Московский авиационный институт

(Национальный исследовательский университет)

Институт: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»
Дисциплина: «Компьютерная графика»

Лабораторная работа № 4-5

Тема: Ознакомление с технологией OpenGL

Студент: Мукин Юрий
Группа: 80-304
Преподаватель: Чернышов Л.Н.
Дата:
Оценка:

Москва, 2020

1. Постановка задачи

   Создать графическое приложение с использованием OpenGL. Используя

результаты Л.Р.No3, изобразить заданное тело (то же, что и в л.р. No3) 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.

2. Описание программы

Задание выполнено на языка **C++** с использованием библиотек **openGL**, **glfw, glew** и **glm.**

openGL - мультиплатформенная библиотека для работы с графикой, предоставляющая обширный инструментарий, который, однако, применен не был.

glfw - фреймворк для работы с openGL в среде языка C++. Позволяет отрисовывать окна и работать с ивентами.

glew - библиотека облегчающая использование методов OpenGL в среде языка C++.

glm - библиотека облегчающая работу с матрицами и векторами. Также типы данных glm полностью совместимы с типами GLSL, что позволяет сильно облегчить подготовку данных для шейдера.

В шейдере была применена модель освещения Фонга.

3. Набор тестов

тест №1

```
enter  top cut radius(it could be a number in between 0 and 1)
0.3
enter  bottom cut radius(it could be a number in between 0 and 1)
1
enter the distance from the center of the circle to the top cut(taking into account the sign)
0.95
enter the distance from the center of the circle to the bottom cut(taking into account the sign)
0
enter accuracy(a number from 6 to 20)
16
```

рис.1-2

тест №2

```
enter  top cut radius(it could be a number in between 0 and 1)
0.9
enter  bottom cut radius(it could be a number in between 0 and 1)
0.7
enter the distance from the center of the circle to the top cut(taking into account the sign)
0.2
enter the distance from the center of the circle to the bottom cut(taking into account the sign)
-0.8
enter accuracy(a number from 6 to 20)
6
```

рис.3-4

тест №3

```
enter  top cut radius(it could be a number in between 0 and 1)
0.8
enter  bottom cut radius(it could be a number in between 0 and 1)
0.8
enter the distance from the center of the circle to the top cut(taking into account the sign)
0.55
enter the distance from the center of the circle to the bottom cut(taking into account the sign)
-0.55
```

```
 enter accuracy(a number from 6 to 20)
20
```
рис.5-6

