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: А.С. Синюков
Группа: 8О-406Б
Преподаватель: А.Ю. Морозов

Москва, 2025

Условие

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.

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 gpu и cpu (т.е. дополнительно нужно реализовать алгоритм без использования CUDA).

Вариант 6: Тетраэдр, Додекаэдр, Икосаэдр

Программное и аппаратное обеспечение

Характеристики графического процессора

Compute capability	: 7.5
Name	: Tesla T4
Total Global Memory	: 15835660288
Shared memory per block	: 49152
Registers per block	: 65536
Warp size	: 32

Max threads per block : (1024, 1024, 64)
Max block : (2147483647, 65535, 65535)
Total constant memory : 65536
Multiprocessors count : 40

Характеристики процессора

Architecture: x86_64
CPU op-mode(s): 32-bit, 64-bit
Address sizes: 46 bits physical, 48 bits virtual
Byte Order: Little Endian
CPU(s): 2
On-line CPU(s) list: 0,1
Vendor ID: GenuineIntel
Model name: Intel(R) Xeon(R) CPU @ 2.00GHz
CPU family: 6
Model: 85
Thread(s) per core: 2
Core(s) per socket: 1
Socket(s): 1
Stepping: 3
BogoMIPS: 4000.28
Virtualization features:
Hypervisor vendor: KVM
Virtualization type: full
Caches (sum of all):
L1d: 32 KiB (1 instance)
L1i: 32 KiB (1 instance)
L2: 1 MiB (1 instance)
L3: 38.5 MiB (1 instance)
NUMA:
NUMA node(s): 1
NUMA node0 CPU(s): 0,1

Характеристики оперативной памяти

MemTotal: 13290460 KB

Характеристики жесткого диска

MemTotal: 113 GB

Лабораторная работа выполнялась в Google Colab, компилятор - nvcc.

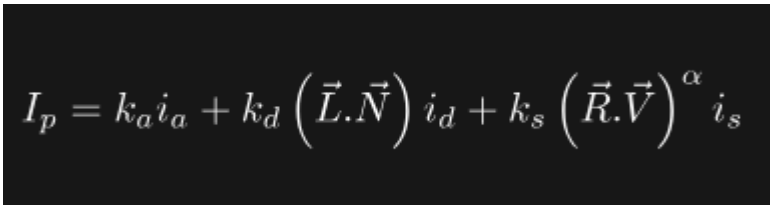
Метод решения

Трассировка лучей — это технология рендеринга компьютерной графики, которая создаёт изображение путём отслеживания траектории лучей через визуализируемое трёхмерное пространство. Лучи могут взаимодействовать с объектами в пространстве, отражаться от них и приобретать такие свойства, как цвет.

Обратная трассировка лучей — это метод, используемый для определения цвета каждого пикселя на экране. В отличии от прямой трассировки, лучши, можно сказать, “исходят” из камеры, а не от источников света. Это позволяет сильно сократить вычисления, так как большинство лучей в среднем уходят “в молоко”.

Рекурсивная трассировка лучей — это метод трассировки лучей, при котором луч может разделиться на 2 компоненты (отраженную и преломленную). Рекурсия позволяет моделировать сложные взаимодействия между объектами, такие как множественные отражения, прозрачность и другие физические свойства света.

Затенение по Фонгу — модель расчёта освещения трёхмерных объектов, в том числе полигональных моделей и примитивов, а также метод интерполяции освещения по всему объекту. Эта модель состоит из трех компонентов: диффузного, отраженного и окружающего освещения. Диффузное освещение моделирует рассеивание света на шероховатой поверхности, отраженное освещение отвечает за “блеск” объекта, а окружающее за фоновое освещение.
Формулу вычисления можно представить в следующем виде:



$$I_p = k_a i_a + k_d \left(\vec{L}.\vec{N}\right) i_d + k_s \left(\vec{R}.\vec{V}\right)^{\alpha} i_s$$

Где:

k_a - свойство окружающего освещения

i_a - мощность окружающего освещения

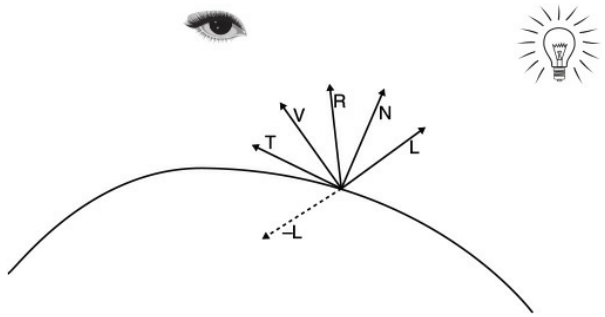
k_d - свойство диффузного освещения

i_d - мощность диффузного освещения

k_s - свойство отраженного освещения

i_s - мощность отраженного освещения

Направления векторов можно увидеть на изображении:



L - вектор направления к источнику освещения

N - нормаль к поверхности тела

R - “отражение” вектора света (т.е. вектор, получаемый при отражении вектора -L от источника света)

V - вектор направления к камере

T - касательная к поверхности в точке.

