МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Ижевский государственный технический университет имени М.Т.Калашникова»
Кафедра «Автоматизированные системы обработки информации и управления»

ОТЧЕТ

по лабораторной работе №2
по дисциплине «Прикладная механика 2. Геометрия сплайнов в анимациях и формообразовании»
на тему «Алгоритм упрощения ломаной линии Рамера-Дугласа-Пекера»
Вариант-24

Выполнил Рафигов Д.А.
студентка группы Б18-783-1

Проверил Соловьева А. Н.
к.т.н., доцент кафедры АСОИУ

Ижевск 2021

1 Постановка задачи

Разработать программу, демонстрирующую работу алгоритма упрощения ломаной линии Рамера-Дугласа-Пекера на линии, которую задаёт мышью пользователь.

Пользователь проводит на форме линию, удерживая нажатой кнопку мыши. Входными данными для алгоритма является последовательность точек (x, y), полученных при перемещениях мыши (MouseMove), образующих *начальную ломаную*, и параметр $\varepsilon > 0$, задающий максимально допустимое расстояние между начальной и упрощённой ломаными.

Результатом работы алгоритма является *упрощённая ломаная*, заданная подмножеством точек, которые определяются из исходной ломаной.

В программе нужно предусмотреть интерактивную перерисовку упрощённой ломаной при изменении параметра ε.

2 Описание алгоритмов работы программы

Программа была создана в среде разработки Visual Studio из шаблона «Приложение Windows Forms».

В разработанном приложении после нажатия на кнопку применить , считываются точки первой ломанной, которую пользователь рисует мышью в pictureBox1, и передаются вместе с параметром, задающим максимально допустимое расстояние между начальной и упрощённой ломаными, в алгоритм упрощения ломаной линии Рамера-Дугласа-Пекера. После преобразования получаем новую картинку , которая выводится на pictureBox2.

2.1 Описание алгоритм обработки перемещения мыши

Алгоритм обработки перемещения мыши представлен на рисунке 2.1.