4. Результаты выполнения тестов

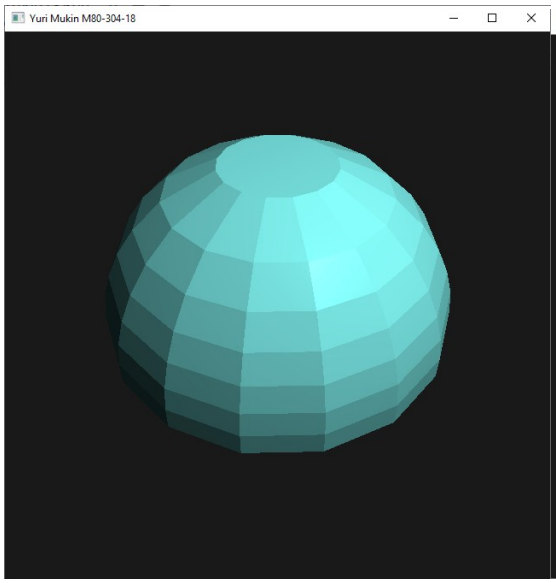

рис.1

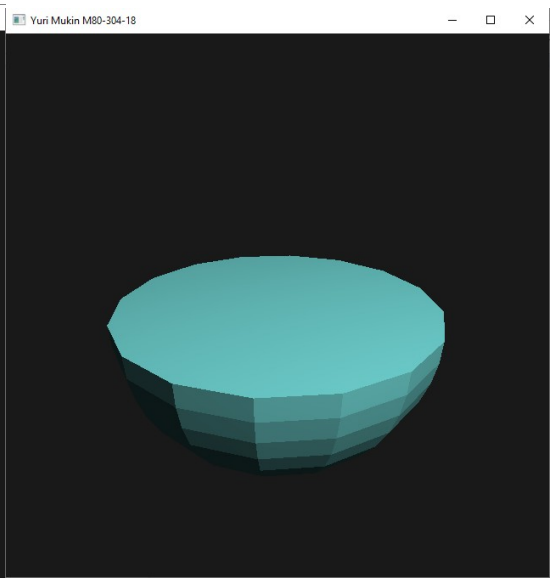

рис.2

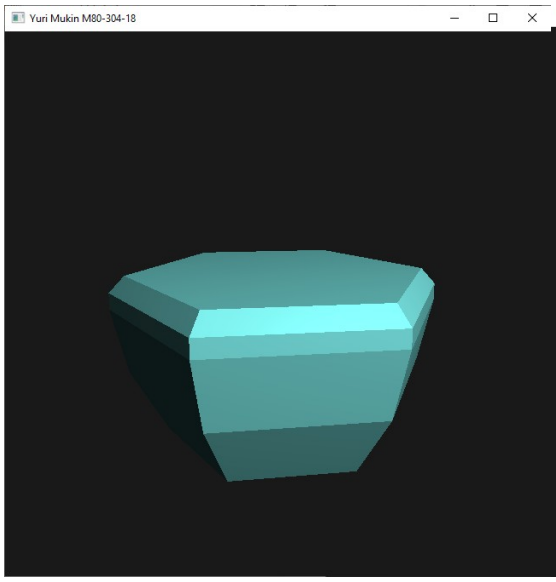

рис.3

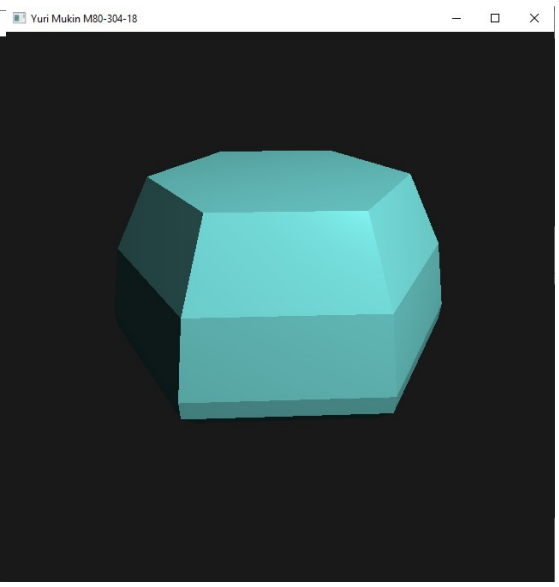

рис.4

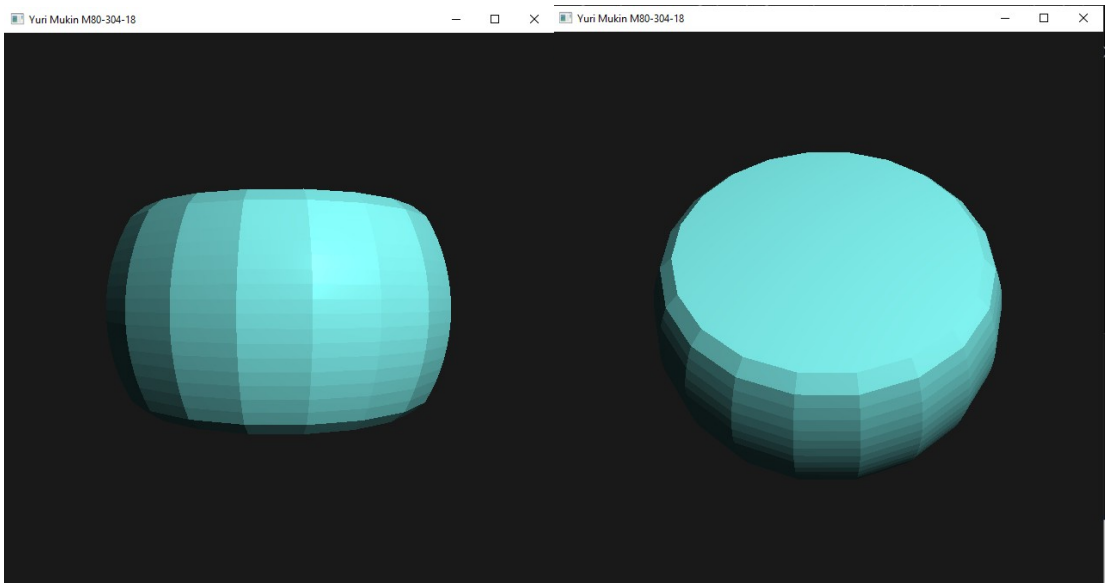

рис.5 рис.6

5. Листинг программы

обертка шейдера для удобного вызова и создания

```
Shader::Shader(const char* VertPath, const char* FigPath)
{
        GLenum err = glewInit();//инициализируем glew
        std::string VertexCode;
        std::string FigureCode;
        std::ifstream VertexCodeF;
        std::ifstream FigureCodeF;
        VertexCodeF.exceptions(std::ifstream::badbit); //считываем код шейдеров
        FigureCodeF.exceptions(std::ifstream::badbit);
        try
        {
                VertexCodeF.open(VertPath);
                FigureCodeF.open(FigPath);
                std::stringstream VertexCodeStream, FigureCodeStream;
                VertexCodeStream << VertexCodeF.rdbuf();
                FigureCodeStream << FigureCodeF.rdbuf();
                VertexCodeF.close();
                FigureCodeF.close();
                VertexCode = VertexCodeStream.str();
                FigureCode = FigureCodeStream.str();
        }
    catch (std::ifstream::failure e)//проверяем успешность считывания
    {
        std::cout << "ERROR::SHADER::FILE_NOT_SUCCESFULLY_READ" << std::endl;
    }
    const GLchar* vShaderCode = VertexCode.c_str();
    const GLchar* fShaderCode = FigureCode.c_str();
    GLuint vertex, fragment;
    GLint success;
    GLchar infoLog[512];
    vertex = glCreateShader(GL_VERTEX_SHADER);
    glShaderSource(vertex, 1, &vShaderCode, NULL);
    glCompileShader(vertex);//компилируем шейдер и проверяем все ли прошло хорошо
    glGetShaderiv(vertex, GL_COMPILE_STATUS, &success);
    if (!success)
    {
        glGetShaderInfoLog(vertex, 512, NULL, infoLog);
        std::cout << "ERROR::SHADER::VERTEX::COMPILATION_FAILED\n" << infoLog << std::endl;
    }
    fragment = glCreateShader(GL_FRAGMENT_SHADER);
    glShaderSource(fragment, 1, &fShaderCode, NULL);
    glCompileShader(fragment);
