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: И. Д. Павлов
Группа: М8О-407Б-21
Преподаватель: А. Ю. Морозов

Москва, 2025

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 которойрасположены три платоновых тела. Сверху находятся несколько источников света. Накаждом ребре многогранника располагается определенное количество точечныхисточников света. Грани тел обладают зеркальным и прозрачным эффектом. За счетмногократного 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 вмомент времени определяется следующим образом:

$$r_c(t) = r_c^0 + A_c^r \sin(\omega_c \cdot t + p_c^r)$$

$$z_c(t) = z_c^0 + A_c^z \sin(\omega_c \cdot t + p_c^z)$$

$$\phi_c(t) = \phi_c^0 + \omega_c t$$

$$r_n(t) = r_n^0 + A_n^r \sin(\omega_n \cdot t + p_n^r)$$

$$z_n(t) = z_n^0 + A_n^z \sin(\omega_n \cdot t + p_n^z)$$

$$\phi_n(t) = \phi_n^0 + \omega_n t$$

где

$$t \in [0, 2\pi]$$

Общая постановка задачи

Требуется реализовать алгоритм обратной трассировки лучей(http://www.ray-tracing.ru/) с использованием технологии CUDA. Выполнить покадровыйрендеринг сцены. Для устранения эффекта «зубчатости», выполнить сглаживание(например с помощью алгоритма SSAA). Полученный набор кадров склеить ванимацию любым доступным программным обеспечением. Подобрать параметрысцены, камеры и освещения таким образом, чтобы получить наиболее красочныйрезультат. Провести сравнение производительности gpu и cpu (т.е. дополнительнонужно реализовать алгоритм без использования CUDA).

Вариант 10

На сцене должны располагаться три тела: Октаэдр, Додекаэдр, Икосаэдр.

Программное и аппаратное обеспечение

- Compute capability: 7.5
- Наименование: NVIDIA GeForce GTX 1650
- Графическая память: 4096MiB
- Разделяемая память на блок: 49152 байт
- Количество регистров на блок: 65536
- Максимальное количество потоков на блок: (1024, 1024, 64)
- Максимальное количество блоков: (2147483647, 65535, 65535)
- Константная память: 65536 байт
- Количество мультипроцессоров: 14

Характеристики системы:

- Процессор: «AMD Ryzen 5 4600H with Radeon Graphics»
- Память: «16 Гб (2x8) SODIMM 3200 МГц C18»
- SSD: «Samsung SSD 1TB»

Программное обеспечение:

- ОС: «Ubuntu23.10»
- Текстовый редактор: «vscode»
- Компилятор: «nvcc: Cuda compilation tools, release 12.0, V12.0.140»

Метод решения

Модель освещения

В основе освещения лежит физическая модель распространения света, включающая прямое и рассеянное освещение, отражения и преломления. Реализована модель, основанная на комбинации диффузного, зеркального (спекулярного) и фонового (амбьентного) компонентов. Каждый из компонентов отвечает за свой вклад в финальный цвет пикселя.

- **Фоновое освещение (Ambient Light):** Этот компонент моделирует рассеянный свет, который не имеет конкретного направления и равномерно освещает все поверхности.
- **Точечные источники света (Point Light):** Эти источники излучают свет из одной точки во всех направлениях. Для каждого точечного источника света вычисляется направление к нему (`L`) и его вклад в освещённость поверхности.
- **Диффузное освещение (Diffuse Reflection):** Это освещение зависит от угла падения света на поверхность. Для расчёта используется косинус угла между нормалью поверхности (`N`) и направлением на источник света (`L`). Это можно выразить через скалярное произведение:

  $$I_{diffuse} = I \cdot \frac{(N \cdot L)}{\|N\|\|L\|}$$

  где I — интенсивность света, N — нормаль поверхности, L — вектор направления к источнику света.
- **Зеркальное отражение (Specular Reflection):** Этот компонент моделирует блики на поверхности и зависит от угла между вектором отражения света (`R`) и вектором наблюдателя (`V`). Можно использовать формулу Фонга:

  $$I_{specular} = I \cdot \left(\frac{R \cdot V}{\|R\|\|V\|} \right)^{\alpha}$$

  где I — интенсивность света, R — отражённый вектор, V — вектор наблюдателя, α — коэффициент блеска.

Процесс освещения включает следующие шаги:

1. Вычисление фоновой освещённости.
2. Для каждого точечного источника света:
   - Проверка наличия теней.
   - Вычисление диффузного и зеркального компонентов.
   - Суммирование вклада источника света.

Тени и затенение

Функция обработки теней проверяет, блокируется ли путь света к точке другими объектами сцены. Если на пути обнаруживается пересечение с объектом, интенсивность света уменьшается, а точка становится частично или полностью затенённой.

Отражения и преломления

Важной частью алгоритма освещения являются отражения и преломления:

- **Отражение (Reflection):** Направление отражённого луча вычисляется по закону отражения: $R = D - 2(n \cdot D)N$, где R - отраженный вектор, D - Падающий вектор, N - нормаль поверхности.
- **Преломление (Refraction):** Направление преломлённого луча вычисляется с использованием закона Снеллиуса.

Антиалиасинг (SSAA)

Для улучшения качества изображения и уменьшения ступенчатости границ используется метод Super Sampling Anti-Aliasing (SSAA). Метод заключается в вычислении нескольких лучей для каждого пикселя и усреднении их цвета.

