Оглавление

[Задание 1](#_Toc90568905)

[Листинг программы 1](#_Toc90568906)

[Результаты 19](#_Toc90568907)

# Задание

Разработать графический редактор, позволяющий формировать и редактировать изображения.

Меню редактора должно обеспечивать выбор следующих режимов:

а) выбор цвета фона,

б) выбор цвета объектов,

в) выбор типа и толщины линий,

г) рисование прямых линий по задаваемым координатам,

д) рисование прямоугольников, треугольников и многоугольников по задаваемым координатам.

е) специальная функция (зависит от номера):

- построение заданного объекта и управление режимами отрисовки (точечный, каркасный, сплошной); поддержка функций изменения масштаба изображения и его вращения (на хорошо);

- моделирование источников света и использование текстур ( на отлично).

# Листинг программы

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Windows.Forms;

using OpenTK;

using OpenTK.Graphics.OpenGL;

using KeyPressEventArgs = System.Windows.Forms.KeyPressEventArgs;

namespace LR4

{

public partial class Form1 : Form

{

private TwoDWorkMode workMode1;

private ThreeDWorkMode workMode2;

private ViewportMode viewportMode;

private Color backgroundColor = Color.WhiteSmoke;

private Sphere sphere = new Sphere(new Point3D(), 3, Color.Blue);

private List<TwoDObject> twoDObjects = new List<TwoDObject>();

private bool mousePressed = false;

int mx, my;

private int sphereAngleZ = 0;

private int sphereAngleY = 0;

private float sphereScale = 1;

private Manual manual;

public Form1()

{

InitializeComponent();

twoDObjects.Add(new TwoDObject());

}

private void glControl1\_Load(object sender, EventArgs e)

{

SetupViewport();

glControl1.Invalidate();

}

private void glControl1\_Resize(object sender, EventArgs e)

{

SetupViewport();

glControl1.Invalidate();

}

private void glControl1\_Paint(object sender, PaintEventArgs e)

{

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

if(viewportMode == ViewportMode.THREE\_D)

{

Matrix4 lMatrix = Matrix4.LookAt(15f, 0f, 0f, 0, 0, 0, 0, 0, 1);

GL.MatrixMode(MatrixMode.Modelview);

GL.LoadMatrix(ref lMatrix);

GL.Rotate(sphereAngleZ, new Vector3d(0, 0, 1));

GL.Rotate(sphereAngleY, new Vector3d(0, 1, 0));

GL.Scale(new Vector3(sphereScale));

sphere.Draw();

}

else

{

foreach (TwoDObject obj in twoDObjects)

obj.Draw();

}

glControl1.SwapBuffers();

}

private void glControl1\_MouseDown(object sender, MouseEventArgs e)

{

if (e.Button != MouseButtons.Left)

return;

if(viewportMode == ViewportMode.TWO\_D && workMode1 == TwoDWorkMode.ADD)

{

twoDObjects[twoDObjects.Count - 1].AddPoint(e.X, e.Y);

glControl1.Invalidate();

}

if (viewportMode == ViewportMode.TWO\_D && workMode1 == TwoDWorkMode.EDIT)

{

foreach (TwoDObject obj in twoDObjects)

{

if (obj.Hit(e.X, e.Y))

{

mousePressed = true;

mx = e.X;

my = e.Y;

}

}

glControl1.Invalidate();

}

if (viewportMode == ViewportMode.THREE\_D)

{

mousePressed = true;

mx = e.X;

my = e.Y;

}

}

private void glControl1\_MouseUp(object sender, MouseEventArgs e)

{

if (e.Button != MouseButtons.Left)

return;

mousePressed = false;

}

private void glControl1\_MouseMove(object sender, MouseEventArgs e)

{

if (!mousePressed)

return;

if (viewportMode == ViewportMode.TWO\_D && workMode1 == TwoDWorkMode.EDIT)

{

foreach (TwoDObject obj in twoDObjects)

{

obj.Drag(e.X - mx, e.Y - my);

}

}

if (viewportMode == ViewportMode.THREE\_D && workMode2 == ThreeDWorkMode.ROTATE)

{

sphereAngleZ += (e.X - mx);

sphereAngleY += (e.Y - my);

}

if (viewportMode == ViewportMode.THREE\_D && workMode2 == ThreeDWorkMode.SCALE)

