МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Компьютерные науки и прикладная математика»

Курсовая работа
по курсу «Параллельная обработка данных»

Обратная трассировка лучей (Ray Tracing) на GPU

Выполнил: Ермаков Я. В.
Группа: М8О-407Б-22
Преподаватели: А.Ю. Морозов

Москва, 2025

Условие

1. **Цель работы:** работы. Использование GPU для создание фотореалистической визуализации. Рендеринг полузеркальных и полупрозрачных правильных геометрических тел. Получение эффекта бесконечности. Создание анимации.
2. **Вариант 8:** Гексаэдр, Октаэдр, Икосаэдр.

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).

Программа должна принимать на вход следующие параметры:
1. Количество кадров.
2. Путь к выходным изображениям. В строке содержится спецификатор %d, на место которого должен подставляться номер кадра. Формат изображений

соответствует формату описанному в лабораторной работе 2.
3. Разрешение кадра и угол обзора в градусах по горизонтали.
4. параметры движения камеры: r_c^0, z_c^0, φ_c^0, A_c^r, A_c^z, ω_c^r, ω_c^z, ω_c^φ, p_c^r, p_c^z, r_n^0, z_n^0, φ_n^0, A_n^r, A_n^z, ω_n^r, ω_n^z, ω_n^φ, p_n^r, p_n^z;
5. Параметры тел: центр тела, цвет (нормированный), радиус (подразумевается радиус сферы в которую можно было бы вписать тело), коэффициент отражения, коэффициент прозрачности, количество точечных источников света на ребре.
6. Параметры пола: четыре точки, путь к текстуре, оттенок цвета и коэффициент отражения.
7. Количество (не более четырех) и параметры источников света: положение и цвет.
8. Максимальная глубина рекурсии и квадратный корень из количества лучей на один пиксель (для SSAA).

Программа должна поддерживать следующие ключи запуска:

--cpu Для расчетов используется только центральный процессор

--gpu Для расчетов задействуется видеокарта

--default В stdout выводится конфигурация входных данных (в формате описанном ранее) при которой получается наиболее красочный результат, после чего программа завершает свою работу.

Запуск программы без аргументов подразумевает запуск с ключом --gpu.

В процессе работы программа должна выводить в stdout статистику в формате: {номер кадра}\t{время на обработку кадра в миллисекундах}\t{общее количество лучей}\n

Программное и аппаратное обеспечение

Аппаратная и программная конфигурация использовалась на облачных серверах Google Collab. Использованная для выполнения программа включала графический ускоритель NVIDIA Tesla T4 с объемом видеопамяти 15360 МБ. Ошибки памяти ECC отсутствовали. Драйвер графического ускорителя имел версию 550.54.15, а установленная версия CUDA составляла 12.4.

Характеристики CUDA-устройства (Tesla T4):

1. Computecapability: 7.5 (sm_75)
2. Количество SM: 40
3. Размер варпа: 32
4. Макс. потоков на блок: 1024
5. Макс. потоков на SM: 1024
6. Shared memory на блок: 49 152 байт
7. Shared memory на SM: 65 536 байт
8. Регистров на блок: 65 536
9. Макс. размеры сетки (grid): 2 147 483 647 × 65 535 × 65 535
10. Макс. размеры блока (block dims): 1024 × 1024 × 64

11. Компиляция и модель запуска:
12. Компилятор: nvcc 12.4
13. Ключи сборки:-O3-arch=sm_75

Распределение вычислений: сетка потоков, данные параметров классов размещаются в константной памяти устройства.

Метод решения

Сцена собирается из треугольных полигонов: 2 треугольника пола + 12 (гексаэдр) + 8 (октаэдр) + 20 (икосаэдр) = **42 треугольника**. Для каждого пикселя строится луч из камеры, ищется ближайшее пересечение с треугольником, затем считается освещение по **Lambert (диффузия) поканально** с учётом теней. Сглаживание выполняется **SSAA**: затем усреднение в итоговый кадр. GPU-версия параллелит обработку пикселей в CUDA-ядрах; CPU-версия делает то же последовательно.

Описание программы

Программа - один файл main.cu: содержит структуры для векторов/треугольников/источников света, функции пересечения, шейдинг, CPU- и GPU-ядра рендера и SSAA-downsample. Параметры читаются из stdin в формате задания, ключ --default печатает конфиг, --cpu/--gpu выбирают реализацию; каждый кадр сохраняется в .data и печатается статистика {кадр}\t{мс}\t{лучи}

Исследовательская часть

В рамках исследовательской части были проведены замеры производительности реализации алгоритма обратной трассировки лучей на **CPU** и **GPU (CUDA)**.
Измерения выполнялись для различных разрешений кадров, а также анализировалось время построения каждого кадра при фиксированном разрешении.