Трассировка лучей

В основе метода лежит принцип отслеживания пути луча света, выпущенного из камеры, через каждый пиксель изображения и взаимодействующего с объектами сцены. Для каждого луча вычисляется его пересечение с геометрическими примитивами (треугольниками, прямоугольниками и другими фигурами). После этого определяется, как луч взаимодействует с поверхностью: отражается, преломляется или поглощается. Алгоритм:

1. Выпуск лучей из камеры через каждый пиксель изображения.
2. Определение точки пересечения луча с объектами сцены.
3. Вычисление нормали в точке пересечения.
4. Оценка освещённости точки с учётом фонового, диффузного и зеркального освещения.
5. Рекурсивная генерация новых лучей для обработки отражений и преломлений.
6. Усреднение цветов с использованием антиалиасинга (SSAA).

Описание программы

Описание программы

Функции работы с векторами

Математические операции над трёхмерными векторами представлены структурой `double3`. Для упрощения вычислений определены вспомогательные функции:

- `dot` — скалярное произведение двух векторов: $dot(a,\ b) = a.x \cdot b.x + a.y \cdot b.y + a.z \cdot b.z$

- `prod` — векторное произведение двух векторов:

$$\texttt{prod(a, b)} = \begin{bmatrix} a.y \cdot b.z - a.z \cdot b.y \\ a.z \cdot b.x - a.x \cdot b.z \\ a.x \cdot b.y - a.y \cdot b.x \end{bmatrix}$$

- `norm` — нормализация вектора:

$$\texttt{norm(v)} = \frac{v}{\|v\|}$$

- `diff` — разность двух векторов:

$$\texttt{diff(a, b)} = a - b$$

- `add` — сумма двух векторов:

$$\texttt{add(a, b)} = a + b$$

- `prod_num` — умножение вектора на число:

$$\texttt{prod\_num(a, b)} = a \cdot b$$

- `reflect` — вычисление отражённого вектора по нормали:

$$R = D - 2(N \cdot D)N$$

- `refract` — вычисление преломлённого вектора с использованием закона Снеллиуса.

Эти функции обеспечивают базовые операции для геометрических расчётов в трассировке лучей.

Работа с треугольниками и прямоугольниками

Треугольники (`Trig`)

Треугольники описываются структурой `Trig`, включающей вершины (`a`, `b`, `c`), цвет (`color`), коэффициенты отражения (`r`) и преломления (`tr`).

Основные функции:

- `trig_init` — инициализация треугольника параметрами.
- `trig_at` — возвращает цвет в заданной точке пересечения, определяя границы треугольника с помощью барицентрических координат.
- `trig_intersect` — вычисляет пересечение луча с треугольником, используя алгоритм Мёллера-Трамбора.
- `trig_normal` — вычисляет нормаль треугольника через векторное произведение его рёбер.

Прямоугольники (`Rect`)

Прямоугольники описываются структурой `Rect`, включающей вершины (`a`, `b`, `c`, `d`), цвет (`color`), текстуру (`texW`, `texH`), коэффициенты отражения (`r`) и преломления (`tr`).

Основные функции:

- `rect_init` — инициализация прямоугольника параметрами.
- `rect_intersect` — проверяет пересечение луча с прямоугольником, представляя его как два треугольника.
- `rect_at` — возвращает текстурный цвет в заданной точке.
- `rect_normal` — вычисляет нормаль прямоугольника через векторное произведение рёбер.

Построение сцены

Функция `build_space` отвечает за инициализацию объектов сцены, включая пол, октаэдр, додекаэдр и икосаэдр. Каждый объект описывается своей геометрией, положением, масштабом и материалами.

Построение пола

Пол задаётся в виде прямоугольника, инициализированного с использованием структуры данных. Параметры включают четыре вершины, цвет и размеры текстуры. Прямоугольник добавляется в массив объектов сцены.

Октаэдр

Октаэдр создаётся на основе предопределённых координат его вершин, которые масштабируются относительно радиуса фигуры и сдвигаются в пространстве согласно положению центра. Каждая грань октаэдра задаётся в виде треугольника и добавляется в массив треугольников.

Додекаэдр

Додекаэдр имеет более сложную структуру, включающую 20 граней. Его вершины определяются вручную, после чего они масштабируются и смещаются относительно центра фигуры. Для каждой грани выполняется инициализация треугольников, которые добавляются в массив.

Икосаэдр

Икосаэдр создаётся аналогично додекаэдру, но с меньшим количеством граней. Его вершины определяются, затем масштабируются и смещаются. Каждая грань представлена треугольником, который добавляется в массив объектов.

Инициализация источников света

Функция `init_lights` отвечает за настройку массива источников света, которые используются для расчёта освещения сцены.

Типы источников света

Функция поддерживает два типа источников:

- **Фоновое освещение** (`AMBIENT`) — равномерное освещение сцены. Оно задаётся с фиксированной интенсивностью (`0.5`) и добавляется как первый элемент массива источников света.
- **Точечное освещение** (`POINT`) — источники света, излучающие свет из конкретных точек в пространстве. Их параметры, такие как положение (`position`) и интенсивность (`1.0`), задаются на основе данных из структуры `LightDto`.

