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: М. А. Инютин
Группа: М8О-407Б-19
Преподаватель: 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.

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

Общая постановка задачи

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 7

На сцене должны располагаться три тела: гексаэдр, октаэдр, додекаэдр.

Программное и аппаратное обеспечение

Характеристики графического процессора:

- Compute capability: 8.6
- Наименование: NVIDIA GeForce RTX 3060
- Графическая память: 12627148800
- Разделяемая память на блок: 49152
- Количество регистров на блок: 65536
- Максимальное количество потоков на блок: (1024, 1024, 64)
- Максимальное количество блоков: (2147483647, 65535, 65535)
- Константная память: 65536
- Количество мультипроцессоров: 28

Характеристики системы:

- Процессор: «AMD Ryzen 7 5800X (16) @ 3.800GHz»
- Память: комплект памяти «VENGEANCE LPX 32 Гб (2 x 16 Гб) DDR4 DRAM 3600 МГц C18»
- SSD: «Samsung SSD 980 1TB (2B4QFXO7) 1,0 TB»
- HDD: «WDC WD20EZBX-00AYRA0 (01.01A01) 2,0 TB»

Программное обеспечение:

- ОС: «Ubuntu 22.04.1 LTS x86_64»
- Текстовый редактор: «Atom 1.58.0»
- Компилятор: «nvcc: Cuda compilation tools, release 10.1, V10.1.243»

Метод решения
Освещение

Для построения фотореалистичного изображения буду использовать простую модель освещения:

$$I = I_a + I_d + I_s$$

где I_a — фоновая составляющая, I_d — рассеянная составляющая, I_a — зеркальная составляющая.

Фоновая составляющая

Даже при отсутсвии света в комнате мы может различать объекты, потому что лучи отражаются от всех поверхностей в комнате. Расчёт всех отражений слишком сложный, поэтому для простоты каждый объект получает часть фонового освещения:

$$I_a = k_a \cdot i_a$$

где k_a — свойство материала воспринимать фоновое освещени, i_a — мощность фонового освещения.

Рассеянная составляющая

Рассеянное отражение света происходит, когда свет как бы проникает под поверхность объекта, поглощается, а затем вновь испускается. При этом положение наблюдателя не имеет значения, так как диффузно отраженный свет рассеивается равномерно по всем направлениям.

Интенсивность света обратно пропорциональна квадрату расстояния от источника, следовательно, объект, лежащий дальше от него, должен быть темнее. Для простоты буду применять модель $d + K$, так как в модели обратных квадратов для $|d| \leqslant 1$ будет осветление объекта, что нужно отдельно обрабатывать. Константа K в этом выражении подбиратеся из соображений эстетики.

Выражение для рассеянного освещения имеет следующий вид:

$$I_d = \frac{k_d \cdot i_l}{d + K} \cdot \cos(\vec{L}, \vec{N})$$

где k_d — свойство материала воспринимать рассеянное освещение, i_l — интенсивность точеченого источника, $\vec{L}$ — направление из точки на источник света, $\vec{N}$ — вектор нормали в точке, K — произвольная постоянная, d — расстояние от источника света до точки.

Зеркальная составляющая

Интенсивность зеркально отраженного света зависит от угла падения, длины волны падающего света и свойств вещества отражающей поверхности. Зеркальное отражение света является направленным. Так как физические свойства зеркального отражения очень сложны, используется коээфициент глянцевости материала:

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

где k_s — свойство материала воспринимать зекральное освещение, $\vec{R}$ — вектор отражённого от поверхности луча, $\vec{S}$ — вектор наблюдения, p — глянцевость материала.

Трассировка лучей

При обратной трассировке лучи испускаются из камеры. Каждый пиксель на экране — это отдельный луч, который попадает на объекты сцены, отражается и преломляется. С помощью модели освещения вычисляется интенсивность цвета в точке, результат записывается в пиксель, соответствующий лучу.

Рекурсивная трассировка лучей

Так как каждый луч может породить два луча (отражённый и преломлённый), вычисление интенсивности происходит рекурсивно. Честная рекурсивная реализация достаточно медленная, поэтому на каждом шаге рекурсии нужно хранить два массива лучей — для текущего шага и для следующего. Для каждого луча вычисляется интенсивность, если луч отражается или преломлеятся, то эти лучи записываются в новый массив.

Загрузка сцены

Все полигоны на сцене хранятся в едином массиве. Многогранники загружаются из `obj` файлов, для них генерируются рёбра и источника света на рёбрах.

Наложение текстур

Для произвольного наложения текстуры на пол для каждого полигона пола я ввожу базис из двух векторов, достраивая треугольник до параллелограмма так, что любую точку в нём можно разложить по формуле

$$\vec{p} = \alpha \cdot \vec{v_1} + \beta \cdot \vec{v_2}$$

где $\vec{p}$ — точка параллелограмма, $\vec{v_1}$ и $\vec{v_2}$ — базисные векторы, $0 \leq \alpha, \beta \leq 1$ — коэффициенты разложения.

Зная коэффициент α и β, достаточно умножить их на высоту и ширину текстуры, чтобы получить цвет соответствующего пикселя текстуры.

Сглаживание

Для получения более красивой картинки используется алгоритм `SSAA`. Для каждого пикселя испускается несколько лучей, полученные интенсивности усредняются.

Описание программы

phong_shading вычисляет интенсивность освещения в точке. Функция возвращает трёхмерный вектор цветов, принимает указатели, поэтому работает как на центральном, так и на графическом процессоре.

Трассировка лучей реализована в функциях trace и trace_gpu, реализующих алгоритм, описанный выше. Каждый луч пересекается с каждым полигоном на сцене. Для параллельной записи новых лучей используется функция atomicAdd.

Текстурные объекты класса texture_t хранят два указателя — в оперативной памяти и в графической памяти. Поэтому доступ к пикселям текстур одинаковый как на центральном, так и графическом процессоре.

Сглаживание реализовано в функциях ssaa_cpu и ssaa_gpu.

