Министерство науки и высшего образования Российской Федерации

НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Лабораторная работа №1

по дисциплине

«Компьютерная графика»

Факультет прикладной математики и информатики

Группа ПМ-63

Студенты Майер В.А.

Сыпко Н.Е.

Преподаватели Задорожный А.Г.

Вариант Polygon

Новосибирск 2019

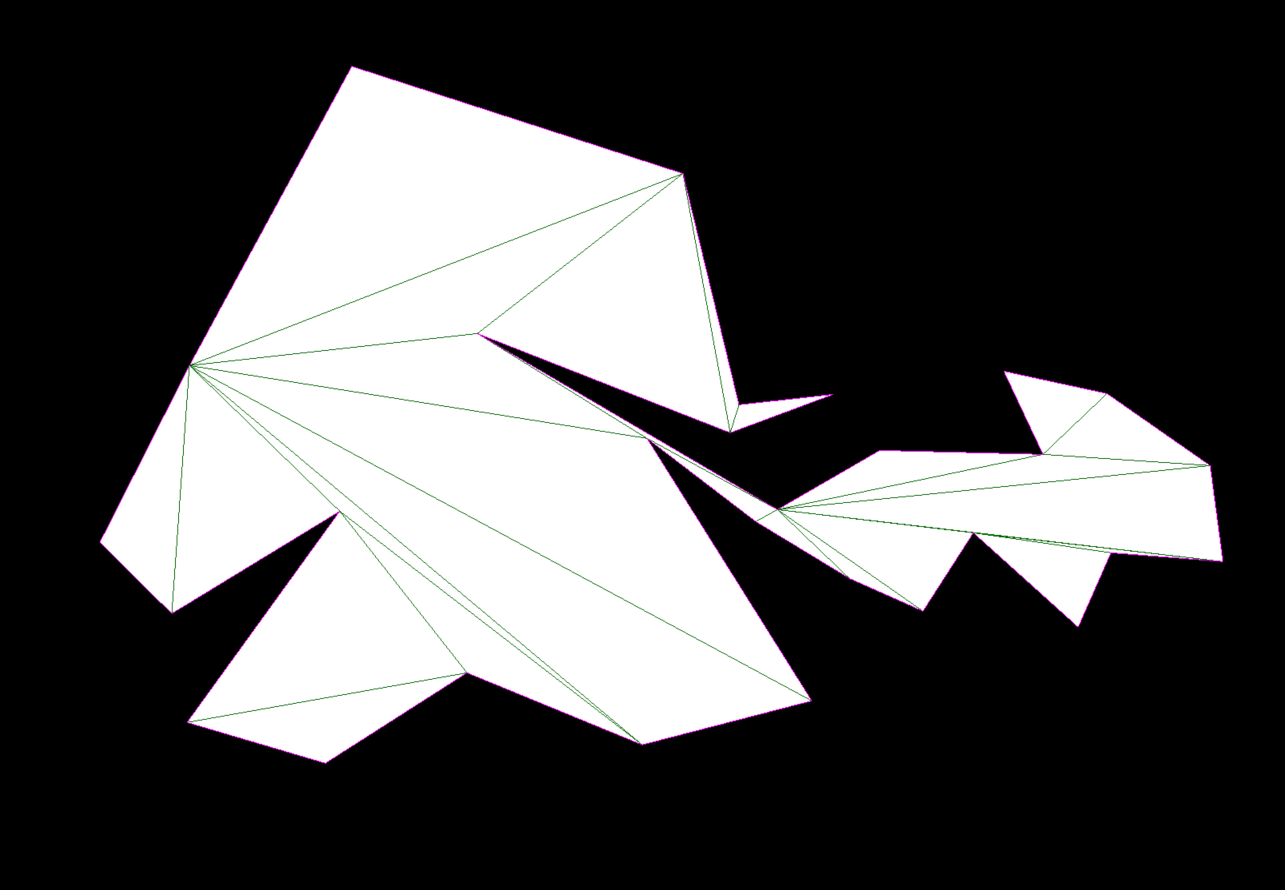
1. Цель работы

Ознакомиться с основными использования библиотеки OpenGL и работе с приметивами.

1. Текст задания
2. Отображать в окне множество примитивов – полигона
3. Для завершения текущего набора примитивов и начала нового использовать специальную клавишу
4. Предоставить возможность изменения цвета и координат вершин текущего набора примитивов
5. Текущее множество примитивов выделять среди других
6. Использовать List для хранения атрибутов
7. Предусмотреть возможность удаления примитивов
8. Продублировать команды в меню
9. Математическая модель

Для отрисвоки полигона используется метод триангуляции(метод отрезания уха), чтобы была возможность рисовать невыпуклые фигуры.

