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
по дисциплине «Анализ алгоритмов»

Тема: Исследование многопоточности
Студент: Княжев А. В.
Группа: ИУ7-52Б
Оценка (баллы): _____
Преподаватели: Волкова Л. Л., Строганов Ю. В.

Москва — 2022 г.

Оглавление

Введение

Поток — наименьшая единица выполнения программы. В рамках процесса может параллельно выполняться несколько потоков [1]. Потоки разделяют адресное пространство процесса. Порядок выполнение потоков зависит от количества ядер процессора, так как параллельно может выполняться количество задач не большее количества ядер процессора. Если потоков больше, то они выполняются квазипараллельно, то есть потоки поочередно выполняются на ядрах процессора.

Использование потоков позволяет увеличить производительность задач, требующих большого объема вычислений, так как часть задач выполняется параллельно.

Цель работы

Получение навыков кодирования программного продукта, тестирования и проведения замерного эксперимента работы программы на различных данных. Все это на примере решения задачи реализации алгоритма обратной трассировки лучей в виде однопоточной и многопоточной версий.

Задачи работы

1) изучение алгоритма обратной трассировки лучей;
2) разработка многопоточной версии алгоритма обратной трассировки лучей;
3) кодирование данных алгоритмов;
4) проведение замерного эксперимента для данных алгоритмов, с измерением времени работы;
5) проведение сравнительного анализа алгоритмов на основе полученных данных.

1. Аналитическая часть

В данном разделе рассмотрены теоретические выкладки по алгоритму обратной трассировки лучей, а также по его многопоточной версии.

1.1. Алгоритм обратной трассировки лучей

Алгоритмы трассировки лучей, являются методами грубой силы, так как не учитывают специфику обрабатываемых объектов [2]. Идея алгоритмов заключается в отслеживании световых лучей между источниками света, объектами и наблюдателем.

Алгоритм прямой трассировки лучей отслеживает лучи от источников света, с учетом их отражения от объектов сцены. Данный алгоритм является неэффективным [2], так как множество лучей не доходят до наблюдателя, но все равно оказываются обработаны.

Альтернативой методу прямой трассировки является алгоритм обратной трассировки лучей, в котором происходит отслеживание световых лучей в обратном направлении, то есть от наблюдателя к объектам, от объектов к источникам света. Это позволяет обрабатывать только видимые наблюдателем лучи. Визуализация алгоритма приведена на рис. 1.1.

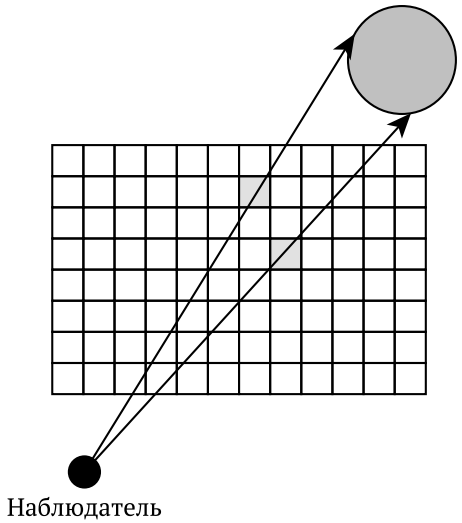

Рисунок 1.1 — Визуализация алгоритма обратной трассировки лучей

Алгоритмы позволяют учитывать тени и освещения, работают со сценами любой сложности. Кроме того, из-за присущей алгоритмам параллельности — лучи можно обрабаты-

вать независимо друг от друга, алгоритм можно реализовывать с использованием методов параллельной обработки [3].

1.2. Многопоточный алгоритм трассировки лучей

Основной идеей алгоритма трассировки лучей является то, что к каждому пикселу экрана пускается луч, и проверяется его пересечение с объектами сцены. Соответственно, так как пиксели не зависят друг от друга, то алгоритм можно использованием с использованием многопоточной технологии.

Для использования алгоритма на нескольких потоках, пикселы экрана нумеруются от 0 до $w \cdot h$, где w — ширина экрана в пикселах, h — высота экрана в пикселах.