Алгоритм инициализации

- Первый элемент массива света инициализируется как фоновой источник.
- Остальные элементы массива заполняются точечными источниками, информация о которых берётся из массива `lights_data`.

Инициализированные источники света добавляются в массив `lights` и используются для расчёта освещения и теней в сцене.

Расчёт теней и освещения

Функции `compute_shadow` и `compute_lighting` реализуют расчёт теней и освещения в сцене, учитывая различные объекты и источники света.

Расчёт теней

Функция `compute_shadow` проверяет, блокируется ли луч от точки к источнику света другими объектами. Для этого выполняются следующие шаги:

- Для каждого прямоугольника вызывается функция `rect_intersect`, чтобы проверить пересечение луча с прямоугольником. Если пересечение найдено, коэффициент прозрачности (`tr`) прямоугольника учитывается в итоговом индексе теней.
- Аналогично, для каждого треугольника вызывается функция `trig_intersect`, чтобы проверить пересечение луча с треугольником. Его коэффициент прозрачности также влияет на индекс теней.
- Итоговый индекс теней (`shadow_index`) уменьшается в зависимости от прозрачности объектов, перекрывающих источник света.

Расчёт освещения

Функция `compute_lighting` вычисляет освещение в заданной точке сцены с учётом фонового, диффузного и зеркального компонентов. Основные этапы:

- Для фонового света (`AMBIENT`) добавляется его интенсивность к общему освещению.
- Для каждого точечного света (`POINT`) выполняется:
  - Вычисление направления от точки к источнику света (`L`).
  - Определение, блокируется ли свет объектами, с помощью функции `compute_shadow`.
  - Расчёт диффузного компонента, пропорционального скалярному произведению вектора нормали (`N`) и направления света (`L`).
  - Расчёт зеркального компонента, зависящего от угла между отражённым светом (`R`) и направлением на наблюдателя (`V`). Вклад зеркального света повышается для гладких поверхностей.
- Итоговое освещение складывается из фонового, диффузного и зеркального компонентов с учётом теней.

Обе функции совместно обеспечивают реалистичное освещение и корректную обработку теней в сцене.

Рекурсивная трассировка лучей

Функция `ray` реализует процесс трассировки лучей для вычисления цвета пикселя, учитывая эффекты отражения, преломления, освещения и теней. Она используется для расчёта цвета точки сцены при попадании луча в объект.

Основные этапы трассировки

- **Проверка попадания в источник света:** На начальном этапе функция проверяет, находится ли луч в непосредственной близости от источника света. Если это так, возвращается белый цвет.
- **Поиск пересечения:** Луч проверяется на пересечение со всеми объектами сцены (прямоугольниками и треугольниками). Выбирается ближайший объект, если пересечение найдено.
- **Расчёт локального освещения:** После нахождения точки пересечения:
  - Определяется базовый цвет объекта в точке пересечения.
  - Вычисляется локальное освещение с учётом источников света и теней с помощью функции `compute_lighting`.
- **Обработка отражения и преломления:** Если объект обладает свойствами отражения (`r`) или преломления (`tr`), рассчитываются дополнительные лучи:
  - Направление отражённого луча вычисляется через функцию `reflect`.

  - Направление преломлённого луча вычисляется через функцию `refract`, если коэффициент преломления больше нуля.

- **Комбинирование цвета:** Итоговый цвет точки вычисляется как взвешенная сумма локального цвета, отражённого и преломлённого компонентов, где веса зависят от коэффициентов материала объекта.

Шаблонная рекурсия

Функция `ray` реализована с использованием шаблонной рекурсии для ограничения глубины трассировки:

- Шаблонный параметр `depth` отслеживает текущую глубину рекурсии.
- Базовый случай рекурсии (`ray<4>`) возвращает чёрный цвет, чтобы предотвратить бесконечное вычисление (4 можно заменить на 8 или другую глубину рекурсии).
- Рекурсивные вызовы `ray<depth + 1>` используются для расчёта цвета от отражённых и преломлённых лучей.

Рендеринг сцены

Функции `render_kernel` и `render` реализуют рендеринг сцены на видеокарте и процессоре соответственно. Они вычисляют цвета каждого пикселя изображения, выполняя трассировку лучей для всех объектов сцены.

Рендеринг на видеокарте (`render_kernel`)

- **Распределение потоков:** Каждому потоку назначается уникальный пиксель для вычисления цвета. Индексы пикселей вычисляются на основе параметров сетки (`blockDim`, `blockIdx`, `threadIdx`) и шага сетки (`offsetx`, `offsety`).
- **Формирование лучей:** Для каждого пикселя создаётся луч, который определяется:
  - Направлением камеры (`pv`) и её позиции (`pc`).
  - Полем зрения (`angle`).
  - Координатами пикселя в нормализованной системе координат.

  Луч нормализуется перед использованием в трассировке.
- **Расчёт цвета:** Цвет пикселя вычисляется с использованием функции `ray`, которая возвращает результат трассировки для заданного луча. Результат сохраняется в массив `data`.

Рендеринг на процессоре (`render`)

Функция `render` выполняет аналогичную задачу, но без параллельного распределения вычислений:

- Циклы `for` используются для итерации по всем пикселям изображения.

- Для каждого пикселя формируется луч и вычисляется его цвет с помощью функции `ray`.
- Результат сохраняется в массив `data`.

