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"

Тема Параллельные вычисления на основе нативных потоков

Студент Золотухин А.В.

Группа ИУ7-54Б

Оценка (баллы)

Преподаватели Волкова Л.Л., Строганов Ю.В.

Москва — 2022 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 реализовано параллельное программирование на примере задачи трассировки лучей.

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

В рамках выполнения работы необходимо решить следующие задачи:

— описание основных положений базового алгоритма расчёта;

— применение изученных основ для реализации многопоточности на материале трассировки лучей;

— получение практических навыков параллельных вычислений на основе нативных потоков;

— проведение сравнительного анализа параллельной и однопоточной реализации алгоритма трассировки лучей;

— экспериментальное подтверждение различий во временной эффективности реализации однопоточной и многопоточной трассировки лучей;

— описание и обоснование полученных результатов;

1 Аналитическая часть

В этом разделе будут представлено описание алгоритма обратной трассировки лучей.

1.1 Алгоритм обратной трассировки лучей

Из источников света испускается множество лучей (первичные лучи). Часть этих лучей не встретит никаких препятствий, а часть попадет на объекты. При попадании на них лучи преломляются и отражаются. При этом часть энергии луча поглотится. Преломленные и отраженные лучи образуют новое поколение лучей. Далее эти лучи опять же преломятся, отразятся и образуют новое поколение лучей. В конечном итоге часть лучей попадет в камеру и сформирует изображение. Это описывает работу прямой трассировки лучей.

Метод обратной трассировки лучей позволяет значительно сократить перебор световых лучей. В этом методе отслеживаются лучи не от источников, а из камеры. Таким образом, трассируется определенное число лучей, равное разрешению картинки.

Так как для каждого пикселя на экране цвет можно вычислять независимо друг от друга, то для параллельной обратной трассировки лучей будет достаточно равным образом распределить пиксели между потоками.

Вывод

В данном разделе было представлено описание алгоритма обратной трассировки лучей.

2 Конструкторская часть

В этом разделе будут приведены схемы алгоритмов и вычисления трудоемкости данных алгоритмов.

2.1 Разработка алгоритмов

На рисунках 2.1, 2.2, 2.3 представлены схемы алгоритмов рабочего потока, потока диспетчера, и последовательного алгоритма обратной трассировки лучей.

