Минобрнауки РФ

ФГБОУ ВО «Ярославский государственный университет

имени П.Г. Демидова»

Кафедра теории функций и функционального анализа

КУРСОВАЯ РАБОТА

на тему: «Вычисление максимального в симплексе отрезка данного направления»

Выполнил студент

Группы ПМИ-32БО

Ефимов И.В.

Научный руководитель:

д.ф.м.н., доцент,

Невский М.В.

Ярославль, 2016.

**Содержание**

[Введение 3](#_Toc474143655)

[Числовые характеристики n-мерного симплекса 4](#_Toc474143656)

[Вычисление отрезка максимальной длинны для n=2 5](#_Toc474143657)

[Вычисление отрезка максимальной длинны для произвольного n 7](#_Toc474143658)

[Пример 1 9](#_Toc474143659)

[Описание программы 10](#_Toc474143660)

[Примеры работы программы 11](#_Toc474143661)

[Список литературы 13](#_Toc474143662)

[Приложение А 14](#_Toc474143663)

## Введение

В поставленной передо мной задачей мне необходимо вычислить координаты и максимальную длину отрезка, принадлежащего n-мерному симплексу и параллельного заданному нам вектору. Для того что бы в результате получить искомые данные, на вход нам даются вершины n-мерного симплекса и ненулевой вектор заданные своими координатами. Все вычисления будут происходить в n-мерном пространстве.

Для того что бы получить необходимые результаты в своей работе, я использовал алгоритм “Вычисление максимального в симплексе отрезка данного направления”, разработанный М. В. Невским.

В моей предыдущей курсовой работе я придумал свой собственный алгоритм по нахождению этого отрезка в двумерном случае.

Для вычисления максимального в симплекс отрезка данного направления мной была создана программа на языке программирования C#, с помощью которой я смогу посчитать длину и новые координаты отрезка.

# Числовые характеристики n-мерного симплекса

В этом пункте, я дам определения всем данным которые используется в алгоритме вычисления максимального в симплексе отрезка данного направления.

Для начала введем определение осевого диаметра выпуклого тела: Пусть С - выпуклое тело в. Обозначим через максимальную длину отрезка, содержащегося в *C* и параллельного оси . Величину будем называть *i-м* осевым диаметром *C.*

Теперь введем некоторые необходимые нам обозначения: Пусть *S* —невырожденный симплекс в и *v* —ненулевой *n*-мерный вектор. Обозначим через максимальную длину отрезка, принадлежащего *S* и параллельного *v*. В случае когда *v* коллинеарен *i*-й координатной оси, положим :=. Обозначим через = вершины *S*. Пусть барицентрические координаты точки относительно *S*. Они определяются свойствами:

Симплекс *S* задаётся неравенствами точки его -мерных граней удовлетворяют уравнениям Рассмотрим матрицу

По определению положим Тогда справедливы равенства

# Вычисление отрезка максимальной длинны для n=2

Данный алгоритм отсекает отрезок  , с помощью его параметрического представления:

Для начала рассмотрим отдельно ребро    у отсекающего треугольника. Нормаль к нему ориентируем во внешнюю сторону отсекающего треугольника, для этого будет удобно считать, что точки отсекающего контура обходятся против часовой стрелки; тогда если ребро – это  , то нормаль будет пропорциональна

Из этого следует что область, образуемая при отсечении прямой, на которой лежит ребро (обозначим ее ), соответствует точкам  P, для которыхскалярное произведение

>0

где  - любая точка на ребре  . Точка пересечения прямой, на которой лежит отрезок с отсекающей прямой , находится из уравнения

Разрешая его, получаем, что



В случае если  . Если же

это означает, что отсекаемый отрезок параллелен  и не существует единственной точки их пересечения. Такие случаи алгоритм игнорирует.

Для алгоритма Цируса-Бека также важно, в каком направлении (внутрь отсекающего треугольника или из него) проходит точка при движении по отрезку от   к  , т.е. при изменении t от 0 до 1. Это определяется знаком . Будем обозначать такие точки пересечения как:

Потенциально входящие (ПВх): < 0

Потенциально выходящие (ПВых): > 0

После того как рассчитаны координаты    для всех возможных пересечений с прямыми  , следует выбрать максимальную координату из потенциально входящих пересечений  и минимальную из потенциально выходящих . Если прямая, на которой лежит отрезок  , пересекает отсекающий многоугольник, то  < . В этом случае, если пересечение   непустое, то будет искомым отсеченным отрезком.

После того как алгоритм сработает для каждой из трех отрезков выходящих из вершин треугольника мы получим девять точек входа и выхода, три для каждого из отрезков.

