Assignment 1 Report

Made by Fedor Ivachev 费杰 2019280373

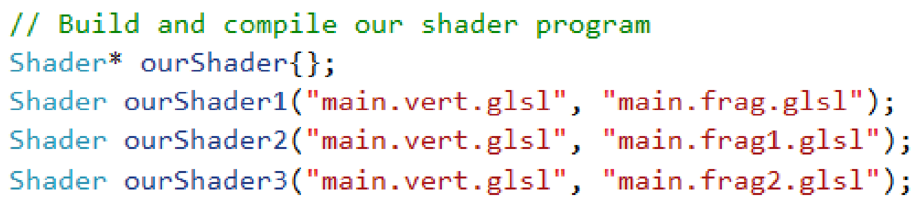
Display of 3D model

The code is based on Tutorial 3 and 4.

Loading an obj file requires parsing the data and saving it in vertices array (vector).

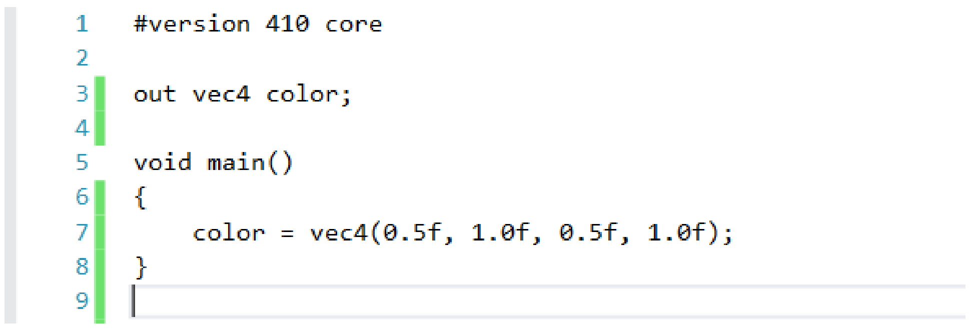
The vertices’ coordinates and the faces’ vertices indexes are read from the input obj file, the result is stored in out_vertices.

For each of the modes a shader is initialized



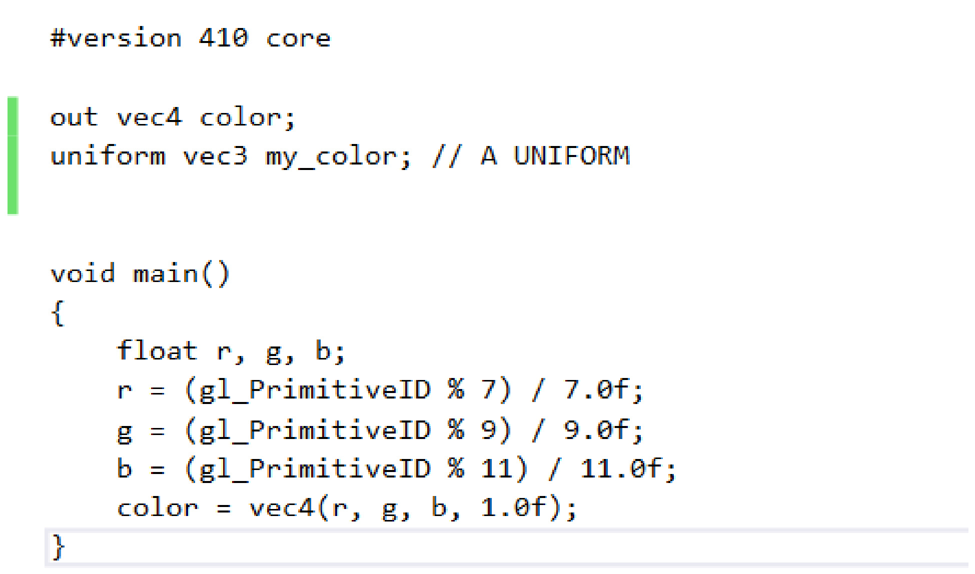
```
// Build and compile our shader program
Shader* ourShader{};
Shader ourShader1("main.vert.glsl", "main.frag.glsl");
Shader ourShader2("main.vert.glsl", "main.frag1.glsl");
Shader ourShader3("main.vert.glsl", "main.frag2.glsl");
```

