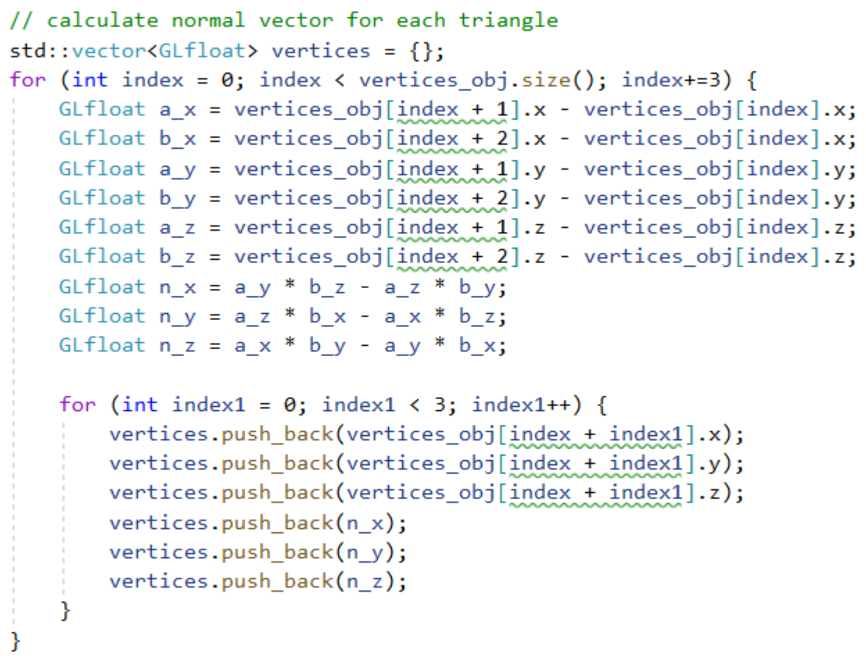
Assignment 6

Illumination effect

费杰 Fedor Ivachev 2019280373

The code is based on the light tutorial.

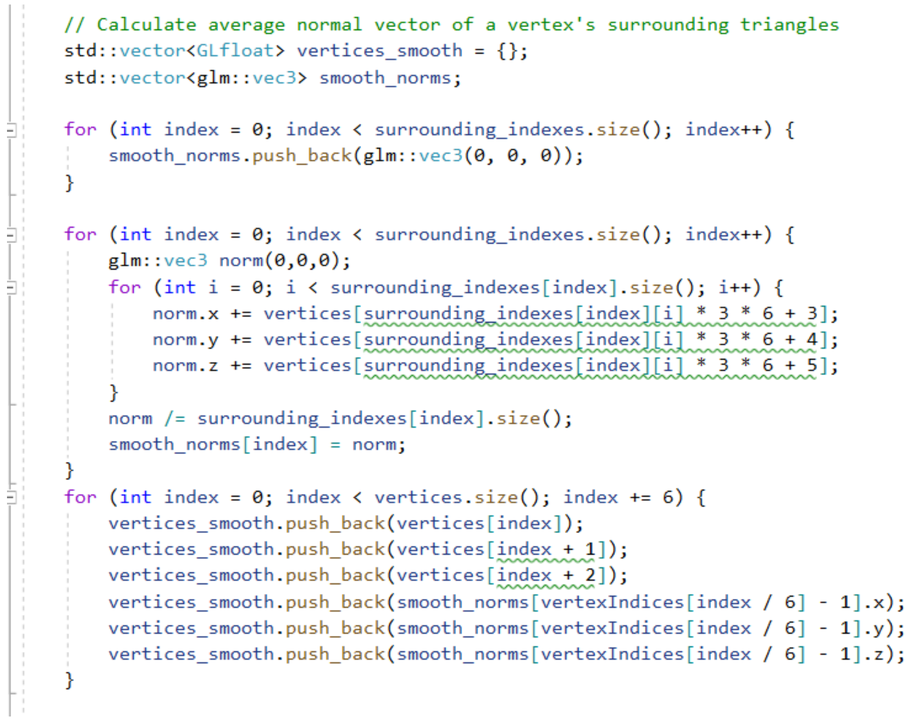
Firstly, we need to load the .obj file. Since the normals for each face are not calculated, we need to do it manually:



```
// calculate normal vector for each triangle
std::vector<GLfloat> vertices = {};
for (int index = 0; index < vertices_obj.size(); index+=3) {
    GLfloat a_x = vertices_obj[index + 1].x - vertices_obj[index].x;
    GLfloat b_x = vertices_obj[index + 2].x - vertices_obj[index].x;
    GLfloat a_y = vertices_obj[index + 1].y - vertices_obj[index].y;
    GLfloat b_y = vertices_obj[index + 2].y - vertices_obj[index].y;
    GLfloat a_z = vertices_obj[index + 1].z - vertices_obj[index].z;
    GLfloat b_z = vertices_obj[index + 2].z - vertices_obj[index].z;
    GLfloat n_x = a_y * b_z - a_z * b_y;
    GLfloat n_y = a_z * b_x - a_x * b_z;
    GLfloat n_z = a_x * b_y - a_y * b_x;

    for (int index1 = 0; index1 < 3; index1++) {
        vertices.push_back(vertices_obj[index + index1].x);
        vertices.push_back(vertices_obj[index + index1].y);
        vertices.push_back(vertices_obj[index + index1].z);
        vertices.push_back(n_x);
        vertices.push_back(n_y);
        vertices.push_back(n_z);
    }
}
```

