МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»



**ОТЧЁТ**

**по лабораторной работе №2**

**«*Построение многогранников»***

по дисциплине:

**«*Графические системы*»**

|  |  |
| --- | --- |
| Выполнил: *Астафьев В.Е.*  Группа  *АО-71* | Проверил:  Кухто А.В. |

Новосибирск

2018

**Цель работы:**

Реализовать вращение многогранников, их сдвиг и масштабирование. Изучить методы удаления невидимых линий у многогранников.

**Ход работы:**

На основе программы из ЛР №2

Реализовать функцию поворота (вращения) многогранника по осям X, Y, Z относительно центра координат.

Реализовать функцию поворота (вращения) многогранника по осям X, Y, Z относительно центра многогранника.

Реализовать функцию сдвига многогранника по осям X, Y, Z.

Реализовать функцию масштабирования многогранника.

Реализовать вывод на экран многогранника с учетом видимости или невидимости граней с использованием алгоритма Робертса или Z-буфера

**Программный код**:

**Figure.cs:**

using System;

using System.Linq;

using System.Windows.Forms;

using System.IO;

using System.Drawing;

using System.Windows.Media.Media3D;

namespace WindowsFormsApp1

{

public class Figure

{

private struct Edge

{

public int start, end;

};

private struct Face

{

public int[] p;

};

//направление источника света

private Vector3D light\_dir = new Vector3D(0, 0, 1);

private StreamReader reader;

private int countPoints, countEdges, countFaces;

private Point3D[] points, pointsToDraw;

private PointF[] projPoints;

private Edge[] edges;

private Face[] faces;

private String name;

private const double focus = 2000;

private const double minScale = 0.1;

private int defTranslationX, defTranslationY;

private double scale;

private PictureBox pb;

private int[,] zBuffer;

private Color[,] colorBuffer;

private Matrix3D resultTransformMatrix;

public Figure(PictureBox newPicBox)

{

try

{

pb = newPicBox;

defTranslationX = pb.Width / 2;

defTranslationY = pb.Height / 2;

name = @"Тетраэдр.txt";

ReadFromFile(name);

scale = 80;

resultTransformMatrix = new Matrix3D();

points.CopyTo(pointsToDraw, 0);

Scale(0);

zBuffer = new int[pb.Width, pb.Height];

colorBuffer = new Color[pb.Width, pb.Height];

for (int i = 0; i < pb.Width; i++)

for (int j = 0; j < pb.Height; j++)

{

zBuffer[i, j] = int.MinValue;

colorBuffer[i, j] = Color.Transparent;

}

}

catch (Exception e)

{

MessageBox.Show(e.Message);

}

}

private void ReadFromFile(String figure)

{

try

{

name = figure;

reader = new StreamReader(figure);

countPoints = int.Parse(reader.ReadLine());

points = new Point3D[countPoints];

pointsToDraw = new Point3D[countPoints];

projPoints = new PointF[countPoints];

int i = 0;

while (i != countPoints)

{

var line = reader.ReadLine();

var tempVals = line.Split().Select(Convert.ToDouble).ToList();

points[i].X = tempVals[0];

points[i].Y = tempVals[1];

points[i].Z = tempVals[2];

i++;

}

i = 0;

countEdges = int.Parse(reader.ReadLine());

edges = new Edge[countEdges];

while (i != countEdges)

{

var line = reader.ReadLine();

var tempVals = line.Split().Select(Convert.ToDouble).ToList();

edges[i].start = (int)tempVals[0];

edges[i].end = (int)tempVals[1];

i++;

}

if (name == @"Икосаэдр.txt")

{

i = 0;

countFaces = int.Parse(reader.ReadLine());

faces = new Face[countFaces];

while (i != countFaces)

{

var line = reader.ReadLine();

var tempVals = line.Split().Select(Convert.ToDouble).ToList();

faces[i].p = new int[3];

faces[i].p[0] = (int)tempVals[0];

faces[i].p[1] = (int)tempVals[1];

faces[i].p[2] = (int)tempVals[2];

i++;

}

}

if (name == @"Тетраэдр.txt")

{

i = 0;

countFaces = int.Parse(reader.ReadLine());

faces = new Face[countFaces];

while (i != countFaces)

{

var line = reader.ReadLine();

var tempVals = line.Split().Select(Convert.ToDouble).ToList();

faces[i].p = new int[3];

faces[i].p[0] = (int)tempVals[0];

faces[i].p[1] = (int)tempVals[1];

faces[i].p[2] = (int)tempVals[2];

i++;