The difference between them is only in Fragment shaders:



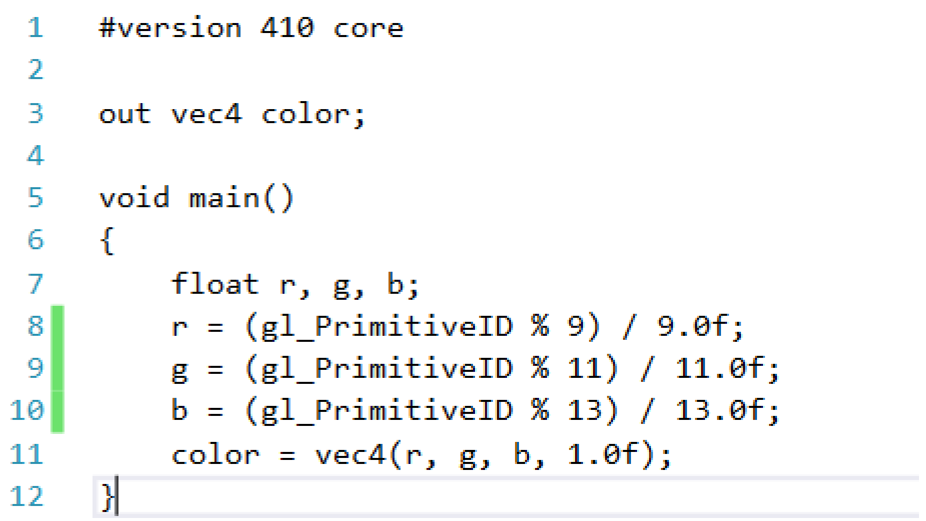
```
#version 410 core

out vec4 color;

void main()
{
    color = vec4(0.5f, 1.0f, 0.5f, 1.0f);
}
```

The first shader is using in displaying vertex mode



```
#version 410 core

out vec4 color;
uniform vec3 my_color; // A UNIFORM


void main()
{
    float r, g, b;
    r = (gl_PrimitiveID % 7) / 7.0f;
    g = (gl_PrimitiveID % 9) / 9.0f;
    b = (gl_PrimitiveID % 11) / 11.0f;
    color = vec4(r, g, b, 1.0f);
}
```

The second shader is using in displaying wireframe mode



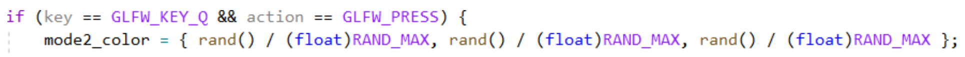
```
#version 410 core

out vec4 color;

void main()
{
    float r, g, b;
    r = (gl_PrimitiveID % 9) / 9.0f;
    g = (gl_PrimitiveID % 11) / 11.0f;
    b = (gl_PrimitiveID % 13) / 13.0f;
    color = vec4(r, g, b, 1.0f);
}
```

And the third shader is using in displaying face mode. Each of the primitives is rendered with its unique color.

Face and edge mode is displayed by rendering the mesh in face mode and then changing shader and rendering in wireframe mode.

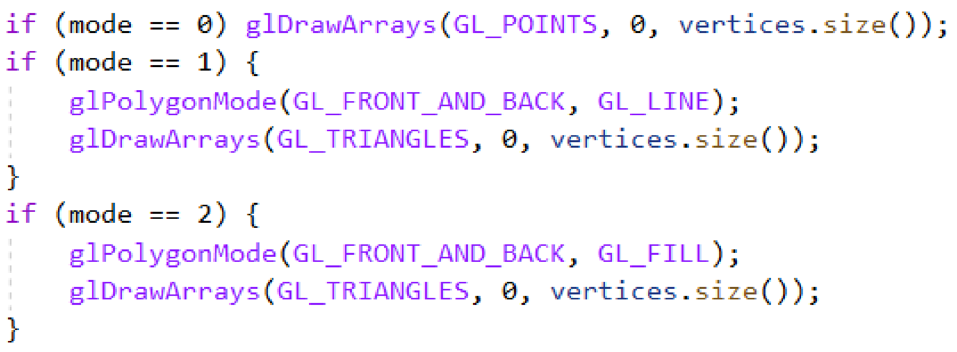
To switch between modes, press 1,2,3,4.