Описание программы

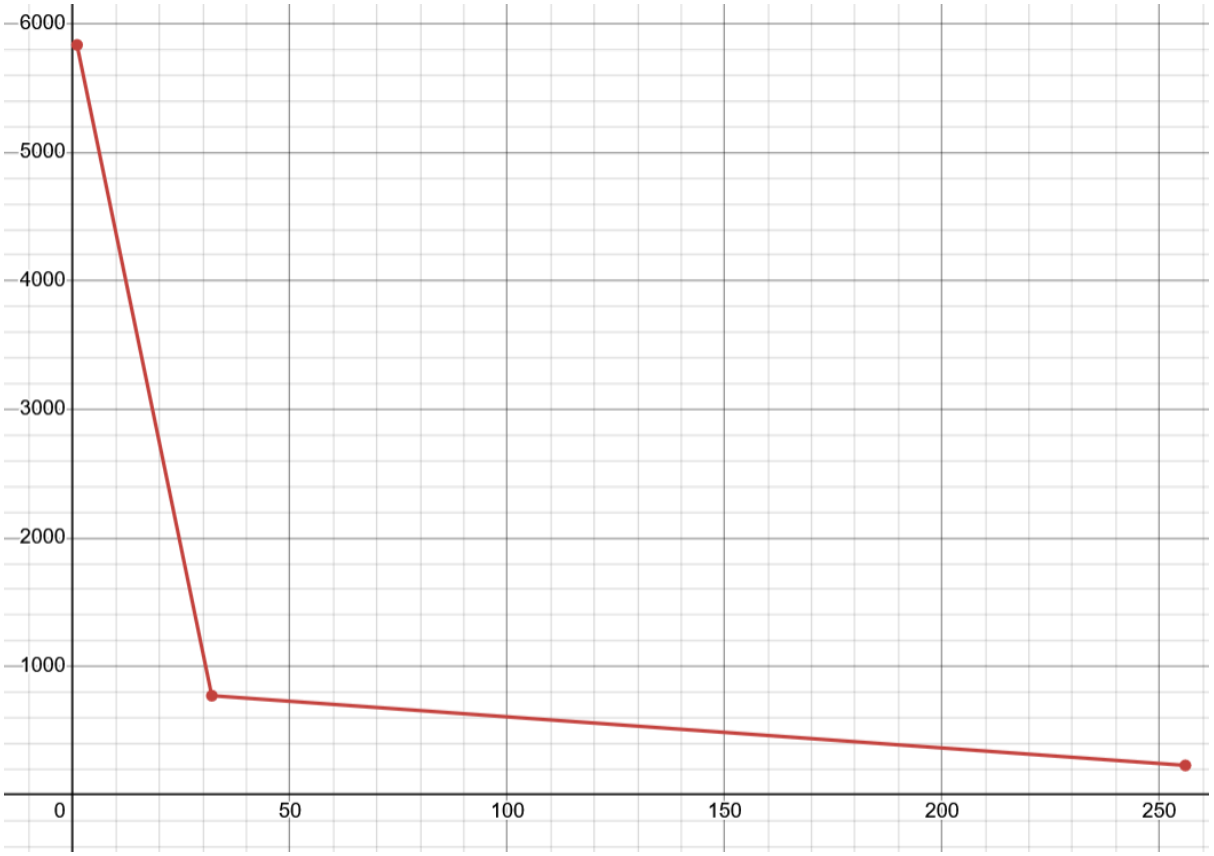
Все функции и ядра находятся в файле main.cu, дополнительно есть obj файлы с фигурами (фигуры в файлах описаны около сферы с радиусом 1 и центром в (0 0 0) для перемещения и масштабирования входными данными), а также текстура пола (в формате .data из предыдущих ЛР).

