Московский авиационный институт
(Национальный исследовательский университет)

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

Лабораторная работа № 4

Тема: Ознакомление с технологией OpenGL

Студент: Пивницкий Даниэль Сергеевич
Группа: 80-306
Преподаватель: Чернышов Л.Н.
Дата:
Оценка:

Москва, 2021

1. **Постановка задачи**

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. **Описание программы**

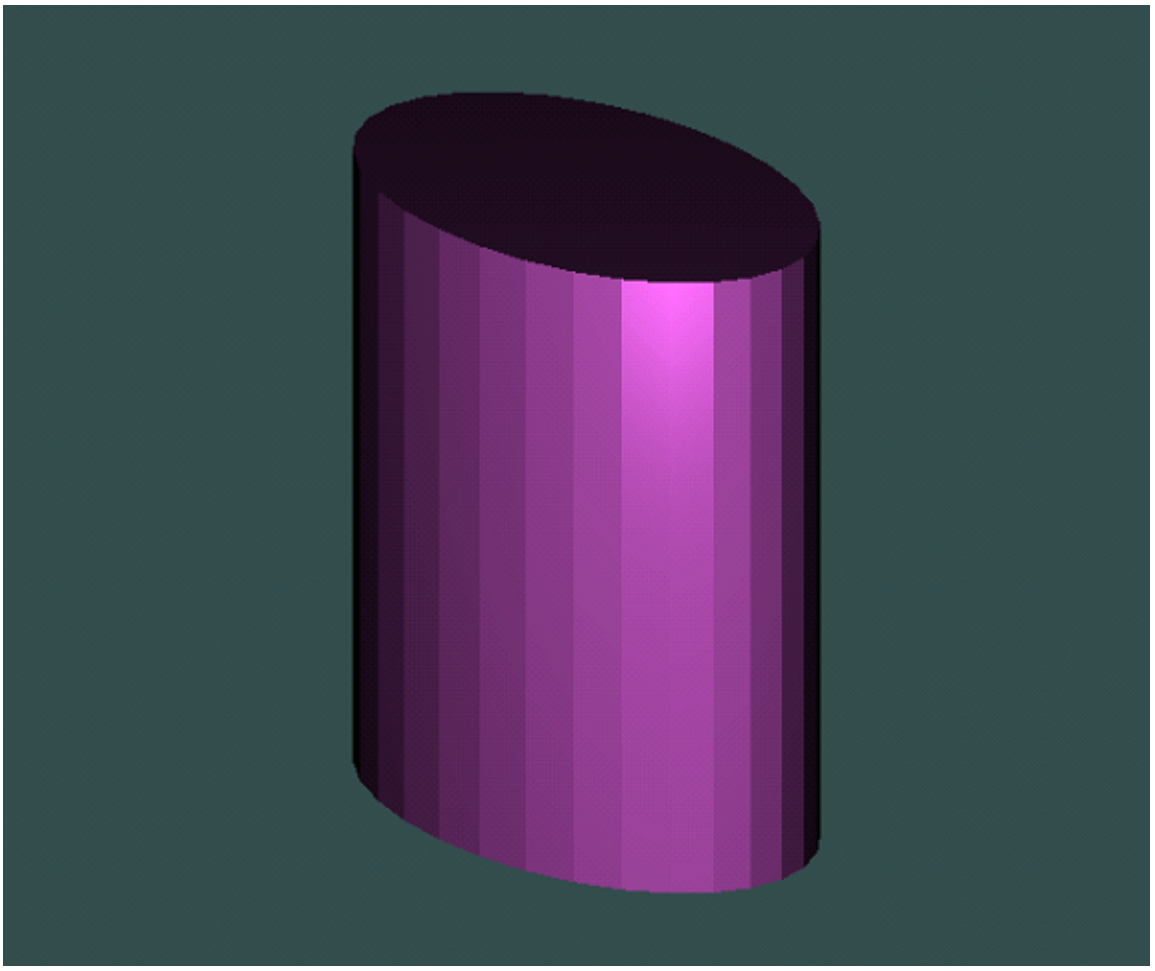
При вызове программы пользователь задает количество боковых граней цилиндра. На основе полученных данных о гранях, запускается генератор фигуры void GenCylinder(), после чего происходит триангуляция модели. Создается шейдерная программа и заполняется вершинный буфер. Подключаем буфер глубины для правильной отрисовки модели. Передаем в шейдерную программу данные о свете, объекте и зрителе. Внутри вершинного шейдера считаются конечные позиции точек модели и нормали. Во фрагментном шейдере рассчитывается освещенность полигона с учетом диффузного освещения и зеркального блика.