}

}

if (name == @"Октаэдр.txt")

{

i = 0;

countFaces = int.Parse(reader.ReadLine());

faces = new Face[countFaces];

while (i != countFaces)

{

var line = reader.ReadLine();

var tempVals = line.Split().Select(Convert.ToDouble).ToList();

faces[i].p = new int[3];

faces[i].p[0] = (int)tempVals[0];

faces[i].p[1] = (int)tempVals[1];

faces[i].p[2] = (int)tempVals[2];

i++;

}

}

reader.Close();

}

catch (Exception e)

{

MessageBox.Show(e.Message);

}

}

private void DrawFace(Face face, ref Bitmap bmp, Color color, bool perspective)

{

double a, b, c, d;

Point3D[] t = new Point3D[face.p.Length];

if (perspective)

{

//коэффициенты уравнения плоскости текущей грани

a = +pointsToDraw[face.p[0]].Y \* pointsToDraw[face.p[1]].Z

- pointsToDraw[face.p[0]].Y \* pointsToDraw[face.p[2]].Z

- pointsToDraw[face.p[1]].Y \* pointsToDraw[face.p[0]].Z

+ pointsToDraw[face.p[2]].Y \* pointsToDraw[face.p[0]].Z

+ pointsToDraw[face.p[1]].Y \* pointsToDraw[face.p[2]].Z

- pointsToDraw[face.p[2]].Y \* pointsToDraw[face.p[1]].Z;

b = -pointsToDraw[face.p[0]].X \* pointsToDraw[face.p[1]].Z

+ pointsToDraw[face.p[0]].X \* pointsToDraw[face.p[2]].Z

+ pointsToDraw[face.p[1]].X \* pointsToDraw[face.p[0]].Z

- pointsToDraw[face.p[2]].X \* pointsToDraw[face.p[0]].Z

- pointsToDraw[face.p[1]].X \* pointsToDraw[face.p[2]].Z

+ pointsToDraw[face.p[2]].X \* pointsToDraw[face.p[1]].Z;

c = +pointsToDraw[face.p[0]].X \* pointsToDraw[face.p[1]].Y

- pointsToDraw[face.p[0]].X \* pointsToDraw[face.p[2]].Y

- pointsToDraw[face.p[1]].X \* pointsToDraw[face.p[0]].Y

+ pointsToDraw[face.p[2]].X \* pointsToDraw[face.p[0]].Y

+ pointsToDraw[face.p[1]].X \* pointsToDraw[face.p[2]].Y

- pointsToDraw[face.p[2]].X \* pointsToDraw[face.p[1]].Y;

d = -pointsToDraw[face.p[0]].X \* pointsToDraw[face.p[1]].Y \* pointsToDraw[face.p[2]].Z

+ pointsToDraw[face.p[0]].X \* pointsToDraw[face.p[2]].Y \* pointsToDraw[face.p[1]].Z

+ pointsToDraw[face.p[1]].X \* pointsToDraw[face.p[0]].Y \* pointsToDraw[face.p[2]].Z

- pointsToDraw[face.p[2]].X \* pointsToDraw[face.p[0]].Y \* pointsToDraw[face.p[1]].Z

+ pointsToDraw[face.p[1]].X \* pointsToDraw[face.p[2]].Y \* pointsToDraw[face.p[0]].Z

+ pointsToDraw[face.p[2]].X \* pointsToDraw[face.p[1]].Y \* pointsToDraw[face.p[0]].Z;

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

{

float x = projPoints[face.p[i]].X;

float y = projPoints[face.p[i]].Y;

t[i] = new Point3D(x, y, -(a \* x + b \* y + d) / c);

}

}

else

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

t[i] = pointsToDraw[face.p[i]];

if (t[0].Y == t[1].Y && t[0].Y == t[2].Y) return; // отсеиваем дегенеративные треугольники

if (t[0].Y > t[1].Y) Swap(ref t[0], ref t[1]); //сортировка вершин треугольника

if (t[0].Y > t[2].Y) Swap(ref t[0], ref t[2]);

if (t[1].Y > t[2].Y) Swap(ref t[1], ref t[2]);

int total\_height = (int)(t[2].Y - t[0].Y);

for (int i = 0; i < total\_height; i++) //построчное закрашивание треугольника

{

bool second\_half = i > t[1].Y - t[0].Y || t[1].Y == t[0].Y;

double segment\_height = second\_half ? (t[2].Y - t[1].Y) : (t[1].Y - t[0].Y);

double alpha = (double)i / total\_height;

double beta = (i - (second\_half ? t[1].Y - t[0].Y : 0)) / segment\_height;