Структура vector3d реализует трехмерный вектор и основные функции, для работы с ними. Структура реализована на шаблоне, так что можно имплементировать как с double, так и с int, float, long и т.д. ttexture (название изменено из-за странного бага с конфликтом имен) создан для загрузки текстур и получения данных о пикселях на ней. ray и triangle реализованы для хранения информации о луче и треугольнике (а также нормали, края, дополнительные функции для сдвигов и т.п.). polygon содержит структуру полигона, а также методы для вычисления цвета в зависимости от “материала”, который реализован различными конструкторами. Также с полигонами работают функции пересечений intersect_ray_and_plane и intersect_ray_and_polygon. figure и light_source являются хранящими структурами данных о фигурах и источниках света, также переопределяют для них ввод со стандартного потока. read_obj считывает данные из файлов .obj и формирует фигуры из полигонов. функция scene “строит” пол и тела, а функция shading строит затенения по формулам. Для рендера используются функции render_gpu и render_cpu в зависимости от флагов.

Исследовательская часть и результаты

На всех тестах разрешение 480p. Время для одного кадра в миллисекундах.

Глубина рекурсии -	**2**	**4**
<<<32,32>>>	**772.062**	**1410.388**
<<<256,256>>>	**229.821**	**448.846**
CPU	**5836.147**	**10389.494**



Время работы на GPU значительно ниже, чем на CPU. При этом увеличение количества блоков и потоков дает все еще значимый результат.

