Муниципальное автономное образовательное учреждение

«Лицей № 38»

Трассировка лучей

Выполнил

обучающийся МАОУ “Лицей№38”

Головинский Павел

Нижний Новгород

2020 год

Оглавление

1. Введение….....2
2. Теоретическая часть…….....4
   2.1. Что такое трассировка лучей?.....4
   2.2. Подвиды RTX.....4
   2.3. Достоинства.....7
   2.4. Недостатки.....7
3. Практическая часть.....7
   3.1. Основные сущности программы……...7
   3.2. Метод RTX……..9
   3.3. Результаты работы……10
4. Заключение……...1
   3
   4.1. Возможные пути развития……...13
5. Список источников……..……14

1. ВВЕДЕНИЕ

С начала появления компьютерной графики программисты и художники задались вопросом реализации фотореалистичных изображений. Первый предложенный метод реализации был взят с реального мира, этот метод симулировал попадание светового луча на сетчатку глаза, однако реализация его в микросхемах того времени оказалась невозможно по причине того, что нужные расчёты требовали огромных вычислительных ресурсов, из-за чего был принят метода буфера глубины, который был проще в реализации в процессоре однако не выдавал фотореалистичного изображения, однако первый метод получил название “Метод трассировки лучей” и стал широко использоваться там, где не нужна была высокая скорость формирования изображений(30 кадров в секунду). В частности этот метод широко используется в кинематографии для создания спецэффектов.

С развитием вычислительных мощностей и появлением высокопроизводительных видеокарт о методе трассировки лучей в реальном времени (Real-Time Ray Tracing или RTX) снова вспомнили. Его рабочий прототип “Reflections” был представлен компанией Epic на конференции разработчиков компьютерных игр(GDC) в Сан-Франциско в 21 Марта 2018 года, однако даже он требовал высокопроизводительного суперкомпьютера. С того момента компании как Microsoft, nVidia, ATI и Intel начали разработку видеокарт, способных воспроизвести методологию RTX на домашнем компьютере. На данный момент существует только две видеокарты способные сделать это, однако их стоимость превышает 2000 usd (>128,000 rub).

Использование RTX на примере игры Minecraft.

В данном проекте была поставлена цель – создать модель метода RTX , изучить эту модель. Для этого были сформулированы следующие задачи исследования:

1. Ознакомиться с существующими методами и приёмами RayTracing`a;
2. Визуализировать упрощенную версию данного метода - RayCasting имитирующий трёхмерное пространство;
3. Научится основным приёмам индустриального программирования.

При проведении работы были использованы следующие методы:

1. Теоретический анализ литературы по данному вопросу;
2. Моделирование объекта исследования;
3. Анализ созданной модели;
4. Констатация выводов;

2. Теоретическая часть.

Одним из первых методов визуализации графического пространства является трассировка лучей. Благодаря ему можно получить одно из самых реалистичных изображений, доступное в данный момент. Он работает схоже тому, как мы видим объекты в реальной жизни.

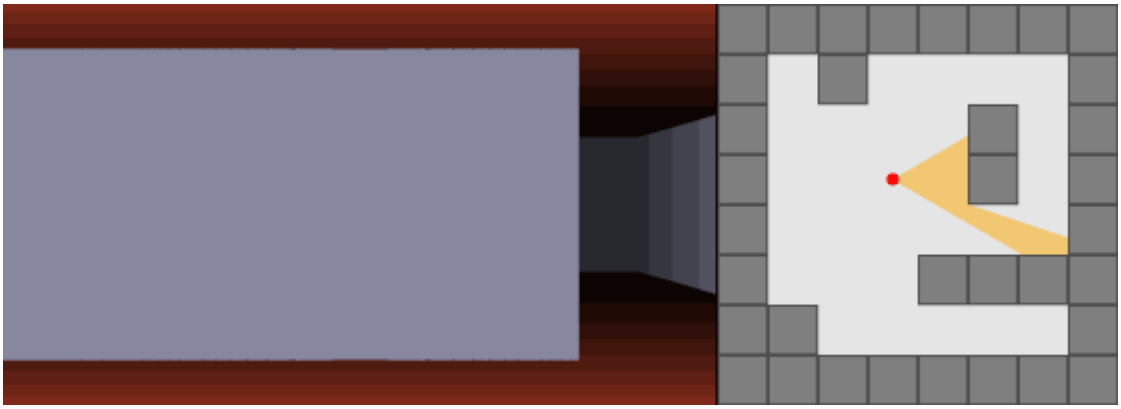
2.1. Что такое трассировка лучей.