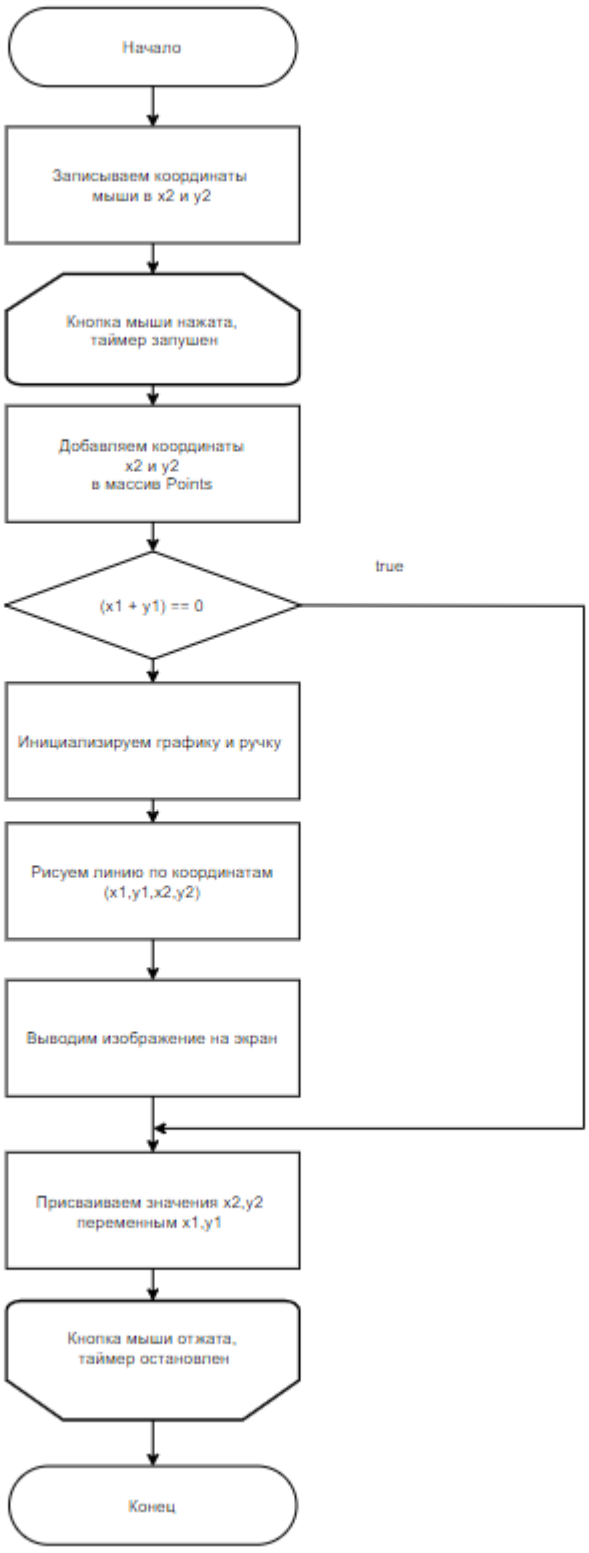

Рисунок 2.1 – Алгоритм обработки перемещения мыши

2.2 Описание алгоритма расчёта и отображения упрощённой ломаной

Алгоритма расчёта и отображения упрощённой ломаной представлен на рисунке 2.2.

