OpenGL Museum Corridor Demo
ECE-433

User Manual and Future Development Plans

Nikolaos Koxenoglou
Eleni Veroni
Irene Tsitsopoulou

Contents

Table of Contents

Chapter 1: Intro

This chapter serves to introduce the user to the main concept of this demo.

As it stands, the corridor features six of the most basic shapes and actions that the OpenGL community deems as the bare basics for someone to begin their journey in its development. These shapes and transformations are taken from the lectures, labs as well as other OpenGL tutorials. They are: basic shapes, textures, lighting, rotations and combinations of these. The aim is to cover these concepts in a simple and minimalistic way, to make the experience of learning OpenGL more interactive, educational and fun.

This demo has many possible uses: from a simple tech demo to demonstrate what OpenGL is capable of to a user with no technical know-how, as well as a quick reference to someone already knowing or learning OpenGL as it contains important snippets of code regarding the most used features in one source code.

Chapter 2: Basic Usage

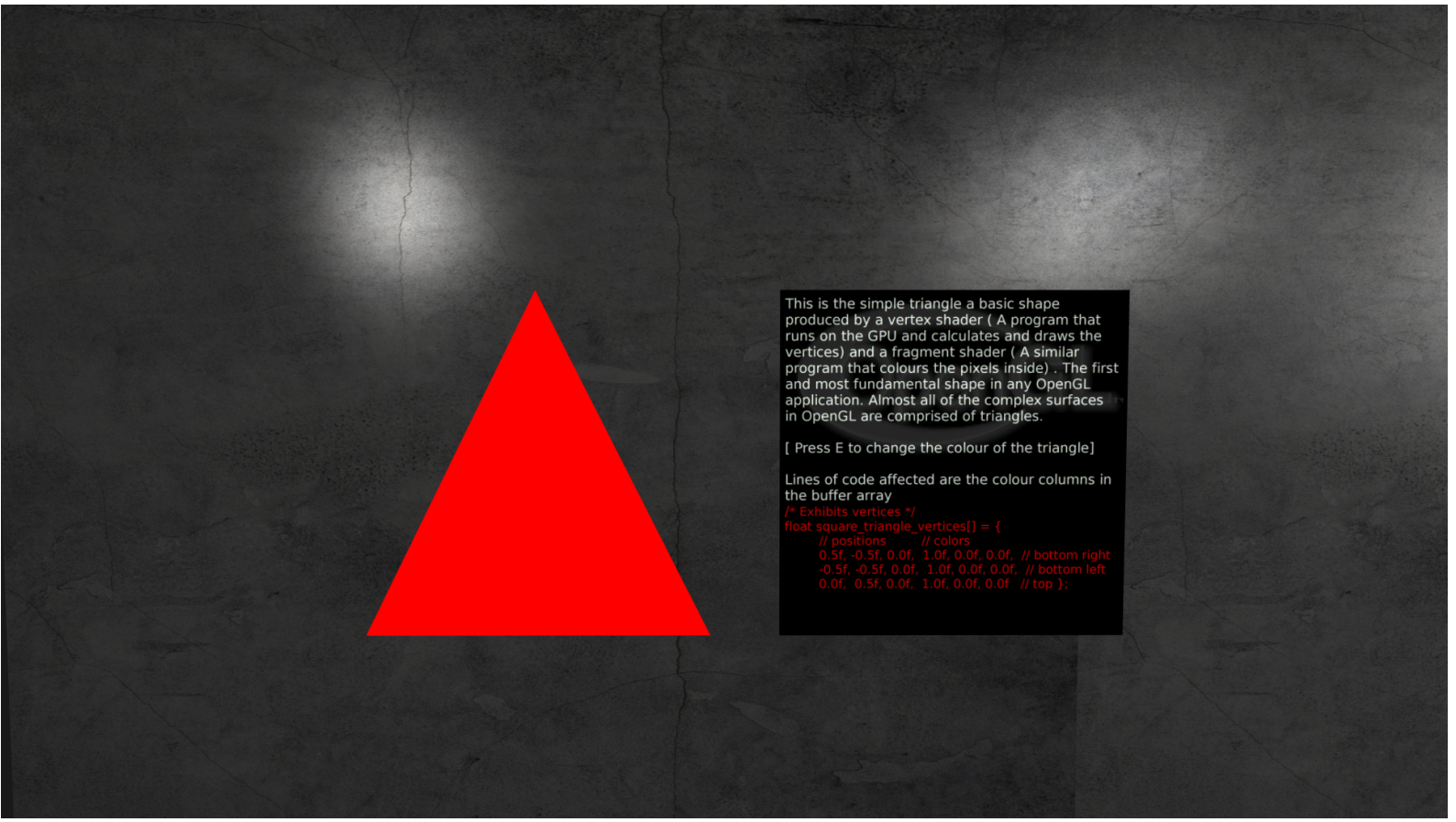
As mentioned in the previous paragraph the demo is made in order to be used by a user who has no knowledge of OpenGL as well as someone well versed in the field. That is why we chose the corridor approach as seen below.

