namespace \_2zh\_kiváltó

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private readonly float vavonal = 3; // vastag vonal

private readonly float vevonal = 1; // vékony vonal

float ax, ay, bx, by, cx, cy; // háromszög csúcsai

int katt = 0; // kattintások száma

private PointF? randomPont = null; // pont a véletlengenerátorhoz

private bool sierpinski = false; // rajzoljon-e mintát

private bool minta = false; // ki van-e rajzolva a minta

private PointF sierpinskiPont; // a minta kezdőpontja

private int reszlet = 10000; // minta részletessége

private readonly Random random = new Random();

private void button1\_Click(object sender, EventArgs e)

{

if (katt == 3) // Csak ha kész a háromszög

{

randomPont = LegyenRandomPont();

sierpinskiPont = randomPont.Value; // a generált random pont lesz a minta kezdőpontja

sierpinski = false; //még ne rajzoljon

Refresh(); // újrarajzolás

}

else

{

MessageBox.Show("Kérlek fejezd be a háromszöget!");

}

}

private void LegyenSierpinski(PaintEventArgs e)

{

// részletességtől függő pontszám

for (int r = 0; r < reszlet; r++)

{

// egy véletlen csúcs (A, B, vagy C)

PointF csucs = new PointF();

int melyik = random.Next(3);

switch (melyik)

{

case 0:

csucs = new PointF(ax, ay); // A

break;

case 1:

csucs = new PointF(bx, by); // B

break;

case 2:

csucs = new PointF(cx, cy); // C

break;

}

// új pont a két kiválasztott pont között

sierpinskiPont = new PointF(

(sierpinskiPont.X + csucs.X) / 2,

(sierpinskiPont.Y + csucs.Y) / 2

);

// Draw the current point (this is where the fractal is created)

e.Graphics.FillEllipse(Brushes.Blue, sierpinskiPont.X - 1, sierpinskiPont.Y - 1, 2, 2);

}

}

private PointF LegyenRandomPont()

{

float r1 = (float)random.NextDouble();

float r2 = (float)random.NextDouble();

if (r1 + r2 > 1)

{

r1 = 1 - r1;

r2 = 1 - r2;

}

float u = r1;

float v = r2;

float w = 1 - u - v;

float px = u \* ax + v \* bx + w \* cx;

float py = u \* ay + v \* by + w \* cy;

return new PointF(px, py);

}

private void Form1\_MouseMove(object sender, MouseEventArgs e)

{

if (katt == 1)

{

bx = e.X;

by = e.Y;

Refresh();

}

if (katt == 2)

{

cx = e.X;

cy = e.Y;

Refresh();

}

}

private void button2\_Click(object sender, EventArgs e)

{

if (katt == 3)

{

if (!sierpinski) // csak ha kész nincs rajzolva

{

sierpinski = true; // most már rajzolhat

minta = true; // azaz ki van rajzolva

Refresh(); // újrarajzolás

}

else

{

MessageBox.Show("A minta kirajzolva, kérlek nyomj az újrakezdésre!");

}

}

if (!randomPont.HasValue) // ha nincs random pont

{

MessageBox.Show("Kérlek generálj egy véletlen pontot!");

}

}

private void numericUpDown1\_ValueChanged(object sender, EventArgs e)

{

// a részletesség állítása 1000-es léptékben

reszlet = (int)(numericUpDown1.Value \* 1000);

}

private void button3\_Click(object sender, EventArgs e)

{

// a háromszögön kívül minden rajz törlése

sierpinski = false;

minta = false;

randomPont = null;

Refresh(); // újrarajzolás

katt = 3; //kattintásszám marad

}

private void button5\_Click(object sender, EventArgs e)

{

Close();

}

private void Form1\_Paint(object sender, PaintEventArgs e)

{

this.Text = katt.ToString() + ". kattintás ";

if (katt == 1)

{

e.Graphics.DrawLine(new Pen(Color.Red, vevonal), ax, ay, bx, by);

}

if (katt == 2)

{

e.Graphics.DrawLine(new Pen(Color.Red, vavonal), ax, ay, bx, by);

e.Graphics.DrawLine(new Pen(Color.Red, vevonal), ax, ay, cx, cy);

e.Graphics.DrawLine(new Pen(Color.Red, vevonal), bx, by, cx, cy);

