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: Параллельный алгоритм трассировки пути

Дисциплина: Анализ алгоритмов

Студент ИУ7-55Б (Группа) _____ (Подпись, дата) И. Е. Афимин (И.О. Фамилия)

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

Москва, 2020

Содержание

Введение

Трассировка лучей — один из методов геометрической оптики — исследование оптических систем путём отслеживания взаимодействия отдельных лучей с поверхностями. Трассировка пути — методика рендеринга в компьютерной графике, которая стремится симулировать физическое поведения света настолько близко к реальному, насколько это возможно.

Целью данной лабораторной работы является изучение и реализация параллельных вычислений для алгоритма трассировки пути.

Задачи данной лабораторной работы:

1) изучить параллельный алгоритм трассировки пути;

2) реализовать параллельный алгоритм трассировки пути;

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

4) провести сравнение параллельной реализации алгоритма трассировки пути со стандартной реализацией.

1 Аналитический раздел

В данном разделе будут рассмотрены основные теоритические понятия алгоритма трассировки пути.

1.1 Алгоритм трассировки пути

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 опыт восприятия окружающего мира.

Метод трассировки пути 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).

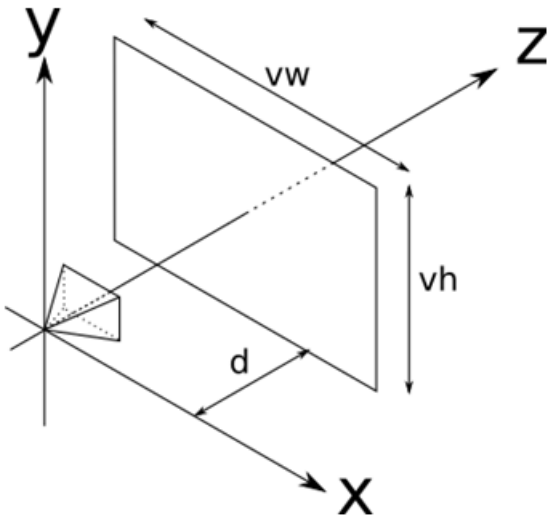

Рисунок 1.1 — Расположение камеры и сцены

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.

Исходя из того, как работает данный алгоритм, можно сделать вывод о том, что выпускать лучи для каждого пиксела можно независимо друг от друга. Поэтому данную часть алгоритма целесообразно выполнять параллельно.

1.2 Вывод

Был рассмотрен алгоритм трассировки пути и сделан вывод о том, что данный алгоритм является лучшим для визуализации реалистичного изображения. Так же рассмотрен вариант распараллеливания алгоритма.

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 трассировки пути.

Алгоритм трассировки пути (рисунок 2.1);

2.2 Параллельные вычисления

Распараллеливание программы должно уменьшать время работы. Это достигается за счет выполнения параллельно участков кода (например, 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 количества потоков;
2) обеспечить возможность просмотра полученного с помощью алгоритма трассировки пути изображения;
3) обеспечить вывод замеров времени работы алгоритма трассировки пути.

2.4 Методы тестирования

Тестирование ПО будет проводиться методом черного ящика.

2.5 Вывод

В данном разделе рассмотрена схема алгоритма трассировки пути и конкретизирован способ распараллеливания алгоритма.

