**ЛАБОРАТОРНА РОБОТА №2**

З дисципліни «Комп’ютерна графіка»

|  |  |
| --- | --- |
| **Виконав**  Студент групи КВ-21  Комарницький О.Б.  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | **Перевірив**  Ст. викл. кафедри СПіСКС  Клятченко Я.М.  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

**Завдання**

* Побудувати криву Безьє 3го порядку.
* Реалізувати геометричні перетворення кривої.

**Код С#**

Програма представлена класами:

**MyDrawer** – здійснює прорисовку екрану точок і кривої Безьє

**MyModifier** – здійснює модифікації кривої (покрутити, збільшити, зменшити, перемістити)

**Mainform** – аналізує зчитані дії користувача та викликає методи із **MyDrawer** і **MyModifier**

Клас MyDrawer

class MyDrawer

{

private Color colorLine = Color.LightGreen, colorPoint = Color.Red;

private int x\_size, y\_size;

public Bitmap bitmap;

private PictureBox display;

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

public MyDrawer(Bitmap bmp, PictureBox pBox, int \_x\_size, int \_y\_size)

{

display = pBox;

bitmap = bmp;

x\_size = \_x\_size;

y\_size = \_y\_size;

}

public void reloadBitmap()

{

bitmap = new Bitmap(x\_size, y\_size);

display.Image = bitmap;

}

private double getFact(double x)

{

return (x == 0) ? 1 : x \* getFact(x - 1);

}

private double getB\_i(int i, int n, double t)

{

return getFact(n) / (double)(getFact(i) \* getFact(n - i)) \* Math.Pow(1 - t, n - i) \* Math.Pow(t, i);

}

public void draw\_curve()

{

if (points.Count >= 2)

for (double t = 0; t <= 1; t += 0.00001)

{

double x = 0;

double y = 0;

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

{

double b\_i = getB\_i(i, points.Count - 1, t);

x += b\_i \* points[i].X;

y += b\_i \* points[i].Y;

}

bitmap.SetPixel((int)x, (int)y, colorLine);

}

display.Image = bitmap;

}

public void draw\_point(Point point, int size)

{

if (point.X + size > x\_size || point.Y + size > y\_size || point.X - size < 0 || point.Y - size < 0)

return;

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

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

{

bitmap.SetPixel(point.X + i, point.Y + j, colorPoint);

bitmap.SetPixel(point.X - i, point.Y - j, colorPoint);

bitmap.SetPixel(point.X + i, point.Y - j, colorPoint);

bitmap.SetPixel(point.X - i, point.Y + j, colorPoint);

}

display.Image = bitmap;

}

public void draw\_points()

{

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

draw\_point(points[i], 3);

}

}

Клас MyModifier

class MyModifier

{

private int x\_size, x\_center;

private int y\_size, y\_center;

private short MODE = 0; // 1 - rotate; 2 - zoom; 3 - move;

private const int rotate\_lines\_len = 40;

private const int zoom\_lines\_half\_len = 20;

private const int zoom\_lines\_offset = 10;

private const int move\_lines\_offset = 10;

private const int point\_size = 3;

private const int circle\_radius = 150;

private Bitmap bitmap;

private Label dataX, dataY;

public MyModifier(Label data\_x, Label data\_y, int \_x\_zize, int \_y\_size)

{

dataX = data\_x;

dataY = data\_y;

x\_size = \_x\_zize;

y\_size = \_y\_size;

x\_center = (int)(x\_size \* .5);

y\_center = (int)(y\_size \* .5);

}

public void setMode(short mode)

{

MODE = mode;

}

public short getMode()

{

return MODE;

}

public void setBitmap(Bitmap bmp)

{

bitmap = bmp;

}

private Point point\_pointer\_vector(Point curr\_pos, int R)

{

double alpha = get\_alpha(curr\_pos);

int xf = x\_center + (int)(R \* Math.Cos(alpha));

int yf = y\_center + (int)(R \* Math.Sin(alpha));

return new Point(xf, yf);

}

public double get\_alpha(Point curr\_pos)

{

int x = curr\_pos.X;

int y = curr\_pos.Y;

double alpha = Math.Atan((y - y\_center) / (double)(x - x\_center));

if (x < x\_center && y < y\_center)

alpha = Math.PI + alpha;

else

if (x < x\_center && y\_center < y)

alpha = -(Math.PI - alpha);

return alpha;

}

public void Rotate(Point coords, List<Point> points, List<Point> fixed\_points)

{

double angle = get\_alpha(coords);

for (int i = 0; i < fixed\_points.Count; i++)

{

int dx = fixed\_points[i].X - x\_center;

int dy = fixed\_points[i].Y - y\_center;

points[i] = (new Point((int)(x\_center + dx \* Math.Cos(angle) - dy \* Math.Sin(angle)),

(int)(y\_center + dx \* Math.Sin(angle) + dy \* Math.Cos(angle))));

}

dataX.Text = "Rad: " + Convert.ToString(angle);

dataY.Text = "Grad: " + Convert.ToString(angle / (double)Math.PI \* 180);

