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»

Лабораторная работа № 4
по дисциплине ”Анализ алгоритмов”

Тема Паралелизация трассировщика лучей

Студент Калашников С.Д.

Группа ИУ7-53Б

Преподаватель Волкова Л.Л., Строганов Ю.В.

Москва, 2022

СОДЕРЖАНИЕ

Введение

В современном мире большое внимание уделяется различным алгоритмам преобразования трехмерных объектов в двумерное пространство экрана. Их принято разделять на две ветви.

1. Алгоритмы растеризации.
2. Алгоритм трассировки лучей.

В данной работе будет использован алгоритм трассировки лучей.

Целью данной лабораторной работы является анализ времени работы реализации алгоритма параллельной трассировки лучей и последовательной.

Для достижения поставленной цели требуется решить ряд задач:

1) разработать последовательный алгоритм визуализации и вывода на экран заданной трехмерной модели с использованием трассировки лучей;
2) разработать параллельную версию алгоритма;
3) реализовать параллельный и последовательный алгоритмы трассировки лучей;
4) разработать программное обеспечение для вывода на экран заданной трехмерной модели с использованием трассировки лучей;
5) провести замеры времени для каждой из реализаций алгоритма;
6) выполнить анализ полученных результатов;
7) по итогам работы составить отчет.

1. Аналитическая часть

1.1 Описание алгоритма трассировки лучей

Первые упоминания о трассировке лучей относятся к шестидесятым годам двадцатого века. Однако широкое распространение данный метод рендера получил относительно недавно. Связанно это с ростом производительности графических процессоров.

Алгоритм трассировки лучей работает идентично художнику рисующему картину на холсте. В связи с этим определим основные положения: наблюдатель — точка, из которой проводится наблюдение, может свободно перемещаться по сцене; холст — бесконечная плоскость перпендикулярная взгляду наблюдателя.

Из глаза наблюдателя выпускается луч, который проходит по каждой точке холста.

В данном алгоритме принято выделять четыре основных шага. Шаги 3–4 повторяются для каждого пикселя из холста.

1. Поместить наблюдателя и рамку в необходимое место.
2. Определить квадрат сетки, соответствующий текущему пикселю.
3. Определить цвет, видимый через этот квадрат.
4. Закрасить пиксель этим цветом.

2. Конструкторская часть

2.1 Описание алгоритма

Рассмотрим подробнее каждый шаг.

На первом шаге устанавливаются размеры холста, расстояние от наблюдателя до холста и местоположение наблюдателя в трехмернном пространстве.

Второй шаг не представляет реальной сложности из-за условий расположения холста. Для перехода от координат рамки к координатам пространства необходимо изменить масштаб. Пусть C_x, C_y координаты пикселя на холсте по осям X, Y соответственно. Тогда координаты точки в пространстве будут вычислятся по следующей формуле, где V_w, V_h и C_w, C_h размеры по ширине и высоте для пространства и холста соответсвенно:

$$V_x = C_x * \frac{V_w}{C_w}$$

$$V_y = C_y * \frac{V_h}{C_h}$$

Третий шаг самый сложный и трудоемкий в данном алгоритме. Для определения цвета необхлдимо определить какая фигура находится ближе всего к наблюдателю в данной точке. Для этого используются параметрические уравнения. Для луча, исходящего от наблюдателя оно задается следующим образом:

$$P = O + t(V - O)$$

где O — положение наблюдателя, V — положение точки в пространстве, t — произвольное действительное число. Для оптимизации алгоритма принято простейщие примитивы описывать в сферы. Уравнение сферы для P — точки на сфере, C — центра сферы и r — радиуса сферы принимает вид:

$$< P - C, P - C >= r^2$$

Тогда для нахождения точки пересечения необходимо решить квадратное уравнение, где $\vec{D} = V - O$

$$t^2 < \vec{D}, \vec{D} > +2t < \vec{D}, \vec{OC} > + < \vec{OC}, \vec{OC} > -r^2 = 0$$

В случае, если объектом является не сфера, то вычисления становятся более трудоемкими и более затратными по времени.

2.2 Параллелизация алгоритма

В связи с тем, что данный алгоритм проходит по всем пикселям холста, то с увеличением размеров холста увеличивается и время работы алгоритма.

