Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»
КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчёт

по лабораторной работе №4

Название: Трассировщик лучей с использованием параллельных вычислений

Дисциплина: Анализ алгоритмов

Студент ИУ7-55Б (Группа) _____ (Подпись, дата) Д.Р. Жигалкин (И.О. Фамилия)

Преподователь _____ (Подпись, дата) Л.Л. Волкова (И.О. Фамилия)

Москва, 2020

Содержание

Введение

Трассировка лучей (англ. Ray tracing; рейтрейсинг) — один из методов геометрической оптики — исследование оптических систем путём отслеживания взаимодействия отдельных лучей с поверхностями. В узком смысле — технология построения изображения трёхмерных моделей в компьютерных программах, при которых отслеживается обратная траектория распространения луча (от экрана к источнику).

Целью данной лабораторной работы является изучение и реализация параллельных вычислений для обратного трассировщика лучей.

Задачи данной лабораторной работы:

1) изучить алгоритм обратной трассировки лучей;

2) реализовать алгоритм обратной трассировки лучей;

3) провести замеры процессорного времени работы от разного числа параллельных потоков;

4) провести сравнение параллельных реализаций алгоритма обратной трассировки лучей со стандартной реализацией.

1 Аналитический раздел

В данном разделе будут рассмотрены основные теоритические понятия алгоритма обратной трассировки лучей.

1.1 Алгоритм обратной трассировки лучей

Методы трассировки лучей на сегодняшний день считаются наиболее мощными методами создания реалистических изображений. Универсальность методов трассировки в значительной степени обусловлена тем, что в их основе лежат простые и ясные понятия, отражающие наш опыт восприятия окружающего мира.

Метод обратной трассировки лучей позволяет значительно сократить перебор световых лучей по сравнению с классической трассировкой. В нем отслеживаются лучи не от источников, а из камеры. Таким образом, трассируется определенное число лучей, равное разрешению картинки.

Предположим, что у нас есть камера и экран, находящийся на расстоянии d от нее (рисунок 1.1).

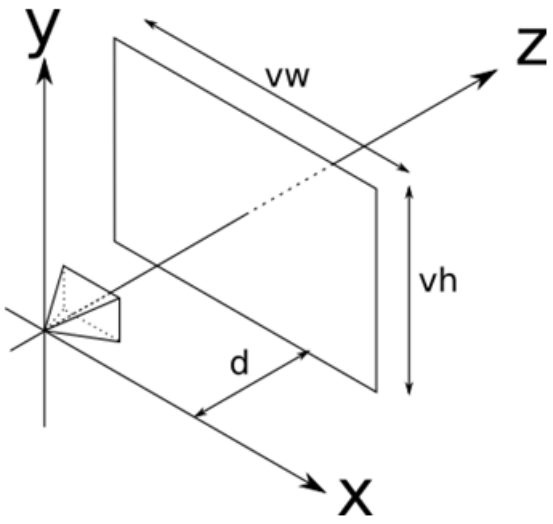

Рисунок 1.1 — Расположение камеры и сцены

Разобьем экран на квадратики. Дальше будем по очереди проводить лучи из камеры в центр каждого квадратика (первичные лучи). Найдем пересечение каждого такого луча с объектами сцены и выберем среди всех пересечений самое близкое к камере. Далее, применив нужную модель освещения, можно получить изображение сцены.

Но можно пойти дальше. Если мы хотим смоделировать такое явление, как отражение, нам необходимо из самого близкого пересечения пустить вторичные лучи. Например, если поверхность отражает свет и она идеально ровная, то необходимо отразить первичный луч от поверхности и пустить по этому направлению вторичный луч.

2 Констукторский раздел

В данном разделе будет рассмотрена схема алгоритма, требования к функциональности ПО, и опредены способы тестирования.

2.1 Разработка алгоритма

Ниже будет представлена схема алгоритма обратной трассировки лучей.

Алгоритм обратной трассировки лучей (рисунок 2.1);

2.2 Параллельные вычисления

Распараллеливание программы должно уменьшать время работы. Это достигается за счет реализации в узких участках (напимер в циклах с большим количеством независимых вычилений).

В предложенном алгоритме данным участком будет являться основной двойной цикл вычислений. Данный блок программы как раз предлагается распараллелить.

2.3 Требования к функциональности ПО

В данной работе требуется обеспечить следующую минимальную функциональность оконного приложения:

1) обеспечить ввод координат положения камеры наблюдателя;

2) обеспечить ввод значения угла, на который необходимо повернуть камеру наблюдателя относительно оси oY;

3) обеспечить вывод полученного с помощью алгоритма обратной трассировки лучей изображения;

4) обеспечить вывод замеров времени работы алгоритма обратной трассировки лучей.