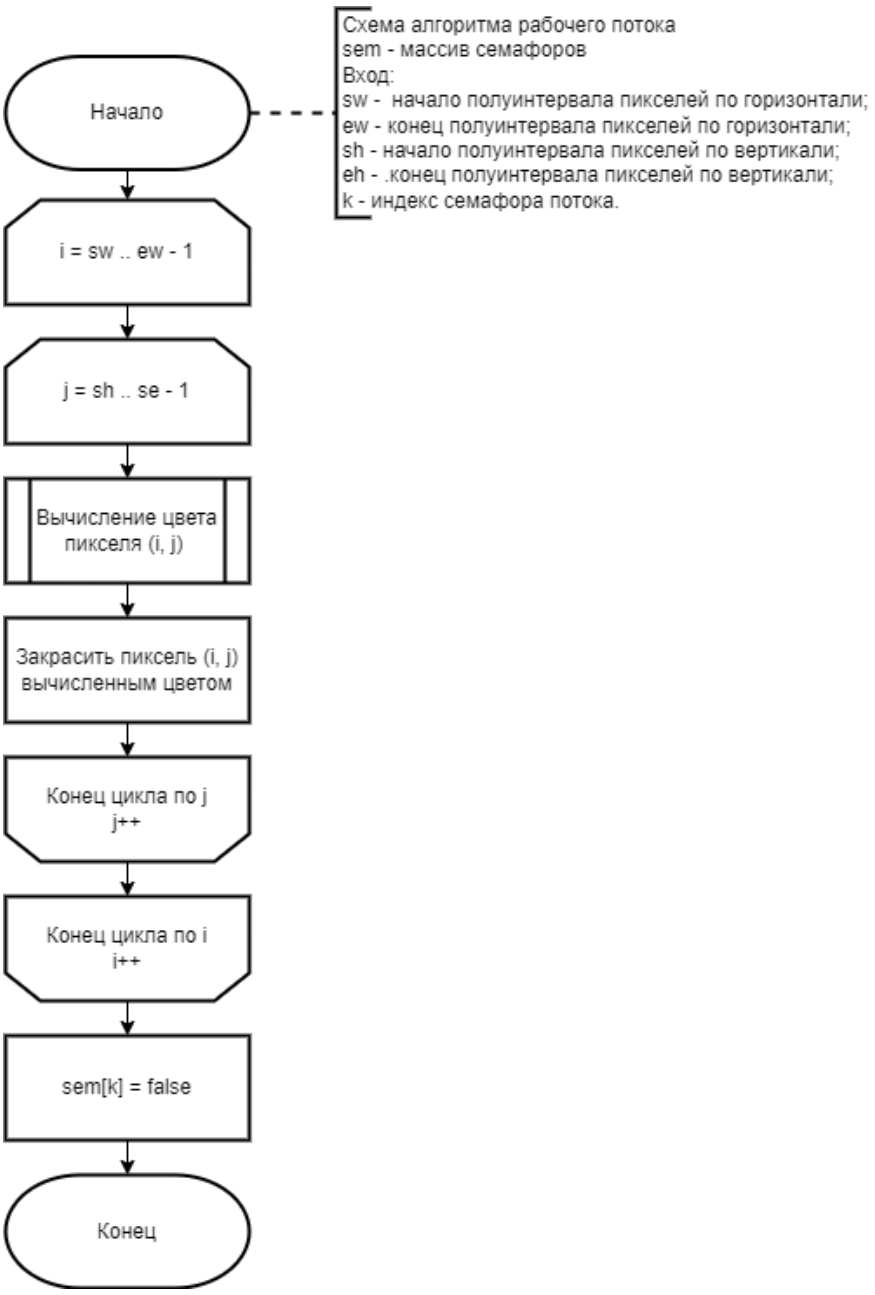

Рисунок 2.1 – Схема алгоритма рабочего потока

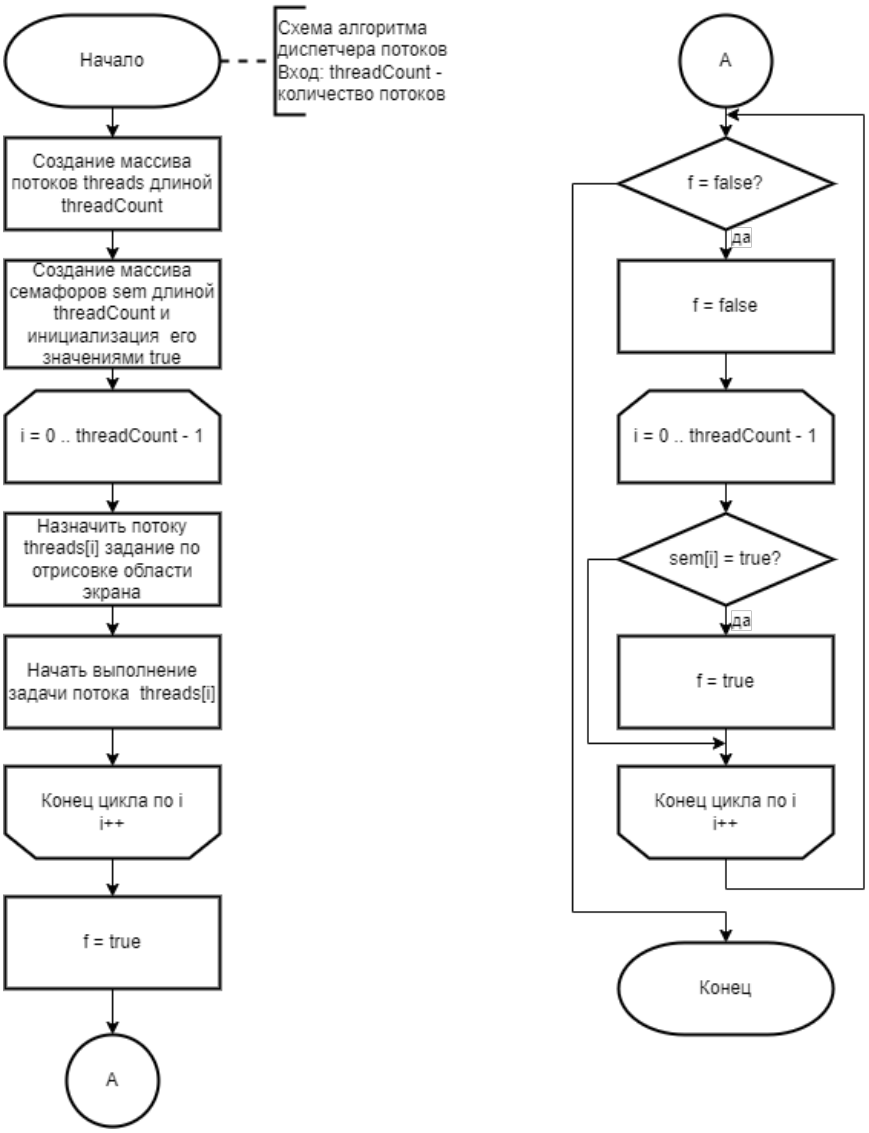

Рисунок 2.2 – Схема алгоритма потока диспетчера

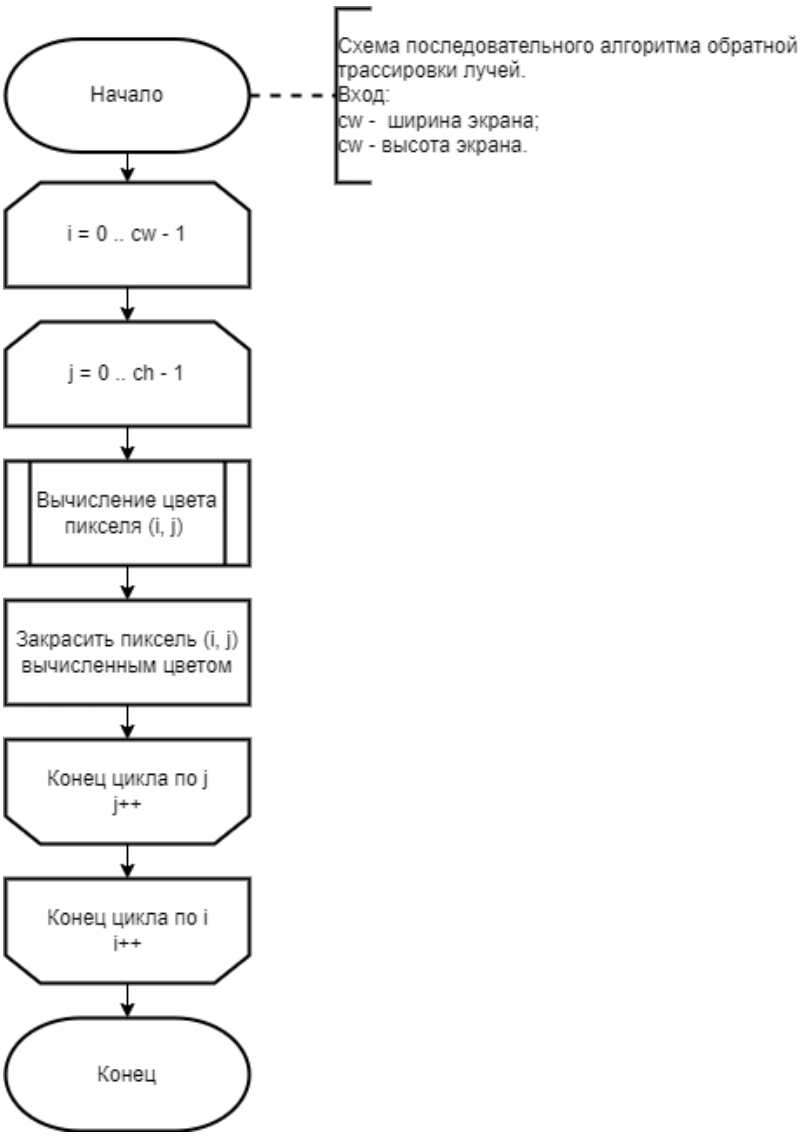

Рисунок 2.3 – Схема последовательного алгоритма обратной трассировки лучей

Вывод

Были разработаны схемы трех алгоритмов, а именно алгоритмов для рабочего потока, для потока-диспетчера, и последовательного алгоритма обратной трассировки лучей.

3 Технологическая часть

В данном разделе будут приведены требования к программному обеспечению, средства реализации и листинги кода.

3.1 Средства реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран язык программирования C# [1].

Язык C# является полностью объектно-ориентированным. Все необходимые библиотеки для реализации поставленной задачи являются стандартными.

Также в этом языке используются нативные потоки [?].

Время работы алгоритмов было замерено с помощью класса Stopwatch [4].

Для создания потоков использовалась библиотека System.Threading.

3.2 Сведения о модулях программы

Программа состоит из следующих модулей.

1. Programm.cs — главный файл программы, в котором располагается код меню.

2. Scene.cs — файл с классом сцены в котором располагаются методы исследуемых алгоритмов.

3. Composite.cs, Cube.cs, LightSource.cs, LockBitmap.cs, Ray.cs, Trace.cs, SceneObject.cs, Sphere.cs, Trace.cs — файлы с кодами классов, необходимых для реализации обратной трассировки лучей.

3.3 Реализация алгоритмов

В листингах 3.1, 3.2, 3.3 представлены реализации алгоритмов, а именно алгоритм рабочего потока, потока диспетчера, и последовательного алгоритма обратной трассировки лучей.

Листинг 3.1 – Метод рабочего потока

```
1  private void RenderPiece(object parametrs)
2  {
3      Params p = (Params)parametrs;
4      int sw = p.WidthStart;