Разные части изображения могут обрабатываться независимо от других. По этой причине можно обрабатывать части холста одновременно, используя потоки, таким образом уменьшая общее время работы реализации алгоритма.

Разделим холст по оси X на количество логических ядер процессора. Именно такое количество потоков будет оптимально. Для передачи данных в поток создадим структуру, в которой будут содержаться начало и конец текущего отрезка для обработки.

3. Технологическая часть

В данном разделе будут рассмотрены средства реализации, а также представлены листинги сортировок.

3.1 Средства реализации

В данной работе для реализации был выбран язык программирования *c#*. В текущей лабораторной работе требуется замерить время выполнения реализаци трассировщика лучей.

Время работы было замерено с помощью класса *Stopwatch* из библиотеки $System.Diagnostics$ [1].

Для создания потоков был выбран класс *Tread* из библиотеки $System.Threading$.

3.2 Сведения о модулях программы

Программа состоит из следующих модулей:

- $Scene.cs$ - файл, содержащий сцену;
- $Form1.cs$ - файл, реализующий интерфейс программы;
- $RayTraiser.cs$ - файл, содержащий трассировку лучей.

3.3 Реализация алгоритмов

В данной работе используется параметризированный поток, который на вход

принимает структуру, содержащую начало, конец интервала для трассировки и остальные параметры, требуемые для работы алгоритма.

В листинге 3.1 представлена реализация общей части: функция для расчета пересечения луча и объекта, функция определяющая ближайший объект к наблюдателю, функция построения выпуклой оболочки – для трассировки лучей. В листингах 3.2 и 3.3 представлены реализации параллельной и не паралелльной трассировки лучей соответсвенно. В листинге 3.4 представлена реализация структуры для передачи в поток.