}

if (katt == 3)

{

e.Graphics.DrawLine(new Pen(Color.Red, vavonal), ax, ay, bx, by);

e.Graphics.DrawLine(new Pen(Color.Red, vavonal), ax, ay, cx, cy);

e.Graphics.DrawLine(new Pen(Color.Red, vavonal), bx, by, cx, cy);

this.Text += " A(" + ax.ToString() + ", " + ay.ToString() + ") B(" + bx.ToString() + ", " + by.ToString() + ") C(" + cx.ToString() + ", " + cy.ToString() + ")";

if (randomPont.HasValue)

{

e.Graphics.FillEllipse(Brushes.Blue, randomPont.Value.X - 2, randomPont.Value.Y - 2, 4, 4);

}

if (sierpinski && minta && randomPont.HasValue)

{

LegyenSierpinski(e);

}

}

if (katt > 3)

{

e.Graphics.Clear(Color.White);

minta = false; //ne rajzoljon újra

randomPont = null; //pontot se tartsa meg

katt = 0;

}

}

private void Form1\_MouseDown(object sender, MouseEventArgs e)

{

++katt;

switch (katt)

{

case 1:

{

ax = e.X;

ay = e.Y;

cx = ax; cy = ay; bx = ax; by = ay;

break;

}

case 2:

{

bx = e.X;

by = e.Y;

break;

}

case 3:

{

cx = e.X;

cy = e.Y;

break;

}

// a három csúcsa lesz A(ax, ay); B(bx, by); C(cx, cy)

}

Refresh();

}

private void Form1\_Load(object sender, EventArgs e)

{

button1.Text = "Véletlen pont";

button2.Text = "Minta rajzolása";

button3.Text = "Újrakezdés";

button5.Text = "Kilépés";

label1.Text = "Részletesség(\*1000)";

numericUpDown1.Value = 10;

//fullscreenben is működjön

label1.Anchor = AnchorStyles.Top | AnchorStyles.Right;

numericUpDown1.Anchor = AnchorStyles.Top | AnchorStyles.Right;

button1.Anchor = AnchorStyles.Top | AnchorStyles.Right;

button2.Anchor = AnchorStyles.Top | AnchorStyles.Right;

button3.Anchor = AnchorStyles.Top | AnchorStyles.Right;

button4.Anchor = AnchorStyles.Bottom | AnchorStyles.Right;

}

}

}

if (katt < polygonVerticesCount && katt > 0) // Visualize in-progress edges { Refresh(); }

Exists: Button1, Button2, Button3, Button4, Button5, Timer1

define a 3D cube with edgelength 8 with a center point 0,0,0

define a 3D pyramid with edgelength and height 8 with the center point of 0,0,0

centerObjectOnScreen

project calculated center to 2d with GetProjected2Dcenter and GetCenter

translate the shapes to align with center with Translate – ONLY IF NEEDED so projected center isnt (0;0)

project the 3D shapes to 2D using ProjectTo2D

by pressing Button1, make the cube reveal as its original shape using Form1\_Paint

by pressing Button2, make it rotate around its center point using Form1\_Paint and a timer-controlled random axis and angle chooser – for this i have skeleton tho

by pressing Button3, make the pyramid reveal as its original shape using Form1\_Paint

by pressing Button4, make the pyramid rotate around its center point using Form1\_Paint and a timer-controlled random axis and angle chooser – for this i have skeleton tho

by pressing Button5, close the program

THE WRONG ONE FOR COPY PARTS

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Diagnostics;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace szorgalmi

{

public delegate void RotationDelegate(ref Point3D point, double degrees);

public struct Point2D

{

public float X, Y;

public Point2D(float x, float y)

{

X = x;

Y = y;

}

}

public struct Point3D

{

public float X, Y, Z;

public Point3D(float x, float y, float z)

{

X = x;

Y = y;

Z = z;