Трассировка лучей представляет собой метод визуализации графического пространства, использующий камеру, как излучатель лучей (луч на пиксель соответственно). При попадании луча на объект, пиксель, соответствующий этому лучу окрашивается в цвет объекта.

2.2. Подвиды RTX.

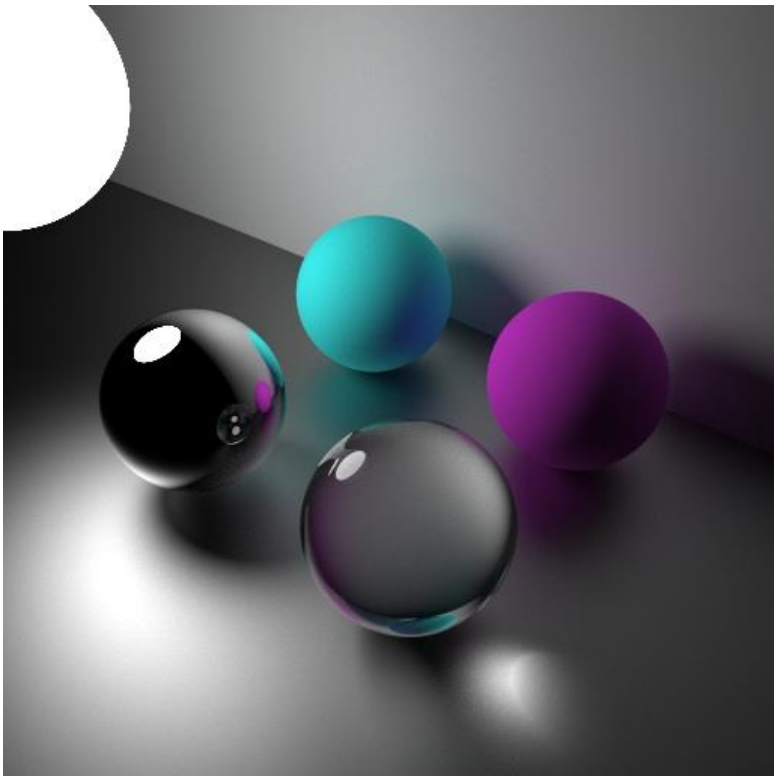
Ray Casting (бросание лучей) - один из первых и простейших методов трассировки лучей, ограничено просчитывающий отражения и не просчитывающий пропускную способность прозрачных поверхностей. Долгое время использовался в кинематографии. Наиболее известная реализация этого метода - визуализатор ScanLine в программе AutoDesk:3ds Max.

Целью данного проекта является создание упрощённой версии этого метода, когда луч отвечает не за пиксель, а за одну вертикальную полосу на экране, отражающем вид от первого лица.



Простейшая визуализация метода RayCasting.

Path Tracing (Трассировка пути) - Метод, схожий с трассировкой лучей, но луч отражаются до того момента, как полностью не погасится. Не используется в визуализации в реальном времени, так как требует огромных вычислительных мощностей, но используется с ограничениями в количестве переотражений луча в современном кинематографе для создания фотореалистичных изображений. Наиболее известные реализации этого метода - визуализаторы V-Ray, Mental Ray и подобные.



Пример использования технологии Path Tracing

2.3. Достоинства

Сложность вычислений не зависит от сложности сцены.

Не нужно просчитывать перспективу, правильное поле зрения и невидимые объекты.

Высокая фотореалистичность.

2.4. Недостатки

Низкая производительность по причине необходимости просчитывания всех пикселей не экране с каждым кадром.

Шумность получаемого изображения.

3. Практическая часть.

Для реализации данного метода была написана программа на языке C#, используя индустриальный стандарт программирования - Google Coding Guidelines. Для построения программной модели все сущности метода были разделены на классы.