{

sphereScale += ((e.X - mx) + (e.Y - my)) / 10.0f;

if (sphereScale < 0.2)

sphereScale = 0.2f;

if (sphereScale > 3)

sphereScale = 3;

}

mx = e.X;

my = e.Y;

glControl1.Invalidate();

}

private void glControl1\_KeyPress(object sender, KeyPressEventArgs e)

{

if (viewportMode == ViewportMode.THREE\_D)

return;

switch (e.KeyChar)

{

case (char)13: //Enter

twoDObjects.Add(new TwoDObject());

break;

case (char)32: //Space

twoDObjects[twoDObjects.Count - 1].IsClosed = true;

break;

}

glControl1.Invalidate();

}

private void SetupViewport()

{

GL.LoadIdentity();

GL.Viewport(0, 0, glControl1.Width, glControl1.Height);

GL.ClearColor(backgroundColor);

switch (viewportMode)

{

case ViewportMode.TWO\_D:

GL.Disable(EnableCap.DepthTest);

GL.Disable(EnableCap.Lighting);

GL.Disable(EnableCap.Normalize);

GL.Disable(EnableCap.Light0);

GL.MatrixMode(MatrixMode.Projection);

GL.PushMatrix();

GL.LoadIdentity();

GL.Ortho(0, glControl1.Width, glControl1.Height, 0, 0, 1);

GL.MatrixMode(MatrixMode.Modelview);

break;

case ViewportMode.THREE\_D:

GL.Enable(EnableCap.DepthTest);

GL.DepthFunc(DepthFunction.Less);

GL.DepthMask(true);

Matrix4 perspectiveMatrix = Matrix4.CreatePerspectiveFieldOfView(1f, (float)glControl1.Width / glControl1.Height, 0.5f, 50);

GL.MatrixMode(MatrixMode.Projection);

GL.LoadMatrix(ref perspectiveMatrix);

GL.Enable(EnableCap.Lighting);

GL.Enable(EnableCap.Normalize);

GL.ShadeModel(ShadingModel.Smooth);

GL.Light(LightName.Light0, LightParameter.Ambient, new float[] { 0.3f, 0.3f, 0.3f });

GL.Light(LightName.Light0, LightParameter.Diffuse, new float[] { 0.3f, 0.3f, 0.3f });

GL.Light(LightName.Light0, LightParameter.Specular, new float[] { 0.5f, 0.5f, 0.5f });

GL.Light(LightName.Light0, LightParameter.Position, new float[] { 5.0f, 5.0f, 5.0f });

GL.Enable(EnableCap.Light0);

break;

}

}

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

{

foreach (TwoDObject obj in twoDObjects)

obj.LineType = LineTypes.SOLID;

solidToolStripMenuItem.Checked = true;

dottedToolStripMenuItem.Checked = false;

glControl1.Invalidate();

}

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

{

foreach (TwoDObject obj in twoDObjects)

obj.LineType = LineTypes.DOTTED;

solidToolStripMenuItem.Checked = false;

dottedToolStripMenuItem.Checked = true;

glControl1.Invalidate();

}

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

{

ColorDialog colorDialog = new ColorDialog();

if (colorDialog.ShowDialog() != DialogResult.OK)

return;

backgroundColor = colorDialog.Color;

SetupViewport();

glControl1.Invalidate();

}

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

{

ColorDialog colorDialog = new ColorDialog();

if (colorDialog.ShowDialog() != DialogResult.OK)

return;

foreach (TwoDObject obj in twoDObjects)

obj.LinesColor = colorDialog.Color;

glControl1.Invalidate();

}

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

{

mode1ToolStripMenuItem.Text = "Добавление";

mode2ToolStripMenuItem.Text = "Редактирование";

mode1ToolStripMenuItem.Checked = workMode1 == TwoDWorkMode.ADD ? true : false;

mode2ToolStripMenuItem.Checked = workMode1 == TwoDWorkMode.EDIT ? true : false;

viewportMode = ViewportMode.TWO\_D;

twoDToolStripMenuItem.Checked = true;

threeDToolStripMenuItem.Checked = false;

SetupViewport();

glControl1.Invalidate();

}

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

{

mode1ToolStripMenuItem.Text = "Вращение";

mode2ToolStripMenuItem.Text = "Масштабирование";

mode1ToolStripMenuItem.Checked = workMode2 == ThreeDWorkMode.ROTATE ? true : false;

mode2ToolStripMenuItem.Checked = workMode2 == ThreeDWorkMode.SCALE ? true : false;

viewportMode = ViewportMode.THREE\_D;

twoDToolStripMenuItem.Checked = false;

threeDToolStripMenuItem.Checked = true;