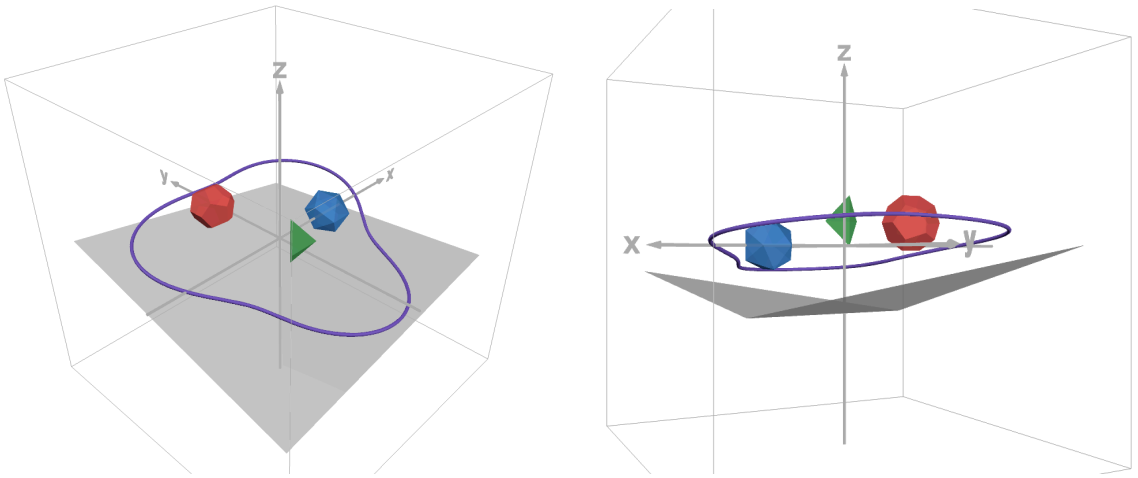
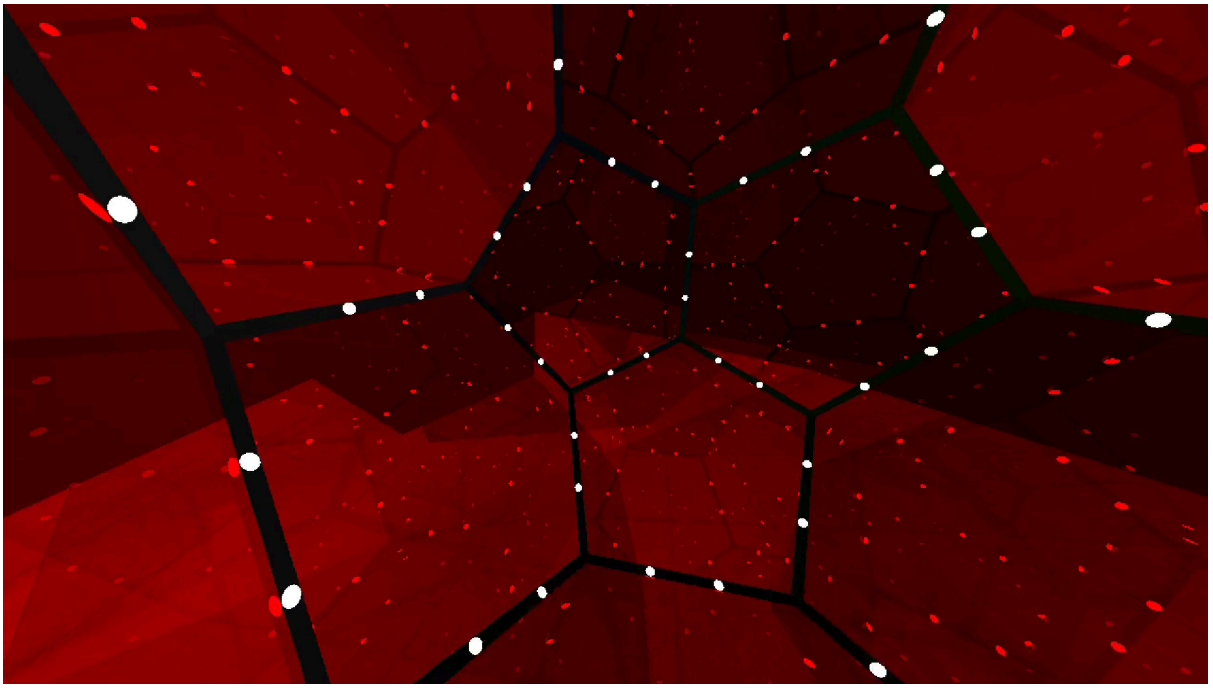
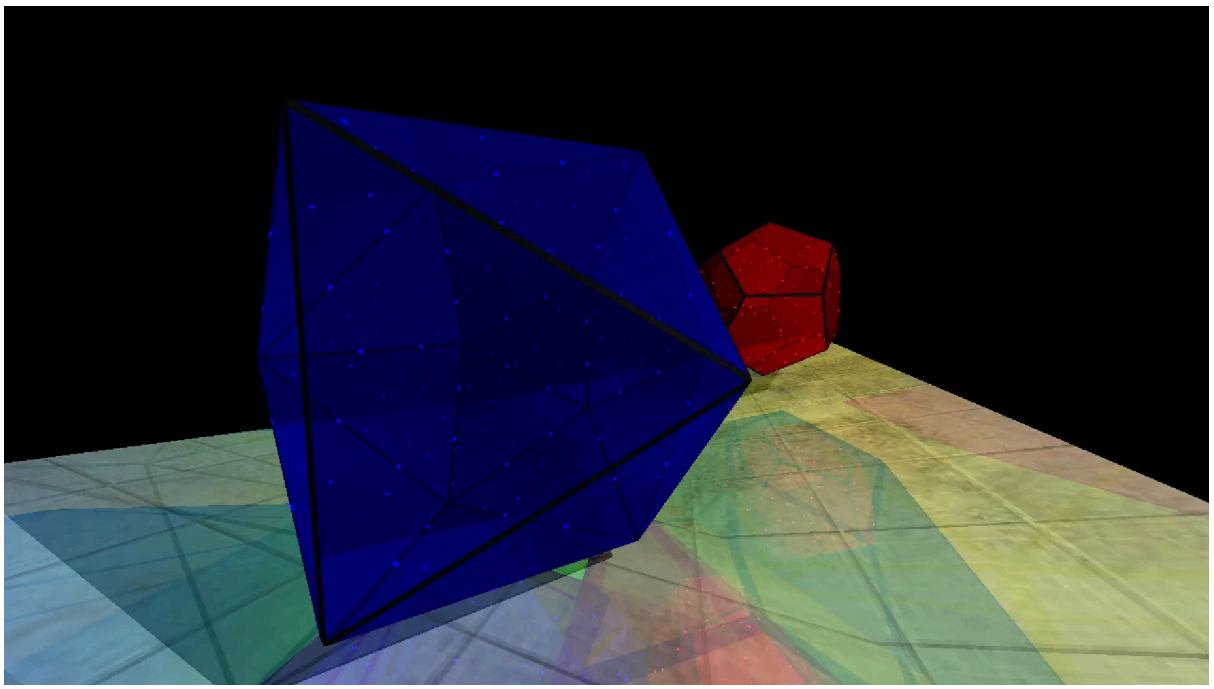
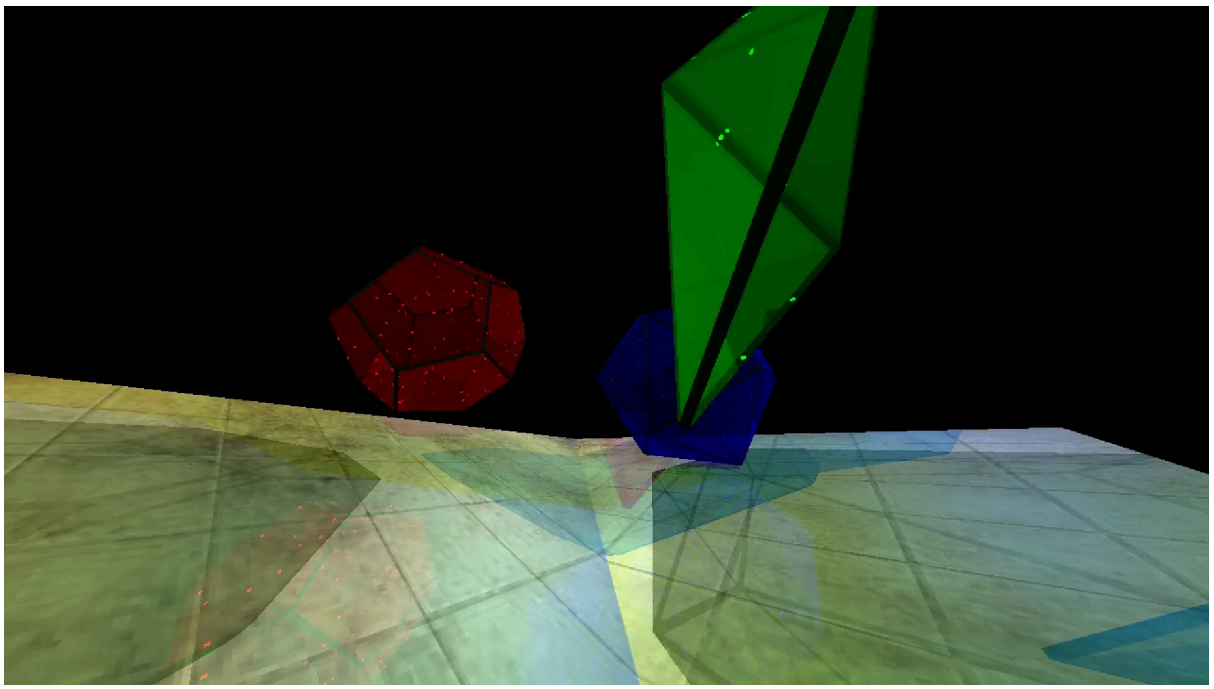