Общие этапы рендеринга

- **Камера:** Определяются базисные векторы (`bx`, `by`, `bz`), описывающие ориентацию камеры. Позиция и направление камеры задаются параметрами `pc` и `pv`.
- **Проекция:** Пиксели на плоскости проекции преобразуются в трёхмерные лучи с учётом размеров изображения (`w`, `h`) и угла обзора (`angle`).
- **Результат трассировки:** Цвет пикселя вычисляется на основе взаимодействия луча с объектами сцены, включая отражения, преломления и освещение.

Super Sampling Anti-Aliasing (SSAA)

Super Sampling Anti-Aliasing (SSAA) используется для сглаживания изображения путём уменьшения разрешения с более высокого (исходного) уровня детализации до целевого. Это позволяет снизить эффект ступенчатости на границах объектов.

Функция `ssaa_pixel`

Функция `ssaa_pixel` выполняет усреднение цвета пикселя из высокодетализированного изображения (`big_data`) для генерации пикселя с меньшим разрешением. Основные шаги:

- Для каждого целевого пикселя определяются `kxk` соседних пикселей из исходного изображения.
- Суммируются значения цветов (компоненты `R`, `G`, `B`, `A`) этих пикселей.
- Средние значения компонент вычисляются и обрезаются до диапазона [0, 255], чтобы избежать переполнения.
- Результат возвращается в виде сглаженного пикселя.

Функция `ssaa_kernel`

Функция `ssaa_kernel` выполняет SSAA в параллельном режиме на GPU. Основные этапы:

- Каждому потоку CUDA назначается уникальный пиксель целевого изображения.
- Вызывается функция `ssaa_pixel`, чтобы рассчитать цвет целевого пикселя на основе исходного изображения.
- Результат записывается в массив `small_data`.

Функция `main`

Главная функция `main` обеспечивает управление всей программой, включая ввод параметров, инициализацию объектов и выполнение рендеринга. Основные этапы:

1. Обработка аргументов командной строки

- Флаг `-cpu` переключает выполнение на процессор.
- Флаг `-default` выводит шаблонные параметры рендеринга.

2. Инициализация параметров сцены и объектов

- Чтение параметров камеры, источников света, объектов и пола с использованием `scanf`.
- Инициализация массива фигур (`FigureDto`) и структуры пола (`FloorDto`).
- Загрузка текстуры пола из файла.
- Вызов функций `build_space` и `init_lights` для создания объектов сцены и источников света.

3. Установка памяти для GPU

Если выбрано выполнение на GPU:

- Выделяется память на устройстве для объектов, текстур и данных рендеринга.
- Выполняется копирование данных с хоста на устройство (`cudaMemcpy`).

4. Рендеринг фреймов

В цикле рендеринга:

- Для каждого фрейма вычисляются параметры движения камеры.
- Вызов `render_kernel` для рендеринга на GPU или `render` для выполнения на CPU.
- Выполняется антиалиасинг с помощью `ssaa_kernel` или `ssaa_pixel`.
- Сохранение результата в файл.
- Замер времени выполнения (на CPU или GPU).

5. Очистка ресурсов

- Освобождается память на устройстве и хосте.
- Закрываются файлы.

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.