Разделение по файлам

Программа состоит из следующих файлов:

- io.cuh — функция для чтения и записи изображения в бинарном формате;
- polygon.cuh — описание класса луча, треугольника и полигона;
- ray_cpu.cuh — функции для трассировки лучей на центральном процессоре;
- ray_gpu.cuh — ядра для трассировки лучей на графическом процессоре;
- scene.cuh — функция для чтения и обработки obj файлов, загрузка сцены;
- ssaa.cuh — функция и ядро, реализующие алгоритм SSAA;
- textures.cuh — описание класса текстуры;
- utils.cuh — функция для печати информации о графическом процессоре, класс таймера и макрос для проверки успешного выполнения системных вызовов;
- variables_cpu.cuh — переменные, необходимые для ввода данных и исполнения программы;
- variables_gpu.cuh — указатели в графической памяти, необходимые для исполнения программы на графическом процессоре;
- vector3d.cuh — описание шаблонного класса вектора с перегруженными операторами;
- main.cu — главный исполняемый файл.

Исследовательская часть

Замеры времени работы CPU и ядер с различными конфигурациями проводятся на изображении с разрешением 640 на 480 пикселей без сглаживания. Здесь k — глубина рекурсии.

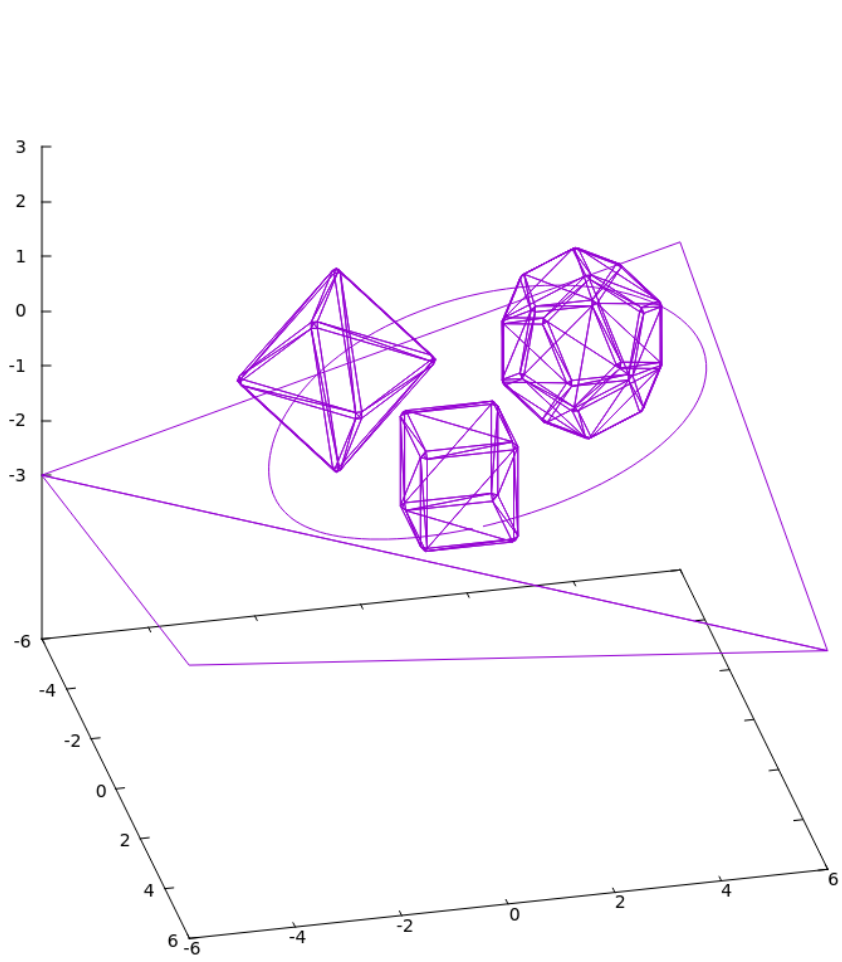
Общее количество лучей	307200	560757	961274	1763305
Конфигурация	$k = 1$	$k = 2$	$k = 4$	$k = 8$
CPU	2788.919	4580.225	8266.978	15293.571
(1, 32)	11456.789	22303.668	45185.785	94536.047
(32, 32)	415.182	797.055	1644.077	3496.258
(256, 32)	213.678	397.268	818.838	1745.377
(256, 256)	210.404	390.930	790.606	1693.551

Входные данные

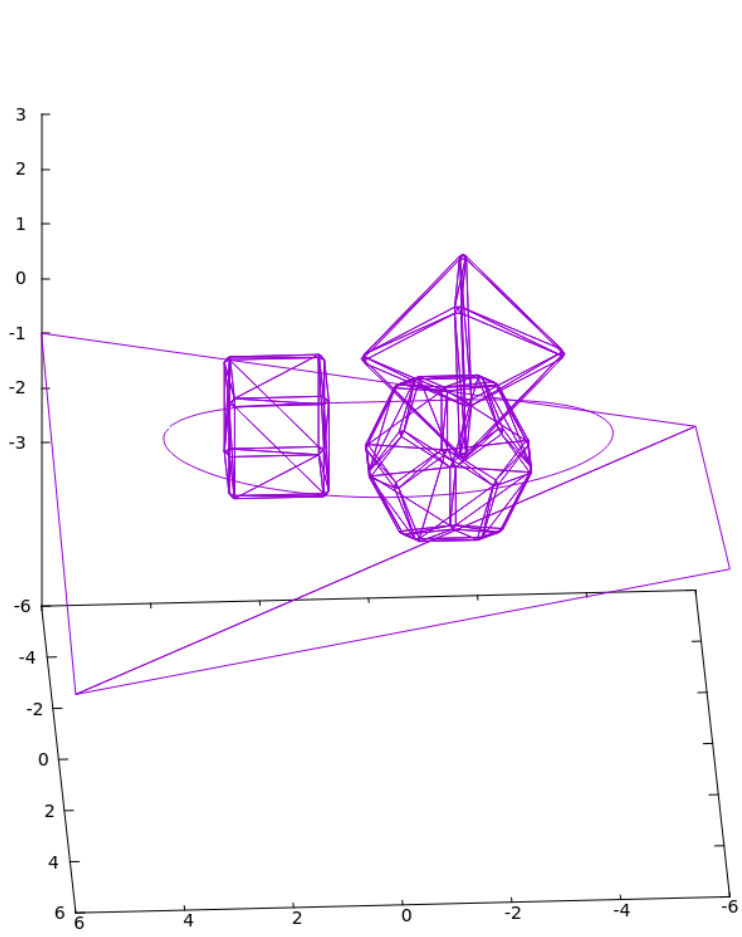
Входные данные, на которых получается наиболее красочный результат:

```
225
./res/%d.data
1920 1080 120

4.0 0.0 0.0   1.0 1.0   1.0 1.0 1.0   0.0 0.0
2.0 0.0 0.0   0.5 0.1   1.0 1.0 1.0   0.0 0.0

2.0  0.0  0.0    1.0  0.0  1.0   2.0    0.65   0.25   10
-1.5 -1.5  0.5    0.0  1.0  0.0   1.75   0.25   0.65   5
-1.0  3.0  0.75   0.75 0.75 0.0   1.5    0.45   0.45   2

-6.0 -6.0 -3.0   -6.0  6.0  0.0   6.0 6.0 -2.0   6.0 -6.0 -1.0
./textures/checkerboard.data 0.75 0.75 0.75 0.5

4
-5  5 4     1.0 0.5 0.0
-5 -5 2     0.0 1.0 0.0
5 -5 1     0.0 0.0 1.0
5  5 3     1.0 1.0 1.0

10 2
```