To calculate the normal for the triangle ABC, we need to count AB x AC.

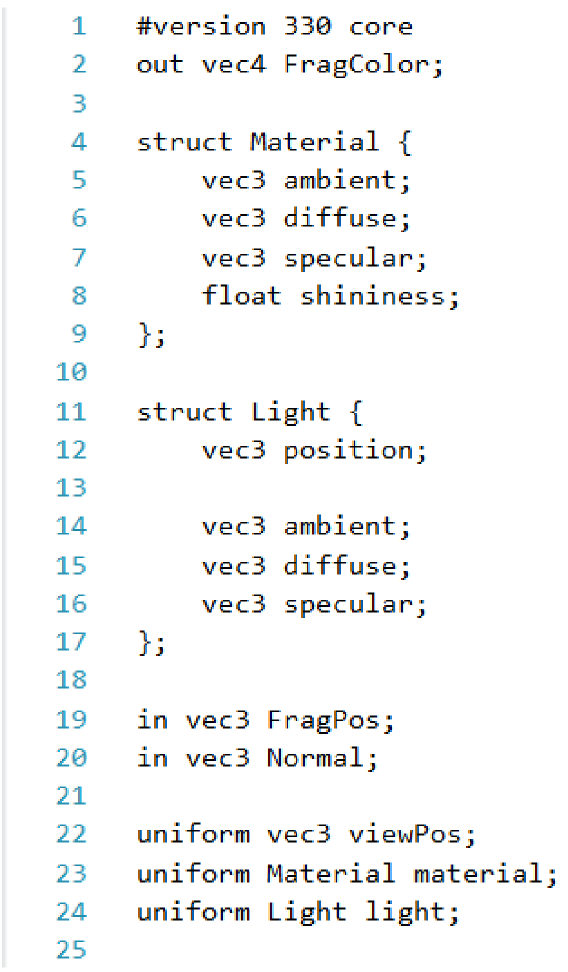
For the smooth shading, the average normal vector of a vertex's surrounding triangles need to be counted.