5      int ew = p.WidthEnd;
6      int sh = p.HeightStart;
7      int eh = p.HeightEnd;
8
9      for (int i = sw; i < ew; i++)
10     for (int j = sh; j < eh; j++)
11     {
12         Trace t = TraceRay(i - (Cw / 2), -j + (Ch / 2));
13         CastShadow(t);
14         RenderSmoke(t);
15         lbmp.SetPixel(i, j, t.Color);
16     }
17     sem[p.SemaphoreIndex] = false;
18 }
```

Листинг 3.2 – Метод потока диспетчера

```
1  public void Render(int threadCount)
2  {
3      Thread[] threads = new Thread[3 * (int)Math.Pow(2,
           threadCount + 1)];
4      sem = new bool[3 * (int)Math.Pow(2, threadCount + 1)];
5      int dtc = threadCount / 2;
6      int mtc = threadCount % 2;
7      int jh = 3 * (int)Math.Pow(2, dtc);
8      int iw = (int)Math.Pow(2, dtc + mtc + 1);
9      int k = 0;
10     lbmp.LockBits();
11     for (int i = 0; i < iw; i++)
12     for (int j = 0; j < jh; j++)
```

```
13      {
14          sem[k] = true;
15
16          threads[k] = new Thread(RenderPiece);
17          threads[k].Start(new Params(i * Cw / iw, j * Ch / jh, (i
                + 1) * Cw / iw, (j + 1) * Ch / jh, k));
18          k++;
19      }
20      bool f = true;
21      while (f)
22      {
23          f = false;
24          for (int i = 0; i < sem.Length; i++)
25          if (sem[i])
26          f = true;
27      }
28      lbmp.UnlockBits();
29  }
```

Листинг 3.3 – Последовательный алгоритм обратной трассировки лучей

```
1  public void RenderFollow()
2  {
3      for (int i = 0; i < Cw; i++)
4      for (int j = 0; j < Ch; j++)
5      {
6          Trace t = TraceRay(i - (Cw / 2), -j + (Ch / 2));
7          CastShadow(t);
8          RenderSmoke(t);
9          bmp.SetPixel(i, j, t.Color);
10     }
11 }
```

3.4 Тестирование

Тестирование выполнено по методологии белого ящика. Полученное изображение 3.1 совпадает с ожидаемым.