To change the color of the model in wireframe mode, press Q.



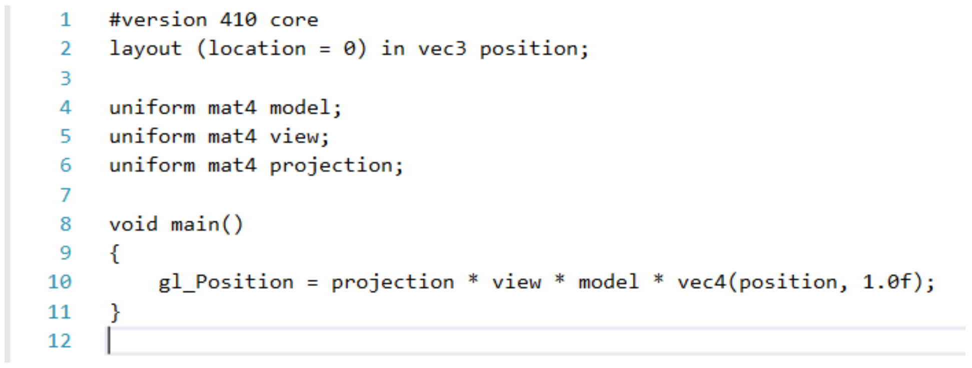
```
if (key == GLFW_KEY_Q && action == GLFW_PRESS) {
    mode2_color = { rand() / (float)RAND_MAX, rand() / (float)RAND_MAX, rand() / (float)RAND_MAX };
```

The color is defined by this formula and is the uniform value my_color in the second fragment shader.



```
if (mode == 0) glDrawArrays(GL_POINTS, 0, vertices.size());
if (mode == 1) {
    glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
    glDrawArrays(GL_TRIANGLES, 0, vertices.size());
}
if (mode == 2) {
    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
    glDrawArrays(GL_TRIANGLES, 0, vertices.size());
}
```

Drawing the modes.



