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»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №4
По курсу: "Анализ алгоритмов"

Студент Сукочева Алис
Группа ИУ7-53Б
Название предприятия МГТУ им. Н. Э. Баумана, каф. ИУ7
Тема Параллельное программирование

Студент:	_____	Сукочева А.
	подпись, дата	Фамилия, И.О.
Преподаватель:	_____	Волкова Л.Л.
	подпись, дата	Фамилия, И. О.
Преподаватель:	_____	Строганов Ю.В.
	подпись, дата	Фамилия, И. О.

Москва — 2020 г.

Оглавление

Введение

Параллельные вычисления часто используются для увеличения скорости выполнения программ. Однако приемы, применяемые для однопоточных машин, для параллельных могут не подходить.

В данной лабораторной работе будет рассмотрено и реализованно параллельное программирование на примере задачи трассировки лучей.

Целью данной работы является изучения параллельных вычислений на материале трассировки лучей.

В рамках выполнения работы необходимо решить следующие задачи.

1. Изучения основ параллельных вычислений.
2. Применение изученных основ для реализации многопоточности на материале трассировки лучей.
3. Получения практических навыков.
4. Сравнительный анализ параллельной и однопоточной реализации алгоритма трассировки лучей.
5. Экспериментально подтверждение различий во временной эффективность реализации однопоточной и многопоточной трассировки лучей.
6. Описание и обоснование полученных результатов.
7. Выбор и обоснование языка программирования, для решения данной задачи.

1 Аналитическая часть

Для данной лабораторной работе, которая предполагает распараллеливание знакомого нам алгоритма был выбран алгоритм трассировки лучей. В данной части будет рассмотрено теоретическое описание алгоритмов.

1.1 Алгоритм прямой трассировки лучей

Для понимания алгоритма обратной трассировки лучей следует изначально разобраться с алгоритм прямой трассировки лучей.

Основная идея алгоритма прямой трассировки лучей состоит в том, что наблюдатель видит объекты, благодаря световым лучам, испускаемым некоторым источником, которые падают на объект, отражаются, преломляются или проходят через него и в результате достигают нас. Если проследить за лучами, то становится понятно, что среди них лишь малая часть дойдет до наблюдателя, что приведет к большим затратам ЭВМ. Заменой данному алгоритму служит метод обратный трассировки лучшей.

1.2 Алгоритм обратной трассировки лучей

Алгоритм обратной трассировки лучей отслеживает лучи в обратном направление (от наблюдателя к объекту). Считается, что наблюдатель расположен на положительной полуоси z в бесконечности, поэтому все световые лучи параллельны оси z. В ходе работы испускаются лучи от наблюдателя и ищутся пересечения луча и всех объектов сцены. В результате пересечение с максимальным значением z является видимой частью поверхности и атрибуты данного объекта используются для определения

характеристик пикселя, через центр которого проходит данный световой луч. Эффективность процедуры определения пересечений луча с поверхностью объекта оказывает самое большое влияние на эффективность всего алгоритма. Чтобы избавиться от ненужного поиска пересечений было придумано искать пересечение луча с объемной оболочкой рассматриваемого объекта. Под оболочкой понимается некоторый простой объект, внутрь которого можно поместить рассматриваемый объект, к примеру параллелепипед или сфера (рис. 1.1).