    glGetShaderiv(fragment, GL_COMPILE_STATUS, &success);
    if (!success)
    {
```

```
        glGetShaderInfoLog(fragment, 512, NULL, infoLog);
        std::cout << "ERROR::SHADER::FRAGMENT::COMPILATION_FAILED\n" << infoLog << std::endl;
    }
    this->Program = glCreateProgram();
    glAttachShader(this->Program, vertex);//записываем шейдеры в в переменную
    glAttachShader(this->Program, fragment);
    glLinkProgram(this->Program);
    glGetProgramiv(this->Program, GL_LINK_STATUS, &success);
    if (!success)
    {
        glGetProgramInfoLog(this->Program, 512, NULL, infoLog);
        std::cout << "ERROR::SHADER::PROGRAM::LINKING_FAILED\n" << infoLog << std::endl;
    }
    glDeleteShader(vertex);
    glDeleteShader(fragment);

}

void Shader::Use()
{
        glUseProgram(Shader::Program); // используем полученый шейдер
}
```

Создание и вызов шейдера осуществляется следующим образом:

```
Shader DifractionShader("path/to/vertexshader.vs", path/to/figureshader.frag");
DifractionShader.Use();
```

Код шейдеров

вершинный

```
#version 330 core
layout (location = 0) in vec3 position;
layout (location = 1) 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(model * vec4(position, 1.0f));
    Normal = mat3(transpose(inverse(model))) * normal;
}
```

фрагментный

```
#version 330 core
out vec4 color;

in vec3 FragPos;
in vec3 Normal;


uniform vec3 lightPos;
uniform vec3 viewPos;
uniform vec3 lightColor;
uniform vec3 objectColor;

void main()
{
    float ambientStrength = 0.1f;
    vec3 ambient = ambientStrength * lightColor;

    vec3 norm = normalize(Normal);
    vec3 lightDir = normalize(lightPos - FragPos);
    float diff = max(dot(norm, lightDir), 0.0);
```

```
    vec3 diffuse = diff * lightColor;

    float specularStrength = 0.5f;
    vec3 viewDir = normalize(viewPos - FragPos);
    vec3 reflectDir = reflect(-lightDir, norm);
    float spec = pow(max(dot(viewDir, reflectDir), 0.0), 32);
    vec3 specular = specularStrength * spec * lightColor;

    vec3 result = (ambient + diffuse + specular) * objectColor;
    color = vec4(result, 1.0f);
}
```