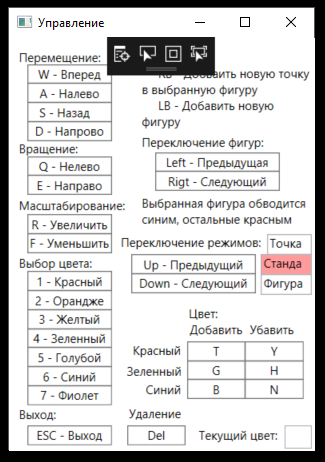
Пример:



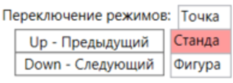
Для поворота и масштабирования используются матрицы поворота и масштабирования.

1. Руководство пользователя

Меню, которое появляется с запуском программы, его можно закрыть нажав на кнопку F1(Этой функции нет в меню, потому что это является проверкой на работоспособность клавиатуры, и чтобы не был ситуаций, когда закрыл меню и не можешь его открыть)



У приложения есть 3 режима, которые можно переключать нажатием стрелочек вверх и вниз, красный идентификатор показывает в каком режиме находится сейчас приложение.



***Режим стандарт, или добавление***, в этом режиме осуществляется добавление точек и фигур.

*Левая кнопка мыши - служит за добавление точки в активную фигуру*

*Правая кнопка мыши - служит за переход к новой фигуре*

Переход между фигурами осуществляется кнопками вправо или влево.

Активная фигура обводится синим, остальные красным.

В этом режиме применение перемещения, вращения или масштабирования применяется ко всей фигуре, а выбранный цвет, для следующей добавленной точки.

Цвет может задаваться сразу одним из предложенных, или изменением эффективности RGB, текущий цвет показан в правом нижнем угле меню, изначально он белый.

Удаление в стандартном происходит всей фигуры.

***Режим точки,*** в этом режиме изменение происходят с точкой

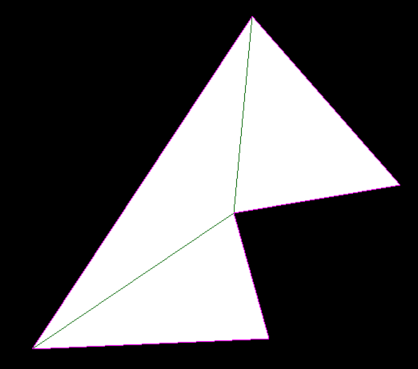
Удаление, изменение цвета, перемещение, происходят с точкой, остальные действия так же происходят с фигурой. Активная точка подсвечивается красным 

***Режим фигуры,*** этот режим нужен для того чтобы изменить цвет всей фигуры или выбрать фигуру как активную, путем наведением курсора на фигуру и нажатием на LB, после выбора режим меняется на стандартный.

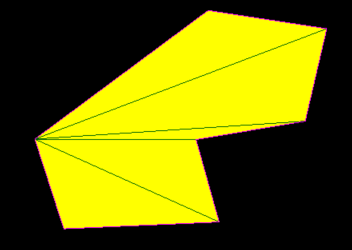
Выбор происходит верхней фигуры(может не меняется, если будет выбрана активная).