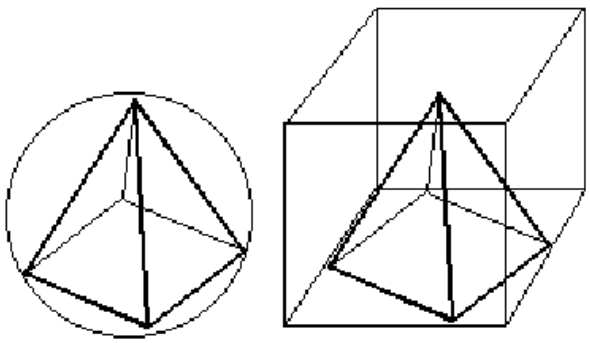

Рис. 1.1: Сферическая и прямоугольная оболочки

В дальнейшем при рассмотрении пересечения луча и объемной оболочкой рассматриваемого объекта, если такого пересечения нет, то и соответственно пересечения луча и самого рассматриваемого объекта нет, и наоборот, если будет найдено пересечение, то возможно, есть пересечение луча и рассматриваемого объекта. Для расчета эффектов освещения сцены проводятся вторичные лучи от точек пересечения ко всем источникам света. Если на пути этих лучей встречается непрозрачное тело, значит данная точка находится в тени, иначе он влияет на освещение данной точки. Также для получения более реалистичного изображения сцены, нужно учитывать вклады отраженных и преломленных лучей. К недостатку алгоритма относится его производительность.

1.3 Применение алгоритма

Алгоритм трассировки лучей используется для визуализации реалистического изображения, учитывая тени, отражение объектов, преломление

и т.д. Данный алгоритм активно используется, к примеру, в игровой индустрии.

1.4 Параллельная реализация алгоритма обратной трассировки лучей

Поскольку алгоритм обратной трассировки лучей обрабатывает каждый пиксель экрана независимо, можно распараллелить обработку всего экрана, разбив его на некоторые части. В данной лабораторной работе будет представлено разбиение горизонтально и вертикально. Т.е. каждый поток будет обрабатывать свой участок экрана.

На рис. 1.2 показано, как можно вертикально разбить экран на несколько частей. На рис. 1.3 показано горизонтальное разбиение экрана. Разбив экран на части можно реализовывать параллельное вычисление цвета пикселей каждой части экрана.

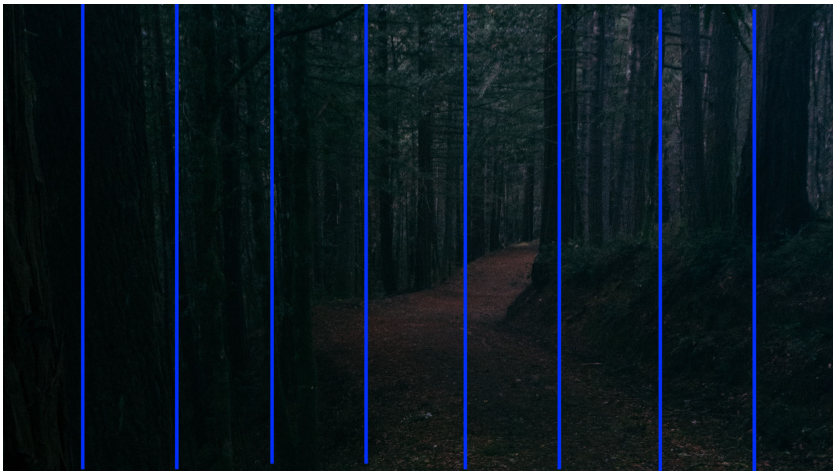

Рис. 1.2: Вертикальное разбиение экрана

1.5 Вывод

В данном разделе были рассмотрены основополагающие материалы, которые в дальнейшем потребуются при параллельной и однопоточной ре-