Аппроксиматор

```
class Approximator
{
public:
        GLfloat* PointSet;
        int weight;

        Approximator(GLfloat  Rt, GLfloat  Rb, GLfloat  hu, GLfloat  hd)
        {
                Rtop = Rt;
                Rbottom = Rb;
                heghtU = hu;
                heghtD = hd;
                weight = 0;
        }

        void Approximaze(int  pr)
        {
                Approximator::PointSet = new GLfloat[pr * 72 * (pr - 2)];// определяем размер выходного массива
                Approximator::weight = pr * 72 * (pr - 2);
                Dot* t = Approximator::GenCircleR(pr); // генирируем набор окружностей между 2 заданными
                int offset = 0;
                for (int j = 0; j < pr - 2; j++) // генирируем итоговую фигуру по слоям
                {
                        Dot* CrTop = Approximator::GenCircle(t[j].X, t[j].Y, pr); // выбираем 2 соседние окружности
                        Dot* CrDown = Approximator::GenCircle(t[j + 1].X, t[j + 1].Y, pr);
                        for (int i = 1; i <= pr; i++)
                        {
                                if (j == 0)
                                {
                                        GLfloat   tmp0[] = { CrTop[i - 1].X, CrTop[i - 1].Y, CrTop[i - 1].Z,
CrTop[i].X, CrTop[i].Y, CrTop[i].Z, 0, CrTop[i].Y, 0 };
                                        Dot  norm0  =  Approximator::GetNorm(tmp0);  //  изем  нормаль  поверности
(необходимо для расчета освещения)
                                        norm0.Normolize();
                                        PointSet[0 + offset] = CrTop[i - 1].X; PointSet[1 + offset] = CrTop[i - 1].Y;
PointSet[2 + offset] = CrTop[i - 1].Z;
                                        PointSet[3 + offset] = norm0.X; PointSet[4 + offset] = norm0.Y; PointSet[5 +
offset] = norm0.Z;
                                        PointSet[6  +  offset]  =  CrTop[i].X;  PointSet[7  +  offset]  =  CrTop[i].Y;
PointSet[8 + offset] = CrTop[i].Z;
                                        PointSet[9 + offset] = norm0.X; PointSet[10 + offset] = norm0.Y; PointSet[11 +
offset] = norm0.Z;
                                        PointSet[12 + offset] = 0; PointSet[13 + offset] = CrTop[i].Y; PointSet[14 +
offset] = 0;
                                        PointSet[15 + offset] = norm0.X; PointSet[16 + offset] = norm0.Y; PointSet[17
+ offset] = norm0.Z;
                                        offset += 18;
                                }
                                if (j == pr - 3)
                                {
                                        GLfloat  tmp1[] = { CrDown[i].X, CrDown[i].Y, CrDown[i].Z, CrDown[i - 1].X,
CrDown[i - 1].Y, CrDown[i - 1].Z,0, CrDown[i].Y, 0 };
                                        Dot norm1 = Approximator::GetNorm(tmp1);
                                        norm1.Normolize();
                                        PointSet[0 + offset] = CrDown[i - 1].X; PointSet[1 + offset] = CrDown[i -
1].Y; PointSet[2 + offset] = CrDown[i - 1].Z;
                                        PointSet[3 + offset] = norm1.X; PointSet[4 + offset] = norm1.Y; PointSet[5 +
offset] = norm1.Z;
```

```
                                        PointSet[6 + offset] = CrDown[i].X; PointSet[7 + offset] = CrDown[i].Y;
PointSet[8 + offset] = CrDown[i].Z;
                                        PointSet[9 + offset] = norm1.X; PointSet[10 + offset] = norm1.Y; PointSet[11 +
offset] = norm1.Z;
                                        PointSet[12 + offset] = 0; PointSet[13 + offset] = CrDown[i].Y; PointSet[14 +
offset] = 0;
                                        PointSet[15 + offset] = norm1.X; PointSet[16 + offset] = norm1.Y; PointSet[17
+ offset] = norm1.Z;
                                        offset += 18;
                                }
                                GLfloat    tmp2[] = { CrTop[i].X,CrTop[i].Y,CrTop[i].Z, CrTop[i - 1].X,CrTop[i -
1].Y,CrTop[i - 1].Z, CrDown[i].X, CrDown[i].Y, CrDown[i].Z };
                                Dot norm2 = Approximator::GetNorm(tmp2);
                                norm2.Normolize();
                                PointSet[0 + offset] = CrTop[i - 1].X; PointSet[1 + offset] = CrTop[i - 1].Y;
PointSet[2 + offset] = CrTop[i - 1].Z;
