Министерство образования и науки Российской Федерации

**Федеральное государственное бюджетное образовательное учреждение науки и высшего образования**

«Владимирский государственный университет имени Александра Григорьевича и Николая Григорьевича Столетовых»

(ВлГУ)

Кафедра информационных систем и программной инженерии

**Лабораторная работа № 11**

**по дисциплине**

**«Программирование компьютерной графики»**

**Тема: «Тела вращения»**

Выполнил:

ст. гр. ПРИ-117

Хлызова В.Г.

Принял:

Жигалов И.Е.

Владимир, 2021

ЦЕЛЬ РАБОТЫ

Изучение методов формирования моделей объемных объектов в виде тел вращения, способов построения изображений на основе таких моделей, приобретение навыков использования тел вращения при составлении графических программ.

ЗАДАНИЕ

1. Ознакомиться по методическим указаниям и литературе с теоретическим материалом.

2. Выполнить действия, приведенные в разделе 11.2. При разработке программы имя проекта, создаваемого в MS Visual Studio, должно содержать фамилию студента и группу (например, Ivanov\_Ivan\_ISG\_105\_lab\_1).

3. При выполнении программы тело вращения должно быть описано кривой, которая реализуется функцией, соответствующей вашему варианту.

ХОД РАБОТЫ

using System;

using System.Windows.Forms;

using Tao.FreeGlut;

using Tao.OpenGl;

namespace Khlyzova\_Valeria\_PRI\_117\_lab\_11

{

public partial class Form1 : Form

{

private float rot\_1, rot\_2;

private double[,] GeometricArray = new double[40, 3];

private double[,,] ResaultGeometric = new double[40, 40, 3];

private int count\_elements = 0;

private double Angle = 2 \* Math.PI / 40;

private int Iter = 40;

public Form1()

{

InitializeComponent();

AnT.InitializeContexts();

}

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

{

Glut.glutInit();

Glut.glutInitDisplayMode(Glut.GLUT\_RGB | Glut.GLUT\_DOUBLE | Glut.GLUT\_DEPTH);

Gl.glClearColor(255, 255, 255, 1);

Gl.glViewport(0, 0, AnT.Width, AnT.Height);

Gl.glMatrixMode(Gl.GL\_PROJECTION);

Gl.glLoadIdentity();

Glu.gluPerspective(45, (float)AnT.Width / (float)AnT.Height, 0.1, 200);

Gl.glMatrixMode(Gl.GL\_MODELVIEW);

Gl.glLoadIdentity();

Gl.glEnable(Gl.GL\_DEPTH\_TEST);

Gl.glEnable(Gl.GL\_LIGHTING);

Gl.glEnable(Gl.GL\_LIGHT0);

count\_elements = 40;

// функция для вычисления точек тела вращения

calculateGeometricArray();

comboBox1.SelectedIndex = 0;

for (int ax = 0; ax < count\_elements; ax++)

{

for (int bx = 0; bx < Iter; bx++)

{

if (bx > 0)

{

double new\_x = ResaultGeometric[ax, bx - 1, 0] \* Math.Cos(Angle) - ResaultGeometric[ax, bx - 1, 1] \* Math.Sin(Angle);

double new\_y = ResaultGeometric[ax, bx - 1, 0] \* Math.Sin(Angle) + ResaultGeometric[ax, bx - 1, 1] \* Math.Cos(Angle);

ResaultGeometric[ax, bx, 0] = new\_x;

ResaultGeometric[ax, bx, 1] = new\_y;

ResaultGeometric[ax, bx, 2] = GeometricArray[ax, 2];

}

else

{

double new\_x = GeometricArray[ax, 0] \* Math.Cos(0) - GeometricArray[ax, 1] \* Math.Sin(0);

double new\_y = GeometricArray[ax, 1] \* Math.Sin(0) + GeometricArray[ax, 1] \* Math.Cos(0);

ResaultGeometric[ax, bx, 0] = new\_x;

ResaultGeometric[ax, bx, 1] = new\_y;

ResaultGeometric[ax, bx, 2] = GeometricArray[ax, 2];

}

}

}

RenderTimer.Start();

}

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

{

Draw();

}

private void Draw()

{

rot\_1++;

rot\_2++;

Gl.glClear(Gl.GL\_COLOR\_BUFFER\_BIT | Gl.GL\_DEPTH\_BUFFER\_BIT);

Gl.glClearColor(255, 255, 255, 1);

Gl.glLoadIdentity();

Gl.glTranslated(0, 0, -7 - trackBar1.Value);

Gl.glRotated(rot\_1, 1, 0, 0);

Gl.glRotated(rot\_2, 0, 1, 0);