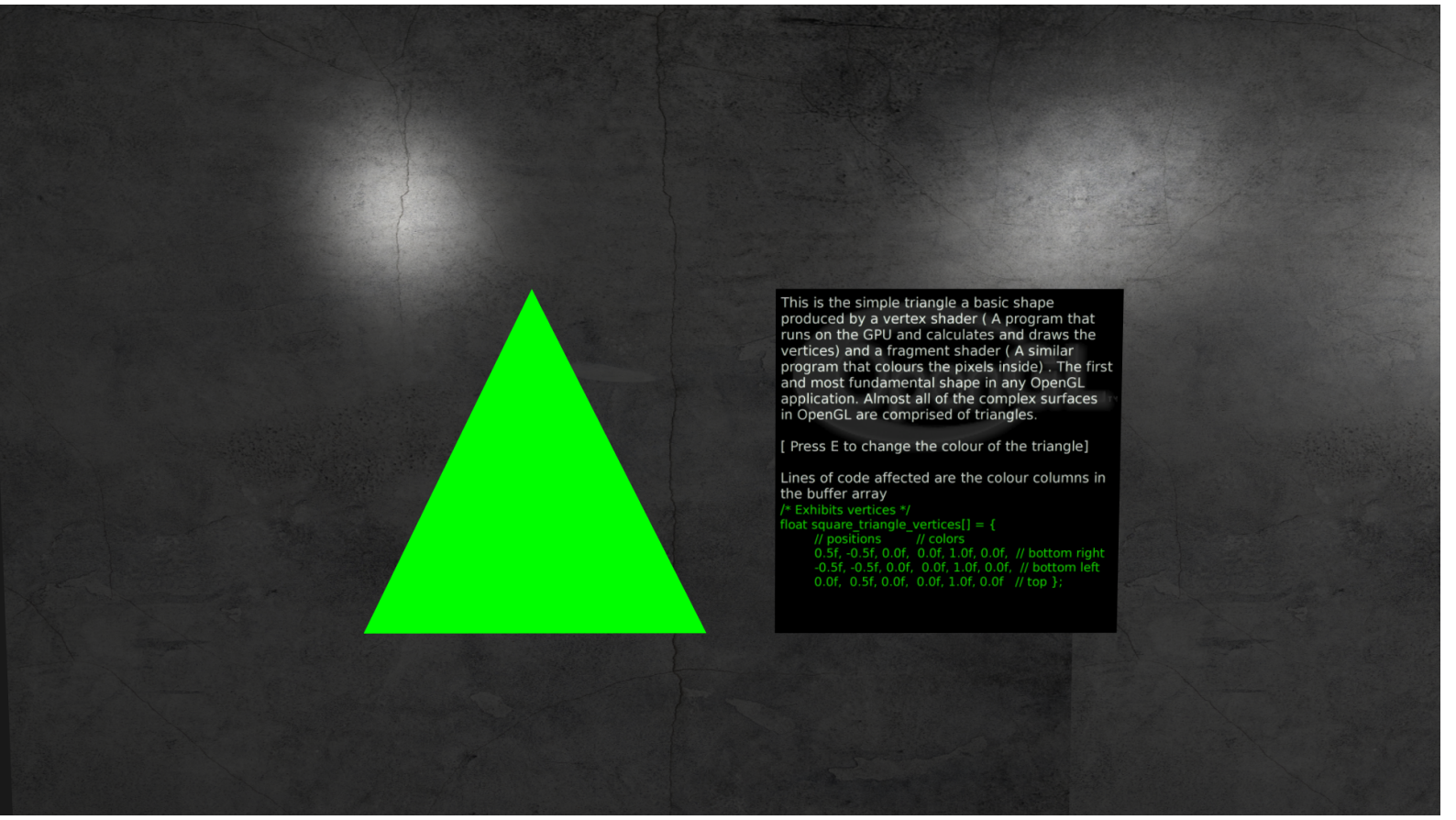
The user can fly in the corridor and observe in real-time the exhibits which display an OpenGL function, and either go in the source code to see how each one is implemented or read the exhibit description about what everything is in a more simplistic manner. The view is first-person, in order to give the user a more realistic feel that he or she is inside a real exhibition. Using the classic WSAD buttons, the user can move inside the corridor and browse the exhibits.

