Московский авиационный институт

(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №6 по курсу «Компьютерная графика»

Студент: М. А. Бронников

Преподаватель: Г. С. Филиппов

Группа: М8О-307Б

Дата:

Оценка:

Подпись:

Москва, 2019

Создание шейдерных анимационных эффектов в OpenGL 2.1

Задача: для поверхности, созданной в л.р. №5, обеспечить выполнение следующего шейдерного эффекта.

Вариант №12:

Анимация. Координата Y изменяется по закону Y = Y\*cos(t+Y)

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 кросслатформенности, а также язык C++.

OpenGL (Open Graphics Library) — спецификация, определяющая платформонезависимый (независимый от языка программирования) программный интерфейс для написания приложений, использующих двумерную и трёхмерную компьютерную графику. Включает более 300 функций для рисования сложных трёхмерных сцен из простых примитивов.

Шейдер — это просто программа, выполняемая в графическом конвейере. Она сообщает компьютеру, как рендерить каждый пиксель. Эти программы называются шейдерами («затенителями»), потому что их часто используют для управления эффектами освещения и затенения, но ничего не мешает использовать их и для других спецэффектов.

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

2n треугольников(боковых граней) для соединения 2-ух оснований. Значение n зави-

сит 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 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

точек, силу которого можно регулировать с помощью значения вводимого с клавиатуры, так-же как и значения силы фонового и бликового освещения реализованного

в программе.

2. Исходный код

Вначале подключаются необходимые библиотеки. В файле «multyplyes.h» реализо-

ваны кватернионы вращения и все необходимые функции для работы с ними. В

файле «Camera.h» реализованы функции для работы и оптимального расположе-

ния камеры в отрисовываемом пространстве. Остальные включения - подключения

библиотек: GLFW, GLM, GLEW.

1 #include <iostream>

2 #define GLEW\_STATIC

3 #include <GL/glew.h>

4 #include <GLFW/glfw3.h>

5 #include <glm/glm.hpp>

6 #include <glm/gtc/matrix\_transform.hpp>

7 #include <glm/gtc/type\_ptr.hpp>

8 #include "multyplyes.h"

9 #include "Camera.h"

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 const GLuint WIDTH = 800, HEIGHT = 600;

2 Camera camera(glm::vec3(0.0f, 0.0f, 3.0f));

3 GLfloat lastX = WIDTH / 2.0;

4 GLfloat lastY = HEIGHT / 2.0;

5 bool keys[1024];

6 bool ret = true;

7 GLfloat APPROX = 0.01f;

8 GLfloat chill\_height = 0.8f;

9 glm::vec3 lightPos(1.2f, 1.0f, 2.0f);

10 GLfloat deltaTime = 0.0f; // Time between current frame and last frame

11 GLfloat lastFrame = 0.0f; // Time of last frame

12 GLfloat x\_rotation = 0.0f, y\_rotation = 0.0f, z\_rotation = 0.0f;

13 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 следую-

щая функция, способная сгенерировать цилиндр в зависимости от параметра апрок-

симации.

1 GLfloat\* get\_figure();

В функции main происходит инициализация библиотек, последовательный вызов

всех этих функций и проводятся основные математические операции.

3. Консоль

В консоли необходимо скомпилировать исходный код и запустить. Согласно заданию

в окне необходимо будет ввести параметры освещения и точности апроксимации.

(base) max@max-X550CC:~/KG/lab3$ g++ main.cpp -o hello -lGL -lGLEW -lglfw

(base) max@max-X550CC:~/KG/lab3$ ./hello

Enter params of light:

1)Strenght of ambient light from 0.0 to 1.0(recomend 0.1): 0.1

2)Strenght of diffusion light from 0.0 to 1.0(recomend 1.0): 1.0

3)Strenght of specular light from 0.0 to 1.0(recomend 0.5): 0.5

4)Enter approximation parametr less then 1.0: 0.00006

--------------------------------

Starting programm!

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

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

(base) max@max-X550CC:~/KG/lab3$ exit

После откроется изображение фигуры в окне.