## Вычисление отрезка максимальной длинны для произвольного n

Теперь нам необходимо отсортировать эти точки так что бы в этоге остался один отрезок выходящий из вершины двумерного симплекса с координатой одной из вершин и одной точкой входа которая будет служить послдней точкой этого орезка в треугольнике. Для этого нам необхоимо определить не лежит ли точка на каждом ребре симплекса. Тоесть для каждой из точек входа и выходы мы должны проверить условие на принадлежность к каждому ребру.

Пусть точки концы нашего ребра. Необходимым условием принадлежности точки отрезку является ее принадлежность прямой проходящей через , . Далее нам нужно определить лежит ли точка между точками  и , для этого мы воспользуемся скалярным произведение векторов только на этот раз других: . Где М точка входа или входа. Если оно меньше либо равно нулю, то точка лежит на отрезке, иначе вне отрезка.

Но для начала нам необходимо посчитать для каждой точки и вершины треугольника, координаты вектора с помощью формулы:

Теперь, когда мы знаем все координаты векторов, мы посчитаем три скалярных произведения для первой точки каждой прямой, что бы проверить, где она лежит. Если хотя бы одно из трех скалярных произведений получились то мы считаем следующие три скалярных произведения для второй точки. Если же все три значения больше нулю то данная прямая нам не подходит, так как точка входа или выхода не лежит на ребре треугольника.

В нужном нам отрезке две точки выхода будут находиться в одной и той же вершине, а точка входа будет лежать на противоположном ребре. Поэтому если первая и вторая точка не будут равны между собой, нам будет достаточно посчитать скалярные произведения для третей точки, и если, как и раньше хотя бы одно из них будет меньше либо равно нулю, то все три точки будут лежать на ребрах данного треугольника и данная прямая состоящая из вершины и точки входа будет нашим отрезком.

Если же первая и вторая точка равны то, посчитав скалярные произведения для второй точки, мы определим ее принадлежность к нашим ребрам, если принадлежит то это необходимый нам отрезок, если вторая точка не принадлежит ни одному из ребер, то данный отрезок не является искомым.

# Пример 1

Пусть A,B,C вершины треугольника, а концы вектора

A = (60,120) B = (200,100/40) C= (280,190)

= (40,50) = (10,240)

Переносим первую точку вектора в каждую вершину, а вторую на такое же расстояние и применяем алгоритм Цируса-Бека для трех получившихся отрезков.

Получаем две точки выхода и одну точку входа для каждого отрезка.

=(60,120) =(111.47,-206.02) = (60,120)

=(200,2) =(200,2) = (175.56,156.76)

=(326.52,-104.64) =(280,190) = (290,190)

Получив точки, считаем координаты вектора для:

(A, ) ; (B, ) ; (C, ) ; (A, ) ; (B, ) ; (C, ) ;

(A, ) ; (B, ) ; (C, )

Теперь считаем скалярное произведение для того что бы определить принадлежность той или иной точки к нашему треугольнику. Посчитав мы определим, что принадлежит ребру AC, а и есть точки вершины, и теперь мы можем построить максимальный отрезок в симплексе данного направления с координатами , и длинной d :



= (200,2) = (175.56,156.76)



## Описание программы

Задача моей программы состоит в том, чтобы вычислить максимальный в симплексе отрезка данного направления.

Первый шаг – Координаты вершин треугольника и координаты точек вектора задаются в коде программы и хранятся в двумерном массиве *pointsOfPolygon*(вершины) и *p*(точки вектора).

Второй шаг – Переносим вектор в каждую вершину, для этого найдем расстояние между точкой вектора и каждой вершиной, g – расстояние по x, z – расстояние по y.

Третий шаг – с помощью Алгоритма Цируса-Бека находим точки входа () и точки выхода () для трех отрезков и заносим их в новый массив *pointsIn*.