Пример входных данных:

```
120
res/%d.data
800 800 90
7 3 0 2 1 2 6 1 0 0
2 0 0 0.5 0.1 1 4 1 0 0
0 -2 0 1 0.78 0 1 0 0 0
0 0 0 0 1 0.7 1 0 0 0
0 2 0 0.47 0.7 1 1 0 0 0
-5 -5 -1 -5 5 -1 5 5 -1 5 -5 -1 floor.data 0 1 0 0
1
8 0 10 1 1 1
10 4
```

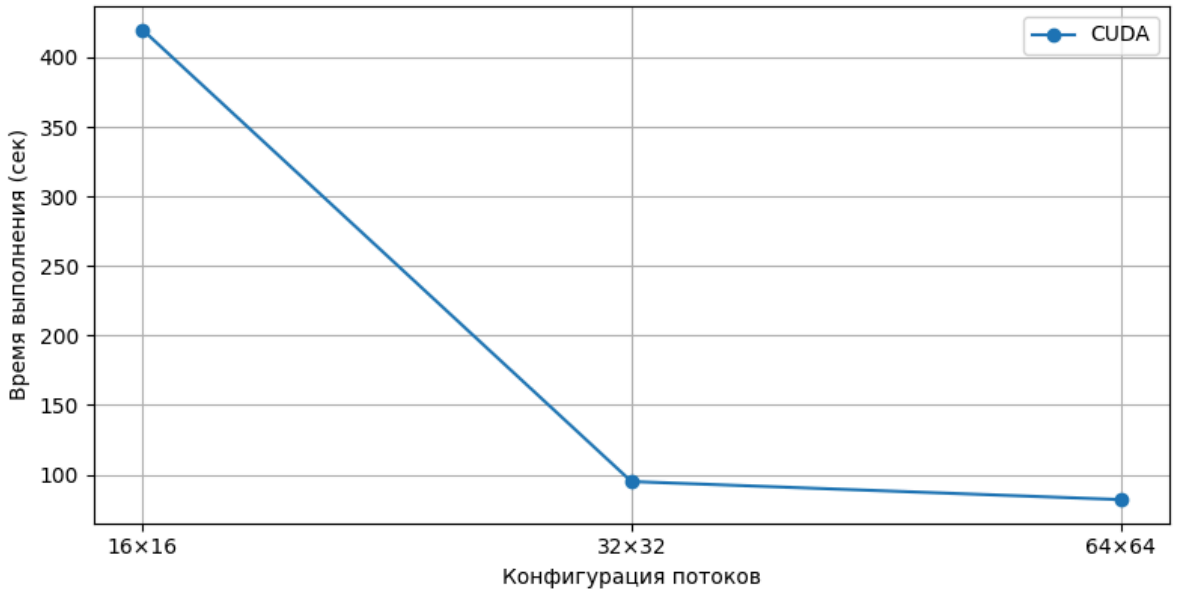
Результаты

1) **Зависимость времени выполнения программы от количества используемых потоков**

(рендеринг **50 кадров**, разрешение **1920×1080**, CUDA)

Таблица 1 — Время выполнения при различных конфигурациях потоков

Конфигурация потоков	Время выполнения (сек)
16×16	420
32×32	95
64×64	82



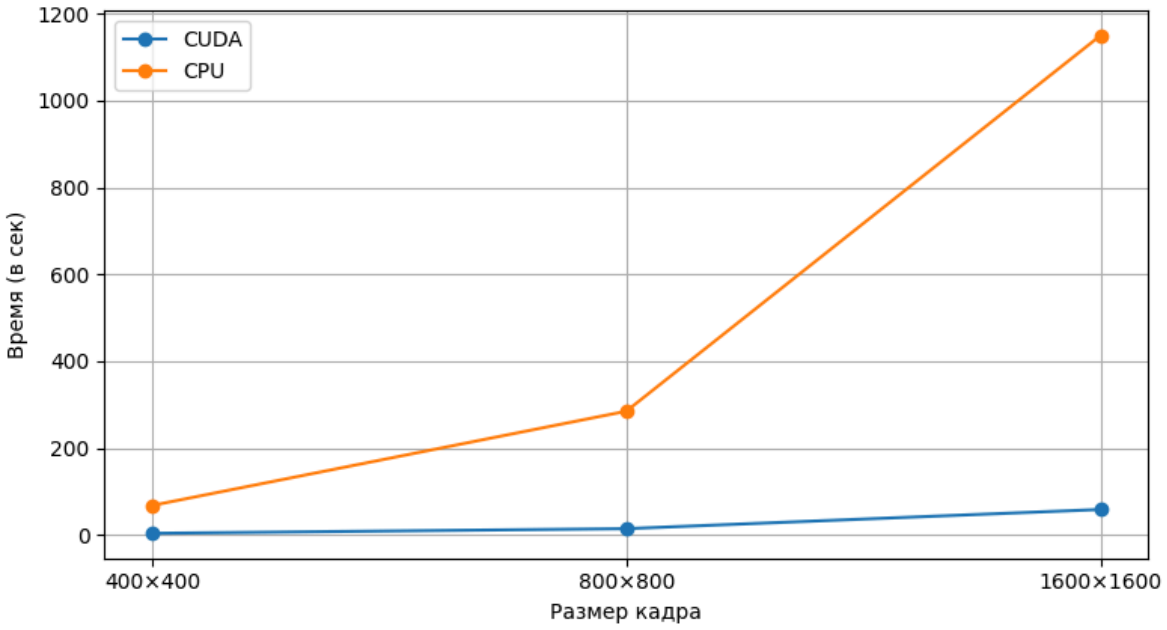
Увеличение количества потоков приводит к резкому сокращению времени выполнения. Наибольший прирост производительности достигается при переходе от 16×16 к 32×32, дальнейшее увеличение (64×64) даёт меньший эффект из-за насыщения ресурсов GPU и накладных расходов.