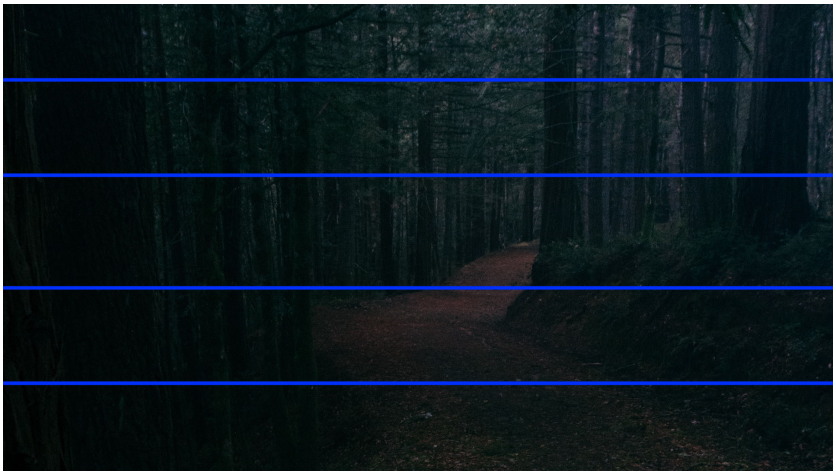

Рис. 1.3: Горизонтальное разбиение экрана

ализации алгоритма трассировки лучей.

2 | Конструкторская часть

В данном разделе будет рассмотрены схемы.

На рис. 2.1 показана схема алгоритма главного потока.

На рис. 2.2 показана схема работы потока, который разбивает экран вертикально.

На рис. 2.3 показана схема работы потока, который разбивает экран горизонтально.

На рис. 2.4 показана схема однопоточного алгоритма трассировки лучей.

2.1 Вывод

В данном разделе были рассмотрены схемы однопоточной (рис. 2.4) и многопоточной (рис. 2.1 - 2.3) реализации алгоритма обратной трассировки лучей