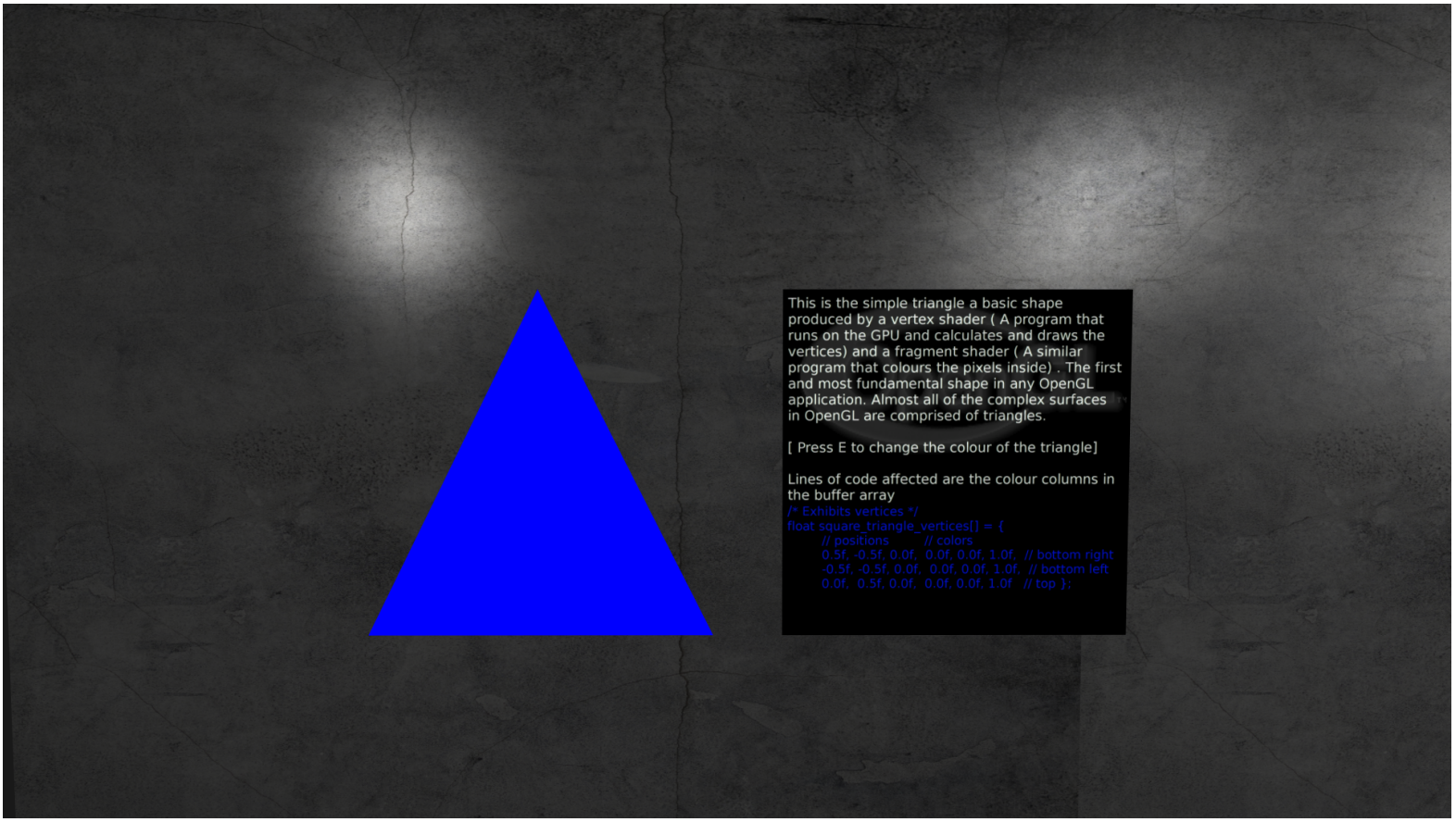
At this point, the exhibit’s explanations are filed with placeholder items in order to determine the correct dimensions for each explanation and exhibit placement.

From left to right and then back the exhibits display:

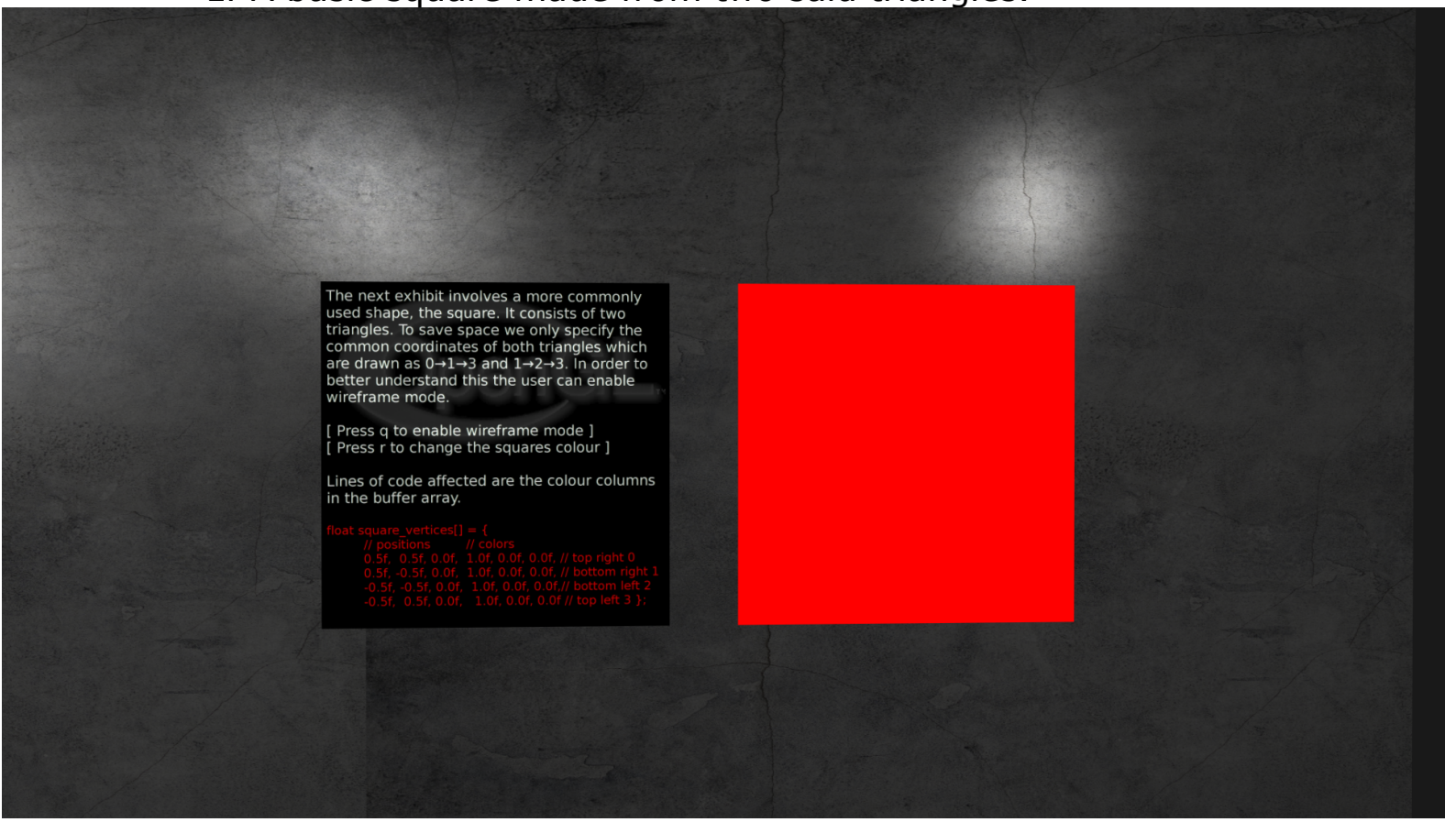
1. The basic triangle, which is used to create almost any other shape in OpenGL.







