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: Расширения OpenGL, программируемый графический конвейер. Шейдеры.

Студент гр. 7383 _____ Бергалиев М.

Студент гр. 7383 _____ Власов Р.А.

Преподаватель _____ Герасимова Т.В.

Санкт-Петербург

2020

Цель работы

Ознакомление с возможностями использования программируемого графического конвейера.

Постановка задачи

Разработать визуальный эффект по заданию, реализованный средствами языка шейдеров GLSL.

Рельефное текстурирование на базе шейдеров:

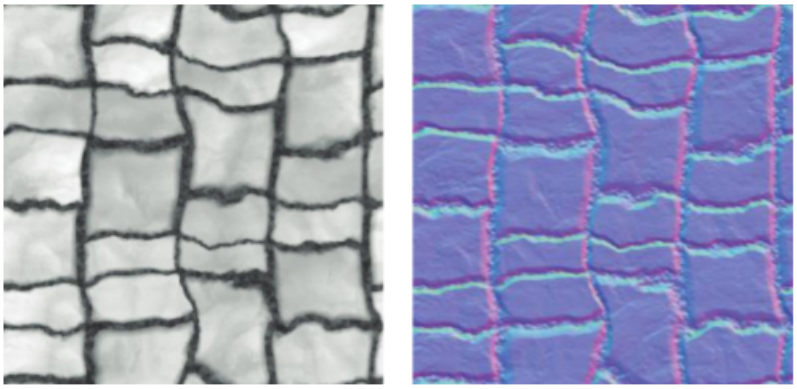

Рис. 0.1: Карта высот и соответствующая ей карта возмущённых нормалей. D В карте нормалей преобладает пастельный фиолетовый цвет, так как невозмущённый нормальный вектор $\langle 0, 0, 1 \rangle$ соответствует RGB-цвету $(\frac{1}{2}, \frac{1}{2}, 1)$.

Ход работы

Для создания и использования шейдерной программы создадим класс Shader:

```
class Shader
{
public:
    GLuint Program;
    Shader(const GLchar* vertexPath, const GLchar* fragmentPath);
    void Use();

    void setVec3(const char* uniform, GLfloat x, GLfloat y, GLfloat z);
    void setVec3(const char* uniform, glm::vec3 vec);
    void setMat4(const char* uniform, glm::mat4 mat);
    void setFloat(const char* uniform, GLfloat f);
};
```

Определим конструктор:

```
Shader::Shader(const GLchar* vertexPath, const GLchar* fragmentPath)
{
    std::string vertexCode;
```

```
    std::string fragmentCode;
    std::ifstream vShaderFile;
    std::ifstream fShaderFile;
    vShaderFile.exceptions(std::ifstream::badbit);
    fShaderFile.exceptions(std::ifstream::badbit);
    try
    {
        vShaderFile.open(vertexPath);
        fShaderFile.open(fragmentPath);
        std::stringstream vShaderStream, fShaderStream;
        vShaderStream << vShaderFile.rdbuf();
        fShaderStream << fShaderFile.rdbuf();
        vShaderFile.close();
        fShaderFile.close();
        vertexCode = vShaderStream.str();
        fragmentCode = fShaderStream.str();
    }
    catch(std::ifstream::failure e)
    {
              std::cout  <<  "ERROR::SHADER::FILE_NOT_SUCCESFULLY_READ"  <<
std::endl;
    }
    const GLchar* vShaderCode = vertexCode.c_str();
    const GLchar* fShaderCode = fragmentCode.c_str();
    GLuint vertex, fragment;
    GLint success;
    GLchar infoLog[512];
    vertex = glCreateShader(GL_VERTEX_SHADER);
    glShaderSource(vertex, 1, &vShaderCode, NULL);
    glCompileShader(vertex);
    glGetShaderiv(vertex, GL_COMPILE_STATUS, &success);
    if(!success)
    {
        glGetShaderInfoLog(vertex, 512, NULL, infoLog);
            std::cout  <<  "ERROR::SHADER::VERTEX::COMPILATION_FAILED\n"  <<
infoLog << std::endl;
    };
    fragment = glCreateShader(GL_FRAGMENT_SHADER);
    glShaderSource(fragment, 1, &fShaderCode, NULL);
    glCompileShader(fragment);
    glGetShaderiv(fragment, GL_COMPILE_STATUS, &success);
    if(!success)
    {
        glGetShaderInfoLog(fragment, 512, NULL, infoLog);
          std::cout << "ERROR::SHADER::FRAGMENT::COMPILATION_FAILED\n" <<
infoLog << std::endl;
    }
    this->Program = glCreateProgram();
    glAttachShader(this->Program, vertex);
    glAttachShader(this->Program, fragment);
    glLinkProgram(this->Program);
```

```
    glGetProgramiv(this->Program, GL_LINK_STATUS, &success);
    if(!success)
    {
        glGetProgramInfoLog(this->Program, 512, NULL, infoLog);
              std::cout  <<  "ERROR::SHADER::PROGRAM::LINKING_FAILED\n"  <<
infoLog << std::endl;
    }
    glDeleteShader(vertex);
    glDeleteShader(fragment);
}
```

Здесь происходит считывание исходных кодов шейдоров из файлов, компиляция вершинного и фрагментного шейдеров и связывание их в шейдерную программу. Чтобы использовать шейдерную программу, нужно использовать следующий метод класса Shader:

```
void Shader::Use(){
    glUseProgram(this->Program);
}
```

Для удобства определим класс камеры Camera:

```
class Camera
{
public:
    static Camera& get_instance() {
      static Camera camera;
      return camera;
    }
    void do_movement();
    glm::vec3& get_position() { return cameraPos; }
    glm::mat4 get_view();
    glm::mat4 get_projection(double side_ratio);
    void key_callback(int key, int action);
    void mouse_callback(double xpos, double ypos);
    void scroll_callback(double xoffset, double yoffset);

private:
    Camera() {}
    glm::vec3 cameraPos   = glm::vec3(0.0f, 0.0f,  3.0f);
    glm::vec3 cameraFront = glm::vec3(0.0f, 0.0f, -1.0f);
    glm::vec3 cameraUp    = glm::vec3(0.0f, 1.0f,  0.0f);
    GLfloat deltaTime = 0.0f;
    GLfloat lastFrame = 0.0f;
    bool keys[1024];
    GLfloat lastX = 400, lastY = 300;
    GLfloat yaw   = -90.0f;
    GLfloat pitch = 0.0f;
    bool firstMouse = true;
    GLfloat fov = 45.0f;
};
```

Этот класс содержит в себе информацию об ориентации в пространстве наблюдателя и угол обзора. Методы `get_view` и `get_projection` вычисляют матрицы вида и проекции соответственно.