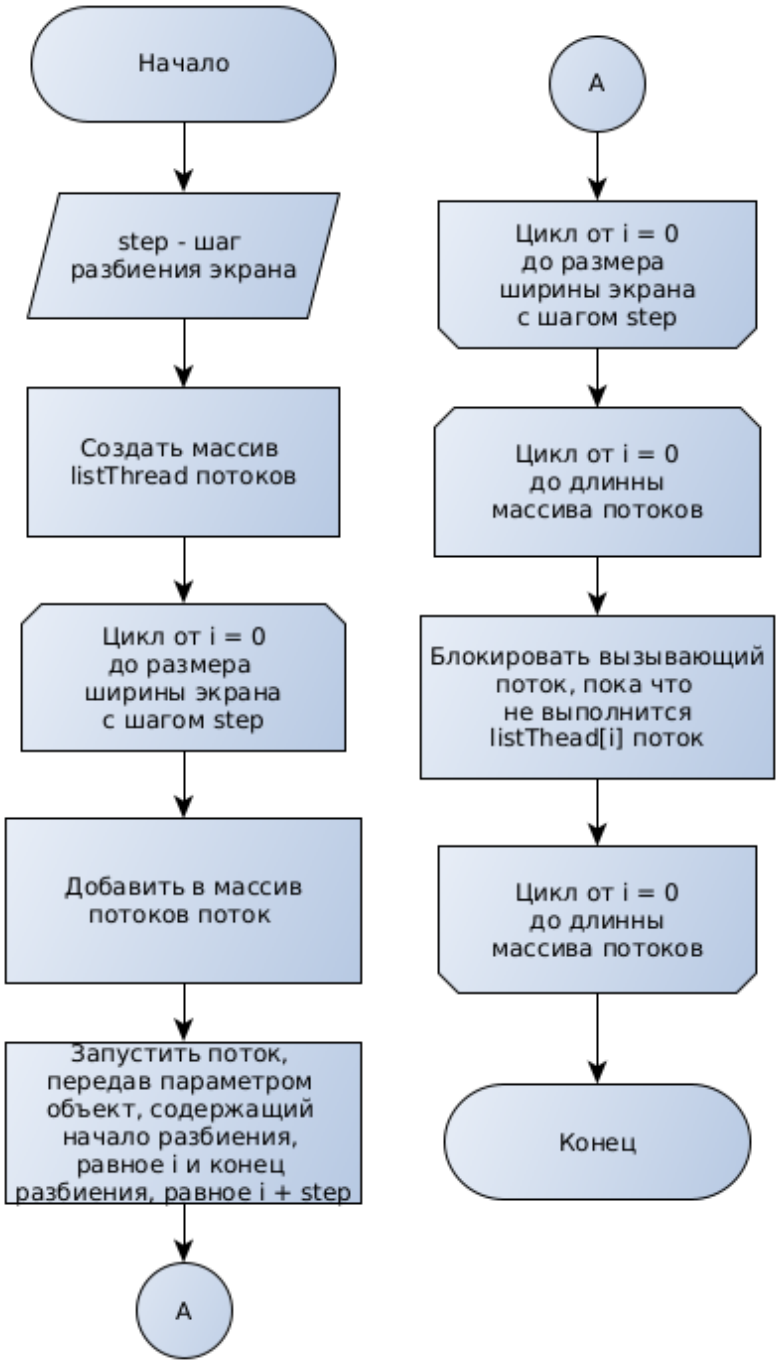

Рис. 2.1: Схема алгоритма главного потока

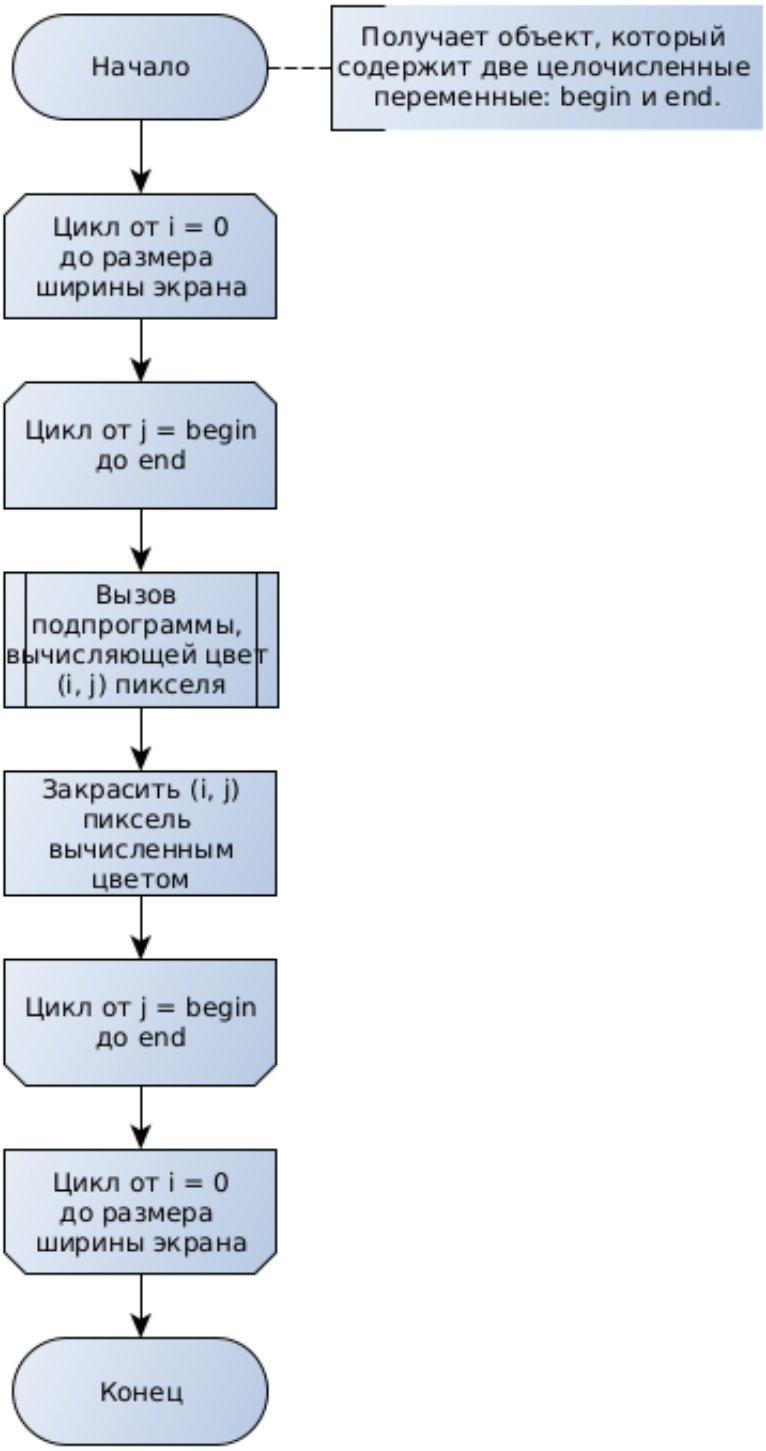

Рис. 2.2: Вертикальное разбиение экрана

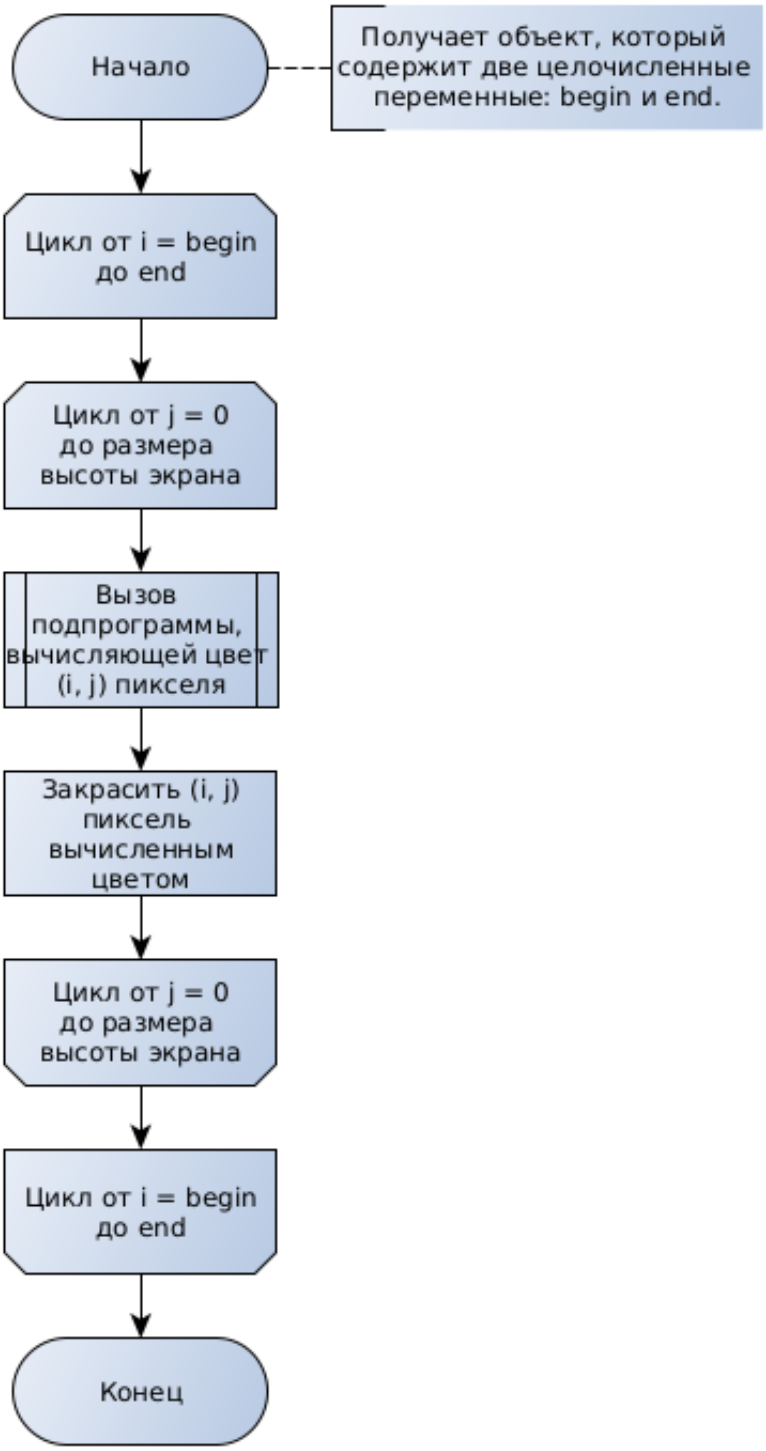

Рис. 2.3: Горизонтальное разбиение экрана

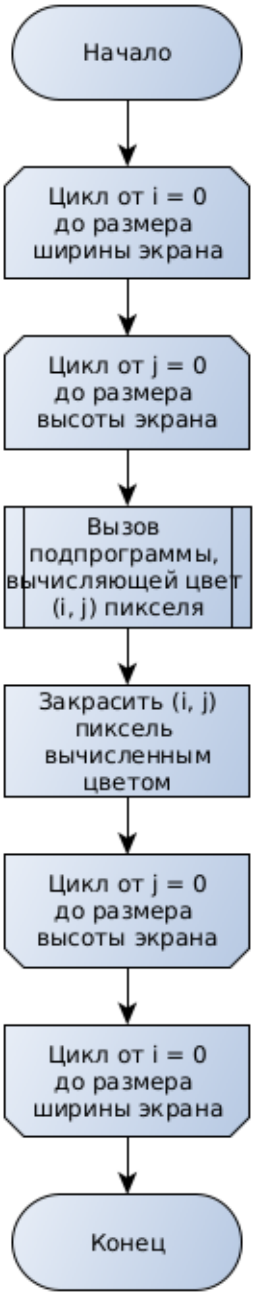

Рис. 2.4: Однопоточная трассировка лучей

3 | Технологическая часть

3.1 Выбор ЯП

В данной лабораторной работе использовался язык программирования - C# [4]. Данный язык является нативным. Также я знакома с ним. Поэтому данный язык был выбран. В качестве среды разработки я использовала Visual Studio Code [1]. Visual Studio Code подходит не только для Windows [2], но и для Linux [3], это причина, по которой я выбрала VS code, т.к. у меня установлена ОС Ubuntu 18.04.4 [5]. В моей архитектуре присутствует 8 ядер.