2. A basic square made from two said triangles.

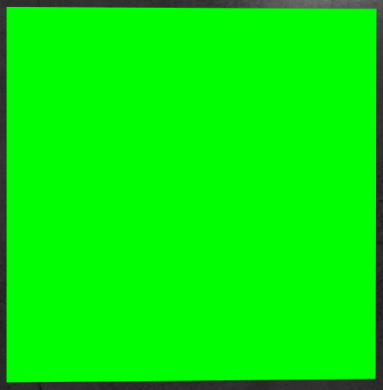


The next exhibit involves a more commonly used shape, the square. It consists of two triangles. To save space we only specify the common coordinates of both triangles which are drawn as 0→1→3 and 1→2→3. In order to better understand this the user can enable wireframe mode.

[Press q to enable wireframe mode]
[Press r to change the squares colour]

Lines of code affected are the colour columns in the buffer array.

```
float square_vertices[] = {
        // positions         // colors
        0.5f,  0.5f, 0.0f,  0.0f, 1.0f, 0.0f, // top right 0
        0.5f, -0.5f, 0.0f,  0.0f, 1.0f, 0.0f, // bottom right 1
        -0.5f, -0.5f, 0.0f,  0.0f, 1.0f, 0.0f,// bottom left 2
        -0.5f,  0.5f, 0.0f,   0.0f, 1.0f, 0.0f // top left 3 };
```