Схема сцены (построена в desmos)

Примеры работы:







Конфигурация:
50
./res/%d.data
640 480 130

5.0 2.0 0.0 1.0 1.0 3.0 1.0 1.0 0.0 0.0
2.0 0.0 0.0 0.5 0.1 3.0 1.0 1.0 0.0 0.0

 3.0 0.0 0.0   0.0 0.0 1.0 2.0   0.25 0.5 2
-1.0 -1.5 1.0   0.0 1.0 0.0 1.75   0.25 0.5 2
-1.0 3.0 0.75 1.0 0.0 0.0 1.5   0.25 0.5 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 ./floor1.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

4 1

Выводы

Такая реализация алгоритма, в принципе, может подойти для создания фотореалистичных сцен для фильмов, роликов и интернет-изображений. Но при этом она плохо подходит для компьютерных игр, ибо даже в разрешении 480p (шакалы) алгоритм может выдать в лучшем случае 1-2 кадра в секунду. Текущие игры либо используют более оптимальные реализации трассировки лучей от AMD и NVIDIA, либо даже используют некоторые хитрости (по типу дорисовки кадра с помощью DLSS). Основной сложность в разработке КП выступала сложность рендеринга, а также ограниченность ресурсов (отсутствие видеокарты NVIDIA побудило использовать google colab, а в колабе сложно работать).

Литература

1) Ray-tracing URL: http://ray-tracing.ru/ (дата обращения: 05.01.2025).
2) Рейтрейсинг: что это такое и зачем нужно URL: https://thecode.media/raytracing/?ysclid=m5lpb9hzh5131217917 (дата обращения: 06.01.2025).
3) Немного о затенении по Фонгу URL: https://habr.com/ru/articles/441862/ (дата обращения: 05.01.2025).
4) Особенности моделирования света: Аппроксимации Фонга URL: https://www.ixbt.com/video/light-model-phong.html?ysclid=m5lp8z4j6h321294188 (дата обращения: 06.01.2025).

5) Написал рейтрейсинг с нуля | Как работает 3D-графика URL: https://www.youtube.com/watch?v=jKjbeWHujV0 (дата обращения: 04.01.2025).