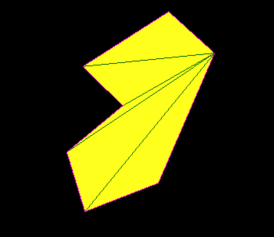
1. **Тесты**

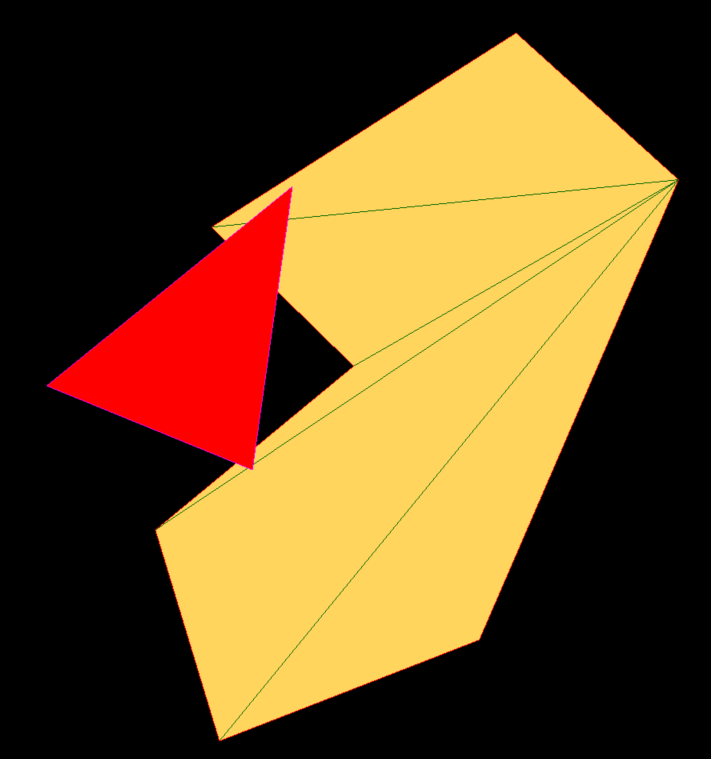
Добавление фигкры

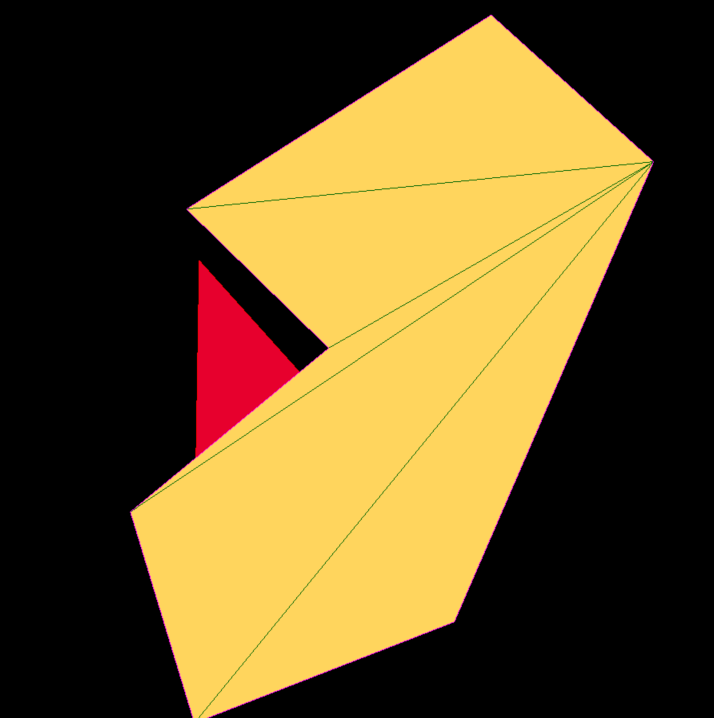


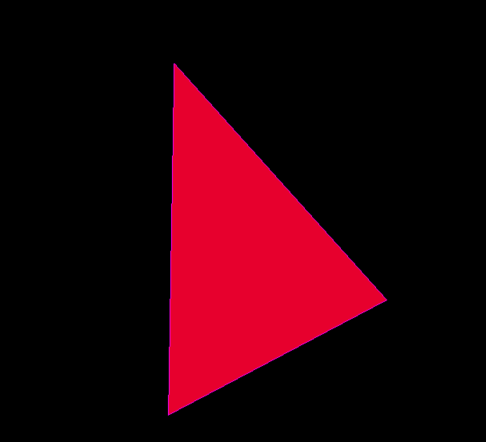
Изменение цвета



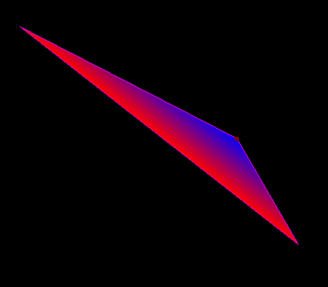
Поворот, передвижение масштабирование  


Добавление новой фигуры  


Изменение активной фигуры  


Удаление фигуры  


Изменение положение и цвета одной точки



1. **Текст программы**

using System;

using System.Threading;

using System.IO;

using System.Text;

using System.Collections.Generic;

using System.Drawing;

using OpenTK;

using OpenTK.Graphics;

using OpenTK.Graphics.OpenGL;

using OpenTK.Input;

using WpfApp1;

namespace Example

{

class Point

{

public float x, y;

public Color color { get; set; }

public Point()

{

}

public Point(float px, float py)

{

x = px;

y = py;

}

static public Point operator-(Point p1, Point p2)

{

Point point = new Point();

point.x = p1.x - p2.x;

point.y = p1.y - p2.y;

return point;

}

}

partial class MyApplication

{

public static List<List<Point>> points { get; set; }

public static int NumberActiv;

public static int ActivPoint;

public static int n;

public static int r;

private static bool S;

private static GameWindow game;

private static bool q;

public static Color color;

[STAThread]

public static void Start(MainWindow window)

{

q = true;