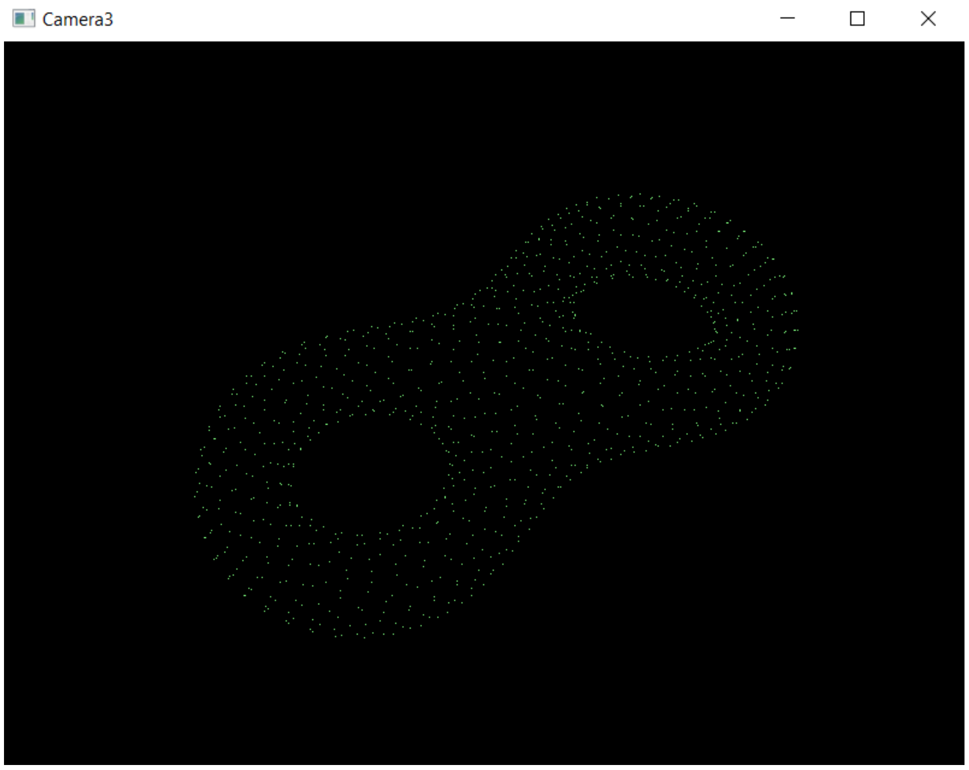
```
#version 410 core
layout (location = 0) in vec3 position;

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

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

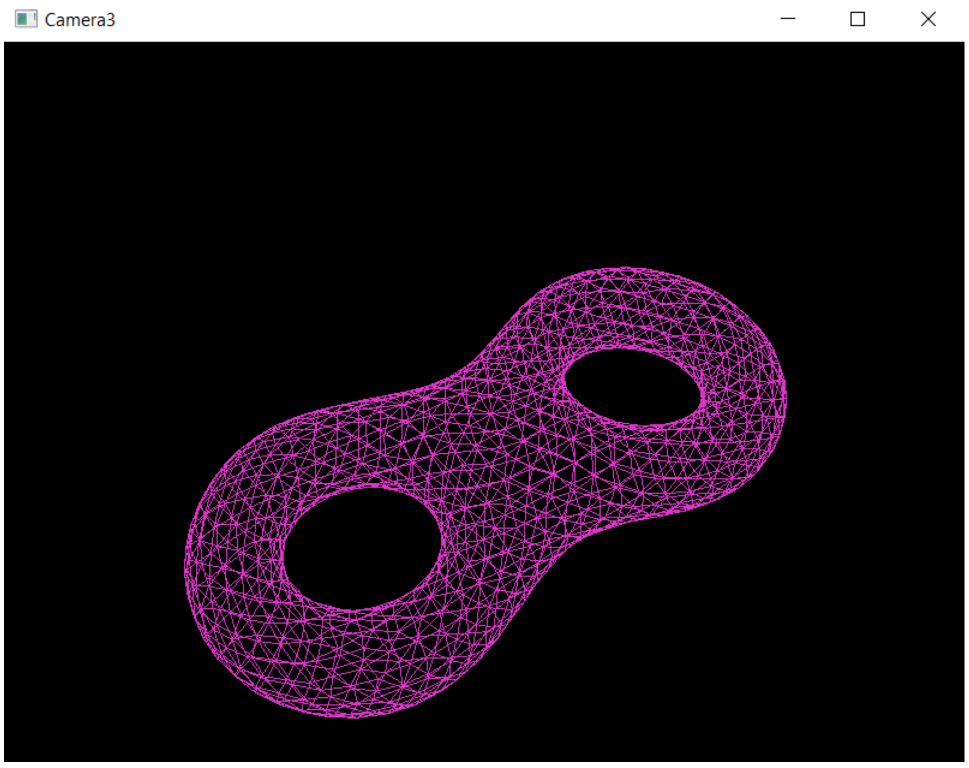
The vertex shader. The position of each vertex is counted using the projection, view, model and position matrices.

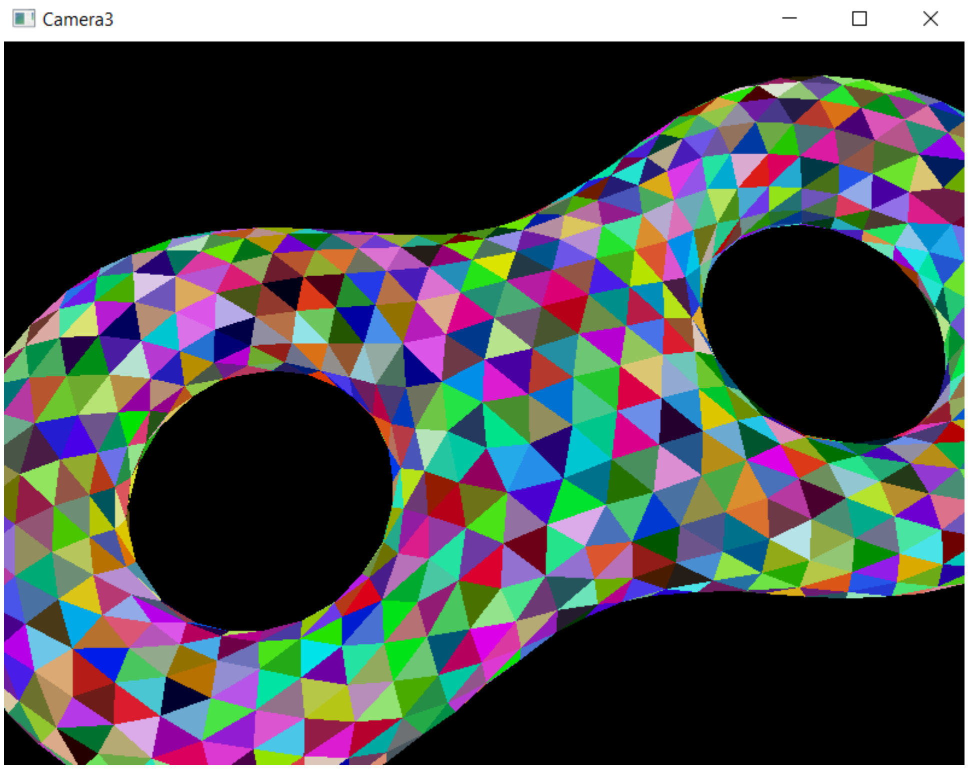
