**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

Студент: Попова Наталья Сергеевна

Группа: 08-305

Преподаватель: Чернышов Л.Н.

Дата: 23.11.22

Оценка:

Москва, 2022

1. **Постановка задачи**

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

**Вариант №9:** Прозрачность вершины обратно пропорциональна расстоянию от заданной точки

1. **Описание программы**

Для реализации шейдерного эффекта были внесены изменения как в саму программу, так и в шейдеры.

Были добавлены две новые uniform: transparentPoint и transparentKoeff – заданная точка и коэффициент обратной пропорциональности.

int TransparentPointLoc = gl.GetUniformLocation(program, "transparentPoint");

int TransparentKoeffLoc = gl.GetUniformLocation(program, "transparentKoeff");

gl.Uniform3(TransparentPointLoc, (float)TransparentPoint.X, (float)TransparentPoint.Y, (float)TransparentPoint.Z);

gl.Uniform1(TransparentKoeffLoc, (float)TransparentKoeff);

В вершинном шейдере сохраняется глобальная координата вершины и пересылается фрагментному шейдеру.

#version 330 core

in vec3 position;

in vec3 normal;

out vec3 Normal;

out vec3 FragPos;

out vec3 GlobalPos;

uniform mat4 model;

uniform mat4 view;

uniform mat4 projection;

void main()

{

vec3 Vector\_Position = position;

gl\_Position = projection \* view \* model \* vec4(Vector\_Position, 1.0f);

FragPos = vec3(view \* model \* vec4(Vector\_Position, 1.0f));

GlobalPos = Vector\_Position;

Normal = mat3(transpose(inverse(view \* model))) \* normal;

}

Фрагментный шейдер по transparentPoint, transparentKoeff и глобальной координате вершины вычисляет прозрачность.

#version 330 core

out vec4 color;

in vec3 FragPos;

in vec3 Normal;

in vec3 GlobalPos;

uniform vec3 lightPos;

uniform vec3 viewPos;

uniform vec3 lightColor;

uniform vec3 objectColor;

uniform vec3 ambient;

uniform vec3 diffuse;

uniform vec3 spectr;

uniform int power;

uniform float time;

uniform vec3 transparentPoint;

uniform float transparentKoeff;

void main()

{

// Ambient

vec3 ambient = ambient \* lightColor;

// Diffuse

vec3 norm = normalize(Normal);

vec3 lightDir = normalize(lightPos - FragPos);

float diff = max(dot(norm, lightDir), 0.0);

vec3 diffuse = diffuse \* diff \* lightColor;

// Specular

vec3 viewDir = normalize(viewPos - FragPos);

vec3 reflectDir = reflect(-lightDir, norm);

float spec = pow(max(dot(viewDir, reflectDir), 0.0), power);

vec3 specular = spectr \* spec \* lightColor;

vec3 result = (ambient + diffuse + specular) \* objectColor;

color = vec4(result, 1.0f);

if(dot(norm, lightDir) < 0)

color = vec4(ambient \* objectColor, 1.0f);

//Animation

color.a \*= transparentKoeff / length(GlobalPos - transparentPoint);

