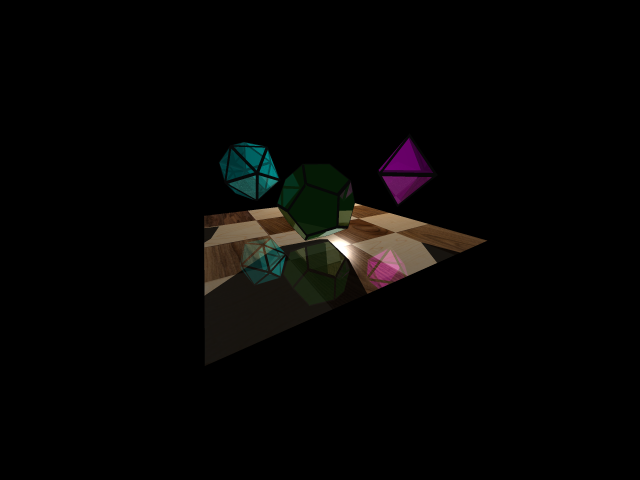


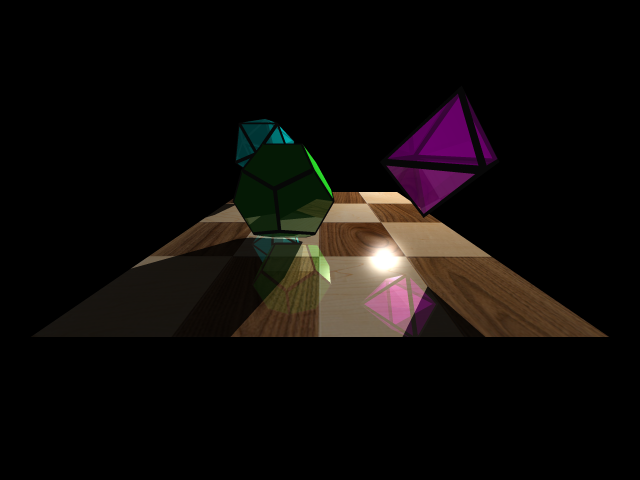


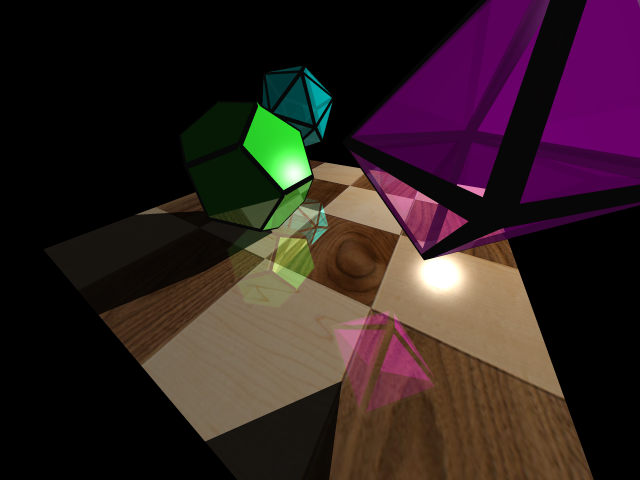












**Выводы**

Ray tracing применяется везде, где нужно получить фотореалистичное изображение – в играх, кино и т.д. Его ключевая особенность - трудоемкость вычислений и возможность легкого распараллеливания, что делает эту задачу очень подходящей для GPU. При реализации основной упор здесь идет именно на физические законы и правильную их интерпретацию, основная сложность заключается в этом. Для меня оказались сложными этапы генерации полигонов, так как фигуры неэлементарные, и легко запутаться с соединением и направлением нормали, а также на этапе реализации преломления, где необходимо рассчитывать сразу несколько пересечений луча с полигонами. Программа может быть улучшена – во-первых, с помощью правильно развернутой рекурсии, необходимо учитывать, что при запуске рекурсии многие лучи обрабатывать нет смысла, во-вторых, можно более грамотно наложить текстуры, при этом используя текстурную память, а также на грани можно добавить источники света. Но если даже при такой реализации GPU дает солидный выигрыш в небольшой задаче, то в реальных задачах без GPU просто нельзя обойтись.

Список литературы

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>