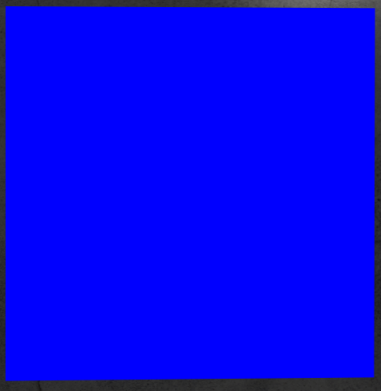


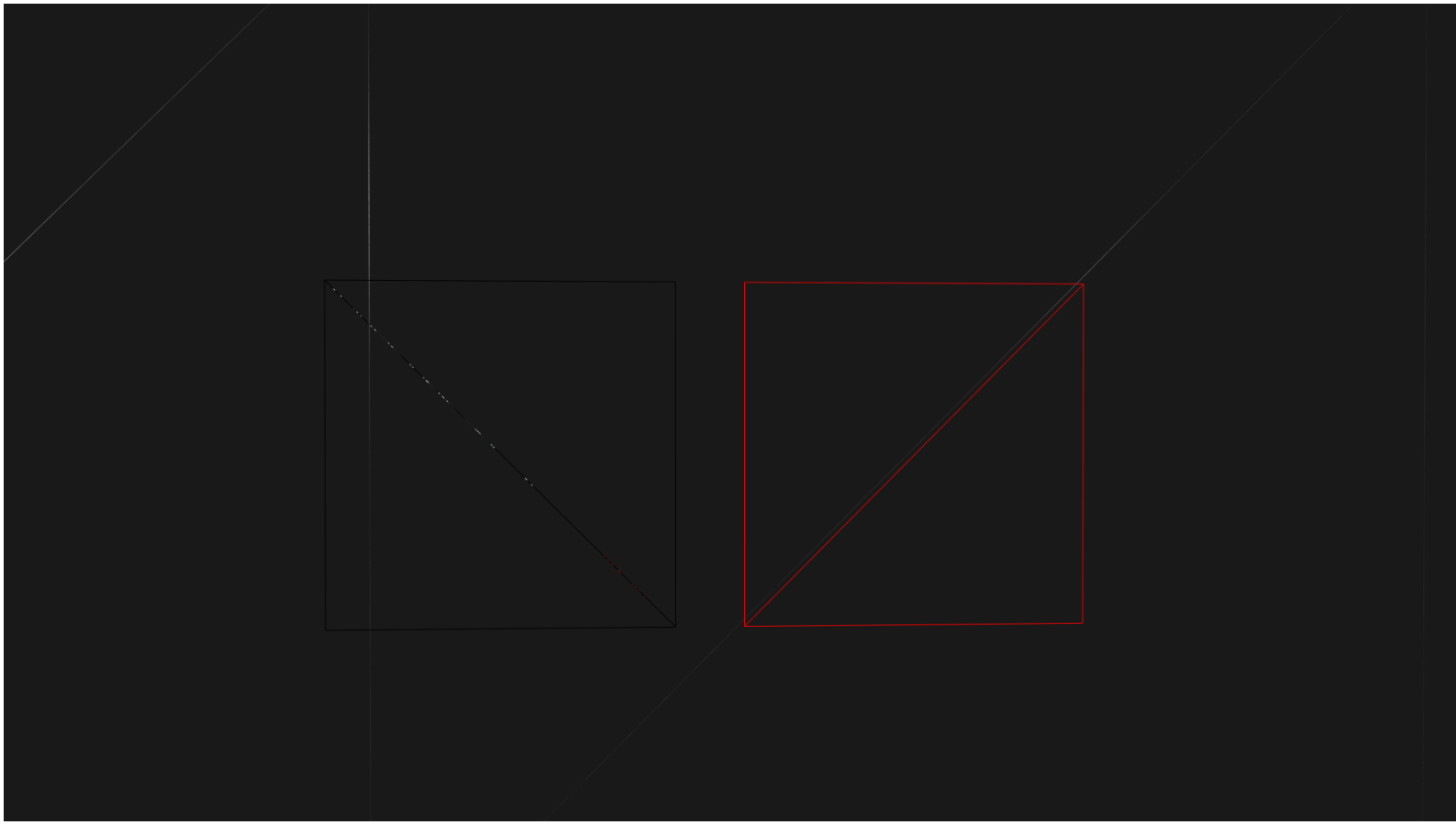
The next exhibit involves a more commonly used shape, the square. It consists of two triangles. To save space we only specify the common coordinates of both triangles which are drawn as 0→1→3 and 1→2→3. In order to better understand this the user can enable wireframe mode.

[Press q to enable wireframe mode]
[Press r to change the squares colour]

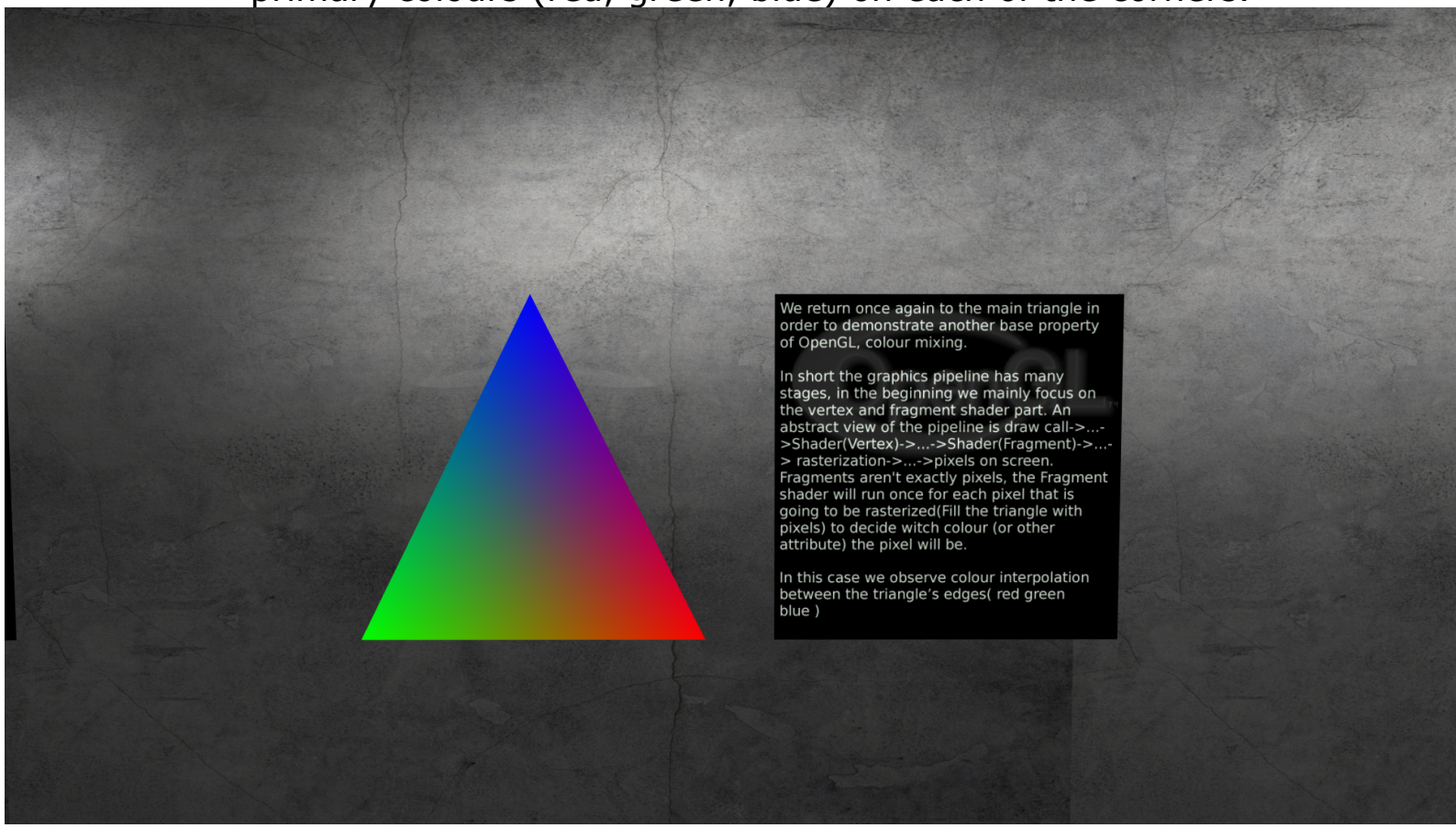
Lines of code affected are the colour columns in the buffer array.

```
float square_vertices[] = {
        // positions         // colors
        0.5f,  0.5f, 0.0f,  0.0f, 0.0f, 1.0f, // top right 0
        0.5f, -0.5f, 0.0f,  0.0f, 0.0f, 1.0f, // bottom right 1
        -0.5f, -0.5f, 0.0f,  0.0f, 0.0f, 1.0f,// bottom left 2
        -0.5f,  0.5f, 0.0f,   0.0f, 0.0f, 1.0f // top left 3 };
```

