МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Факультет прикладной математики и физики
Кафедра вычислительной математики и программирования

Курсовой проект
по курсу «Параллельная обработка данных»

Обратная трассировка лучей (Ray Tracing) на GPU

Выполнил: Куликов А.В.
Группа: М8О-408Б-17
Преподаватели: К.Г. Крашенинников,
Ю. Морозов

Москва, 2020

Условие

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

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:

$$r_c(t)=r_c^0+A_c^r\sin\left(\omega_c^r\cdot t+p_c^r\right)$$
$$z_c(t)=z_c^0+A_c^z\sin\left(\omega_c^z\cdot t+p_c^z\right)$$
$$\varphi_c(t)=\varphi_c^0+\omega_c^\varphi\cdot t$$
$$r_n(t)=r_n^0+A_n^r\sin\left(\omega_n^r\cdot t+p_n^r\right)$$
$$z_n(t)=z_n^0+A_n^z\sin\left(\omega_n^z\cdot t+p_n^z\right)$$
$$\varphi_n(t)=\varphi_n^0+\omega_n^\varphi\cdot t$$

Требуется реализовать алгоритм обратной трассировки лучей с использованием технологии CUDA. Выполнить покадровый рендеринг сцены. Для устранения эффекта «зубчатости», выполнить сглаживание (например с помощью алгоритма SSAA). Полученный набор кадров склеить в видеоролик 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).

Согласно варианту 1 на сцене должны располагаться три тела: тетраэдр, тексаэдр, октаэдр.

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, и 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

Видеокарта	GeForce GTX 1650
Compute capability	7.5
Графическая память	3911 Мб
Разделяемая память	48 Кб
Константная память	64 Кб
Количество регистров на блок	65536
Максимальное кол-во блоков	2147483647*65535*65535
Максимальное кол-во нитей в блоке	1024
Кол-во мультипроцессоров	16
Ядер CUDA	896

Процессор	AMD Ryzen 5 3550H
ОЗУ	8 Гб
ЖД	512 Гб

Операционная система	Ubuntu 20.04.6 LTS
IDE	VS Code
Компилятор	nvcc V10.1

Метод решения

Трассировка лучей — это метод построения трёхмерных моделей, в котором используется принцип, аналогичный реальным физическим процессам. То есть для построения объекта система отслеживает траекторию виртуального луча от экрана к этому самому объекту.

Суть метода довольно проста. Для построения изображения из точки наблюдения через центры каждого пикселя в сцену испускается луч. Далее ищется объект, с которым этот луч пересекается. Если такой объект существует, то определяется точка пересечения. Для точки пересечения рассчитывается одна основной цвет по одной из моделей освещения (в моем случае моделей Фонга). Далее, если пересекаемый объект обладает отражающей способностью, из точки пересечения в направлении отражения испускается точно такой же луч. Также, если объект прозрачен и обладает преломляющей способностью, испускается преломленный луч. Испускание лучей продолжается до достижения максимальной глубины, задаваемой пользователем.

Цветом луча считается взвешенная сумма основного цвета, и цветов отраженных и преломленных лучей.

$$I = k_a i_a + (1 - k_t - k_r)\frac{i_l}{d+K}\left(k_d(\vec{L}\cdot\vec{N}) + k_s\cos(\vec{V}\cdot\vec{R})^p\right) + k_t I_t + k_r(I_r \circ k_a),$$

где i_a— мощность фонового освещения, k_a— способность материала воспринимать фоновое освещение, k_d— способность материала воспринимать рассеянное освещение, k_s— способность материала воспринимать зеркальную составляющую освещения,k_t— прозрачность материала, k_r— отражающая способность материала, i_l — интенсивность точечного источника света, d — расстояние от источника до точки пересечения, K — произвольная постоянная, $\vec{L}$ — вектор направления из точки на источник света, $\vec{N}$ — вектор нормали в точке пересечения, $\vec{S}$ — вектор наблюдения, $\vec{R}$ —направление отраженного луча, I_t — цвет преломленного луча, I_r — цвет отраженного луча, p — “блестящесть” объекта.

Поэлементное умножение $(I_r \circ k_a)$ добавлено для придания цвета прозрачным поверхностям.

За цвет пикселя принимается цвет первичного луча.

Пересечение луча с треугольником считается следующим образом.
Пусть

$R(t) = O + tD, t > 0$ — луч. Здесь O — позиция “источника” луча, D — направление луча.

$S(u,v) = (1-u-v)V_0 + uV_1 + vV_2, u \geq 0, v \geq 0, u+v \leq 1$ — поверхность треугольника.

Чтобы найти точку пересечения нужно решить уравнение

$$R(t) = S(u,v)$$
$$O + tD = (1-u-v)V_0 + uV_1 + vV_2$$
$$O + V_0 = u(V_1 - V_0) + v(V_2 - V_0) - tD$$

Т.о.

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \begin{bmatrix} -D & V_1 - V_0 & V_2 - V_0 \end{bmatrix}^{-1}(O - V_0)$$

Теперь, если для треугольника и луча выполняется условие $t > 0, u > 0, v > 0, u+v < 1$, то имеет место пересечение. Координату t удобно использовать при поиски ближайшего пересечения. А u, v — барицентрические координаты для данного треугольника. Они могут быть использованы для наложения текстур и интерполяции вектора нормалей в точке пересечения.

Природа распространения света со всеми ее отражениями и преломлениями рекурсивна. Именно поэтому алгоритм довольно прост в реализации рекурсивным способом. Для использования же GPU желательно иметь алгоритм, рассчитывающий то же самое итеративно. Поэтому было произведено разворачивание рекурсии.

