Московский Авиационный Институт

(Национальный исследовательский Университет)

Факультет: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

**Лабораторная работа №3**

**по курсу «Компьютерная графика»**

|  |  |
| --- | --- |
| Студент: | Аксенов А.Е. |
| Группа: | М80-308Б-18 |
| Преподаватель: | Филиппов Г.С. |
| Оценка: |  |
| Дата: |  |

Москва

2021

# Основы построения фотореалистичных изображений.

**Задача:** 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

**Вариант №16:**

Усеченный прямой эллиптический цилиндр.

1. **Описание**

Для реализации я использовал библиотеки GLFW для работы с оконными приложениями, GLM для математических операций, GLEW для упрощения работы с функциями библиотек и кроссплатформенности, а также язык C++ и OpenGL/.

Библиотека позволяет отрисовывать следующие геометрические единицы: точку, прямую, треугольник. Поэтому для реализации отрисуем цилиндр как n-ти угольник в основании, составленный из n треугольников с общей вершиной в центре масс и еще 2 \* n треугольников (боковых граней) для соединения 2-ух оснований. Значение n зависит от точности аппроксимации, задаваемой с консоли.

Координаты вершин треугольников из которых составим фигуру вычислим и подадим библиотеке в нормализованных координатах в диапазоне [−1, 1].

Вращение реализуем при помощи кватернионов для более гладкого вращения и решения проблемы шарнирного замка. Для расчета новых координат при вращении будем преобразовывать кватернионы в эквивалентные матрицы вращения и делать матричное произведение для получения новых кординат.

Для каждой точки фигуры в программе вычисляется нормаль поверхности в этой точке. Так же в программе задано положение (в координатах) источника света. С помощью этих координат и нормали в точке реализуется основное освещение в конкретной точке, чилу которого можно регулировать с помощью значения вводимого с клавиатуры, также как и значения силы фонового и бликового освещения реализованного в программе.

1. **Исходный код**

Вначале подключаются необходимые библиотеки. В файле «multyplyes.h» реализованы кватернионы вращения и все необходимые функции для работы с ними. В файле «Camera.h» реализованы функции для работы и оптимального расположения камеры в отрисовываемом пространстве. Остальные включения - подключения библиотек: GLFW, GLM, GLEW.

1. #include <iostream>
2. #include <cmath>
3. *// GLEW*
4. #define GLEW\_STATIC
5. #include <GL/glew.h>
6. *// GLFW*
7. #include <GLFW/glfw3.h>
8. *// GLM Mathematics*
9. #include <glm/glm.hpp>
10. #include <glm/gtc/matrix\_transform.hpp>
11. #include <glm/gtc/type\_ptr.hpp>
12. *// Other includes*
13. #include "Shader.h"
14. #include "Camera.h"
15. #include "multyplyes.h"
16. using namespace std;

Затем задаются глобальные переменные через которые реализуется мгновенное вращение, масштаб фигуры и сигнал к возврату в исходное положение. Помимо этого здесь инициализируется камера, положение источника света и создается массив для регистрации нажатий клавиатуры. Мелкость разбиения для более точного отображения фигуры хранится в переменной APPROX.

1. GLfloat\* get\_figure();
2. *// Window dimensions*
3. const GLuint WIDTH = 800, HEIGHT = 600;
4. *// Camera*
5. Camera camera(glm::vec3(0.0f, 0.0f, 3.0f));
6. GLfloat lastX = WIDTH / 2.0;
7. GLfloat lastY = HEIGHT / 2.0;
8. bool keys[1024];
9. bool ret = true;
10. GLfloat APPROX = 0.01f;
11. GLfloat chill\_height = 1.5f;
12. *// Light attributes*
13. glm::vec3 lightPos(1.2f, 1.0f, 2.0f);
14. *// Deltatime*
15. GLfloat deltaTime = 0.0f; *// Time between current frame and last frame*
16. GLfloat lastFrame = 0.0f; *// Time of last frame*
17. GLfloat x\_rotation = 0.0f, y\_rotation = 0.0f, z\_rotation = 0.0f;
18. GLfloat scale = 1.0f;

Функции реализующие в себе обработку пользовательских действий, таких как об- работка изменений экрана, нажатия клавиш и движения мыши, имеют следующие объявления:

1. void key\_callback(GLFWwindow\* window, int key, int scancode, int action, int mode);
2. void mouse\_callback(GLFWwindow\* window, double xpos, double ypos);
3. void scroll\_callback(GLFWwindow\* window, double xoffset, double yoffset);
4. void do\_movement();
5. void new\_func\_size\_callback(GLFWwindow\* window, int width, int heigh);

Для генерации вектора с вершинами для отрисовки цилиндра используется следующая функция, способная сгенерировать цилиндр в зависимости от параметра апроксимации.

1 GLfloat\* get\_figure();

В функции main происходит инициализация библиотек, последовательный вызов всех этих функций и проводятся основные математические операции.

Для отображения окна использую функции, встроенные в OpenGl.

1. glfwWindowHint(GLFW\_CONTEXT\_VERSION\_MAJOR, 3);
2. glfwWindowHint(GLFW\_CONTEXT\_VERSION\_MINOR, 3);
3. glfwWindowHint(GLFW\_OPENGL\_PROFILE, GLFW\_OPENGL\_CORE\_PROFILE);
4. glfwWindowHint(GLFW\_RESIZABLE, GL\_TRUE);

Затем нужно инициализировать GLEW и остальные функции для отображения фигуры, света и подгрузить шейдеры.