2.4 Методы тестирования

Тестирование ПО будет проводиться методом черного ящика.

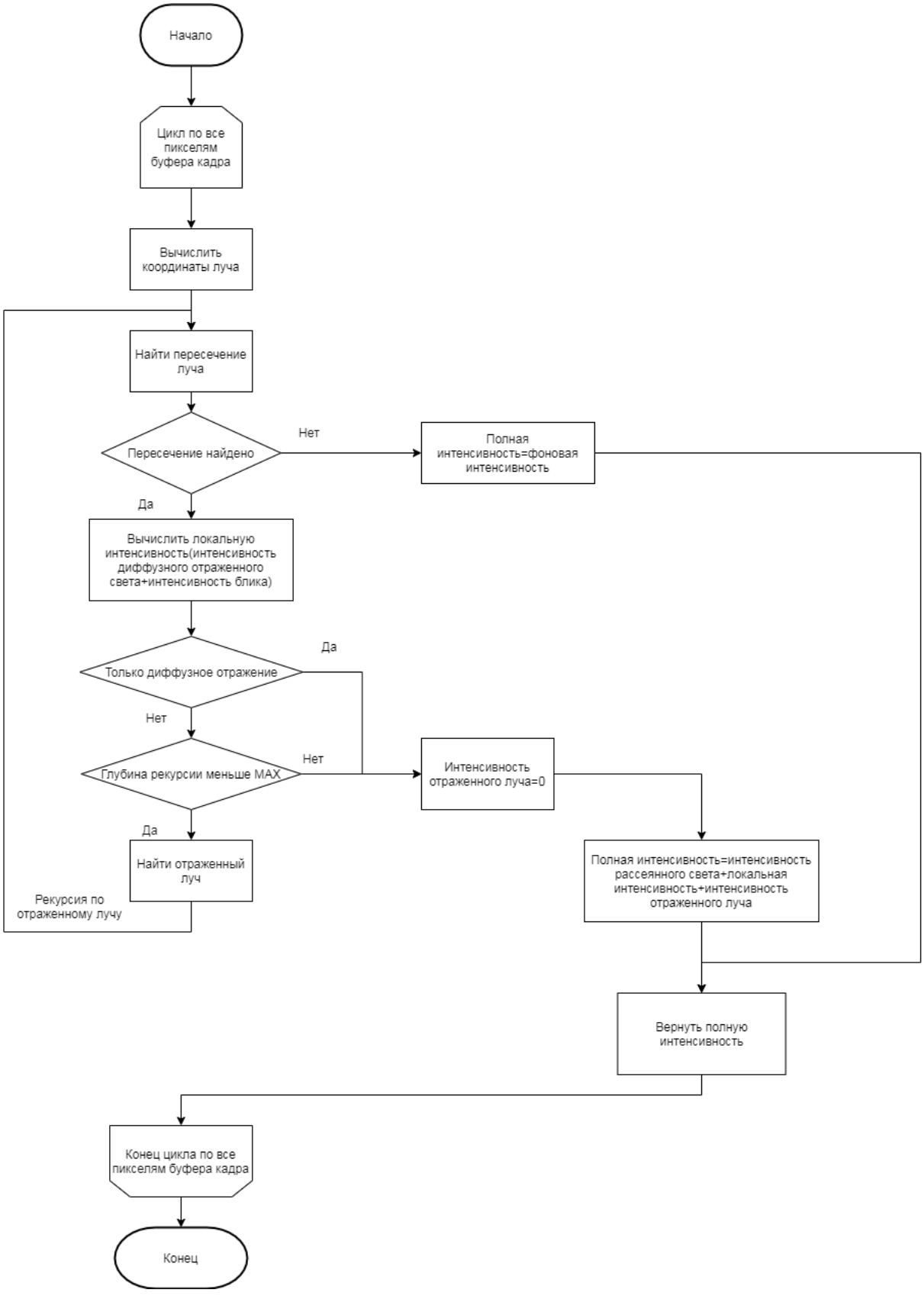

Рисунок 2.1 — Схема алгоритма обратной трассировки лучей

3 Технологический раздел

В данном разделе будут выбраны средства реализации ПО и представлен листинг кода.