Соответственно координаты точки экрана под номером n можно вычислить по формулам:

$$x = n \bmod w, \tag{1.1}$$

$$y = n \div w. \tag{1.2}$$

Количество пикселов, обрабатываемых каждым потоком, можно вычислить по формуле:

$$p = \frac{w \cdot h}{T}, \tag{1.3}$$

где

— p — количество пикселов, обрабатываемых каждый потоком;

— T — количество потоков.

Соответственно, каждый поток будет обрабатывать свой набор пикселов, где номер пиксела будет представлять собой элемент подмножества всех номеров пикселов длиной p.

2. Конструкторская часть

В данном разделе представлены схемы однопоточного и многопоточного алгоритма обратной трассировки лучей.

2.1. Разработка алгоритма обратной трассировки лучей

На рис. 2.1 представлена схема алгоритма обратной трассировки лучей.

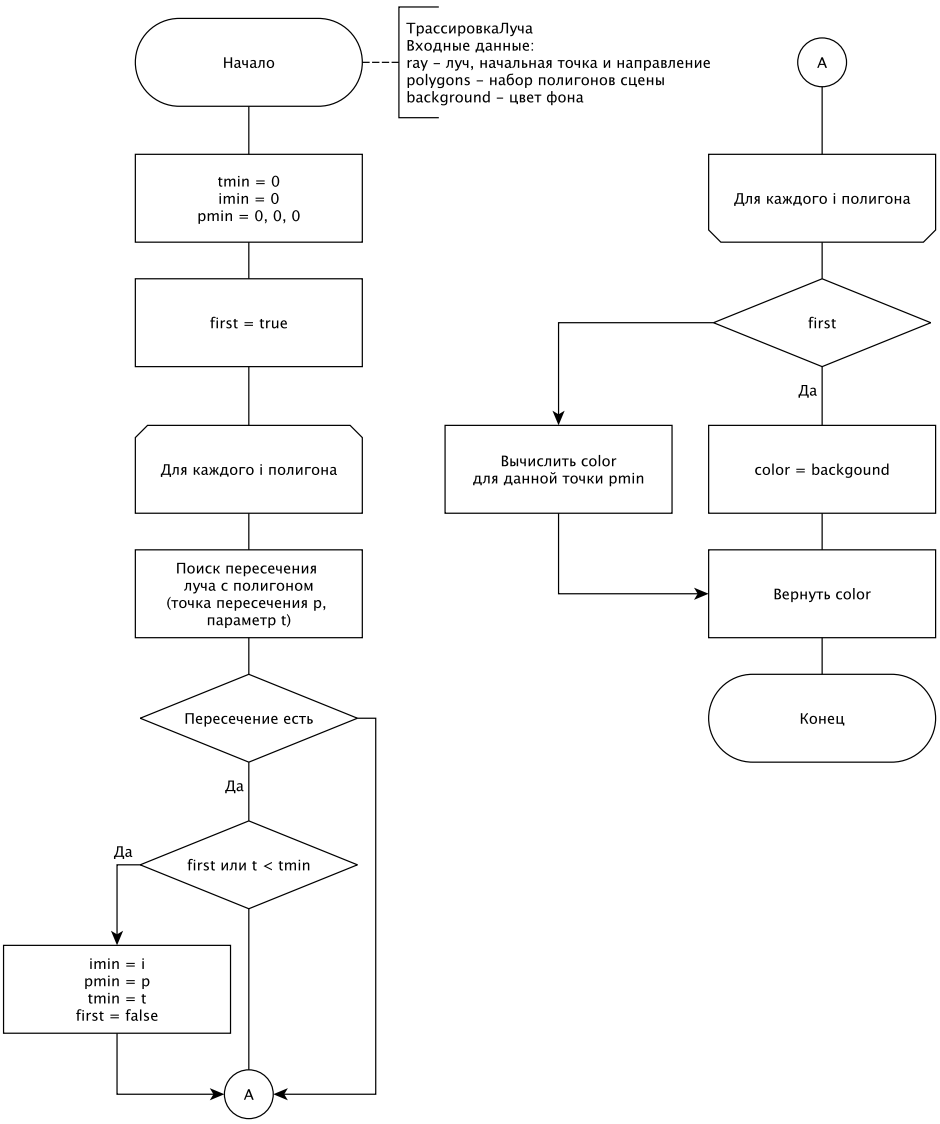

Рисунок 2.1 — Схема алгоритма обратной трассировки лучей

2.2. Разработка однопоточного алгоритма обратной трассировки лучей

На рис. 2.2 представлена схема однопоточного алгоритма обратной трассировки лучей.