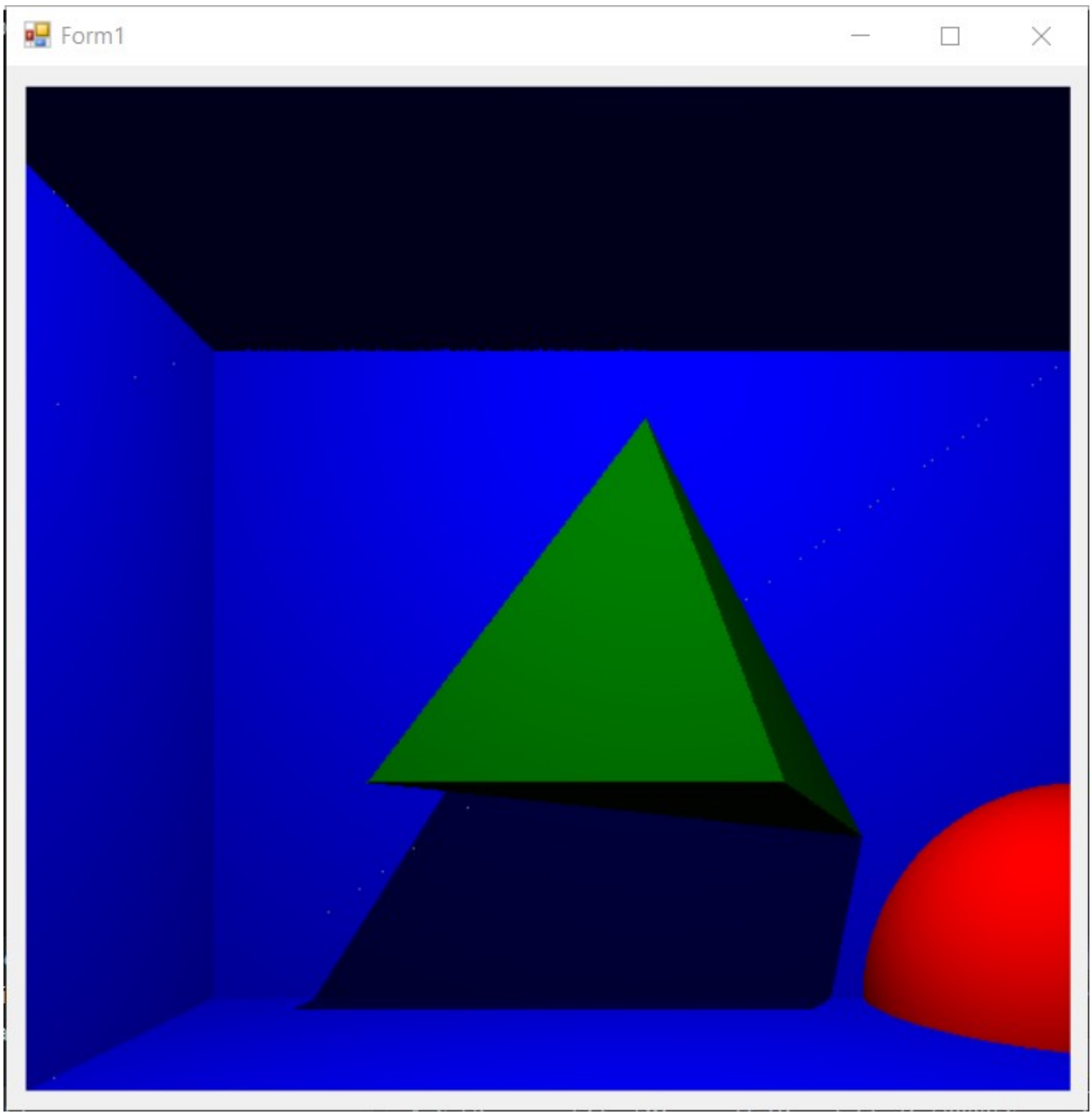

Рисунок 3.1 – Полученное изображение в результате работы алгоритма

Вывод

Были реализованы алгоритмы, а именно алгоритм рабочего потока, потока диспетчера, и последовательного алгоритма обратной трассировки лучей.

4 Исследовательская часть

В данном разделе будут приведены примеры работы программ, постановка эксперимента и сравнительный анализ алгоритмов на основе полученных данных.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось исследование.

1. Операционная система: Windows 10 Корпоративная, Версия 21H1, Сборка ОС 19043.2006.
2. Оперативная память: 8 ГБ.
3. Процессор: AMD Ryzen 5 4600H с видеокартой Radeon Graphics 3.00 ГГц [5].

Исследование проводилось на ноутбуке, включенном в сеть электропитания. Во время исследования ноутбук был нагружен только встроенными приложениями окружения, а также непосредственно системой.

4.2 Демонстрация работы программы

На рисунках 4.1, 4.2, 4.3, представлены результат работы алгоритма обратной трассировки лучей.