}

1. **Набор тестов**

**Тест №1**

Положение (X, Y, Z): -90, 72, 0

TransparentPo (X, Y, Z): 120, 0, 20

TransparentKo: 80

**Тест №2**

Положение (X, Y, Z): 45, 30, 0

TransparentPo (X, Y, Z): 120, 0, 20

TransparentKo: 80

**Тест №3**

Положение (X, Y, Z): 45, 30, 0

TransparentPo (X, Y, Z): 25, 52, 20

TransparentKo: 80

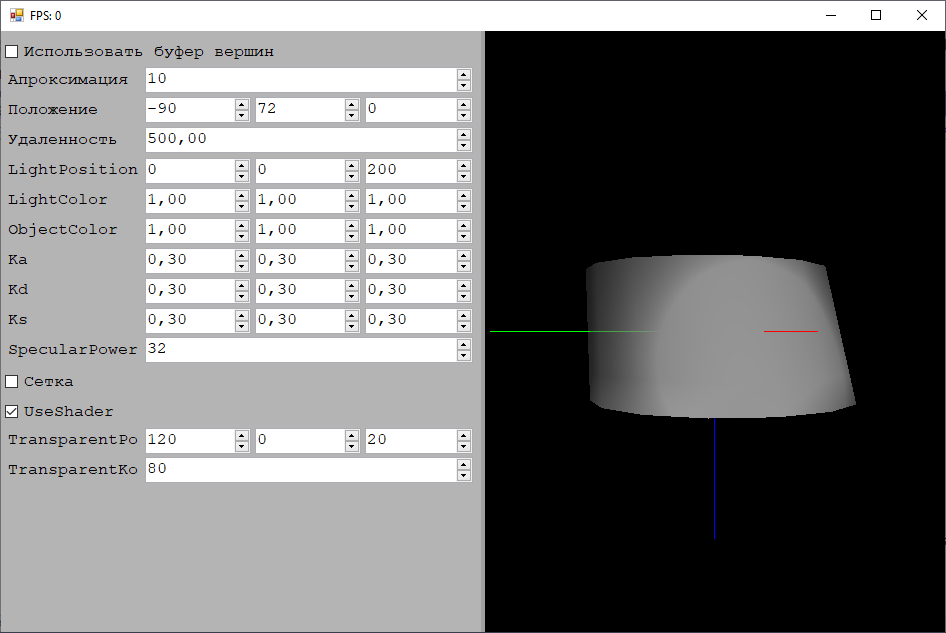
**Тест №4**

Положение (X, Y, Z): 45, 30, 0

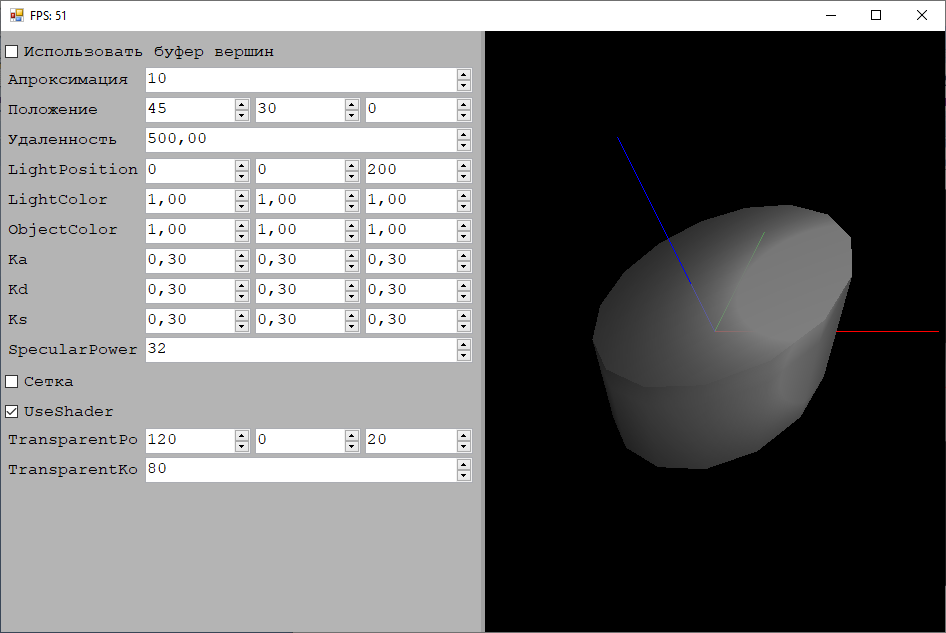
TransparentPo (X, Y, Z): 25, -145, -51

TransparentKo: 80

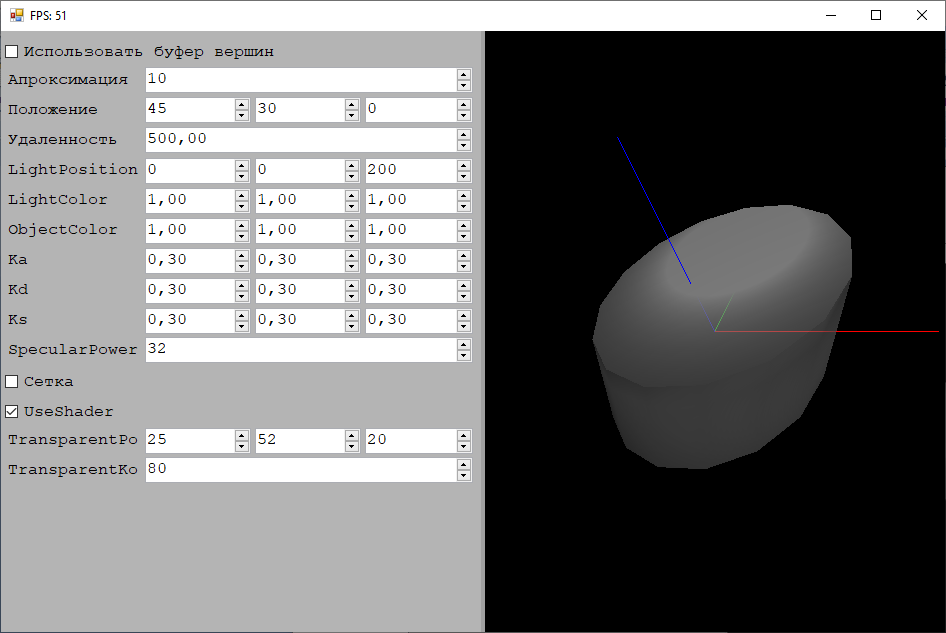
1. **Результаты выполнения тестов**



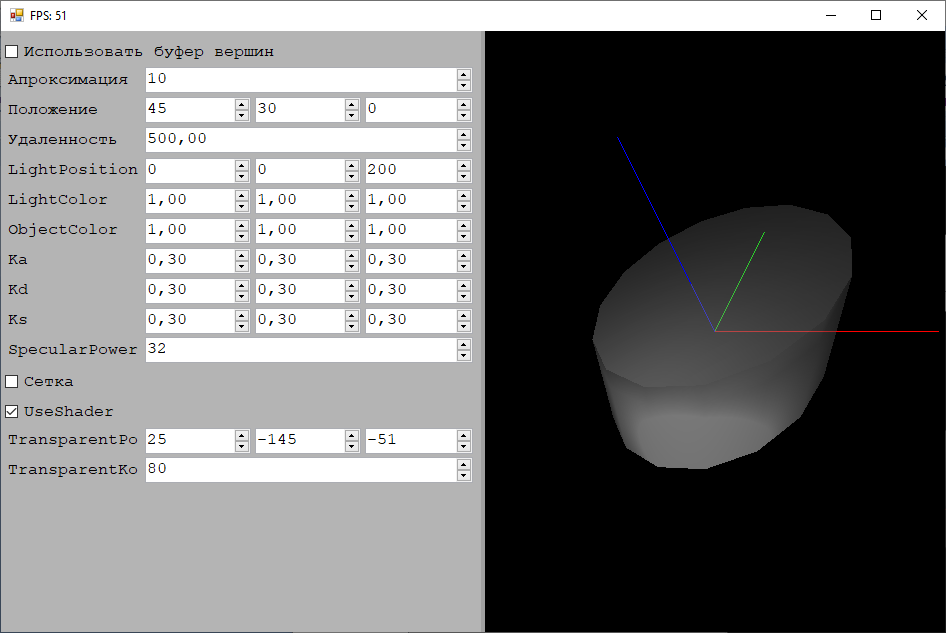
*Рисунок 1 - Тест №1*



*Рисунок 2 - Тест №2*



*Рисунок 3 - Тест №3*



*Рисунок 4 - Тест №4*

1. **Листинг программы**

using System;

using SharpGL;

using CGLabPlatform;

using System.Runtime.InteropServices;

using System.Windows.Forms;

using System.Diagnostics;

using System.Text;

// Создание и работа с классом приложения аналогична предыдущим примерам, только в

// в данном случае наследуемся от шаблона OGLApplicationTemplate<T>, в силу чего

// для вывода графики будет использоваться элемент управления OGLDevice работающий