In the requirements the camera is not moving, while the model needs to be rotated and translated. At the same time, if we say that camera coordinates were always (0,0,0), then the mesh is moving instead. So the requirements are completed.

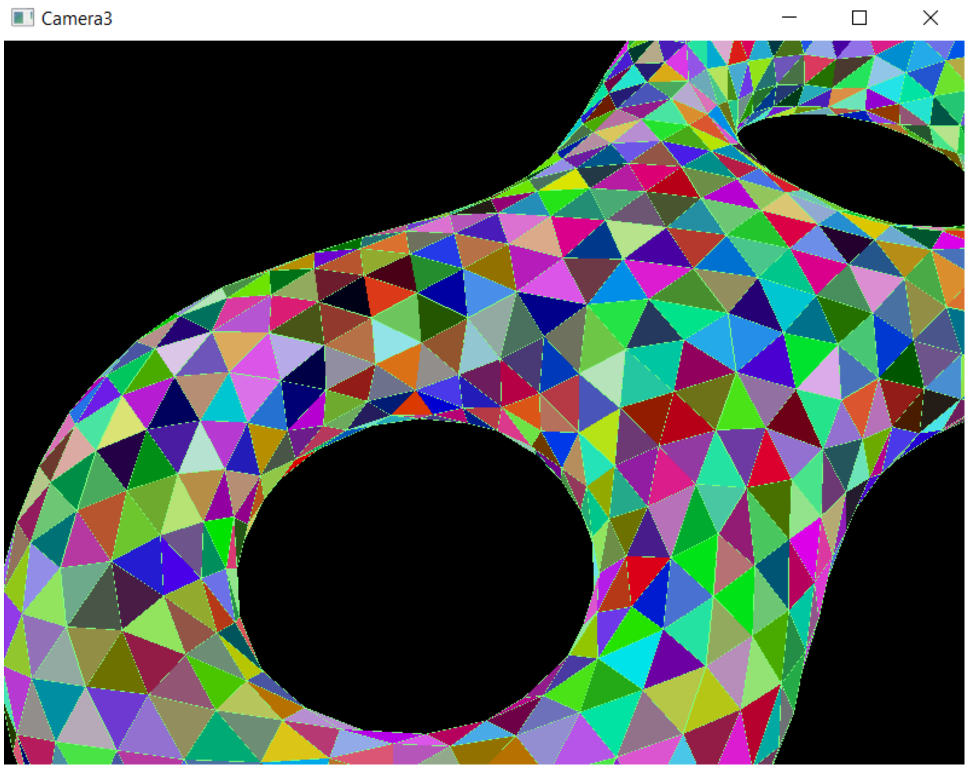
The model is rotated by using the touchpad, but it can be easily changed to using keyboard only, similar to the translation of the model.

As a result, here are some screenshots of the program:


Camera3


Camera3


Camera3


Camera3