

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ
по лабораторной работе №5
по дисциплине «Компьютерная графика»
Тема: «Программируемый графический конвейер. Шейдеры.»

Студент гр. 6381	_____	Фиалковский М.С.
Студент гр. 6381	_____	Афийчук И.И.
Преподаватель	_____	Герасимова Т.В.

Санкт-Петербург
2019

Задание.

Реализовать программу, симулирующую движение фонариком по стене при отсутствии других источников света с помощью современных подходов по использованию библиотеки OpenGL.

Варианты 5_1_5:

Задание:

5. Эффект "освещение фонариком"

Пример:



Общие сведения.

Под современным подходом, указанным выше, имеется ввиду core-profile режим — более эффективный и гибкий чем официально устаревший в OpenGL 3.3 immediate режим. В нем мы сами специфицируем некоторые этапы графического конвейера, разрабатывая так называемый шейдеры.

В данной работе мы будем разрабатывать вершинный и фрагментный шейдеры.

Ход работы.

Сборка и компиляция программы осуществляется с помощью утилиты `stake` и компилятора `gcc` из пакета `MinGW`. Графический интерфейс выполнен с помощью библиотеки `FLTK`. В качестве обертки над библиотекой `OpenGL`, используется `GLEW`. `FLTK` обеспечивает создание контекста и имеет интерфейс для обращения к некоторым командам `OpenGL`.

Загрузка шейдеров происходит в несколько этапов. В самом начале их исходный код читается из указанных файлов в `vShaderCode` и `fShaderCode`. Затем этот исходный код компилируется:

```
vertex = glCreateShader(GL_VERTEX_SHADER);
glShaderSource(vertex, 1, &vShaderCode, NULL);
glCompileShader(vertex);
...
fragment = glCreateShader(GL_FRAGMENT_SHADER);
glShaderSource(fragment, 1, &fShaderCode, NULL);
glCompileShader(fragment);
```

При удачной компиляции полученные шейдерные программы линкуются в один шейдер с помощью команд:

```
this->Program = glCreateProgram();
glAttachShader(this->Program, vertex);
glAttachShader(this->Program, fragment);
glLinkProgram(this->Program);
```

Полученный таким образом шейдер мы затем используем в игровом цикле при отрисовке требуемой графики.

Рассмотрим сам код шейдеров:

1) Фрагментный:

```
#version 330 core
in vec3 ourColor;
in vec2 TexCoord;
out vec4 color;
uniform vec2 mousePos;
uniform sampler2D ourTexture;
uniform float fadeDistance;

float getDistance(vec2 p1, vec2 p2) {
    return sqrt(pow(p1.x-p2.x, 2) + pow(p1.y-p2.y, 2));
}
```

```

float getLightIntensity() {
    float dist = getDistance(gl_FragCoord.xy, mousePos);
    return fadeDistance / dist;
}

void main(){
    color = texture(ourTexture, TexCoord) * getLightIntensity();
}

```

2) Вершинный:

```

#version 330 core
layout (location = 0) in vec3 position;
layout (location = 1) in vec3 color;
layout (location = 2) in vec2 texCoord;

out vec3 ourColor;
out vec2 TexCoord;

void main()
{
    gl_Position = vec4(position, 1.0f);
    ourColor = color;
    TexCoord = texCoord;
}

```

Для получения требуемого эффекта в шейдер требуется передавать текущие координаты мыши при их изменении и текстуру для наложения. Для этого объявим в фрагментном шейдере (так как именно он отвечает за вычисление цвета) переменные:

```

uniform vec2 mousePos;
uniform sampler2D ourTexture;

```

Их значения будем менять с помощью специальных команд OpenGL, например:

```

if (event == FL_MOVE){
    auto x = FL::event_x();
    auto y = FL::event_y();
    glUniform2f(glGetUniformLocation(shaderProgram.Program, "mousePos"),
x, y);
    redraw();
}

```

Тестирование.

Справа от области отрисовки находятся элементы UI, с помощью которых выводятся сообщения от графической системы.

Вслед за движением мыши по области отрисовки меняется и часть освещенной фонариком стены. Двигая колёсиком мыши можно отдалять и приближать источник света.



Вывод.

В процессе выполнения лабораторной работы была разработана программа, требуемый по заданию эффект. При выполнении работы были приобретены навыки работы с графическим конвейером и шейдерным программным из графической библиотеки OpenGL.