3.1. Основные сущности программы (classes).

В программе были выделены пять основных компонентов - основное окно с областью для рисования (class Form1), Камера - объект, испускающий лучи (class Camera), Испускаемые лучи, каждый отвечающий за одну вертикальную линию формируемого изображения(class Ray), вид от первого лица - отдельный экран, показывающий результирующее изображение (class EyeSight) и стены - препятствия на пути луча (class Wall).

Объект Form1 создан системой автоматически и является основным окном программы.

Камера

```
class Camera //Излучатель
{
    public float m_x = 500; //Первая координата
    public float m_y = 480; //Вторая координата
    private float m_angle = 0; // Угол поворота излучения лучей
    public int m_n_rays = 20; // Количество лучей
    public float FOV = 3.14f/4; // Поле зрения
    private float m_power; // Начальная яркость луча
    private float m_loss; // Потери яркоти луча с расстоянием от камеры
```

Свойства класса Camera

Луч

```
class Ray //Луч
{
    public Camera m_camera; // Камера из которой исходит луч
    public float m_direction_x = 0; // Первая координата направления конца луча
    public float m_direction_y = 0; // Вторая координата направления конца луча
    public float m_ray_length = 0; //Длина луча
```

Свойства класса Ray

Стена

```
class Wall // Стена
{
    public float m_x1; // X первой точки стены
    public float m_x2; // Y первой точки стены
    public float m_y1; // X второй точки стены
    public float m_y2; // Y второй точки стены
```

Свойства класса Wall

Поле Зрения

```
class EyeSight // поле зрения
{
    public float m_ray_width; // Ширина сегмента
    public float m_color_depth; // Максимально возможная яркость
```

Свойства класса EyeSight

3.2. Метод RTX

Основой метода RTX является поиск и определение точки пересечения луча с препятствием. Для поиска точки пересечения использовалась следующая техника: Луч и препятствие представлялись в виде двух отрезков прямых, которые можно представить формулой:

$$y = k*x + b;$$

Параметры k и b рассчитывались исходя из того, что прямые в программе задаются двумя точками (x1;y1):(x2;y2). Для этого было решена следующая система уравнений:

$$y1 = k*x1 + b$$

$$y2 = k*x2 + b;$$

Результатом решения системы этих уравнений является:

$$k = \frac{y2 - y1}{x2 - x1}$$

$$b = y1 - k * x1;$$

В программе поиск этих значений был реализован в методе CalculateLineParameters.

```
if (x2 != x1)
{
    line.K = (y2 - y1) / (x2 - x1);
    line.B = y1 - line.K * x1;
    line.Vertical = false;

    if (Math.Abs(y2 - line.K * x2 - line.B) > 0.1)
    {
        throw new InvalidProgramException();
    }
}
```

Реализация поиска параметров линии по двум точкам на плоскости

Основная задача RTX: поиск точки пересечения является результатом решения следующей системы уравнений:

$$y = k1 * x + b1$$

$$y = k2 * x + b2;$$

Результатом решения этой системы уравнений являются координаты точки пересечения двух прямых.

$$x = \frac{b2 - b1}{k1 - k2}$$

$$y = k1 * x + b$$

В программе данное уравнение было реализовано в функции CalculateIntersectionPoint.

```
intersection_point.X = (wall_param.B - ray.B) / (ray.K - wall_param.K);
intersection_point.Y = ray.K * intersection_point.X + ray.B;
```

Реализация поиска точки пересечения

В виде от первого лица каждая ширина каждой вертикальной полосы зависит от количества выпущенных лучей и рассчитывается по формуле.

$$Ширина\ полосы = \frac{Ширина\ экрана}{Общее\ количество\ лучей}$$

```
public EyeSight(int width, int n_rays, float power, float loss)
{
    m_color_depth = power / loss; // Максимально-возможная яркость полосы
    m_ray_width = width / n_rays; // Ширина сегмента
}
```

Расчёт свойств полосы(Яркость и ширина)

Эффекта псевдо трёхмерности удалось добится заполняя ячейки финального изображения не полностью, а используя нормированную длину луча.