// через OpenGL (далее OGL). Код OGLDevice размещается в Controls\OGLDevice.cs

public abstract class CGLabDemoOGL : OGLApplicationTemplate<CGLabDemoOGL>

{

[STAThread]

static void Main() { RunApplication(); }

#region Свойства

[DisplayCheckerProperty(false, "Использовать буфер вершин")]

public virtual bool useVBO { get; set; }

public double Height = 100; // высота цилиндра

[DisplayNumericProperty(30, 1, "Апроксимация", 6, 90)]

public abstract int Aprox { get; set; }

[DisplayNumericProperty(new[] { 0d, 0d, 0d }, 1, 0, "Положение камеры (X/Y/Z)")]

public virtual DVector3 cameraAngle

{

get { return Get<DVector3>(); }

set { if (Set(value)) UpdateModelViewMatrix(); }

}

[DisplayNumericProperty(300d, 0.1, 2, "Удаленность камеры")]

public virtual double cameraDistance

{

get { return Get<double>(); }

set { if (Set(value)) UpdateModelViewMatrix(); }

}

[DisplayNumericProperty(new[] { 0, 0d, 200d }, 1, "LightPosition")]

public DVector3 LightPosition { get; set; }

[DisplayNumericProperty(new[] { 1d, 1d, 1d }, 0.01, "LightColor", 0, 1)]

public DVector3 LightColor { get; set; }

[DisplayNumericProperty(new[] { 1d, 1d, 1d }, 0.01, "ObjectColor", 0, 1)]

public DVector3 ObjectColor { get; set; }

[DisplayNumericProperty(new[] { 0d, 0d, .30d }, 0.01, "Ka", 0, 1)]

public DVector3 Ka { get; set; }

[DisplayNumericProperty(new[] { 0d, 0d, 0d }, 0.01, "Kd", 0, 1)]

public DVector3 Kd { get; set; }

[DisplayNumericProperty(new[] { 1d, 0d, 0d }, 0.01, "Ks", 0, 1)]

public DVector3 Ks { get; set; }

[DisplayNumericProperty(32, 1, "SpecularPower", 1, 100)]

public abstract int SpecularPower { get; set; }

[DisplayCheckerProperty(false, "Сетка")]

public virtual bool Grid { get; set; }

[DisplayTextBoxProperty("X - красный", null)]

public abstract string Label1 { get; set; }

[DisplayTextBoxProperty("Y - зелёный", null)]

public abstract string Labe2 { get; set; }

[DisplayTextBoxProperty("Z - синий", null)]

public abstract string Label3 { get; set; }

[DisplayCheckerProperty(true, "UseShader")]

public bool UseShader {

get {

return Get<bool>();

}

set {

if(Set<bool>(value) )

RenderDevice.AddScheduleTask((gl, s) => gl.UseProgram(value?program:0));

}

}

#endregion

// Само создание объекта типа OpenGL осуществляется при создании устройства вывода (класс OGLDevice)

// и доступ к нему можно получить при помощи свойства gl данного объекта (RenderDevice) или объекта

// типа OGLDeviceUpdateArgs передаваемого в качестве параметра методу OnDeviceUpdate(). Данный метод,

// как и сама работа с устройством OpenGL реализуются в параллельном потоке. Обращение к устройству

// OpenGL из другого потока не допускается (создание многопоточного рендера возможно, но это достаточно

// специфическая архитектура, например рендинг частей экрана в текустуры а потом их объединение).

// Для большинства функций библиотеки OpenGL при отладке DEBUG конфигурации осуществляется проверка

// ошибок выполнения и их вывод в окно вывода Microsoft Visual Studio. Поэтому при отладке и написании

// кода связанного с OpenGL необходимо также контролировать ошибки библиотеки OpenGL в окне вывода.

uint program;

public uint[] attrib\_loc = new uint[2];

public uint[] uniform\_loc = new uint[3];

public uint[] buffers = new uint[2];

protected override void OnMainWindowLoad(object sender, EventArgs args)

{

base.VSPanelWidth = 260;

base.ValueStorage.RightColWidth = 60;

base.RenderDevice.VSync = 1;

#region Обработчики событий мыши и клавиатуры -------------------------------------------------------

RenderDevice.MouseMoveWithLeftBtnDown += (s, e) => cameraAngle += new DVector3(-e.MovDeltaY, -e.MovDeltaX, 0);

RenderDevice.MouseWheel += (s, e) => cameraDistance += e.Delta / 1000.0;

#endregion

// Как было отмеченно выше вся работа связанная с OGL должна выполнятся в одном потоке. Тут работа с OGL

// осуществляется в отдельном потоке, а метод OnMainWindowLoad() является событием возбуждаемым потоком

// пользовательского интерфейса (UI). Поэтой причине весь код ниже добавляется в диспетчер устройства

// вывода (метод AddScheduleTask() объекта RenderDevice) и выполняется ассинхронно в контексте потока

// OGL. Сам диспетчер является очередью типа FIFO (First In First Out - т.е. задания обрабатываются

// строго в порядке их поступления) и гарантирует, что все задания добавленные в OnMainWindowLoad()

// будут выполнены до первого вызова метода OnDeviceUpdate() (aka OnPaint)

#region Инициализация OGL и параметров рендера -----------------------------------------------------

RenderDevice.AddScheduleTask((gl, s) =>

{

//gl.Disable(OpenGL.GL\_DEPTH\_TEST);

gl.Enable(OpenGL.GL\_DEPTH\_TEST);

gl.Enable(OpenGL.GL\_CULL\_FACE);

gl.CullFace(OpenGL.GL\_BACK); // отбросить полигоны, повёрнутые изнанкой

gl.ClearColor(0, 0, 0, 0);

gl.FrontFace(OpenGL.GL\_CW); // обход вершин по часовой стрелки

//gl.PolygonMode(OpenGL.GL\_FRONT, OpenGL.GL\_FILL); // закрашиваются полигоны, повёрнутые лицевой стороной

gl.PolygonMode(OpenGL.GL\_FRONT, OpenGL.GL\_LINE);

gl.PolygonMode(OpenGL.GL\_BACK, OpenGL.GL\_FILL);

});

#endregion

#region Загрузка и компиляция шейдера

var errorhandler = new Action<string, object, object>((format, arg0, arg1) => {

string errormessage = String.Format(format, arg0, arg1);

Trace.WriteLine(errormessage);

throw new Exception(errormessage);

MessageBox.Show(errormessage, "SHADER CREATION ERROR", MessageBoxButtons.OK, MessageBoxIcon.Error);

Application.Exit();

});

RenderDevice.AddScheduleTask((gl, s) =>

{

var parameters = new int[1];

uint uit\_ver = gl.CreateShader(OpenGL.GL\_VERTEX\_SHADER);

uint uit\_frag = gl.CreateShader(OpenGL.GL\_FRAGMENT\_SHADER);

var compile\_shader = new Action<uint>(shader =>

{

gl.CompileShader(shader);

// Проверяем была ли компиляция выполнена успешно

gl.GetShader(shader, OpenGL.GL\_COMPILE\_STATUS, parameters);

if (parameters[0] != OpenGL.GL\_TRUE)