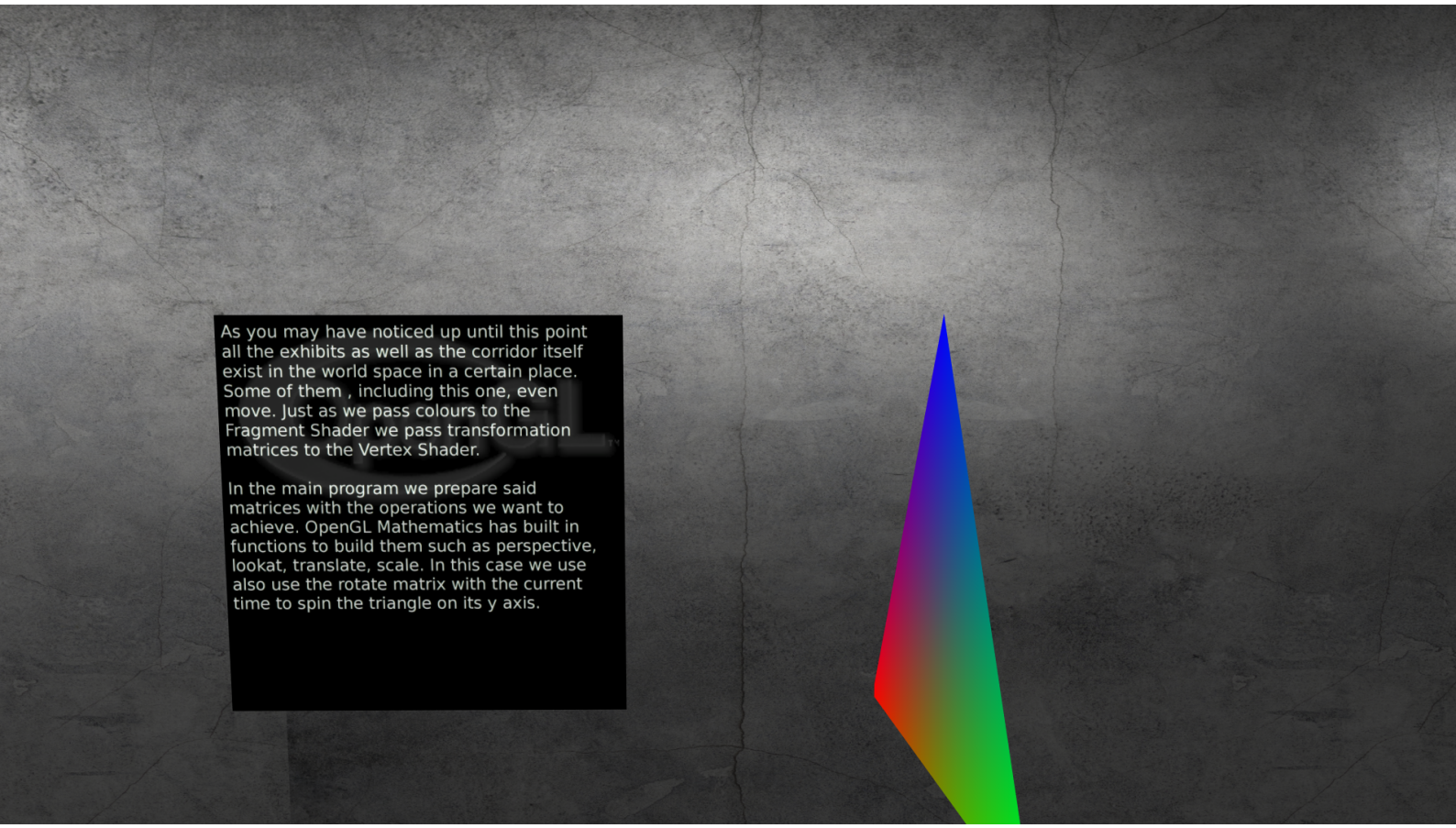




3. A basic triangle displaying colour interpolation from the primary colours (red, green, blue) on each of the corners.



4. The same triangle with applied transformations to make it rotate.



5. A square with a blend of two textures.

Texture loading in OpenGL is a relativity straightforward affair. We use a simple yet effective public domain image loader in C++ (stb_image – v2.23) to load the image, then using OpenGL functions we use texturing to allow the elements such as height, width etc to be read by the Shader. After that a minimap (a set of images with progressively reduced resolution to be used in far away objects in order to make the program more resource effective) is generated. The final step we pass several parameters to the texture such as how it should wrap around the object etc.

[Press t in order to change the texture]