S = true;

n = 0;

r = 0;

points = new List<List<Point>>();

points.Add(new List<Point>());

color = Color.White;

using (game = new GameWindow())

{

game.Title = "Lab1";

game.WindowState = WindowState.Maximized;

game.Load += (sender, e) =>

{

game.VSync = VSyncMode.On;

};

game.Resize += (sender, e) =>

{

GL.Viewport(0, 0, game.Width, game.Height);

};

game.MouseMove += (sender, e) =>

{

if (r == 1)

ActivPoint = FindNearPoint(points[NumberActiv], new Point((e.X - game.Width / 2f) / game.Width \* 2f, (game.Height / 2f - e.Y) / game.Height \* 2f));

};

game.MouseDown += (sender, e) =>

{

if (e.Button == MouseButton.Left)

switch (r)

{

case 1:

break;

case 0:

ActivPoint = 0;

PushPoint(points[NumberActiv], new Point((e.X - game.Width / 2f) / game.Width \* 2f, (game.Height / 2f - e.Y) / game.Height \* 2f));

break;

case -1:

Point p = new Point((e.X - game.Width / 2f) / game.Width \* 2f, (game.Height / 2f - e.Y) / game.Height \* 2f);

if(PointInPolygon(points[NumberActiv], p))

{

r = 0;

break;

}

for (int i = points.Count - 1; i >= 0; i--)

{

if (PointInPolygon(points[i], p ))

{

NumberActiv = i;

r = 0;

break;

}

}

ActivPoint = 0;

break;

}

if (e.Button == MouseButton.Right)

{

points.Add(new List<Point>());

NumberActiv = points.Count - 1;

n++;

}

};

game.KeyDown += (sender, e) =>

{

if (e.Key == Key.Left)

if (NumberActiv > 0)

NumberActiv--;

if (e.Key == Key.Right)

if (NumberActiv < points.Count - 1)

NumberActiv++;

if (e.Key == Key.Up)

if (r < 1)

r++;

if (e.Key == Key.Down)

if (r > -1)

r--;

if (e.Key == Key.Delete)

if (r == 1)

{

points[NumberActiv].Remove(points[NumberActiv][ActivPoint]);

ActivPoint = 0;

}

else

{

points.Remove(points[NumberActiv]);

if (points.Count == 0)

points.Add(new List<Point>());

NumberActiv = 0;

}

if (e.Key == Key.Number1)

color = Color.Red;

if (e.Key == Key.Number2)

color = Color.Orange;

if (e.Key == Key.Number3)

color = Color.Yellow;

if (e.Key == Key.Number4)

color = Color.Green;

if (e.Key == Key.Number5)

color = Color.Cyan;

if (e.Key == Key.Number6)

color = Color.Blue;

if (e.Key == Key.Number7)

color = Color.Purple;

if (e.Key == Key.Escape)

{

S = false;

q = false;

//window.CloseWindow();

game.Exit();

}

};

game.UpdateFrame += (sender, e) =>

{

var state = Keyboard.GetState();

if (state[Key.T])

color = Color.FromArgb(Math.Min(255, color.R + 3), color.G, color.B);

if (state[Key.G])

color = Color.FromArgb(color.R, Math.Min(255, color.G + 3), color.B);

if (state[Key.B])

color = Color.FromArgb(color.R, color.G, Math.Min(255, color.B + 3));

if (state[Key.Y])

color = Color.FromArgb(Math.Max(0, color.R - 3), color.G, color.B);

if (state[Key.H])

color = Color.FromArgb(color.R, Math.Max(0, color.G - 3), color.B);

if (state[Key.N])

color = Color.FromArgb(color.R, color.G, Math.Max(0, color.B - 3));

if ((r == 1 || r == -1) && (state[Key.N] || state[Key.H] || state[Key.Y] || state[Key.B] || state[Key.G] || state[Key.T] ||

state[Key.Number1] || state[Key.Number2] || state[Key.Number3] || state[Key.Number4] || state[Key.Number5] || state[Key.Number6] || state[Key.Number7]))

{

if(r == 1)

points[NumberActiv][ActivPoint].color = color;

if(r == -1)

for (int i = 0; i < points[NumberActiv].Count; i++)

points[NumberActiv][i].color = color;

}

if (state[Key.D])

Move(0.05f, 0);

if (state[Key.A])

Move(-0.05f, 0);

if (state[Key.W])

Move(0, 0.05f);

if (state[Key.S])

Move(0, -0.05f);

if (state[Key.F1])

{

if (S)

{

window.Hide(); S = false;

}

else

{

window.Show(); S = true;

}

Thread.Sleep(200);

}

if (state[Key.E])