3.1 Средства реализации

В данной работе используется язык программирования C# [1], так как он позволяет написать программу в относительно малый срок. В качестве среды разработки использовалась Visual Studio 2017 [2]. Для замера процессорного времени был использован класс Stopwatch [3].

Многопоточное программирование было реализовано с помощью пространства имен System.Threading [4].

Листинг 3.1 — Реализация алгоритма обратной трассировки лучей с параллельными вычислениями

```
            int n = 4; // Количество потоков

            int step = data.Width / n;

            Thread[] t = new Thread[n];

            int x1 = 0;
            int x2 = step;

            int y1 = 0;
            int y2 = data.Height;

            for (int i = 0; i < n; i++)
            {
                AllParameters p = new AllParameters(x1, y1, x2, y2);

                t[i] = new Thread(Process);
                t[i].Start(p);

                x1 = x2;
                x2 += step;
            }

            foreach (Thread thread in t)
            {
                thread.Join();
            }

            void Process(object obj)
            {
                AllParameters p = (AllParameters)obj;

```

```
33                  int x = p.x;
34                  int y = p.y;
35
36                  int endx = p.endx;
37                  int endy = p.endy;
38
39                  for (int i = x; i < endx; i++)
40                  {
41                      for (int j = y; j < endy; j++)
42                      {
43                          int[] work = { i - data.Height / 2, j - data.Width / 2 };
44
45                          double[] direction = RayTracing.CanvasToViewport(data.Width,
                                data.Height, work);
46                          direction = MyMath.MultiplyMV(cameraRotationOY, direction);
47
48                          double[] color = RayTracing.TraceRay(recursionDepth, lights,
                                objects, cameraPosition, direction, 1,
                                Double.PositiveInfinity, 0);
49
50                          var offset = ((j * data.Width) + i) * depth;
51
52                          buffer[offset + 0] = (byte)color[2];
53                          buffer[offset + 1] = (byte)color[1];
54                          buffer[offset + 2] = (byte)color[0];
55                      }
56                  }
57
58              }
```

4 Экспериментальный раздел

В данном разделе будут выполнены эксперименты для проведения сравнительного анализа алгоритмов обратной трассировки лучей по затрачиваемому процессорному времени в зависимости от числа потоков.

4.1 Сравнительный анализ на основе замеров времени работы алгоритмов

В рамках данного проекта был произведен замер времени работы алгоритма обратной трассировки лучей при количестве потоков равном 1, 2, 4, 8, 16, 32, 64. Результаты данных измерений приведены на графике 4.1.

Тестирование проводилось на ноутбуке с процессором Intel(R) Core(TM) i3-8130U CPU [5], 4 логических процессора, под управлением Windows 10 с 8 Гб оперативной памяти.