Листинг 3.1. Реализация общей части трассировки лучей

```
1    private Color RayT(Model model, MatrixCoord3D D, MatrixCoord3D
        position)
2 {
3
4   PolygonComponent closest = null;
5   double closest_t = double.MaxValue;
6   //   Parallel.ForEach<PolygonComponent>(model.Polygons, p=> {
7     foreach (PolygonComponent p in model.Polygons)
8     {
9       //    if (MatrixCoord3D.scalar(p.Normal, cam.Direction) > 0)
10      {
11        MatrixCoord3D tt = GetTimeAndUvCoord(position, D, p.Points[0].
             Coords, p.Points[1].Coords, p.Points[2].Coords);
12        if (tt != null)
13        {
14          if (tt.X < closest_t && tt.X > 1)
15          {
16            closest_t = tt.X;
17            closest = p;
18          }
19        }
20      }
21    }
22
23    if (closest == null)
24    return Color.White;
25    return closest.ColorF;
26  }
27  private Tuple<MatrixCoord3D, double> FoundCenter(Container<
       PointComponent> points)
28  {
29    double minX, maxX, minY, maxY, minZ, maxZ;
30    minX = points[0].X;
31    maxX = points[0].X;
32    minY = points[0].Y;
33    maxY = points[0].Y;
34    minZ = points[0].Z;
35    maxZ = points[0].Z;
```

```
36
37      foreach (PointComponent p in points)
38      {
39        if (p.X < minX)
40        minX = p.X;
41        if (p.X > maxX)
42        maxX = p.X;
43        if (p.Y < minY)
44        minY = p.Y;
45        if (p.Y > maxY)
46        maxY = p.Y;
47        if (p.Z < minZ)
48        minZ = p.Z;
49        if (p.Z > maxZ)
50        maxZ = p.Z;
51      }
52      MatrixCoord3D position = new MatrixCoord3D((minX + maxX) / 2f, (
            minY + maxY) / 2f, (minZ + maxZ) / 2f);
53      double r = 0;
54      foreach (PointComponent p in points)
55      {
56        LineComponent minx = new LineComponent(new PointComponent(
              position), p);
57        r = Math.Max(minx.Len(), r);
58      }
59
60      return new Tuple<MatrixCoord3D, double>(position, r);
61    }
62    private double RaySphereIntersection(MatrixCoord3D rayOrigin,
        MatrixCoord3D rayDirection, MatrixCoord3D spos, double r)
63    {
64      double t = Double.MaxValue;
65      //a == 1; // because rdir must be normalized
66      MatrixCoord3D k = rayOrigin - spos;
67      double b = MatrixCoord3D.scalar(k, rayDirection);
68      double c = MatrixCoord3D.scalar(k, k) - r * r;
69      double d = b * b - c;
70      if (d >= 0)
71      {
```

```
72          double sqrtfd = Math.Sqrt(d);
73          // t, a == 1
74          double t1 = -b + sqrtfd;
75          double t2 = -b - sqrtfd;
76          double min_t = Math.Min(t1, t2);
77          double max_t = Math.Max(t1, t2);
78          t = (min_t >= 0) ? min_t : max_t;
79        }
80        return t;
81      }
82
83      private const double Epsilon = 0.000001d;
84
85      private MatrixCoord3D? GetTimeAndUvCoord(MatrixCoord3D rayOrigin,
          MatrixCoord3D rayDirection, MatrixCoord3D vert0, MatrixCoord3D
          vert1, MatrixCoord3D vert2)
86      {
87        var edge1 = vert1 - vert0;
88        var edge2 = vert2 - vert0;
89
90        var pvec = (rayDirection * edge2);
91
92        var det = MatrixCoord3D.scalar(edge1, pvec);
93
94        if (det > -Epsilon && det < Epsilon)
95        {
96          return null;
97        }
98
99        var invDet = 1d / det;
100
101       var tvec = rayOrigin - vert0;
102
103       var u = MatrixCoord3D.scalar(tvec, pvec) * invDet;
104
105       if (u < 0 || u > 1)
106       {
107         return null;
108       }
```

```
109
110     var qvec = (tvec * edge1);
111
112     var v = MatrixCoord3D.scalar(rayDirection, qvec) * invDet;
113
114     if (v < 0 || u + v > 1)
115     {
116       return null;
117     }
118
119     var t = MatrixCoord3D.scalar(edge2, qvec) * invDet;
120
121     return new MatrixCoord3D(t, u, v);
122   }
123
124   private MatrixCoord3D GetTrilinearCoordinateOfTheHit(float t,
        MatrixCoord3D rayOrigin, MatrixCoord3D rayDirection)
125   {
126     return rayDirection * t + rayOrigin;
127   }
128
129   private MatrixCoord3D CanvasToVieport(int x, int y, double aspect,
        double fov)
130   {
131
132     double fx = aspect * fov * (2 * ((x + 0.5f) / screen.Width) - 1);
133     double fy = (1 - (2 * (y + 0.5f) / screen.Height)) * fov;
134
135     return new MatrixCoord3D(fx / scale, fy / scale, -1);
136   }
```

Листинг 3.2. Алгоритм паралельной трассировки

```
1         public override void RayTrasing(Model model)
2 {
3
4   Tuple<MatrixCoord3D, double> sphere = FoundCenter(model.Points);
5   List<Thread> threads = new List<Thread>();
6
7   MatrixCoord3D CamPosition = cam.Position.Coords;
8   MatrixTransformation3D RotateMatrix = cam.RotateMatrix;
9
10  int x = 0;
11  for (int h = 0; h < NumberofThreads; h++)
12  {
13    threads.Add(new Thread(new ParameterizedThreadStart(ByVertecal)));
14    threads[threads.Count - 1].Start(new Limit(x, x + screen.Width /
        NumberofThreads,model,CamPosition, RotateMatrix,sphere ));
15    x += screen.Width / NumberofThreads;
16  }
17  foreach (var elem in threads)
18  {
19    elem.Join();
20  }
21 }
22
23 private void ByVertecal(object obj)
24 {
25  Limit limit = (Limit)obj;
26  Color c;
27  double aspect = screen.Width / screen.Height;
28  double field = Math.Tan(cam.Fovy / 2 * Math.PI / 180.0f);
29  for (int x = limit.begin; x < limit.end; x++)
30  for (int y = 0; y < screen.Height; y++)
31  {
32    MatrixCoord3D D = CanvasToVieport(x, y,aspect,field) * limit.
        RotateMatrix;// new MatrixCoord3D(cam.RotateMatrix.Coeff[0,2],
        cam.RotateMatrix.Coeff[1, 2], cam.RotateMatrix.Coeff[2, 2]);
33    D.Normalise();
34    c = Color.White;
35    if (RaySphereIntersection(limit.CamPosition, D, limit.sphere.Item1,
```

```
            limit.sphere.Item2) != double.MaxValue)
36      {
37        c = RayT(limit.model, D, limit.CamPosition);
38      }
39      PictureBuff.SetPixel(x, y, c.ToArgb());
40    }
41  }
```