Turn(Math.PI / 18f);

if (state[Key.Q])

Turn(Math.PI / -18f);

if (state[Key.R])

Scale(1.111111111111111); // 1/0.9

if (state[Key.F])

Scale(0.9);

};

game.RenderFrame += (sender, e) =>

{

// render graphics

GL.Clear(ClearBufferMask.ColorBufferBit | ClearBufferMask.DepthBufferBit);

GL.MatrixMode(MatrixMode.Projection);

for (int j = 0; j < points.Count; j++)

if(j != NumberActiv)

PrintFigure(points[j]);

PrintFigure(points[NumberActiv], true);

if(r == 1 && points[NumberActiv].Count > ActivPoint)

{

GL.Begin(PrimitiveType.Polygon);

GL.Color3(Color.DarkRed);

double px = 5d / game.Width, py = 5d / game.Height;

GL.Vertex2(points[NumberActiv][ActivPoint].x + px, points[NumberActiv][ActivPoint].y + py );

GL.Vertex2(points[NumberActiv][ActivPoint].x + px, points[NumberActiv][ActivPoint].y - py);

GL.Vertex2(points[NumberActiv][ActivPoint].x - px, points[NumberActiv][ActivPoint].y - py);

GL.Vertex2(points[NumberActiv][ActivPoint].x - px, points[NumberActiv][ActivPoint].y + py);

GL.End();

}

game.SwapBuffers();

};

//60 кадров в сек

game.Run(60.0);

}

}

public static void Close()

{

if(q)

game.Close();

}

#region Взаимодействие с точкой

//Поиск ближайшей точки

private static int FindNearPoint(List<Point> figure, Point mouse)

{

if (figure.Count < 2)

return 0;

double min = Scal(mouse, figure[0]);

int k = 0;

double scal;

for (int i = 1; i < figure.Count; i++)

{

scal = Scal(mouse, figure[i]);

if(scal < min)

{

min = scal;

k = i;

}

}

return k;

}

#endregion

#region Взаимодействие с фигурой

//Поворот фигуры

public static void Turn(double alfa)

{

float x, y;

Point Center = new Point();

for (int i = 0; i < points[NumberActiv].Count; i++)

{

Center.x += points[NumberActiv][i].x;

Center.y += points[NumberActiv][i].y;

}

Center.x /= points[NumberActiv].Count;

Center.y /= points[NumberActiv].Count;

for (int i = 0; i < points[NumberActiv].Count; i++)

{

x = (points[NumberActiv][i].x - Center.x) \* game.Width;

y = (points[NumberActiv][i].y - Center.y) \* game.Height;

points[NumberActiv][i].x = Convert.ToSingle(Math.Cos(alfa) \* x - Math.Sin(alfa) \* y) / game.Width + Center.x;

points[NumberActiv][i].y = Convert.ToSingle(Math.Sin(alfa) \* x + Math.Cos(alfa) \* y) / game.Height + Center.y;

}

}

//Движение фигуры

public static void Move(float dx, float dy)

{

switch (r)

{

case 1:

if(points[NumberActiv].Count > ActivPoint)

{

points[NumberActiv][ActivPoint].x += dx;

points[NumberActiv][ActivPoint].y += dy;

}

break;

case 0:

case -1:

for (int i = 0; i < points[NumberActiv].Count; i++)

{

points[NumberActiv][i].x += dx;

points[NumberActiv][i].y += dy;

}

break;

}

}

//Растяжение

public static void Scale(double cScale)

{

float x, y;

Point Center = new Point();

for (int i = 0; i < points[NumberActiv].Count; i++)

{

Center.x += points[NumberActiv][i].x;

Center.y += points[NumberActiv][i].y;

}

Center.x /= points[NumberActiv].Count;

Center.y /= points[NumberActiv].Count;

for (int i = 0; i < points[NumberActiv].Count; i++)

{

x = (points[NumberActiv][i].x - Center.x) \* Convert.ToSingle(cScale);

y = (points[NumberActiv][i].y - Center.y) \* Convert.ToSingle(cScale);

points[NumberActiv][i].x = x + Center.x;

points[NumberActiv][i].y = y + Center.y;

}

}

#endregion

#region Вспомогательные функции

private static double Norm(Point a)

{

return Math.Sqrt(Math.Pow(a.x, 2) + Math.Pow(a.y, 2));

}

//Скалярное произведение

private static double Scal(Point a, Point b)

{

return Math.Sqrt(Math.Pow(a.x - b.x, 2) + Math.Pow(a.y - b.y, 2));

}

#endregion

#region Пересечение линий

private static bool angle(Point p1, Point p2)