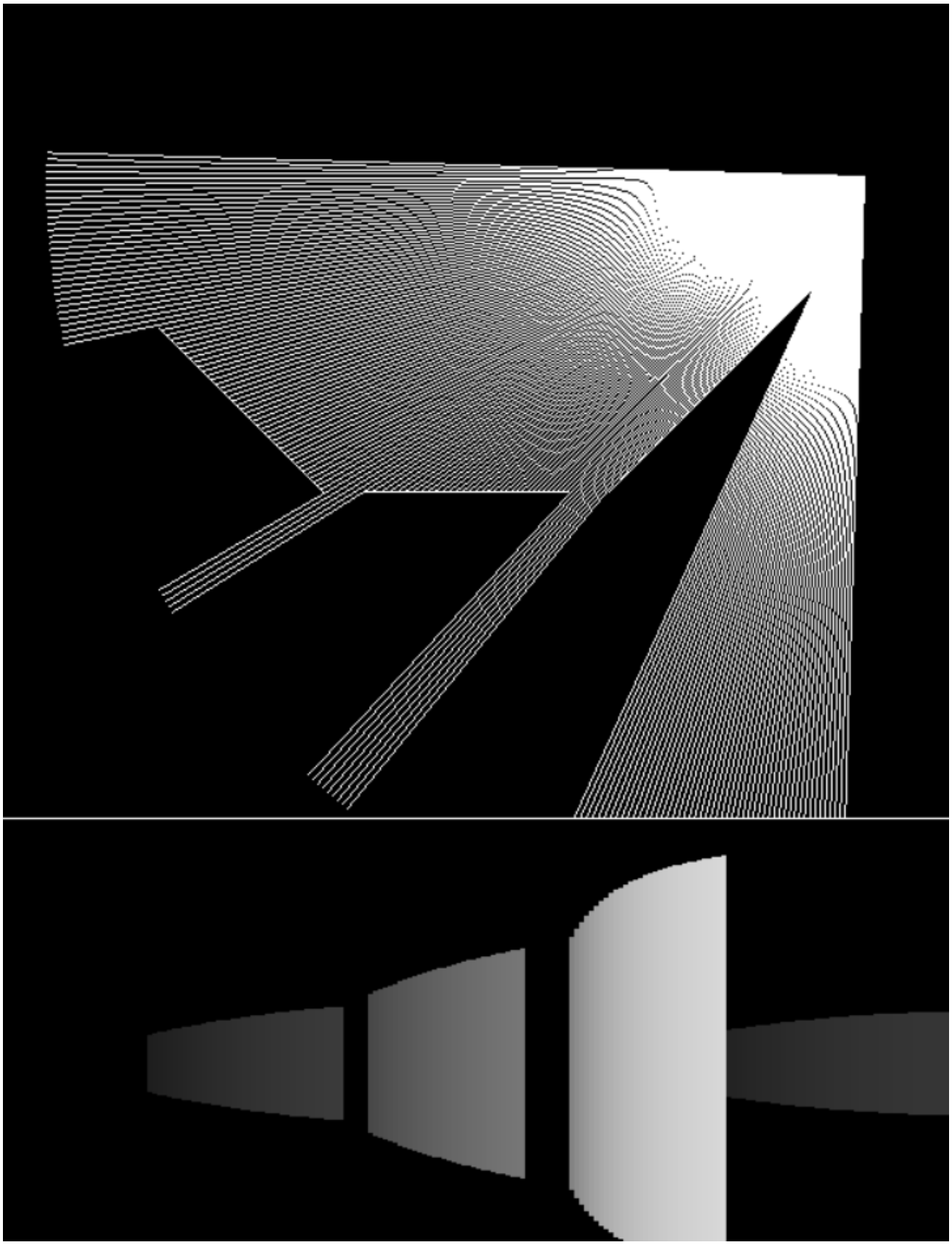
```
int color_power = (int)(255f * (1f - ray.GetRayLength() / m_color_depth)); // Выходной цвет стены
float wall_height = 1f - (ray.GetRayLength() / m_color_depth); // Выходная высота стены
```

Расчёт высоты и яркости сегментов финального изображения

3.3. Результаты работы

Была написана программа, реализующая метод RayCasting в псевдо-трёхмерном пространстве. Лучи испускались только в одной плоскости.
Экран был разделён на две части: Рабочую область и вид от первого лица.
Рабочая область представляет собой схематическое изображение, где есть возможность управлять положением камеры и направление излучения лучей; так-же присутствуют препятствия, необходимые для формирования финального изображения.

