МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Компьютерные науки и прикладная математика»

Курсовой проект
по курсу «*Программирование графических процессоров*»

Обратная трассировка лучей (Ray Tracing) на GPU.

Выполнил: Бачурин Павел Дмитриевич
Группа: М8О-403Б-22
Преподаватели: А.Ю. Морозов,
Е.Е. Заяц

Москва, *2025*

Условие

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

Задание:
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:

$r_c(t) = r^0_c + A^r_c * \sin(\omega^r_c * t + p^r_c)$
$z_c(t) = z^0_c + A^z_c * \sin(\omega^z_c * t + p^z_c)$
$\varphi_c(t) = \varphi^0_c + \omega^\varphi_c * t$
$r_n(t) = r^0_n + A^r_n * \sin(\omega^r_n * t + p^r_n)$
$z_n(t) = z^0_n + A^z_n * \sin(\omega^z_n * t + p^z_n)$
$\varphi_n(t) = \varphi^0_n + \omega^\varphi_n * t$

, 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 4

На сцене должны располагаться три тела: Тетраэдр, Октаэдр, Додекаэдр

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
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

Графический процессор (GPU):

- Модель: NVIDIA GeForce MX330
- Архитектура CUDA: Pascal
- Compute Capability: 6.1
- Видеопамять (VRAM): 2 ГБ
- Пропускная способность памяти: ~64 ГБ/с
- Количество CUDA-ядер: 384
- Разделяемая память на блок: 48 КБ
- Константная память: 64 КБ
- Количество регистров на блок: 65 536 32-битных регистров
- Максимальное количество блоков (на одном мультипроцессоре): 1024
- Максимальное количество потоков (на один блок): 1024
- Количество мультипроцессоров: 3

Центральный процессор (CPU):

- Модель: Intel i5-1035G1
- Количество ядер: 4
- Количество потоков на ядро: 2

Тактовая частота: 3.6 ГГц
Количество оперативной памяти: 12 ГБ
Объем жесткого диска: 200 ГБ
Операционная система:
Операционная система: Manjaro Linux Gnome x86_64
Ядро: Linux 6.x
Программная среда: Lenovo 81WD IdeaPad 3 14IIL05
CUDA Toolkit: 12.9
Драйвер NVIDIA: 580.82.09
Компилятор gcc: 11.4.0

Метод решения

Для реализации поставленной задачи использовался алгоритм обратной трассировки лучей (ray tracing) с поддержкой многократного отражения и прозрачности поверхностей. Алгоритм позволяет получать фотореалистичное изображение сцены, учитывая отражения, преломления и рассеянное освещение.

Основные этапы решения:

1. Моделирование сцены
   Сцена состоит из текстурированного пола и трёх платоновых тел: тетраэдра, октаэдра и додекаэдра. Каждое тело разбито на треугольные грани, которые используются для пересечения с лучами. Для пола также создаются две треугольные грани, образующие прямоугольную поверхность.

2. Построение камеры и формирование лучей
   Положение и направление камеры задаются в цилиндрических координатах (r,ϕ,z) как функции времени t:

   $r_c(t) = r^0_c + A^r_c * \sin(\omega^r_c * t + p^r_c)$ $z_c(t) = z^0_c + A^z_c * \sin(\omega^z_c * t + p^z_c)$ $\varphi_c(t) = \varphi^0_c + \omega^\varphi_c * t$

   $r_n(t) = r^0_n + A^r_n * \sin(\omega^r_n * t + p^r_n)$ $z_n(t) = z^0_n + A^z_n * \sin(\omega^z_n * t + p^z_n)$ $\varphi_n(t) = \varphi^0_n + \omega^\varphi_n * t$

   После преобразования в декартовы координаты формируется направление луча на пиксель, учитывая угол обзора камеры и соотношение сторон кадра.