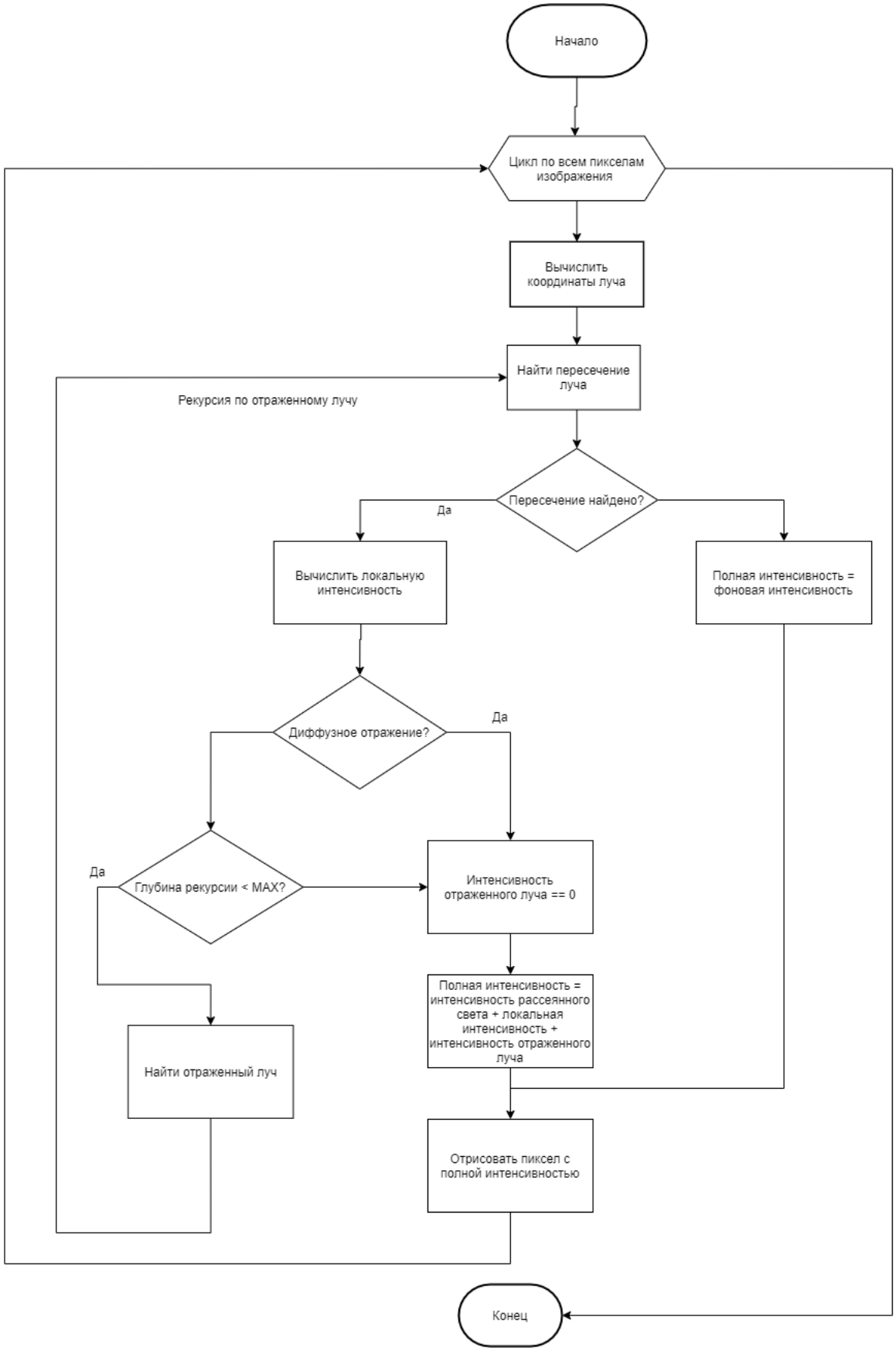

Рисунок 2.1 — Схема алгоритма трассировки пути

3 Технологический раздел

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

3.1 Средства реализации

Для реализации программ я выбрал язык программирования C++, так как имею большой опыт работы с ним. Для замера процессорного времени была использована стандартная библиотека C++ - chrono [1]. В качестве среды разработки использовался CLion 2020 [4].

Для распараллеливания были использованы потоки стандартной библиотеки C++ [5].

Листинг 3.1 — Реализация алгоритма трассировки пути с параллельными вычислениями

```
1     int n = 8; // Количество потоков
2
3     int step = image_width / n;
4
5     std::vector<std::thread *> threadArr;
6
7     int x1 = 0;
8     int x2 = step;
9
10    int y1 = 0;
11    int y2 = image_height;
12
13    auto buf = new color_buf[image_width * image_height];
14    for (int i = 0; i < image_width * image_height; i++)
15    {
16        buf[i].x1 = 0;
17        buf[i].x2 = 0;
18        buf[i].x3 = 0;
19    }
20
21    for (int i = 0; i < n; i++)
22    {
23        auto* newThread = new std::thread(cycle, x1, y2, x2, y1, samples_per_pixel,
              cam, background, world, max_depth, image_width, image_height, buf);
24
25        threadArr.push_back(newThread);
26
27        x1 = x2;
28        x2 += step;
29    }
30
31    for (int i = 0; i < n; i++)
32    {
33        threadArr[i]->join();
34    }
```

```
35
36      std::ofstream out;
37      out.open("img228.ppm");
38      out << "P3\n" << image_width << ' ' << image_height << "\n255\n";
39
40      for (int i = 0; i < 90000; i++)
41      {
42          out << buf[i].x1 << ' ' << buf[i].x2 << ' ' << buf[i].x3 << "\n";
43      }
```

3.2 Вывод

В данном разделе была рассмотрены средства реализации алгоритма и представлена реализация алгоритма трассировки пути.

4 Экспериментальный раздел

В данном разделе будут выполнены эксперименты для проведения сравнительного анализа алгоритма трассировки пути 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 трассировки пути 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) i5-8250U CPU, 8 логических процессоров, под управлением Windows 10 с 8 Гб оперативной памяти.

