**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Компьютерная графика»

**Лабораторная работа № 6**

Тема: Создание шейдерных анимационных эффектов в OpenGL 2.1

Студент: Попов Матвей

Группа: 08-308

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2022

1. Постановка задачи

Для поверхности, созданной в л.р. №5, обеспечить выполнение следующего шейдерного эффекта.

**Вариант 4**: Анимация. Вращение относительно оси OY.

1. Описание программы

Использовался язык программирования C# и OpenGL версии 3.1.0.

Управление:

* W, A, S, D — вращение полусферы
* Стрелки вверх/вниз — увеличение/уменьшение точности аппроксимации
* Стрелки влево/вправо — управление освещением
* Клавиши +/- — увеличение/уменьшение масштаба
* Пробел — возобновить/остановить анимацию

1. Набор тестов

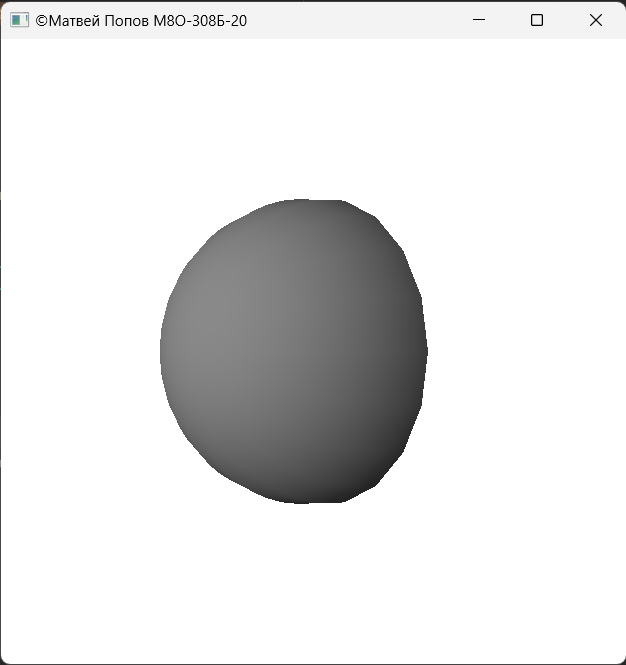


Рисунок 1. Вращение вокруг оси OY.

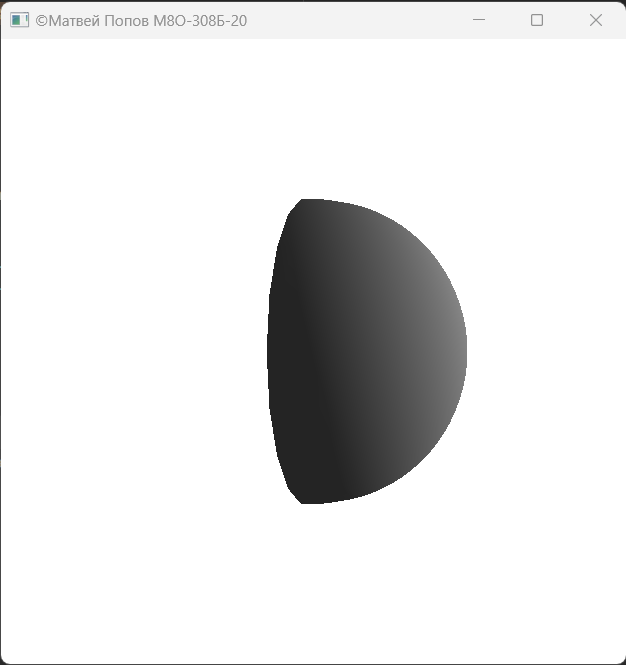


Рисунок 2. Вращение вокруг оси OY.

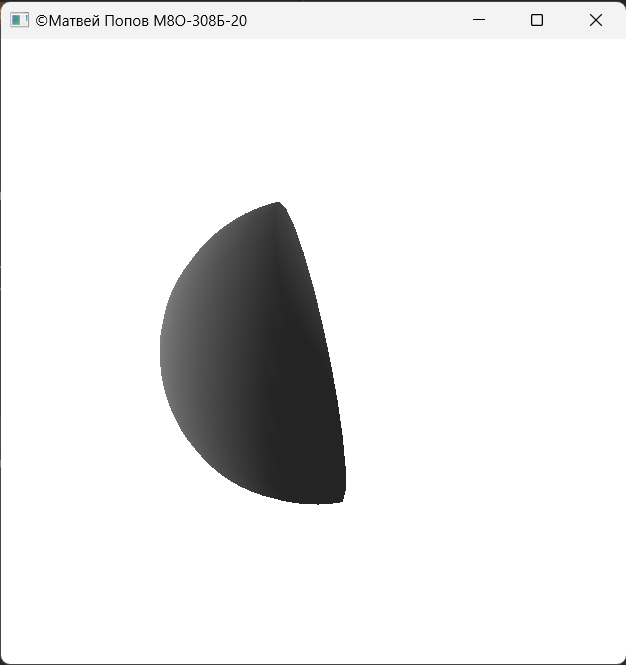


Рисунок 3. Вращение пользователя.