Определим функции создания буфера графического конвейера, заполнение его данными и привязки атрибутов шейдеров к буферам:

```
void initBuffer(GLuint* buff, void* data, size_t size, GLuint type) {
    glGenBuffers(1, buff);
    glBindBuffer(type, *buff);
    glBufferData(type, size, data, GL_STATIC_DRAW);
}

void initVAO(GLuint* VAO, void* vertices, int vsize, void* indices, int
esize, std::vector<int> asize, int step, std::vector<int> offset) {
    GLuint VBO, EBO;
    glGenVertexArrays(1, VAO);
    glBindVertexArray(*VAO);
    initBuffer(&VBO, vertices, vsize*sizeof(GLfloat), GL_ARRAY_BUFFER);
                  initBuffer(&EBO,    indices,    esize*sizeof(GLfloat),
GL_ELEMENT_ARRAY_BUFFER);
    for(int i=0; i<asize.size(); ++i) {
           glVertexAttribPointer(i, asize[i], GL_FLOAT, GL_FALSE, step *
sizeof(GLfloat), (GLvoid*)(offset[i]*sizeof(GLfloat)));
        glEnableVertexAttribArray(i);
    }
    glBindVertexArray(0);
}
```

Определим процедуру загрузки текстуры:

```
void loadTexture(const char* file, GLuint* buff){
    glGenTextures(1, buff);
    glBindTexture(GL_TEXTURE_2D, *buff);
                  glTexParameteri(GL_TEXTURE_2D,    GL_TEXTURE_MIN_FILTER,
GL_LINEAR_MIPMAP_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);


    int width, height;
      unsigned  char*  image  =  SOIL_load_image(file,  &width,  &height,  0,
SOIL_LOAD_RGB);


      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);
}
```

Определим процедуру рисования рельефной плоскости:

```
void drawPlane(Shader& planeShader, GLuint planeVAO, glm::vec3 lightPos,
int width, int height) {
    planeShader.Use();

    glActiveTexture(GL_TEXTURE2);
    if(brick_plane)
        glBindTexture(GL_TEXTURE_2D, plane_diff);
    else glBindTexture(GL_TEXTURE_2D, plane_white);
                    glUniform1i(glGetUniformLocation(planeShader.Program,
"material.diffuse"), 2);

    glActiveTexture(GL_TEXTURE3);
    if(!without_normal_mapping) {
        glBindTexture(GL_TEXTURE_2D, plane_norm);
    }
    else {
        glBindTexture(GL_TEXTURE_2D, 0);
    }
    glUniform1i(glGetUniformLocation(planeShader.Program, "normal"), 3);

                planeShader.setVec3("viewPos",    camera.get_position().x,
camera.get_position().y, camera.get_position().z);

    planeShader.setFloat("material.shininess", 256.0f);
    planeShader.setVec3("material.specular", 0.2f, 0.2f, 0.2f);

    planeShader.setVec3("light.position", lightPos);
    planeShader.setVec3("light.ambient",  0.2f, 0.2f, 0.2f);
    planeShader.setVec3("light.diffuse",  0.5f, 0.5f, 0.5f);
    planeShader.setVec3("light.specular", 1.0f, 1.0f, 1.0f);

    glm::mat4 model(1.0f);
    model = glm::translate(model, glm::vec3(0.0f, -2.0f, -1.0f));
    model = glm::scale(model, glm::vec3(2.0f));

    draw(planeShader, width/height, model, planeVAO, 3*2);
}
```

Вершинный шейдер без рельефного тектурирования:

```
#version 330 core
layout (location = 0) in vec3 position;
layout (location = 1) in vec3 normal;
layout (location = 2) in vec2 texcoords;

uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;

out vec3 FragPos;
out vec3 Normal;
```

```
out vec2 TexCoords;

void main()
{
    gl_Position = projection * view * model * vec4(position, 1.0f);
    FragPos = vec3(model * vec4(position, 1.0f));
    Normal = normal;
    TexCoords = texcoords;
}
```

Фрагментный шейдер без рельефного текстурирования:

```
#version 330 core
in vec3 FragPos;
in vec3 Normal;
in vec2 TexCoords;

out vec4 color;

uniform vec3 viewPos;

struct Material {
    sampler2D diffuse;
    vec3 specular;
    float shininess;
};

uniform Material material;

struct Light {
    vec3 position;

    vec3 ambient;
    vec3 diffuse;
    vec3 specular;
};

uniform Light light;

void main()
{
    // ambient
        vec3  ambient  =  light.ambient  *  vec3(texture(material.diffuse,
TexCoords));

    // diffuse
    vec3 norm = normalize(Normal);
    vec3 lightDir = normalize(light.position - FragPos);
    float diff = max(dot(norm, lightDir), 0.0);
    vec3 diffuse = light.diffuse * (diff * vec3(texture(material.diffuse,
TexCoords)));

    // specular
```

```
    vec3 viewDir = normalize(viewPos - FragPos);
    vec3 halfwayDir = normalize(lightDir + viewDir);
              float   spec   =   pow(max(dot(halfwayDir,   norm),   0.0),
material.shininess);
    vec3 specular = light.specular * (spec * material.specular);

    vec3 result = ambient + diffuse + specular;
    color = vec4(result, 1.0f);
}
```

