Московский авиационный институт

(Национальный исследовательский университет)

Факультет: «Информационные технологии и прикладная математика»

Кафедра: «Вычислительная математика и программирование»

Дисциплина: «Компьютерная графика»

Лабораторная работа №4-6 по курсу «Компьютерная графика»

Тема: Ознакомление с технологией OpenGL и языком GLSL. Создание шейдерных анимационных эффектов в OpenGL 2.1.

Студент: Тимофеев А. В.

Преподаватель: Морозов А. В.

Группа: М80-307Б

Дата:

Оценка:

Подпись:

Москва, 2022

**Постановка задачи**

**Задание**: 1)Создать графическое приложение с использованием OpenGL. Используя результаты Л.Р.№3, изобразить заданное тело (то же, что и в л.р. №3) с использованием средств OpenGL 2.1. Использовать буфер вершин. Точность аппроксимации тела задается пользователем. Обеспечить возможность вращения и масштабирования многогранника и удаление невидимых линий и поверхностей. Реализовать простую модель освещения на GLSL. Параметры освещения и отражающие свойства материала задаются пользователем в диалоговом режиме.  
2)Для поверхности, созданной в л.р. №5, обеспечить выполнение следующего шейдерного эффекта:  
Вариант: 5. Анимация. Координата X изменяется по закону X\*cos(t), координата Y изменяется по закону Y = Ysin(X+t).

**Решение задачи**

Возьмем за основу способ генерации фигуры и вычисление интенсивностей цвета такие же, какие они былы в Л.Р. 3. При этом важно понимать, что затенение Фонга отличается от Гуро тем, что при его использовании для определения цвета в каждой точке интерполируются не интенсивности отраженного света, а векторы нормалей - это будет осуществляться автоматически при загрузке нормалей как выходные атрибуты в вершинном шейдере OpenGL. Стоит также отметить, что все вычисления, связанные с освещением, будут происходить исключительно во фрагментном шейдере OpenGL. Для корректной отрисовки фигуры в среде OpenGL необходимо задать матрицу проекций (свойство OpenGL.GL\_PROJECTION) и объектно-видовую матрицу (свойство OpenGL.GL\_MODELVIEW) - их следует предварительно вычислить и только затем подгрузить в OpenGL. После задания этих матриц будет автоматически производиться следующее преобразование для координат некоторой вершины P :