{

if (p1.x \* p2.y - p1.y \* p2.x > 0)

return true;

else

return false;

}

//Проверка на точку внутри полигона

private static bool PointInPolygon(List<Point> polygon, Point p)

{

List<Point> toprint = new List<Point>();

for (int i = 0; i < polygon.Count; i++)

toprint.Add(polygon[i]);

List<Point> toprint2 = new List<Point>();

bool b, f;

while (toprint.Count >= 3)

{

Triangulation(ref toprint, ref toprint2);

f = false;

b = angle(p - toprint2[toprint2.Count - 1], toprint2[0] - toprint2[toprint2.Count - 1]);

for (int i = 1; i < toprint2.Count; i++)

if (b != angle(p - toprint2[i - 1], toprint2[i] - toprint2[i - 1]))

f = true;

if (!f)

return true;

}

return false;

}

//Пересичения прямой с концамы p1 и p2 с фигурой polygon

private static bool CrossingPolygon(List<Point> polygon, Point p1, Point p2)

{

for (int j = 0; j < polygon.Count; j++)

if (Crossing(p1, p2, polygon[j], polygon[(j + 1) % polygon.Count]))

{

return false;

}

return true;

}

//Пересечение двух прямых

private static bool Crossing(Point p1, Point p2, Point p3, Point p4)

{

double EPS = 1e-14;

var x = -((p1.x \* p2.y - p2.x \* p1.y) \* (p4.x - p3.x) - (p3.x \* p4.y - p4.x \* p3.y) \* (p2.x - p1.x)) / ((p1.y - p2.y) \* (p4.x - p3.x) - (p3.y - p4.y) \* (p2.x - p1.x));

if (x > Math.Max(p3.x, p4.x) - EPS || x < Math.Min(p3.x, p4.x) + EPS || x > Math.Max(p1.x, p2.x) - EPS || x < Math.Min(p1.x, p2.x) + EPS) // если за пределами отрезков

return false;

else

return true;

}

//Проверка направления обхода фигуры(по часовой или против)

private static bool Clockwise(List<Point> points)

{

int sum = 0;

if (angle(points[0] - points[points.Count - 1] , points[1] - points[0]))

sum++;

else

sum--;

for (int i = 2; i < points.Count; i++)

{

if (angle(points[i - 1] - points[i - 2], points[i] - points[i - 1]))

sum++;

else

sum--;

}

return sum > 0;

}

#endregion

#region Добавление точки в фигуры без пересечения

//Добавление точки

private static void PushPoint(List<Point> points, Point point)

{

point.color = color;

if (points.Count < 3)

{

points.Add(point);

return;

}

//Сначала ищем ближайшую точку

double minScal = Scal(point, points[0]);

int k = 0;

double scal;

for (int i = 1; i < points.Count; i++)

{

scal = Scal(point, points[i]);

if (scal < minScal)

{

k = i;

minScal = scal;

}

}

//Находим плижайшую соседнюю к ближайшей

bool b = Scal(point, points[(k + 1) % points.Count]) < Scal(point, points[(k - 1 + points.Count) % points.Count]);

//Проверяем можно ли с ближайшеми построить треугольник

if (!b)

k = (k - 1 + points.Count) % points.Count;

for (int i = 0; i < 2; i++)

{

if (CrossingPolygon(points, point, points[k]) && CrossingPolygon(points, point, points[(k + 1) % points.Count]))

{

points.Add(points[points.Count - 1]);

for (int j = points.Count - 1; j > k; j--)

points[j] = points[j - 1];

points[k + 1] = point;

return;

}

if (b)

k = (k - 1 + points.Count) % points.Count;

else

k = (k + 1) % points.Count;

}

//Если нельзя то начинаем перебор всех вариантов

for (int i = 0; i < points.Count; i++)

{

if (CrossingPolygon(points, point, points[i]) && CrossingPolygon(points, point, points[(i + 1) % points.Count]))

{

points.Add(points[points.Count - 1]);

for (int j = points.Count - 1; j > i; j--)

points[j] = points[j - 1];

points[i + 1] = point;

break;

}

}

}

#endregion

#region Отрисовка

private static void PrintFigure(List<Point> points, bool activ = false)

{

//for (int i = 0; i < points.Count; i++)

//{

// GL.Color4(Color.FromArgb(i \* 255 / points.Count));

// GL.Vertex2(points[i].x, points[i].y);

// points[i].color = Color.FromArgb(i \* 255 / points.Count);

//}

PrintPolygon(points);

GL.Begin(PrimitiveType.LineLoop);

if (activ)

GL.Color3(Color.Magenta);

else

GL.Color3(Color.Red);

for (int i = 0; i < points.Count; i++)