3. **Набор тестов**

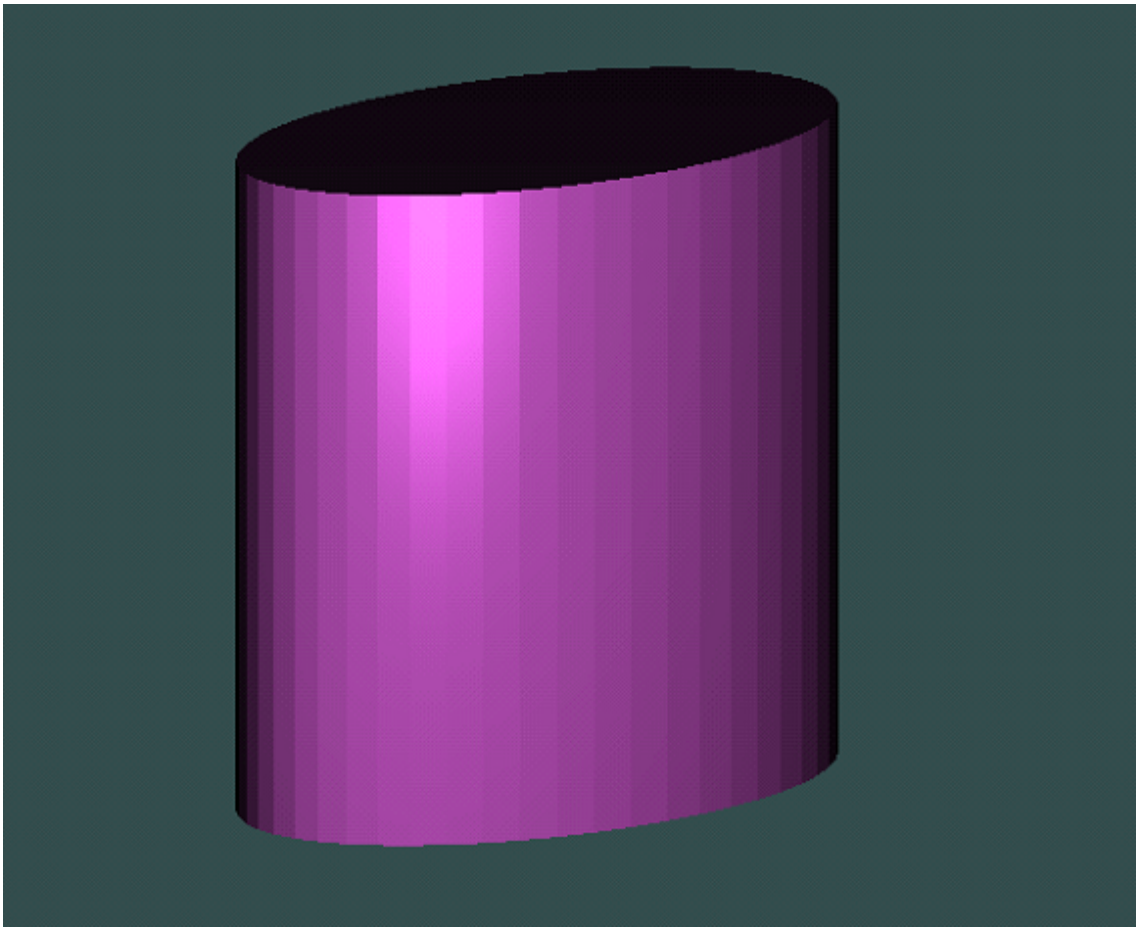
```
test 1 - ./result 30

test 2 - ./result 50

test 3 - ./result 9
```