Вершинный шейдер с рельефным тектурированием:

```
#version 330 core
layout (location = 0) in vec3 position;
layout (location = 1) in vec2 texcoords;

uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;

out vec3 FragPos;
out vec2 TexCoords;

void main()
{
    gl_Position = projection * view * model * vec4(position, 1.0f);
    FragPos = vec3(model * vec4(position, 1.0f));
    TexCoords = texcoords;
}
```

Фрагментный шейдер с рельефным тектурированием:

```
#version 330 core
in vec3 FragPos;
in vec2 TexCoords;

out vec4 color;

uniform vec3 viewPos;

uniform sampler2D normal;
struct Material {
    sampler2D diffuse;
    vec3 specular;
    float shininess;
};

uniform Material material;

struct Light {
    vec3 position;

    vec3 ambient;
    vec3 diffuse;
```

```
    vec3 specular;
};

uniform Light light;

void main()
{
        vec3  ambient  =  light.ambient  *  vec3(texture(material.diffuse,
TexCoords));

    vec3 norm = texture(normal, TexCoords).rgb;
    norm = normalize(norm * 2.0 - 1.0);

    vec3 lightDir = normalize(light.position - FragPos);
    float diff = max(dot(norm, lightDir), 0.0);
    vec3 diffuse = light.diffuse * (diff * vec3(texture(material.diffuse,
TexCoords)));

    vec3 viewDir = normalize(viewPos - FragPos);
    vec3 reflectDir = reflect(-lightDir, norm);
    vec3 halfwayDir = normalize(lightDir + viewDir);
             float   spec   =   pow(max(dot(halfwayDir,   norm),   0.0),
material.shininess);
    vec3 specular = light.specular * (spec * material.specular);

    vec3 result = ambient + diffuse + specular;
    color = vec4(result, 1.0f);
}
```

Тестирование

Белая плоскость без и с текстурой рельефа, предложенной в задании, с двух ракурсов показаны на рис. 1-4.