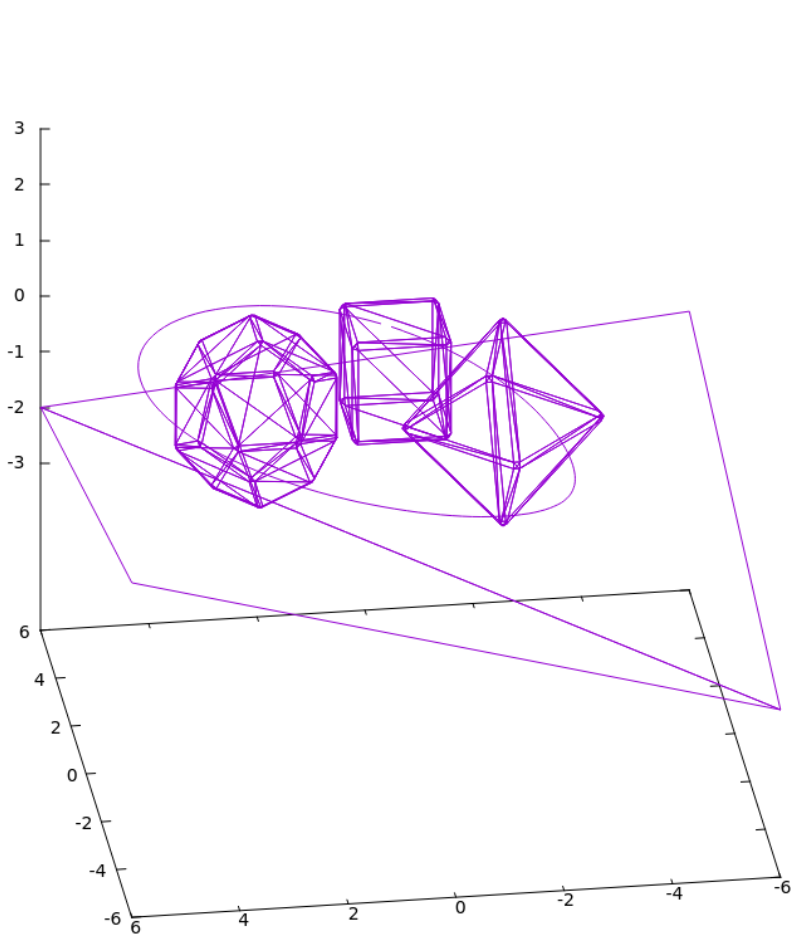
Трёхмерные графики



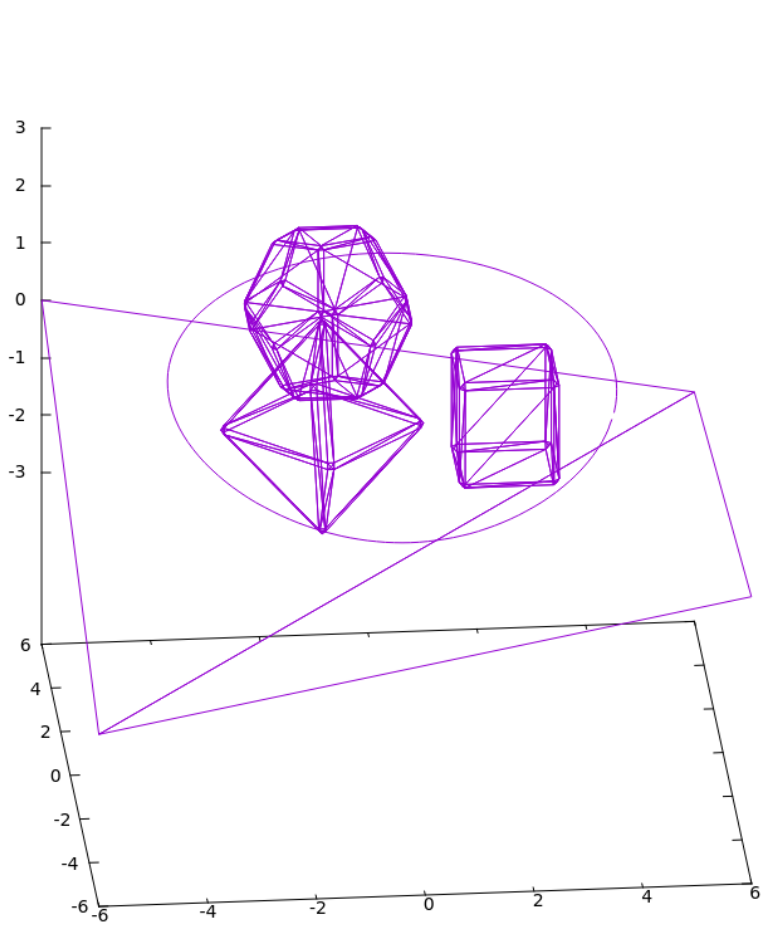
Вид спереди



Вид справа