3.2 Сведения о модулях программы

Данная программа разбита на модули:

- Program.cs - Файл, содержащий точку входа в программу.
- MainFrom.cs - Файл, содержащий основной код программы.

На листингах 3.1-3.4 представлен основной код программы.

Листинг 3.1: Метод создания и запуска потоков

```
1 private void DrawScene()
2 {
3   List<Thread> listThread = new List<Thread>();
4
5   int step = 50;
6   for (int i = 0; i < (int)SizeObjects.WidthCanvas; i +=
      step)
```

```
7   {
8     listThread.Add(new Thread(new ParameterizedThreadStart(
          FuncHorizontally)));
9     listThread[listThread.Count - 1].Start(new Limit(i, i +
          step));
10  }
11
12  foreach (var elem in listThread)
13    elem.Join();
14
15  _imgBox.Image = _img;
16 }
```

Листинг 3.2: Метод потока разбиения горизонтально

```
1 public void FuncHorizontally(object obj)
2 {
3   Limit limit = (Limit)obj;
4   for (Int32 i = limit.begin; i < limit.end; i++)
5     for (Int32 j = 0; j < (int)SizeObjects.HeightCanvas; j
          ++)
6       _img.SetPixel(i, j, TraceRay(new Point(i, j)));
7 }
```

Листинг 3.3: Метод потока разбиения вертикально

```
1 public void FuncVertically(object obj)
2 {
3   Limit limit = (Limit)obj;
4   for (Int32 i = 0; i < (int)SizeObjects.WidthCanvas; i++)
5     for (Int32 j = limit.begin; j < limit.end; j++)
6       _img.SetPixel(i, j, TraceRay(new Point(i, j)));
7 }
```