Point3D A = t[0] + (t[2] - t[0]) \* alpha;

Point3D B = second\_half ? t[1] + (t[2] - t[1]) \* beta : t[0] + (t[1] - t[0]) \* beta;

if (A.X > B.X)

Swap(ref A, ref B);

try

{

for (int j = (int)Math.Round(A.X); j <= (int)Math.Round(B.X); j++)

{

double phi = B.X == A.X ? 1 : (j - A.X) / (B.X - A.X);

Point3D P = A + (B - A) \* phi;

int newCoordX = (int)Math.Round(P.X + defTranslationX);

int newCoordY = (int)Math.Round(P.Y + defTranslationY);

if (0 <= newCoordX && newCoordX < pb.Width &&

0 <= newCoordY && newCoordY < pb.Height)

if (zBuffer[newCoordX, newCoordY] <= P.Z)

{

//заполнение буфера и вывод картинки в буфер

zBuffer[newCoordX, newCoordY] = (int)P.Z;

colorBuffer[newCoordX, newCoordY] = color;

}

}

}

catch (Exception e)

{

MessageBox.Show("Проблема в заполнении zbuffer:\n" + e.Message);

}

}

}

private static void Swap<T>(ref T lhs, ref T rhs)

{

T temp;

temp = lhs;

lhs = rhs;

rhs = temp;

}

public void ResizeBuffer()

{

defTranslationX = pb.Width / 2;

defTranslationY = pb.Height / 2;

zBuffer = (int[,])ResizeArray(zBuffer, new int[] { pb.Width, pb.Height });

colorBuffer = (Color[,])ResizeArray(colorBuffer, new int[] { pb.Width, pb.Height });

}

private static Array ResizeArray(Array arr, int[] newSizes)

{

if (newSizes.Length != arr.Rank)

MessageBox.Show(@"Ошибка при изменении размера массива");

var temp = Array.CreateInstance(arr.GetType().GetElementType(), newSizes);

int length = arr.Length <= temp.Length ? arr.Length : temp.Length;

Array.ConstrainedCopy(arr, 0, temp, 0, length);

return temp;

}

public void Draw(bool perspective)

{

resultTransformMatrix.Transform(pointsToDraw);

Bitmap bmp = new Bitmap(pb.Width, pb.Height);

if (perspective)

{

// Получаем проекцию точек

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

{

// Получение координат проекции

projPoints[i].X = (float)(focus / (focus + pointsToDraw[i].Z) \* pointsToDraw[i].X);

projPoints[i].Y = (float)(focus / (focus + pointsToDraw[i].Z) \* pointsToDraw[i].Y);

}

}

if (name == @"Тетраэдр.txt")

{

DrawIcosaedr(ref bmp, perspective);

}

if (name == @"Икосаэдр.txt")

{

DrawIcosaedr(ref bmp, perspective);

}

if (name == @"Октаэдр.txt")

{

DrawIcosaedr(ref bmp, perspective);

}

pb.Image = bmp;

points.CopyTo(pointsToDraw, 0);

resultTransformMatrix = new Matrix3D();

}

private void DrawIcosaedr(ref Bitmap bmp, bool perspective)

{

for (int t = 0; t < pb.Width; t++)

for (int j = 0; j < pb.Height; j++)

zBuffer[t, j] = int.MinValue;

foreach (Face face in faces)

{

//находим угол меджу гранью и источником света

Point3D[] t = new Point3D[face.p.Length];

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

{

t[i] = pointsToDraw[face.p[i]];

}

Vector3D n = Vector3D.CrossProduct((t[1] - t[0]), (t[2] - t[0]));

n.Normalize();

//используем этот угол для цвета

double intensity = Vector3D.DotProduct(n, light\_dir) / 2 + 0.5;

if (intensity > 0.5)

{

DrawFace(face, ref bmp, Color.FromArgb(255, (int)(intensity \* 255), (int)(intensity \* 255), (int)(intensity \* 255)), perspective);