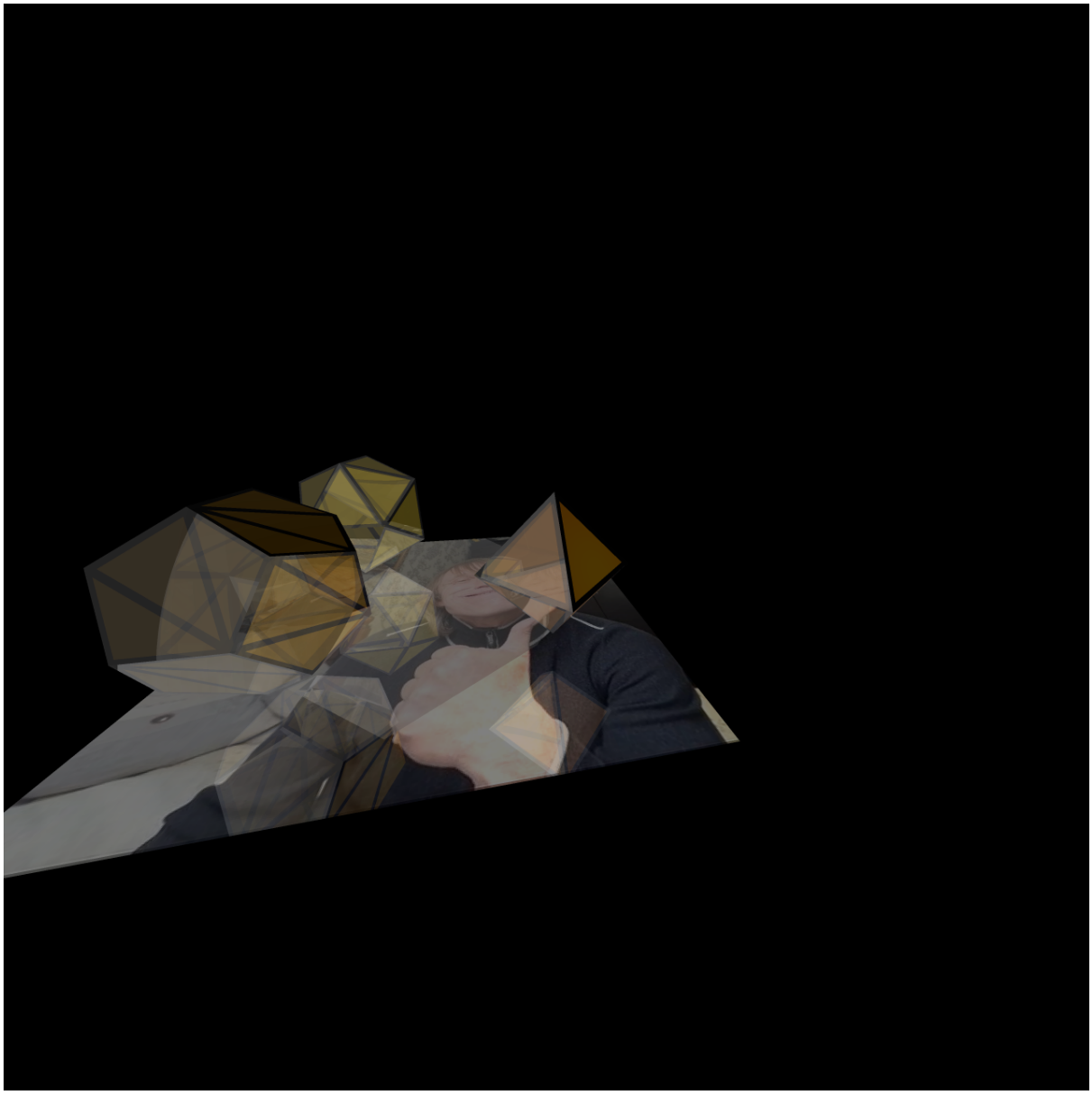
Конфигурация	$k = 1$	$k = 2$	$k = 4$	$k = 8$
CPU	1793.745972	2596.300049	3561.424072	6200.254883
(16, 16)	98.345671	147.523540	197.653487	356.432156
(64, 64)	67.432112	101.148170	135.789045	243.567890
(256, 256)	56.576576	78.791267	91.677856	162.876542

Входные данные

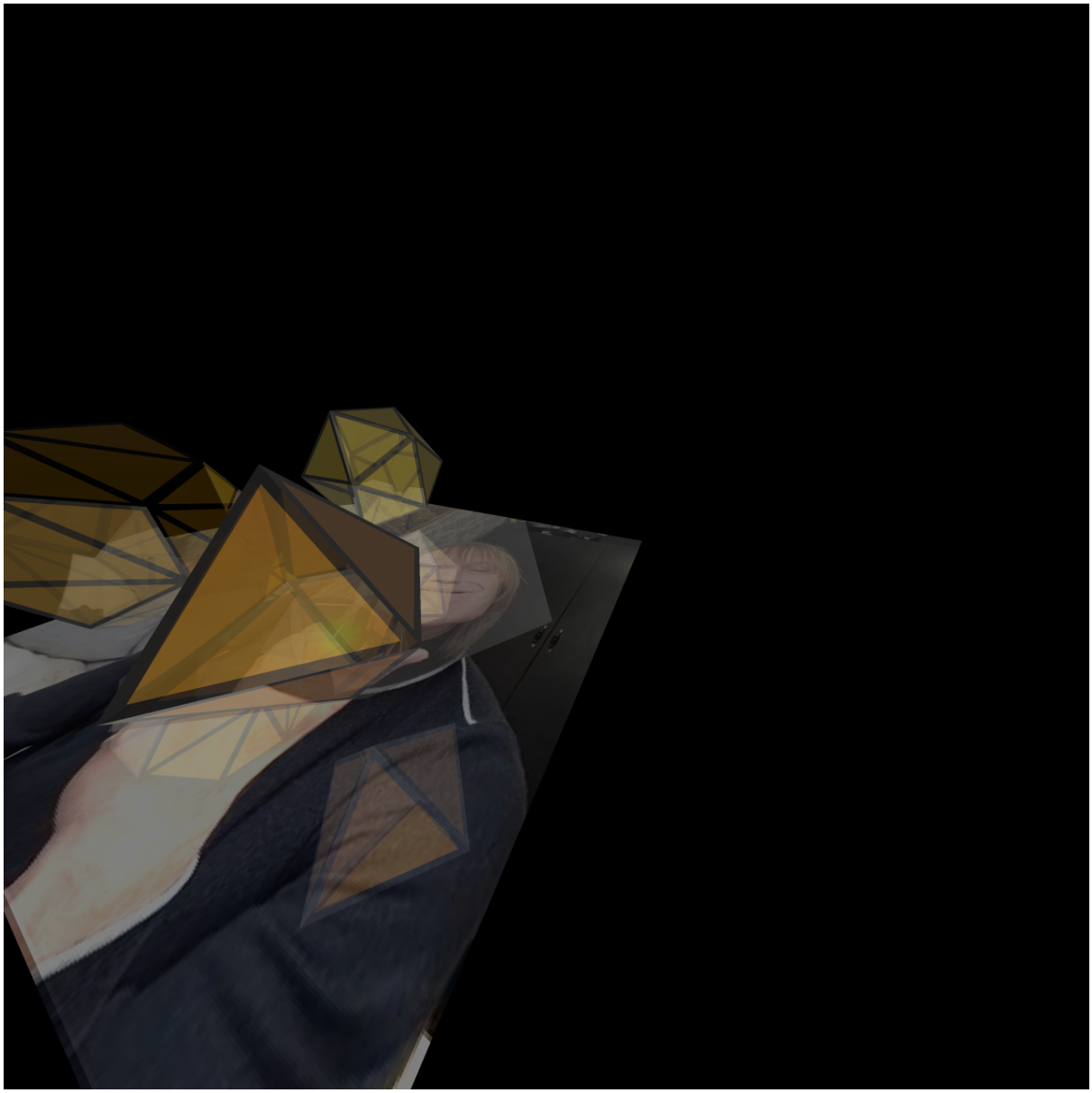
Входные данные, на которых получается наиболее красочный результат:

```
300
floor.data
frames/frame%d.out
1280 720 120
6.0 5.0 0.0 2.0 1.0 0.05 0.1 0.05 0.0 0.0
3.0 0.0 3.1415926 1.5 0.5 0.03 0.07 0.02 0.0 0.0
-2.5 2.5 2 0.9843 0.4745 0.0156 2.25 0.1 0.6
2.5 2.5 2 0.49 0.3216 0.0078 1.732 0.1 0.6
2.5 -2.5 2 0.949 0.773 0.05 2.118 0.1 0.6
-5 -5 0
5 -5 0
5 5 0
-5 5 0
2
-4 4 3 1 1 1
5 0 4 1 1 1
4 3
```