SetupViewport();

glControl1.Invalidate();

}

private void mode1ToolStripMenuItem\_Click(object sender, EventArgs e)

{

if (viewportMode == ViewportMode.TWO\_D)

workMode1 = TwoDWorkMode.ADD;

else

workMode2 = ThreeDWorkMode.ROTATE;

mode1ToolStripMenuItem.Checked = true;

mode2ToolStripMenuItem.Checked = false;

}

private void mode2ToolStripMenuItem\_Click(object sender, EventArgs e)

{

if (viewportMode == ViewportMode.TWO\_D)

workMode1 = TwoDWorkMode.EDIT;

else

workMode2 = ThreeDWorkMode.SCALE;

mode1ToolStripMenuItem.Checked = false;

mode2ToolStripMenuItem.Checked = true;

}

private void solidToolStripMenuItem1\_Click(object sender, EventArgs e)

{

sphere.DrawType = DrawingType.SOLID;

solidToolStripMenuItem1.Checked = true;

wiredToolStripMenuItem.Checked = false;

dottedToolStripMenuItem1.Checked = false;

glControl1.Invalidate();

}

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

{

sphere.DrawType = DrawingType.WIRED;

solidToolStripMenuItem1.Checked = false;

wiredToolStripMenuItem.Checked = true;

dottedToolStripMenuItem1.Checked = false;

glControl1.Invalidate();

}

private void dottedToolStripMenuItem1\_Click(object sender, EventArgs e)

{

sphere.DrawType = DrawingType.DOTTED;

solidToolStripMenuItem1.Checked = false;

wiredToolStripMenuItem.Checked = false;

dottedToolStripMenuItem1.Checked = true;

glControl1.Invalidate();

}

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

{

foreach (TwoDObject obj in twoDObjects)

obj.LineWidth = 1;

glControl1.Invalidate();

oneToolStripMenuItem.Checked = true;

twoToolStripMenuItem.Checked = false;

threeToolStripMenuItem.Checked = false;

}

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

{

foreach (TwoDObject obj in twoDObjects)

obj.LineWidth = 2;

glControl1.Invalidate();

oneToolStripMenuItem.Checked = false;

twoToolStripMenuItem.Checked = true;

threeToolStripMenuItem.Checked = false;

}

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

{

foreach (TwoDObject obj in twoDObjects)

obj.LineWidth = 3;

glControl1.Invalidate();

oneToolStripMenuItem.Checked = false;

twoToolStripMenuItem.Checked = false;

threeToolStripMenuItem.Checked = true;

}

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

{

Close();

}

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

{

if(manual == null)

manual = new Manual();

manual.Show();

}

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

{

new AboutBox().ShowDialog();

}

}

enum ViewportMode

{

TWO\_D, THREE\_D

}

enum TwoDWorkMode

{

ADD, EDIT

}

enum ThreeDWorkMode

{

ROTATE, SCALE

}

}

using System;

using System.Collections.Generic;

using System.Drawing;

using OpenTK.Graphics.OpenGL;

namespace LR4

{

/// <summary>

/// Класс, реализующий сферу

/// </summary>

public class Sphere : List<List<Point3D>>

{

#region Свойства

/// <summary>

/// Координаты центра сферы

/// </summary>

public Point3D Center { get; set; }

/// <summary>

/// Радиус сферы

/// </summary>

public double Radius { get; set; }

/// <summary>

/// Цвет сферы

/// </summary>

public Color Color { get; set; }

public DrawingType DrawType { get; set; }

#endregion

#region Конструкторы

/// <summary>

/// Конструктор по умолчанию

/// </summary>

public Sphere() : base()

{

Center = new Point3D();

Radius = 1;

Color = Color.Black;

}

/// <summary>

/// Параметрический конструктор

/// </summary>

/// <param name="center">Координата центра</param>

/// <param name="radius">Радиус</param>

/// <param name="color">Цвет</param>

public Sphere(Point3D center, double radius, Color color) : base()

{

Center = center;

Radius = radius;

Color = color;

Init();

}

#endregion

#region Методы

/// <summary>

/// Инициализация сферы

/// </summary>

private void Init()

{

for (double u = -Math.PI; u <= Math.PI; u += Math.PI / 25)