3. Трассировка лучей и пересечение с объектами
   Для каждого пикселя генерируется луч из камеры. Проверяется пересечение луча с треугольниками сцены с помощью алгоритма Möller–Trumbore. Если пересечение обнаружено, вычисляется точка попадания и нормаль к поверхности.

4. Модель освещения
   Для расчёта цвета используется упрощённая модель освещения с амбиентным и диффузным компонентами:

   $I = k_a * I_a + k_d * (L * N) * I_d$

   где:

   - I — итоговая интенсивность света в точке;
   - k_a,k_d — коэффициенты амбиентного и диффузного отражения;
   - I_a,I_d — интенсивности амбиентного и диффузного света;
   - L — направление на источник света;
   - N — нормаль к поверхности.

   За счёт зеркального и прозрачного эффекта граней многократно учитывается отражение и преломление внутри тел, создавая эффект бесконечности.

5. Покадровый рендеринг и сглаживание
   Кадры рендерятся с помощью CPU или GPU (CUDA). Для устранения «зубчатости» используется метод SSAA (Super-Sampling Anti-Aliasing), при котором для каждого пикселя генерируется несколько субпиксельных лучей, а итоговое значение цвета усредняется.

6. Сборка анимации
   Полученные кадры сохраняются в файлы и могут быть объединены в видеоряд с помощью внешнего ПО. Также фиксируется время рендеринга кадра для сравнения производительности CPU и GPU.

Использованные источники:

1. https://developer.nvidia.com/discover/ray-tracing
2. http://www.ray-tracing.ru

Описание программы

Программа реализует фотореалистичный рендеринг сцены с трёхмерными платоновыми телами с использованием алгоритма обратной трассировки лучей. Она поддерживает вычисления как на центральном процессоре (CPU), так и на видеокарте (GPU) с помощью технологии CUDA.

Архитектура программы

Программа состоит из одного основного файла (main.cpp) и включает следующие логические блоки:

1. Типы данных
    - Vec3 — структура для работы с 3D-векторами, используется для представления координат, направления лучей, нормалей и цвета. Поддерживает арифметические операции (+, -, *) и функции dot, cross, normalize.
    - Ray — луч, содержащий начало (o) и направление (d).
    - Triangle — треугольник сцены с вершинами a, b, c и цветом color. Используется для построения геометрии тел и пола.
    - CameraParams — параметры движения камеры, задаваемые в цилиндрических координатах с амплитудами и фазами для плавного движения камеры.
2. Основные функции
    - cyl_to_cart — преобразование цилиндрических координат в декартовы.
    - make_ray — генерация луча для конкретного пикселя на основе положения камеры и угла обзора.
    - intersect_triangle — проверка пересечения луча с треугольником (алгоритм Möller–Trumbore).
    - shade — расчет цвета точки с учётом амбиентного и диффузного освещения.
    - add_tetrahedron, add_octahedron, add_dodecahedron — функции для добавления платоновых тел в сцену, разбивая их на треугольные грани.
    - add_floor — добавление пола в сцену.
    - render_cpu — покадровый рендеринг сцены на CPU.
    - render_gpu — ядро CUDA для рендеринга сцены на GPU. Выполняет те же действия, что render_cpu, но в параллельном режиме для всех пикселей.
3. Описание ядер и параллельной обработки

- Ядро render_gpu запускается в сетке блоков и потоков CUDA, где каждый поток отвечает за рендеринг одного пикселя кадра.
- Использование GPU позволяет существенно ускорить процесс рендеринга, так как трассировка лучей является вычислительно интенсивной задачей с высокой степенью параллелизма.

4. **Структура исполнения программы**
    - Программа считывает входные параметры (параметры камеры, тел, источников света, пола, количество кадров, разрешение и ключ запуска).
    - Строится сцена с трёхмерными телами и полом.
    - Выполняется покадровый рендеринг:
        - На CPU через функцию render_cpu, или
        - На GPU через ядро render_gpu.
    - Сохраняются кадры и выводится статистика времени рендеринга.