1. glewInit();
2. *// Define the viewport dimensions*
3. glViewport(0, 0, WIDTH, HEIGHT);
4. *// OpenGL options*
5. glEnable(GL\_DEPTH\_TEST);
6. *// Build and compile our shader programm*
7. Shader lightingShader("shaders/lighting.vs", "shaders/lighting.frag");
8. Shader lampShader("shaders/lamp.vs", "shaders/lamp.frag");
9. GLfloat\* vertices = get\_figure();
10. *// First, set the container’s VAO (and VBO)*
11. GLuint VBO, containerVAO;
12. glGenVertexArrays(1, &containerVAO);
13. glGenBuffers(1, &VBO);
14. glBindBuffer(GL\_ARRAY\_BUFFER, VBO);
15. glBufferData(GL\_ARRAY\_BUFFER, sizeof(GLfloat) \* (1 + (unsigned)(2/APPROX)) \* 9 \* 4

\* 2, vertices, GL\_STATIC\_DRAW);

1. glBindVertexArray(containerVAO);
2. *// Position attribute*
3. glVertexAttribPointer(0, 3, GL\_FLOAT, GL\_FALSE, 6 \* sizeof(GLfloat), (GLvoid\*)0);
4. glEnableVertexAttribArray(0);
5. *// Normal attribute*
6. glVertexAttribPointer(1, 3, GL\_FLOAT, GL\_FALSE, 6 \* sizeof(GLfloat), (GLvoid\*)(3 \* sizeof(GLfloat)));
7. glEnableVertexAttribArray(1);
8. glBindVertexArray(0);

Остается только описать функцию обработки клавиш.

1 void key\_callback(GLFWwindow\* window, int key, int scancode, int action, int mode)

2 {

|  |  |  |
| --- | --- | --- |
| 3 | if | (key == GLFW\_KEY\_ESCAPE && action == GLFW\_PRESS) |
| 4 |  | glfwSetWindowShouldClose(window, GL\_TRUE); |
| 5 | if | (key >= 0 && key < 1024) |
| 6 | { |  |
| 7 |  | if (action == GLFW\_PRESS) |
| 8 |  | keys[key] = true; |
| 9 |  | else if (action == GLFW\_RELEASE) |
| 10 |  | keys[key] = false; |
| 11 | } |  |
| 12 } |  |  |

1. **Консоль**

В консоли необходимо скомпилировать исходный код и запустить. Согласно заданию в окне необходимо будет ввести параметры освещения и точность апроксимации.

fallfire13@DESKTOP-M7F3IHA:~/CG3\_lab$ g++ main.cpp -o test -lGL -lGLEW -lglfw

fallfire13@DESKTOP-M7F3IHA:~/CG3\_lab$ ./test

Enter params of light:

>> Strenght of ambient light [0.0, 1.0] (default 0.5): 0.5

>> Strenght of diffusion light [0.0, 1.0] (default 0.5): 0.5

>> Strenght of specular light [0.0, 1.0] (default 0.5): 0.5

4>> Enter approximation parametr less then 1.0 (default ~0.002): 0.002

Success

Start

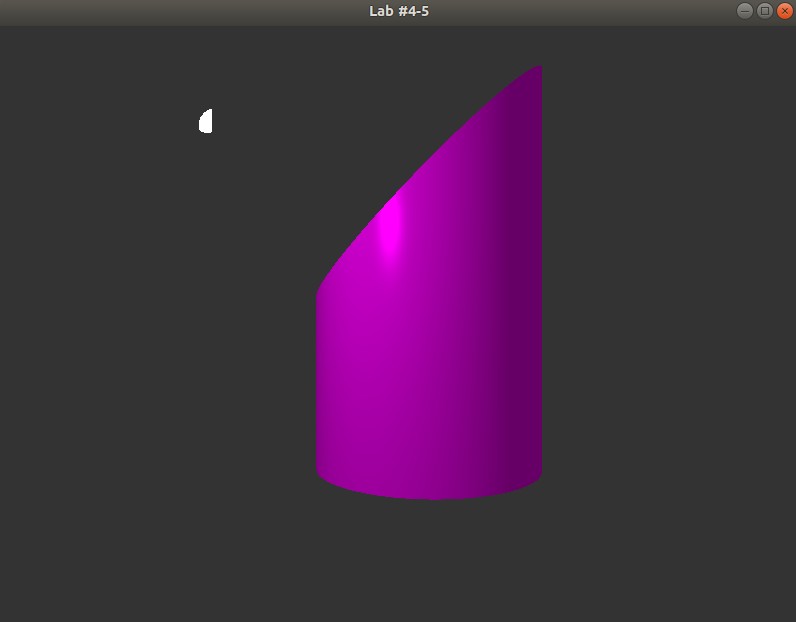
SUCCESSFUL::SHADER::PROGRAM::LINKING\_SUCCESS

SUCCESSFUL::SHADER::PROGRAM::LINKING\_SUCCESS

После откроется изображение фигуры в окне.

Это окно можно изменять по размерам и перемещать по экрану без всяких побочных эффектов, фигура подстраивается под изменение размеров экрана и масштабируется соответствующим образом.

С помощью нажатий клавиатуры можно вращать и масштабировать фигуру произвольным образом:



1. **Выводы**

Выполнив данную лабораторную работу по курсу «Компьютерная графика», я познакомился с OpenGl и ее функциями на языке C++. Как оказалось, данная библиотека очень удобна для отображения фигур. В процессе написания необходимо было прочитать достаточное количество информации о билиотеке и про то, как в ней реализуются и отображаются объемные фигуры.