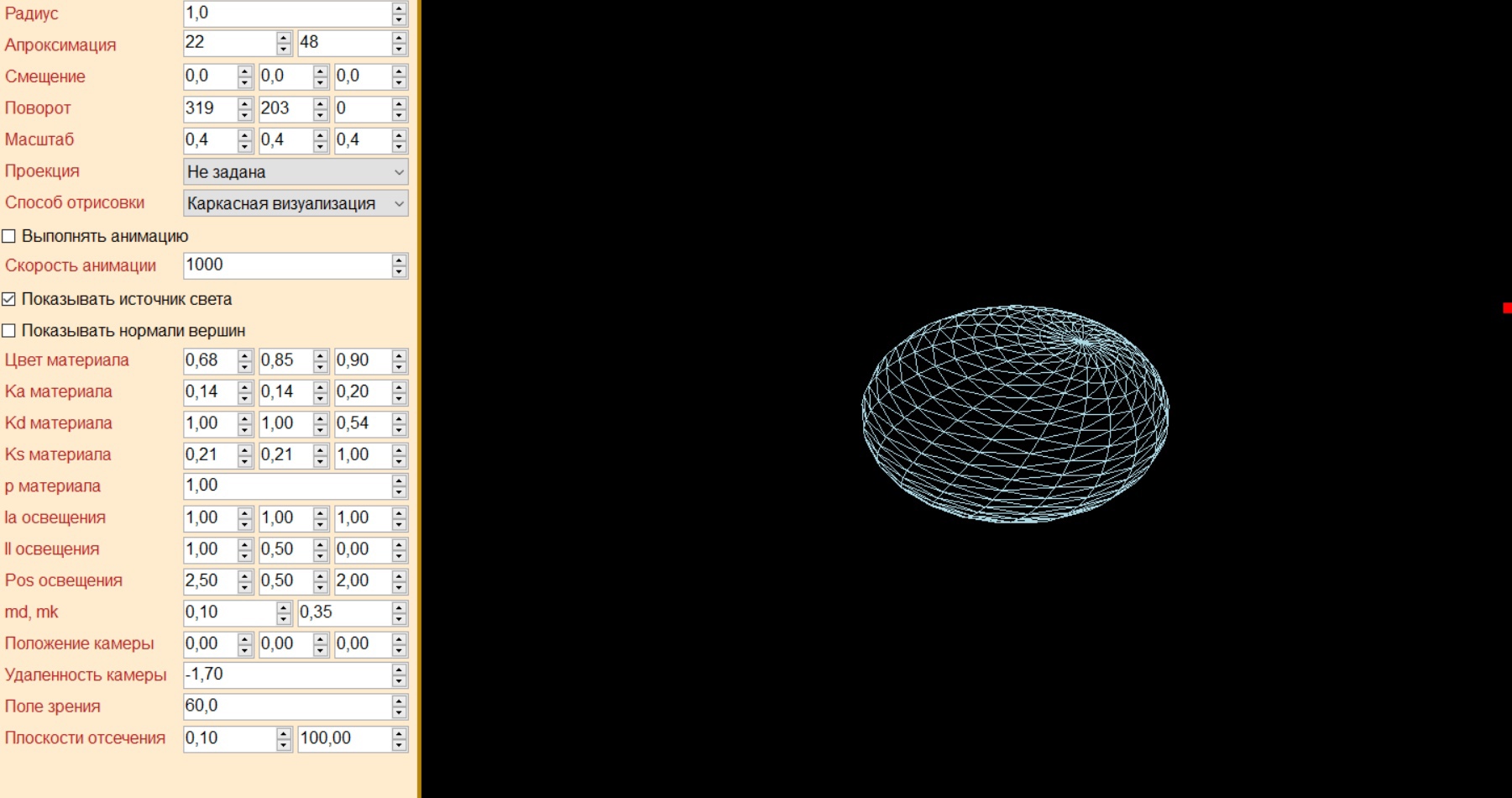
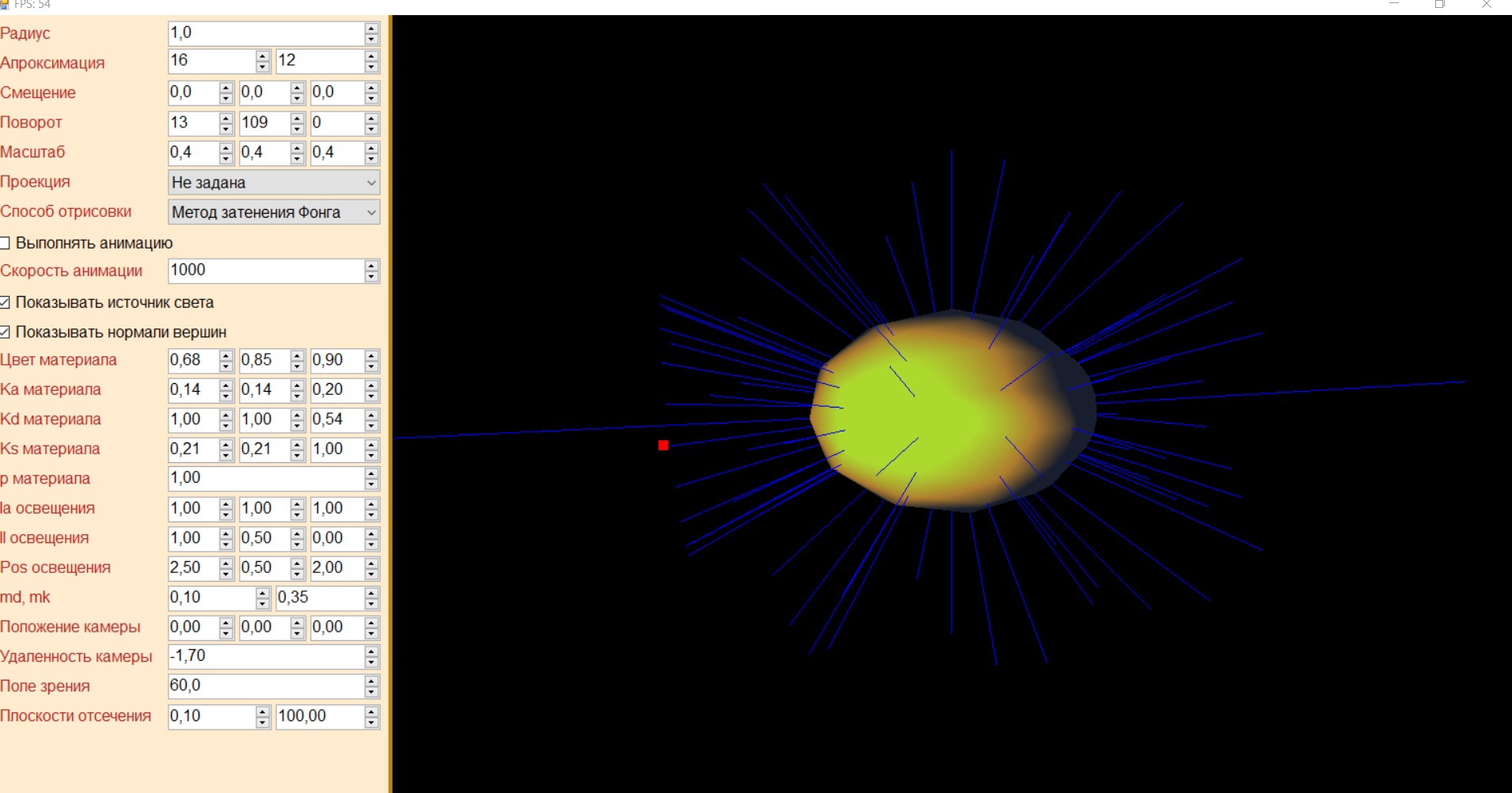
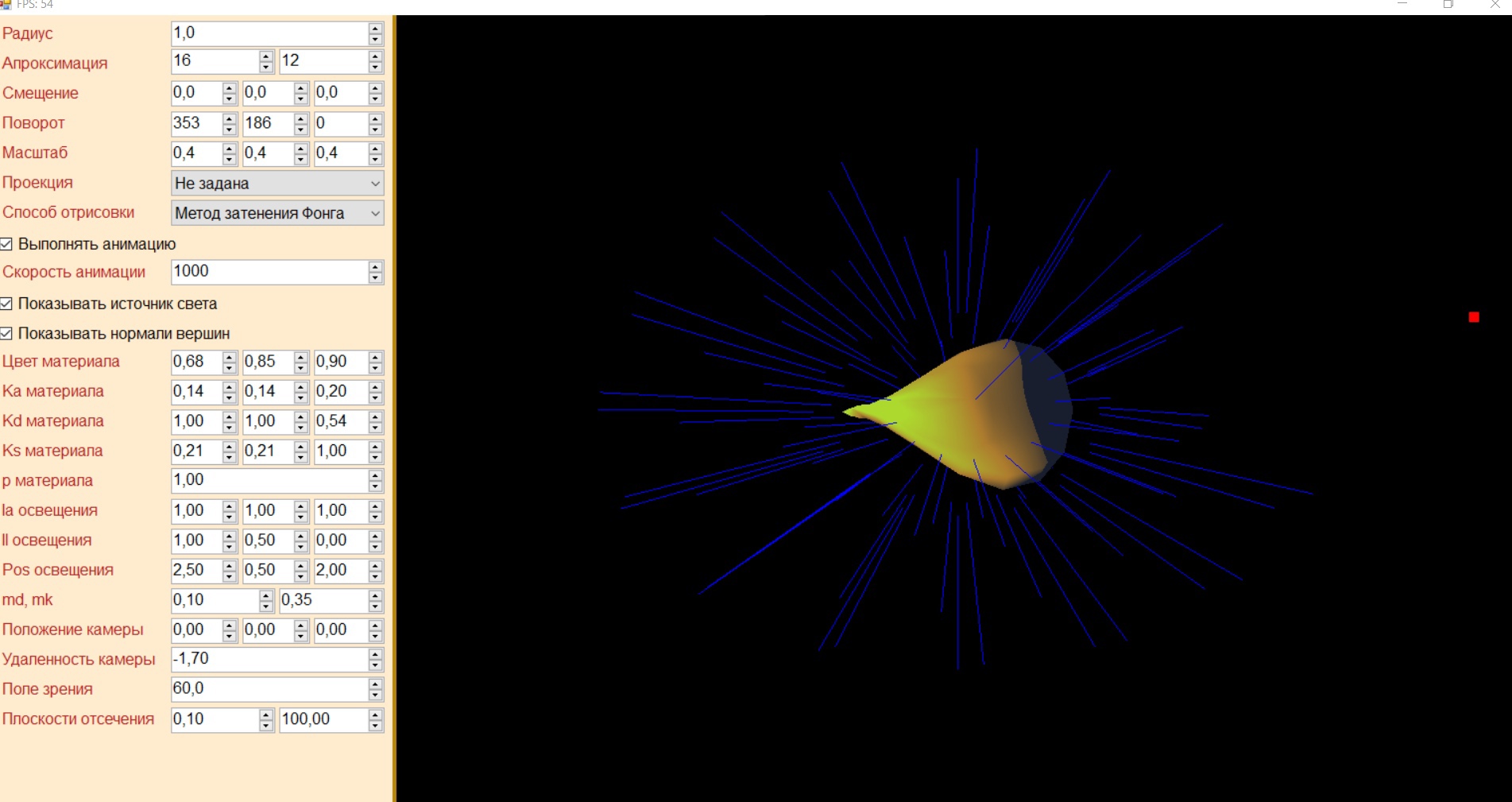
Чтобы вся информация о фигуре хранилось исключительно в GPU необходимо загружать её в буфер вершин VBO. При этом для вызова метода отрисовки OpenGL необходимо знать, в каком порядке следуют вершины в массиве. При работе с прямым эллиптическим цилиндром будем использоваться порядок обхода GL\_TRIANGLE\_FAN - для оснований, и GL\_TRIANGLE\_STRIP - для боковой поверхности (вызывая этот метод отрисовки таким обходом для каждого этажа соответственно).