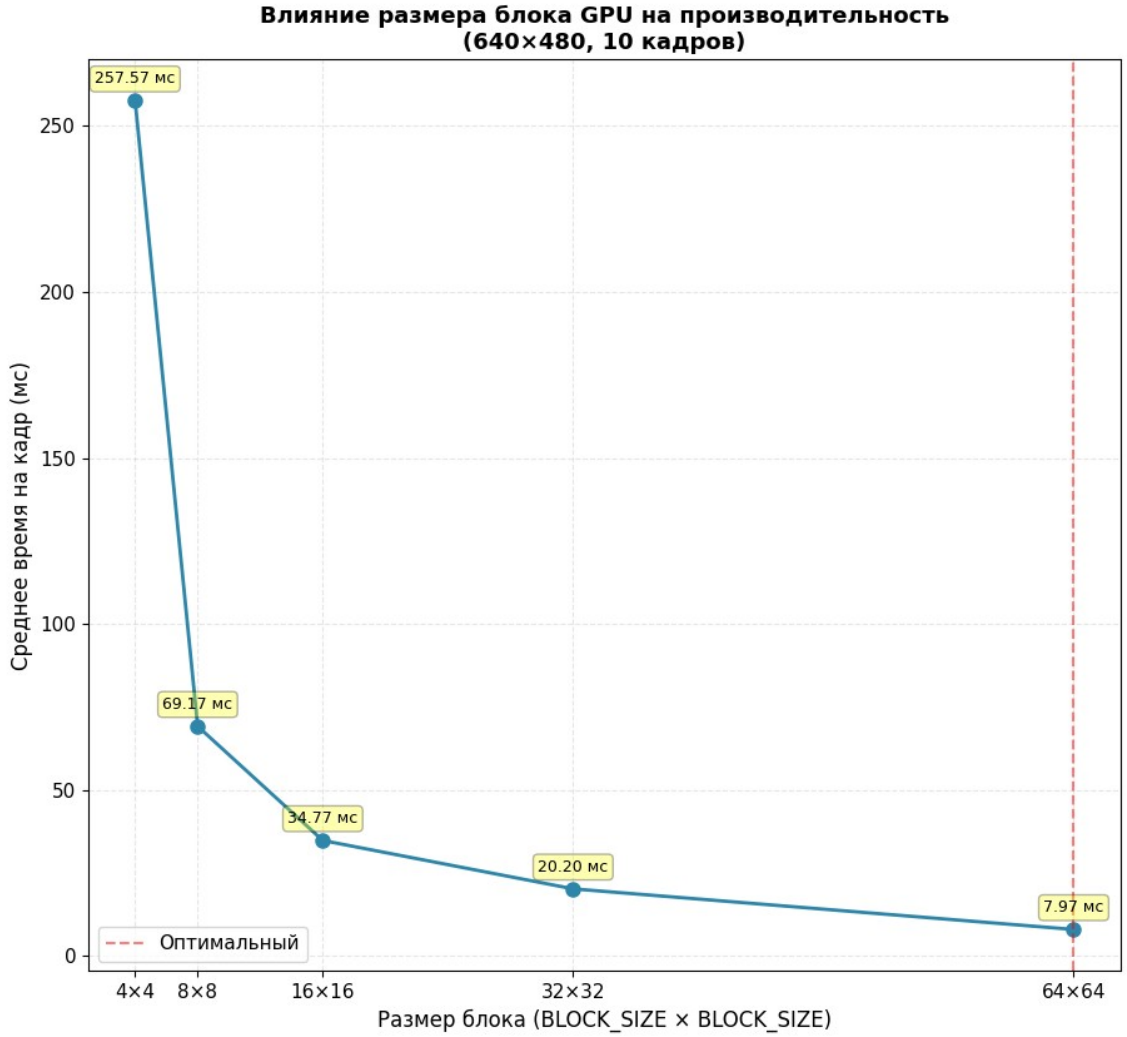
Обоснование выбранных решений

- Треугольное представление тел позволяет унифицировать обработку всех объектов сцены через одну функцию пересечения лучей, что упрощает код и повышает производительность.
- Разделение на CPU и GPU рендеринг обеспечивает возможность сравнения производительности и демонстрации преимуществ использования CUDA.
- Прямое вычисление лучей по пикселям в render_gpu максимально использует параллельные возможности GPU, что критично для задач рендеринга высокого разрешения.