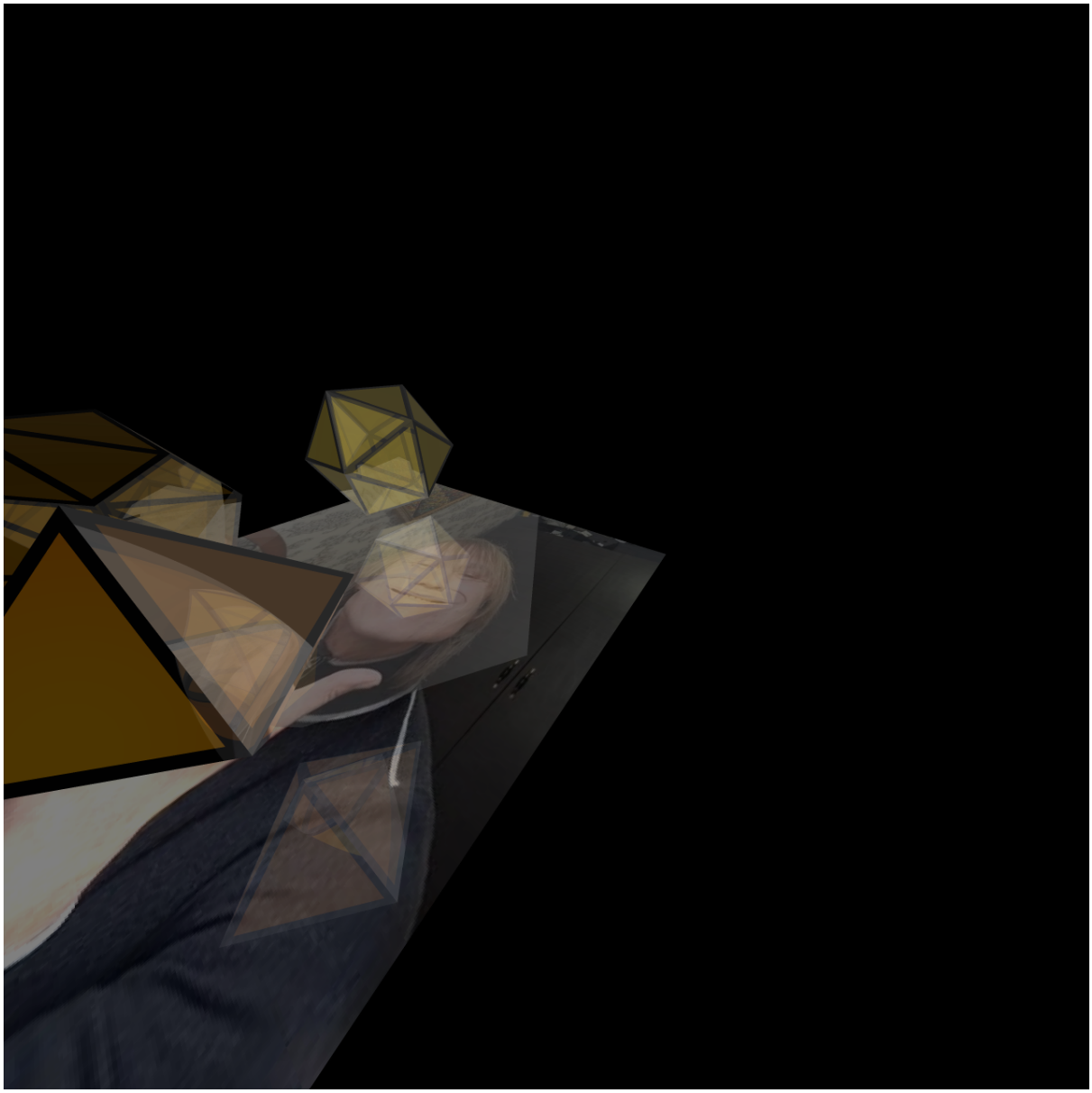
Результаты



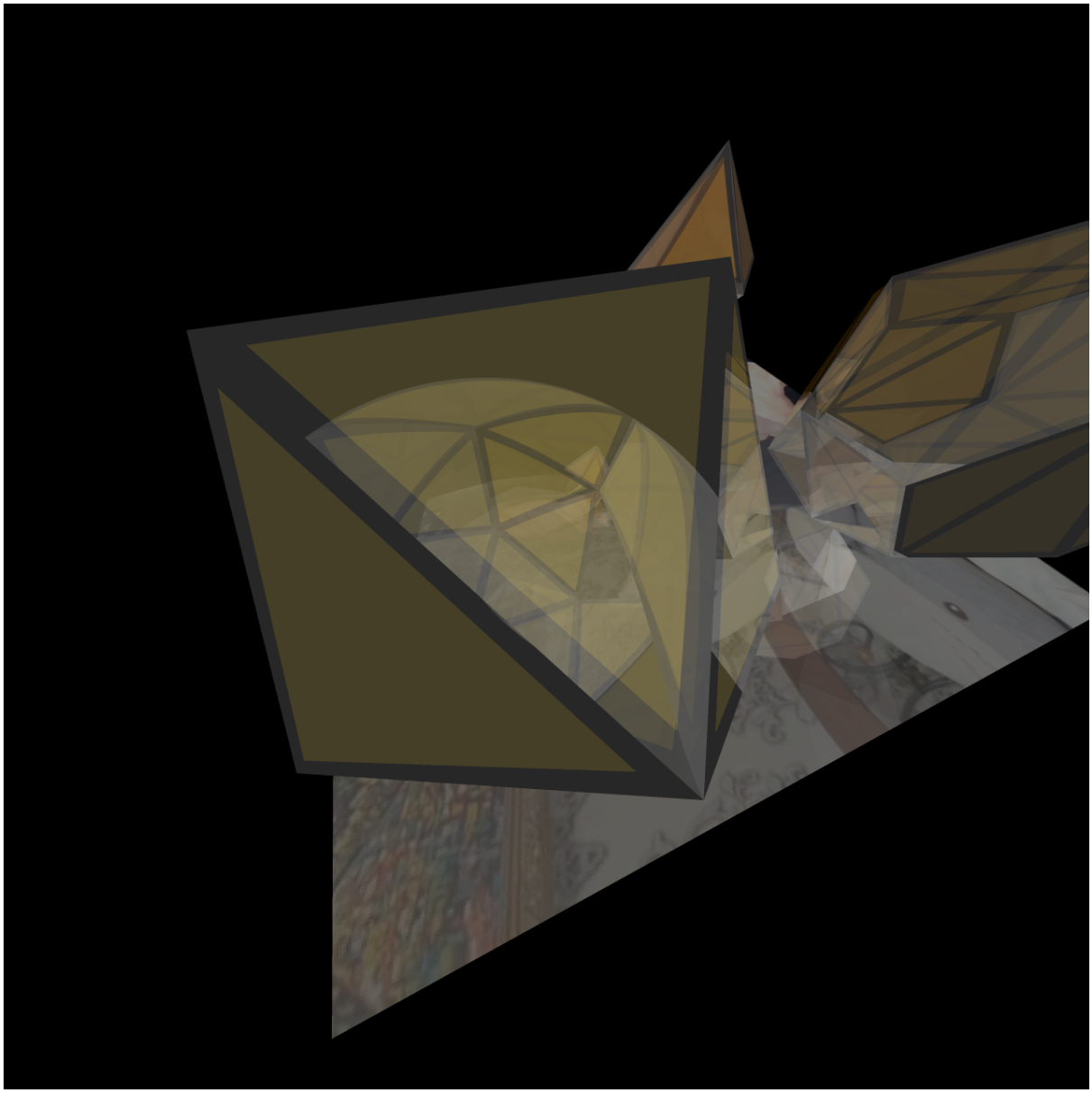
Отражения, преломления и тени на фигурах