}

}

public partial class Form1 : Form

{

private const float EdgeLength = 8;

private Point3D[] cube3D = {

new Point3D(-EdgeLength / 2, -EdgeLength / 2, -EdgeLength / 2),

new Point3D(EdgeLength / 2, -EdgeLength / 2, -EdgeLength / 2),

new Point3D(EdgeLength / 2, EdgeLength / 2, -EdgeLength / 2),

new Point3D(-EdgeLength / 2, EdgeLength / 2, -EdgeLength / 2),

new Point3D(-EdgeLength / 2, -EdgeLength / 2, EdgeLength / 2),

new Point3D(EdgeLength / 2, -EdgeLength / 2, EdgeLength / 2),

new Point3D(EdgeLength / 2, EdgeLength / 2, EdgeLength / 2),

new Point3D(-EdgeLength / 2, EdgeLength / 2, EdgeLength / 2)

};

private Point3D[] pyramid3D = {

new Point3D(-EdgeLength / 2, -EdgeLength / 2, EdgeLength / 2), // Base

new Point3D(EdgeLength / 2, -EdgeLength / 2, EdgeLength / 2),

new Point3D(EdgeLength / 2, EdgeLength / 2, EdgeLength / 2),

new Point3D(-EdgeLength / 2, EdgeLength / 2, EdgeLength / 2),

new Point3D(0, 0, -EdgeLength / 2) // Apex

};

private Point3D[] cube3DOriginal = new Point3D[8];

private Point3D[] pyramid3DOriginal = new Point3D[5];

private int[] cubeIndices = {

0, 1, 1, 2, 2, 3, 3, 0,

4, 5, 5, 6, 6, 7, 7, 4,

0, 4, 1, 5, 2, 6, 3, 7

};

private int[] pyramidIndices = {

0, 1, 1, 2, 2, 3, 3, 0,

0, 4, 1, 4, 2, 4, 3, 4

};

private Point3D[] current3DObject;

private Point3D[] originalState;

private int[] activeIndices;

private Point2D[] projectedPoints = new Point2D[8];

private Timer rotationTimer = new Timer();

private bool isReset = false;

public Form1()

{

InitializeComponent();

InitializeShapes();

// Center and render

CenterObjectOnScreen();

// Set timer

this.Paint += Form1\_Paint;

Invalidate();

rotationTimer.Interval = 100;

rotationTimer.Tick += timer1\_Tick;

}

private void InitializeShapes()

{

cube3D = new Point3D[]

{

new Point3D(-EdgeLength / 2, -EdgeLength / 2, -EdgeLength / 2),

new Point3D(EdgeLength / 2, -EdgeLength / 2, -EdgeLength / 2),

new Point3D(EdgeLength / 2, EdgeLength / 2, -EdgeLength / 2),

new Point3D(-EdgeLength / 2, EdgeLength / 2, -EdgeLength / 2),

new Point3D(-EdgeLength / 2, -EdgeLength / 2, EdgeLength / 2),

new Point3D(EdgeLength / 2, -EdgeLength / 2, EdgeLength / 2),

new Point3D(EdgeLength / 2, EdgeLength / 2, EdgeLength / 2),

new Point3D(-EdgeLength / 2, EdgeLength / 2, EdgeLength / 2)

};

pyramid3D = new Point3D[]

{

new Point3D(-EdgeLength / 2, -EdgeLength / 2, EdgeLength / 2), // Base

new Point3D(EdgeLength / 2, -EdgeLength / 2, EdgeLength / 2),

new Point3D(EdgeLength / 2, EdgeLength / 2, EdgeLength / 2),

new Point3D(-EdgeLength / 2, EdgeLength / 2, EdgeLength / 2),

new Point3D(0, 0, -EdgeLength / 2) // Apex

};

// Clone original states for reset functionality

cube3DOriginal = (Point3D[])cube3D.Clone();

pyramid3DOriginal = (Point3D[])pyramid3D.Clone();

// Default to cube

current3DObject = cube3D;

originalState = cube3DOriginal;

activeIndices = cubeIndices;

// Debug: Print initial 3D coordinates

Debug.WriteLine("Original 3D Points:");

foreach (var point in cube3D)

{

Debug.WriteLine($"Point3D: X={point.X}, Y={point.Y}, Z={point.Z}");

}

}

private void CenterObjectOnScreen()