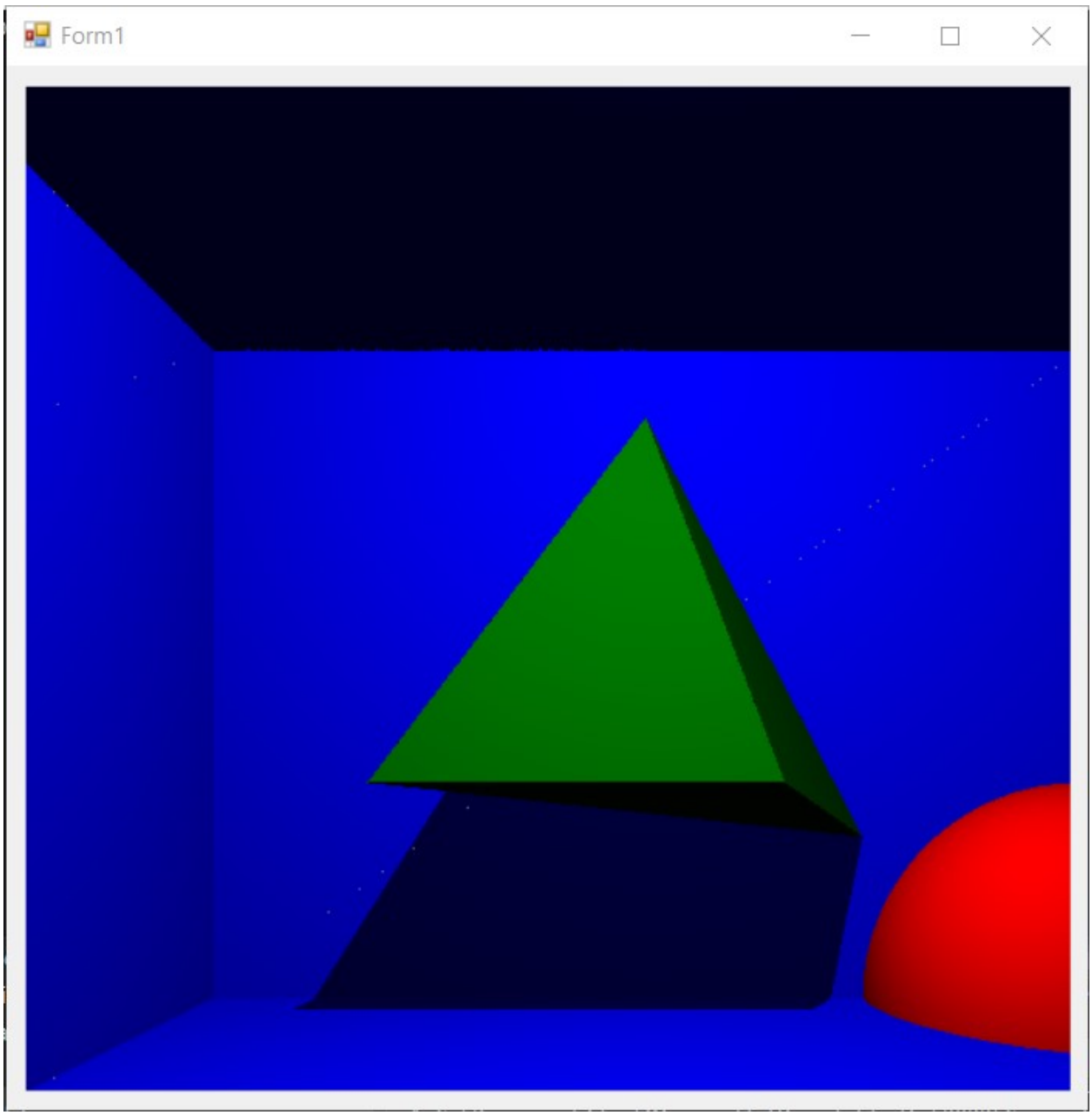

Рисунок 4.1 – Демонстрация работы программы в графическом интерфейсе без дыма

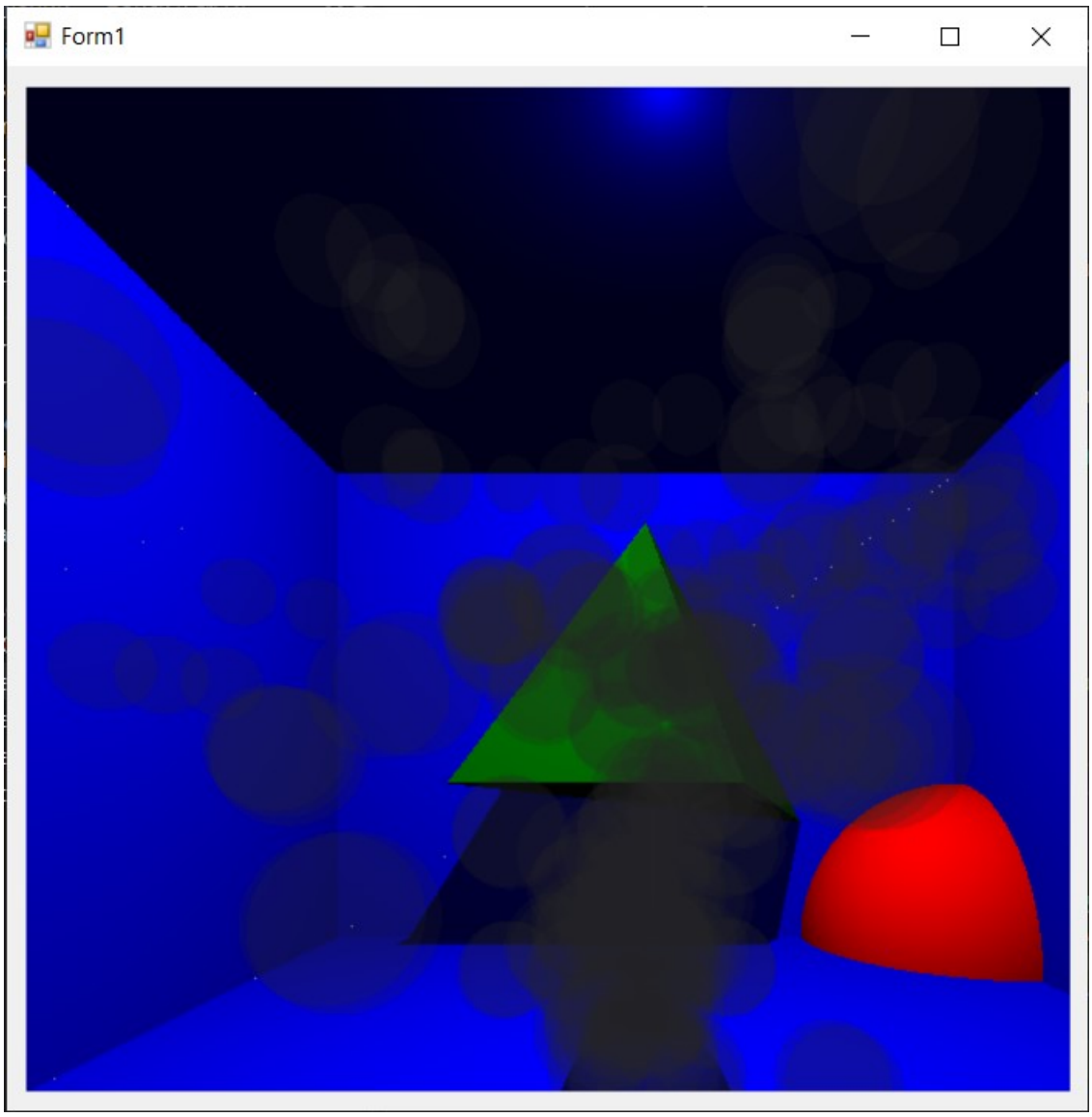

Рисунок 4.2 – Демонстрация работы программы в графическом интерфейсе с дымом

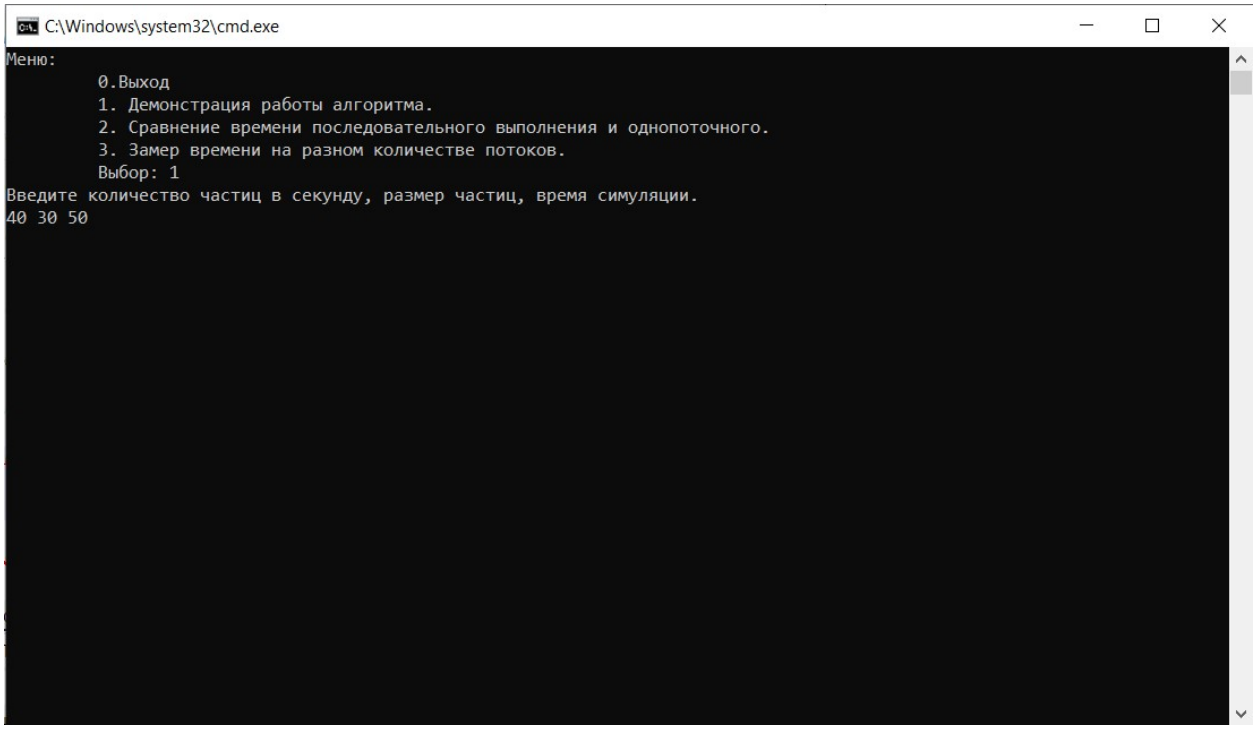

Рисунок 4.3 – Демонстрация работы программы в консоли

4.3 Время выполнения реализаций алгоритмов

Время выполнения реализаций алгоритмов было замерено при помощи класса Stopwatch [4].

Замеры времени для каждого количества потоков, каждого количества дополнительных объектов на сцене проводились 50 раз. В качестве результата взято среднее время работы алгоритма на данном количестве потоков, данном количестве дополнительных объектов на сцене.

4.3.1 Сравнение времени работы реализаций последовательного алгоритма и однопоточного

Результаты замеров приведены в таблице 4.1. На рисунке 4.4, приведен график зависимости времени работы реализаций алгоритмов при последовательном и однопоточном выполнении от количества объектов. Под

дополнительными объектами понимаются объекты, добавленные пользователем на сцену после ее создания.