Исходный код метода Draw:

```
#include "SimpleGL3Window.hpp"

SimpleGL3Window::SimpleGL3Window(int x, int y, int w, int h) :
FL_GL_Window(x, y, w, h) {
    mode(FL_RGB8 | FL_DOUBLE | FL_OPENGL3);
}

void SimpleGL3Window::draw(void) {
    shaderProgram.readAndCompile("Shaders/vertex.shader",
    "Shaders/fragment.shader");
    LoadTexture("texture.jpg");
    LoadBuffers();

    glClearColor(0.2f, 0.3f, 0.3f, 1.0f);
    glClear(GL_COLOR_BUFFER_BIT);

    glBindTexture(GL_TEXTURE_2D, texture);
    shaderProgram.Use();
    glUniform1f(glGetUniformLocation(shaderProgram.Program, "fadeDistance"),
    fadeDistance);

    glBindVertexArray(VAO);
    glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, 0);
    glBindVertexArray(0);
}

int SimpleGL3Window::handle(int event) {
    static int first = 1;
    if (first && event == FL_SHOW && shown()) {
        first = 0;
        make_current();
        {
            GLenum err = glewInit(); // defines pters to functions of OpenGL
V 1.2 and above
            if (err)
                FL::warning("glewInit() failed returning %u", err);
            else
                add_output("Using GLEW %s\n", glewGetString(GLEW_VERSION));
        }
        const uchar* glv = glGetString(GL_VERSION);
        add_output("GL_VERSION = %s\n", glv);
    }

    if (event == FL_MOVE){
        auto x = FL::event_x();
        auto y = FL::event_y();
    }
}
```

```

        glUniform2f(glGetUniformLocation(shaderProgram.Program, "mousePos"),
x, y);
        redraw();
    }

    if (event == FL_MOUSEWHEEL) {
        auto scroll_size = FL::event_dy();
        fadeDistance -= static_cast<GLfloat>(scroll_size);
        if (fadeDistance < 0)
            fadeDistance = 0.0f;
        glUniform1f(glGetUniformLocation(shaderProgram.Program,
"fadeDistance"), fadeDistance);
        redraw();
    }
    return FL_GL_Window::handle(event);
}

void SimpleGL3Window::reset(void) {

}

void SimpleGL3Window::loadTexture(const char *file){
    static bool isLoading = false;
    if (isLoading)
        return;
    add_output("Loading file %s...\n", file);
    image = SOIL_load_image(file, &width, &height, 0, SOIL_LOAD_RGB);
    add_output("Size of texture: %d x %d.\n", width, height);
    // Load and create a texture
    glGenTextures(1, &texture);
    glBindTexture(GL_TEXTURE_2D, texture);
    // Set the texture wrapping parameters
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
    // Set texture filtering parameters
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    // Load image, create texture and generate mipmaps
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB,
GL_UNSIGNED_BYTE, image);
    glGenerateMipmap(GL_TEXTURE_2D);
    SOIL_free_image_data(image);
    glBindTexture(GL_TEXTURE_2D, 0);
    isLoading = true;
}

void SimpleGL3Window::loadBuffers(){
    static bool isLoading = false;

```



```

if (isLoading)
    return;
GLfloat vertices[] = {
    // Positions          // Colors          // Texture Coords
    1.0f,  1.0f, 0.0f,    1.0f, 0.0f, 0.0f,    1.0f, 1.0f, // Top Right
    1.0f, -1.0f, 0.0f,    0.0f, 1.0f, 0.0f,    1.0f, 0.0f, // Bottom Right
    -1.0f, -1.0f, 0.0f,    0.0f, 0.0f, 1.0f,    0.0f, 0.0f, // Bottom Left
    -1.0f,  1.0f, 0.0f,    1.0f, 1.0f, 0.0f,    0.0f, 1.0f // Top Left
};
GLuint indices[] = {
    0, 1, 3, // First Triangle
    1, 2, 3  // Second Triangle
};
GLuint VBO, EBO;
glGenVertexArrays(1, &VAO);
glGenBuffers(1, &VBO);
glGenBuffers(1, &EBO);

glBindVertexArray(VAO);
glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices), indices,
GL_STATIC_DRAW);

// Position attribute
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat),
(GLvoid*)0);
glEnableVertexAttribArray(0);
// Color attribute
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat),
(GLvoid*)(3 * sizeof(GLfloat)));
glEnableVertexAttribArray(1);
// TexCoord attribute
glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat),
(GLvoid*)(6 * sizeof(GLfloat)));
glEnableVertexAttribArray(2);
glBindVertexArray(0);
}

```