GL.Vertex2(points[i].x, points[i].y);

GL.End();

}

private static void PrintPolygon(List<Point> points)

{

if (points.Count < 3)

return;

List<Point> toprint = new List<Point>();

for (int i = 0; i < points.Count; i++)

toprint.Add(points[i]);

List<Point> toprint2 = new List<Point>();

while (toprint.Count >= 3)

{

Triangulation(ref toprint, ref toprint2);

GL.Begin(PrimitiveType.Polygon);

for (int i = 0; i < toprint2.Count; i++)

{

GL.Color3(toprint2[i].color);

GL.Vertex2(toprint2[i].x, toprint2[i].y);

}

GL.End();

GL.Begin(PrimitiveType.LineLoop);

GL.Color3(Color.DarkGreen);

for (int i = 0; i < toprint2.Count; i++)

{

GL.Vertex2(toprint2[i].x, toprint2[i].y);

}

GL.End();

}

}

//Проверака на выпуклость

private static bool CheckConvex(List<Point> points, int i, bool b)

{

return angle(points[i] - points[(i - 1 + points.Count) % points.Count], points[(i + 1) % points.Count] - points[i]) == b;

}

//Проверка на ухо

private static bool CheckEar(List<Point> points, int i)

{

return CrossingPolygon(points, points[(i - 1 + points.Count) % points.Count], points[(i + 1) % points.Count]);

}

//Триангуляция пногоугольника

private static void Triangulation(ref List<Point> points, ref List<Point> triangle)

{

if(points.Count <= 3)

{

triangle.Clear();

for (int j = 0; j < points.Count; j++)

triangle.Add(points[j]);

points.Clear();

return;

}

int i;

bool b = Clockwise(points);

bool f = false;

for (i = 0; i < points.Count; i++)

{

if(CheckConvex(points, i, b) && CheckEar(points, i))

{

f = true;

break;

}

}

if (f)

{

triangle.Clear();

triangle.Add(points[i]);

triangle.Add(points[(i + 1) % points.Count]);

triangle.Add(points[(i - 1 + points.Count) % points.Count]);

points.Remove(points[i]);

}

else

points.Clear();

//do

//{

// Random r = new Random();

// i = r.Next(points.Count);

//} while (!(CheckConvex(points, i, b) && CheckEar(points, i)));

//triangle.Clear();

//triangle.Add(points[i]);

//triangle.Add(points[(i + 1) % points.Count]);

//triangle.Add(points[(i - 1 + points.Count) % points.Count]);

//points.Remove(points[i]);

}

#endregion

/\*WPF window\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Drawing;

using System.Windows.Threading;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

using System.Threading;

using Example;

namespace WpfApp1

{

/// <summary>

/// Логика взаимодействия для MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

private Thread t;

public MainWindow()

{

t = new Thread(() => MyApplication.Start(this));

t.Start();

this.Topmost = true;

var timer = new DispatcherTimer();

timer.Tick += new EventHandler(timer\_Tick);

timer.Interval = new TimeSpan(0, 0, 0, 0, 100);

timer.Start();

this.Closed += (sender, e) =>

{

MyApplication.Close();

};

InitializeComponent();

}

private void timer\_Tick(object sender, EventArgs e)

{

if (!t.IsAlive)

this.Close();

this.Color.Background = new SolidColorBrush(System.Windows.Media.Color.FromArgb(255, MyApplication.color.R, MyApplication.color.G, MyApplication.color.B));

switch (MyApplication.r)

{

case -1:

this.Down.Background = new SolidColorBrush(System.Windows.Media.Color.FromArgb(100, 255, 0, 0));

this.Mid.Background = new SolidColorBrush(System.Windows.Media.Color.FromArgb(0, 255, 255, 255));

this.Up.Background = new SolidColorBrush(System.Windows.Media.Color.FromArgb(0, 255, 255, 255));

break;

case 0:

this.Mid.Background = new SolidColorBrush(System.Windows.Media.Color.FromArgb(100, 255, 0, 0));

this.Down.Background = new SolidColorBrush(System.Windows.Media.Color.FromArgb(0, 255, 255, 255));

this.Up.Background = new SolidColorBrush(System.Windows.Media.Color.FromArgb(0, 255, 255, 255));

break;

case 1:

this.Up.Background = new SolidColorBrush(System.Windows.Media.Color.FromArgb(100, 255, 0, 0));

this.Mid.Background = new SolidColorBrush(System.Windows.Media.Color.FromArgb(0, 255, 255, 255));

this.Down.Background = new SolidColorBrush(System.Windows.Media.Color.FromArgb(0, 255, 255, 255));

break;

