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: А. Л. Ядров
Группа: 8О-408Б
Преподаватель: А. Ю. Морозов

Москва, 2024

Условие

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

Вариант: 4. Тетраэдр, Октаэдр, Додекаэдр

Программное и аппаратное обеспечение

Характеристики графического процессора:

- Наименование: NVIDIA GeForce RTX 4070 Ti SUPER
- Compute capability: 8.9
- Графическая память: 16375 Mb
- Разделяемая память на блок: 48 Kb
- Количество регистров на блок: 65536
- Максимальное количество потоков на блок: 1024
- Максимальное количество блоков: [1024, 1024, 64]
- Константная память: 64 Kb
- Количество мультипроцессоров: 66

Характеристики системы:

- Процессор: Intel(R) Core(TM) i7-14700KF
- Память: 65536 Mb

Программное обеспечение:

- ОС: Windows 11 (22H2)
- IDE: Microsoft Visual Studio 2022 (Comminity Edition)
- Компилятор: nvcc V12.2.140, MSVC V19.37.32824
- Версия CUDA: 12.3

Метод решения

В курсовой работе реализован алгоритм обратной трассировки лучей с фиксированным числом переотражений и без явного использования рекурсии. Основная единица исполнения трассировки лучей — трассировка одного луча (вычисление точки его пересечения со сценой), вычисления цвета в данной точке и создание отраженных или преломленных лучей. Исходные и новые лучи можно хранить в двух массивах и таким образом, одна итерация трассировки всех лучей соответствует одному рекурсивному спуску.

Для вычисления цвета в точке пересечения используется затенение по Фонгу, согласно которому он складывается из цветовых интенсивностей трех компонент освещения: фоновой, рассеянной и бликовой

$$I = I_a + I_d + I_s$$

Фоновая компонента:

$$I_a = K_a i_a$$

где K_a — способность материала воспринимать фоновое освещение, i_a — интенсивность фонового освещения.

Рассеянная компонента:

$$I_d = K_d \cos(\vec{L}, \vec{N}) i_d = K_d (\vec{L} \cdot \vec{N}) i_d$$

где K_d — способность материала воспринимать рассеянное освещение, i_a — интенсивность рассеянного освещения, $\vec{L}$ — направление из точки на источник света, $\vec{N}$ — нормаль в точке.

Бликовая (зеркальная) компонента:

$$I_s = K_s \cos^p(\vec{R}, \vec{V}) i_s = K_s (\vec{R} \cdot \vec{V})^p i_s$$

где K_s — способность материала воспринимать бликовое освещение, p — коэффициент блеска, i_a — интенсивность бликового освещения, $\vec{R}$ — направление отраженного луча, $\vec{V}$ — направление из точки на наблюдателя.

Для создания теней интенсивность источника света в точке корректируется с учетом коэффицента пропускания и цвета материалов, через которые проходит луч на пути к этой точке.

В качестве алгоритма сглаживания используется SSAA — изображение рендерится в N раз большем размере, итоговое изображение получается усреднением цвета в непересекающихся окнах $N \times N$.

Описание программы

Проект я разделил на следущие модули:

1. $common$ - содержит множество полезных дефайнов, структур данных для более приятной работы с $CUDA$ и $C++$
2. $Math$ - содержит классы векторов (математических), матриц и операций над ними
3. $Engine$ - содержит в себе класс объекта, сцены, камеры.
4. $Render$ - содержит в себе классы рендереров, отвечающих за непосредественно рендер сцены

Также для замера времени была написана программа на языке программирования $Python$, состоящая из слеудющих модулей:

1. $build$ используется для компиляции $Microsoft\ Visual\ Studio$ проекта (файла с расширением $.sln$).
2. $converter$ используется для конвертации изображения в бинарный вид и для получения изображения из бинарных данных.
3. $video$ используется для конвертации изображений в $.gif$
4. $main$ используется в качестве обертке над описанными выше модулями для более комфортного запуска модулей.