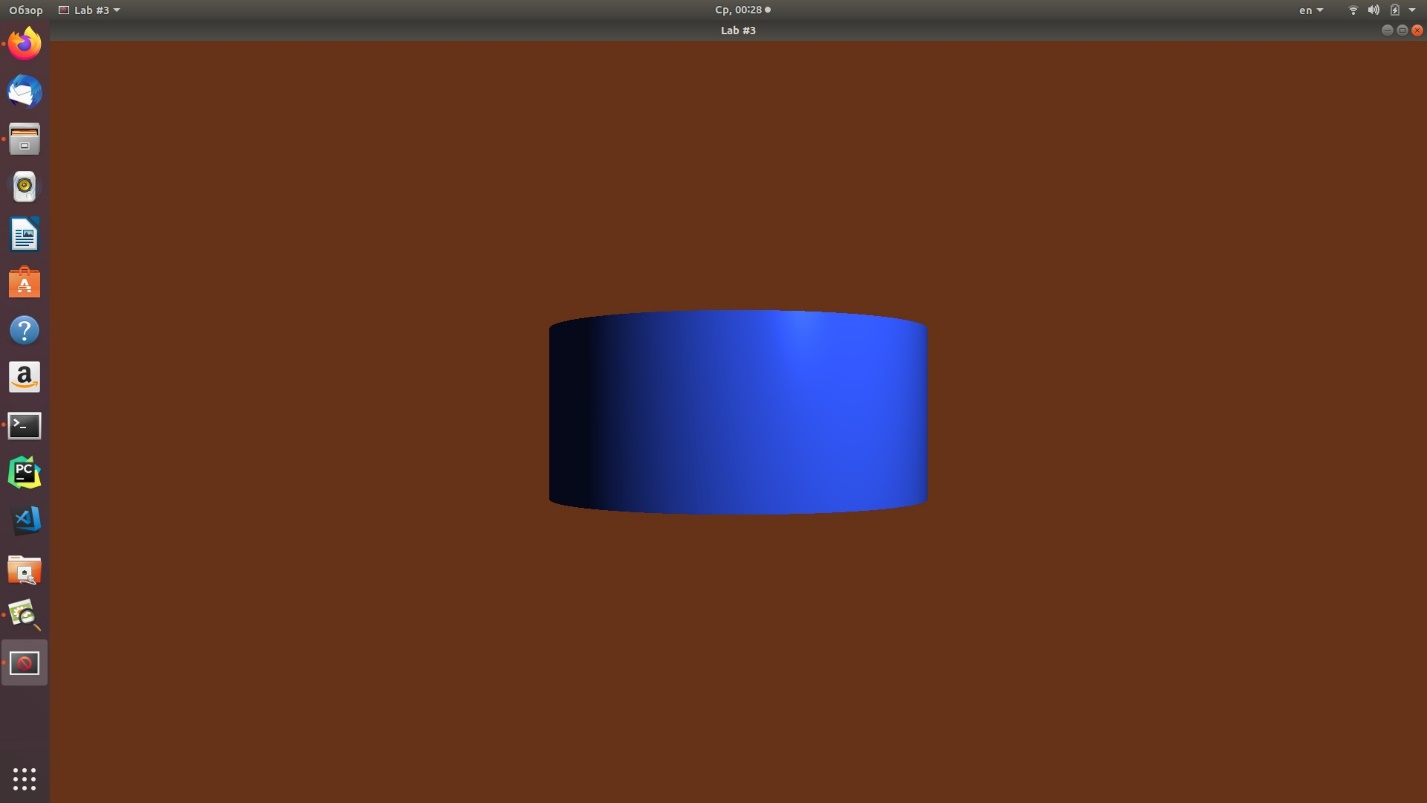
Это окно можно изменять по размерам и перемещать по экрану без всяких побочных

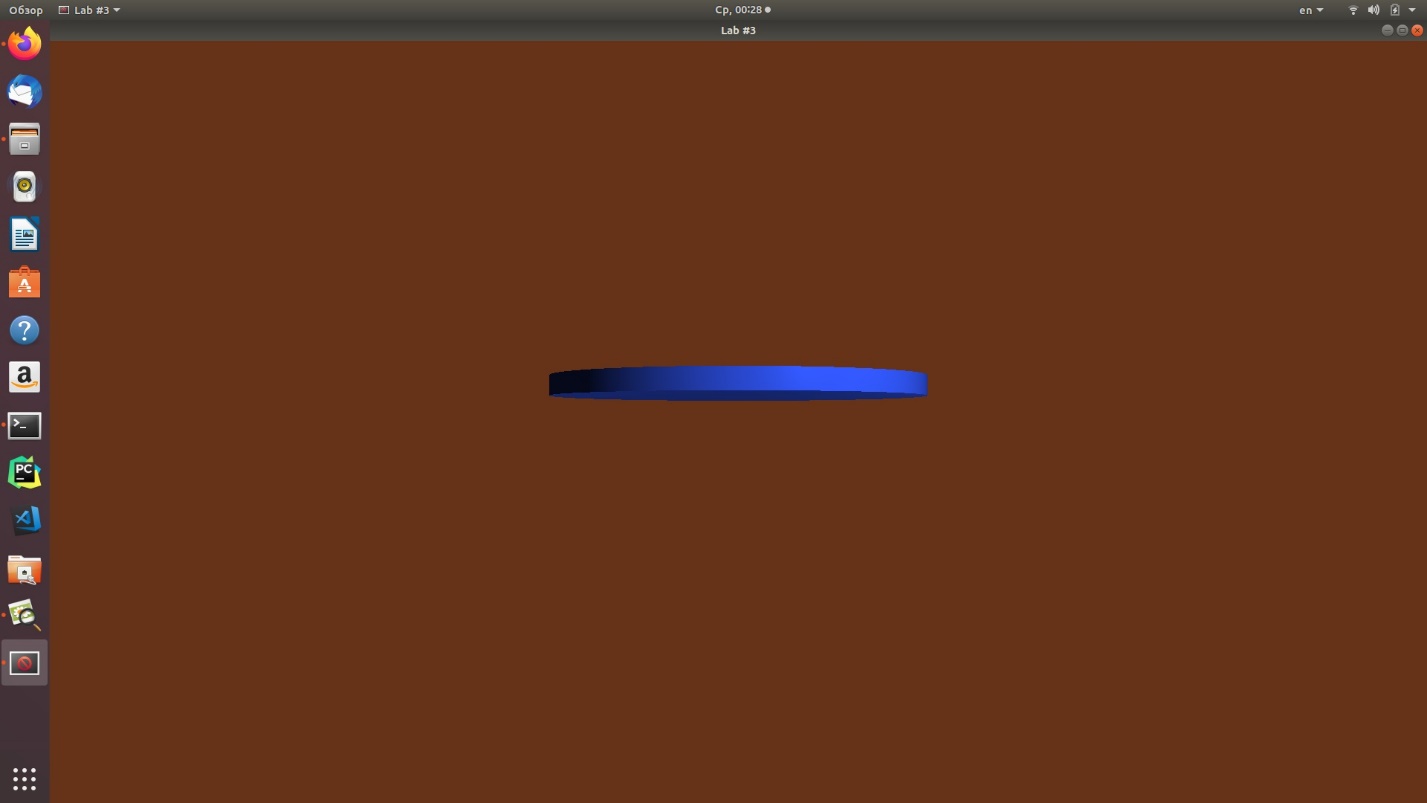
эффектов, график подстраивается под изменение размеров экрана и масштабируется

соответствующим образом.

С помощью нажатий клавиатуры можно вращать и масштабировать фигуру произ-

вольным образом:





4. Выводы

Выполнив 6 лабораторную работу по курсу «Компьютерная графика», я получил представление о как работают шейдеры, потренировался в работе с OpenGL, повторил как реализуется освещение в графических приложениях и

попрактиковался в этом сам на языке C++. Также я научился создавать объемные

фигуры.

В процессе решения задачи мне пришлось изучить много материала на эту тему.

Наиболее трудным для меня было вникнуть в математическую сторону этого вопроса.

Данная лабораторная работа показала мне важность математики в программировании, а также дала новые практические навыки, которые я непременно буду использовать в будущем.