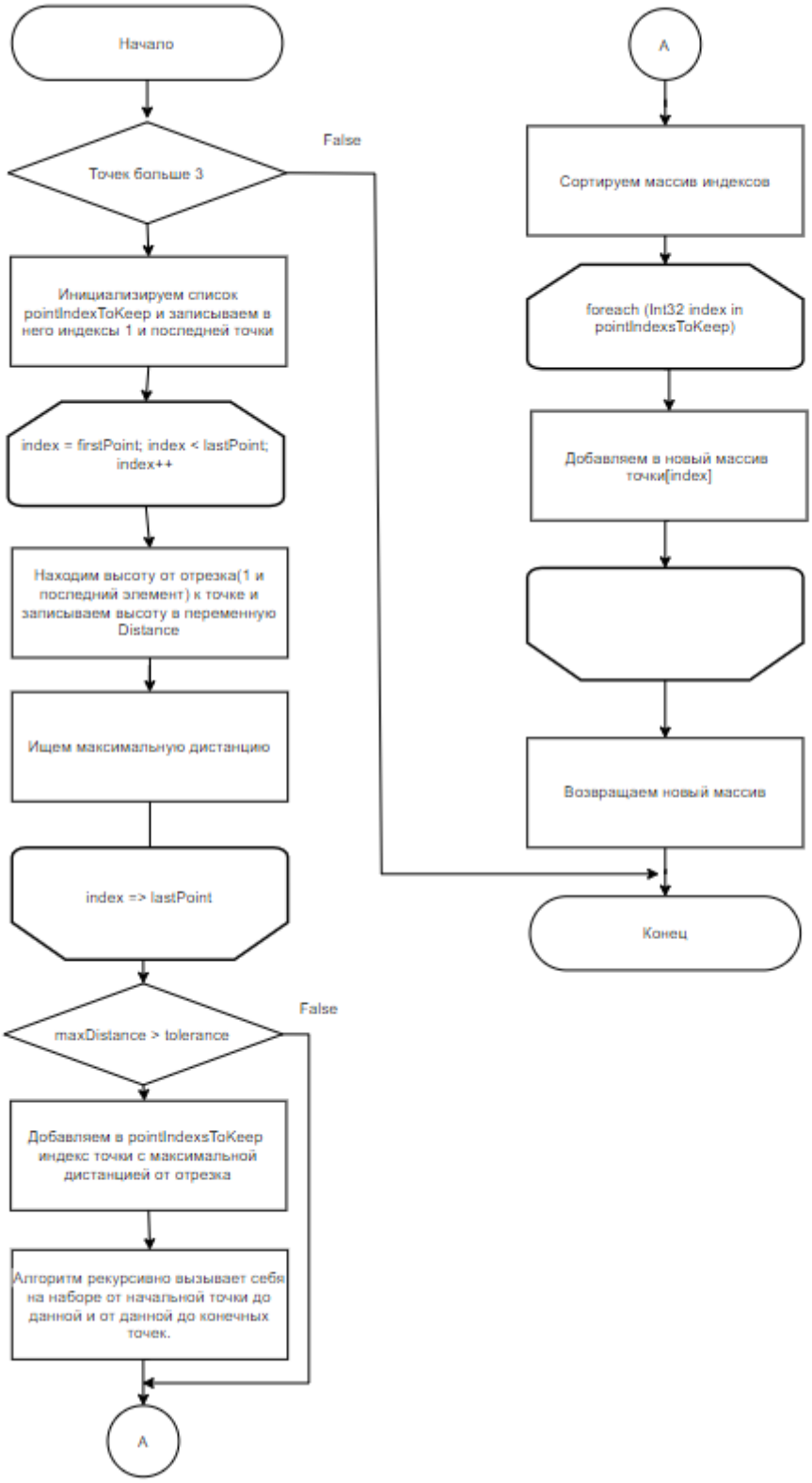

Рисунок 2.2 – Алгоритма расчёта и отображения упрощённой ломаной

.

3 Текст программы C#

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WinFormsApp2
{
    public partial class Form1 : Form
    {

        int x1, y1, x2, y2;
        private Bitmap bmp;
        private Bitmap bmp2;
        private Pen blackPen;
        private Pen greenPen;
        List<Point> points = new List<Point>();
        List<Point> points2 = new List<Point>();
        private void timer1_Tick(object sender, EventArgs e)
        {
            reset();
        }

        private void pictureBox1_MouseDown(object sender, MouseEventArgs e)
        {
            timer1.Start();
        }

        private void pictureBox1_MouseUp(object sender, MouseEventArgs e)
        {
            timer1.Stop();
        }

        public Form1()
        {
            InitializeComponent();
            bmp = new Bitmap(pictureBox1.Width, pictureBox1.Height);
            bmp2 = new Bitmap(pictureBox2.Width, pictureBox2.Height);
            blackPen = new Pen(Color.Blue, 1);
        }

        private void pictureBox1_MouseMove(object sender, MouseEventArgs e)
        {
            x2 = Convert.ToInt32(e.X); // координата по оси X
            y2 = Convert.ToInt32(e.Y);

        }

        private void reset()
        {
            //x2 = Convert.ToInt32(e.X); // координата по оси X
            //y2 = Convert.ToInt32(e.Y); // координата по оси Y
            points.Add(new Point(x2, y2));
            this.Invalidate();

            if ((x1 + y1) == 0)
```

```
            {

                x1 = x2;
                y1 = y2;
            }
            else
            {
                Graphics g = Graphics.FromImage(bmp);

                blackPen = new Pen(Brushes.Red, 2);
                g.DrawLine(blackPen, x1, y1, x2, y2);

                blackPen.Dispose();
                g.Dispose();
                pictureBox1.Image = bmp;
                pictureBox1.Invalidate();

                x1 = x2;
                y1 = y2;
            }


        }

        private void button1_Click(object sender, EventArgs e)
        {

            greenPen = new Pen(Brushes.Green, 2);
            Graphics br = Graphics.FromImage(bmp2);
            br.Clear(Color.White);
            points2 = DouglasPeuckerReduction(points, Convert.ToDouble(
numericUpDown1.Value));
            for (int i = 1; i < points2.Count; i++)
            {
                br.DrawLine(greenPen, points2[i - 1], points2[i]);

            }
            pictureBox2.Image = bmp2;
            pictureBox2.Invalidate();

        }


        public static List<Point> DouglasPeuckerReduction (List<Point> Points, Double
Tolerance)
        {
            if (Points == null || Points.Count < 3)
                return Points;

            Int32 firstPoint = 0;
            Int32 lastPoint = Points.Count - 1;
            List<Int32> pointIndexsToKeep = new List<Int32>();

            pointIndexsToKeep.Add(firstPoint);
            pointIndexsToKeep.Add(lastPoint);

            while (Points[firstPoint].Equals(Points[lastPoint]))
            {
                lastPoint--;
            }

            DouglasPeuckerReduction(Points, firstPoint, lastPoint,  Tolerance, ref
pointIndexsToKeep);