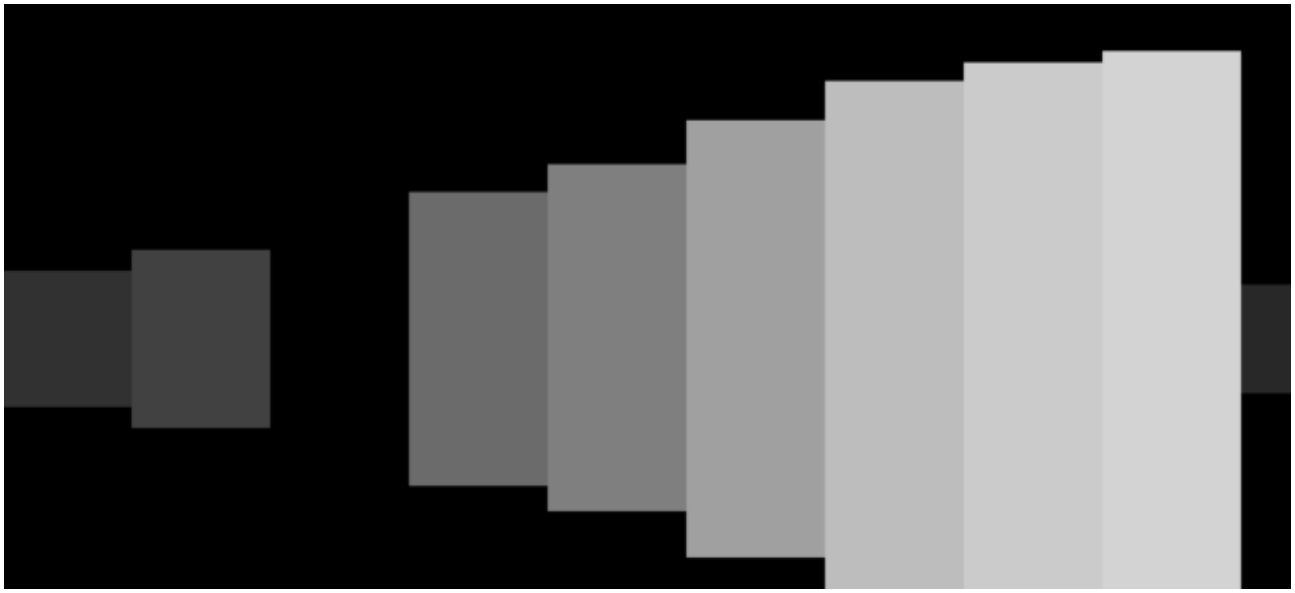
Вторая часть экрана - вид от первого лица в котором формируется финальное изображение.