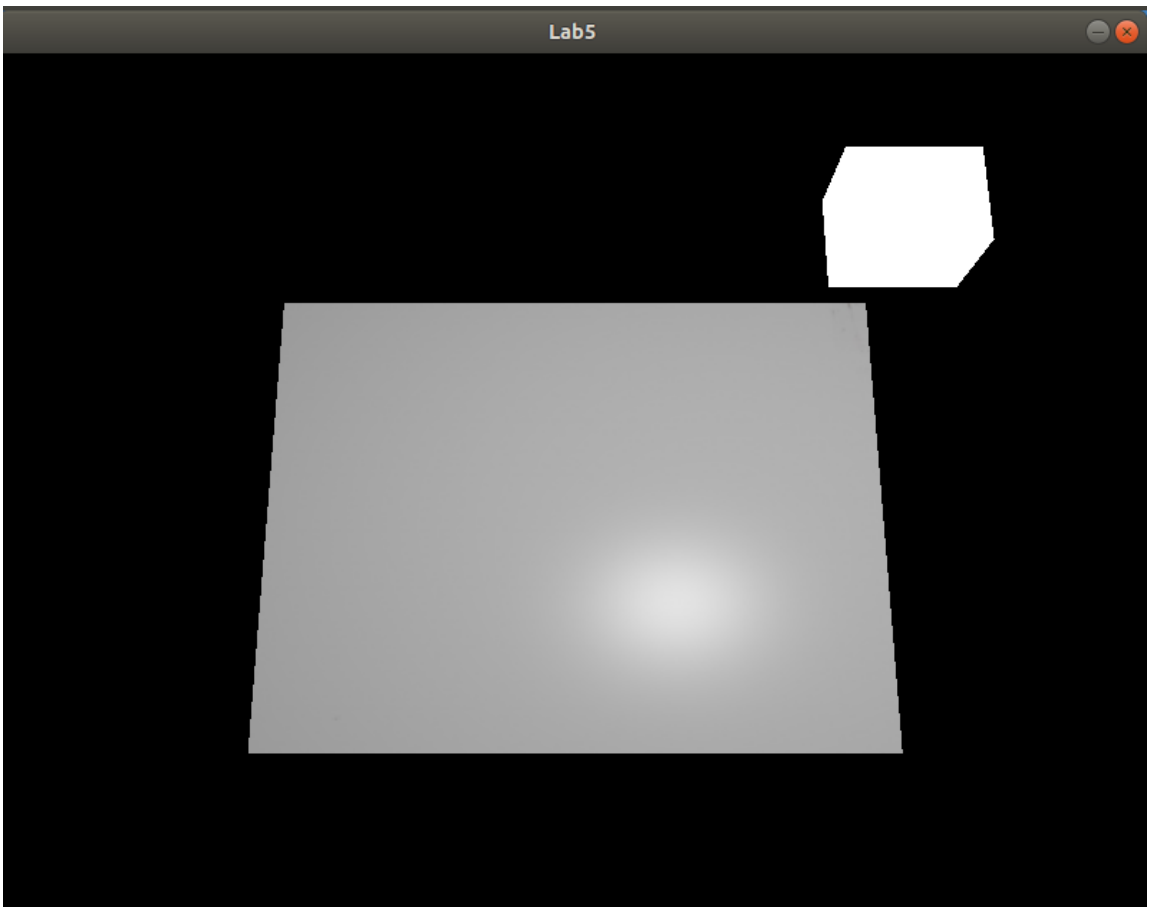

Рисунок 1 — Белая плоскость без рельефа

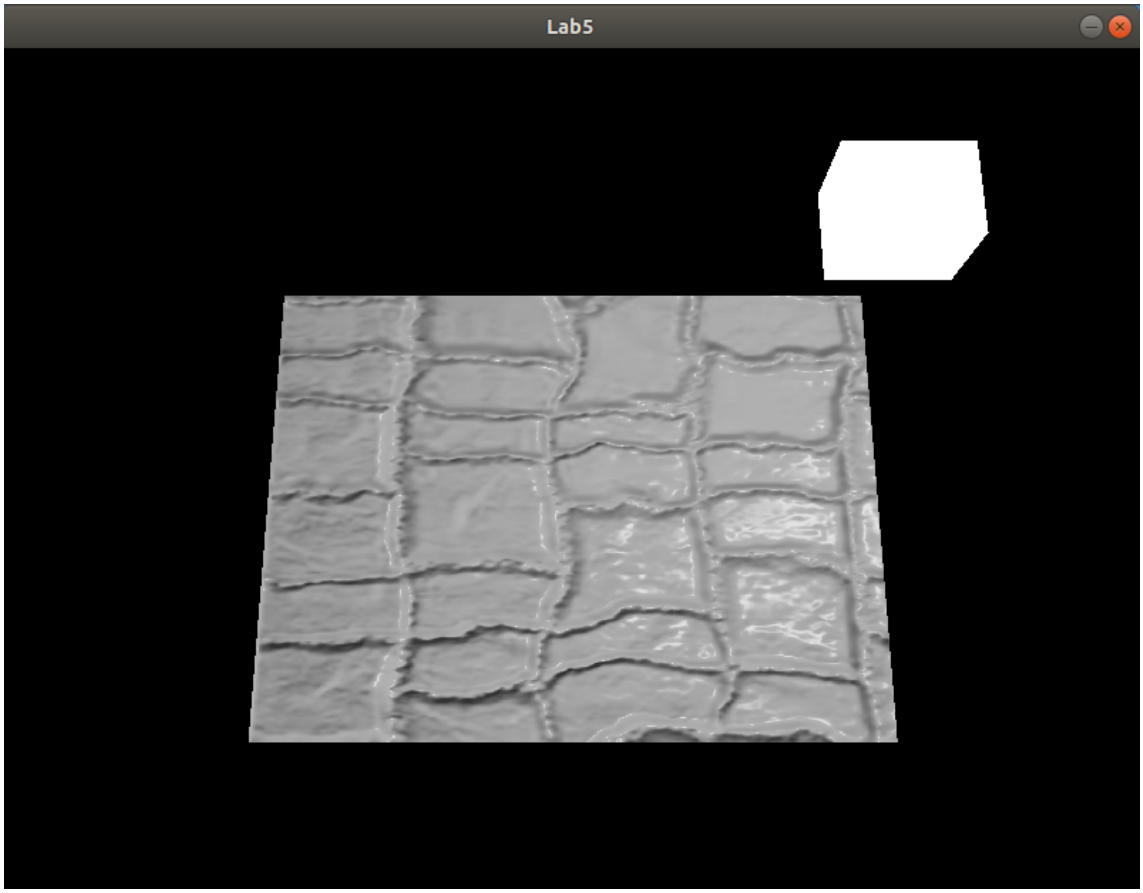

Рисунок 2 — Белая плоскость с рельефом

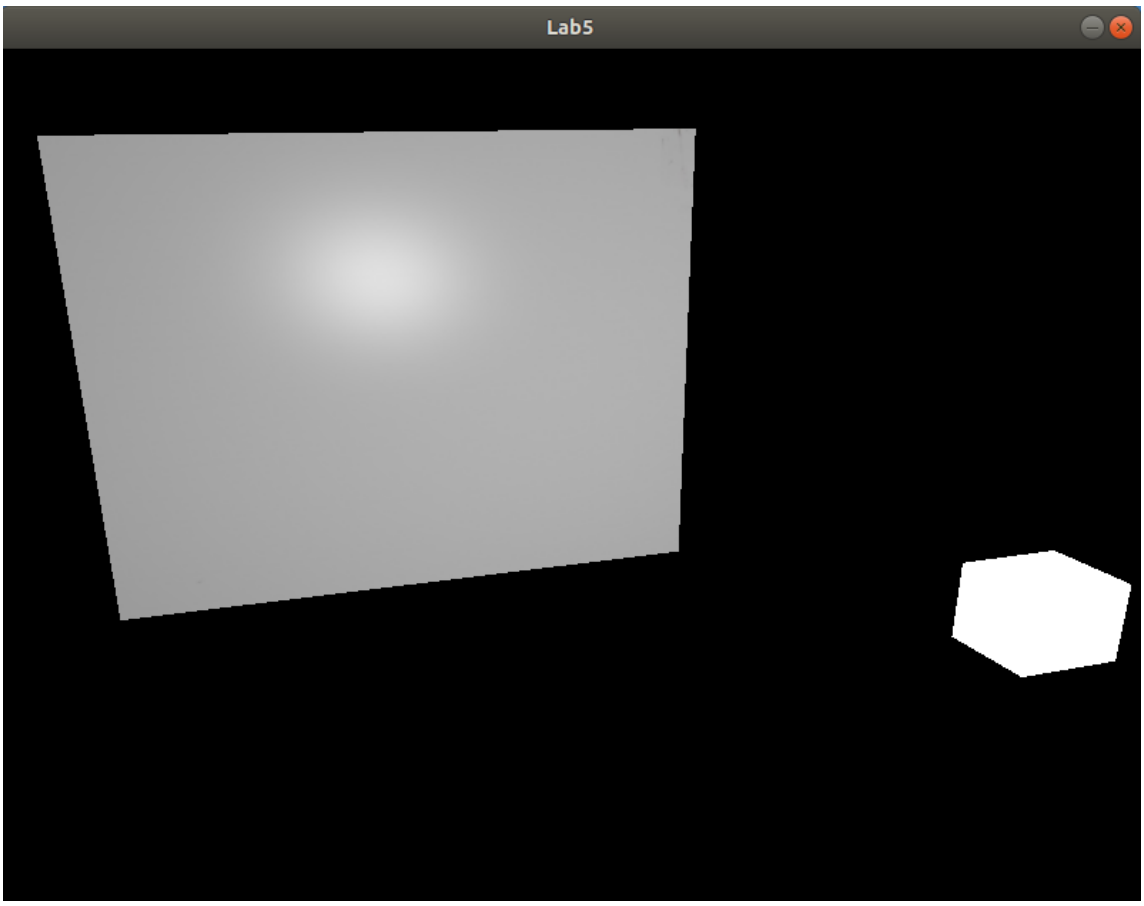

Рисунок 3 — Белая поверхность без рельефа с другого ракурса