}

}

for (int i = 0; i < pb.Width; i++)

for (int j = 0; j < pb.Height; j++)

if (zBuffer[i, j] > int.MinValue)

bmp.SetPixel(i, j, colorBuffer[i, j]);

Kostil(ref bmp);

}

public void RotateFigure(double angleX, double angleY, double angleZ)

{

angleX \*= Math.PI / 180;

angleY \*= Math.PI / 180;

angleZ \*= Math.PI / 180;

Matrix3D RotateXMatrix = new Matrix3D(1, 0, 0, 0,

0, Math.Cos(angleX), -Math.Sin(angleX), 0,

0, Math.Sin(angleX), Math.Cos(angleX), 0,

0, 0, 0, 1);

Matrix3D RotateYMatrix = new Matrix3D(Math.Cos(angleY), 0, Math.Sin(angleY), 0,

0, 1, 0, 0,

-Math.Sin(angleY), 0, Math.Cos(angleY), 0,

0, 0, 0, 1);

Matrix3D RotateZMatrix = new Matrix3D(Math.Cos(angleZ), -Math.Sin(angleZ), 0, 0,

Math.Sin(angleZ), Math.Cos(angleZ), 0, 0,

0, 0, 1, 0,

0, 0, 0, 1);

resultTransformMatrix = Matrix3D.Multiply(resultTransformMatrix, RotateXMatrix);

resultTransformMatrix = Matrix3D.Multiply(resultTransformMatrix, RotateYMatrix);

resultTransformMatrix = Matrix3D.Multiply(resultTransformMatrix, RotateZMatrix);

}

public void Scale(int approximation)

{

//Применяем масштаб

if (approximation != 0)

{

if (approximation > 0)

scale \*= 1.111111;

else

if (scale > minScale)

scale /= 1.111111;

}

Matrix3D ScaleMatrix3D = new Matrix3D(scale, 0, 0, 0,

0, scale, 0, 0,

0, 0, scale, 0,

0, 0, 0, 1);

resultTransformMatrix = Matrix3D.Multiply(resultTransformMatrix, ScaleMatrix3D);

}

public void MoveFigure(double translationX, double translationY, double translationZ)

{

Matrix3D TranslateMatrix3D = new Matrix3D(1, 0, 0, 0,

0, 1, 0, 0,

0, 0, 1, 0,

translationX, translationY, translationZ, 1);

resultTransformMatrix = Matrix3D.Multiply(resultTransformMatrix, TranslateMatrix3D);

}

public void ChangeFigure(String nameFigure)

{

try

{

nameFigure += ".txt";

ReadFromFile(nameFigure);

scale = 100;

resultTransformMatrix = new Matrix3D();

points.CopyTo(pointsToDraw, 0);

Scale(0);

}

catch (Exception e)

{

MessageBox.Show(e.Message);

Scale(0);

}

}

private void Kostil(ref Bitmap bmp)

{

int minX = int.MaxValue;

int maxX = int.MinValue;

int minY = int.MaxValue;

int maxY = int.MinValue;

foreach (PointF point in projPoints)

{

if (point.X < minX)

minX = (int)point.X;

if (point.X > maxX)

maxX = (int)point.X;

if (point.Y < minY)

minY = (int)point.Y;

if (point.Y > maxY)

maxY = (int)point.Y;

}

minX += defTranslationX;

maxX += defTranslationX;

minY += defTranslationY;

maxY += defTranslationY;

for (int i = minX <= 1 ? 1 : minX; i < maxX && i < pb.Width - 1; i++)

{

for (int j = minY <= 1 ? 1 : minY; j < maxY && j < pb.Height - 1; j++)

{

if ((int.MinValue == zBuffer[i, j]) && (bmp.GetPixel(i, j - 1) == bmp.GetPixel(i, j + 1)))

{

bmp.SetPixel(i, j, bmp.GetPixel(i, j - 1));

}

}

}

}

}

}

**Programm.cs:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Forms;

using WindowsFormsApp1;

namespace GraphSystems\_Laboratory1\_2\_3

{

static class Program

{

/// <summary>

/// Главная точка входа для приложения.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new Form1());

}

}

}

**Form1.cs:**

using System;

using System.Windows.Forms;

namespace WindowsFormsApp1

{

public partial class Form1 : Form

{

public static bool draw;

private Figure figure;

private int trackBarValueXCentre, trackBarValueYCentre, trackBarValueZCentre;

private int trackBarValueXAxis, trackBarValueYAxis, trackBarValueZAxis;

public Form1()