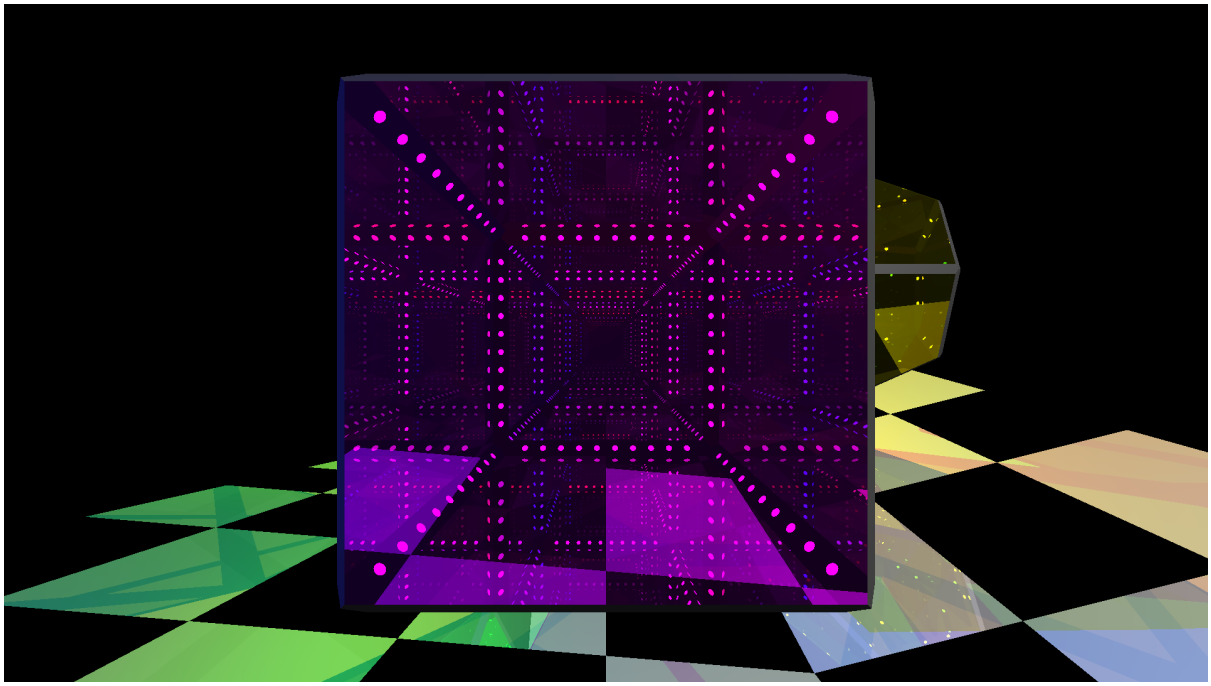


Вид сзади

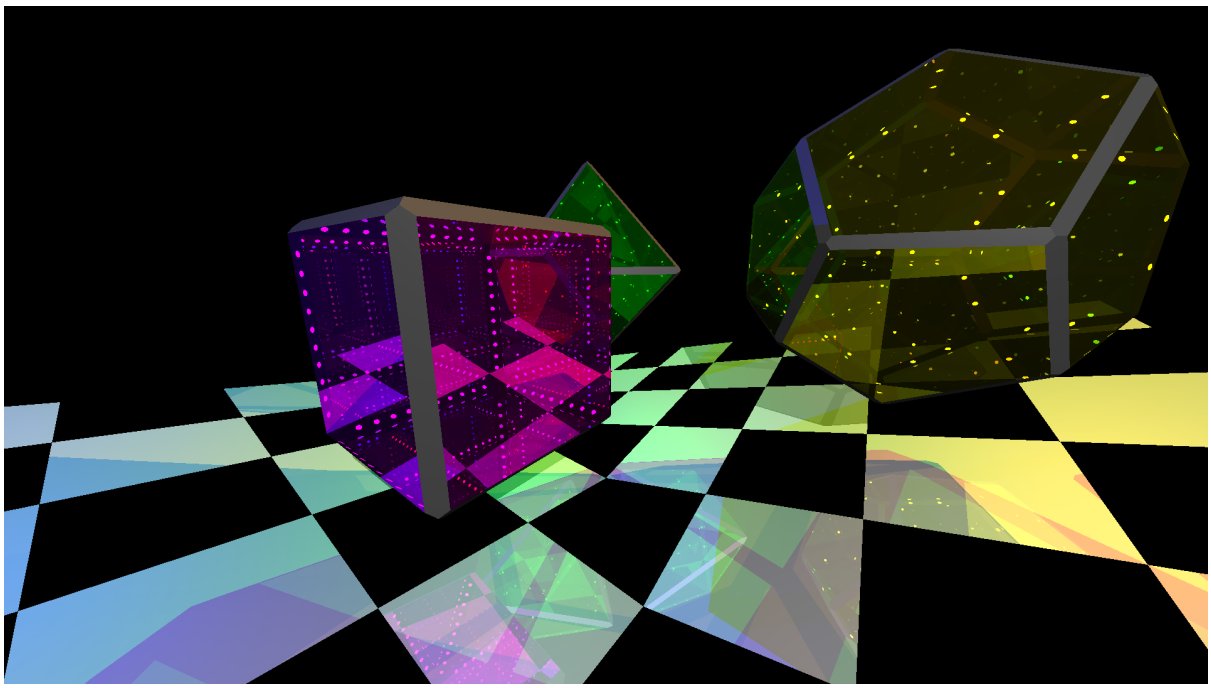


Вид слева

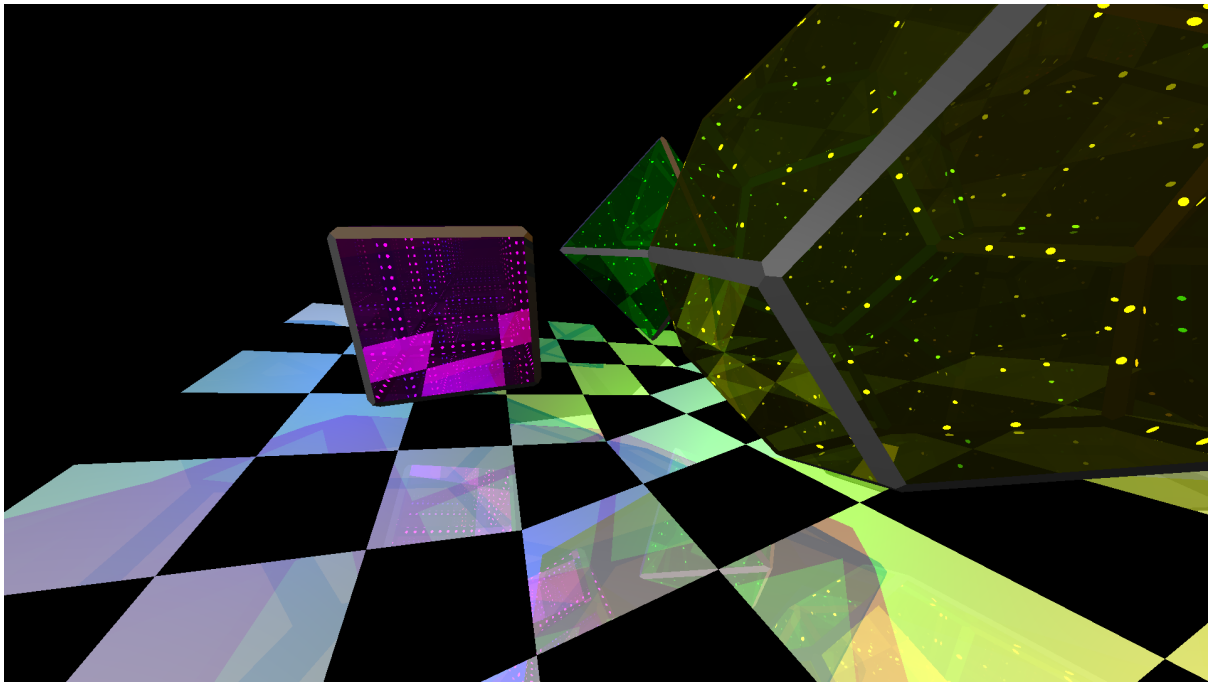
Результаты