Листинг 3.4: Однопоточный метод трассировки лучей.

```
1 private void funcTrace()
2 {
3   for (Int32 i = 0; i < (int)SizeObjects.WidthCanvas; i++)
4     for (Int32 j = 0; j < (int)SizeObjects.HeightCanvas; j
          ++)
```

```
5            _img.SetPixel(i, j, TraceRay(new Point(i, j)));
6        }
```

3.3 Тестирование

В данном разделе приведен рис. 3.1 на котором показан результат работы программы.

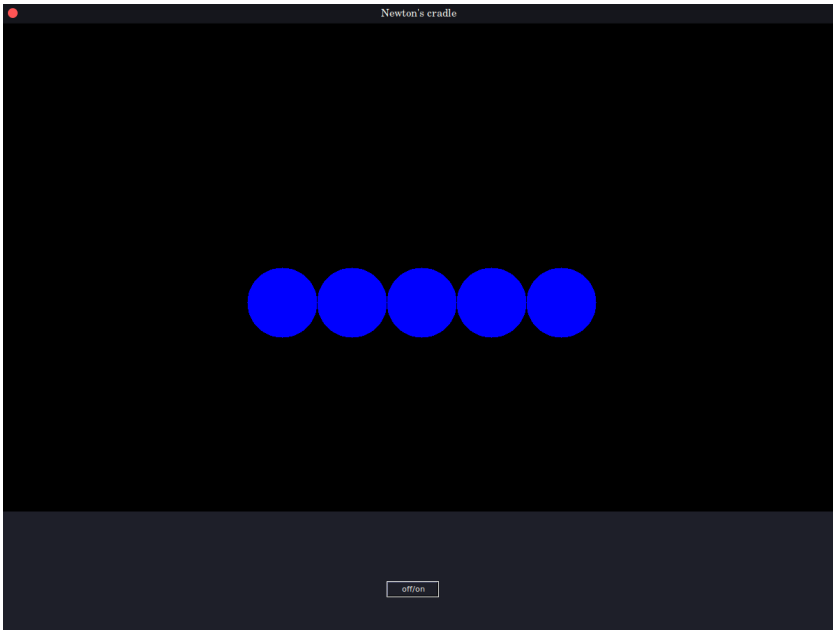

Рис. 3.1: Результат работы программы

3.4 Вывод

В данном разделе были разобраны листинги рис 3.1-3.4, показывающие работу как однопоточного, так и многопоточного алгоритма трассировки лучей. Также приведен рис. 3.1, показывающий корректную работу алгоритма.

4 | Экспериментальная часть

В данном разделе будет произведено сравнение вышеизложенных алгоритмов.

4.1 Временные характеристики

Для сравнения возьмем изображение размером 1200×700. Так как трассировка данного изображения считается достаточно короткой задачей, воспользуемся усреднением массового эксперимента. Сравнение произведем при $n = 50$.

Результат представлен на рис. 4.1.

Обычная реализация работает быстрее, чем создание одного потока, потому что на создание потока тратится некоторое время. При двух потоках выигрыш получается почти в два раза, так как два потока трассируют одновременно свою часть экрана. При увеличении числа потоков отслеживается уменьшение времени трассировки. При 8 потоках достигается пик, при котором все ядра процессора одновременно выполняют трассировку экрана. Далее при увеличении числа потоков производительность падает. Это объясняется тем, что создается очередь потоков, которая замедляет работу программы.

4.2 Вывод

В данном разделе было произведено сравнение алгоритма трассировки лучей при простой реализации и многопоточной (рис. 4.1). Результат показал, что выгоднее всего использовать все ядра процессора.