{

InitializeComponent();

pictureBox1.MouseWheel += new MouseEventHandler(pictureBox1\_MouseWheel);

}

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

{

figure = new Figure(pictureBox1);

figure.Draw(comboBox1.Text == "Центральная");

}

private void pictureBox1\_MouseWheel(object sender, MouseEventArgs e)

{

if (!checkBox1.Checked)

figure.RotateFigure(trackBarAngleX.Value, trackBarAngleY.Value, trackBarAngleZ.Value);

figure.Scale(e.Delta);

figure.MoveFigure(trackBarMoveX.Value, trackBarMoveY.Value, trackBarMoveZ.Value);

if (checkBox1.Checked)

figure.RotateFigure(trackBarAngleX.Value, trackBarAngleY.Value, trackBarAngleZ.Value);

figure.Draw(comboBox1.Text == "Центральная");

}

private void comboBox1\_SelectedIndexChanged(object sender, EventArgs e)

{

trackBarMoveX.Value = 0;

trackBarMoveY.Value = 0;

trackBarMoveZ.Value = 0;

trackBarAngleX.Value = 0;

trackBarAngleY.Value = 0;

trackBarAngleZ.Value = 0;

figure.Scale(0);

figure.Draw(comboBox1.Text == "Центральная");

}

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

{

if (figure != null)

{

figure.ResizeBuffer();

if (!checkBox1.Checked)

figure.RotateFigure(trackBarAngleX.Value, trackBarAngleY.Value, trackBarAngleZ.Value);

figure.Scale(0);

figure.MoveFigure(trackBarMoveX.Value, trackBarMoveY.Value, trackBarMoveZ.Value);

if (checkBox1.Checked)

figure.RotateFigure(trackBarAngleX.Value, trackBarAngleY.Value, trackBarAngleZ.Value);

figure.Draw(comboBox1.Text == "Центральная");

}

}

private void checkBox2\_CheckedChanged(object sender, EventArgs e)

{

draw = true;

}

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

{

if (!checkBox1.Checked)

figure.RotateFigure(trackBarAngleX.Value, trackBarAngleY.Value, trackBarAngleZ.Value);

figure.Scale(0);

figure.MoveFigure(trackBarMoveX.Value, trackBarMoveY.Value, trackBarMoveZ.Value);

if (checkBox1.Checked)

figure.RotateFigure(trackBarAngleX.Value, trackBarAngleY.Value, trackBarAngleZ.Value);

figure.Draw(comboBox1.Text == "Центральная");

}

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

{

if (!checkBox1.Checked)

figure.RotateFigure(trackBarAngleX.Value, trackBarAngleY.Value, trackBarAngleZ.Value);

figure.Scale(0);

figure.MoveFigure(trackBarMoveX.Value, trackBarMoveY.Value, trackBarMoveZ.Value);

if (checkBox1.Checked)

figure.RotateFigure(trackBarAngleX.Value, trackBarAngleY.Value, trackBarAngleZ.Value);

figure.Draw(comboBox1.Text == "Центральная");

}

private void comboBox2\_SelectedIndexChanged(object sender, EventArgs e)

{

trackBarMoveX.Value = 0;

trackBarMoveY.Value = 0;

trackBarMoveZ.Value = 0;

trackBarAngleX.Value = 0;

trackBarAngleY.Value = 0;

trackBarAngleZ.Value = 0;

figure.ChangeFigure(comboBox2.Text);

figure.Draw(comboBox1.Text == "Центральная");

}

private void checkBox1\_CheckedChanged(object sender, EventArgs e)

{

if (checkBox1.Checked)

{

trackBarValueXCentre = trackBarAngleX.Value;

trackBarValueYCentre = trackBarAngleY.Value;

trackBarValueZCentre = trackBarAngleZ.Value;

trackBarAngleX.Value = trackBarValueXAxis;

trackBarAngleY.Value = trackBarValueYAxis;

trackBarAngleZ.Value = trackBarValueZAxis;

}

else

{

trackBarValueXAxis = trackBarAngleX.Value;

trackBarValueYAxis = trackBarAngleY.Value;

trackBarValueZAxis = trackBarAngleZ.Value;

trackBarAngleX.Value = trackBarValueXCentre;

trackBarAngleY.Value = trackBarValueYCentre;

trackBarAngleZ.Value = trackBarValueZCentre;

}

}

}

}

**Результат работы программы**:



 