Рекурсия эмулируется с использованием явного сохранения контекста на стеке. Контекст составляют все данные, необходимые для испускания луча: источник, направление, текущий цвет и т.д. Также для определения следующего этапа обработки сохраняется переменная состояния (аналог счетчика команд), благодаря которой не происходит повторного выполнения этапа.

Для устранения эффекта “лесенки” применен SSAA.

Описание программы

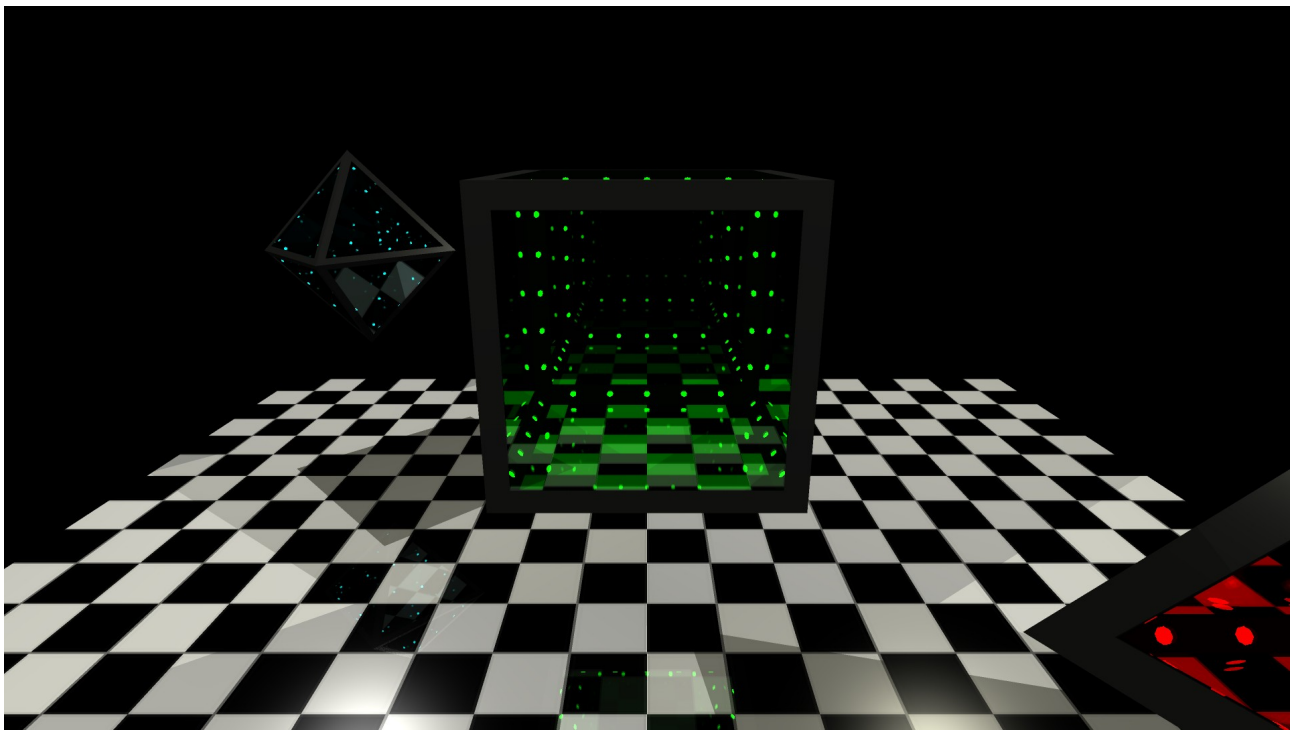
triangle_intersection* – функции для проверки пересечения луча с треугольником.
shadow_ray_hit* – функции для «испускания» теневых лучей.
phong_model* – функции для рассчета освещения по модели Фонга.
hit* – функции для проверки пересечения луча с объектами.

cast_ray* – функции для испускания основных лучей.
render_kernel – ядро для построения изображения на GPU.

Camera – класс, реализующий функциональность камеры. Его методы render и render_gpu позволяют построить изображение 3D сцены методом рейтрейсинга.
Scene – класс, содержащий в себе всю информацию об объектах и источниках света на сцене, а так же берущий на себя операции по отправке сцены на GPU.
Класс Texture отвечает за нанесение текстуры на объекты.
Класс Object содержит всю информацию об объекте: вершины составляющие объект, нормали, текстурные координаты, треугольники, величины описывающие оптические свойства объекта.

Функция gen_lights отвечает за генерацию точечных источников света на гранях тел.

Результаты



Выполнение	Время (мс)
CPU	956032
GPU 16*16 блока по 16*16 потоков	13608
GPU 32*32 блока по 16*16 потоков	13044
GPU 64*64 блока по 16*16 потоков	12768

Видно, что реализация с использованием GPU превосходит реализацию на CPU почти на 2 порядка.

Выводы

В ходе выполнения курсовой работы был реализован рейтрейсер позволяющий производить рендеринг сцены с полузеркальными и полупрозрачными объектами. Также достигнут эффект “бесконечности” из-за множественных переотражений источников света на внутренних гранях платоновых тел.
Стоит отметить, что рекурсивная версия данного алгоритма довольно проста в реализации, но при этом позволяет получать довольно приятное качество изображения. Несомненным плюсом данного алгоритма является хорошая распараллеливаемость, что можно увидеть в сравнении с CPU версией.
Более продвинутые версии данного алгоритма широко используются при создании компьютерной графики, визуализациях, играх.

Литература

1. www.ray-tracing.ru
2. www.scratchapixel.com
3. Быстрое нахождение пересечения луча и треугольника - Thömas Moller, Ben Trumbore