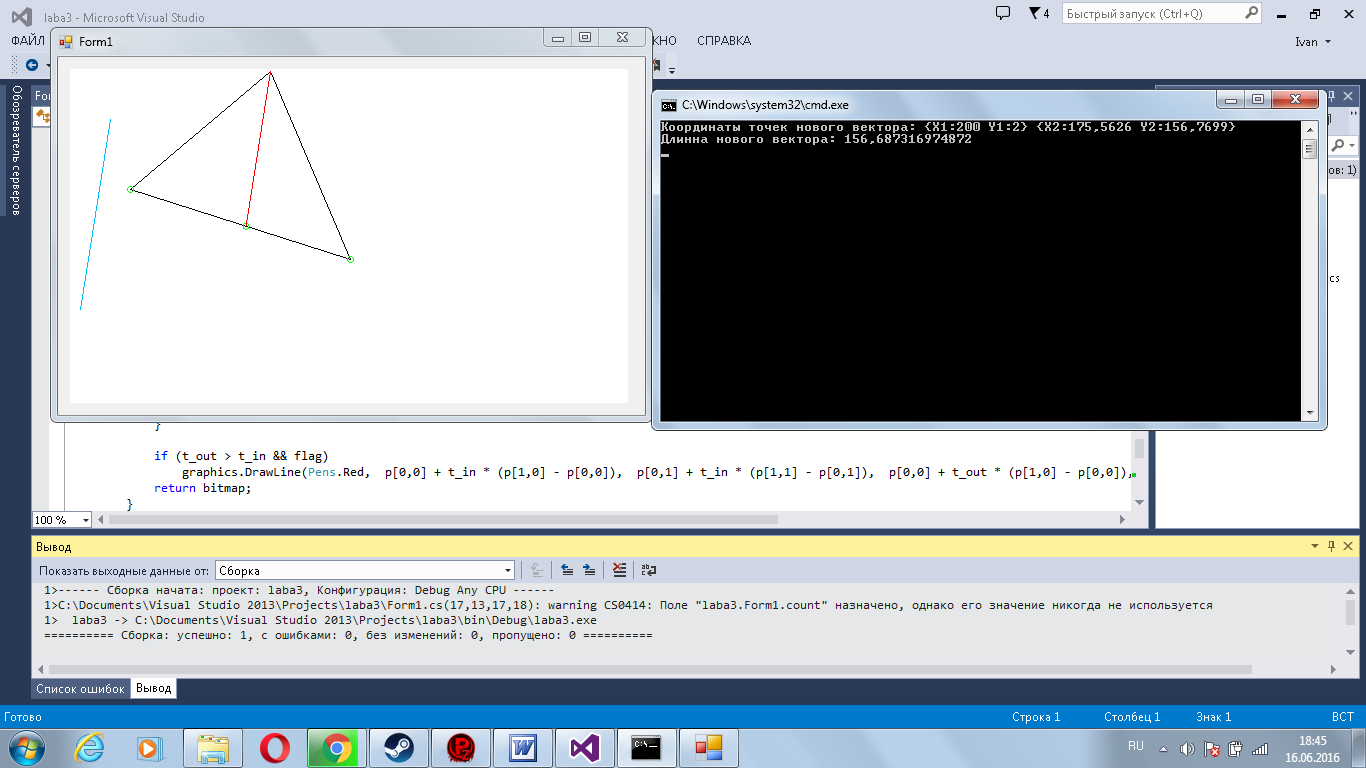
Четвертый шаг – считаем координаты векторов для найденных точек и вершин, результат заносится в массив *vect*.

Пятый шаг – считаем скалярное произведение a, b, c для посчитанных векторов *vect* что бы определить лежит ли точка на одном из ребер треугольника.