Результаты

Сравним результаты рендеринга одного и того же кадра с использованием алгоритма $SSAA$ и без его использования, а также кадры сгенерированные на CPU и на GPU

Примеры работы программы:

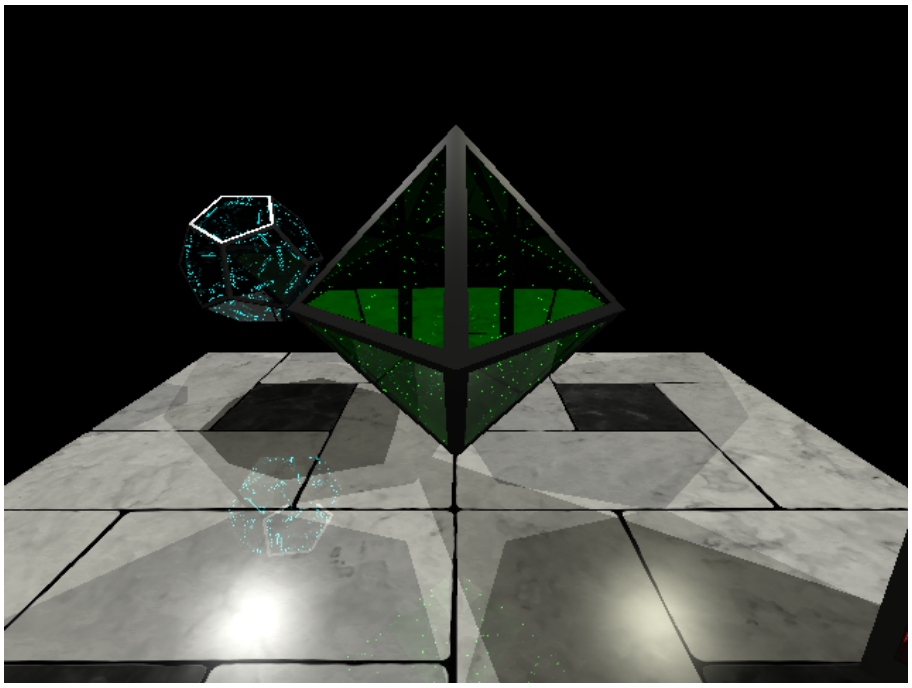


Рис. 1a: Кадр, отрендеренный на GPU без использования $SSAA$