            List<Point> returnPoints = new List<Point>();
```

```
            pointIndexsToKeep.Sort();
            foreach (Int32 index in pointIndexsToKeep)
            {
                returnPoints.Add(Points[index]);
            }

            return returnPoints;
        }

        private static void DouglasPeuckerReduction(List<Point>points, Int32 firstPoint,
Int32 lastPoint, Double tolerance, ref List<Int32> pointIndexsToKeep)
        {
            Double maxDistance = 0;
            Int32 indexFarthest = 0;

            for (Int32 index = firstPoint; index < lastPoint; index++)
            {
                Double distance = PerpendicularDistance
                    (points[firstPoint], points[lastPoint], points[index]);
                if (distance > maxDistance)
                {
                    maxDistance = distance;
                    indexFarthest = index;
                }
            }

            if (maxDistance > tolerance && indexFarthest != 0)
            {
                pointIndexsToKeep.Add(indexFarthest);

                DouglasPeuckerReduction(points, firstPoint,
                indexFarthest, tolerance, ref pointIndexsToKeep);
                DouglasPeuckerReduction(points, indexFarthest,
                lastPoint, tolerance, ref pointIndexsToKeep);
            }
        }


        public static Double PerpendicularDistance (Point Point1, Point Point2, Point
Point)
        {


            Double area = Math.Abs(.5 * (Point1.X * Point2.Y + Point2.X *
            Point.Y + Point.X * Point1.Y - Point2.X * Point1.Y - Point.X *
            Point2.Y - Point1.X * Point.Y));
            Double bottom = Math.Sqrt(Math.Pow(Point1.X - Point2.X, 2) +
            Math.Pow(Point1.Y - Point2.Y, 2));
            Double height = area / bottom * 2;

            return height;