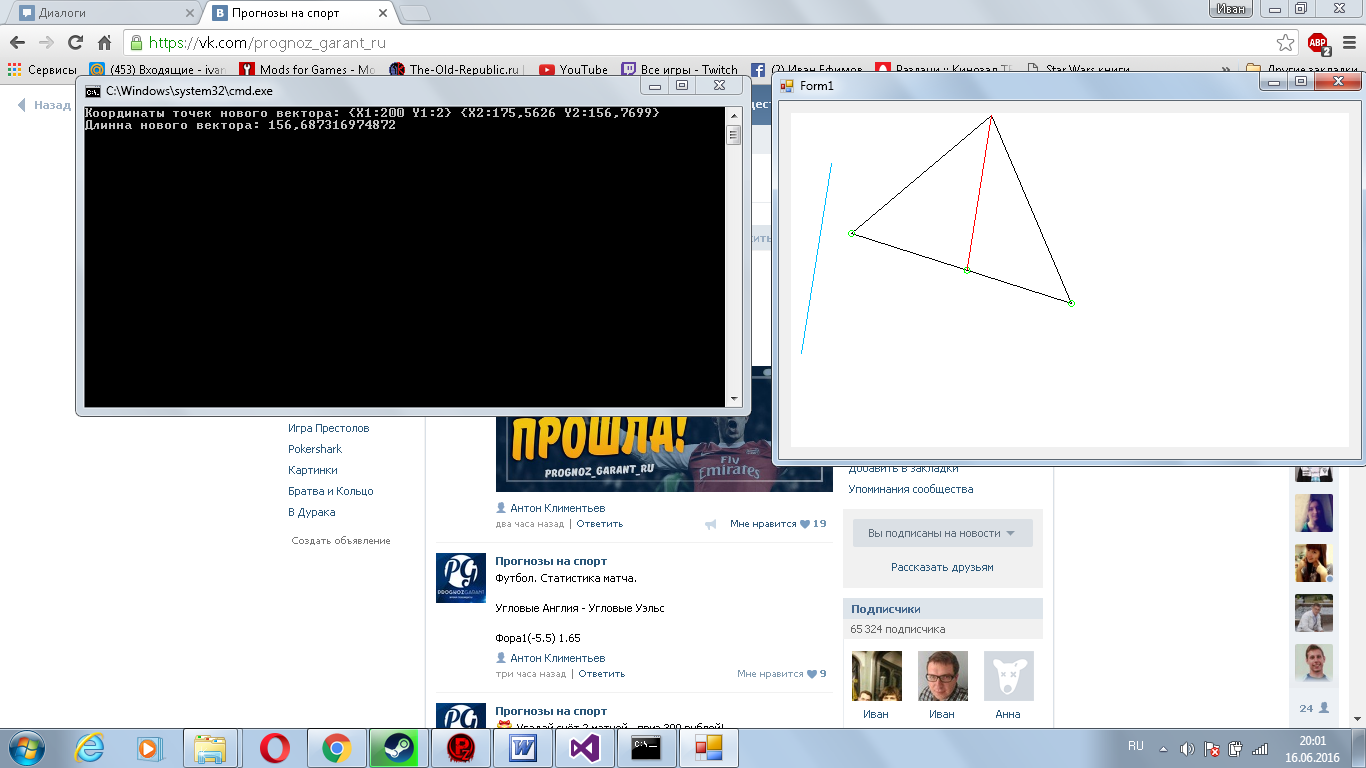
Шестой шаг – найдя необходимую точку, рисуем максимальный отрезок параллельный заданному нам вектору, проходящий через эту точку и одну из вершин, и находим его расстояние *dlina*.

# Примеры работы программы

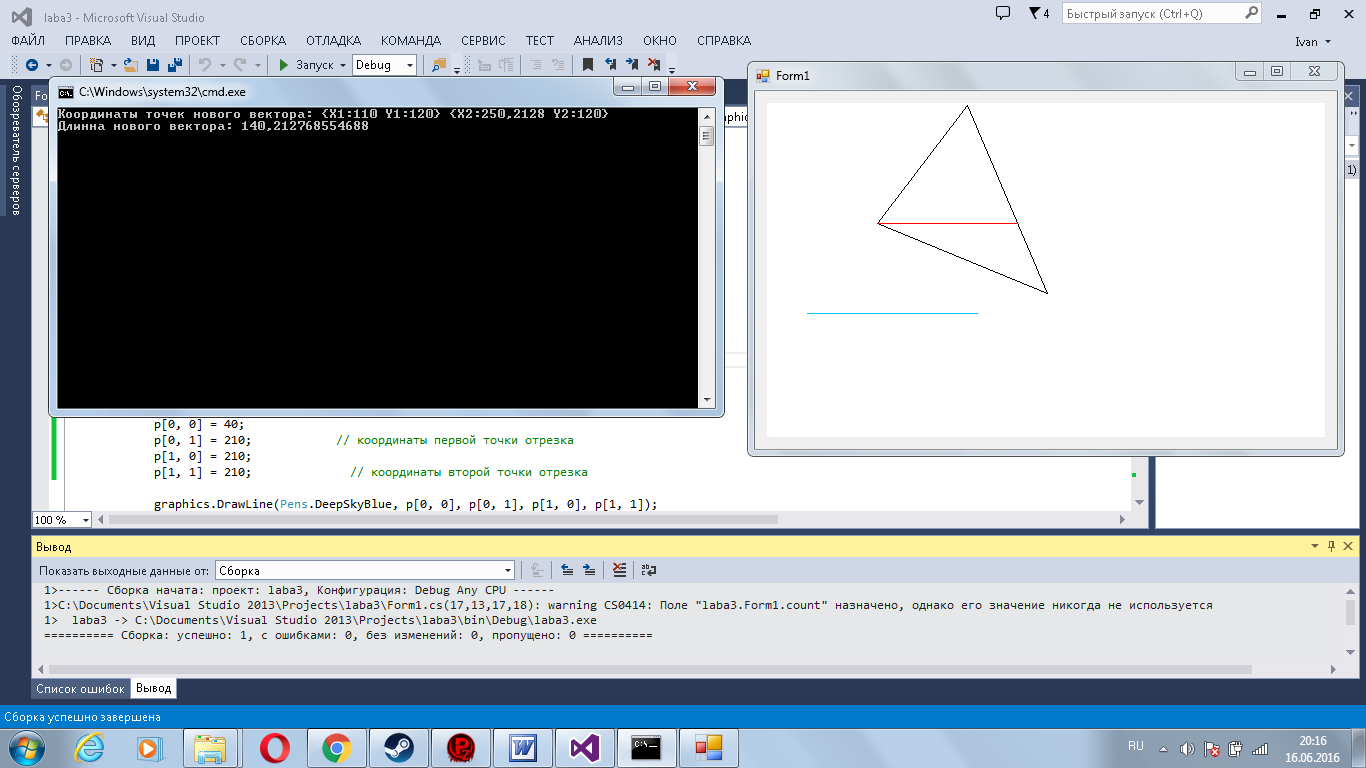
В данном разделе я прикрепляю скриншоты выполнения программы для некоторых векторов и треугольников. Где синим, выделен исходный вектор, а красным максимальный параллельный отрезок