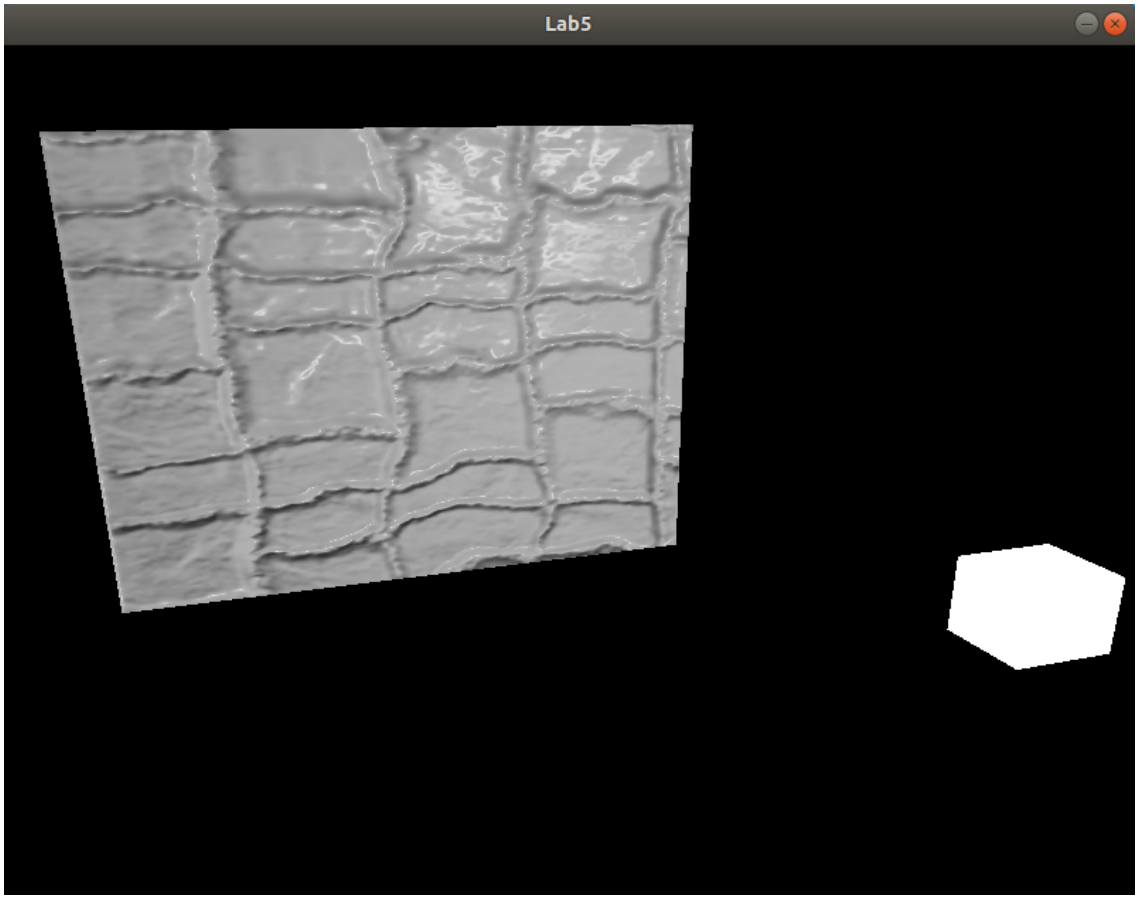

Рисунок 4 — Белая поверхность с рельефом с другого ракурса

Используем текстуру рельефа кирпичной стены и применим ее к белой поверхности. Результат показан на рис. 5, 6.