        }

    }
}
```

4 Пример работы программы

На рисунке 4.1 представлен интерфейс разработанной программы.

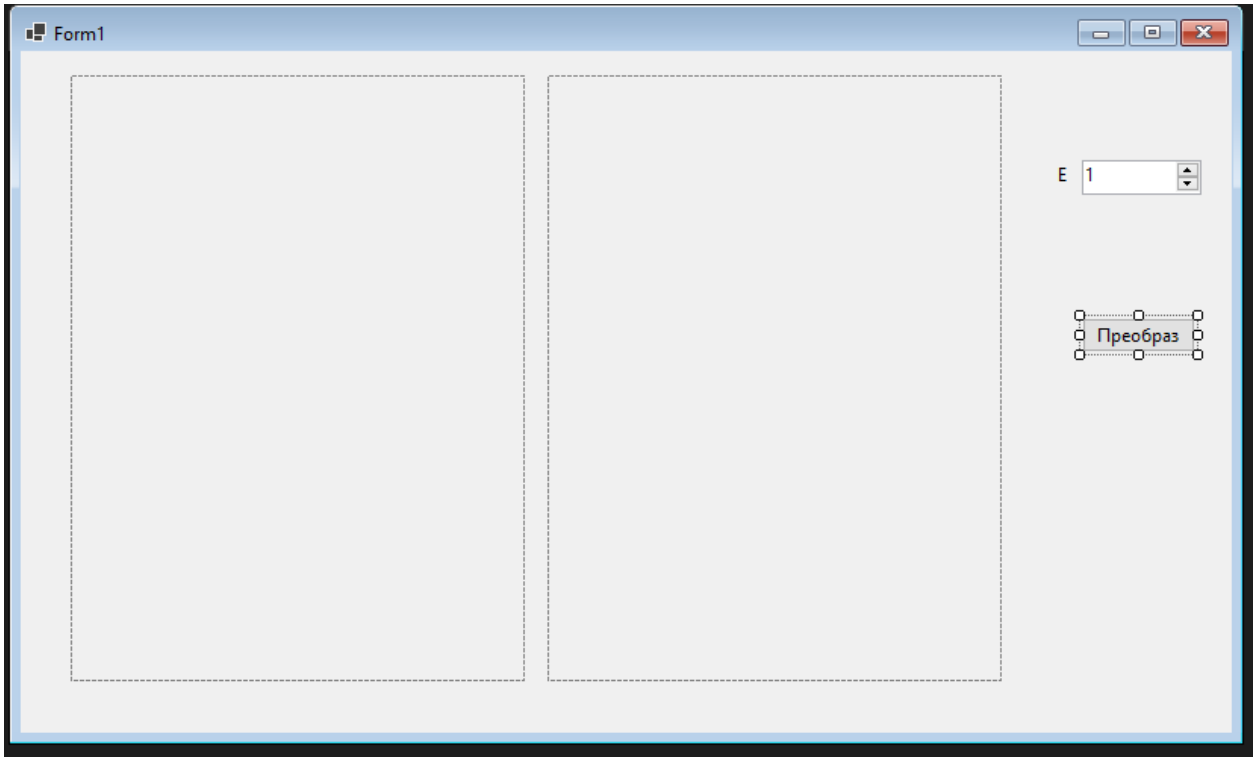

Рисунок 4.1 – Интерфейс

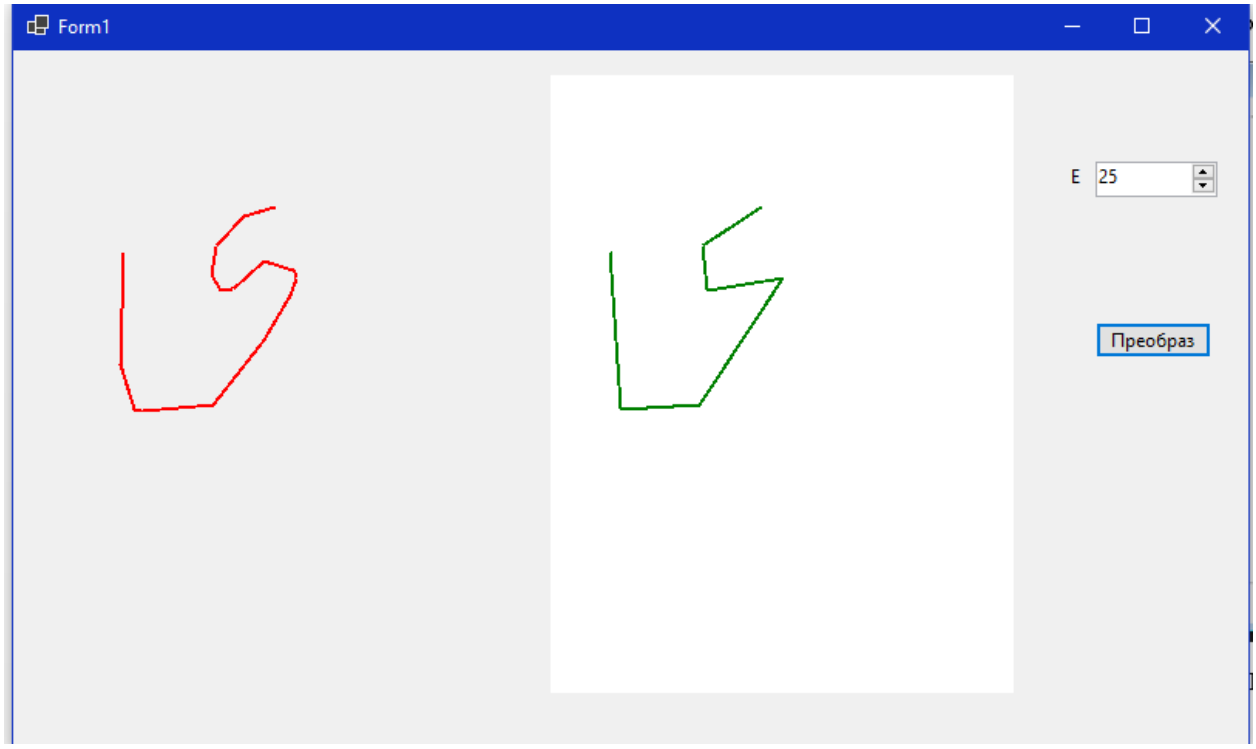

Рисунок 4.2 – Упрощенная линия