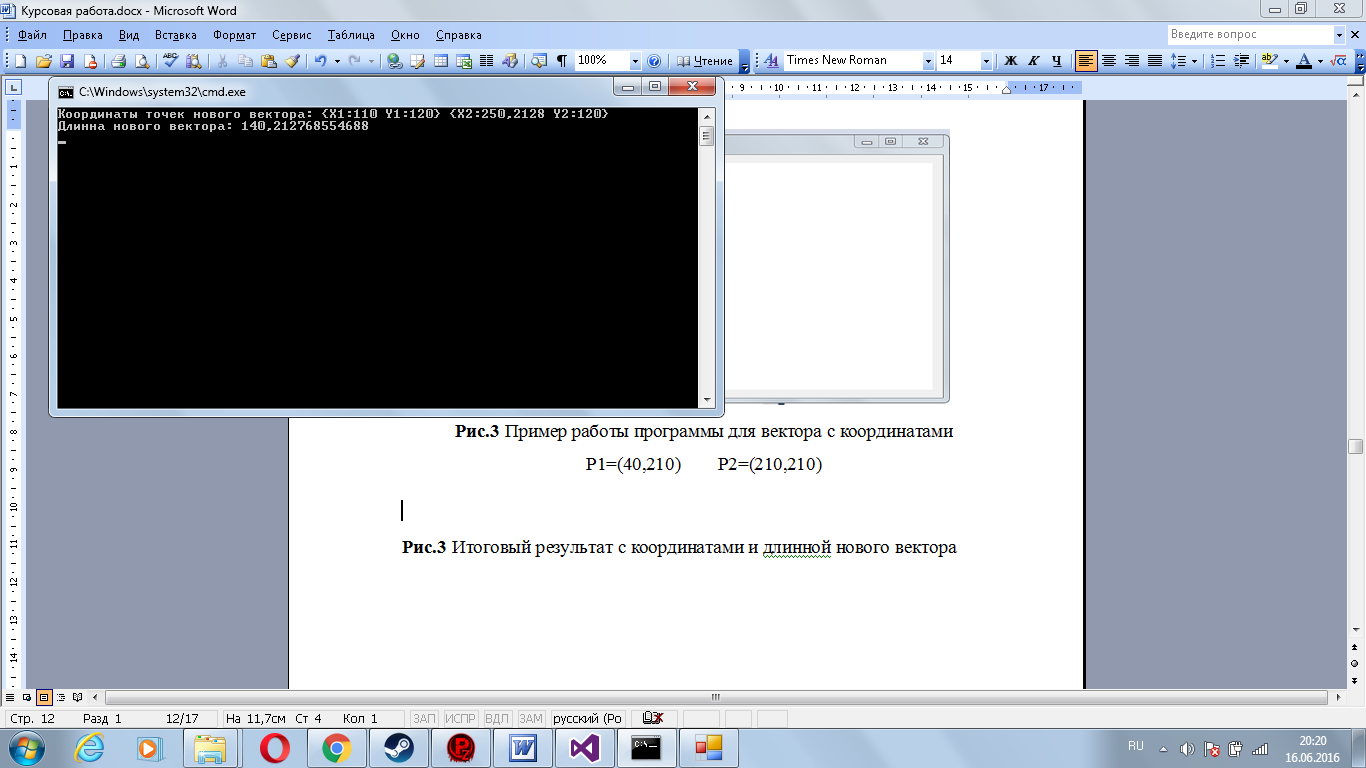
**Рис.1** Пример работы программы для вектора с координатами =(40,50) =(10,240)