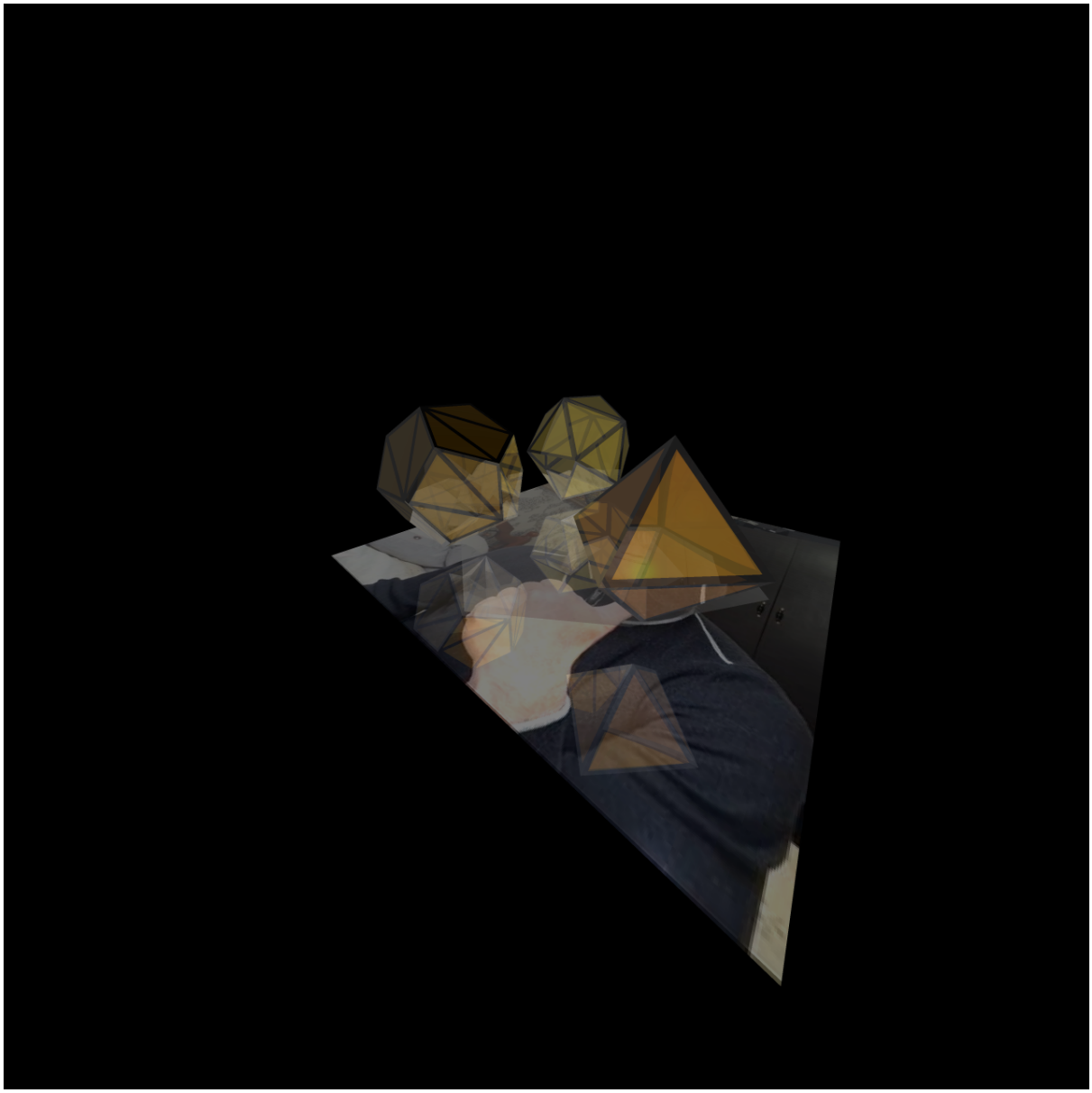
Отражение октаэдра от пола



Икосаэдр отражается в октаэдре, тень падает на октаэдр



Эффект бесконечности в икосаэдре



Все 3 фигуры

Выводы

В ходе работы был разработан высокоэффективный трассировщик лучей с использованием технологий CUDA для ускорения вычислений.

Выполнение данной работы существенно усложнил факт неподдержки абстрактных классов на GPU, невозможность создания двумерного массива и прочая работа с памятью. Отсутствие рекурсии в классическом понимании также вызвало ряд проблем. Однако использование шаблонной рекурсии отчасти их решило. Этот подход существенно увеличил время компиляции, но сократил время разработки и читаемость кода. Альтернативой было бы построение стековой машины.

В результате удалось получить гибкий интерфейс для рендеринга сцены с возможностью выбора положения и размера фигур, текстуры пола, положения источников освещения, движение камеры. Были реализованы тени от объектов, отражение и преломление лучей, эффект бесконечности и прозрачности фигур. А с помощью алгоритма сглаживания удалось получить очень качественную картинку.

Замеры времени показали, что использование графического процессора позволило существенно ускорить рендеринг сцены и выполнение алгоритма антиалиасинга.

Список литературы

[1] Боресков А. В., Харламов А. А. *Основы работы с технологией CUDA*. — Москва: ДМК Пресс, 2010. — 368 с. ISBN: 978-5-94074-535-1.

[2] *Recursive Template Instantiation Exceeded Maximum Depth of 256 — Stack Overflow.* URL: https://stackoverflow.com/questions/22800245/recursive-template-instantiation-exceeded-maximum-depth-of-256 (дата обращения: 03.01.2025).

[3] *Ray Tracing — Трассировка лучей: основы и методы.* URL: http://www.ray-tracing.ru/articles202.html (дата обращения: 03.01.2025).

[4] *Трёхмерная графика с нуля. Часть 1: трассировка лучей — Habr.* URL: https://habr.com/ru/articles/342510/ (дата обращения: 03.01.2025).

[5] *Regular Dodecahedron — Wikipedia.* URL: https://translated.turbopages.org/proxy_u/en-ru.ru.dcb98f78-677bc25c-3d2d659c-74722d776562/https/en.wikipedia.org/wiki/Regular_dodecahedron (дата обращения: 03.01.2025).

[6] *Regular Icosahedron — Wikipedia.* URL: https://translated.turbopages.org/proxy_u/en-ru.ru.874f494b-677bc278-c38574e5-74722d776562/https/en.wikipedia.org/wiki/Regular_icosahedron (дата обращения: 03.01.2025).