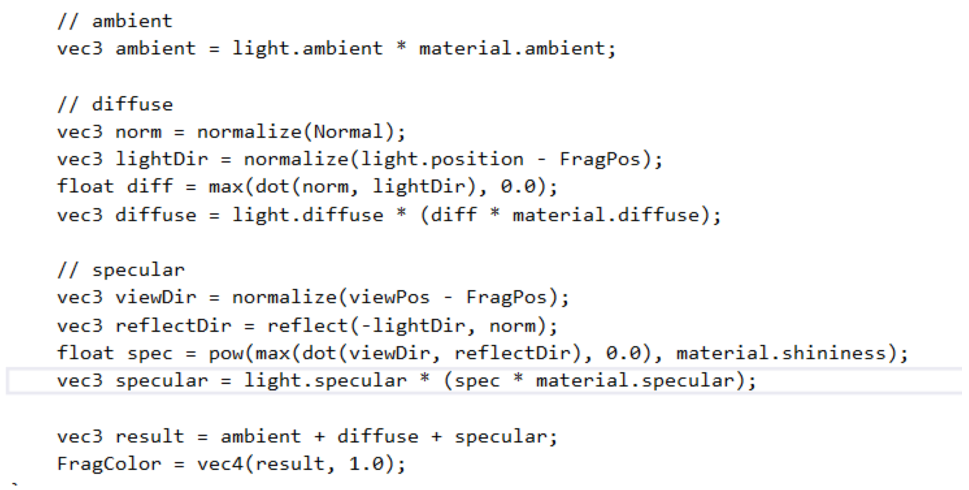
```
// Calculate average normal vector of a vertex's surrounding triangles
std::vector<GLfloat> vertices_smooth = {};
std::vector<glm::vec3> smooth_norms;

for (int index = 0; index < surrounding_indexes.size(); index++) {
    smooth_norms.push_back(glm::vec3(0, 0, 0));
}

for (int index = 0; index < surrounding_indexes.size(); index++) {
    glm::vec3 norm(0,0,0);
    for (int i = 0; i < surrounding_indexes[index].size(); i++) {
        norm.x += vertices[surrounding_indexes[index][i] * 3 * 6 + 3];
        norm.y += vertices[surrounding_indexes[index][i] * 3 * 6 + 4];
        norm.z += vertices[surrounding_indexes[index][i] * 3 * 6 + 5];
    }
    norm /= surrounding_indexes[index].size();
    smooth_norms[index] = norm;
}
for (int index = 0; index < vertices.size(); index += 6) {
    vertices_smooth.push_back(vertices[index]);
    vertices_smooth.push_back(vertices[index + 1]);
    vertices_smooth.push_back(vertices[index + 2]);
    vertices_smooth.push_back(smooth_norms[vertexIndices[index / 6] - 1].x);
    vertices_smooth.push_back(smooth_norms[vertexIndices[index / 6] - 1].y);
    vertices_smooth.push_back(smooth_norms[vertexIndices[index / 6] - 1].z);
}
```

Since we have already counted the normals for each face, we can access these values to count the average normal for each vertex. By storing the vertex surrounding indexes of faces and vertex indexes in each face counting it will not be a problem.

The lighting is calculated in material.frag.glsl as a combination of ambient, diffuse and specular components.

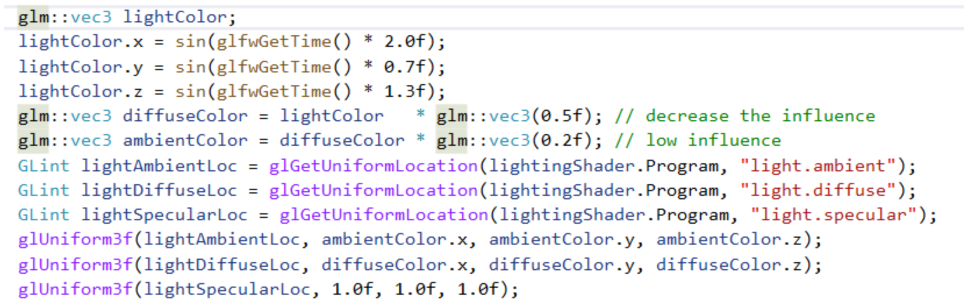


```
#version 330 core
out vec4 FragColor;

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

struct Light {
    vec3 position;

    vec3 ambient;
    vec3 diffuse;
    vec3 specular;
};

in vec3 FragPos;
in vec3 Normal;

uniform vec3 viewPos;
uniform Material material;
uniform Light light;

```



```
// ambient
vec3 ambient = light.ambient * material.ambient;

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

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

vec3 result = ambient + diffuse + specular;
FragColor = vec4(result, 1.0);
```

The light color changes according to time to generate different illumination effect:



```
glm::vec3 lightColor;
lightColor.x = sin(glfwGetTime() * 2.0f);
lightColor.y = sin(glfwGetTime() * 0.7f);
lightColor.z = sin(glfwGetTime() * 1.3f);
glm::vec3 diffuseColor = lightColor   * glm::vec3(0.5f); // decrease the influence
glm::vec3 ambientColor = diffuseColor * glm::vec3(0.2f); // low influence
GLint lightAmbientLoc = glGetUniformLocation(lightingShader.Program, "light.ambient");
GLint lightDiffuseLoc = glGetUniformLocation(lightingShader.Program, "light.diffuse");
GLint lightSpecularLoc = glGetUniformLocation(lightingShader.Program, "light.specular");
glUniform3f(lightAmbientLoc, ambientColor.x, ambientColor.y, ambientColor.z);
glUniform3f(lightDiffuseLoc, diffuseColor.x, diffuseColor.y, diffuseColor.z);
glUniform3f(lightSpecularLoc, 1.0f, 1.0f, 1.0f);
```

To change the mode, press X. Two modes are supported: