Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №4-5 по курсу «Компьютерная графика»

Студент: Я. С. Поскряков
Преподаватель: Г. С. Филиппов
Группа: М8О-306Б

Дата:
Оценка:
Подпись:

Москва, 2019

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 №14:
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 кросс-платформенности, а также язык C++ и OpenGL/.

Библиотека позволяет отрисовывать следующие геометрические еденицы: 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 кординатах 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 приобразовывать 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) источнка света. С помощью этих кординат и нормали в точке реализуется основное освещение в конкретной точке, чилу которого можно регулировать с помощю значения вводимого с клавиатуры, также как и значения силы фонового и бликового освещения реализованного в программе.

2 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.

```
#include <iostream>
#include <cmath>
// GLEW
#define GLEW_STATIC
#include <GL/glew.h>
// GLFW
#include <GLFW/glfw3.h>
// GLM Mathematics
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>
// Other includes
#include "Shader.h"
#include "Camera.h"
#include "multyplyes.h"
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.

```
GLfloat* get_figure();
// Window dimensions
const GLuint WIDTH = 800, HEIGHT = 600;
// Camera
Camera camera(glm::vec3(0.0f, 0.0f, 3.0f));
GLfloat lastX = WIDTH / 2.0;
GLfloat lastY = HEIGHT / 2.0;
bool keys[1024];
bool ret = true;
GLfloat APPROX = 0.01f;
GLfloat chill_height = 1.5f;
// Light attributes
glm::vec3 lightPos(1.2f, 1.0f, 2.0f);
// Deltatime
GLfloat deltaTime = 0.0f; // Time between current frame and last frame
```

```
GLfloat lastFrame = 0.0f;  // Time of last frame
GLfloat x_rotation = 0.0f, y_rotation = 0.0f, z_rotation = 0.0f;
GLfloat scale = 1.0f;
```

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

```
void key_callback(GLFWwindow* window, int key, int scancode, int action, int mode);
void mouse_callback(GLFWwindow* window, double xpos, double ypos);
void scroll_callback(GLFWwindow* window, double xoffset, double yoffset);
void do_movement();
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.

```
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.

```
glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
    glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
    glfwWindowHint(GLFW_RESIZABLE, GL_TRUE);
```

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

```
    glewInit();
    // Define the viewport dimensions
    glViewport(0, 0, WIDTH, HEIGHT);
    // OpenGL options
    glEnable(GL_DEPTH_TEST);
    // Build and compile our shader programm
    Shader lightingShader("shaders/lighting.vs", "shaders/lighting.frag");
    Shader lampShader("shaders/lamp.vs", "shaders/lamp.frag");
    GLfloat* vertices = get_figure();
    // First, set the container's VAO (and VBO)
    GLuint VBO, containerVAO;
    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);
16      glBindVertexArray(containerVAO);
17      // Position attribute
18      glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(GLfloat), (GLvoid*)0);
19      glEnableVertexAttribArray(0);
20      // Normal attribute
21      glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(GLfloat), (GLvoid*)(3 *
            sizeof(GLfloat)));
22      glEnableVertexAttribArray(1);
23      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 }
```

3 Консоль

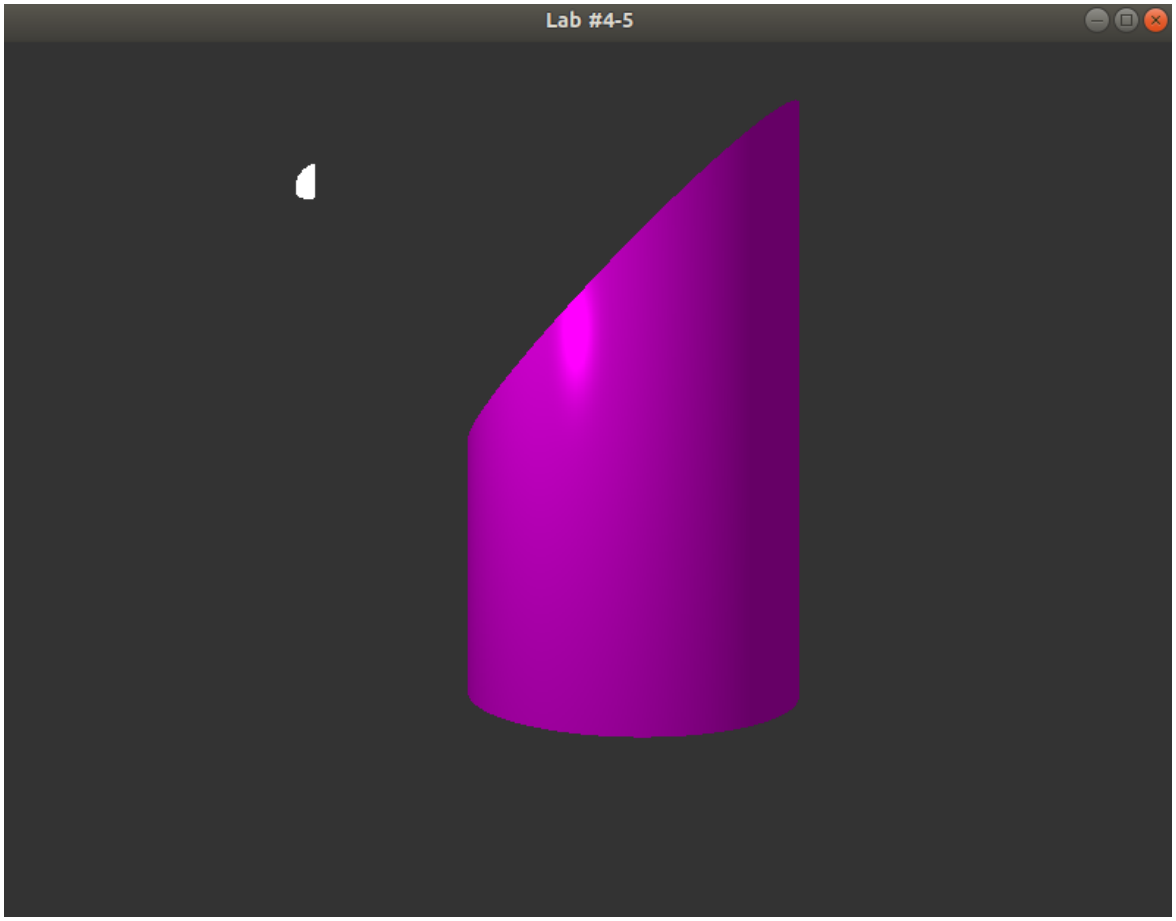
В консоли необходимо скомпилировать исходный код и запустить. Согласно заданию в окне необходимо будет ввести параметры освещения и точность апроксимации.

```
(base) yar@yarmachine:~/CG/already/LAB4_5 g++ main.cpp -o test -lGL -lGLEW
-lglfw
(base) yar@yarmachine:~/CG/already/LAB4_5 ./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
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:



4 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 фигрур.

В процессе написания необходимо было прочитать достаточное количество информации о билиотеке и про то, как в ней реализуются и отображаются объемные фигуры.