Here the fragment Shader takes in two textures and displays them with an 80 % to 20% ratio, pressing t swaps them.

```
exhibit_cubeTextureShader.setInt("exhibit_5_texture_1", 0);
exhibit_cubeTextureShader.setInt("exhibit_5_texture_2", 1);
```

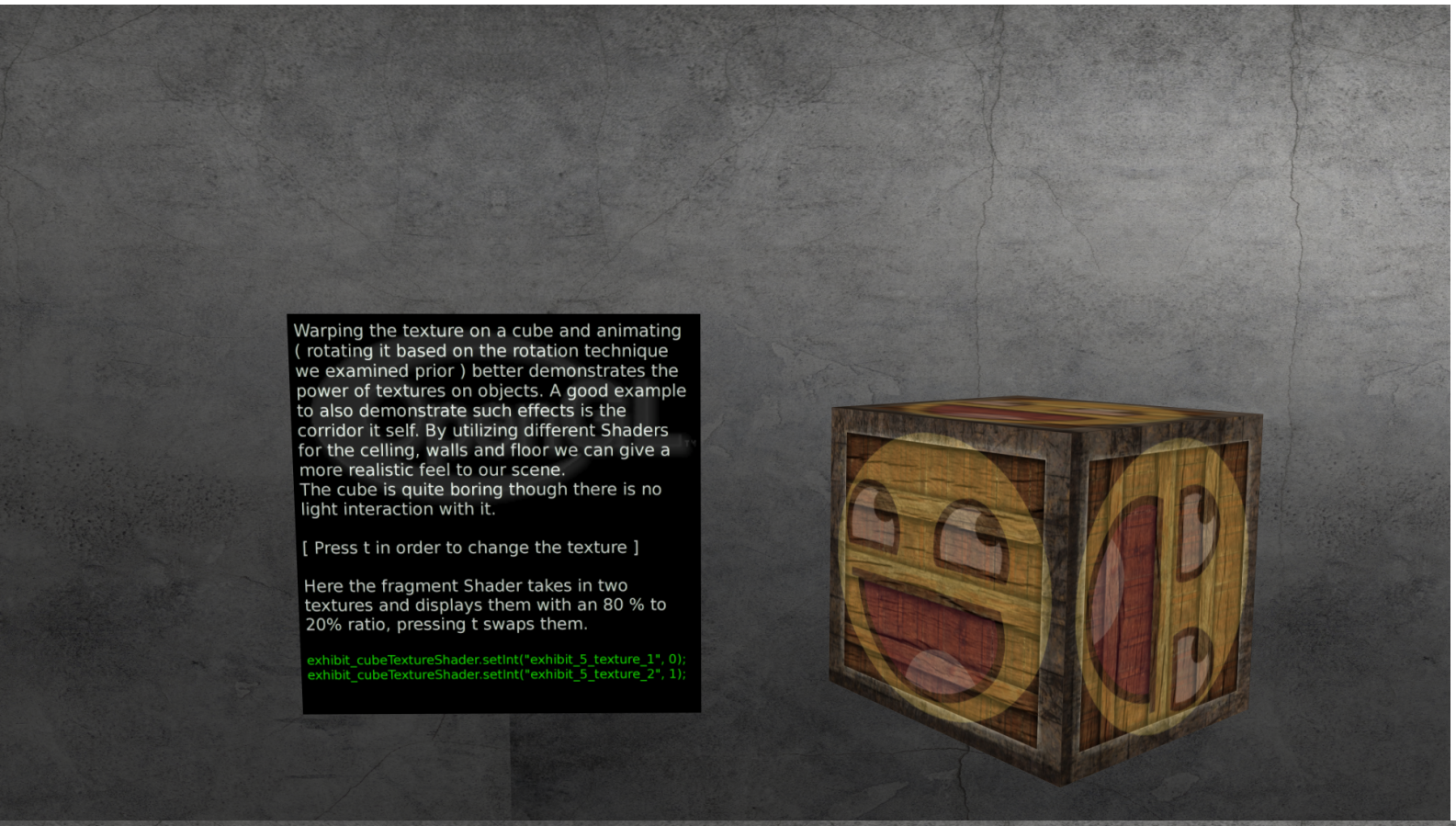
Texture loading in OpenGL is a relativity straightforward affair. We use a simple yet effective public domain image loader in C++ (stb_image – v2.23) to load the image, then using OpenGL functions we use texturing to allow the elements such as height, width etc to be read by the Shader. After that a minimap (a set of images with progressively reduced resolution to be used in far away objects in order to make the program more resource effective) is generated. The final step we pass several parameters to the texture such as how it should wrap around the object etc.