{

Point2D projectedCenter2D = GetProjectedCenter2D(current3DObject);

Point2D offset = new Point2D(

ClientSize.Width / 2 - projectedCenter2D.X,

ClientSize.Height / 2 - projectedCenter2D.Y

);

for (int i = 0; i < current3DObject.Length; i++)

{

current3DObject[i].X += offset.X;

current3DObject[i].Y += offset.Y;

}

}

private Point2D GetProjectedCenter2D(Point3D[] object3D)

{

float sumX = 0;

float sumY = 0;

foreach (var point in object3D)

{

Debug.WriteLine(object3D.Length);

Point2D projectedPoint = ProjectTo2D(point);

sumX += projectedPoint.X;

sumY += projectedPoint.Y;

Debug.WriteLine($" sum of coords{sumX} {sumY}");

}

float centerX = sumX / object3D.Length;

float centerY = sumY / object3D.Length;

Debug.WriteLine($"{centerX} {centerY}");

return new Point2D(centerX, centerY);

}

private Point3D GetCenter(Point3D[] object3D)

{

float centerX = 0, centerY = 0, centerZ = 0;

foreach (var point in object3D)

{

centerX += point.X;

centerY += point.Y;

centerZ += point.Z;

}

int count = object3D.Length;

return new Point3D(centerX / count, centerY / count, centerZ / count);

}

private void Form1\_Load(object sender, EventArgs e)

{

}

private void SwitchObject(Point3D[] newObject, Point3D[] original, int[] indices)

{

rotationTimer.Stop();

current3DObject = newObject;

originalState = original;

activeIndices = indices;

isReset = true;

Refresh();

}

private void button1\_Click(object sender, EventArgs e)

{

if (isReset) // Ensure the cube is reset before starting rotation

{

// Start or stop rotation for the cube

Debug.WriteLine("rotation beginned");

rotationTimer.Enabled = !rotationTimer.Enabled; // Toggle the rotation

Invalidate(); // Redraw the cube after starting or stopping the rotation

}

else

{

MessageBox.Show("Please reset the cube first by pressing Button 2.");

}

}

private void RotateAroundCenter(ref Point3D[] object3D, double degrees, RotationDelegate rotation)

{

Debug.WriteLine("3D Points before rotation:");

foreach (var point in object3D)

{

Debug.WriteLine($"Point3D: X={point.X}, Y={point.Y}, Z={point.Z}");

}

// Get the center of the object

Point3D center = GetCenter(object3D);

// Translate object to origin

for (int i = 0; i < object3D.Length; i++)

{

object3D[i].X -= center.X;

object3D[i].Y -= center.Y;

object3D[i].Z -= center.Z;

}

// Rotate each point

for (int i = 0; i < object3D.Length; i++)

{

rotation(ref object3D[i], degrees);

}

// Translate object back

for (int i = 0; i < object3D.Length; i++)

{

object3D[i].X += center.X;

object3D[i].Y += center.Y;

object3D[i].Z += center.Z;

}

// Debug: Print 3D coordinates after rotation

Debug.WriteLine("3D Points after rotation:");

foreach (var point in object3D)

{

Debug.WriteLine($"Point3D: X={point.X}, Y={point.Y}, Z={point.Z}");

}

}

private void timer1\_Tick(object sender, EventArgs e)

{

if (isReset) // Ensure we are rotating the cube

{

Random random = new Random();

RotationDelegate rotation;

int randomAxis = random.Next(3); // Randomly choose an axis to rotate

double angle = random.Next(30); // Random rotation angle (you can adjust as needed)

// Debug: Print rotation details

Debug.WriteLine($"Rotation: Axis={randomAxis}, Angle={angle}");

switch (randomAxis)

{

case 0: rotation = RotateX; break;

case 1: rotation = RotateY; break;

default: rotation = RotateZ; break;

}

// Rotate the cube around its center

RotateAroundCenter(ref current3DObject, angle, rotation);

Invalidate(); // Redraw the cube after rotation

}

}

private static Point2D ProjectTo2D(Point3D point3D)