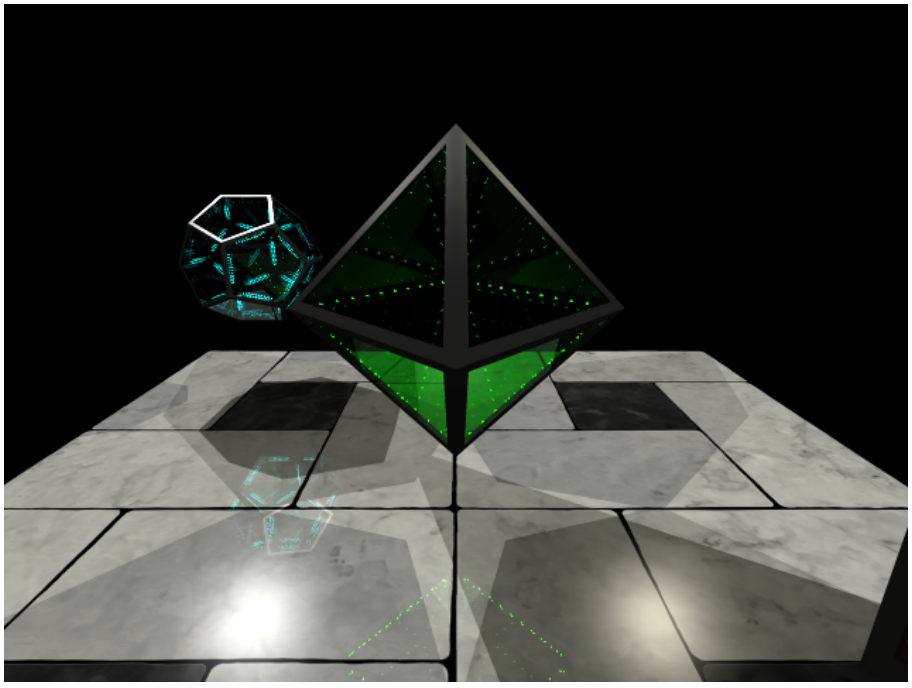


Рис. 1b: Кадр, отрендеренный на GPU с использованием $SSAA$

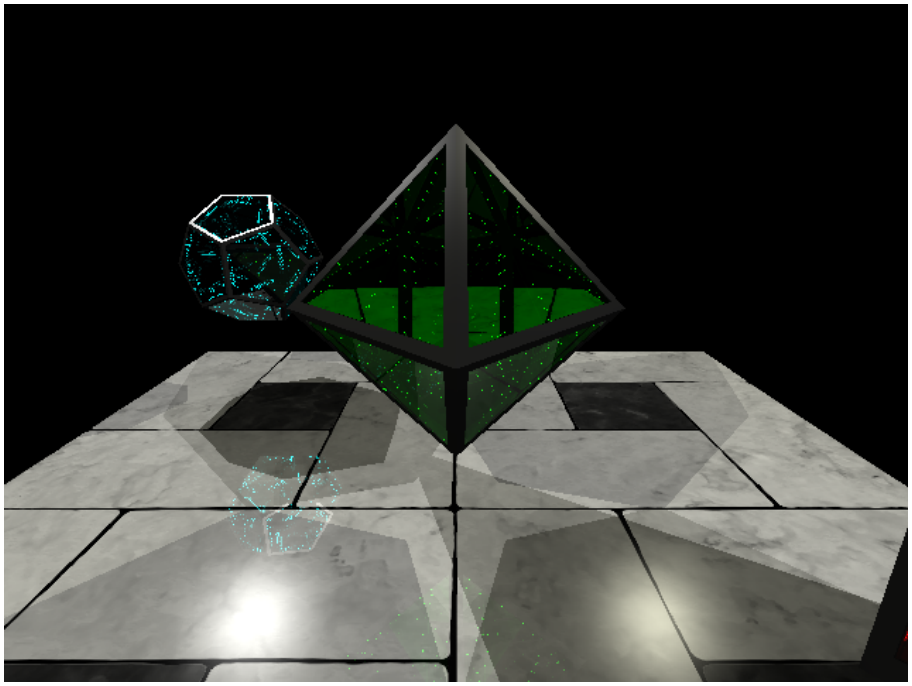


Рис. 2a: Кадр, отрендеренный на CPU без использования $SSAA$

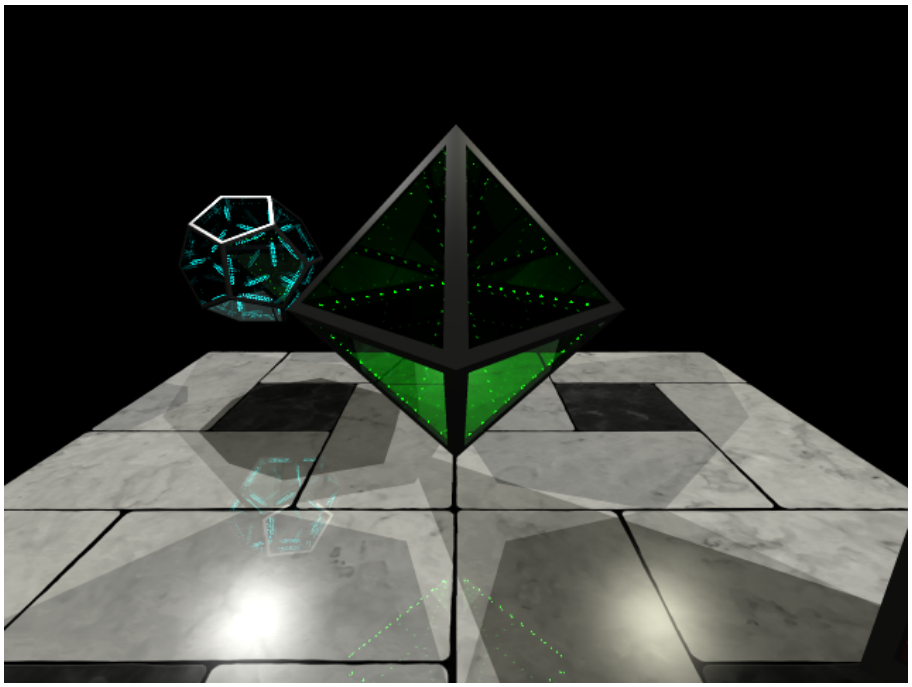
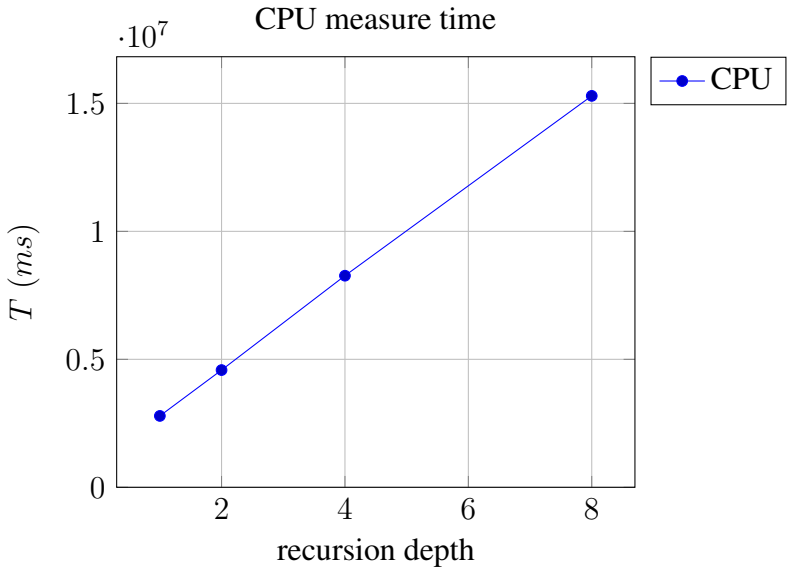