Листинг 3.3. Алгоритм последовательной трассировки

```
1         public virtual void RayTrasing(Model model)
2 {
3   Tuple<MatrixCoord3D, double> sphere = FoundCenter(model.Points);
4   MatrixCoord3D CamPosition = cam.Position.Coords;
5   MatrixTransformation3D RotateMatrix = cam.RotateMatrix.InversedMatrix
      ();
6   double aspect = screen.Width / (double)screen.Height;
7   double field = Math.Tan(cam.Fovy / 2 * Math.PI / 180.0f);
8   for (int x = 0; x < screen.Width; x++)
9   {
10    for (int y = 0; y < screen.Height; y++)
11    {
12      MatrixCoord3D D = CanvasToVieport(x, y, aspect, field) *
          RotateMatrix;// new MatrixCoord3D(cam.RotateMatrix.Coeff[0,2],
           cam.RotateMatrix.Coeff[1, 2], cam.RotateMatrix.Coeff[2, 2]);
13      D.Normalise();
14      Color c = Color.White;
15      if (RaySphereIntersection(CamPosition, D, sphere.Item1, sphere.
          Item2) != double.MaxValue)
16      {
17        c = RayT(model, D, CamPosition);
18      }
19      PictureBuff.SetPixel(x, y, c.ToArgb());
20    }
21  }
22 }
```

Листинг 3.4. Структура для передачи в поток

```
1     class Limit
2   {
3     public int begin;
4     public int end;
5     readonly public Model model;
6     readonly public MatrixCoord3D CamPosition;
7     readonly public MatrixTransformation3D RotateMatrix;
8     readonly public Tuple<MatrixCoord3D, double> sphere;
9     public Limit(int begin, int end, Model model, MatrixCoord3D
          CamPosition, MatrixTransformation3D RotateMatrix, Tuple<
          MatrixCoord3D, double> sphere)
10    {
11      this.begin = begin; this.end = end; this.model = model;
12      this.CamPosition = CamPosition; this.RotateMatrix = RotateMatrix.
            InversedMatrix();
13      this.sphere = sphere;
14    }
15  }
```

4. Исследовательская часть

В данном разделе будут приведены примеры работы программы, а также проведен сравнительный анализ алгоритмов при различных ситуациях на основе полученных данных.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры времени, представлены далее:

- операционная система Windows 11 Pro Версия 22H2 (22621.674) [2];
- оперативная память 16 ГБ;
- процессор 11th Gen Intel(R) Core(TM) i5-11400, 2.59 ГГц [3].

При тестировании компьютер был включен в сеть электропитания. Во время замеров времени устройство было нагружено только встроенными приложениями окружения, а также системой тестирования.

4.2 Демонстрация работы программы

На рисунке 4.1 представлен результат работы программы. На экран выводится результат работы программы.