[Press t in order to change the texture]

Here the fragment Shader takes in two textures and displays them with an 80 % to 20% ratio, pressing t swaps them.

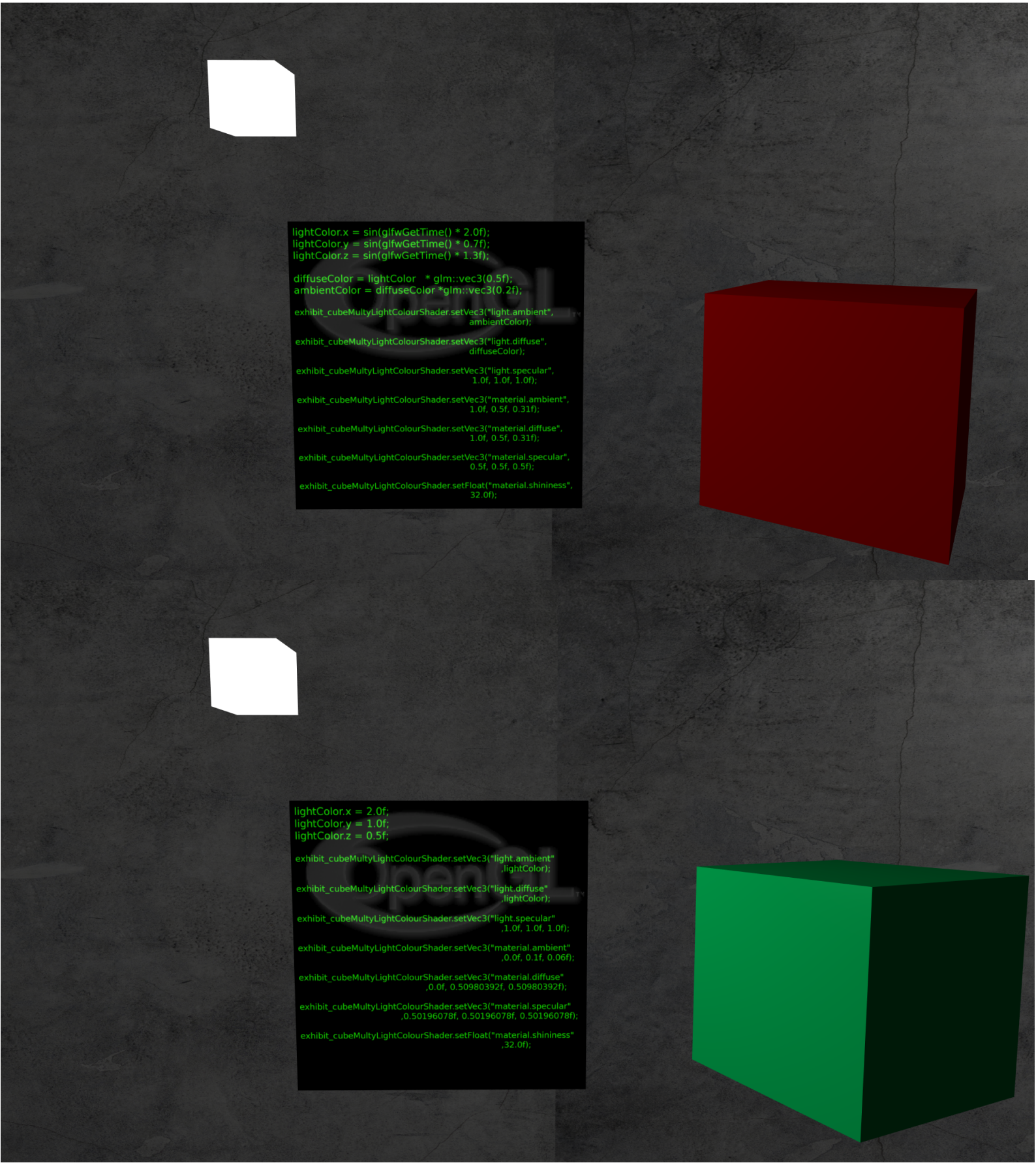
```
exhibit_cubeTextureShader.setInt("exhibit_5_texture_2", 0);
exhibit_cubeTextureShader.setInt("exhibit_5_texture_1", 1);
```

6. The same square with applied transformations to make it rotate in order to demonstrate the correct depth display. As the object rotates the back side remains in the back and it doesn’t ruin the front view.



7. A cube that reacts to lighting effects.

8. The same cube rotation in order to better showcase the effects.



Chapter 3: Future Development

Future development plans include making the scene look more realistic with the use of more detailed textures and a greater fine grain control of the lighting parameters. We could also implement more exhibits showing lighting and material properties, other transformations etc.

Even more ambitiously the corridor can be extended to many rooms with more user interaction with the exhibits instead of just displaying them.