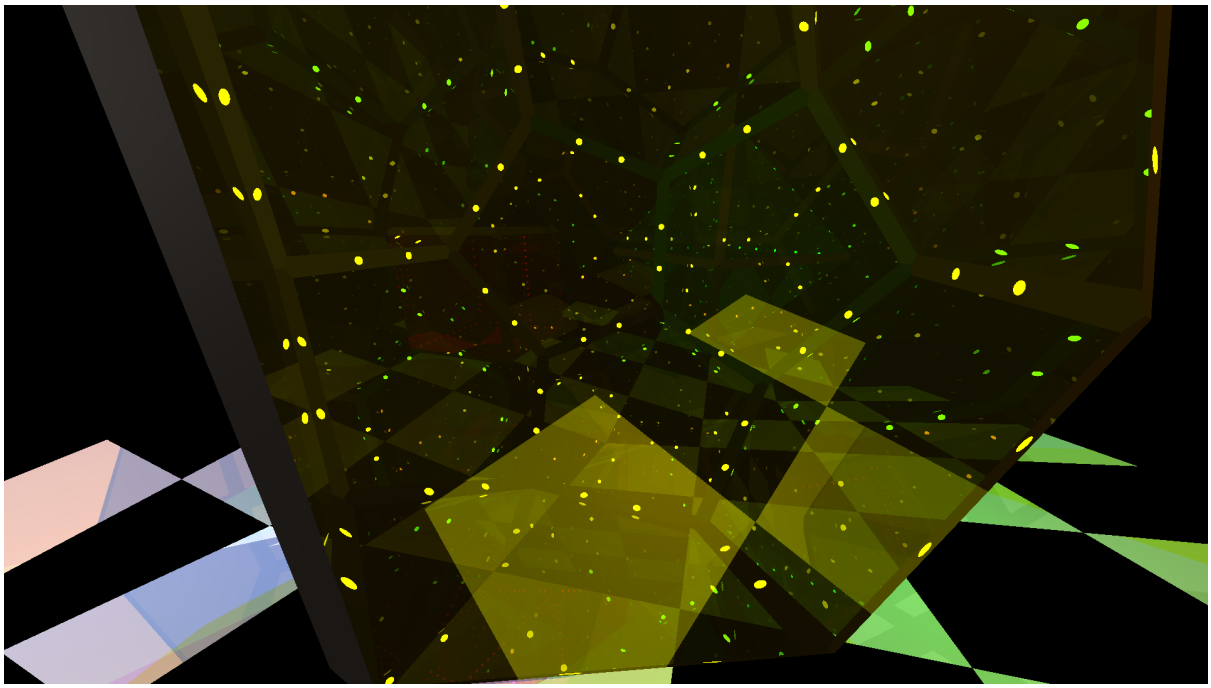
Эффект бесконечности в кубе



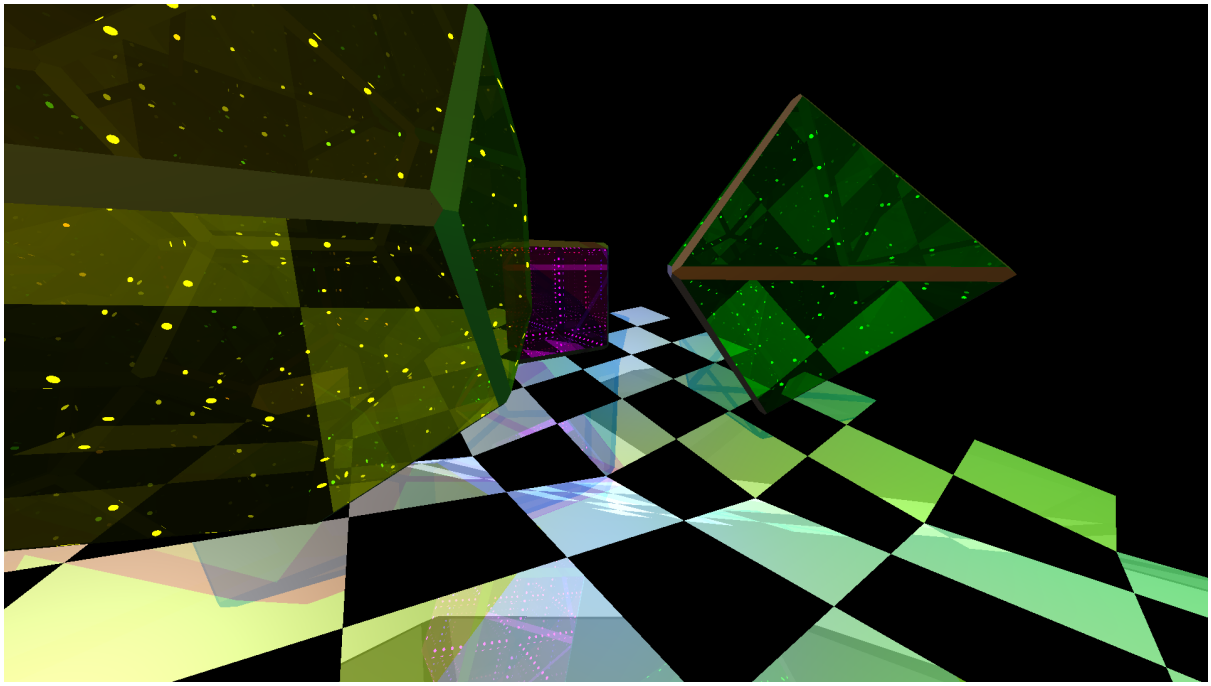
Два отражения октаэдра от пола