4. Результаты выполнения тестов

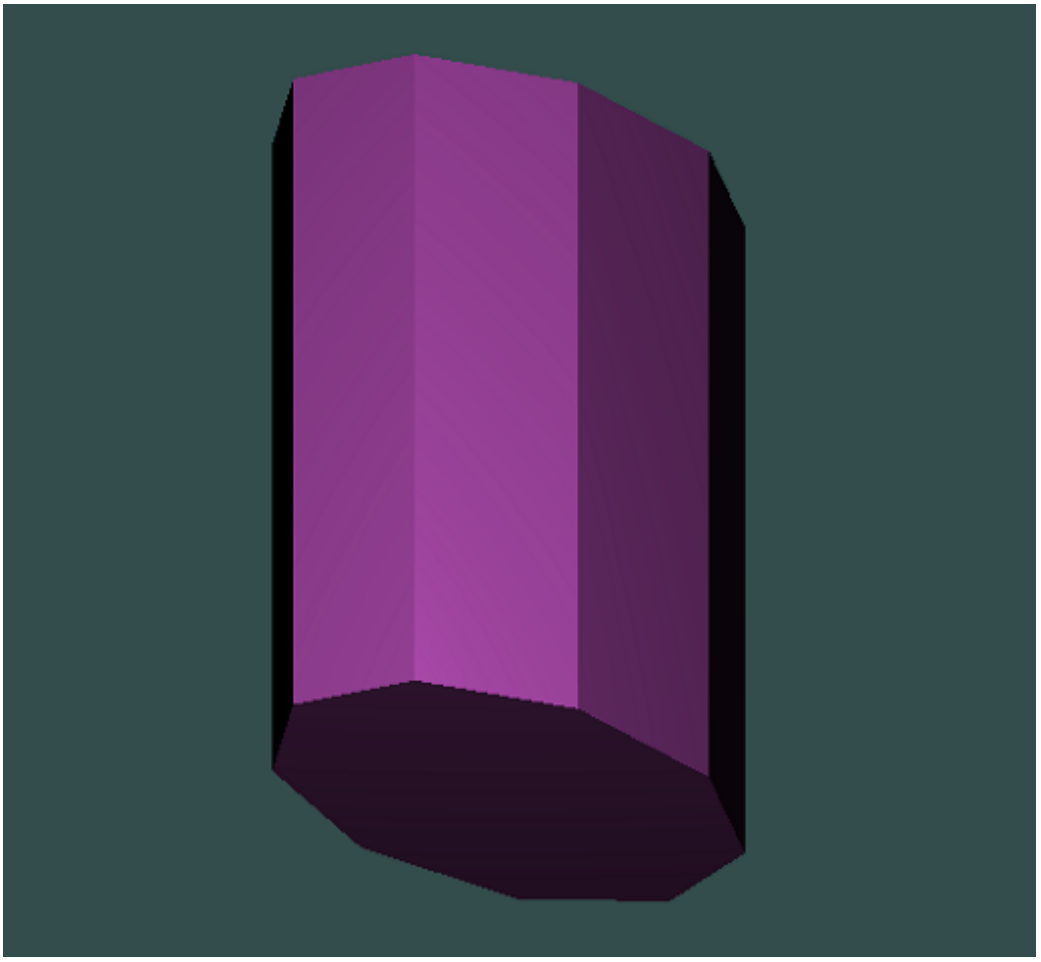
test 1



test 2



test 3



5. Листинг программы

```
#include <iostream>
#include <cmath>
#include <map>
#include <fstream>
#define GLEW_STATIC
#include <GL/glew.h>
#include <GLFW/glfw3.h>
#include "../glm/glm/glm.hpp"
#include "../glm/glm/gtc/matrix_transform.hpp"
#include "../glm/glm/gtc/type_ptr.hpp"

#include "shaders.h"
#include "object.h"
#include "matrix.h"

void GenCylinder(int countOfPoints = 18, float
scircle = 3, float bcircle = 5, float stepY = 0.1){
          std::ofstream file("figures/cylinder.obj",
std::ofstream::out | std::ofstream::trunc);
    if(!file.is_open()){
        std::cout << "Can't open file\n";
    }
    else{
        float stepDgr = (float)360/countOfPoints;
        float x, y, z;
        int count = 0;
        for(y=-5.0; y<5.0; y=y+stepY){
            count++;
                    for(float dgr = 0; dgr < 360;
dgr=dgr+stepDgr){
                x = cos(M_PI/180 * dgr)*bcircle;
                        z = sqrt(1 - pow(x/bcircle,
2.0))*scircle;
                if(sin(M_PI/180 * dgr) < 0){
```

```
                    z *= -1;
                }
                 file << "v " << x << " " << y << " "
<< z << "\n";
            }
        }
        for(int i = 0; i < count-1; i++){
            for(int j = 0; j < countOfPoints; j++){
                 file << "f " << j + i*countOfPoints
<<  "  "  <<    j+(i+1)*countOfPoints  <<  "  "  <<
(j+1)%countOfPoints + (i+1)*countOfPoints << " " <<
(j+1)%countOfPoints + i*countOfPoints << "\n";
            }
        }
        file << "f";
        for(int i = 0; i < countOfPoints; i++){
            file << " " << i;
        }
        file << "\nf";
             for(int i = count*countOfPoints; i >
(count-1)*countOfPoints; i--){
            file << " " << i-1;
        }
    }
    file.close();
}


int main(int argc, char *argv[]){
    if(argc < 2){
        perror("Usage: ./main <points in ring>");
        exit(-1);
    }
    int countPoints = atoi(argv[1]);
    if(countPoints < 3){
```

```
        perror("Small number of points");
        exit(-1);
    }
    GenCylinder(countPoints);

    glfwInit();
    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
                glfwWindowHint(GLFW_OPENGL_PROFILE,
GLFW_OPENGL_CORE_PROFILE);
     GLFWwindow* window = glfwCreateWindow(700, 700,
"Start", NULL, NULL);
    if(window == NULL){
        fprintf(stderr,  "Невозможно открыть окно
GLFW. Если у вас Intel GPU, то он не поддерживает
версию 3.3. Попробуйте версию уроков для OpenGL
2.1.n");
        glfwTerminate();
        return -1;
    }
    glfwMakeContextCurrent(window);