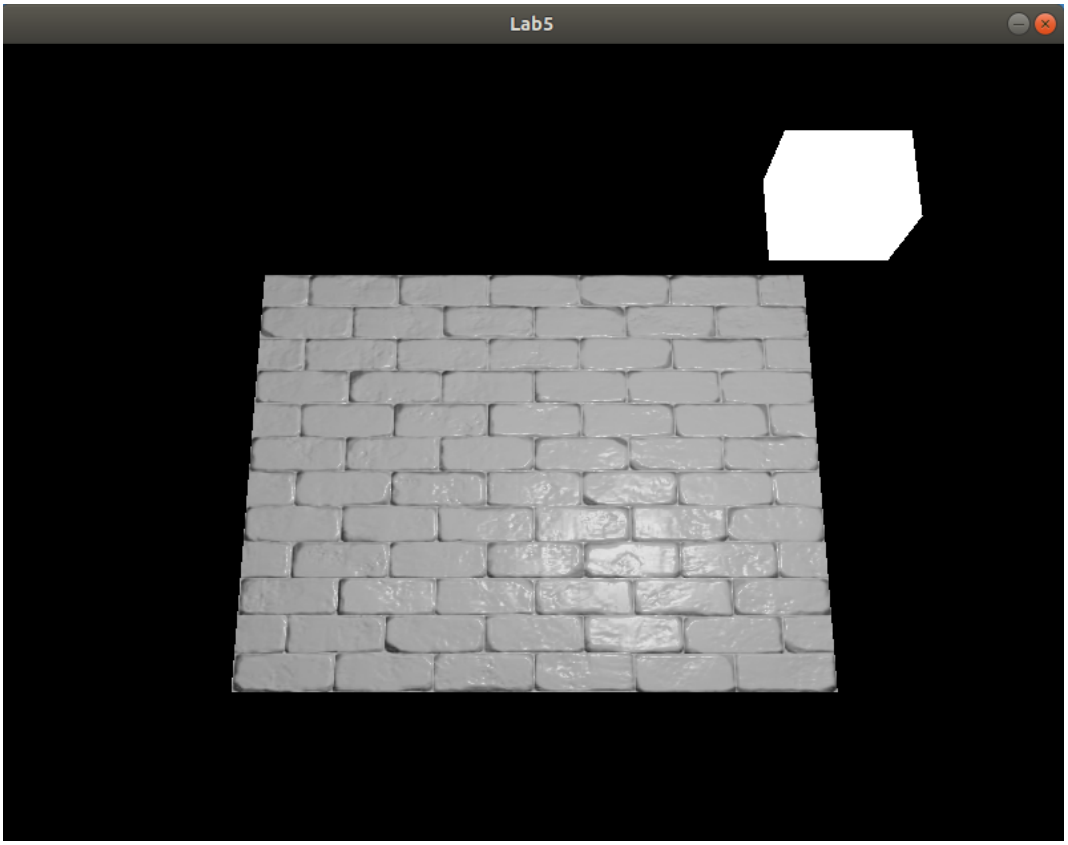

Рисунок 5 — Белая поверхность с текстурой рельефа кирпичной стены

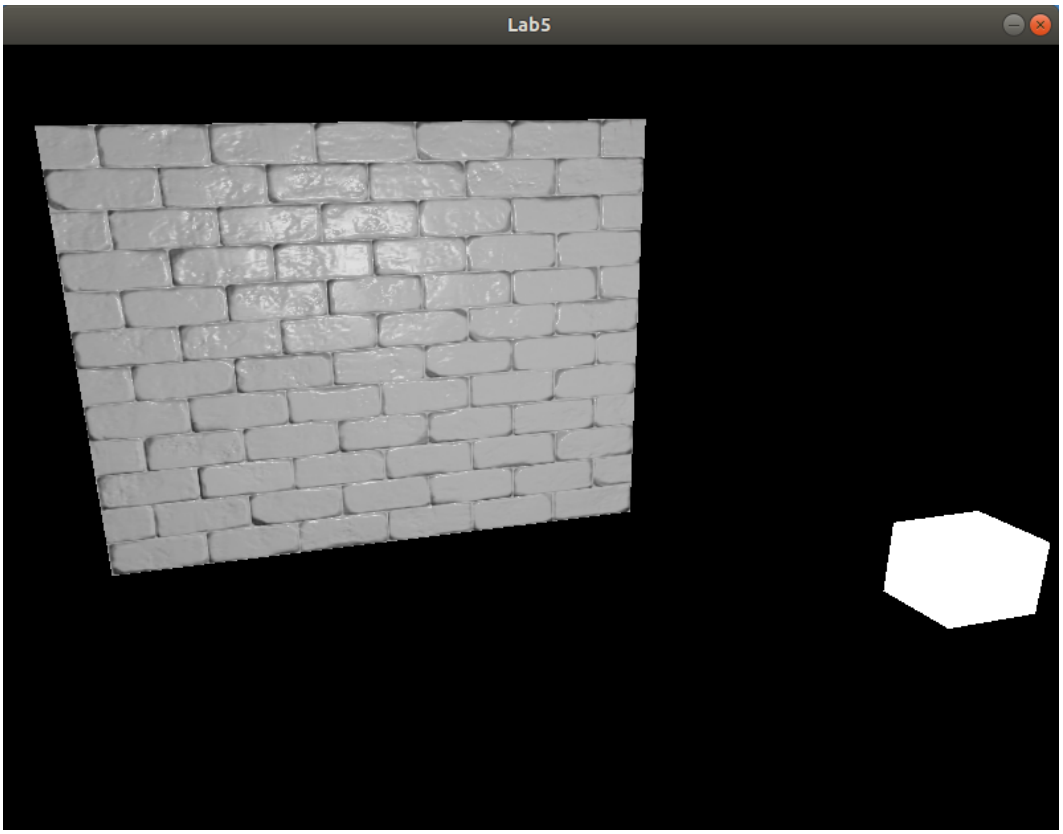

Рисунок 6 — Белая кирпичная стена с другого ракурса

Применим текстуру кирпичной стены и рассмотрим ее без и с рельефным текстурированием. Результаты показаны на рис. 7-10.