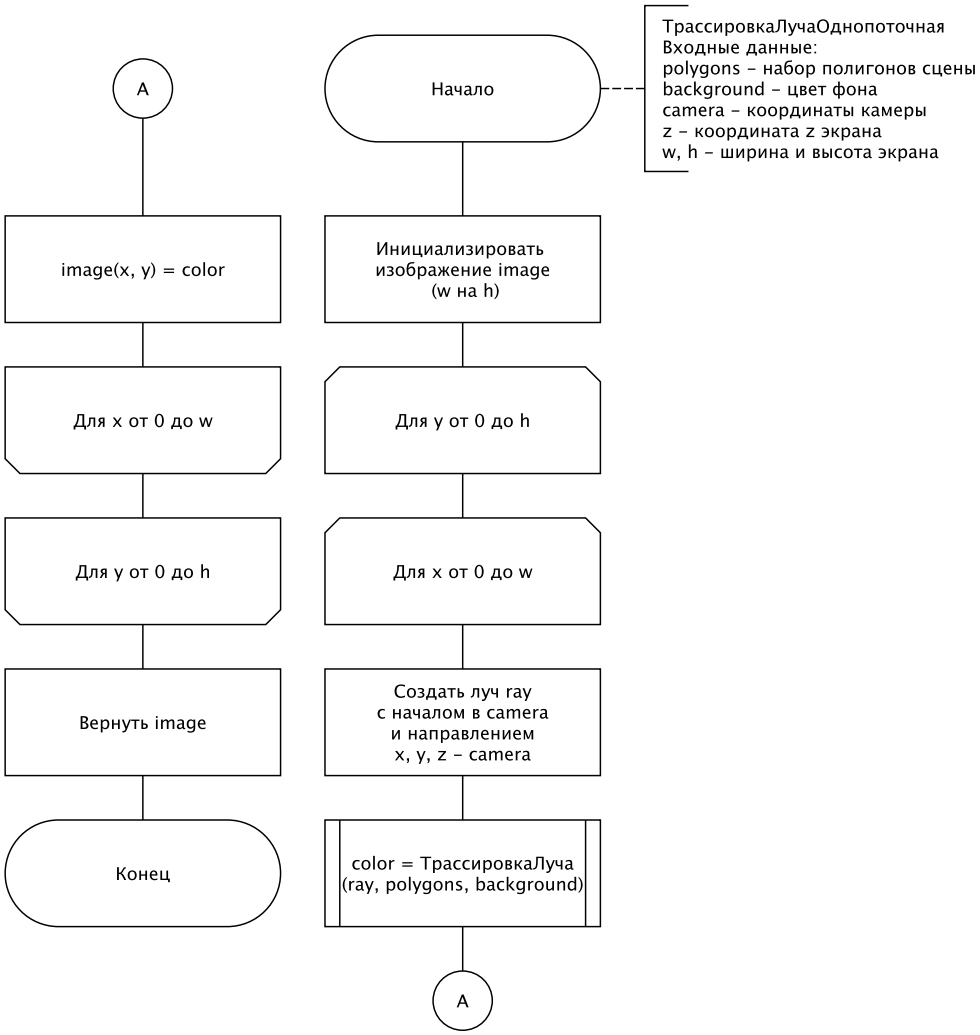

Рисунок 2.2 — Схема однопоточного алгоритма обратной трассировки лучей

2.3. Разработка многопоточного алгоритма обратной трассировки лучей

На рис. 2.3 – 2.4 представлена схема многопоточного алгоритма обратной трассировки лучей.