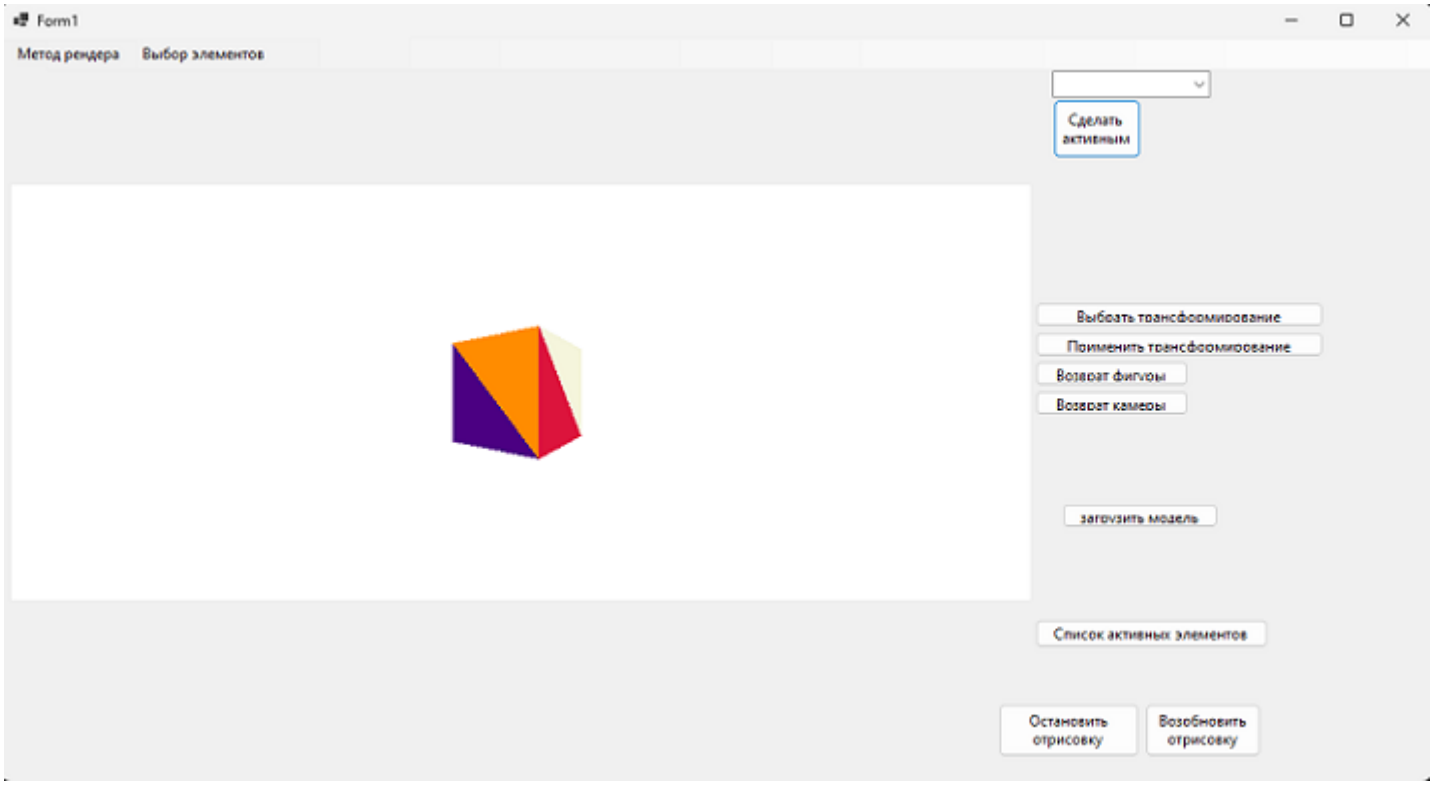

Рис. 4.1. Пример работы программы

4.3 Время выполнения реализаций алгоритмов

Используются функция замера времени Start() и Stop() из библиотеки $System.Diagnostics$. Для получения времени работы реализаций алгоритма алгоритма используется свойство Elapsed того же класса.

На рисунке 4.2, приведены графические результаты замеров времени работы трассировки лучей для параллельного (при количестве потоков, равном количеству логических ядер ЭВМ, на которой проводились замеры времени, затрачиваемого реализацией трассировки лучей) и последовательного случаев при разной доле заполнения экрана.