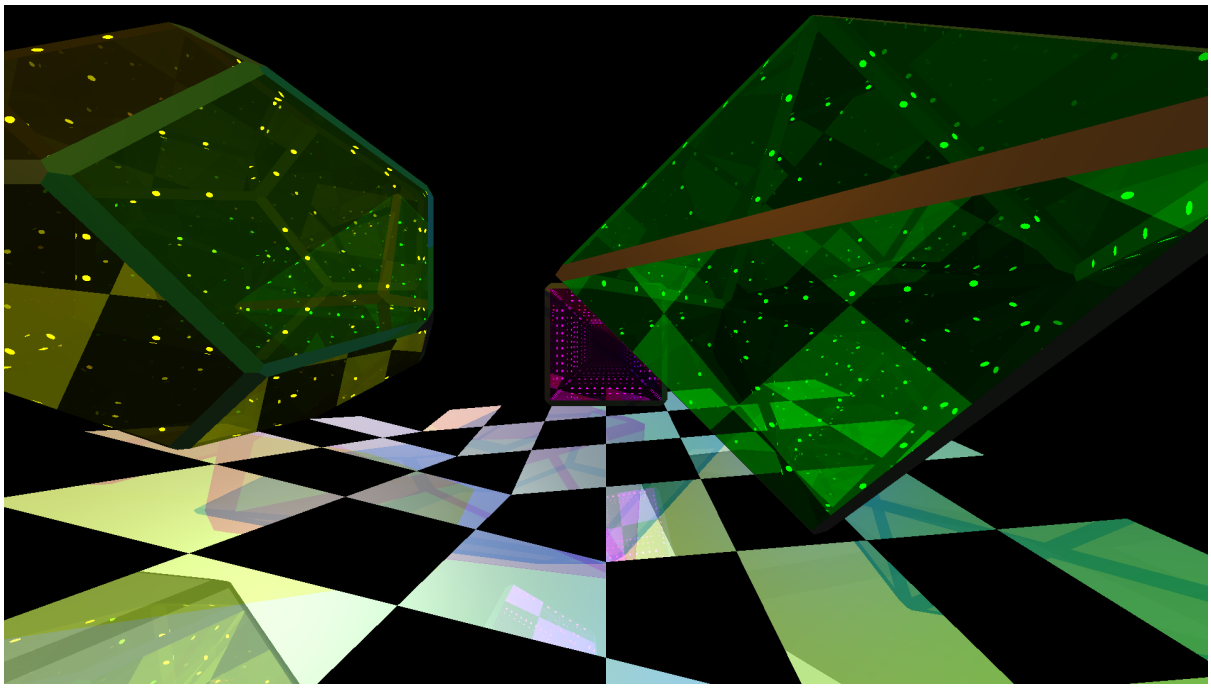
Октаэдр частично виден сквозь додекаэдр



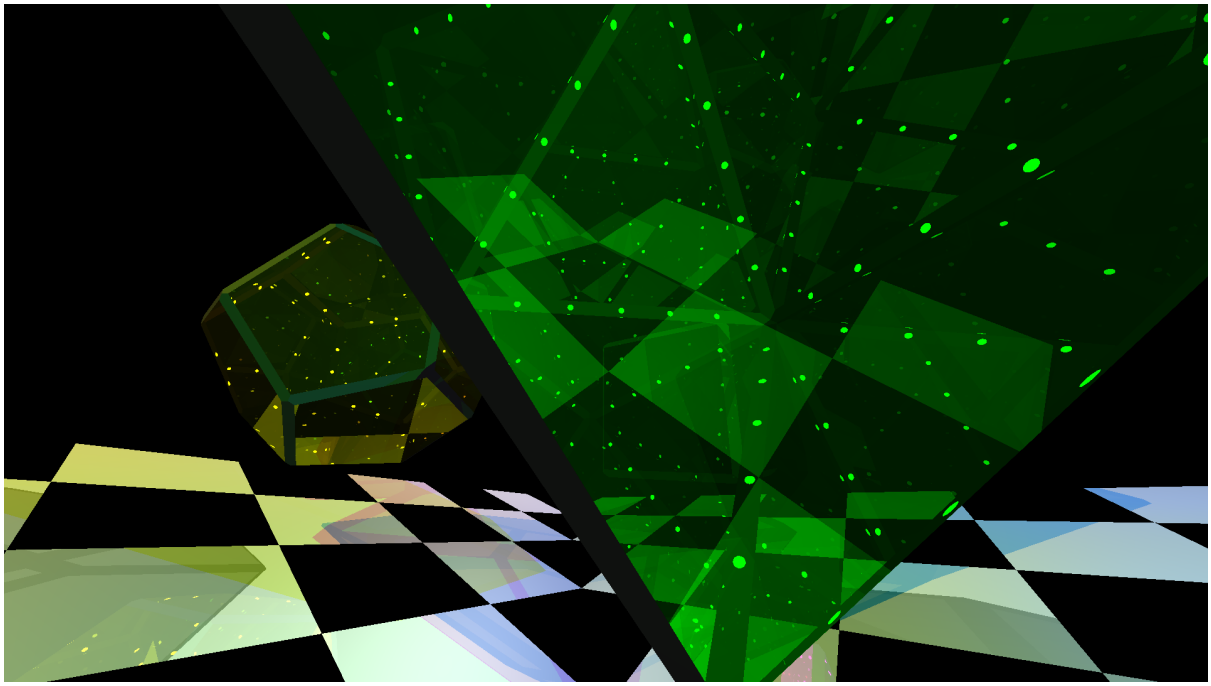
Эффект бесконечности в кубе в додекаэдре