{

// В случае если компиляция не удалась пытаемся добиться от OGL

// что именно ему не понравилось. Для этого вначале получаем длину

// сообщения, выделим память под него, а затем уже запрашиваем

// скопировать туда само сообщение. В случае C# это выглядит

// немного иначе, но суть таже.

gl.GetShader(shader, OpenGL.GL\_INFO\_LOG\_LENGTH, parameters);

StringBuilder strbuilder = new StringBuilder(parameters[0]);

gl.GetShaderInfoLog(shader, parameters[0], IntPtr.Zero, strbuilder);

errorhandler("OpenGL Error: ошибка во время компиляции {1}.\n{0}",

strbuilder.ToString(), shader == uit\_ver ? "VERTEX\_SHADER"

: shader == uit\_frag ? "FRAGMENT\_SHADER" : "??????????\_SHADER");

}

});

var shader\_vert = HelpUtils.GetTextFileFromRes("shader.vert");

var shader\_frag = HelpUtils.GetTextFileFromRes("shader.frag");

//return;

gl.ShaderSource(uit\_ver, shader\_vert);

gl.ShaderSource(uit\_frag, shader\_frag);

compile\_shader(uit\_ver);

compile\_shader(uit\_frag);

program = gl.CreateProgram();

gl.AttachShader(program, uit\_ver);

gl.AttachShader(program, uit\_frag);

//gl.LinkProgram(program);

gl.LinkProgram(program);

gl.GetProgram(program, OpenGL.GL\_LINK\_STATUS, parameters);

if (parameters[0] != OpenGL.GL\_TRUE)

errorhandler("OpenGL Error: ошибка линковки програмы шейдера", null, null);

attrib\_loc[0] = (uint)gl.GetAttribLocation(program, "position");

attrib\_loc[1] = (uint)gl.GetAttribLocation(program, "normal");

uniform\_loc[0] = (uint)gl.GetUniformLocation(program, "projection");

uniform\_loc[1] = (uint)gl.GetUniformLocation(program, "view");

uniform\_loc[2] = (uint)gl.GetUniformLocation(program, "model");

gl.UseProgram(program);

});

#endregion

#region Инициализация буфера вершин -----------------------------------------------------------------

RenderDevice.AddScheduleTask((gl, s) =>

{

gl.EnableClientState(OpenGL.GL\_VERTEX\_ARRAY);

gl.EnableClientState(OpenGL.GL\_NORMAL\_ARRAY);

gl.EnableClientState(OpenGL.GL\_INDEX\_ARRAY);

gl.EnableClientState(OpenGL.GL\_COLOR\_ARRAY);

gl.GenBuffers(buffers.Length, buffers);

}, this);

#endregion

#region Уничтожение буфера вершин по завершению работы OGL ------------------------------------------

RenderDevice.Closed += (s, e) => // Событие выполняется в контексте потока OGL при завершении работы

{

var gl = e.gl;

gl.DisableClientState(OpenGL.GL\_VERTEX\_ARRAY);

gl.DisableClientState(OpenGL.GL\_NORMAL\_ARRAY);

gl.DisableClientState(OpenGL.GL\_INDEX\_ARRAY);

gl.DisableClientState(OpenGL.GL\_COLOR\_ARRAY);

};

#endregion

#region Обновление матрицы проекции при изменении размеров окна и запуске приложения ----------------

RenderDevice.Resized += (s, e) =>

{

var gl = e.gl;

gl.MatrixMode(OpenGL.GL\_PROJECTION);

pMatrix = Perspective(60, (double)e.Width / e.Height, 0.1, 100);

gl.LoadMatrix(pMatrix.ToArray(true));

};

#endregion

}

public DMatrix4 mMatrix;

public DMatrix4 vMatrix;

public DMatrix4 pMatrix;

private void UpdateModelViewMatrix() // метод вызывается при измении свойств cameraAngle и cameraDistance

{

#region Обновление объектно-видовой матрицы ---------------------------------------------------------

RenderDevice.AddScheduleTask((gl, s) =>

{

gl.MatrixMode(OpenGL.GL\_MODELVIEW);

var deg2rad = Math.PI / 180; // Вращается камера, а не сам объект

double phi = deg2rad \* cameraAngle.X;

double teta = deg2rad \* cameraAngle.Y;

double psi = deg2rad \* cameraAngle.Z;

// матрицы поворота вокруг осей

DMatrix3 RX = new DMatrix3(1, 0, 0,

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

0, Math.Sin(phi), Math.Cos(phi));

DMatrix3 RY = new DMatrix3(Math.Cos(teta), 0, Math.Sin(teta),

0, 1, 0,

-Math.Sin(teta), 0, Math.Cos(teta));

DMatrix3 RZ = new DMatrix3(Math.Cos(psi), -Math.Sin(psi), 0,

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

0, 0, 1);

var cameraTransform = (RX \* RY) \* RZ;

var cameraPosition = cameraTransform \* new DVector3(0, 0, cameraDistance);

var cameraUpDirection = cameraTransform \* new DVector3(0, 1, 0);

// Мировая матрица (преобразование локальной системы координат в мировую)

mMatrix = DMatrix4.Identity; // нет никаких преобразований над объекта

// Видовая матрица (переход из мировой системы координат к системе координат камеры)

vMatrix = LookAt(DMatrix4.Identity, cameraPosition, DVector3.Zero, cameraUpDirection);

// матрица ModelView

var mvMatrix = vMatrix \* mMatrix;

gl.LoadMatrix(mvMatrix.ToArray(true));

//gl.Rotate(45, 1f, 0f, 0);

//gl.Rotate(-45, 0f, 1f, 0);

});

#endregion

}

private static DMatrix4 Perspective(double verticalAngle, double aspectRatio, double nearPlane, double farPlane)

{

var radians = (verticalAngle / 2) \* Math.PI / 180;

var sine = Math.Sin(radians);

if (nearPlane == farPlane || aspectRatio == 0 || sine == 0)

return DMatrix4.Zero;

var cotan = Math.Cos(radians) / sine;

var clip = farPlane - nearPlane;

return new DMatrix4(

cotan / aspectRatio, 0, 0, 0,

0, cotan, 0, 0,

0, 0, -(nearPlane + farPlane) / clip, -(2.0 \* nearPlane \* farPlane) / clip,

0, 0, -1.0, 1.0

);

}

private static DMatrix4 LookAt(DMatrix4 matrix, DVector3 eye, DVector3 center, DVector3 up)