                                PointSet[3 + offset] = norm2.X; PointSet[4 + offset] = norm2.Y; PointSet[5 + offset] =
norm2.Z;
                                PointSet[6 + offset] = CrTop[i].X; PointSet[7 + offset] = CrTop[i].Y; PointSet[8 +
offset] = CrTop[i].Z;
                                PointSet[9 + offset] = norm2.X; PointSet[10 + offset] = norm2.Y; PointSet[11 + offset]
= norm2.Z;
                                PointSet[12 + offset] = CrDown[i].X; PointSet[13 + offset] = CrDown[i].Y; PointSet[14
+ offset] = CrDown[i].Z;
                                PointSet[15 + offset] = norm2.X; PointSet[16 + offset] = norm2.Y; PointSet[17 +
offset] = norm2.Z;
                                offset += 18;
                                GLfloat  tmp3[] = { CrDown[i - 1].X, CrDown[i - 1].Y, CrDown[i - 1].Z, CrDown[i].X,
CrDown[i].Y, CrDown[i].Z, CrTop[i - 1].X, CrTop[i - 1].Y, CrTop[i - 1].Z };
                                Dot norm3 = Approximator::GetNorm(tmp3);
                                norm3.Normolize();
                                PointSet[0 + offset] = CrTop[i - 1].X; PointSet[1 + offset] = CrTop[i - 1].Y;
PointSet[2 + offset] = CrTop[i - 1].Z;
                                PointSet[3 + offset] = norm3.X; PointSet[4 + offset] = norm3.Y; PointSet[5 + offset] =
norm3.Z;
                                PointSet[6 + offset] = CrDown[i - 1].X; PointSet[7 + offset] = CrDown[i - 1].Y;
PointSet[8 + offset] = CrDown[i - 1].Z;
                                PointSet[9 + offset] = norm3.X; PointSet[10 + offset] = norm3.Y; PointSet[11 + offset]
= norm3.Z;
                                PointSet[12 + offset] = CrDown[i].X; PointSet[13 + offset] = CrDown[i].Y; PointSet[14
+ offset] = CrDown[i].Z;
                                PointSet[15 + offset] = norm3.X; PointSet[16 + offset] = norm3.Y; PointSet[17 +
offset] = norm3.Z;
                                offset += 18;
                        }
                }
        }
private:
        GLfloat  Rtop, Rbottom, heghtU, heghtD;

        Dot* GenCircle(GLfloat  R, GLfloat  y, int nVert)// аппроксимируем окружность до описанного многогранника
        {
                float alpha = 6.28318530718 / nVert;
                Dot* vert = new Dot[nVert + 1];
                double x, z;
                for (int i = 0; i <= nVert; i++)
                {
                        x = R * sin(alpha * i);
                        z = R * cos(alpha * i);
                        if (abs(x) < 0.001)
                                x = 0;
                        if (abs(z) < 0.001)
                                z = 0;
                        vert[i].SetCord(x, y, z);
                }
                return vert;
        }

        Dot* GenCircleR(int  pr)// генерируем набор окружностей между 2 заданными
        {
                int k = pr / 2 - 1;
```

```
                Dot* res = new Dot[pr-1];
                float tmp = heghtU / k;
                for (int i = 0; i < k ; i++)
                        res[k-i].SetCord(sin(acos(tmp*i)), tmp * i, 0);
                tmp = heghtD / k;
                for (int i = 1; i < k; i++)
                        res[k + i].SetCord(sin(acos(tmp * i)), tmp * i, 0);
                res[0].SetCord(Rtop, heghtU, 0);
                res[pr-2].SetCord(Rbottom, heghtD, 0);
                for (int i = 0; i < pr; i++)
                return res;
        }

        Dot GetNorm(GLfloat  points[])// вычисляем нормаль по 3-м точкам
        {
                GLfloat x1, x2, y1, y2, z1, z2;
                x1 = points[3] - points[0]; x2 = points[6] - points[0];
                y1 = points[4] - points[1]; y2 = points[7] - points[1];
                z1 = points[5] - points[2]; z2 = points[8] - points[2];
                GLfloat x, y, z;
                x = y1 * z2 - z1 * y2;
                y = -x1 * z2 + x2 * z1;
                z = x1 * y2 - y1 * x2;
                return Dot(x, y, z);
        }
};
```

Список литературы

1. Документация **glfw**. URL: https://www.glfw.org/documentation.html.
2. Справочник по языку **C++**. URL: https://docs.microsoft.com/ru-ru/cpp/cpp/cpp-language-reference?view=vs-2019
3. Руководство **openGL**. URL: https://www.opengl.org.ru/docs/pg/index.html