Таблица 4.1 – Время выполнения алгоритма обратной трассировки лучей при однопоточном и последовательном выполнении

Количество дополнительных объектов	Однопоточное выполнение	Последовательное выполнение
0	4284,38	3973,45
100	15784,67	14135,63
200	26939,55	24102,88
300	37635,08	33852,40
400	48896,23	43472,16

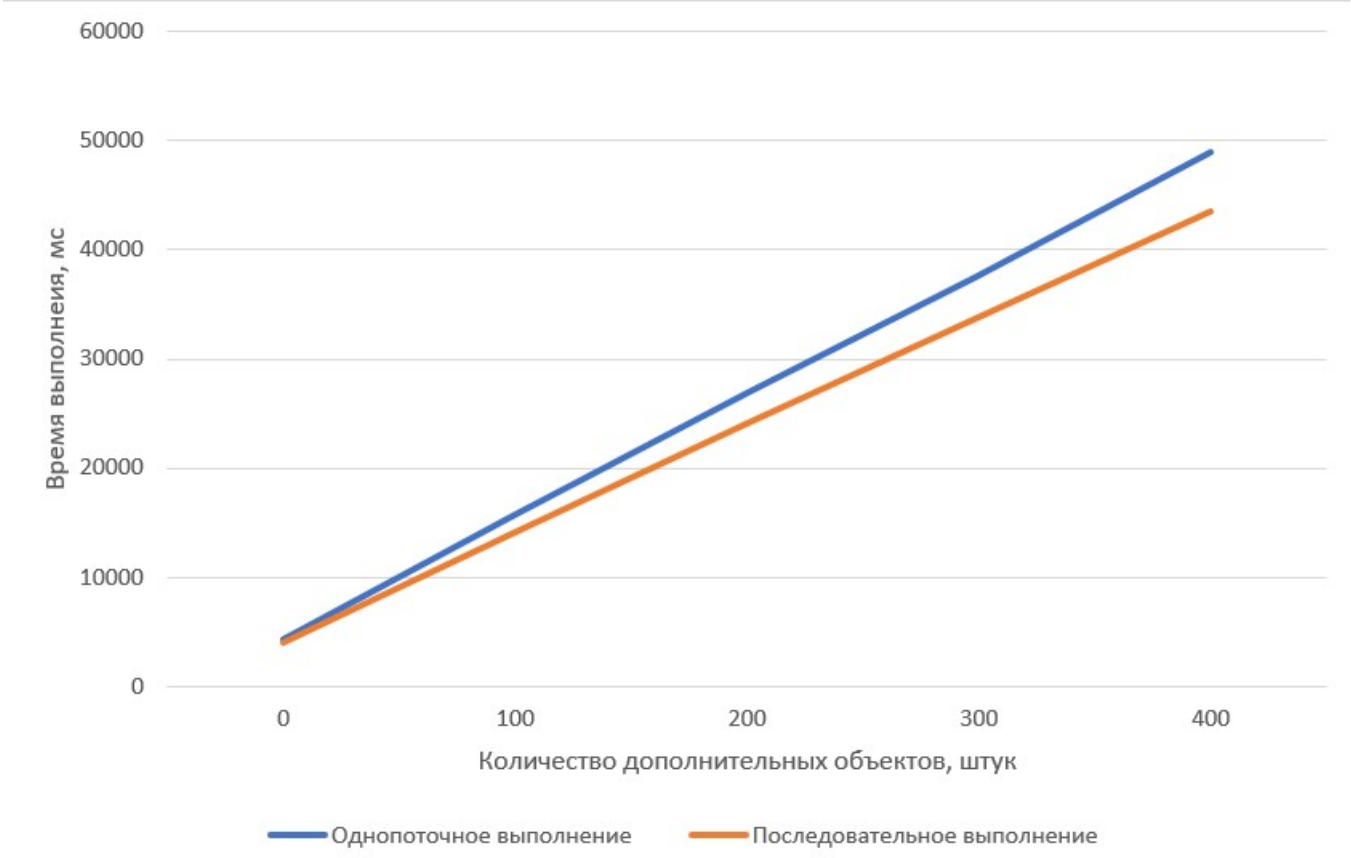

Рисунок 4.4 – Зависимость времени выполнения реализации алгоритма обратной трассировки лучей при однопоточном и последовательном выполнении

Обычная реализация работает быстрее, чем многопоточный с одним рабочим потоком, потому что на создание потока тратится некоторое время.

4.3.2 Сравнение времени выполнения реализаций на разном количестве потоков и дополнительных объектов сцены

Результаты замеров приведены в таблице 4.2. На рисунке 4.5 приведена зависимость времени работы реализации параллельного алгоритма от количества потоков и количества дополнительных объектов на сцене.

Таблица 4.2 – Время выполнения реализации алгоритма обратной трассировки лучей при количества потоков и количества дополнительных объектов на сцене

Количество дополнительных объектов, штук	Количество потоков, штук				
	6	12	24	48	96
0	1042,34	846,74	948,4	1060,27	1100,81
100	4296,44	3985,65	4652,67	5525,89	8310,33
200	7452,28	6961,98	7774,77	9657,25	15114,46
300	10535,29	9723,64	10917,22	13735,63	21985,89
400	13554,05	12617,40	14056,36	17697,46	28090,16
500	16598,27	15430,81	17155,82	21554,69	34441,75
600	19478,49	18153,36	20216,83	25357,24	40506,90
700	22439,01	20859,34	23142,67	28801,52	46301,25
800	25183,41	23443,41	26192,44	32830,08	51871,79