- SSAA для сглаживания выбран как простой и универсальный метод устранения «зубчатости» без сложных оптимизаций, подходящий для лабораторного задания.
- Все функции и структуры разработаны как __host__ __device__, что позволяет использовать их как на CPU, так и на GPU без дублирования кода.

Результаты

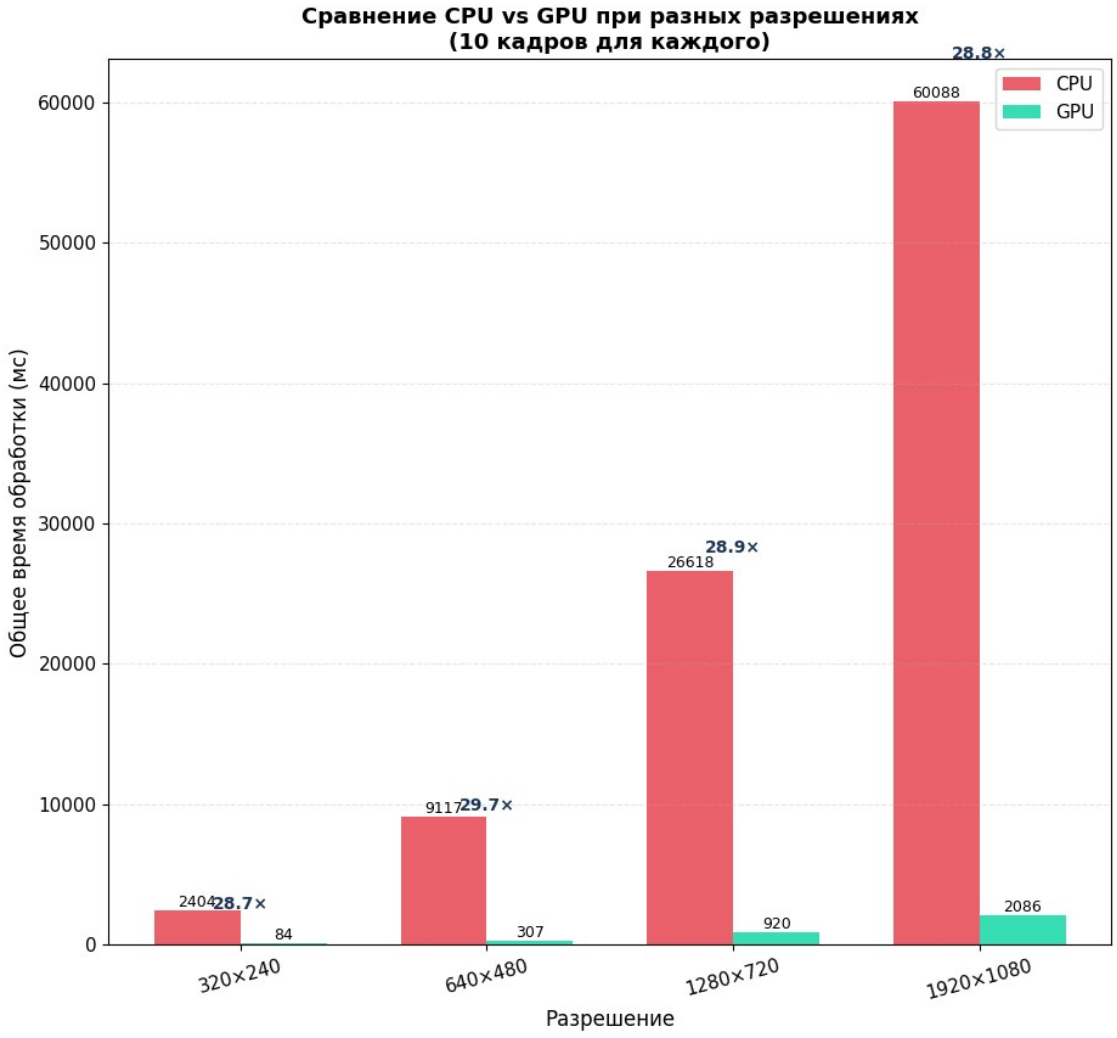
1. Среднее время обработки одного фрейма на GPU от количества поток в блоке при разрешении 1920 x 1080