Для создания требуемого шейдерного эффекта достаточно немного изменить вершинный шейдер - менять указанным способом координаты Y загружаемых вершин:

Y = Y · cos(t + Y )

Обновленная величина времени t будет подгружаться в шейдер каждый раз при выполнении основного цикла работы программы. Стоит отметить, что при изменении координат вершин не требуется пересчитывать все ранее вычисленные нормали фигуры - изменения при анимации незначительные и в принципе этого не требуют.

**Пример работы**



**Листинг программы**

(классы)

**Program.cs**

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

public struct Vertex{

public Vertex(float px, float py, float pz, float pw, byte pr, byte pg, byte pb){

vx = px;

vy = py;

vz = pz;

vw = pw;

nx = 0;

ny = 0;

nz = 0;

nw = 0;

r = pr;

g = pg;

b = pb;

}

public void SetNorm(float nx, float ny, float nz, float nw){

this.nx = nx;

this.nx = ny;

this.nx = nz;

this.nx = nw;

}

public readonly float vx, vy, vz, vw;

public float nx, ny, nz, nw;

public readonly byte r, g, b;

}

private void ChangeNormale(ref Vertex vertex, DVector4 normale){

var check = new DVector4(vertex.nx, vertex.ny, vertex.nz, 0);

check += normale;

vertex.nx = (float) check.X;

vertex.ny = (float) check.Y;

vertex.nz = (float) check.Z;