2) **Сравнение производительности CPU и GPU в зависимости от разрешения**

(рендеринг **120 кадров**)

Таблица 2 — Сравнение времени рендеринга на CPU и GPU

Размер кадра	Время на CUDA (сек)	Время на CPU (сек)
400×400	4	70
800×800	15	285
1600×1600	60	1150



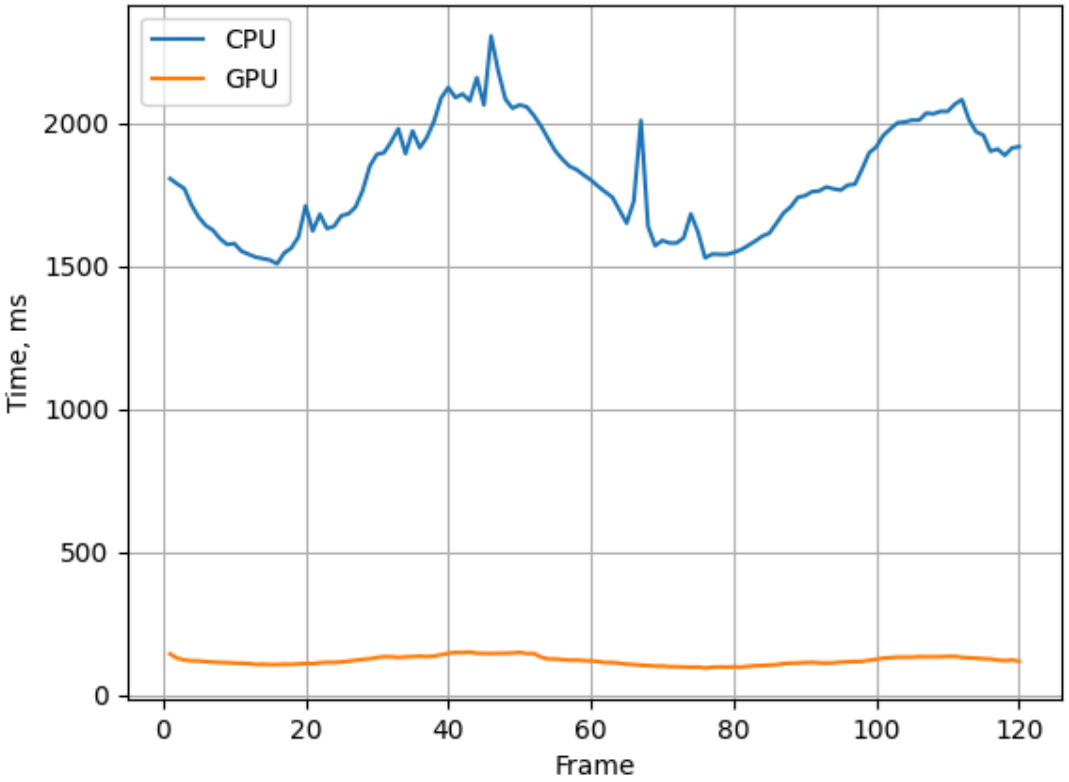
С ростом разрешения время выполнения на CPU увеличивается значительно быстрее, чем на GPU, поскольку GPU обрабатывает пиксели параллельно, а CPU — последовательно.

3) **Время обработки одного кадра (CPU vs GPU)**

(разрешение **800×400**, **120 кадров**)