Gl.glPointSize(5.0f);

switch (comboBox1.SelectedIndex)

{

case 0:

{

Gl.glBegin(Gl.GL\_POINTS);

for (int ax = 0; ax < count\_elements; ax++)

{

for (int bx = 0; bx < Iter; bx++)

{

Gl.glVertex3d(ResaultGeometric[ax, bx, 0], ResaultGeometric[ax, bx, 1], ResaultGeometric[ax, bx, 2]);

}

}

Gl.glEnd();

break;

}

case 1:

{

Gl.glBegin(Gl.GL\_LINE\_STRIP);

for (int ax = 0; ax < count\_elements - 1; ax++)

{

for (int bx = 0; bx < Iter - 1; bx++)

{

Gl.glVertex3d(ResaultGeometric[ax, bx, 0], ResaultGeometric[ax, bx, 1], ResaultGeometric[ax, bx, 2]);

Gl.glVertex3d(ResaultGeometric[ax + 1, bx, 0], ResaultGeometric[ax + 1, bx, 1], ResaultGeometric[ax + 1, bx, 2]);

if (bx + 1 < Iter - 1)

{

Gl.glVertex3d(ResaultGeometric[ax + 1, bx + 1, 0], ResaultGeometric[ax + 1, bx + 1, 1], ResaultGeometric[ax + 1, bx + 1, 2]);

}

else

{

Gl.glVertex3d(ResaultGeometric[ax + 1, 0, 0], ResaultGeometric[ax + 1, 0, 1], ResaultGeometric[ax + 1, 0, 2]);

}

}

}

Gl.glEnd();

break;

}

case 2:

{

Gl.glBegin(Gl.GL\_QUADS);

for (int ax = 0; ax < count\_elements; ax++)

{

for (int bx = 0; bx < Iter; bx++)

{

double x1 = 0, x2 = 0, x3 = 0, x4 = 0, y1 = 0, y2 = 0, y3 = 0, y4 = 0, z1 = 0, z2 = 0, z3 = 0, z4 = 0;

x1 = ResaultGeometric[ax, bx, 0];

y1 = ResaultGeometric[ax, bx, 1];

z1 = ResaultGeometric[ax, bx, 2];

if (ax + 1 < count\_elements)

{

x2 = ResaultGeometric[ax + 1, bx, 0];

y2 = ResaultGeometric[ax + 1, bx, 1];

z2 = ResaultGeometric[ax + 1, bx, 2];

if (bx + 1 < Iter - 1)

{

x3 = ResaultGeometric[ax + 1, bx + 1, 0];

y3 = ResaultGeometric[ax + 1, bx + 1, 1];

z3 = ResaultGeometric[ax + 1, bx + 1, 2];

x4 = ResaultGeometric[ax, bx + 1, 0];

y4 = ResaultGeometric[ax, bx + 1, 1];

z4 = ResaultGeometric[ax, bx + 1, 2];

}

else

{

x3 = ResaultGeometric[ax + 1, 0, 0];

y3 = ResaultGeometric[ax + 1, 0, 1];

z3 = ResaultGeometric[ax + 1, 0, 2];

x4 = ResaultGeometric[ax, 0, 0];

y4 = ResaultGeometric[ax, 0, 1];

z4 = ResaultGeometric[ax, 0, 2];

}

}

else

{

x2 = ResaultGeometric[0, bx, 0];

y2 = ResaultGeometric[0, bx, 1];

z2 = ResaultGeometric[0, bx, 2];

if (bx + 1 < Iter - 1)

{

x3 = ResaultGeometric[0, bx + 1, 0];

y3 = ResaultGeometric[0, bx + 1, 1];

z3 = ResaultGeometric[0, bx + 1, 2];

x4 = ResaultGeometric[ax, bx + 1, 0];

y4 = ResaultGeometric[ax, bx + 1, 1];

z4 = ResaultGeometric[ax, bx + 1, 2];

}

else

{

x3 = ResaultGeometric[0, 0, 0];

y3 = ResaultGeometric[0, 0, 1];

z3 = ResaultGeometric[0, 0, 2];

x4 = ResaultGeometric[ax, 0, 0];

y4 = ResaultGeometric[ax, 0, 1];

z4 = ResaultGeometric[ax, 0, 2];

}

}

double n1 = 0, n2 = 0, n3 = 0;

if (ax == 0)

{

n1 = (y2 - y1) \* (z3 - z1) - (y3 - y1) \* (z2 - z1);

n2 = (z2 - z1) \* (x3 - x1) - (z3 - z1) \* (x2 - x1);

n3 = (x2 - x1) \* (y3 - y1) - (x3 - x1) \* (y2 - y1);