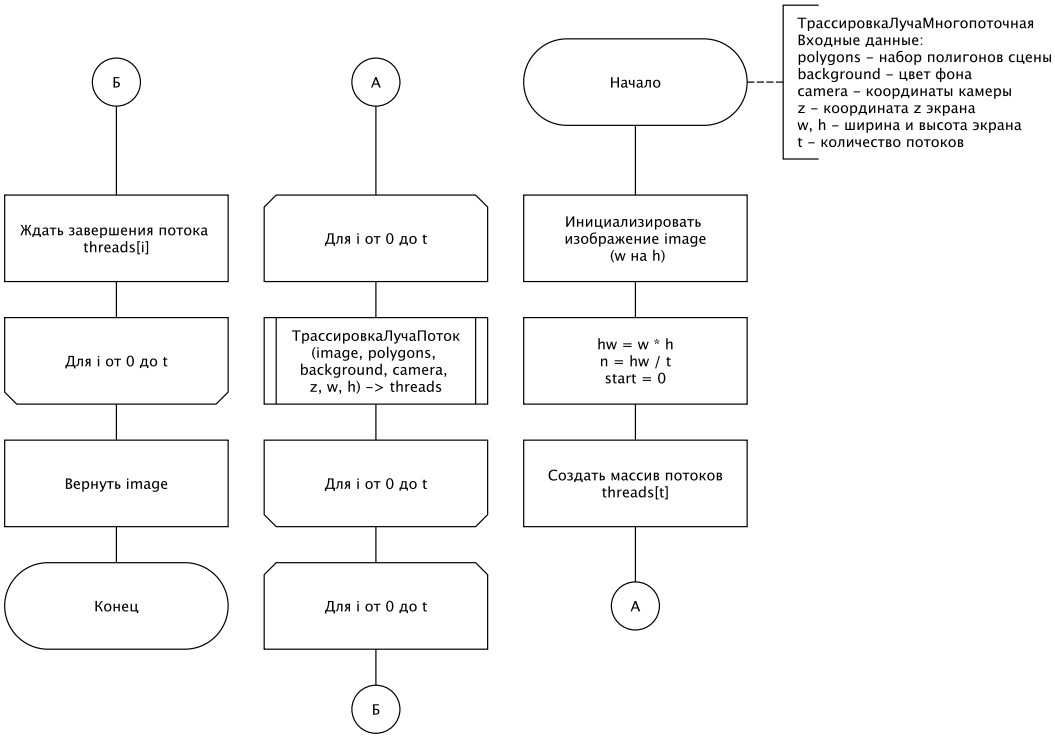

Рисунок 2.3 — Схема многопоточного алгоритма обратной трассировки лучей

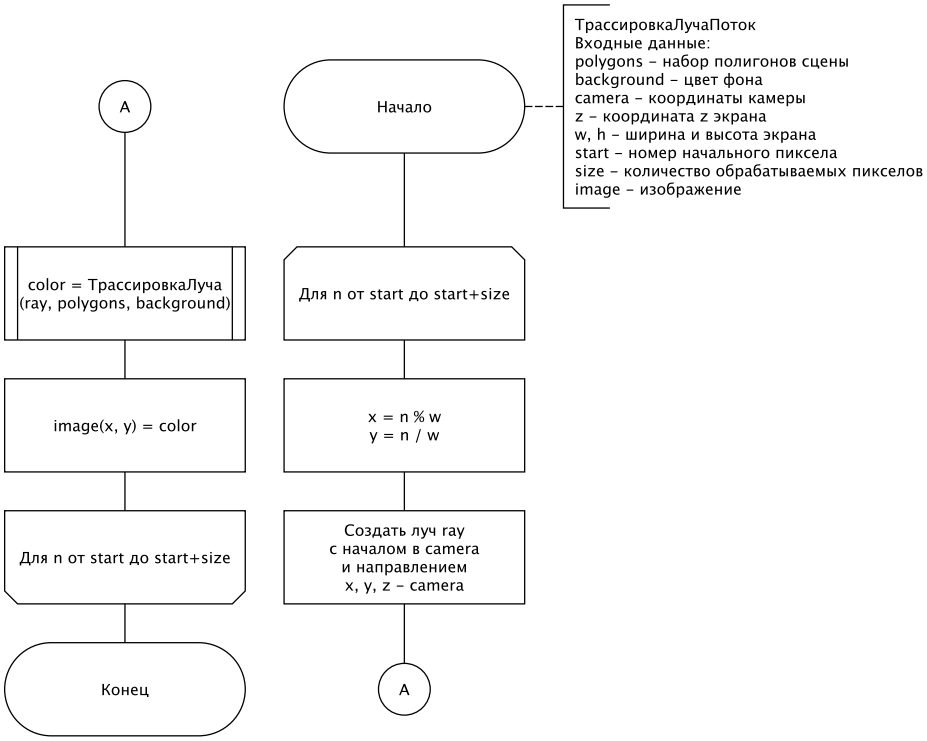

Рисунок 2.4 — Схема алгоритма обратной трассировки лучей (обработка одного потока)

3. Технологическая часть

В данном разделе представлены реализации алгоритмов обратной трассировки лучей. Кроме того, указаны требования к ПО и средства реализации.

3.1. Требования к ПО

— программа позволяет вводить имя файла, содержащего информацию о сцене, с помощью аргументов командной строки;

— программа аварийно завершается в случае ошибок, выводя сообщение о соответствующей ошибке;

— программа выполняет замеры времени работы реализаций алгоритмов;

— программа строит зависимости времени работы реализаций алгоритмов от размеров входных данных;

— программа предоставляет графический интерфейс.

3.2. Средства реализации

Для реализации данной работы выбран язык программирования C++, так как он содержит необходимые для тестирования библиотеки, а также обладает достаточными инструментами для реализации ПО, удовлетворяющего требованиям данной работы [4].

В качестве набора библиотек был выбран `Qt`, так как он обладает достаточным инструменталом для работы с компьютерной графикой, а также имеет набор средств для создания кроссплатформенных графических приложений [7].

3.3. Реализации алгоритмов

В листингах 3.1 – 3.4 представлены реализации алгоритмов обратной трассировки лучей.

Листинг 3.1 — Реализация алгоритма обратной трассировки лучей

```
QColor Scene::traceRay(const Ray &ray) {
    TraceRayData data;
    size_t index = 0;

    bool first = true;

    for (size_t i = 0; i < polygons.size(); i++) {
        auto res = polygons[i].traceRay(ray);

        if (!res.ok) {
            continue;
        }

        if (first || res.t < data.t) {
            index = i;
            data = res;
            first = false;
        }
    }

    if (first) {
        return background;
    }

    auto col = getColor(data.point, ray, lights, polygons[index]);