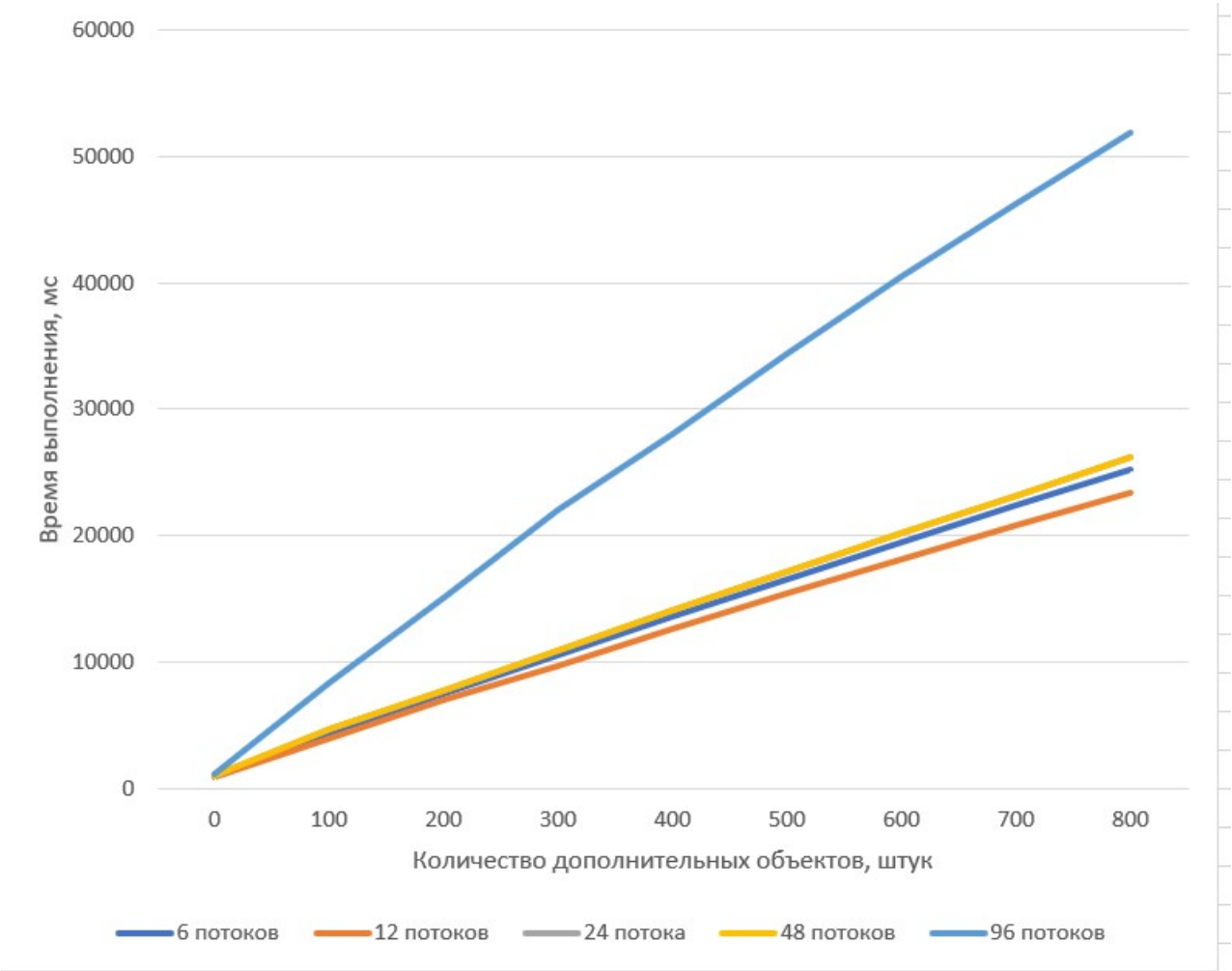

Рисунок 4.5 – Зависимость времени выполнения реализации алгоритма обратной трассировки лучей от количества потоков и количества дополнительных объектов на сцене

При 12 потоках достигается пик, при котором все логические ядра процессора одновременно выполняют трассировку лучей. Далее при увеличении числа потоков производительность падает. Это объясняется тем, что создается очередь потоков, которая замедляет работу программы.

Вывод

В данном разделе было произведено сравнение времени выполнения реализации алгоритма трассировки лучей при последовательной реализации и многопоточной. Результат показал, что выгоднее всего использовать столько потоков, сколько у процессора логических ядер.

Заключение

В ходе выполнения лабораторной работы были решены следующие задачи:

— описаны основные положения базового алгоритма расчёта;

— изучены основы применены для реализации многопоточности на материале трассировки лучей;

— получены практические навыки параллельных вычислений на основе нативных потоков;

— проведен сравнительный анализ параллельной и однопоточной реализации алгоритма трассировки лучей;

— экспериментально подтверждено различие во временной эффективности реализаций однопоточной и многопоточной трассировки лучей;

— описаны и обоснованы полученных результатов;

Поставленная цель достигнута: изучены параллельные вычисления на материале трассировки лучей

Список использованных источников

[1] Краткий обзор языка C# [Электронный ресурс]. Режим доступа: https://learn.microsoft.com/ru-ru/dotnet/csharp/tour-of-csharp/ (дата обращения: 25.09.2022).

[2] Работа с потоками в C [Электронный ресурс]. Режим доступа: https://www.rsdn.org/article/dotnet/CSThreading1.xml(дата обращения 16.11.2022).

[3] Stopwatch Класс [Электронный ресурс]. Режим доступа: https://learn.microsoft.com/ru-ru/dotnet/api/system.diagnostics.stopwatch?view=net-6.0 (дата обращения: 16.11.2022).

[4] AMD Ryzen™ 5 4600H [Электронный ресурс]. Режим доступа: https://www.amd.com/ru/products/apu/amd-ryzen-5-4600h (дата обращения: 25.09.2022).