1. Листинг программы

**Figure.cs**

using System;

using OpenTK.Graphics.OpenGL;

namespace lab04

{

public class Figure

{

private readonly float \_radius;

private int \_precision;

private readonly float \_r;

private readonly float \_g;

private readonly float \_b;

public Figure(float radius, int precision)

{

\_radius = radius;

\_precision = precision;

\_r = 0.5f;

\_g = 0.5f;

\_b = 0.5f;

}

private const int MinPrecision = 3;

public int Precision

{

get => \_precision;

set => \_precision = (value < MinPrecision) ? MinPrecision : value;

}

public void Draw()

{

const float endPhi = (float)Math.PI \* 2.0f;

const float endTheta = (float)Math.PI \* 0.5f;

var dPhi = endPhi / \_precision;

var dTheta = endTheta / \_precision;

for (var pointPhi = 0; pointPhi < \_precision; pointPhi++)

{

for (var pointTheta = 0; pointTheta < \_precision; pointTheta++)

{

var phi = pointPhi \* dPhi;

var theta = pointTheta \* dTheta;

var phiT = (pointPhi + 1 == \_precision) ? endPhi

: (pointPhi + 1) \* dPhi;

var thetaT = (pointTheta + 1 == \_precision) ? endTheta

: (pointTheta + 1) \* dTheta;

float[] p0 = { \_radius \* (float)Math.Sin(theta) \*

(float)Math.Cos(phi), \_radius \*

(float)Math.Sin(theta) \* (float)Math.Sin(phi),

\_radius \* (float)Math.Cos(theta) };

float[] p1 = { \_radius \* (float)Math.Sin(thetaT) \*

(float)Math.Cos(phi), \_radius \*

(float)Math.Sin(thetaT) \* (float)Math.Sin(phi),

\_radius \* (float)Math.Cos(thetaT) };

float[] p2 = { \_radius \* (float)Math.Sin(theta) \*

(float)Math.Cos(phiT), \_radius \*

(float)Math.Sin(theta) \* (float)Math.Sin(phiT),

\_radius \* (float)Math.Cos(theta) };

float[] p3 = { \_radius \* (float)Math.Sin(thetaT) \*

(float)Math.Cos(phiT), \_radius \*

(float)Math.Sin(thetaT) \* (float)Math.Sin(phiT),

\_radius \* (float)Math.Cos(thetaT) };

GL.Begin(PrimitiveType.Triangles);

GL.Normal3(p0[0] / \_radius, p0[1] / \_radius,

p0[2] / \_radius);

GL.Vertex3(p0[0], p0[1], p0[2]);

GL.Normal3(p2[0] / \_radius, p2[1] / \_radius,

p2[2] / \_radius);

GL.Vertex3(p2[0], p2[1], p2[2]);

GL.Normal3(p1[0] / \_radius, p1[1] / \_radius,

p1[2] / \_radius);

GL.Vertex3(p1[0], p1[1], p1[2]);

GL.Normal3(p3[0] / \_radius, p3[1] / \_radius,

p3[2] / \_radius);

GL.Vertex3(p3[0], p3[1], p3[2]);

GL.Normal3(p1[0] / \_radius, p1[1] / \_radius,

p1[2] / \_radius);

GL.Vertex3(p1[0], p1[1], p1[2]);

GL.Normal3(p2[0] / \_radius, p2[1] / \_radius,

p2[2] / \_radius);

GL.Vertex3(p2[0], p2[1], p2[2]);

GL.Normal3(p0[0] / \_radius, p0[1] / \_radius, 0);

GL.Vertex3(p0[0], p0[1], 0);

GL.Normal3(p2[0] / \_radius, p2[1] / \_radius, 0);

GL.Vertex3(p2[0], p2[1], 0);

GL.Normal3(p1[0] / \_radius, p1[1] / \_radius, 0);

GL.Vertex3(p1[0], p1[1], 0);

GL.Normal3(p3[0] / \_radius, p3[1] / \_radius, 0);

GL.Vertex3(p3[0], p3[1], 0);

GL.Normal3(p1[0] / \_radius, p1[1] / \_radius, 0);

GL.Vertex3(p1[0], p1[1], 0);

GL.Normal3(p2[0] / \_radius, p2[1] / \_radius, 0);

GL.Vertex3(p2[0], p2[1], 0);