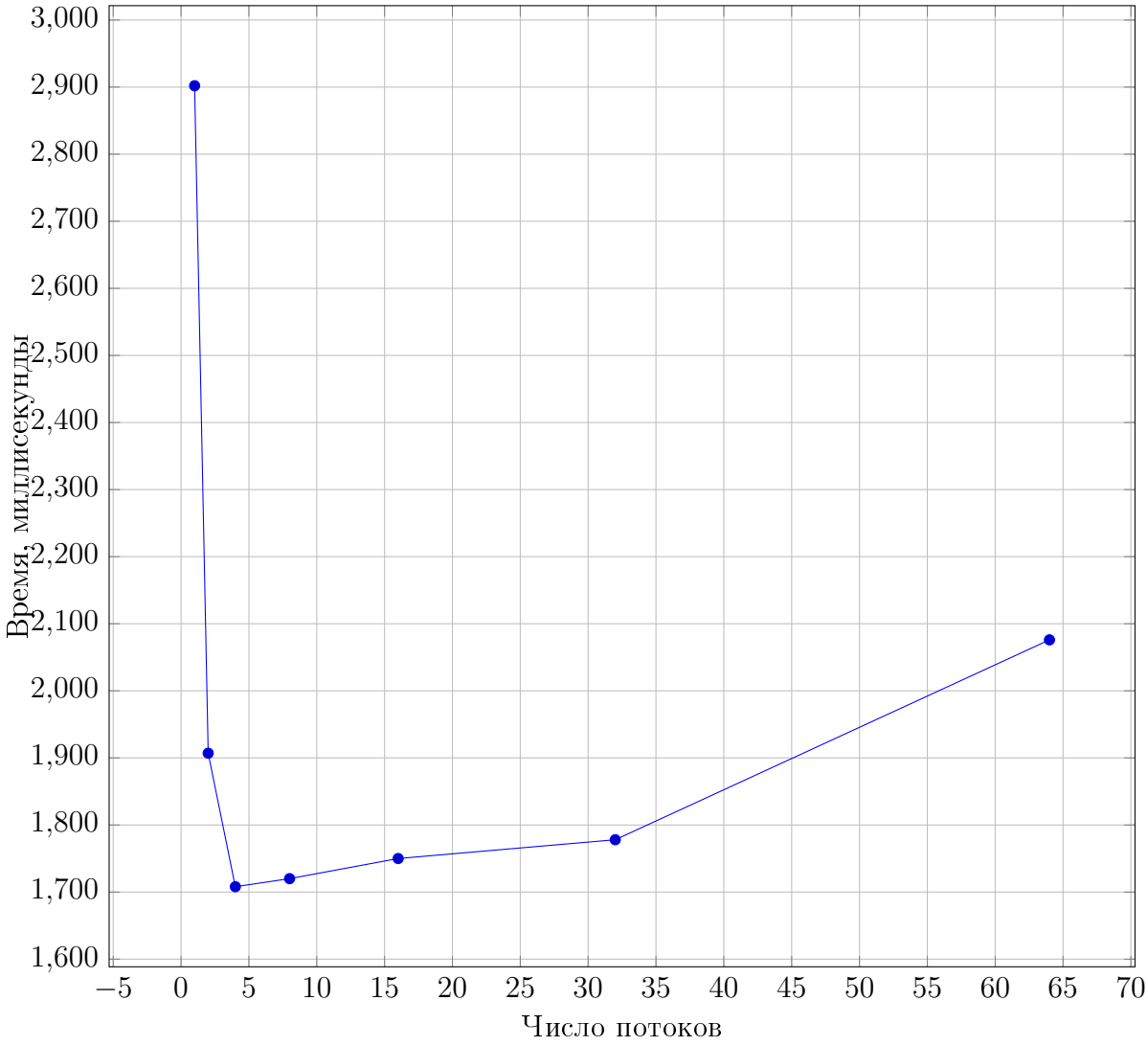

Рисунок 4.1 — График зависимости времени работы алгоритма обратной трассировки лучей от числа потоков

В ходе экспериментов по замеру времени работы было установлено:

1) параллельная реализация алгоритма обратной трассировки лучей в случае увеличения числа потоков в 2 раза, начиная с 4 замедляет скорость своей работы;

2) максимальный прирост скорости параллельной реализации алгоритма обратной трассировки лучей достигается при количестве потоков равных 4 и равен ≈ 1.7 раза.

Заключение

В ходе выполнения лабораторной работы были достигнуты следующие задачи:

1) изучен алгоритм обратной трассировки лучей;

2) реализован алгоритм обратной трассировки лучей;

3) проведены замеры процессорного времени работы от разного числа параллельных потоков;

4) проведено сравнение параллельных реализаций алгоритма обратной трассировки лучей со стандартной реализацией.

В ходе сравнения процессорного времени работы было установлено:

1) параллельная реализация алгоритма обратной трассировки лучей в случае увеличения числа потоков в 2 раза, начиная с 4 замедляет скорость своей работы;

2) максимальный прирост скорости параллельной реализации алгоритма обратной трассировки лучей достигается при количестве потоков равных 4 и равен ≈ 1.7 раза.

Эксперементальным путем было выяснено, что увеличение количества потоков в 2 раза не всегда увеличивает производительность программы.

Список использованных источников

1. C#. // [Электронный ресурс]. Режим доступа: https://docs.microsoft.com/ru-ru/dotnet/csharp/, (дата обращения: 4.11.2020).

2. IDE Visual Studio 2019. // [Электронный ресурс]. Режим доступа: https://visualstudio.microsoft.com/ru/vs/, (дата обращения: 4.11.2020).

3. Stopwatch. // [Электронный ресурс]. Режим доступа: https://docs.microsoft.com/ru-ru/dotnet/api/system.diagnostics.stopwatch?view=netcore-3.1, (дата обращения: 4.11.2020).

4. Thread. // [Электронный ресурс]. Режим доступа: https://docs.microsoft.com/ru-ru/dotnet/api/system.threading.thread?view=netcore-3.1, (дата обращения: 4.11.2020).

5. Intel® Core™ i3-8130U Processor. // [Электронный ресурс]. Режим доступа: https://ark.intel.com/content/www/ru/ru/ark/products/137977/intel-core-i3-8130u-processor-4m-cache-up-to-3-40-ghz.html, (дата обращения: 4.11.2020).