    glewExperimental=true;
    if(glewInit()!=GLEW_OK){
        fprintf(stderr, "Невозможно инициализировать
GLEWn");
        return -1;
    }
    int width, height;
    float intensity = 1.0;
    glfwGetFramebufferSize(window, &width, &height);
    glViewport(0, 0, width, height);
    Object object("figures/cylinder.obj");

    Shader shader;
    GLuint VBO, VAO;
```

```
    glGenVertexArrays(1, &VAO);
    glGenBuffers(1, &VBO);
    glBindVertexArray(VAO);
    glBindBuffer(GL_ARRAY_BUFFER, VBO);
                        glBufferData(GL_ARRAY_BUFFER,
object.getVAndN().size()*sizeof(float),
&object.getVAndN().front(), GL_STATIC_DRAW);
     glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE,
6*sizeof(GLfloat), (GLvoid*)0);
    glEnableVertexAttribArray(0);
     glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE,
6*sizeof(GLfloat), (GLvoid*)(3*sizeof(GLfloat)));

    glEnableVertexAttribArray(1);

    glBindBuffer(GL_ARRAY_BUFFER, 0);
    glBindVertexArray(0);
    glEnable(GL_DEPTH_TEST);
    while(!glfwWindowShouldClose(window)){
        glfwPollEvents();
        glClearColor(0.2f, 0.3f, 0.3f, 1.0f);
                    glClear(GL_COLOR_BUFFER_BIT   |
GL_DEPTH_BUFFER_BIT);
         shader.Use();
         GLfloat timeValue = glfwGetTime();
        object.SetrotX(timeValue*20);
        object.SetrotY(timeValue*15);
        object.Setdz(-15);

glUniform3f(glGetUniformLocation(shader.shaderProgra
m, "objColor"), 0.7f, 0.3f, 0.7f);

glUniform3f(glGetUniformLocation(shader.shaderProgra
m, "lightPos"), 0.0f, 0.0f, 15.0f);
```

```
glUniform3f(glGetUniformLocation(shader.shaderProgra
m, "lightColor"), 1.0f, 1.0f, 1.0f);

glUniform3f(glGetUniformLocation(shader.shaderProgra
m, "viewPos"), 0.0f, 0.0f, 0.0f);

glUniform1f(glGetUniformLocation(shader.shaderProgra
m, "reflectionIntensity"), 0.7);
                                GLuint    matLoc    =
glGetUniformLocation(shader.shaderProgram,
"transform");
            glUniformMatrix4fv(matLoc,  1,  GL_FALSE,
glm::value_ptr(mtx::ModelMatrix(object)));
                                          matLoc     =
glGetUniformLocation(shader.shaderProgram,
"projection");
            glUniformMatrix4fv(matLoc,  1,  GL_FALSE,
glm::value_ptr(mtx::ProjectMatrix()));

        glBindVertexArray(VAO);
                      glDrawArrays(GL_TRIANGLES,   0,
object.getAllVerteces().size()/3);

        glBindVertexArray(0);
        glfwSwapBuffers(window);
    }

    glDeleteVertexArrays(1, &VAO);
    glDeleteBuffers(1, &VBO);
    glfwTerminate();
    return 0;
}
```

6. Литература

1.Документация GLEW:
http://glew.sourceforge.net/
2.Документация GLFW:
https://www.glfw.org/docs/3.3/compile.html