}

private double x\_zoom\_now(double x\_zoom, Point cursor\_location)

{

return x\_zoom - x\_zoom \* (x\_size - cursor\_location.X) / (double)x\_size;

}

private double y\_zoom\_now(double y\_zoom, Point cursor\_location)

{

return y\_zoom - y\_zoom \* (453 - (cursor\_location.Y)) / (double)453;

}

public double max\_y\_zoom(List<Point> points)

{

int max\_coordinate = points[0].Y;

int min\_coordinate = points[0].Y;

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

{

if (points[i].Y < min\_coordinate)

min\_coordinate = points[i].Y;

if (points[i].Y > max\_coordinate)

max\_coordinate = points[i].Y;

}

double y = Math.Max(max\_coordinate, y\_size - min\_coordinate);

return (y\_size \* .5 - 3) / (double)(y - y\_size \* .5);

}

public double max\_x\_zoom(List<Point> points)

{

int max\_coordinate = points[0].X;

int min\_coordinate = points[0].X;

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

{

if (points[i].X < min\_coordinate)

min\_coordinate = points[i].X;

if (max\_coordinate < points[i].X)

max\_coordinate = points[i].X;

}

double x = Math.Max(max\_coordinate, x\_size - min\_coordinate);

return (x\_center - 3) / (double)(x - x\_center);

}

public void Zoom(double x\_zoom, double y\_zoom, Point mouse\_position, List<Point> fixed\_points, List<Point> points)

{

double mx = x\_zoom\_now(x\_zoom, mouse\_position);

double my = y\_zoom\_now(y\_zoom, mouse\_position);

for (int i = 0; i < fixed\_points.Count; i++)

{

int x = (int)((fixed\_points[i].X - x\_center) \* mx + x\_center);

int y = (int)((fixed\_points[i].Y - y\_center) \* my + y\_center);

points[i] = new Point(x, y);

}

draw\_zoom\_lines(mouse\_position);

dataX.Text = "X-zoom: " + Convert.ToString(mx);

dataY.Text = "Y-zoom: " + Convert.ToString(my);

}

public void Move(MouseEventArgs mouse\_events, Point move\_center, List<Point> points\_from\_center, List<Point> points)

{

int dx = mouse\_events.X - move\_center.X;

int dy = mouse\_events.Y - move\_center.Y;

bool let\_X = true;

bool let\_Y = true;

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

{

int x = points\_from\_center[i].X + mouse\_events.X;

int y = points\_from\_center[i].Y + mouse\_events.Y;

if (x < point\_size || x\_size - point\_size <= x)

let\_X = false;

if (y < point\_size || y\_size - point\_size <= y)

let\_Y = false;

}

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

{

int x = points[i].X;

int y = points[i].Y;

if (let\_X)

x = points\_from\_center[i].X + mouse\_events.X;

if (let\_Y)

y = points\_from\_center[i].Y + mouse\_events.Y;

points[i] = new Point(x, y);

}

draw\_move\_lines(points);

dataX.Text = "X: " + Convert.ToString(dx);

dataY.Text = "Y: " + Convert.ToString(dy);

}

public int min\_coordinate\_X(List<Point> points)

{

int min\_coordinate\_x = points[0].X;

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

if (points[i].X < min\_coordinate\_x)

min\_coordinate\_x = points[i].X;

return min\_coordinate\_x;

}

public int max\_coordinate\_X(List<Point> points)

{

int max\_coordinate\_x = points[0].X;

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

if (max\_coordinate\_x < points[i].X)

max\_coordinate\_x = points[i].X;

return max\_coordinate\_x;

}

public int min\_coordinate\_Y(List<Point> points)

{

int min\_coordinate\_y = points[0].Y;

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

if (points[i].Y < min\_coordinate\_y)

min\_coordinate\_y = points[i].Y;

return min\_coordinate\_y;

}

public int max\_coordinate\_Y(List<Point> points)

{

int max\_coordinate\_y = points[0].Y;

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

if (max\_coordinate\_y < points[i].Y)

max\_coordinate\_y = points[i].Y;

return max\_coordinate\_y;

}

}

Клас Mainform

public partial class MainForm : Form

{

private MyDrawer myDrawer;

private MyModifier myModifier;

private List<Point> fixed\_points = new List<Point>();

private List<Point> points\_from\_center = new List<Point>();

private Point move\_center;

double x\_zoom, y\_zoom;

public MainForm()

{

InitializeComponent();

int x\_size = pbox.Size.Width;

int x\_center = (int)(x\_size \* .5);

int y\_size = pbox.Size.Height;

int y\_center = (int)(y\_size \* .5);

myDrawer = new MyDrawer(new Bitmap(x\_size, y\_size), pbox, x\_size, y\_size);

myModifier = new MyModifier(dataX, dataY, x\_size, y\_size);

}

private void btnDraw\_Click(object sender, EventArgs e)

{

myDrawer.reloadBitmap();

myDrawer.draw\_points();

myDrawer.draw\_curve();