GL.End();

}

}

}

public void LightConfigure(float lpx)

{

float[] lightPosition = {lpx, 20, 80};

float[] lightDiffuse = {\_r, \_g, \_b};

GL.Light(LightName.Light0, LightParameter.Position, lightPosition);

GL.Light(LightName.Light0, LightParameter.Diffuse, lightDiffuse);

GL.Light(LightName.Light0, LightParameter.Ambient, lightDiffuse);

}

}

}

**Output.cs**

using System;

using OpenTK;

using OpenTK.Graphics;

using OpenTK.Graphics.OpenGL;

using OpenTK.Input;

namespace lab04

{

public class Output

{

private readonly GameWindow \_window;

private Figure \_figure;

private float \_scaling = 10.0f;

private float \_xAngle;

private float \_yAngle;

private float \_lightPositionX = 20.0f;

private bool \_animation = true;

private const float Step = 0.5f;

public Output(int size)

{

\_window = new GameWindow(size, size,

GraphicsMode.Default, "");

\_window.Load += Window\_Load;

\_window.Resize += Window\_Resize;

\_window.RenderFrame += Window\_RenderFrame;

\_window.UpdateFrame += Window\_UpdateFrame;

\_window.KeyDown += Window\_KeyDown;

}

public void Start()

{

\_figure = new Figure(2, 20);

\_window.Run(1.0 / 60.0);

}

private static void Window\_Load(object sender, EventArgs e)

{

GL.ClearColor(1.0f, 1.0f, 1.0f, 1.0f);

GL.Enable(EnableCap.DepthTest);

GL.Enable(EnableCap.Lighting);

GL.Enable(EnableCap.Light0);

}

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

{

GL.Viewport(0, 0, \_window.Width, \_window.Height);

GL.MatrixMode(MatrixMode.Projection);

GL.LoadIdentity();

var matrix =

Matrix4.CreatePerspectiveFieldOfView((float)Math.PI / 4,

1.0f, 1.0f, 100.0f);

GL.LoadMatrix(ref matrix);

GL.MatrixMode(MatrixMode.Modelview);

}

private void Window\_KeyDown(object sender, KeyboardKeyEventArgs e)

{

switch (e.Key)

{

case Key.Left:

\_lightPositionX -= 10.0f;

if (\_lightPositionX < -360.0f)

{

\_lightPositionX = 360.0f;

}

break;

case Key.Right:

\_lightPositionX += 10.0f;

if (\_lightPositionX > 360.0f)

{

\_lightPositionX = -360.0f;

}

break;

case Key.Up:

\_figure.Precision++;

break;

case Key.Down:

\_figure.Precision--;

break;

case Key.Plus:

\_scaling -= 0.5f;

break;

case Key.Minus:

\_scaling += 0.5f;

break;

case Key.S:

\_xAngle += 10.0f;

if (\_xAngle > 360.0f)

{

\_xAngle = 0.0f;

}

break;

case Key.W:

\_xAngle -= 10.0f;

if (\_xAngle < 0.0f)

{

\_xAngle = 360.0f;

}

break;

case Key.D:

\_yAngle += 10.0f;

if (\_yAngle > 360.0f)

{

\_yAngle = 0.0f;

}

break;

case Key.A:

\_yAngle -= 10.0f;

if (\_yAngle < 0.0f)

{

\_yAngle = 360.0f;

}

break;

case Key.Space:

\_animation = !\_animation;

break;

}

}

private void Window\_UpdateFrame(object sender, FrameEventArgs e)

{

\_window.Title = "©Матвей Попов М8О-308Б-20";

if (\_animation)

{

\_yAngle += Step;

}

}

private void Window\_RenderFrame(object sender, FrameEventArgs e)

{

GL.LoadIdentity();

GL.Clear(ClearBufferMask.ColorBufferBit |

ClearBufferMask.DepthBufferBit);

GL.Translate(0.0, 0.0, -\_scaling);

GL.Rotate(\_xAngle, 1.0, 0.0, 0.0);

GL.Rotate(\_yAngle, 0.0, 1.0, 0.0);

\_figure.Draw();

\_figure.LightConfigure(\_lightPositionX);

\_window.SwapBuffers();

}

}

}

**Program.cs**

namespace lab04

{

public static class MainClass

{

private const int Size = 500;

public static void Main()

{

var output = new Output(Size);

output.Start();

}

}

}

ЛИТЕРАТУРА

1. Документация OpenTK. [Электронный ресурс]

URL: <https://opentk.net/> (дата обращения 27.11.2022).

1. Документация C#. [Электронный ресурс]

URL: <https://learn.microsoft.com/ru-ru/dotnet/csharp/> (дата обращения 27.11.2022).