    if (!col.haveColor) {
        return Qt::black;
    }

    return col.color;
}
```

Листинг 3.2 — Реализация функции, выполняющей трассировку лучей в одном потоке

```
void Scene::paintThread(std::vector<std::vector<QColor>> &cbuf, int start,
int size)
{
    int maxSize = width * height;
    int end = start + size;
    if (end > maxSize) {
        end = maxSize;
    }

    for (int n = start; n < end; n++) {
        int y = n / width;
        int x = n % width;

        Ray ray = Ray(camera, (QVector3D(x, y, 0) - camera).normalized());

        cbuf[y][x] = traceRay(ray);
    }
}
```

Листинг 3.3 — Основная функция трассировки лучей

```
std::shared_ptr<QImage> Scene::paint(int nthreads) {
    polygons = forest.toPolygons();
    auto image = std::make_shared<QImage>(width, height,
     QImage::Format_RGB32);
    image->fill(Qt::black);
    QPainter painter(image.get());
    std::vector<std::vector<QColor>> colorBuffer(height,
     std::vector<QColor>(width, Qt::black));
    if (nthreads == 0) {
        Ray ray = Ray(camera, QVector3D());
```

Листинг 3.4 — Основная функция трассировки лучей (продолжение листинга 3.3)

```
        for (int y = 0; y < height; y++) {
            for (int x = 0; x < width; x++) {
                ray.direction = (QVector3D(x, y, 0) - camera).normalized();
                colorBuffer[y][x] = traceRay(ray);
            }
        }
    } else {
        int hw = height * width;
        int n =  hw / nthreads;
        if (hw % nthreads != 0) {
            n++;
        }
        int start = 0;
        std::vector<std::thread> threads(nthreads);
        for (int i = 0; i < nthreads; i++) {
            threads[i] = std::thread(&Scene::paintThread, this,
            std::ref(colorBuffer), start, n);
            start += n;
        }
        for (int i = 0; i < nthreads; i++)
            threads[i].join();
    }
    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            painter.setPen(colorBuffer[y][x]);
            painter.drawPoint(x, y);
        }
    }
    return image;
}
```