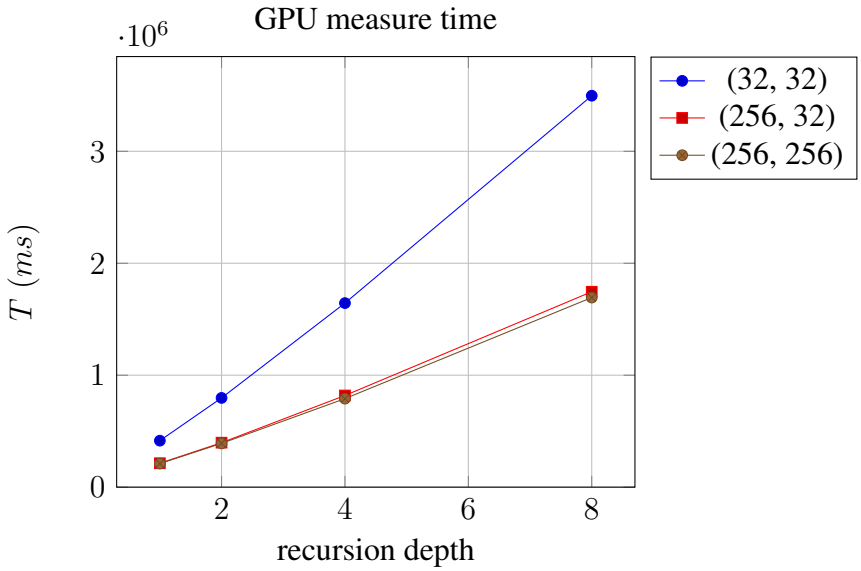
Рис. 2b: Кадр, отрендеренный на CPU с использованием $SSAA$

Невооруженным взглядом можно заметить плавность линий на изображении, построенном с помощью $SSAA$. Также видно, что точность вычислений на CPU и GPU примерно одинаковая, и мы получаем неотличимый результат.

Сравнение времени работы

Неудивительно, что несмотря на многопоточность, используемую в варианте с центральным процессором, вариант с графическим процессором на порядок выигрывает в скорости за счет большего количества потоков.


CPU measure time
$\cdot 10^7$
1.5
1
0.5
0
$T\ (ms)$
2
4
6
8
recursion depth
CPU


GPU measure time
$\cdot 10^6$
3
2
1
0
$T\ (ms)$
2
4
6
8
recursion depth
(32, 32)
(256, 32)
(256, 256)

Выводы

В ходе выполнения курсовой работы был реализован рейтресер, позволяющий производить реднеринг сцены с полузеркальными и полупрозрачными объектами. Также достигнут эффект "бесконечности"из-за множественных переотражений источников света на внутренних гранях платоновых тел.

Стоит отметить, что простая в реализации рекурсивная версия алгоритма позволяет получать довольно приятное глазу изображение. Плюсом алгоритма является хорошая распарллеливаемость, что можно увидеть, сравним показатели времени на GPU и CPU.

Список литературы

[1] *Ray-tracing.ru: 3d движок, трассировка лучей в реальном времени*
URL: `http://www.ray-tracing.ru/`
(дата обращения: 03.01.2024)

[2] *Простые модели освещения — Компьютерная графика*
URL: `https://grafika.me/node/344`
(дата обращения: 05.01.2024)