}

}

private void ButtonW(object sender, RoutedEventArgs e)

{

MyApplication.Move(0, 0.05f);

}

private void ButtonA(object sender, RoutedEventArgs e)

{

MyApplication.Move(-0.05f, 0);

}

private void ButtonS(object sender, RoutedEventArgs e)

{

MyApplication.Move(0, -0.05f);

}

private void ButtonD(object sender, RoutedEventArgs e)

{

MyApplication.Move(0.05f, 0);

}

private void ButtonQ(object sender, RoutedEventArgs e)

{

MyApplication.Turn(Math.PI / 18f);

}

private void ButtonE(object sender, RoutedEventArgs e)

{

MyApplication.Turn(Math.PI / -18f);

}

private void ButtonR(object sender, RoutedEventArgs e)

{

MyApplication.Scale(1.111111111111111); // 1/0.9

}

private void ButtonF(object sender, RoutedEventArgs e)

{

MyApplication.Scale(0.9);

}

private void Button1(object sender, RoutedEventArgs e)

{

MyApplication.color = System.Drawing.Color.Red;

}

private void Button2(object sender, RoutedEventArgs e)

{

MyApplication.color = System.Drawing.Color.Orange;

}

private void Button3(object sender, RoutedEventArgs e)

{

MyApplication.color = System.Drawing.Color.Yellow;

}

private void Button4(object sender, RoutedEventArgs e)

{

MyApplication.color = System.Drawing.Color.Green;

}

private void Button5(object sender, RoutedEventArgs e)

{

MyApplication.color = System.Drawing.Color.Cyan;

}

private void Button6(object sender, RoutedEventArgs e)

{

MyApplication.color = System.Drawing.Color.Blue;

}

private void Button7(object sender, RoutedEventArgs e)

{

MyApplication.color = System.Drawing.Color.Purple;

}

private void ButtonESC(object sender, RoutedEventArgs e)

{

this.Close();

}

private void TextBox\_TextChanged(object sender, TextChangedEventArgs e)

{

}

private void ButtonUp(object sender, RoutedEventArgs e)

{

if (MyApplication.r < 1)

MyApplication.r++;

}

private void ButtonDown(object sender, RoutedEventArgs e)

{

if (MyApplication.r > -1)

MyApplication.r--;

}

private void ButtonT(object sender, RoutedEventArgs e)

{

MyApplication.color = System.Drawing.Color.FromArgb(Math.Min(255, MyApplication.color.R + 20), MyApplication.color.G, MyApplication.color.B);

}

private void ButtonY(object sender, RoutedEventArgs e)

{

MyApplication.color = System.Drawing.Color.FromArgb(Math.Max(0, MyApplication.color.R - 20), MyApplication.color.G, MyApplication.color.B);

}

private void ButtonG(object sender, RoutedEventArgs e)

{

MyApplication.color = System.Drawing.Color.FromArgb(MyApplication.color.R, Math.Min(255, MyApplication.color.G + 20), MyApplication.color.B);

}

private void ButtonH(object sender, RoutedEventArgs e)

{

MyApplication.color = System.Drawing.Color.FromArgb(MyApplication.color.R, Math.Max(0, MyApplication.color.G - 20), MyApplication.color.B);

}

private void ButtonB(object sender, RoutedEventArgs e)

{

MyApplication.color = System.Drawing.Color.FromArgb(MyApplication.color.R, MyApplication.color.G, Math.Min(255, MyApplication.color.B + 20));

}

private void ButtonN(object sender, RoutedEventArgs e)

{

MyApplication.color = System.Drawing.Color.FromArgb(MyApplication.color.R, MyApplication.color.G, Math.Max(0, MyApplication.color.B - 20));

}

private void ButtonDel(object sender, RoutedEventArgs e)

{

if (MyApplication.r == 1)

{

MyApplication.points[MyApplication.NumberActiv].Remove(MyApplication.points[MyApplication.NumberActiv][MyApplication.ActivPoint]);

MyApplication.ActivPoint = 0;

}

else

{

MyApplication.points.Remove(MyApplication.points[MyApplication.NumberActiv]);

if (MyApplication.points.Count == 0)

MyApplication.points.Add(new List<Example.Point>());

MyApplication.NumberActiv = 0;

}

}

private void ButtonLeft(object sender, RoutedEventArgs e)

{

if (MyApplication.NumberActiv > 0)

MyApplication.NumberActiv--;

}

private void ButtonRight(object sender, RoutedEventArgs e)

{

if (MyApplication.NumberActiv < MyApplication.n)

MyApplication.NumberActiv++;

}

}

}