3.4. Тестирование

Тестирование проводилось по методологии белого ящика, с помощью проверки корректности получаемых изображений. **Тесты пройдены успешно.**

Ни рисунках 3.1 – 3.4 представлены тесты для алгоритма обратной трассировки лучей.

Рисунок 3.1 — Обычное дерево

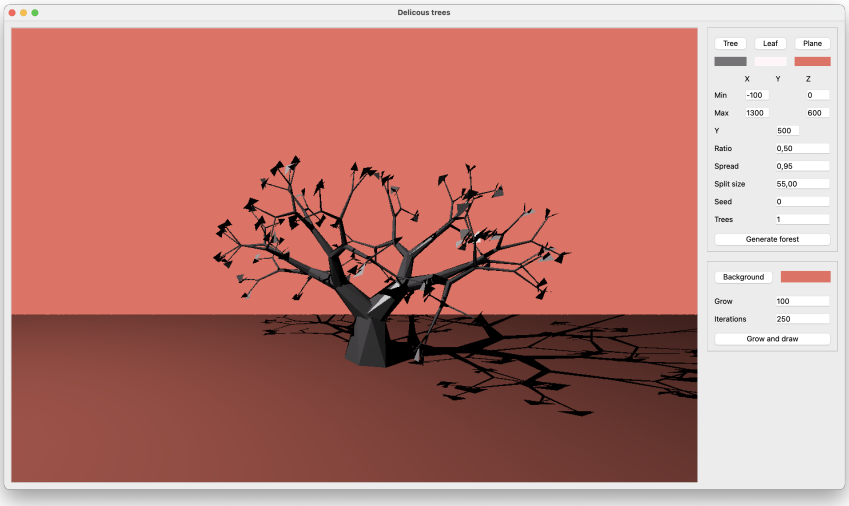

Рисунок 3.2 — Дерево в японском стиле

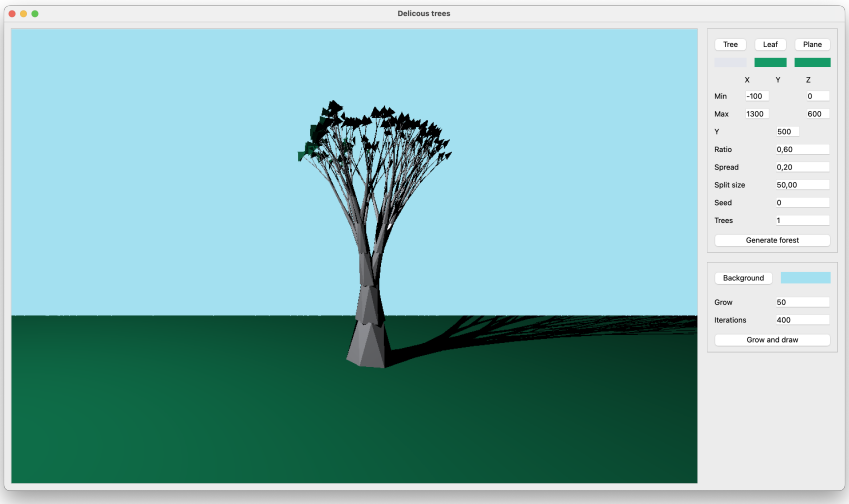

Рисунок 3.3 — Дерево в северном стиле

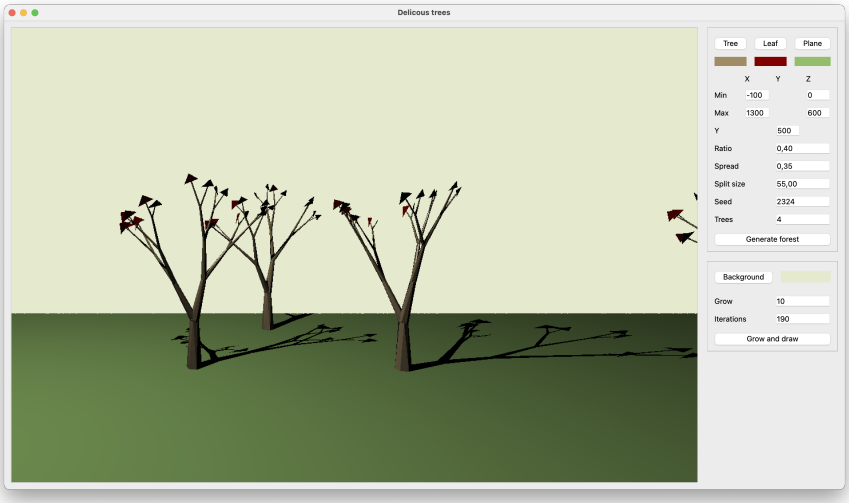

Рисунок 3.4 — Небольшой лес

4. Экспериментальная часть

В данном разделе описаны замерные эксперименты и представлены результаты исследования.

4.1. Технические характеристики

Технические характеристики устройства, на котором выполнялся эксперимент [5]:

— 96 ГБ оперативной памяти;

— процессор Intel Ice Lake (96 ядер);

— операционная система Ubuntu 20.04.

4.2. Постановка исследования

Целью исследования является определение зависимости времени генерации изображения лесистой местности от количества потоков, выделенных на выполнение данной задачи. Исследование проводилось для количества потоков, не превышающих 400, так как уже на этом значении скорость работы алгоритма не увеличивается с увеличением числа потоков. Во время проведения исследования устройство было подключено к блоку питания, и не нагружено программами, кроме операционной системы и средств разработки.

4.3. Средства исследования

Время синтеза изображения измеряется с использованием библиотеки `chrono` [6]. Измеряется реальное время, что особо важно с учетом того, проводится исследования параллельных вычислений.

4.4. Результаты исследования

Полученные результаты измерения описаны в таблице 4.1. Нулевое количество потоков означает отсутствие создания дополнительных потоков.

Таблица 4.1 — Результаты замеров времени (в мс)

Количество потоков	Время
0	28288
1	28817
2	24448
5	11734
10	6135
20	3214
30	2185
40	1741
60	1349
80	1100
100	1058
120	1004
140	1048
160	981
190	925
230	899
260	869
290	848
320	889
350	877
400	872

На рис. 4.1 приведен график, показывающий зависимость времени выполнения программы от количества выделенных потоков.