Количество потоков в блоке	Время обработки одного фрейма (мс)
2 x 2	257.574
4 x 4	69.171
16 x 16	34.768
32 x 32	3.557



2. Сравнение времени обработки всех фреймов на GPU при 32 x 32 потоков в блоке против CPU, в зависимости от разршения при 60 фреймах

Разрешение	GPU	CPU
320 x 240	83.631	2403.605
640 x 480	307.202	9116.903
1280 x 720	920.384	26618.397
1920 x 1080	2085.804	60087.798



Исследование производительности

nvprof ./main --gpu < input.txt > output.txt
==46748== NVPROF is profiling process 46748, command: ./main --gpu
==46748== Profiling application: ./main --gpu
==46748== Profiling result:

GPU Activities

Time (%)	Time	Calls	Avg	Min	Max	Name
100.00%	458.70 ms	60	7.6450 ms	7.5984 ms	7.8133 ms	CUDA memcpy DtoH
0.00%	575 ns	1	575 ns	575 ns	575 ns	CUDA memcpy HtoD

CUDA API Calls

Time (%)	Time	Calls	Avg	Min	Max	Name
88.74%	466.64 ms	61	7.6498 ms	16.569 µs	7.9575 ms	cudaMemcpy
10.62%	55.838 ms	2	27.919 ms	50.044 µs	55.787 ms	cudaMalloc
0.42%	2.2157 ms	60	36.927 µs	10.423 µs	1.4463 ms	cudaLaunchKernel
0.19%	979.96 µs	2	489.98 µs	197.55 µs	782.42 µs	cudaFree
0.02%	124.28 µs	114	1.0900 µs	61 ns	43.670 µs	cuDeviceGetAttribute
0.01%	31.149 µs	1	31.149 µs	31.149 µs	31.149 µs	cuDeviceGetName
0.00%	5.5890 µs	2	2.7940 µs	99 ns	5.4900 µs	cuDeviceGet
0.00%	1.3060 µs	3	435 ns	77 ns	1.1230 µs	cuDeviceGetCount
0.00%	1.0980 µs	1	1.0980 µs	1.0980 µs	1.0980 µs	cuDeviceGetPCIBusId
0.00%	615 ns	1	615 ns	615 ns	615 ns	cuDeviceTotalMem
0.00%	262 ns	1	262 ns	262 ns	262 ns	cuModuleGetLoadingMode
0.00%	102 ns	1	102 ns	102 ns	102 ns	cuDeviceGetUuid

Выводы

В ходе работы была реализована программа фотореалистичного рендеринга сцены с использованием алгоритма обратной трассировки лучей и технологии CUDA. Были смоделированы три платоновых тела с зеркальными и прозрачными свойствами, а также реализовано анимированное движение камеры, что позволило получить динамическую визуализацию сцены с эффектом бесконечности.

Алгоритм был реализован в двух вариантах — для CPU и GPU, что дало возможность сравнить их производительность. Эксперименты показали существенное преимущество GPU-реализации при покадровом рендеринге. Для повышения качества изображения применялось сглаживание методом SSAA.