}

**shader1.vert**

#version 150 core

attribute vec4 Normal;

attribute vec4 Coord;

out vec3 FragNormale;

out vec3 FragVertex;

uniform mat4 Projection;

uniform mat4 ModelView;

uniform mat4 NormalMatrix;

uniform mat4 ModelMatrix;

uniform mat4 PointMatrix;

uniform float Time;

void main(void){

vec4 Coord\_now = Coord;

if (Time > 0){

Coord\_now.x = Coord.x \* cos(Time);

Coord\_now.y = Coord.y \* sin(Time + Coord\_now.x);

}

FragVertex = vec3(PointMatrix \* Coord\_now);

FragNormale = vec3(NormalMatrix \* Normal);

FragNormale = normalize(FragNormale);

gl\_Position = (Projection \* ModelView) \* Coord\_now;

}

**shader2.frag**

#version 150 core

in vec3 FragNormale;

in vec3 FragVertex;

uniform vec3 Ka\_Material;

uniform vec3 Kd\_Material;

uniform vec3 Ks\_Material;

uniform float P\_Material;

uniform vec3 Ia\_Material;

uniform vec3 Il\_Material;

uniform vec3 LightPos;

uniform vec2 Parameters;

uniform vec3 CameraPos;

uniform vec3 FragColor;

void main(void){

vec3 L = vec3(LightPos.x - FragVertex.x, LightPos.y - FragVertex.y, LightPos.z - FragVertex.z);

float dist = length(L);

L = normalize(L);

vec3 FragNormaleW = normalize(FragNormale);

float I\_red = Ia\_Material.x \* Ka\_Material.x;

float I\_green = Ia\_Material.y \* Ka\_Material.y;

float I\_blue = Ia\_Material.z \* Ka\_Material.z;

I\_red += clamp(0, 1, Il\_Material.x \* Kd\_Material.x \* dot(L, FragNormaleW) / (Parameters[0] \* dist + Parameters[1]));

I\_green += clamp(0, 1, Il\_Material.y \* Kd\_Material.y \* dot(L, FragNormaleW) / (Parameters[0] \* dist + Parameters[1]));

I\_blue += clamp(0, 1, Il\_Material.z \* Kd\_Material.z \* dot(L, FragNormaleW) / (Parameters[0] \* dist + Parameters[1]));

if (dot(L, FragNormaleW) > 0){

vec3 S = vec3(CameraPos.x - FragVertex.x, CameraPos.y - FragVertex.y, CameraPos.z - FragVertex.z);

vec3 R = vec3(reflect(-L, FragNormale));

S = normalize(S);

R = normalize(R);

if (dot(R, S) > 0){

I\_red += clamp(0, 1, Il\_Material.x \* Ks\_Material.x \* pow(dot(R, S), P\_Material) / (Parameters[0] \* dist + Parameters[1]));

I\_green += clamp(0, 1, Il\_Material.y \* Ks\_Material.y \* pow(dot(R, S), P\_Material) / (Parameters[0] \* dist + Parameters[1]));

I\_blue += clamp(0, 1, Il\_Material.z \* Ks\_Material.z \* pow(dot(R, S), P\_Material) / (Parameters[0] \* dist + Parameters[1]));

}

}

I\_red = min(1, I\_red);

I\_green = min(1, I\_green);

I\_blue = min(1, I\_blue);

vec4 result = vec4(FragColor.x \* I\_red, FragColor.y \* I\_green, FragColor.z \* I\_blue, 1);

gl\_FragColor = result;

}

**Вывод**

В результате выполнения данной лабораторной работы я познакомился с возможностями OpenGL поотрисовки 3D фигур, в частности, использованием буфера вершин, шейдеров, подгружаемых глобальных преобразующих матриц и многого другого. Полученные знания я применил для отрисовки прямого эллиптического цилиндра с затенением по Фонгу.