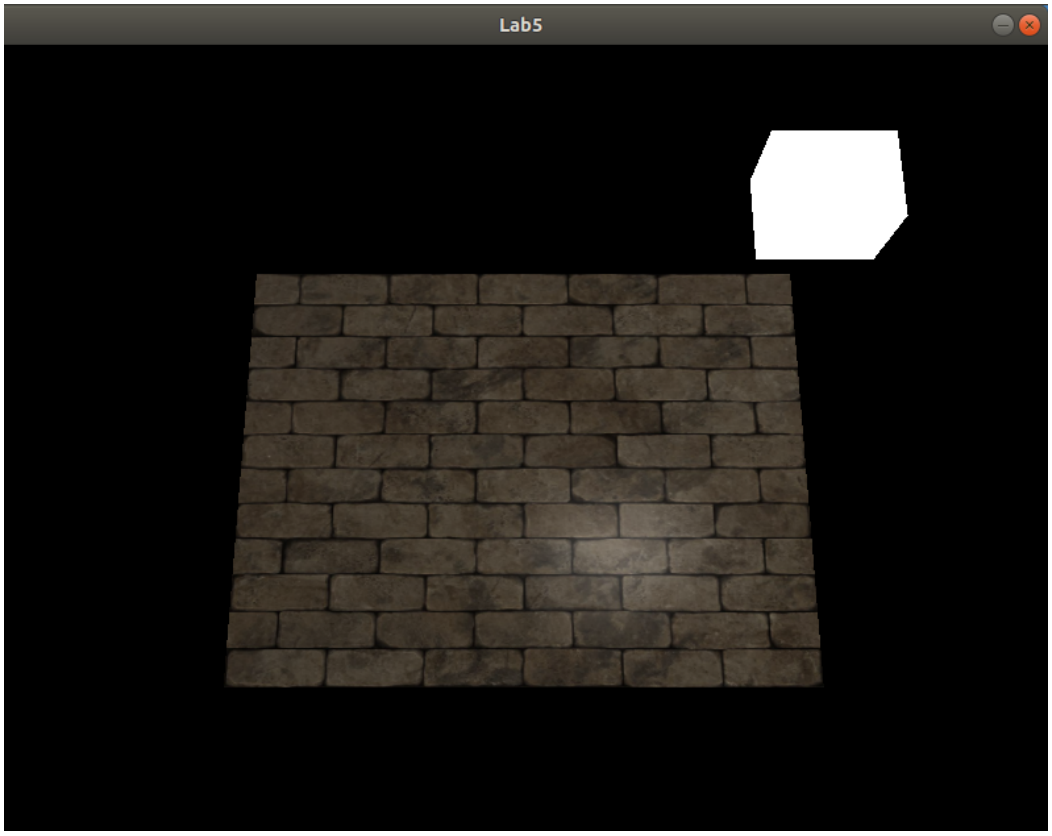

Рисунок 7 — Кирпичная стена без рельефа

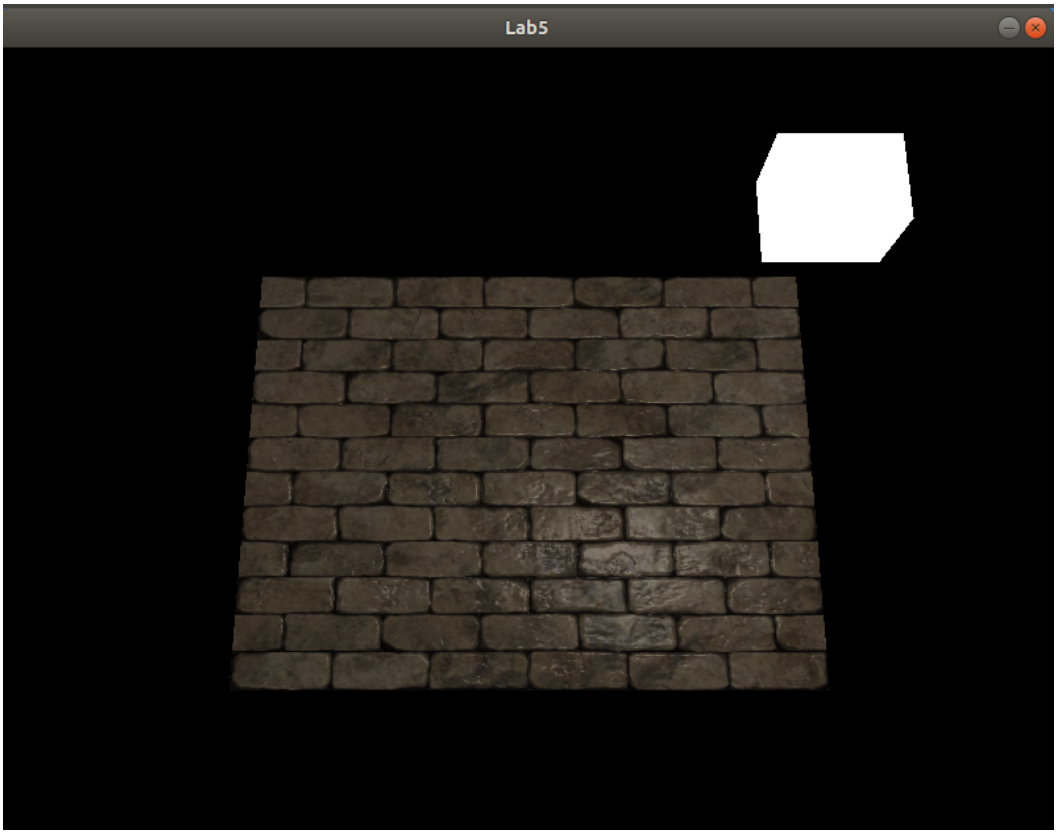

Рисунок 8 — Кирпичная стена с рельефом

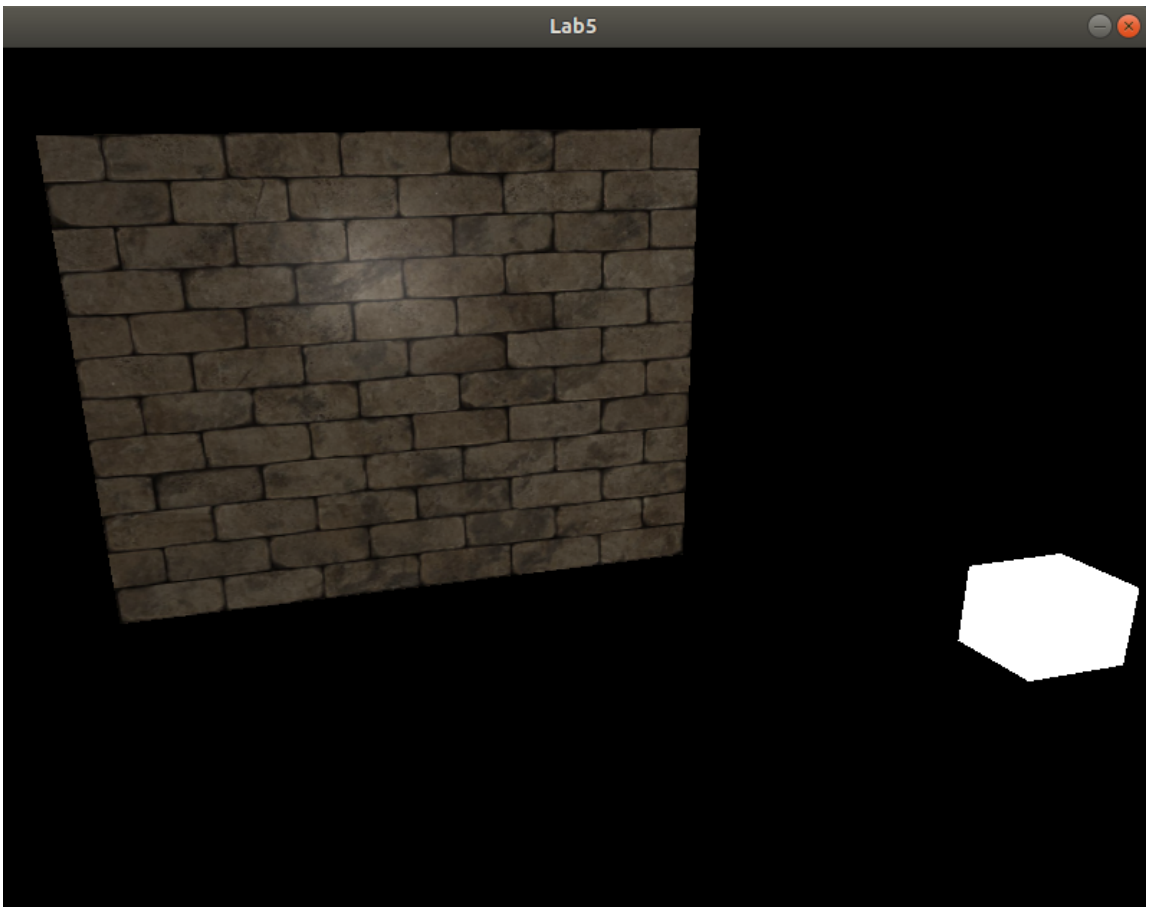

Рисунок 9 — Кирпичная стена без рельефа с другого ракурса

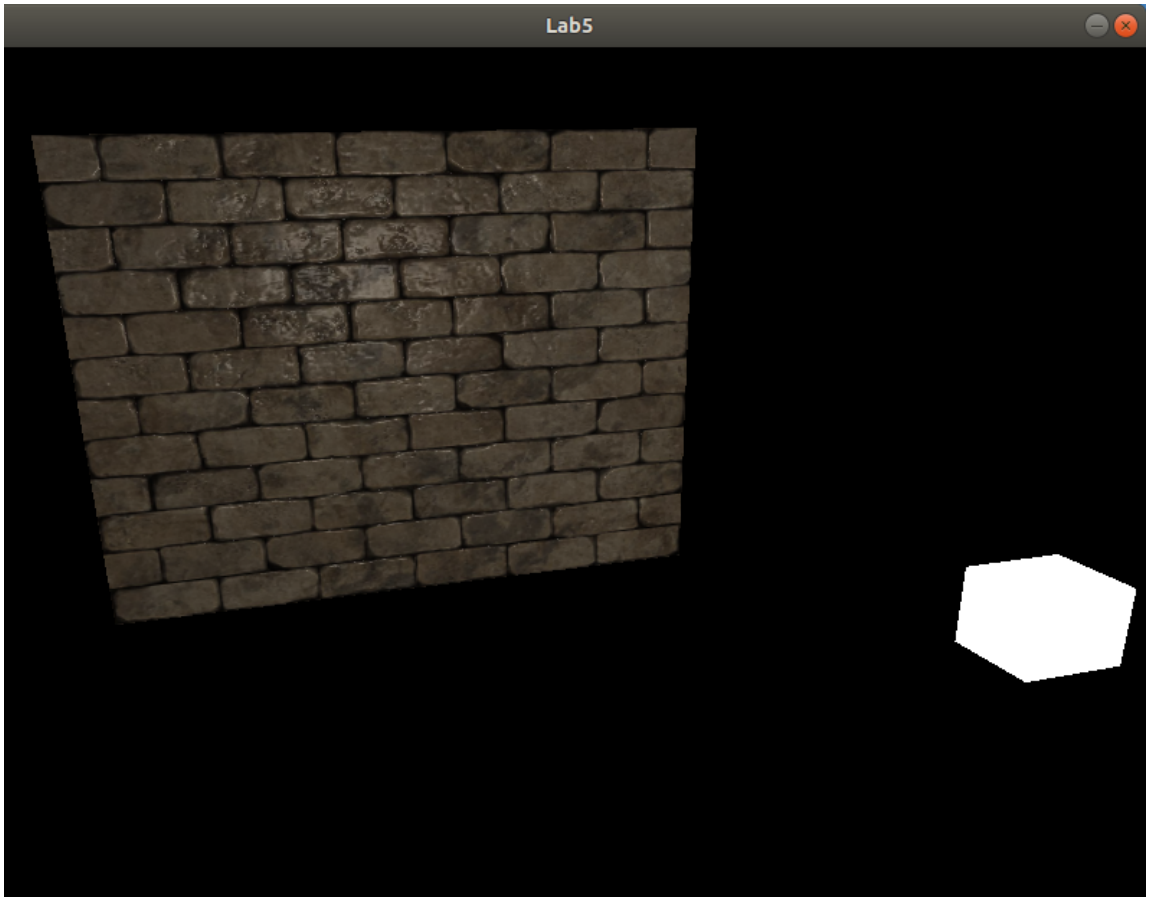

Рисунок 10 — Кирпичная стена с рельефом с другого ракурса

Выводы

Была разработана программа, рисующая плоскости без и с рельефным текстурированием. Рельефное текстурирование придает реалистичность в изображения. При выполнении работы были приобретены навыки работы с графической библиотекой OpenGL, а также навыки использования языка шейдеров GLSL.