{

var forward = (center - eye).Normalized();

if (forward.ApproxEqual(DVector3.Zero, 0.00001))

return matrix;

var side = (forward \* up).Normalized();

var upVector = side \* forward;

var result = matrix \* new DMatrix4(

+side.X, +side.Y, +side.Z, 0,

+upVector.X, +upVector.Y, +upVector.Z, 0,

-forward.X, -forward.Y, -forward.Z, 0,

0, 0, 0, 1

);

result.M14 -= result.M11 \* eye.X + result.M12 \* eye.Y + result.M13 \* eye.Z;

result.M24 -= result.M21 \* eye.X + result.M22 \* eye.Y + result.M23 \* eye.Z;

result.M34 -= result.M31 \* eye.X + result.M32 \* eye.Y + result.M33 \* eye.Z;

result.M44 -= result.M41 \* eye.X + result.M42 \* eye.Y + result.M43 \* eye.Z;

return result;

}

#region Count Figure

[StructLayout(LayoutKind.Sequential, Pack = 1)]

public struct Vertex

{

// Координата

public readonly float vx, vy, vz;

// Нормаль

public readonly float nx, ny, nz;

// Цвет

public readonly float r, g, b;

public Vertex(float vx, float vy, float vz, float nx, float ny, float nz, float r, float g, float b)

{

this.vx = vx; this.vy = vy; this.vz = vz;

this.nx = nx; this.ny = ny; this.nz = nz;

this.r = r; this.g = g; this.b = b;

}

}

public Vertex[] vertices; // массив вершин

public int SIZE = 0; // размер массива вершин

public void Count\_Vertex() // получение размера массива (расчёт кол-ва вершин) и его создание

{

SIZE = 0;

//////////////////////// вершины верхней крышки

for (double j = 1; j < Height; j += (double)Aprox, TopCircles++)

{

CntVertInTopCircles = 0;

for (double phi = 0; phi <= 360; phi += (double)Aprox, CntVertInTopCircles++)

{

SIZE++;

}

}

///////////////////////////////////////////////////////

for (double j = 1; j < Height; j += (double)Aprox, BottomCircles++)

{

CntVertInBottomCircles = 0;

for (double phi = 0; phi <= 360; phi += (double)Aprox, CntVertInBottomCircles++)

{

SIZE++;

}

}

//////////////////////////////////////////////////боковая поверхность

for (double j = 0; j <= Height; j += (double)(Aprox), Circles++)

{

CntVertInCircle = 0;

for (double phi = 0; phi <= 360; phi += (double)Aprox, CntVertInCircle++)

{

SIZE++;

}

}

for (double phi = 0; phi <= 360; phi += (double)Aprox)

{

SIZE++;

}

Circles++;

SIZE += 2;

vertices = new Vertex[SIZE];

}

public int CntVertInCircle = 0;

public int Circles = 0;

public int CntVertInTopCircles = 0;

public int TopCircles = 0;

public int CntVertInBottomCircles = 0;

public int BottomCircles = 0;

public void Create\_Vertex()

{

CntVertInCircle = 0;

Circles = 0;

CntVertInTopCircles = 0;

TopCircles = 0;

CntVertInBottomCircles = 0;

BottomCircles = 0;

double X, Y, Z;

int iterator = 0;

//////////////////////// вершины верхней крышки

for (double j = 1; j < Height; j += (double)Aprox, TopCircles++)

{

CntVertInTopCircles = 0;

for (double phi = 0; phi <= 360; phi += (double)Aprox, CntVertInTopCircles++)

{

if ((360 - phi) < Aprox)

{

X = (Height - j);

Y = 0;

}

else

{

X = (Height - j) \* Math.Cos(phi \* Math.PI / 180);

Y = (-1) \* (Height - j) \* Math.Sin(phi \* Math.PI / 180);

}

Z = (Height / 2);

vertices[iterator] = new Vertex((float)X, (float)Y, (float)Z, 1, 1, 1, 0.0f, 1.0f, 0.0f);

iterator++;

}

}

///////////////////////////////////////////////////////

for (double j = 1; j < Height; j += (double)Aprox, BottomCircles++)

{

CntVertInBottomCircles = 0;

for (double phi = 0; phi <= 360; phi += (double)Aprox, CntVertInBottomCircles++)

{

if ((360 - phi) < Aprox)

{

X = (Height - j);

Y = 0;

Z = ((-Height / 2 + j / 2) \* Math.Cos(0));

}

else

{

X = (Height - j) \* Math.Cos(phi \* Math.PI / 180);

Y = (-1) \* (Height - j) \* Math.Sin(phi \* Math.PI / 180);

Z = ((-Height / 2.0 + j / 2.0) \* Math.Cos(phi \* Math.PI / 180));

}

vertices[iterator] = new Vertex((float)X, (float)Y, (float)Z, 1, 1, 1, 0.0f, 0.0f, 1.0f);

iterator++;

}

}

//////////////////////////////////////////////////боковая поверхность

//for (int i = CntVertInTopCircles \* TopCircles; i < CntVertInTopCircles \* TopCircles + CntVertInBottomCircles; i++)

//{

// vertices[iterator] = new Vertex(vertices[i].vx, vertices[i].vy, vertices[i].vz, vertices[i].nx, vertices[i].ny, vertices[i].nz, 1f, 0, 0);

// iterator++;

//}

//Circles++;

for (double j = 0; j <= Height; j += (double)(Aprox), Circles++)

{

CntVertInCircle = 0;

for (double phi = 0; phi <= 360; phi += (double)Aprox, CntVertInCircle++)

{

if ((360 - phi) < Aprox)

{

X = Height;

Y = 0;

Z = (((-Height / 2 + j / 2) + j / 2));

}

else

{

X = Height \* Math.Cos(phi \* Math.PI / 180);

Y = (-1) \* Height \* Math.Sin(phi \* Math.PI / 180);

Z = (((-Height / 2 + j / 2) \* Math.Cos(phi \* Math.PI / 180) + j / 2) / 1);

}

vertices[iterator] = new Vertex((float)X, (float)Y, (float)Z, 1, 1, 1, 1.0f, 0.0f, 0.0f);

iterator++;

}

}

for (double phi = 0; phi <= 360; phi += (double)Aprox)

{

if ((360 - phi) < Aprox)

{

double a = Height \* (1 - Math.Cos(0 \* Math.PI / 180));

double b = Height \* (1 - Math.Sin(0 \* Math.PI / 180));

X = Height;

Y = 0;

Z = (Height / 2);

}

else

{

X = Height \* Math.Cos(phi \* Math.PI / 180);

Y = (-1) \* Height \* Math.Sin(phi \* Math.PI / 180);

Z = (Height / 2);