{

List<Point3D> vList = new List<Point3D>();

for (double v = 0; v <= 2 \* Math.PI; v += Math.PI / 25)

{

Point3D point = new Point3D();

point.x = Radius \* Math.Cos(u) \* Math.Cos(v) + Center.x;

point.y = Radius \* Math.Cos(u) \* Math.Sin(v) + Center.y;

point.z = Radius \* Math.Sin(u) + Center.z;

vList.Add(point);

}

Add(vList);

}

}

/// <summary>

/// Обновление сферы

/// </summary>

public void Refresh()

{

Clear();

Init();

}

public void Draw()

{

GL.Disable(EnableCap.ColorMaterial);

GL.Material(MaterialFace.FrontAndBack, MaterialParameter.Ambient, new float[] { 0.2f, 0.2f, 0.2f });

GL.Material(MaterialFace.FrontAndBack, MaterialParameter.Diffuse, new float[] { 0.7f, 0.7f, 0.7f });

GL.Material(MaterialFace.FrontAndBack, MaterialParameter.Specular, new float[] { 0.6f, 0.6f, 0.6f });

GL.Material(MaterialFace.FrontAndBack, MaterialParameter.Shininess, new float[] { 0.8f \* 128 });

switch (DrawType)

{

case DrawingType.SOLID:

DrawSolid();

break;

case DrawingType.WIRED:

DrawWired();

break;

case DrawingType.DOTTED:

DrawPoints();

break;

}

GL.Enable(EnableCap.ColorMaterial);

GL.ColorMaterial(MaterialFace.FrontAndBack, ColorMaterialParameter.AmbientAndDiffuse);

}

/// <summary>

/// Режим рисования точками

/// </summary>

private void DrawPoints()

{

GL.Color3(Color);

for (int i = 0; i < Count - 1; i++)

{

for (int j = 0; j < this[i].Count - 1; j++)

{

GL.Begin(BeginMode.Points);

GL.Vertex3(this[i][j].x, this[i][j].y, this[i][j].z);

GL.End();

}

}

}

private void DrawWired()

{

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

{

GL.Begin(BeginMode.LineLoop);

for (int j = 0; j < this[i].Count - 1; j++)

{

GL.Vertex3(this[i][j].x, this[i][j].y, this[i][j].z);

}

GL.End();

}

for (int j = 0; j < this[0].Count - 1; j++)

{

GL.Begin(BeginMode.LineStrip);

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

{

GL.Vertex3(this[i][j].x, this[i][j].y, this[i][j].z);

}

GL.End();

}

}

/// <summary>

/// Режим рисования полигонами (сплошное тело)

/// </summary>

private void DrawSolid()

{

bool c1 = true;

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

{

if(i%2 == 0) c1 = !c1;

for (int j = 0; j < this[i].Count - 1; j++)

{

if (c1)

{

GL.Material(MaterialFace.FrontAndBack, MaterialParameter.Ambient, new float[] { 0.2f, 0.2f, 0.4f });

GL.Material(MaterialFace.FrontAndBack, MaterialParameter.Diffuse, new float[] { 0.7f, 0.7f, 0.7f });

GL.Material(MaterialFace.FrontAndBack, MaterialParameter.Specular, new float[] { 0.6f, 0.6f, 0.6f });

GL.Material(MaterialFace.FrontAndBack, MaterialParameter.Shininess, new float[] { 0.8f \* 128 });

}

else

{

GL.Material(MaterialFace.FrontAndBack, MaterialParameter.Ambient, new float[] { 0.4f, 0.2f, 0.2f });

GL.Material(MaterialFace.FrontAndBack, MaterialParameter.Diffuse, new float[] { 0.7f, 0.7f, 0.7f });

GL.Material(MaterialFace.FrontAndBack, MaterialParameter.Specular, new float[] { 0.6f, 0.6f, 0.6f });

GL.Material(MaterialFace.FrontAndBack, MaterialParameter.Shininess, new float[] { 0.8f \* 128 });

}

int k = i + 1 < this.Count ? i + 1 : 0;

int m = j + 1;

GL.Color3(Color);

GL.Begin(BeginMode.Polygon);

GL.Normal3(this[i][j].x, this[i][j].y, this[i][j].z);

GL.Vertex3(this[i][j].x, this[i][j].y, this[i][j].z);