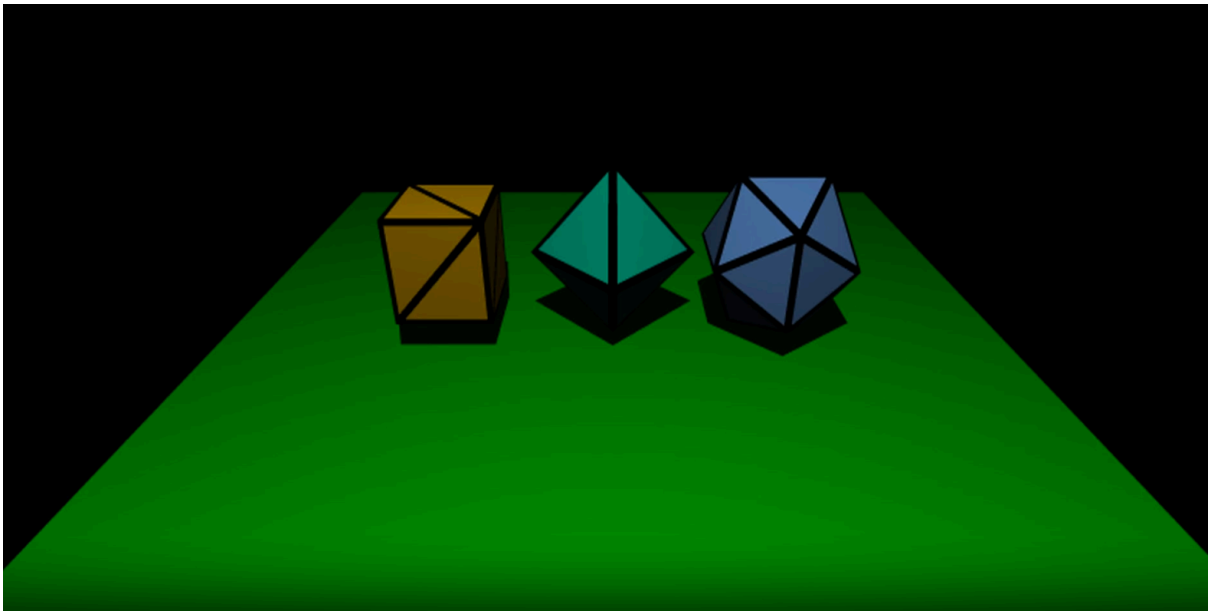
Таблица 3 — Среднее время обработки кадра

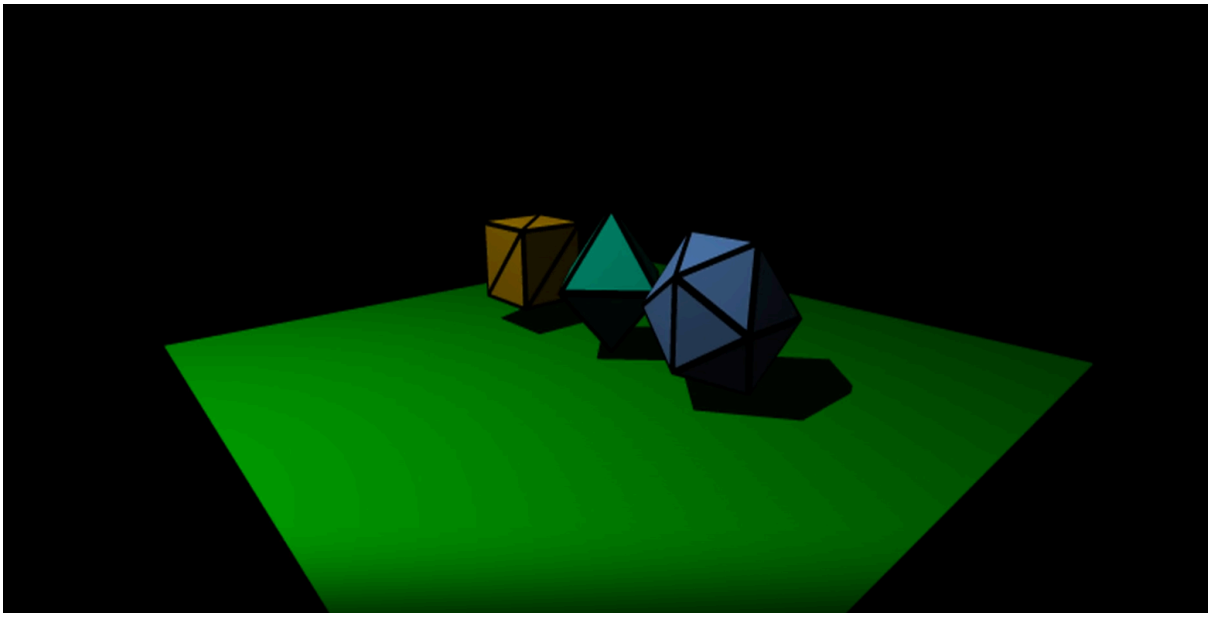
Реализация	Среднее время на кадр (мс)
CPU	~1800
GPU	~120