}

vertices[iterator] = new Vertex((float)X, (float)Y, (float)Z, 1, 1, 1, 1.0f, 0.0f, 0.0f);

iterator++;

}

//for (int i = 0; i < CntVertInTopCircles; i++)

//{

// vertices[iterator] = new Vertex(vertices[i].vx, vertices[i].vy, vertices[i].vz, vertices[i].nx, vertices[i].ny, vertices[i].nz, 1f, 0, 0);

// iterator++;

//}

Circles++;

//int visited = TopCircles \* CntVertInTopCircles + BottomCircles \* CntVertInBottomCircles;

//for (int i = 0; i < Circles; i++)

//{

// vertices[visited + CntVertInCircle - 1] = vertices[visited];

// visited += (CntVertInCircle);

//}

////////////////////////////////////////////////////центры верхней и нижней крышки

vertices[iterator] = new Vertex((float)0, (float)0, (float)(Height / 2), 1, 1, 1, 0.0f, 1.0f, 0.0f);

iterator++;

vertices[iterator] = new Vertex((float)0, (float)0, (float)0, 1, 1, 1, 0.0f, 0.0f, 1.0f);

iterator++;

}

public int Polygons = 0; // число полигонов

public void Count\_Polygons()

{

Polygons = 0;

for (int j = 1; j < Circles; j++)

{

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

{

Polygons += 3;

}

}

for (int j = 1; j < Circles - 0; j++)

{

for (int i = j \* CntVertInCircle; i < (j + 1) \* CntVertInCircle - 1; i++)

{

Polygons += 3;

}

}

//разбивка на полигоны верхней крышки

for (int j = 1; j < TopCircles; j++)

{

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

{

Polygons += 3;

}

}

for (int j = 1; j < TopCircles - 0; j++)

{

for (int i = j \* CntVertInTopCircles; i < (j + 1) \* CntVertInTopCircles - 1; i++)

{

Polygons += 3;

}

}

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

{

Polygons += 3;

}

//разбивка на полигоны нижней крышки

for (int j = 1; j < BottomCircles; j++)

{

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

{

Polygons += 3;

}

}

for (int j = 1; j < BottomCircles; j++)

{

for (int i = j \* CntVertInBottomCircles; i < (j + 1) \* CntVertInBottomCircles - 1; i++)

{

Polygons += 3;

}

}

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

{

Polygons += 3;

}

indices = new int[Polygons];

}

public int[] indices; // массив индексов вершин

public void Create\_Polygons()

{

int iterator = 0;

int visited = 0;

//разбивка на полигоны верхней крышки

for (int j = 1; j < TopCircles; j++)

{

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

{

int a = (j - 1) \* CntVertInTopCircles + i + visited;

int b = (j - 1) \* CntVertInTopCircles + i + 1 + visited;

int c = j \* CntVertInTopCircles + i + visited;

indices[iterator] = a; iterator++;

indices[iterator] = b; iterator++;

indices[iterator] = c; iterator++;

Count\_Normal(a, b, c);

}

}

int z = 1;

for (int j = 1; j < TopCircles; j++)

{

for (int i = j \* CntVertInTopCircles; i < (j + 1) \* CntVertInTopCircles - 1; i++)

{

int a = i + visited;

int b = z + visited;

int c = i + 1 + visited;

indices[iterator] = a; iterator++;

indices[iterator] = b; iterator++;

indices[iterator] = c; iterator++;

z++;

Count\_Normal(a, b, c);

}

z++;

}

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

{

int a = (TopCircles - 1) \* CntVertInTopCircles + i + visited;

int b = (TopCircles - 1) \* CntVertInTopCircles + i + 1 + visited;

int c = vertices.Length - 2;

indices[iterator] = a; iterator++;

indices[iterator] = b; iterator++;

indices[iterator] = c; iterator++;

Count\_Normal(a, b, c);

}

// разбивка на полигоны нижней крышки

visited += TopCircles \* CntVertInTopCircles;

for (int j = 1; j < BottomCircles; j++)

{

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

{

int a = (j - 1) \* CntVertInBottomCircles + i + visited;

int b = j \* CntVertInBottomCircles + i + visited;

int c = (j - 1) \* CntVertInBottomCircles + i + 1 + visited;

indices[iterator] = a; iterator++;

indices[iterator] = b; iterator++;

indices[iterator] = c; iterator++;

Count\_Normal(a, b, c);

}

}

z = 1;

for (int j = 1; j < BottomCircles; j++)

{

for (int i = j \* CntVertInBottomCircles; i < (j + 1) \* CntVertInBottomCircles - 1; i++)

{

int a = i + visited;

int b = i + 1 + visited;

int c = z + visited;

indices[iterator] = a; iterator++;

indices[iterator] = b; iterator++;

indices[iterator] = c; iterator++;

z++;

Count\_Normal(a, b, c);

}

z++;

}

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

{

int a = (BottomCircles - 1) \* CntVertInBottomCircles + i + visited;

int b = vertices.Length - 1;

int c = (BottomCircles - 1) \* CntVertInBottomCircles + i + 1 + visited;

indices[iterator] = a; iterator++;

indices[iterator] = b; iterator++;

indices[iterator] = c; iterator++;

Count\_Normal(a, b, c);

}

visited += BottomCircles \* CntVertInBottomCircles;

//разбивка на полигоны бокового основания

for (int j = 1; j < Circles; j++)

{

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

{

int a = (j - 1) \* CntVertInCircle + i + visited;

int b = (j - 1) \* CntVertInCircle + i + 1 + visited;

int c = j \* CntVertInCircle + i + visited;

indices[iterator] = a; iterator++;

indices[iterator] = b; iterator++;

indices[iterator] = c; iterator++;

Count\_Normal(a, b, c);

}

}

int k = 1;

for (int j = 1; j < Circles; j++)

{

for (int i = j \* CntVertInCircle; i < (j + 1) \* CntVertInCircle - 1; i++)

{

int a = i + visited;

int b = k + visited;

int c = 1 + i + visited;

indices[iterator] = a; iterator++;

indices[iterator] = b; iterator++;

indices[iterator] = c; iterator++;

k++;

Count\_Normal(a, b, c);

}

k++;

}

}

public void Count\_Normal(int a, int b, int c)

{

DVector3 vec1 = new DVector3(vertices[a].vx, vertices[a].vy, vertices[a].vz);

DVector3 vec2 = new DVector3(vertices[b].vx, vertices[b].vy, vertices[b].vz);

DVector3 vec3 = new DVector3(vertices[c].vx, vertices[c].vy, vertices[c].vz);