GL.Normal3(this[k][j].x, this[k][j].y, this[k][j].z);

GL.Vertex3(this[k][j].x, this[k][j].y, this[k][j].z);

GL.Normal3(this[k][m].x, this[k][m].y, this[k][m].z);

GL.Vertex3(this[k][m].x, this[k][m].y, this[k][m].z);

GL.Normal3(this[i][m].x, this[i][m].y, this[i][m].z);

GL.Vertex3(this[i][m].x, this[i][m].y, this[i][m].z);

GL.End();

c1 = !c1;

}

}

}

#endregion

}

public enum DrawingType

{

SOLID, WIRED, DOTTED

}

}

using System.Collections.Generic;

using System.Drawing;

using OpenTK.Graphics.OpenGL;

namespace LR4

{

public class TwoDObject

{

private List<Point2D> points;

private Point2D selectedPoint;

public Color LinesColor { get; set; } = Color.Black;

public LineTypes LineType { get; set; }

public int LineWidth { get; set; } = 1;

public bool IsClosed { get; set; }

public TwoDObject()

{

points = new List<Point2D>();

}

public bool Hit(int x, int y)

{

selectedPoint = null;

foreach (Point2D p in points)

if (p.Hit(x, y))

selectedPoint = p;

return selectedPoint != null;

}

public void Draw()

{

foreach (Point2D p in points)

p.Draw();

GL.Color3(LinesColor);

GL.LineWidth(LineWidth);

if (LineType == LineTypes.DOTTED)

{

GL.Enable(EnableCap.LineStipple);

GL.LineStipple(2, 0X00FF);

}

else

GL.Disable(EnableCap.LineStipple);

if (points.Count > 2 && IsClosed)

GL.Begin(BeginMode.LineLoop);

else

GL.Begin(BeginMode.LineStrip);

foreach (Point2D p in points)

GL.Vertex2(p.X, p.Y);

GL.End();

}

public void Drag(int dx, int dy)

{

if (selectedPoint == null)

return;

selectedPoint.X += dx;

selectedPoint.Y += dy;

}

public void AddPoint(int x, int y)

{

points.Add(new Point2D(x, y));

}

public void RemovePoint()

{

if (selectedPoint == null)

return;

points.Remove(selectedPoint);

selectedPoint = null;

}

public void CloseFigure()

{

IsClosed = true;

}

}

public enum LineTypes

{

SOLID, DOTTED

}

}

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using OpenTK.Graphics.OpenGL;

namespace LR4

{

/// <summary>

/// Класс точки в 2-х мерном пространстве

/// </summary>

public class Point2D

{

#region Свойства

public double X { get; set; }

public double Y { get; set; }

public bool IsSelected { get; set; }

#endregion

#region Конструкторы

/// <summary>

/// Конструктор по умолчанию

/// </summary>

public Point2D()

{

X = 0;

Y = 0;

}

/// <summary>

/// Параметрический конструктор

/// </summary>

/// <param name="\_x">Координата X</param>

/// <param name="\_y">Координата Y</param>

public Point2D(double \_x, double \_y)

{

X = \_x;

Y = \_y;

}

#endregion

#region Методы

public bool Hit(int x, int y)

{

IsSelected = x >= X - 3 && x <= X + 3 && y >= Y - 3 && y <= Y + 3;

return IsSelected;

}

public void Draw()

{

GL.PointSize(6);

if (IsSelected)

GL.Color3(Color.Red);

else

GL.Color3(Color.Blue);

GL.Begin(BeginMode.Points);

GL.Vertex2(X, Y);

GL.End();

}

public void Drag(int dx, int dy)

{

X += dx;

Y += dy;

}

#endregion

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace LR4

{

/// <summary>

/// Класс точки в 3-х мерном пространстве

/// </summary>

public class Point3D

{

#region Свойства

public double x { get; set; }

public double y { get; set; }

public double z { get; set; }

#endregion

#region Конструкторы

/// <summary>

/// Конструктор по умолчанию

/// </summary>

public Point3D()

{

x = 0;

y = 0;

z = 0;

}

/// <summary>

/// Параметрический конструктор

/// </summary>

/// <param name="\_x">Координата X</param>

/// <param name="\_y">Координата Y</param>

/// <param name="\_z">Координата Z</param>

public Point3D(double \_x, double \_y, double \_z)

{

x = \_x;

y = \_y;

z = \_z;

}

#endregion

}

}

# Результаты