**Рис.2** Итоговый результат с координатами и длинной нового вектора



**Рис.3** Пример работы программы для вектора с координатами =(40,210) =(210,210)



**Рис.4** Итоговый результат с координатами и длинной нового вектора

## Список литературы

1. *М. В. Невский,* Вычисление максимального в симплексе отрезка данного направления // Фундаментальная и прикладная математика 2013. — Т.18, № 2. — С.147—152.
2. *Невский М. В.,* Об одном свойстве n-мерного симплекса // Мат. заметки. — 2010. — Т. 87, № 4. — С. 580—593.
3. *http://www.intuit.ru/* , Алгоритмические основы растровой графики, Лекция 5: Отсечение отрезков и многоугольников, Алгоритм Цируса-Бека
4. *Невский М. В.,* Геометрические оценки в полиномиальной интерполяции. — Ярославль: ЯрГУ, 2012.

# Приложение А

**Квадратичная интерполяция на квадрате.**

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 CursLab

{

public partial class Form1 : Form

{

Pen pen = new Pen(Color.Black, 10);

Bitmap bitmap;

Point p1; // первая точка вектора

Point p2; // вторая точка вектора

int count = 0;

public Form1()

{

InitializeComponent();

drawling();

}

public void drawling()

{

pen.Width = 1;

bitmap = new Bitmap(400, 400);

Graphics graphics = Graphics.FromImage(bitmap);

pictureBox1.BackColor = Color.White;

pictureBox1.Image = CirusBek(graphics);

}

public Bitmap CirusBek(Graphics graphics)

{

Point [] pointsOfPolygon={ // Вершины симплекса

new Point {X=40, Y=40},

new Point {X=40, Y=180},

new Point {X=180, Y=80},

new Point {X=40, Y=40}

};

p1 = new Point(200,80); // координаты первой точки отрезка

p2 = new Point(250,50); // координаты второй точки отрезка

graphics.DrawLine(Pens.DeepSkyBlue, p1, p2);

graphics.DrawPolygon(pen, pointsOfPolygon); // рисование многоугольника

float t; // параметр для нахождения t\_in , t\_out

Point D = subtraction(p2, p1); // вычитание координат

float t\_in = 0;

float t\_out = 1;

bool flag=true; // flag - будем ли мы выделять цветом отсечение

float[,] pointsIn = new float[9, 2];

int count = 0;

for (int j = 0; j < pointsOfPolygon.Length-1; j++)

{

int z = p1.Y - pointsOfPolygon[j].Y; //расстояние от первой точки до вершины по y

int g = p1.X - pointsOfPolygon[j].X; // расстояние по х

p1 = new Point(p1.X - g, p1.Y - z);

p2 = new Point(p2.X - g, p2.Y - z);

for (int i = 0; i < pointsOfPolygon.Length; i++)

{

int a2 = (i + 1) % pointsOfPolygon.Length; // прохождение по ребрам против часовйо стрелки, что бы в конце получился вектор (x,0)

int a1 = i % pointsOfPolygon.Length;

Point normal = new Point(pointsOfPolygon[a1].Y - pointsOfPolygon[a2].Y, pointsOfPolygon[a2].X - pointsOfPolygon[a1].X);

float dp = scalarMultiplication(D, normal);

if (dp != 0)

{

t = -scalarMultiplication(subtraction(p1, pointsOfPolygon[i]), normal) / dp;

if (dp < 0) //нормаль и отрезок направлены в разные стороны

{

if (t > t\_in)

t\_in = t;

// заносим точки входа в новый массив точек

pointsIn[count, 0] = resultX(t);

pointsIn[count, 1] = resultY(t);

count++;

}

else

{

if (t < t\_out)

t\_out = t;

// заносим точки выхода в новый массив точек

pointsIn[count, 0] = resultX(t);

pointsIn[count, 1] = resultY(t);

count++;

}

}

else // прямая паралельна ребру

{

Point tmp1 = new Point(pointsOfPolygon[a2].X - pointsOfPolygon[a1].X, pointsOfPolygon[a2].Y - pointsOfPolygon[a1].Y);

Point tmp2 = new Point(pointsOfPolygon[a1].X - p2.X, pointsOfPolygon[a1].Y - p2.Y);

if (tmp1.X \* tmp2.Y - tmp1.Y \* tmp2.X < 0) // считаем определитель матрицы

flag = false;

}

}

}

// считаем координаты векторов для точек и вершин

float[,] vect = new float[54, 2];

for (int i = 0; i < 54; i++)

{

for (int j = 0; j < 3; j++)

{

for (int k = 0; k < 8; k++)

{

vect[i,0]= (pointsOfPolygon[j].X - pointsIn[k, 0]);

vect[i,1] = (pointsOfPolygon[j].Y - pointsIn[k, 1]);