}

else

{

n1 = (y4 - y3) \* (z1 - z3) - (y1 - y3) \* (z4 - z3);

n2 = (z4 - z3) \* (x1 - x3) - (z1 - z3) \* (x4 - x3);

n3 = (x4 - x3) \* (y1 - y3) - (x1 - x3) \* (y4 - y3);

}

double n5 = (double)Math.Sqrt(n1 \* n1 + n2 \* n2 + n3 \* n3);

n1 /= (n5 + 0.01);

n2 /= (n5 + 0.01);

n3 /= (n5 + 0.01);

Gl.glNormal3d(-n1, -n2, -n3);

Gl.glVertex3d(x1, y1, z1);

Gl.glVertex3d(x2, y2, z2);

Gl.glVertex3d(x3, y3, z3);

Gl.glVertex3d(x4, y4, z4);

}

}

Gl.glEnd();

break;

}

}

Gl.glPopMatrix();

Gl.glFlush();

AnT.Invalidate();

}

private void calculateGeometricArray()

{

double x = -5;

for (int i = 0; i<40; i++)

{

// уравнение в соответствии с вариантом

GeometricArray[i, 0] = i == 0 || i == 39 ? 0 : 7 \* x / (3 \* Math.Pow(x, 2) + 2 \* x + 1);

GeometricArray[i, 1] = 0;

GeometricArray[i, 2] = x;

x += 0.25;

}

}

}

}

Результат выполнения:

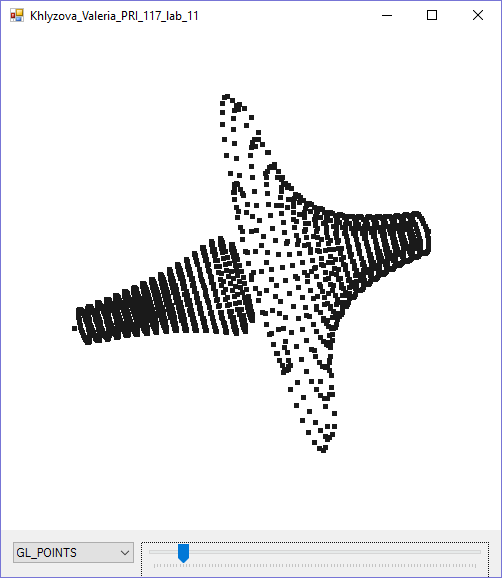


Рисунок 1 - Результат работы программы.

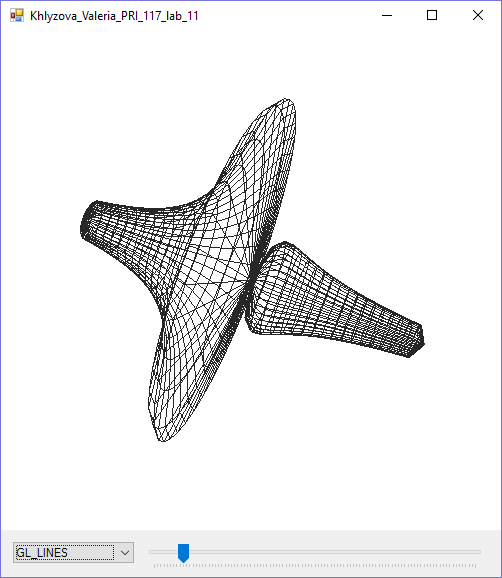


Рисунок 2 - Результат работы программы.

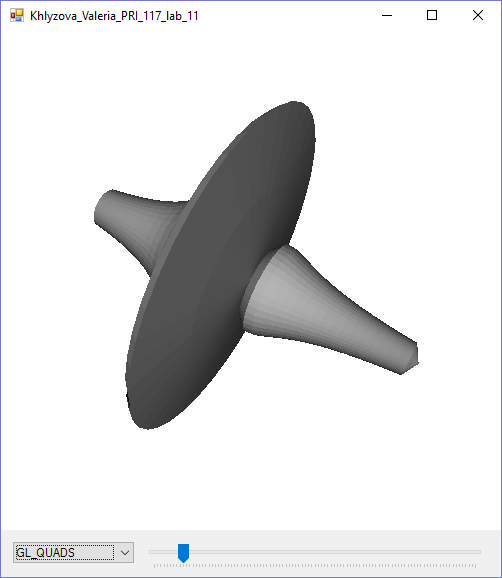


Рисунок 3 - Результат работы программы.

ВЫВОД

В ходе выполнения лабораторной работы были изучены методы формирования моделей объемных объектов в виде тел вращения, способы построения изображений на основе таких моделей, приобретены навыки использования тел вращения при составлении графических программ.