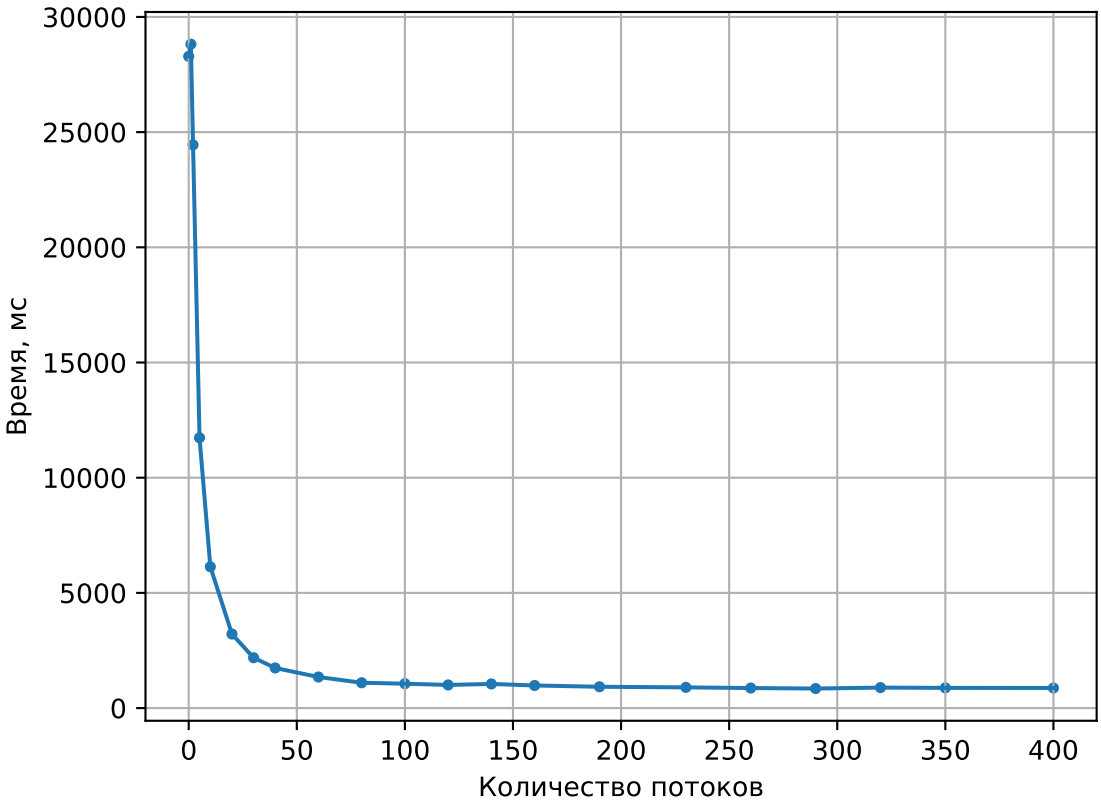

Рисунок 4.1 — График зависимости времени выполнения от количество потоков

Заключение

На основе исследования можно сделать следующие выводы. Скорость работы алгоритма обратной трассировки лучей повышается с увеличением количества потоков. Причем заметное увеличение скорости работы происходит при повышении количества потоков до 100, что близко к количеству логических ядер процессора (96). При дальнейшем повышении количества потоков, время работы алгоритма почти не изменяется.

При количестве потоков, равному 100, алгоритм работает быстрее однопоточной реализации в 26.7 раз.

При количестве потоков, равному 320, алгоритм начинает работать в 1.05 раз медленнее, чем при 290 потоках, так как начинают быть заметными затраты на диспетчеризацию большого числа потоков.

При создании единственного дополнительного потока, алгоритм работает дольше в 1.02 раза, чем без создания потоков, так как распараллеливания алгоритма не происходит, при этом есть затраты на диспетчеризацию.

Цель работы была достигнута: была изучена многопоточная реализация алгоритма обратной трассировки лучей. Были выполнены все задачи:

— изучены алгоритмы обратной трассировки лучей;

— разработана многопоточная версия алгоритма обратной трассировки лучей;

— кодированы данные алгоритмы;

— проведен замерный эксперимент для данных алгоритмов, с измерением времени работы;

— проведен сравнительный анализа алгоритмов на основе полученных данных.

Список использованных источников

1. Свищёва И. В., Беликов И. В. ПОТОКИ В LINUX //СОВРЕМЕННЫЕ ПРОБЛЕМЫ ЛИНГВИСТИКИ И МЕТОДИКИ ПРЕПОДАВАНИЯ РУССКОГО ЯЗЫКА В ВУЗЕ И ШКОЛЕ Учредители: Общество с ограниченной ответственностью"Научно-информационный центр"Интернум". – №. 37. – С. 1275-1278.

2. Роджерс Д. Алгоритмические основы машинной графики. – Москва: Издательство «Мир». Редакция литературы по математическим наукам, 2001.

3. Ульянов А. Ю., Котюжанский Л. А., Рыжкова Н. Г. Метод трассировки лучей как основная технология фотореалистичного рендеринга //Фундаментальные исследования. – 2015. – №. 11-6. – С. 1124-1128.

4. Josuttis N. M. The C++ standard library: a tutorial and reference. – 2012.

5. Yandex Compute Cloud [Электронный ресурс]. Режим доступа: *https://cloud.yandex.ru/services/compute* (дата обращения: 04.11.2022).

6. Date and time utilities [Электронный ресурс]. Режим доступа: *https://en.cppreference.com/w/cpp/chrono* (дата обращения: 06.11.2022).

7. Шлее М. Е. Qt 5.3: Профессиональное программирование на C++. – БХВ-Петербург, 2015.