DVector3 normal = (-1) \* DVector3.CrossProduct(vec1 - vec3, vec2 - vec1);

//normal = (-1) \* DVector3.CrossProduct(vec2 - vec1, vec3 - vec2);

vertices[a] = new Vertex(vertices[a].vx, vertices[a].vy, vertices[a].vz, (float)normal.X, (float)normal.Y, (float)normal.Z, vertices[a].r, vertices[a].g, vertices[a].b);

normal = (-1) \* DVector3.CrossProduct(vec2 - vec1, vec3 - vec2);

vertices[b] = new Vertex(vertices[b].vx, vertices[b].vy, vertices[b].vz, (float)normal.X, (float)normal.Y, (float)normal.Z, vertices[b].r, vertices[b].g, vertices[b].b);

normal = (-1) \* DVector3.CrossProduct(vec3 - vec2, vec1 - vec3);

vertices[c] = new Vertex(vertices[c].vx, vertices[c].vy, vertices[c].vz, (float)normal.X, (float)normal.Y, (float)normal.Z, vertices[c].r, vertices[c].g, vertices[c].b);

}

//public Vertex[] vertices = {

// // vx vy vz nx ny nz r g b

// new Vertex( .5f, .5f, .5f, nf, nf, nf, 1f, 1f, 1f),

// new Vertex(-.5f, .5f, .5f, -nf, nf, nf, 1f, 1f, 0f),

// new Vertex(-.5f, -.5f, .5f, -nf, -nf, nf, 1f, 0f, 0f),

// new Vertex( .5f, -.5f, .5f, nf, -nf, nf, 1f, 0f, 1f),

// new Vertex( .5f, -.5f, -.5f, nf, -nf, -nf, 0f, 0f, 1f),

// new Vertex( .5f, .5f, -.5f, nf, nf, -nf, 0f, 1f, 1f),

// new Vertex(-.5f, .5f, -.5f, -nf, nf, -nf, 0f, 1f, 0f),

// new Vertex(-.5f, -.5f, -.5f, -nf, -nf, -nf, 0f, 0f, 0f) };

//public static float nf = (float)(1 / Math.Sqrt(3));

//public uint[] indices = {

// // Первая последовательность

// 5, 6, 0, 1, // {v0,v5,v6,v1} - верхня грань

// /\* 0, 1 \*/ 3, 2, // {v0,v1,v2,v3} - передняя грань

// /\* 3, 2 \*/ 4, 7, // {v7,v4,v3,v2} - нижняя грань

// // Вторая последовательность

// 2, 1, 7, 6, // {v1,v6,v7,v2} - левая грань

// /\* 7, 6 \*/ 4, 5, // {v4,v7,v6,v5} - задняя грань

// /\* 4, 5 \*/ 3, 0 // {v0,v3,v4,v5} - правая грань

//};

#endregion;

public void Draw\_Axes(OGLDeviceUpdateArgs e)

{

var gl = e.gl;

#region Axes

gl.LoadIdentity();

gl.PushMatrix();

//gl.Rotate(45, 1f, 0f, 0);

//gl.Rotate(-45, 0f, 1f, 0);

gl.Color(1f, 0f, 0);

gl.Begin(OpenGL.GL\_LINES);

gl.Vertex(0, 0, 0);

gl.Vertex(1000, 0, 0);

gl.End();

gl.Color(0, 1f, 0);

gl.Begin(OpenGL.GL\_LINES);

gl.Vertex(0, 0, 0);

gl.Vertex(0, 1000, 0);

gl.End();

gl.Color(0f, 0f, 1f);

gl.Begin(OpenGL.GL\_LINES);

gl.Vertex(0, 0, 0);

gl.Vertex(0, 0, 2000);

gl.End();

gl.PopMatrix();

UpdateModelViewMatrix();

#endregion;

}

protected unsafe override void OnDeviceUpdate(object s, OGLDeviceUpdateArgs e)

{

var gl = e.gl;

UpdateModelViewMatrix();

// Очищаем буфер экрана и буфер глубины (иначе рисоваться все будет поверх старого)

gl.Clear(OpenGL.GL\_COLOR\_BUFFER\_BIT | OpenGL.GL\_DEPTH\_BUFFER\_BIT | OpenGL.GL\_STENCIL\_BUFFER\_BIT);

Count\_Vertex();

Create\_Vertex();

Count\_Polygons();

Create\_Polygons();

Draw\_Axes(e);

int offset\_vx = (int)Marshal.OffsetOf(typeof(Vertex), "vx");

int offset\_nx = (int)Marshal.OffsetOf(typeof(Vertex), "nx");

int offset\_col = (int)Marshal.OffsetOf(typeof(Vertex), "r");

// Рендинг сцены реализуется одним из двух методов - VB (Vertex Buffer) или VA (Vertex Array),

// в зависимости от выбранного пользователем режима.

if (!useVBO)

#region Рендинг сцены методом VA (Vertex Array) -----------------------------------------------------

{

fixed (Vertex\* vrt = vertices)

{

gl.VertexPointer(3, OpenGL.GL\_FLOAT, sizeof(Vertex), IntPtr.Add((IntPtr)vrt, offset\_vx));

gl.NormalPointer(OpenGL.GL\_FLOAT, sizeof(Vertex), IntPtr.Add((IntPtr)vrt, offset\_nx));

gl.ColorPointer(3, OpenGL.GL\_FLOAT, sizeof(Vertex), IntPtr.Add((IntPtr)vrt, offset\_col));

}

fixed (int\* ptr = indices)

{

gl.IndexPointer(OpenGL.GL\_INT, 0, (IntPtr)(&ptr[0]));

}

if (UseShader)

{

gl.EnableVertexAttribArray(attrib\_loc[0]);

gl.EnableVertexAttribArray(attrib\_loc[1]);

fixed (Vertex\* vrt = vertices)

{

gl.VertexAttribPointer(attrib\_loc[0], 3, OpenGL.GL\_FLOAT, false, sizeof(Vertex), IntPtr.Add((IntPtr)(vrt), offset\_vx));

gl.VertexAttribPointer(attrib\_loc[1], 3, OpenGL.GL\_FLOAT, false, sizeof(Vertex), IntPtr.Add((IntPtr)(vrt), offset\_nx));