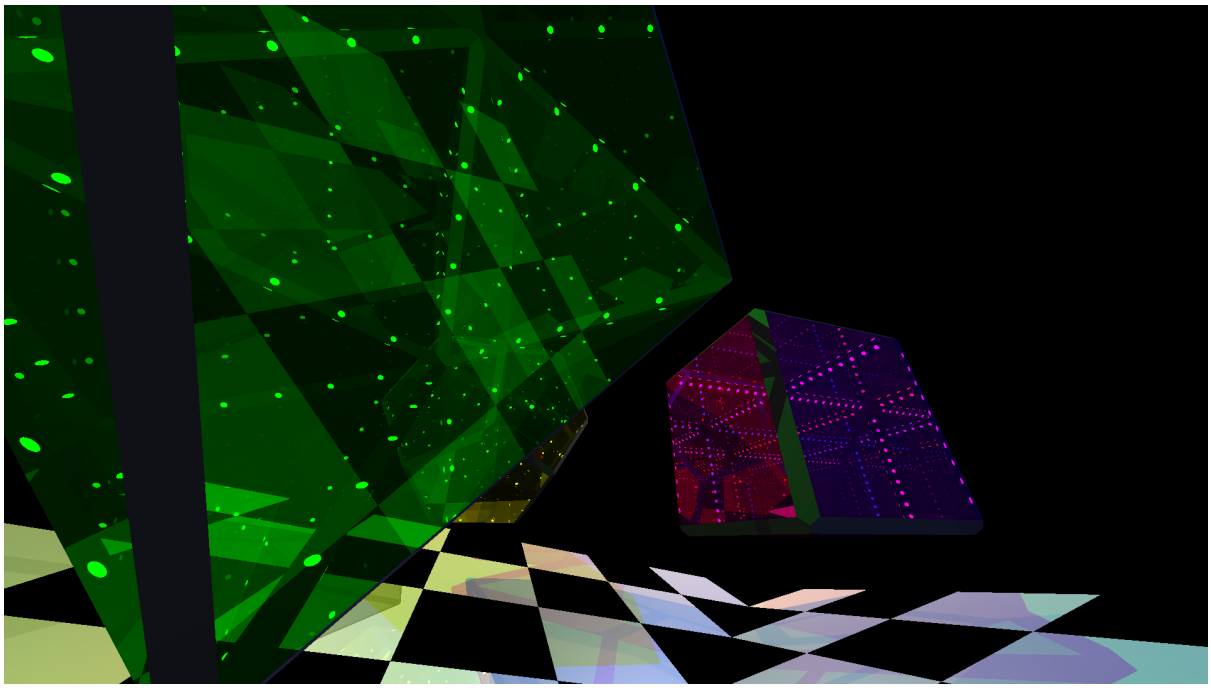
Отражение куба от пола



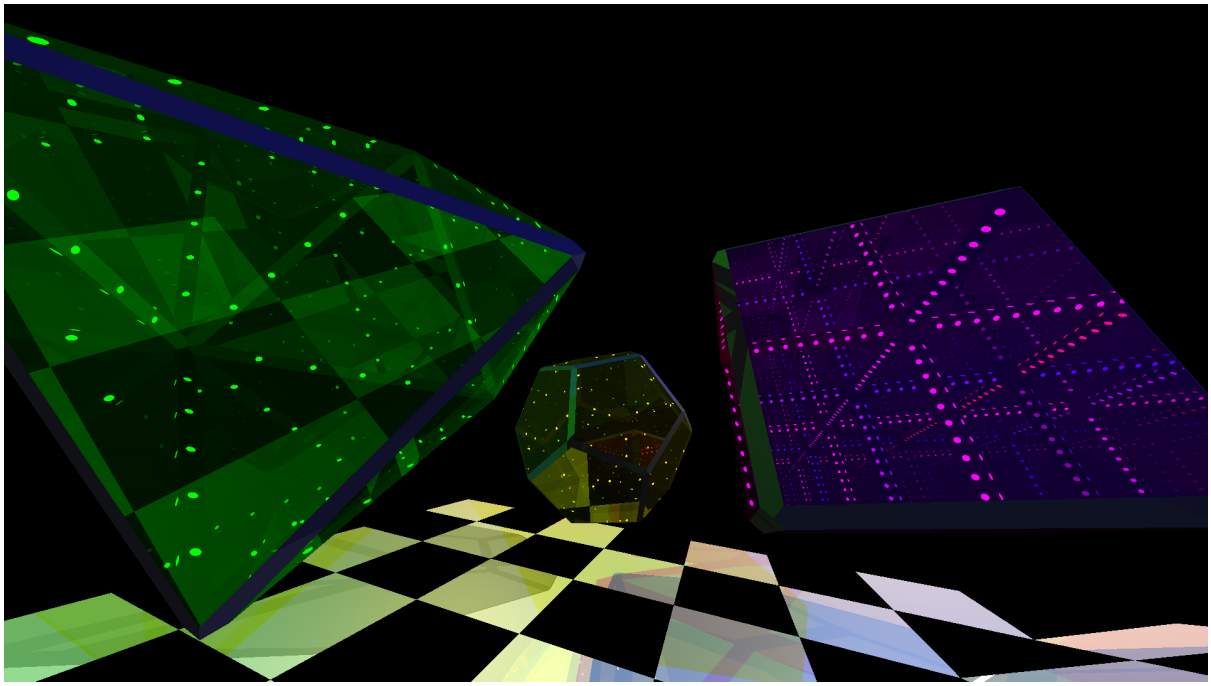
Октаэдр отражается в додекаэдре



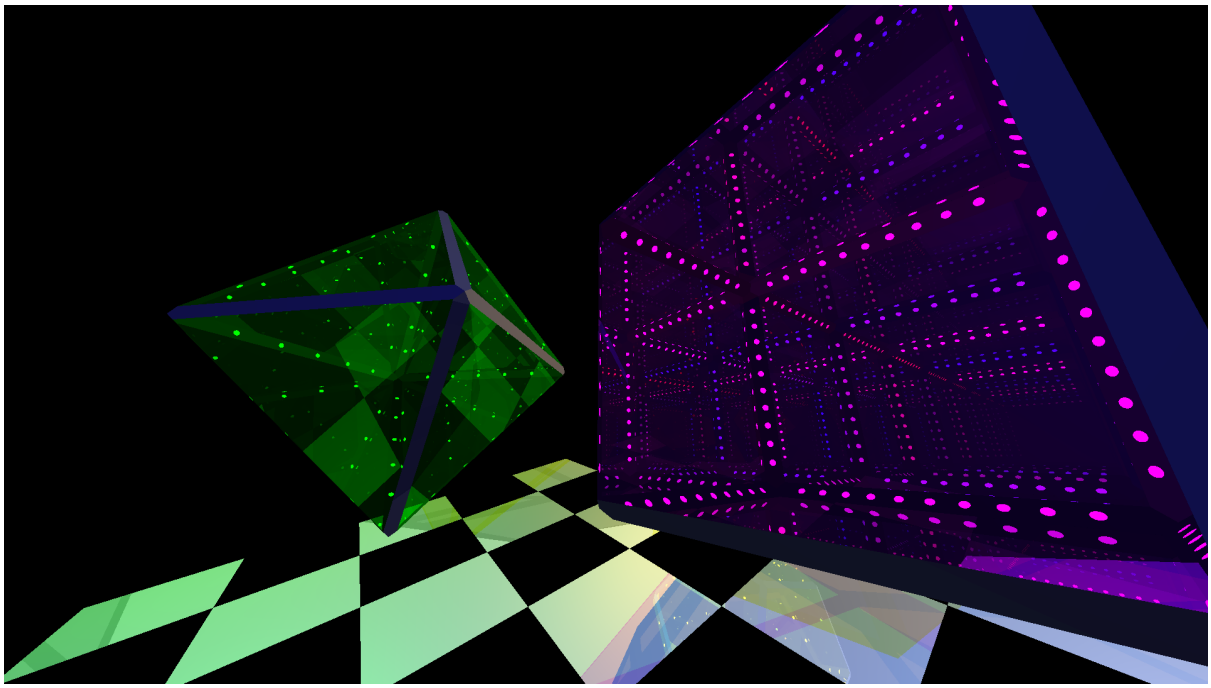
Эффект бесконечности в октаэдре



Додекаэдр частично виден сквозь октаэдр



Все три фигуры



Эффект бесконечности в кубе с другого ракурса

Выводы

Трассировка лучей используется в играх для обеспечения более выского уровня реализма по сравнению с традиционными способа рендеринга. Эта технология подразумевает моделирование распространения луча света, его отражение и преломление от всех поверхностей на сцене, за счёт чего и получается более реалистичесное изображение.

Полагая, что около половины лучей не будет пересекаться ни с одним полигоном и около половины лучей удвоится, сложность обратной трассировки лучей $O(k \cdot n \cdot m \cdot l)$ — для всех k шагов рекурсии вычисляется пересечение n лучей с m полигонами на сцене, затем вычисляются пересечения порядка n теневых лучей со всеми m полигонами для всех l источников света. Если же на каждом шаге рекурсии количество лучей удваивается, то сложность будет $O(2^k \cdot n \cdot m \cdot l)$. Разбиение пространства позволяет не искать пересечение с каждым полигоном на сцене, что позволяет существенно уменьшить сложность алгоритма.

Количество лучей зависит от входных параметров $n = h \cdot w \cdot r$, где h и w — высота и ширина изображения, r — количество лучей на один пиксель.

Замеры времени работы показали, что время исполнения зависит линейно от глубины рекурсии, при этом графический процессор выигрывает у центрального за счёт параллельной обработки лучей.

Список литературы

[1] *Ray-tracing.ru: 3d движок, трассировка лучей в реальном времени*
URL: `http://www.ray-tracing.ru/`
(дата обращения: 03.01.2023)

[2] *Простые модели освещения — Компьютерная графика*
URL: `https://grafika.me/node/344`
(дата обращения: 05.01.2023)