}

}

}

// считаем скалярное произведение

float a = vect[0, 0] \* vect[1, 0] + vect[0, 1] \* vect[1, 1];

float b = vect[2, 0] \* vect[3, 0] + vect[2, 1] \* vect[3, 1];

float c = vect[4, 0] \* vect[5, 0] + vect[4, 1] \* vect[5, 1];

if (a <= 0 || b <= 0 || c <= 0)

{

if (pointsIn[0, 0] == pointsIn[1, 0] && pointsIn[0, 1] == pointsIn[1, 1])

{

a = vect[12, 0] \* vect[13, 0] + vect[12, 1] \* vect[13, 1];

b = vect[14, 0] \* vect[15, 0] + vect[14, 1] \* vect[15, 1];

c = vect[16, 0] \* vect[17, 0] + vect[16, 1] \* vect[17, 1];

if (a <= 0 || b <= 0 || c <= 0)

{

graphics.DrawLine(Pens.Red, pointsIn[0, 0], pointsIn[0, 1], pointsIn[2, 0], pointsIn[2, 1]);

// считаем длину отрезка

double dlina = Math.Sqrt((pointsIn[2, 0] - pointsIn[0, 0]) \* (pointsIn[2, 0] - pointsIn[0, 0]) + (pointsIn[2, 1] - pointsIn[0, 1]) \* (pointsIn[2, 1] - pointsIn[0, 1]));

Console.WriteLine("Координаты точек нового вектора: {X1:" + pointsIn[0, 0] + " Y1:" + pointsIn[0, 1] + "} {X2:" + pointsIn[2, 0] + " Y2:" + pointsIn[2, 1] + "}");

Console.WriteLine("Длинна нового вектора: " + dlina);}}

else

{

a = vect[6, 0] \* vect[7, 0] + vect[6, 1] \* vect[7, 1];

b = vect[8, 0] \* vect[9, 0] + vect[8, 1] \* vect[9, 1];

c = vect[10, 0] \* vect[11, 0] + vect[10, 1] \* vect[11, 1];

if (a <= 0 || b <= 0 || c <= 0)

{

graphics.DrawLine(Pens.Red, pointsIn[0, 0], pointsIn[0, 1], pointsIn[1, 0], pointsIn[1, 1]);

double dlina = Math.Sqrt((pointsIn[1, 0] - pointsIn[0, 0]) \* (pointsIn[1, 0] - pointsIn[0, 0]) + (pointsIn[1, 1] - pointsIn[0, 1]) \* (pointsIn[1, 1] - pointsIn[0, 1]));

Console.WriteLine("Координаты точек нового вектора: {X1:" + pointsIn[0, 0] + " Y1:" + pointsIn[0, 1] + "} {X2:" + pointsIn[1, 0] + " Y2:" + pointsIn[1, 1] + "}");

Console.WriteLine("Длинна нового вектора: " + dlina);}}}

a = vect[18, 0] \* vect[19, 0] + vect[18, 1] \* vect[19, 1];

b = vect[20, 0] \* vect[21, 0] + vect[20, 1] \* vect[21, 1];

c = vect[22, 0] \* vect[23, 0] + vect[22, 1] \* vect[23, 1];

if (a <= 0 || b <= 0 || c <= 0)

{

if (pointsIn[3, 0] == pointsIn[4, 0] && pointsIn[3, 1] == pointsIn[4, 1])

{

a = vect[30, 0] \* vect[31, 0] + vect[30, 1] \* vect[31, 1];

b = vect[32, 0] \* vect[33, 0] + vect[32, 1] \* vect[33, 1];

c = vect[34, 0] \* vect[35, 0] + vect[34, 1] \* vect[35, 1];

//Console.WriteLine(c);

//Console.WriteLine(pointsIn[5, 1]);

if (a <= 0 || b <= 0 || c <= 0)

{

graphics.DrawLine(Pens.Red, pointsIn[3, 0], pointsIn[3, 1], pointsIn[5, 0], pointsIn[5, 1]);

double dlina = Math.Sqrt((pointsIn[5, 0] - pointsIn[3, 0]) \* (pointsIn[5, 0] - pointsIn[3, 0]) + (pointsIn[5, 1] - pointsIn[3, 1]) \* (pointsIn[5, 1] - pointsIn[3, 1]));

Console.WriteLine("Координаты точек нового вектора: {X1:" + pointsIn[3, 0] + " Y1:" + pointsIn[3, 1] + "} {X2:" + pointsIn[5, 0] + " Y2:" + pointsIn[5, 1] + "}");