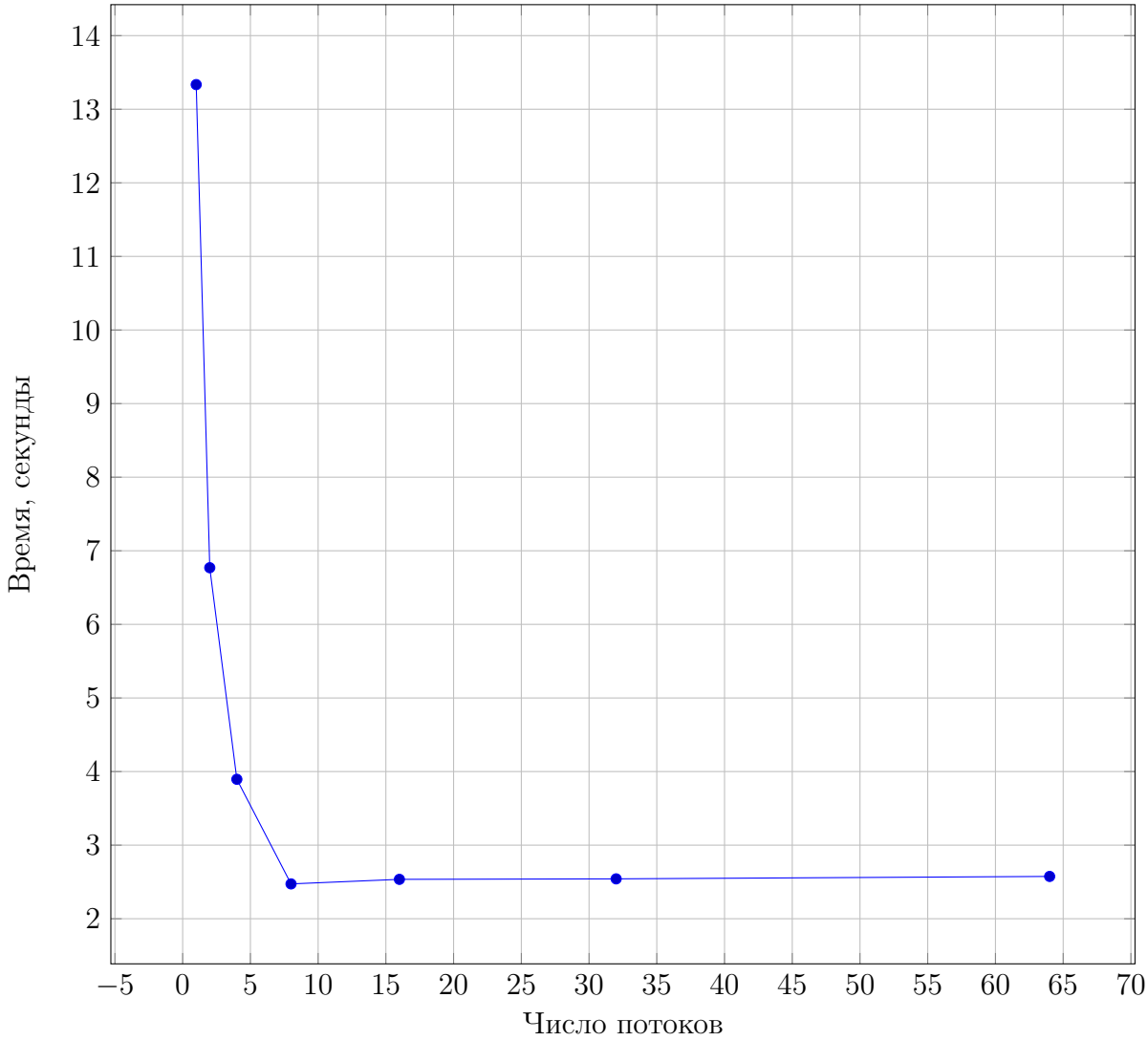

Рисунок 4.1 — График зависимости времени работы алгоритма трассировки пути от числа потоков

4.2 Вывод

В ходе экспериментов по замеру времени работы было установлено:

1) параллельная реализация алгоритма трассировки пути в случае увеличения числа потоков в 2 раза, начиная с 8 замедляет скорость своей работы;

2) максимальный прирост скорости параллельной реализации алгоритма трассировки пути достигается при количестве потоков равных 8 и равен ≈ 5.32 раза.

5 Заключение

В ходе выполнения лабораторной работы были выполнены следующие задачи:

1) изучен алгоритм параллельной трассировки пути;

2) реализован алгоритм параллельной трассировки пути;

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

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

В ходе сравнения процессорного времени работы было установлено:

1) параллельная реализация алгоритма трассировки пути в случае увеличения числа потоков в 2 раза, начиная с 8 замедляет скорость своей работы;

2) максимальный прирост скорости параллельной реализации алгоритма трассировки пути достигается при количестве потоков равных 8 и равен ≈ 5.32 раза.

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

Список использованных источников

1. Date and time utilities. // [Электронный ресурс]. Режим доступа: https://en.cppreference.com/w/cpp/chrono, (дата обращения: 20.10.2020).

2. C/C++: как измерять процессорное время. // [Электронный ресурс]. Режим доступа: https://habr.com/ru/post/282301/, (дата обращения: 20.10.2020).

3. Path tracing. // [Электронный ресурс]. Режим доступа: *https* : *//en.wikipedia.org/wiki/Path*$_t$*racing*, (дата обращения: 20.10.2020).

4. Clion. // [Электронный ресурс]. Режим доступа: *https* : *//www.jetbrains.com/clion/*, (дата обращения: 20.10.2020).

5. std::thread. // [Электронный ресурс]. Режим доступа: *https* : *//en.cppreference.com/w/cpp/thread/thread*, (дата обращения: 20.10.2020).