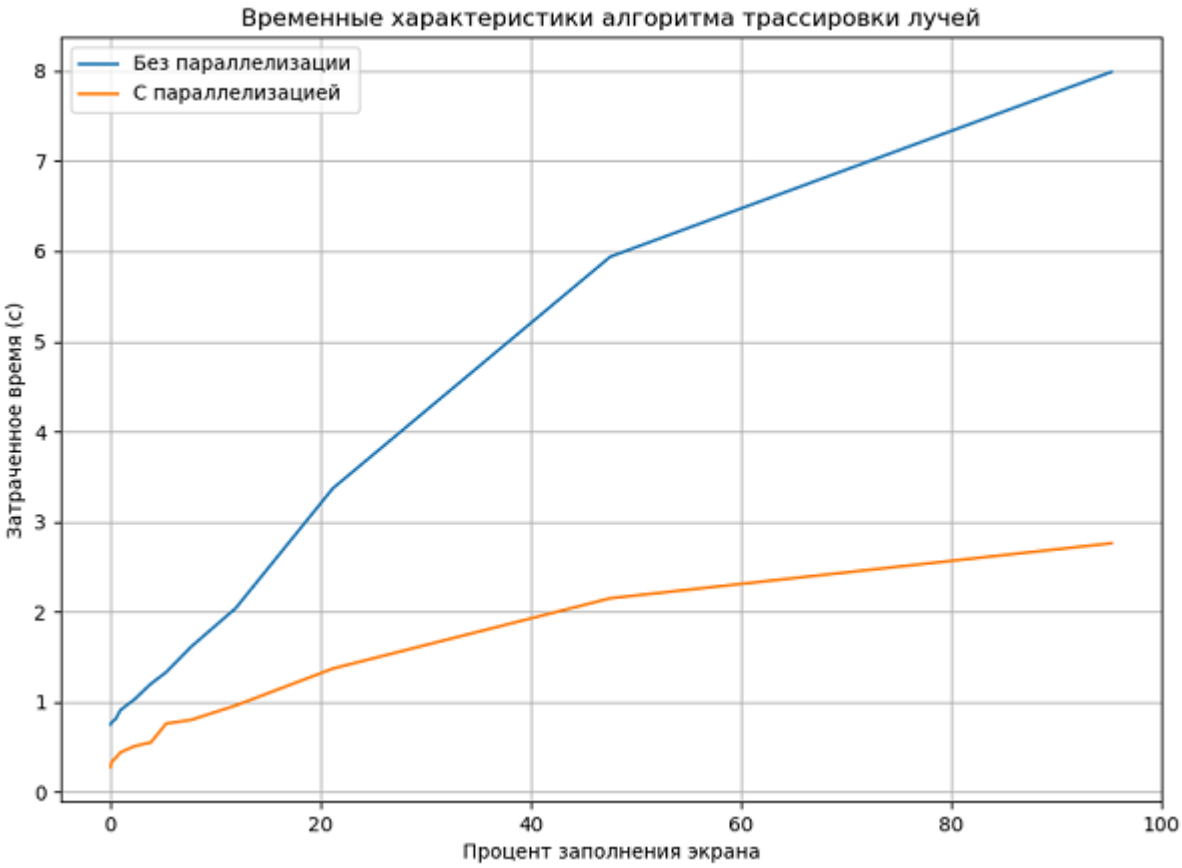

Рис. 4.2. Время работы реализаций алгоритма

Вывод

Исходя из полученных результатов, трассировка лучей с параллелизацией оказалась быстрее в 2.6 раза на большем заполнении экрана и в 2 раза на меньшем.

Заключение

Цель, которая была поставлена в начале лабораторной работы, была достигнута: выполнен анализ времени работы реализации алгоритма параллельной трассировки лучей и последовательной.

Решены все поставленные задачи:

1) разработан последовательный алгоритм визуализации и вывода на экран заданной трехмерной модели с использованием трассировки лучей;
2) разработана параллельную версию алгоритма;
3) реализованы параллельный и последовательный алгоритмы трассировки лучей;
4) разработано программное обеспечение для вывода на экран заданной трехмерной модели с использованием трассировки лучей;
5) проведены замеры времени для каждой из реализаций алгоритма;
6) выполнен анализ полученных результатов;
7) по итогам работы составлен отчет.

В ходе проделанной работы было выявлено, что параллельная трассировка лучей работает гораздо быстрее чем последовательная.

Список использованных источников

[1] Stopwatch [Эл. ресурс]. Режим доступа: https://clck.ru/32NCYi (дата обращения: 13.10.2022).

[2] Windows 11, version 22H2 [Эл. ресурс]. Режим доступа: https://clck.ru/32NCXx (дата обращения: 14.10.2022).

[3] Процессор Intel® Core™ i7 [Эл. ресурс]. Режим доступа: https://clck.ru/yeQa8 (дата обращения: 14.10.2022).