Console.WriteLine("Длинна нового вектора: " + dlina); }}

else

{

a = vect[24, 0] \* vect[25, 0] + vect[24, 1] \* vect[25, 1];

b = vect[26, 0] \* vect[27, 0] + vect[26, 1] \* vect[27, 1];

c = vect[28, 0] \* vect[29, 0] + vect[28, 1] \* vect[29, 1];

if (a <= 0 || b <= 0 || c <= 0)

{

graphics.DrawLine(Pens.Red, pointsIn[3, 0], pointsIn[3, 1], pointsIn[4, 0], pointsIn[4, 1]);

double dlina = Math.Sqrt((pointsIn[4, 0] - pointsIn[3, 0]) \* (pointsIn[4, 0] - pointsIn[3, 0]) + (pointsIn[4, 1] - pointsIn[3, 1]) \* (pointsIn[4, 1] - pointsIn[3, 1]));

Console.WriteLine("Координаты точек нового вектора: {X1:" + pointsIn[3, 0] + " Y1:" + pointsIn[3, 1] + "} {X2:" + pointsIn[4, 0] + " Y2:" + pointsIn[4, 1] + "}");

Console.WriteLine("Длинна нового вектора: " + dlina);} }}

a = vect[36, 0] \* vect[37, 0] + vect[36, 1] \* vect[37, 1];

b = vect[38, 0] \* vect[39, 0] + vect[38, 1] \* vect[39, 1];

c = vect[40, 0] \* vect[41, 0] + vect[40, 1] \* vect[41, 1];

if (a <= 0 || b <= 0 || c <= 0)

{

if (pointsIn[6, 0] == pointsIn[7, 0] && pointsIn[6, 1] == pointsIn[7, 1])

{

a = vect[48, 0] \* vect[49, 0] + vect[48, 1] \* vect[49, 1];

b = vect[50, 0] \* vect[51, 0] + vect[50, 1] \* vect[51, 1];

c = vect[52, 0] \* vect[53, 0] + vect[52, 1] \* vect[53, 1];

if (a <= 0 || b <= 0 || c <= 0)

{

graphics.DrawLine(Pens.Red, pointsIn[6, 0], pointsIn[6, 1], pointsIn[8, 0], pointsIn[8, 1]);

double dlina = Math.Sqrt((pointsIn[8, 0] - pointsIn[6, 0]) \* (pointsIn[8, 0] - pointsIn[6, 0]) + (pointsIn[8, 1] - pointsIn[6, 1]) \* (pointsIn[8, 1] - pointsIn[6, 1]));

Console.WriteLine("Координаты точек нового вектора: {X1:" + pointsIn[6, 0] + " Y1:" + pointsIn[6, 1] + "} {X2:" + pointsIn[8, 0] + " Y2:" + pointsIn[8, 1] + "}");

Console.WriteLine("Длинна нового вектора: " + dlina);}}

else

{

a = vect[42, 0] \* vect[43, 0] + vect[42, 1] \* vect[43, 1];

b = vect[44, 0] \* vect[45, 0] + vect[44, 1] \* vect[45, 1];

c = vect[46, 0] \* vect[47, 0] + vect[46, 1] \* vect[47, 1];

if (a <= 0 || b <= 0 || c <= 0)

{

graphics.DrawLine(Pens.Red, pointsIn[6, 0], pointsIn[6, 1], pointsIn[7, 0], pointsIn[7, 1]);

double dlina = Math.Sqrt((pointsIn[7, 0] - pointsIn[6, 0]) \* (pointsIn[7, 0] - pointsIn[6, 0]) + (pointsIn[7, 1] - pointsIn[6, 1]) \* (pointsIn[7, 1] - pointsIn[6, 1]));

Console.WriteLine("Координаты точек нового вектора: {X1:" + pointsIn[6, 0] + " Y1:" + pointsIn[6, 1] + "} {X2:" + pointsIn[7, 0] + " Y2:" + pointsIn[7, 1] + "}");

Console.WriteLine("Длинна нового вектора: "+dlina ); }

}

}

if (t\_out > t\_in && flag)

graphics.DrawLine(Pens.Red, resultX(t\_in), resultY(t\_in), resultX(t\_out), resultY(t\_out));

return bitmap; }

public float scalarMultiplication(Point a, Point b)

{ return (a.X \* b.X + a.Y \* b.Y);}

public Point subtraction(Point a, Point b)

{ Point tmp = new Point(a.X - b.X, a.Y - b.Y);

return tmp; }

public float resultX(float t)

{ return p1.X + t \* (p2.X - p1.X); }

public float resultY(float t)

{ return p1.Y + t \* (p2.Y - p1.Y);}

}

}