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: М.С.Гаврилов

Группа: 8О-406Б

Преподаватели: К.Г. Крашенинников,

А.Ю. Морозов

Москва, 2022

**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 | |
| Compute capability: | 7.5 |
| Name: | NVIDIA GeForce GTX 1650 |
| Total Global Memory: | 4102029312 |
| Shared memory per block: | 49152 |
| Registers per block: | 65536 |
| Warp size: | 32 |
| Max threads per block: | (1024, 1024, 64) |
| Max block: | (2147483647, 65535, 65535) |
| Total constant memory: | 65536 |
| Multiprocessors count: | 14 |

Процессор: Intel(R) Core(TM) i5-11260H @ 2.60GHz

Оперативная память: 7875 Мб

Накопитель: kimtigo SSD 256GB

OS: Linux Mint 21

IDE: Visual Studio Code 1.72.0

compiler: Cuda compilation tools, release 11.8, V11.8.89

**Метод решения**

Сцена состоит из трех фигур и пола. Текстура накладывается только на пол. Сцена освещается точечным источником света. Помимо этого, присутствует фоновое освещение. Тела могут отражать свет, рассеивать свет и пропускать свет через себя. Цвет каждого пикселя определяется путем испускания из точки обзора виртуальных лучей. Если луч, выпущенный из камеры пересекается с полигоном, то из точки пересечения испускается отраженный и преломленный луч. Также определяется, освещена ли точка пересечения источником света. Цвет, определенный лучом, является суммой цвета материала полигона (рассеянного полигоном света источника и фонового освещения) и цветов, полученных вторичными лучами.

**Описание программы**

Программа состоит из одного файла, в котором реализован класс Scene, описывающий сцену.

**Класс Scene**

|  |  |
| --- | --- |
| Метод | Описание |
| Scene**(**bool gpu\_available\_**,** std**::**string floor\_path**,** float floor\_refl\_ind**,** float3 light\_location**,** float3 light\_color**)** | Конструктор. При создании сцены определяет, использовать ли для рендера gpu, загружается текстура пола и создается источник света. После создания, сцена состоит из пола и источника света. |
| **~**Mesh **()** | Деструктор, освобождает все занятые ресурсы. |
| void add\_tetraeder**(**float3 location**,** float radius**,** material mat**)** | Добавляет в сцену тетраэдер с центром в координатах location радиусом описанной сферы radius и материалом полигонов mat |
| void add\_octaeder**(**float3 location**,** float radius**,** material mat**)** | Добавляет в сцену октаэдер с центром в координатах location радиусом описанной сферы radius и материалом полигонов mat |
| void add\_dodecaeder**(**float3 location**,** float radius**,** material mat**)** | Добавляет в сцену додекаэдер с центром в координатах location радиусом описанной сферы radius и материалом полигонов mat |
| void set\_render\_params(int frame\_width, int frame\_heighth, int frame\_angle, int sqrt\_rays\_per\_pix, int max\_depth) | Задает параметры рендера, общие для всех кадров, инициализирует массивы для хранения кадров и ресурсы gpu, если вычисления на gpu предусмотрены. |
| void render\_frame\_gpu**(**float3 camera\_location**,** float3 camera\_direction**)** | Рендерит кадр на gpu. В качестве параметров принимает позицию камеры и направление взгляда. Кадр рендерится с разрешением большим, чем требуется. Готовый кадр подвергается сглаживанию алгоритмом ssaa и записывается в массив для хранения. |
| void render\_frame\_cpu**(**float3 camera\_location**,** float3 camera\_direction**)** | Рендерит кадр на cpu. В качестве параметров принимает позицию камеры и направление взгляда.  Кадр рендерится с разрешением большим, чем требуется. Готовый кадр подвергается сглаживанию алгоритмом ssaa и записывается в массив для хранения. |
| void print\_frame\_to\_file**(**std**::**string path\_to\_file**,** int frame\_id**)** | Записывает содержимое массива для хранения кадра в файл по адресу path\_to\_file |

**Ядра**

|  |  |
| --- | --- |
| \_\_global\_\_ void kernel\_render\_frame**(**float3 view\_pnt**,** float3 view\_dir**,** int frame\_w**,** int frame\_h**,** double view\_angle**,** uchar4 **\***result**,** uchar4 **\***floor**,** int floor\_size**,** int floor\_texture\_size**,** triangle**\*** triangles**,** int triangle\_num**,** Light light**,** int max\_depth**)** | Для каждого пикселя вычисляет направление луча и рассчитывает сам луч. Готовый кадр записывается в массив result Ядро запускается для большего разрешения, чем задано при запуске программы. Излишнее разрешения нужно для применения ssaa. |
| \_\_global\_\_ void kernel\_ssaa**(**uchar4**\*** data**,** uchar4**\*** result**,** int frame\_w**,** int frame\_h**,** int sqrt\_rays\_per\_pix**)** | Реализует алгоритм ssaa. |

**Исследовательская часть и результаты**

1. Сравнение времени работы программы с различными конфигурациями.