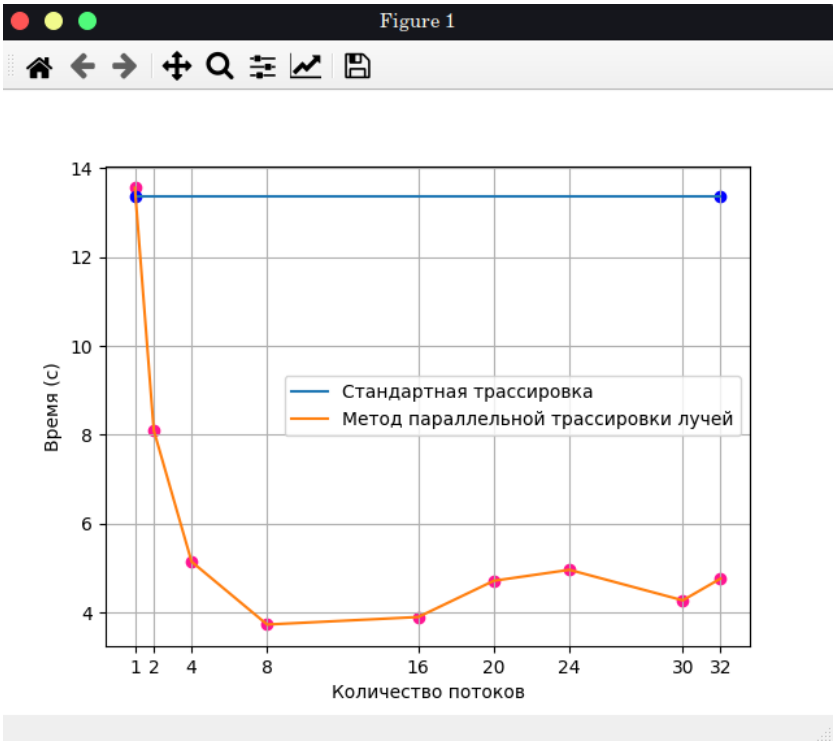

Рис. 4.1: Временные характеристики

Заключение

В данной лабораторной работе были рассмотрены основополагающие материалы которые в дальнейшем потребовались при параллельной и однопоточной реализации алгоритма трассировки лучей. Были рассмотрены схемы однопоточной (рис. 2.4) и многопоточной (рис. 2.1 - 2.3) реализации ранее разобранного алгоритма. Также были разобраны листинги рис 3.1-3.4, показывающие работу как однопоточного, так и многопоточного алгоритма трассировки лучей и был приведен рис. 3.1, показывающий корректную работу алгоритма. Было произведено и показано сравнение работы алгоритма на разных количествах потоков (рис. 4.1).

В рамках выполнения работы решены следующие задачи.

1. Изучены основы параллельных вычислений.
2. Применены изученные основы при реализации многопоточности на материале трассировки лучей.
3. Получены практические навыки.
4. Произведен сравнительный анализ параллельной и однопоточной реализации алгоритма трассировки лучей.
5. Экспериментально подтверждены различия во временной эффективность реализации однопоточной и многопоточной трассировки лучей.
6. Описаны и обоснованы полученные результаты.
7. Выбран и обоснован языка программирования, для решения данной задачи.

Литература

[1] Visual Studio Code [Электронный ресурс], режим доступа: https://code.visualstudio.com/ (дата обращения: 02.10.2020)

[2] Windows [Электронный ресурс], режим доступа:https://www.microsoft.com/ru-ru/windows (дата обращения: 02.10.2020)

[3] Linux [Электронный ресурс], режим доступа:https://www.linux.org.ru/ (дата обращения: 02.10.2020)

[4] Руководство по языку C#[Электронный ресурс], - режим доступа: https://docs.microsoft.com/ru-ru/dotnet/csharp/

[5] Ubuntu 18.04 [Электронный ресурс], режим доступа:https://releases.ubuntu.com/18.04/ (дата обращения: 02.10.2020)