}

int objectColorLoc = gl.GetUniformLocation(program, "objectColor");

int lightColorLoc = gl.GetUniformLocation(program, "lightColor");

int lightPosLoc = gl.GetUniformLocation(program, "lightPos");

int viewPosLoc = gl.GetUniformLocation(program, "viewPos");

int AmbientPosLoc = gl.GetUniformLocation(program, "ambient");

int DiffusePosLoc = gl.GetUniformLocation(program, "diffuse");

int SpectrPosLoc = gl.GetUniformLocation(program, "spectr");

int PowerLoc = gl.GetUniformLocation(program, "power");

gl.Uniform3(objectColorLoc, (float)ObjectColor.X, (float)ObjectColor.Y, (float)ObjectColor.Z);

gl.Uniform3(lightColorLoc, (float)LightColor.X, (float)LightColor.Y, (float)LightColor.Z);

gl.Uniform3(lightPosLoc, (float)LightPosition.X, (float)LightPosition.Y, (float)LightPosition.Z);

gl.Uniform3(viewPosLoc, (float)LightPosition.X, (float)LightPosition.Y, (float)LightPosition.Z);

//gl.Uniform3(viewPosLoc, (float)0, (float)0, (float)cameraDistance);

gl.Uniform3(AmbientPosLoc, (float)Ka.X, (float)Ka.Y, (float)Ka.Z);

gl.Uniform3(DiffusePosLoc, (float)Kd.X, (float)Kd.Y, (float)Kd.Z);

gl.Uniform3(SpectrPosLoc, (float)Ks.X, (float)Ks.Y, (float)Ks.Z);

gl.Uniform1(PowerLoc, SpecularPower);

gl.UniformMatrix4((int)uniform\_loc[0], 1, false, pMatrix.ToFloatArray(true));

gl.UniformMatrix4((int)uniform\_loc[1], 1, false, vMatrix.ToFloatArray(true));

gl.UniformMatrix4((int)uniform\_loc[2], 1, false, mMatrix.ToFloatArray(true));

}

fixed (int\* ptr = indices)

{

gl.DrawElements(OpenGL.GL\_TRIANGLES, indices.Length, (IntPtr)(&ptr[0]));

}

gl.DisableVertexAttribArray(attrib\_loc[0]);

gl.DisableVertexAttribArray(attrib\_loc[1]);

if (Grid)

gl.PolygonMode(OpenGL.GL\_FRONT\_AND\_BACK, OpenGL.GL\_LINE);

else

gl.PolygonMode(OpenGL.GL\_FRONT, OpenGL.GL\_FILL);

}

#endregion

else

#region Рендинг сцены методом VBO (Vertex Buffer Object) --------------------------------------------

{

fixed (uint\* ptr = buffers)

{

gl.BindBuffer(OpenGL.GL\_ARRAY\_BUFFER, ptr[0]);

gl.BindBuffer(OpenGL.GL\_ELEMENT\_ARRAY\_BUFFER, ptr[1]);

}

/////////////////////

fixed (Vertex\* vrt = vertices)

{

gl.VertexPointer(3, OpenGL.GL\_FLOAT, sizeof(Vertex), new IntPtr(offset\_vx));

gl.NormalPointer(OpenGL.GL\_FLOAT, sizeof(Vertex), new IntPtr(offset\_nx));

gl.ColorPointer(3, OpenGL.GL\_FLOAT, sizeof(Vertex), new IntPtr(offset\_col));

}

fixed (Vertex\* ptr = vertices)

{

gl.BufferData(OpenGL.GL\_ARRAY\_BUFFER, vertices.Length \* sizeof(Vertex), IntPtr.Add((IntPtr)ptr, offset\_vx), OpenGL.GL\_STATIC\_DRAW);

}

fixed (int\* ptr = indices)

{

gl.BufferData(OpenGL.GL\_ELEMENT\_ARRAY\_BUFFER, indices.Length \* sizeof(int), (IntPtr)(&ptr[0]), OpenGL.GL\_STATIC\_DRAW);

}

fixed (int\* ptr = indices)

{

gl.DrawElements(OpenGL.GL\_TRIANGLES, indices.Length, OpenGL.GL\_UNSIGNED\_INT, new IntPtr(0 \* sizeof(uint)));

}

gl.DeleteBuffers(2, buffers);

if (Grid)

gl.PolygonMode(OpenGL.GL\_FRONT\_AND\_BACK, OpenGL.GL\_LINE);

else

gl.PolygonMode(OpenGL.GL\_FRONT, OpenGL.GL\_FILL);

}

#endregion

}

}

#version 330 core

in vec3 position;

in vec3 normal;

out vec3 Normal;

out vec3 FragPos;

uniform mat4 model;

uniform mat4 view;

uniform mat4 projection;

void main()

{

gl\_Position = projection \* view \* model \* vec4(position, 1.0f);

FragPos = vec3(view \* model \* vec4(position, 1.0f));

Normal = mat3(transpose(inverse(view \* model))) \* normal;

}

#version 330 core

out vec4 color;

in vec3 FragPos;

in vec3 Normal;

uniform vec3 lightPos;

uniform vec3 viewPos;

uniform vec3 lightColor;

uniform vec3 objectColor;

uniform vec3 ambient;

uniform vec3 diffuse;

uniform vec3 spectr;

uniform int power;

void main()

{

// Ambient

vec3 ambient = ambient \* lightColor;

// Diffuse

vec3 norm = normalize(Normal);

vec3 lightDir = normalize(lightPos - FragPos);

float diff = max(dot(norm, lightDir), 0.0);

vec3 diffuse = diffuse \* diff \* lightColor;

// Specular

vec3 viewDir = normalize(viewPos - FragPos);

vec3 reflectDir = reflect(-lightDir, norm);

float spec = pow(dot(viewDir, reflectDir), power);

vec3 specular = spectr \* spec \* lightColor;

vec3 result = (ambient + diffuse + specular) \* objectColor;

color = vec4(result, 1.0f);

if(dot(norm, lightDir) < 0)

color = vec4(ambient \* objectColor, 1.0f);

//else if(dot(norm, viewDir) < 0)

//color = vec4((ambient + diffuse) \* objectColor, 1.0f);

}

**ЛИТЕРАТУРА**

* + - 1. PyOpenGL с шейдерами [Электронный ресурс] URL: <https://habr.com/ru/post/247123/> (дата обращения: 20.11.22)
      2. Прозрачность - Уроки OpenGL [Электронный ресурс] URL: <http://opengl-tutorial.blogspot.com/p/10.html> (дата обращения: 21.11.22)
      3. OpenGL: Прозрачность [Электронный ресурс] URL: <https://masandilov.ru/opengl/transparency> (дата обращения: 21.11.22)