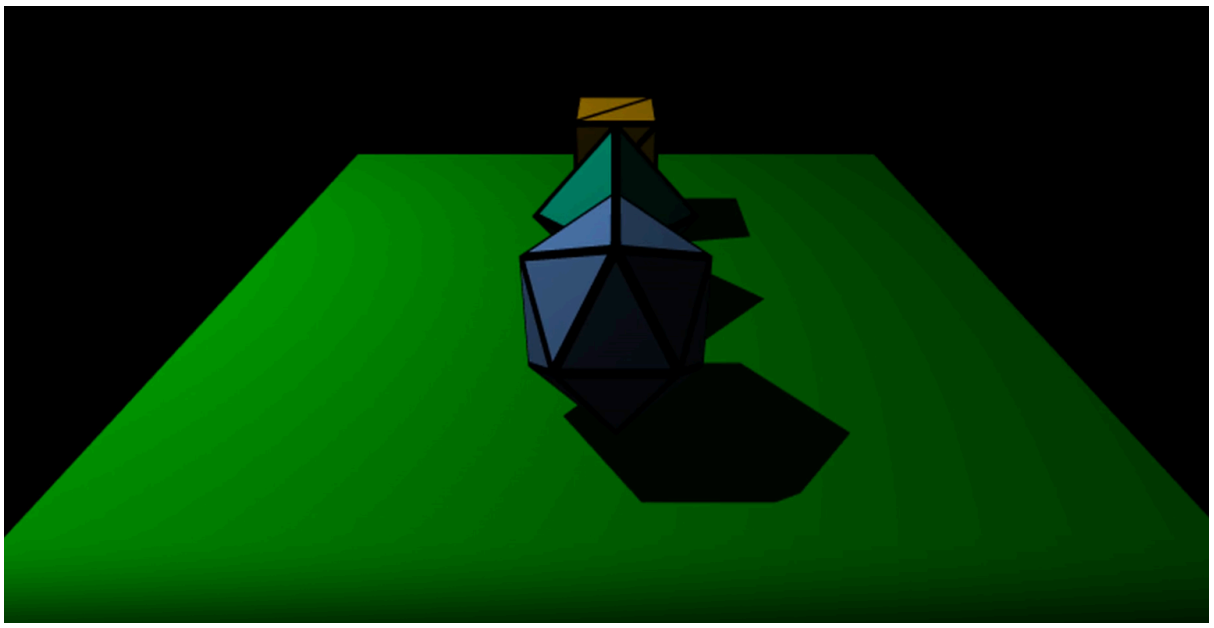


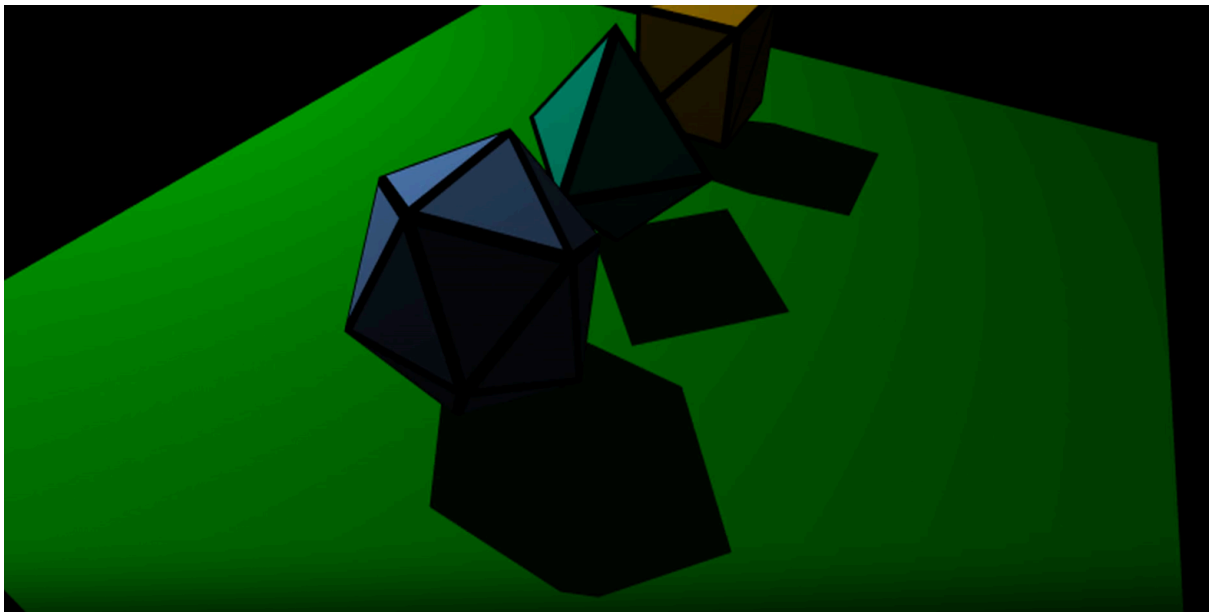
GPU демонстрирует более стабильное время обработки кадра и значительно меньшую длительность по сравнению с CPU; колебания CPU связаны с изменением положения камеры и долей видимых поверхностей/теней в сцене.