{

Debug.WriteLine($"Before Scaling: X={point3D.X}, Y={point3D.Y}, Z={point3D.Z}");

float scale = 1f; // Scale down the coordinates

Point2D result = new Point2D(

20 \* (-0.35F \* point3D.X + point3D.Y) \* scale,

20 \* (-0.35F \* point3D.X + point3D.Z) \* scale

);

Debug.WriteLine($"After Scaling: X={result.X}, Y={result.Y}");

return result;

}

private static void RotateX(ref Point3D point, double degrees)

{

double radians = degrees \* Math.PI / 180;

double cos = Math.Cos(radians);

double sin = Math.Sin(radians);

float y = (float)(point.Y \* cos - point.Z \* sin);

float z = (float)(point.Y \* sin + point.Z \* cos);

point.Y = y;

point.Z = z;

}

private static void RotateY(ref Point3D point, double degrees)

{

double radians = degrees \* Math.PI / 180;

double cos = Math.Cos(radians);

double sin = Math.Sin(radians);

float x = (float)(point.X \* cos + point.Z \* sin);

float z = (float)(-point.X \* sin + point.Z \* cos);

point.X = x;

point.Z = z;

}

private static void RotateZ(ref Point3D point, double degrees)

{

double radians = degrees \* Math.PI / 180;

double cos = Math.Cos(radians);

double sin = Math.Sin(radians);

float x = (float)(point.X \* cos - point.Y \* sin);

float y = (float)(point.X \* sin + point.Y \* cos);

point.X = x;

point.Y = y;

}

private void Translate(ref Point3D point, Point3D translation)

{

for (int i = 0; i < current3DObject.Length; i++)

{

Debug.WriteLine($"Translated Point {i}: X={current3DObject[i].X}, Y={current3DObject[i].Y}, Z={current3DObject[i].Z}");

}

point.X += translation.X;

point.Y += translation.Y;

point.Z += translation.Z;

}

private void button2\_Click(object sender, EventArgs e)

{

SwitchObject(cube3D, cube3DOriginal, cubeIndices);

}

private void button3\_Click(object sender, EventArgs e)

{

if (isReset) // Check if the pyramid is reset

{

// Start or stop rotation

rotationTimer.Enabled = !rotationTimer.Enabled; // Toggle the rotation

Invalidate(); // Redraw the pyramid

}

else

{

MessageBox.Show("Please reset the pyramid first by pressing Button 4.");

}

}

private void button4\_Click(object sender, EventArgs e)

{

SwitchObject(pyramid3D, pyramid3DOriginal, pyramidIndices);

}

Point3D pyramidTranslation = new Point3D(-EdgeLength / 2, -EdgeLength / 2, -EdgeLength / 2);

private void Form1\_Paint(object sender, PaintEventArgs e)

{

if (current3DObject == null || current3DObject.Length == 0)

{

Debug.WriteLine("current3DObject is null or empty during Paint.");

return;

}

if (activeIndices == null || activeIndices.Length == 0)

{

Debug.WriteLine("activeIndices is null or empty during Paint.");

return;

}

projectedPoints = new Point2D[current3DObject.Length]; // Ensure correct sizing

e.Graphics.Clear(Color.White); // Ensure clean slate before drawing

// Project and draw the shape

ProjectAndDraw(e.Graphics);

}

private void ProjectAndDraw(Graphics graphics)

{

projectedPoints = new Point2D[current3DObject.Length];

// Project all 3D points into 2D

for (int i = 0; i < current3DObject.Length; i++)

{

projectedPoints[i] = ProjectTo2D(current3DObject[i]);

}

// Draw the active shape using the indices

for (int i = 0; i < activeIndices.Length; i += 2)

{

Point2D p1 = projectedPoints[activeIndices[i]];

Point2D p2 = projectedPoints[activeIndices[i + 1]];

graphics.DrawLine(Pens.Blue, p1.X, p1.Y, p2.X, p2.Y);

}

}

}

}

int korg = hScrollBar1.Value-1;

int fokx = Math.Sign(-hScrollBar1.Value);

Forgatas mforgx = ForgatX;

for (int i = 0; i < forgatando3D.Length; i++)

{

mforgx(ref forgatando3D[i], fokx-korg);

Debug.WriteLine(hScrollBar1.Value);

}

Refresh();