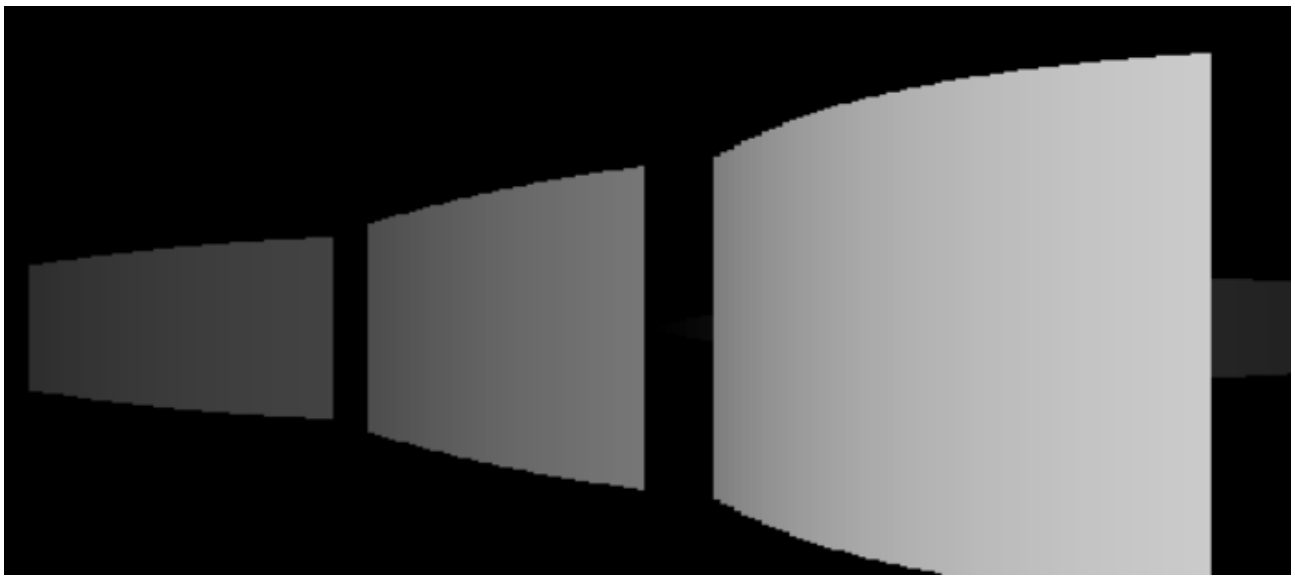


Результат работы программы(200 лучей)

На рабочей областей хорошо заметны выпущенные лучи и “тени” - области, находящиеся за прорисованными объектам; если в области тени есть какой-либо объект его не будет видно на финальном изображении.

В процессе работы программы было выяснено, что качество финального изображения зависит от количества лучей, испускаемых из камеры, однако при большом количестве лучей время, расходуемое на формирование финального изображения существенно увеличивается, что соответствует результатам современных тестов метода RTX.





1. *Выходное изображение, сформированное при выпускании 10 лучей*
2. *Выходное изображение, сформированное при выпускании 200 лучей*

Оба изображения были сформированы при одном и том-же положении камеры.

4. Заключение

В ходе работы были рассмотрены методы формирования изображения с помощью трассировки лучей. Была написана программа, реализующая метод RayCasting, являющаяся одним из подвидов метода RTX.

В ходе работы было установлено, что качество изображения, также как и скорость его формирования, напрямую зависит от количества испускаемых из камеры лучей - чем больше лучей, тем качественнее изображение, но больше времени тратиться на формирование изображения.

В ходе написания программы были получены навыки объектно-ориентированного программирования.

Также во время написания программы произошло ознакомление со стандартами индустриального программирования, принятыми для написания больших программ в современном программировании - Google Coding GuideLines.

Было расширено понимание языка программирования C#.

4.1. Возможные направления развития проекта:

1. Переход от псевдо-трёхмерности к настоящему трёхмерному изображению.
2. Формирование цветного изображения.
3. Получение отражений и преломлений.
4. Множественные переотражения луча.
5. Использование современного подхода RTX DXR (Real-Time RayTracing технология от Microsoft).

5. Список источников

1. GDC 2018: EPIC GAMES, NVIDIA, AND ILMXLAB UNVEIL STUNNING *STAR WARS* REAL-TIME RAY TRACING DEMO. https://www.starwars.com/news/gdc-2018-epic-games-nvidia-and-ilmxlab-unveil-stunning-star-wars-real-time-ray-tracing-demo.
2. Wikipedia - PathTracing https://en.wikipedia.org/wiki/Path_tracing.
3. Wikipedia - RayTracing https://en.wikipedia.org/wiki/Ray_tracing_(graphics).
4. NVIDIA RTX RayTracing https://developer.nvidia.com/rtx/raytracing.
5. CodingTrain - Coding Challenge #146: Rendering Raycasting https://www.youtube.com/watch?v=vYgIKn7iDH8&t=1116s.
6. CodingTrain - Coding Challenge #145: 2D RayCasting https://www.youtube.com/watch?v=TOEi6T2mtHo&t=156s.
7. Google C++ Style guidelines https://google.github.io/styleguide/cppguide.html