20 кадров, разрешение 100 х 50 (5000px)

|  |  |  |
| --- | --- | --- |
| Размерность сетки | Время работы (мс) | Сред. время на кадр (мс) |
| <<<(4,4),(4,4)>>> | 587.494 | 29.3947 |
| <<<(8,8),(8,8)>>> | 144.127 | 13.7257 |
| <<<(16,16),(16,16)>>> | 132.075 | 12.5563 |

20 кадров, разрешение 600 х 400 (240000px)

|  |  |  |
| --- | --- | --- |
| Размерность сетки | Время работы (мс) | Сред. время на кадр (мс) |
| <<<(4,4),(4,4)>>> | 18842.3 | 942.611 |
| <<<(8,8),(8,8)>>> | 5021.07 | 277.441 |
| <<<(16,16),(16,16)>>> | 4175.11 | 219.961 |

20 кадров, разрешение 1200 х 900 (1080000px)

|  |  |  |
| --- | --- | --- |
| Размерность сетки | Время работы (мс) | Сред. время на кадр (мс) |
| <<<(4,4),(4,4)>>> | 74875.0 | 3749.75 |
| <<<(8,8),(8,8)>>> | 42932.9 | 2146.64 |
| <<<(16,16),(16,16)>>> | 34489.4 | 1763.24 |

Результаты в виде графика:

|  |
| --- |
| C:\Users\max\AppData\Local\Microsoft\Windows\INetCache\Content.Word\figure_1.png |

1. Сравнение времени работы при разной предельной глубине луча.

10 кадров, разрешение 600 х 400, Размерность сетки <<<(16,16),(16,16)>>>

|  |  |  |
| --- | --- | --- |
| Макс. Глубина | Время работы (мс) | Сред. время на кадр (мс) |
| 0 | 1288.02 | 128.802 |
| 1 | 1894.9 | 189.49 |
| 2 | 2086.63 | 208.663 |

Результаты в виде графика:

|  |
| --- |
| C:\Users\max\AppData\Local\Microsoft\Windows\INetCache\Content.Word\figure_4.png |

1. Сравнение времени работы при различном числе лучей на пиксель.

10 кадров, разрешение 600 х 400, Размерность сетки <<<(16,16),(16,16)>>>

|  |  |  |
| --- | --- | --- |
| Лучей на пиксель | Время работы (мс) | Сред. время на кадр (мс) |
|  | 158.222 | 15.8222 |
|  | 546.895 | 54.6895 |
|  | 2087.81 | 208.781 |

Результаты в виде графика:

|  |
| --- |
| C:\Users\max\AppData\Local\Microsoft\Windows\INetCache\Content.Word\figure_5.png |

1. Сравнение времени работы на gpu и cpu.

10 кадров, разрешение 100 х 50

|  |  |  |
| --- | --- | --- |
|  | Время работы (мс) | Сред. время на кадр (мс) |
| GPU (<<<(16,16),(16,16)>>>) | 65.2247 | 6.51431 |
| CPU | 29 983 | 2 998.3 |

10 кадров, разрешение 200 х 100

|  |  |  |
| --- | --- | --- |
|  | Время работы (мс) | Сред. время на кадр (мс) |
| GPU (<<<(16,16),(16,16)>>>) | 220.743 | 22.0743 |
| CPU | 112 005 | 11 200.5 |

10 кадров, разрешение 400 х 200

|  |  |  |
| --- | --- | --- |
|  | Время работы (мс) | Сред. время на кадр (мс) |
| GPU (<<<(16,16),(16,16)>>>) | 802.439 | 80.2439 |
| CPU | 452 332 | 45 233.2 |

Результаты в виде графика:

|  |
| --- |
| C:\Users\max\AppData\Local\Microsoft\Windows\INetCache\Content.Word\figure_3.png |

1. Примеры работы программы

10 кадров, полученных при следующих входных данных:

|  |
| --- |
| 10  res/%d.data  600 400 120  7.0 3.0 0.0 2.0 1.0  2.0 6.0 1.0 0.0 0.0  2.0 0.0 0.0 0.5 0.1  1.0 4.0 1.0 0.0 0.0  3 1 2 1 0 1 1 0.5 0.2 0  0 0 1.5 1 0 0 1 0.5 0.9 0  -1 -3 2 0 0 1 1 0.5 0.3 0  -5 -5 0 -5 5 0 5 5 0 5 -5 0  floor\_2.data  1 1 1 0.2  1 7 -7 5 1 1 1  2 4 |

|  |
| --- |
|  |

**Вывод**

В ходе выполнения курсовой работы я ознакомился с алгоритмом обратной трассировки лучей и реализовал программу, осуществляющую рендеринг полупрозрачных и полузеркальных правильных геометрических тел с его помощью. Я получил опыт в создании изображений и анимации при помощи gpu.

Алгоритм обратной трассировки лучей используется для получения фотореалистичных изображений, преимущественно в ситуациях, когда нет необходимости рендерить сцену в реальном времени, ввиду высокой трудоемкости вычислений. Так как этот алгоритм легко распараллелить, а на одном процессоре он работает крайне медленно, он почти всегда исполняется на gpu.

Основные трудности, с которыми я столкнулся в ходе создания программы заключались в необходимости задать точки полигонов геометрических фигур в таком порядке, чтобы нормали указывали наружу.

Полученные в ходе исследования производительности данные показывают, что этот алгоритм гораздо более эффективно работает распараллелено на gpu, чем на cpu. В среднем наблюдался прирост производительности в 500 раз. Производительность также растет с увеличением размерности сетки. Прочие исследования показали, что программа работает тем быстрее, чем меньше лучей необходимо обработать.

**Литература**

1. Статья «Обратная трассировка лучей» [интернет-ресурс] URL: <http://www.ray-tracing.ru/articles164.html>
2. Статься «Ray tracing (graphics)» [интернет-ресурс] URL: <https://en.wikipedia.org/wiki/Ray_tracing_(graphics)>