}

private void bSave\_Click(object sender, EventArgs e)

{

if (saveFileDialog.ShowDialog() == DialogResult.OK)

myDrawer.bitmap.Save(saveFileDialog.FileName);

}

private void pbox\_Click(object sender, EventArgs e)

{

MouseEventArgs mouse\_events = (MouseEventArgs)e;

Point coords = new Point(mouse\_events.X, mouse\_events.Y);

if(myModifier.getMode() == 0)

{

myDrawer.draw\_point(coords, 3);

myDrawer.points.Add(coords);

if (myDrawer.points.Count >= 2)

{

bDraw.BackgroundImage = Image.FromFile("icons\\run1.png");

bRotate.Enabled = bZoom.Enabled = bMove.Enabled = true;

bRotate.ForeColor = bZoom.ForeColor = bMove.ForeColor = Color.LimeGreen;

}

return;

}

if (myModifier.getMode() != 0)

myModifier.setMode(0);

myDrawer.reloadBitmap();

myDrawer.draw\_points();

myDrawer.draw\_curve();

dataX.Text = dataY.Text = "";

}

private void bClear\_Click(object sender, EventArgs e)

{

myDrawer.points.Clear();

myDrawer.reloadBitmap();

bDraw.BackgroundImage = Image.FromFile("icons/run0.png");

bRotate.Enabled = bZoom.Enabled = bMove.Enabled = false;

bRotate.ForeColor = bZoom.ForeColor = bMove.ForeColor = Color.Gray;

myModifier.setMode(0);

dataX.Text = dataY.Text = "";

}

private void bRotate\_Click(object sender, EventArgs e)

{

MouseEventArgs mouse\_events = (MouseEventArgs)e;

Point coords = new Point(mouse\_events.X, mouse\_events.Y);

myModifier.setMode(1);

fixed\_points = new List<Point>(myDrawer.points);

myDrawer.reloadBitmap();

myModifier.setBitmap(myDrawer.bitmap);

myModifier.draw\_initial\_rotate(coords);

myDrawer.draw\_points();

pbox.Image = myDrawer.bitmap;

}

private void pbox\_MouseMove(object sender, MouseEventArgs e)

{

MouseEventArgs mouse\_events = (MouseEventArgs)e;

if (myModifier.getMode() == 0) return;

if (myModifier.getMode() == 1) // Rotate

{

myDrawer.reloadBitmap();

myModifier.setBitmap(myDrawer.bitmap);

myModifier.Rotate(new Point(mouse\_events.X, mouse\_events.Y), myDrawer.points, fixed\_points);

myModifier.draw\_initial\_rotate(new Point(mouse\_events.X, mouse\_events.Y));

}

else

if (myModifier.getMode() == 2) // Zoom

{

myDrawer.reloadBitmap();

myModifier.setBitmap(myDrawer.bitmap);

myModifier.Zoom(x\_zoom, y\_zoom, new Point(mouse\_events.X, mouse\_events.Y), fixed\_points, myDrawer.points);

}

else

if (myModifier.getMode() == 3) // Move

{

myDrawer.reloadBitmap();

myModifier.setBitmap(myDrawer.bitmap);

myModifier.Move(mouse\_events, move\_center, points\_from\_center, myDrawer.points);

}

myDrawer.draw\_points();

pbox.Image = myDrawer.bitmap;

}

private void bZoom\_Click(object sender, EventArgs e)

{

myModifier.setMode(2);

fixed\_points = new List<Point>(myDrawer.points);

x\_zoom = myModifier.max\_x\_zoom(fixed\_points);

y\_zoom = myModifier.max\_y\_zoom(fixed\_points);

}

private void bMove\_Click(object sender, EventArgs e)

{

myModifier.setMode(3);

move\_center = calculate\_center\_figure(myDrawer.points);

points\_from\_center = new List<Point>();

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

points\_from\_center.Add(new Point(myDrawer.points[i].X - move\_center.X, myDrawer.points[i].Y - move\_center.Y));

myModifier.setBitmap(myDrawer.bitmap);

myModifier.draw\_move\_lines(myDrawer.points);

myDrawer.draw\_points();

}

private Point calculate\_center\_figure(List<Point> points)

{

int center\_X = 0;

int center\_Y = 0;

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

{

center\_X += points[i].X;

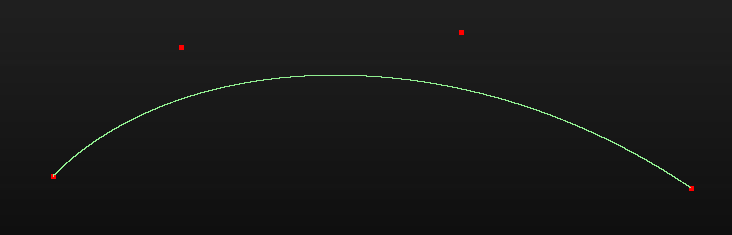
center\_Y += points[i].Y;

}

return new Point(center\_X / points.Count, center\_Y / points.Count);

}

}

**Побудова кривої 3-ого порядку**