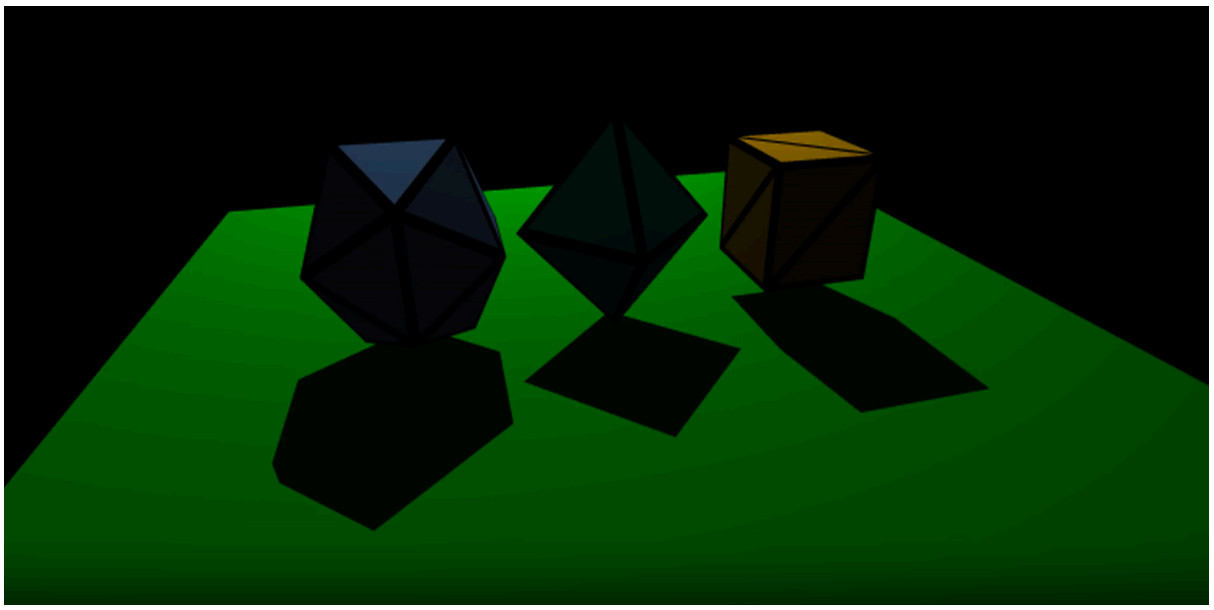
Примеры

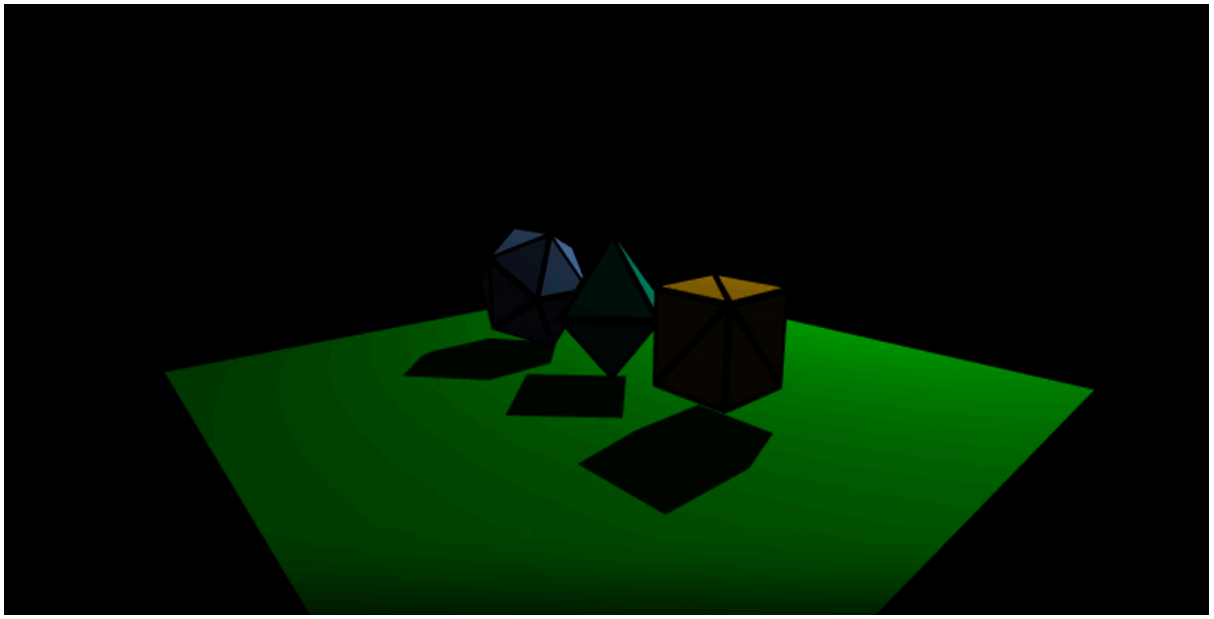


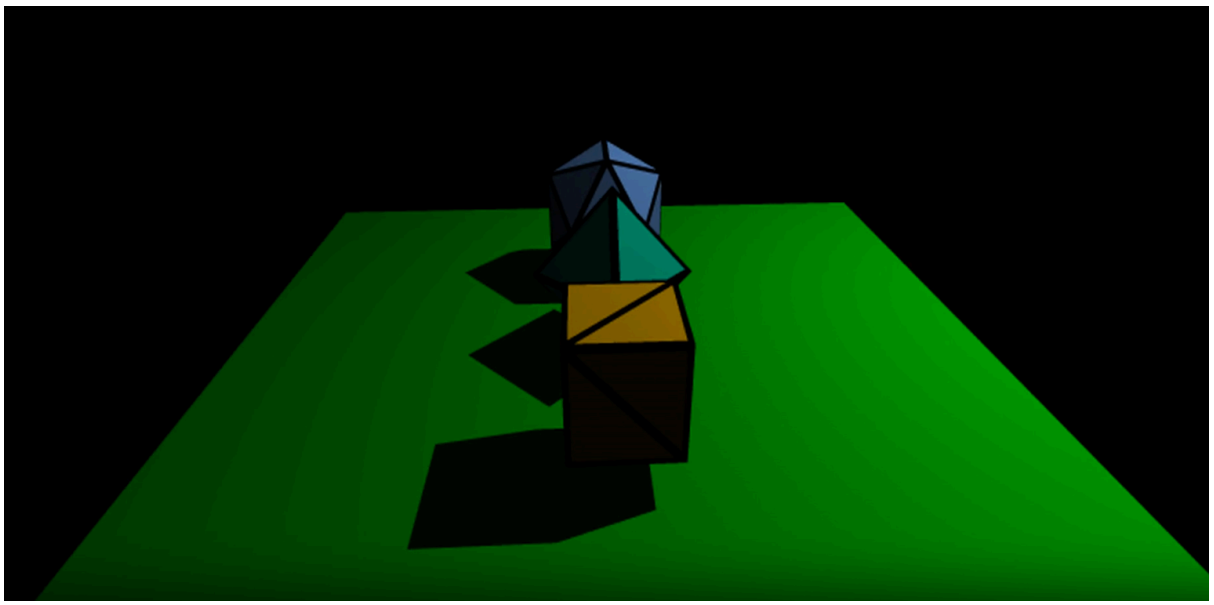


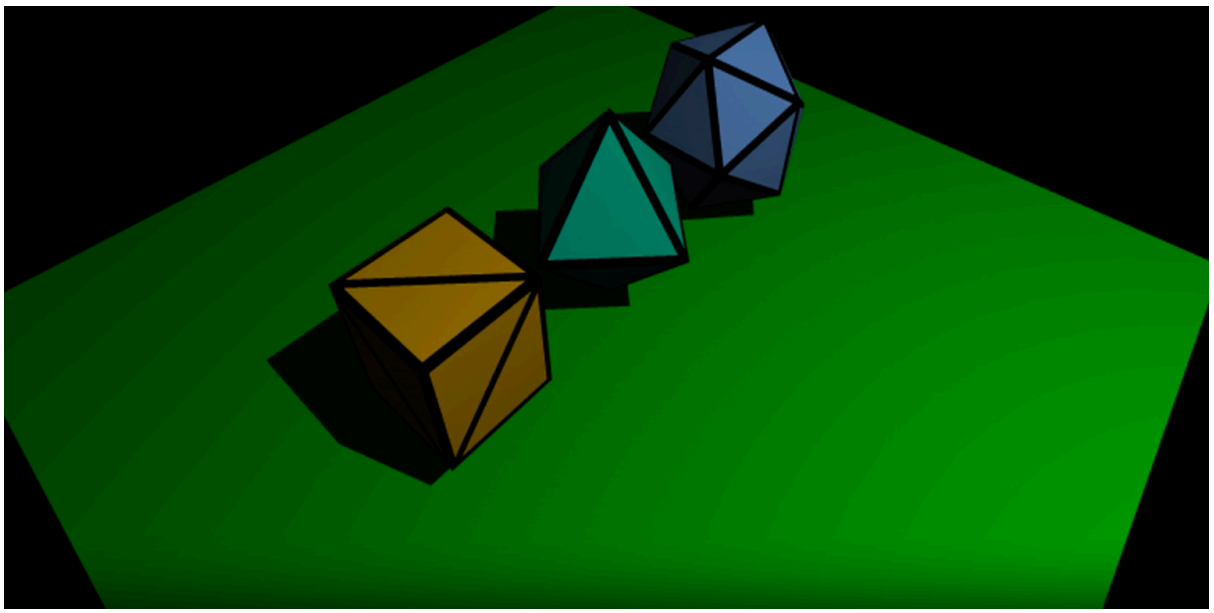


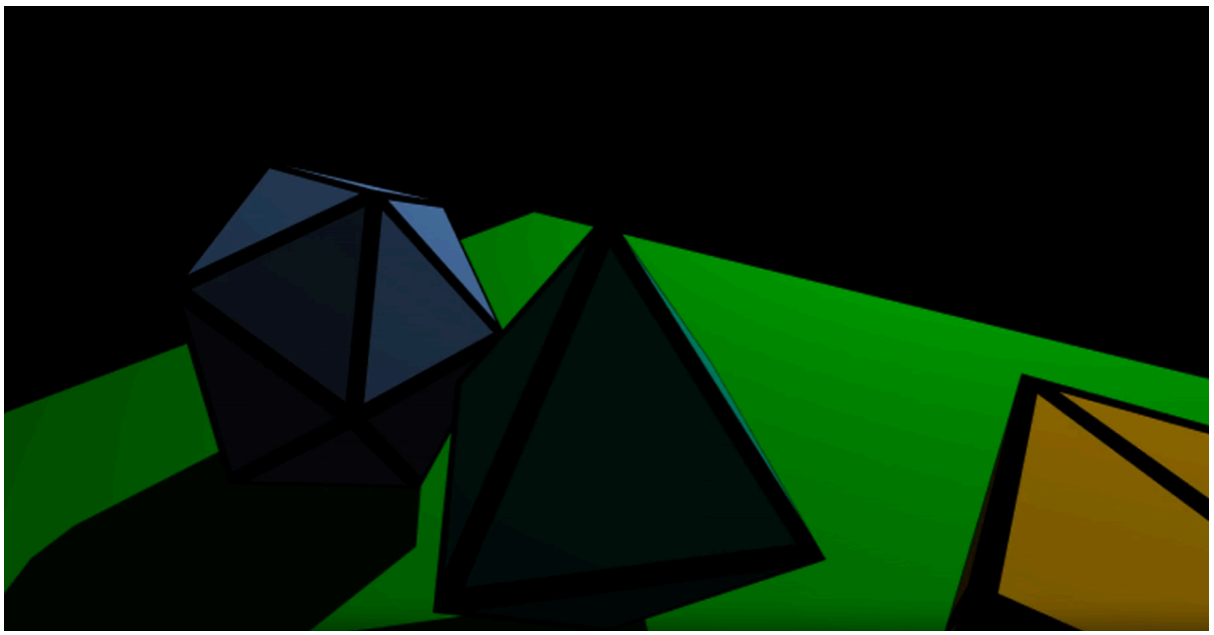


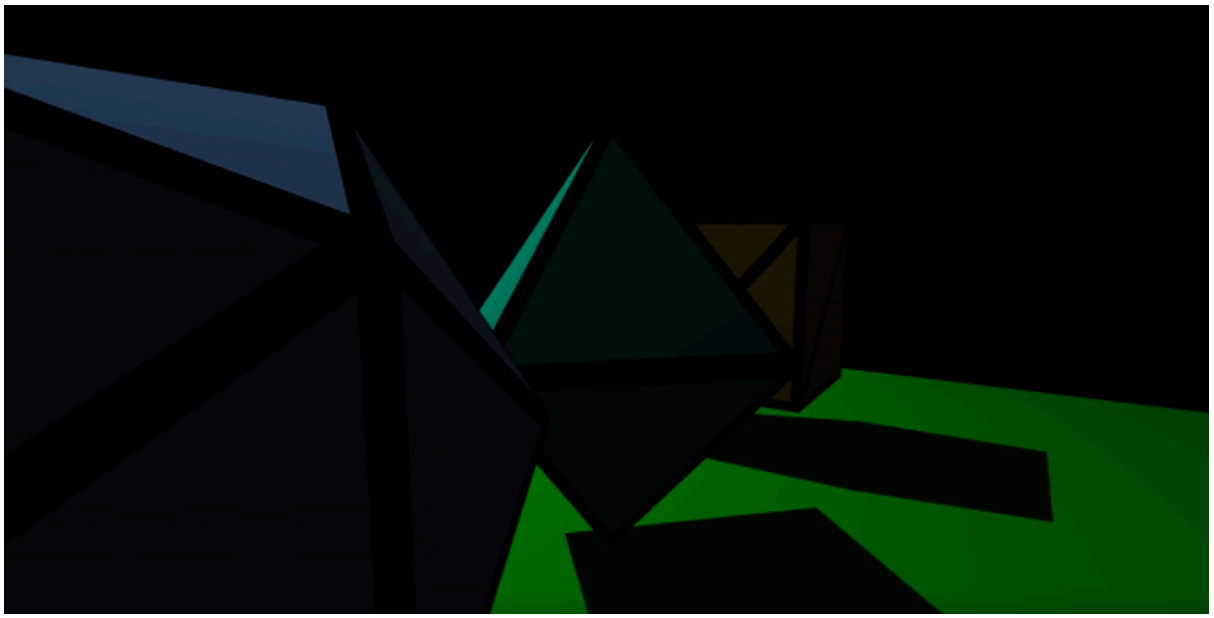












Выводы

В ходе лабораторной работы был реализован алгоритм обратной трассировки лучей с использованием GPU и CPU. Сцена формируется из треугольных полигонов, реализовано освещение, тени и сглаживание SSAA, а также покадровый рендеринг анимации с движущейся камерой. Проведённые измерения показали, что при росте разрешения и количества вычислений время работы на CPU существенно увеличивается, тогда как GPU справляется с задачей значительно быстрее за счёт параллельных вычислений. Это подтверждает эффективность использования видеокарты для задач фотореалистичной визуализации и рендеринга.