Homework 4 - Transformation

陈曦 16340036

1.画一个立方体(cube)：边长为4， 中心位置为(0, 0, 0)。分别启动和关闭深度测试 glEnable(GL_DEPTH_TEST) 、 glDisable(GL_DEPTH_TEST) ，查看区别，并分析原因。

- **顶点着色器**

```
//GLSL顶点着色器
const char *vertexShaderSource = "#version 330 core\n"
"layout(location = 0) in vec3 aPos;\n"
"layout (location = 1) in vec3 aColor;\n"
"out vec3 outColor;\n"
"uniform mat4 model;\n"
"uniform mat4 view;\n"
"uniform mat4 projection;\n"
"void main()\n"
"{\n"
"   gl_Position = projection * view * model * vec4(aPos, 1.0);\n"
"   outColor = aColor;\n"
"}\0";
```

- **片段着色器**

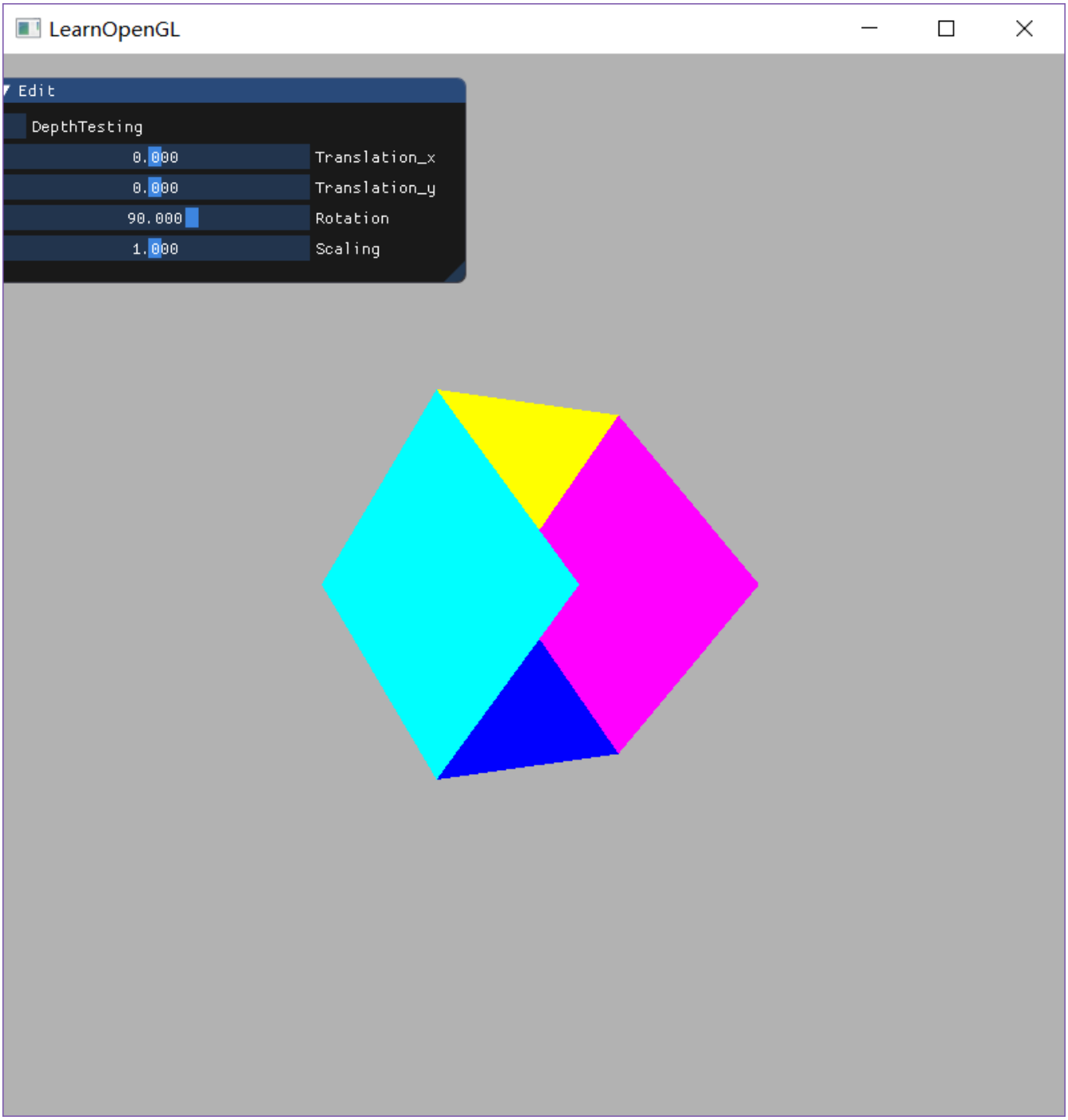
```
//片段着色器
const char *fragmentShaderSource = "#version 330 core\n"
"out vec4 FragColor;\n"
"in vec3 outColor;\n"
"void main()\n"
"{\n"
"   FragColor = vec4(outColor, 1.0);\n"
"}\n\0";
```

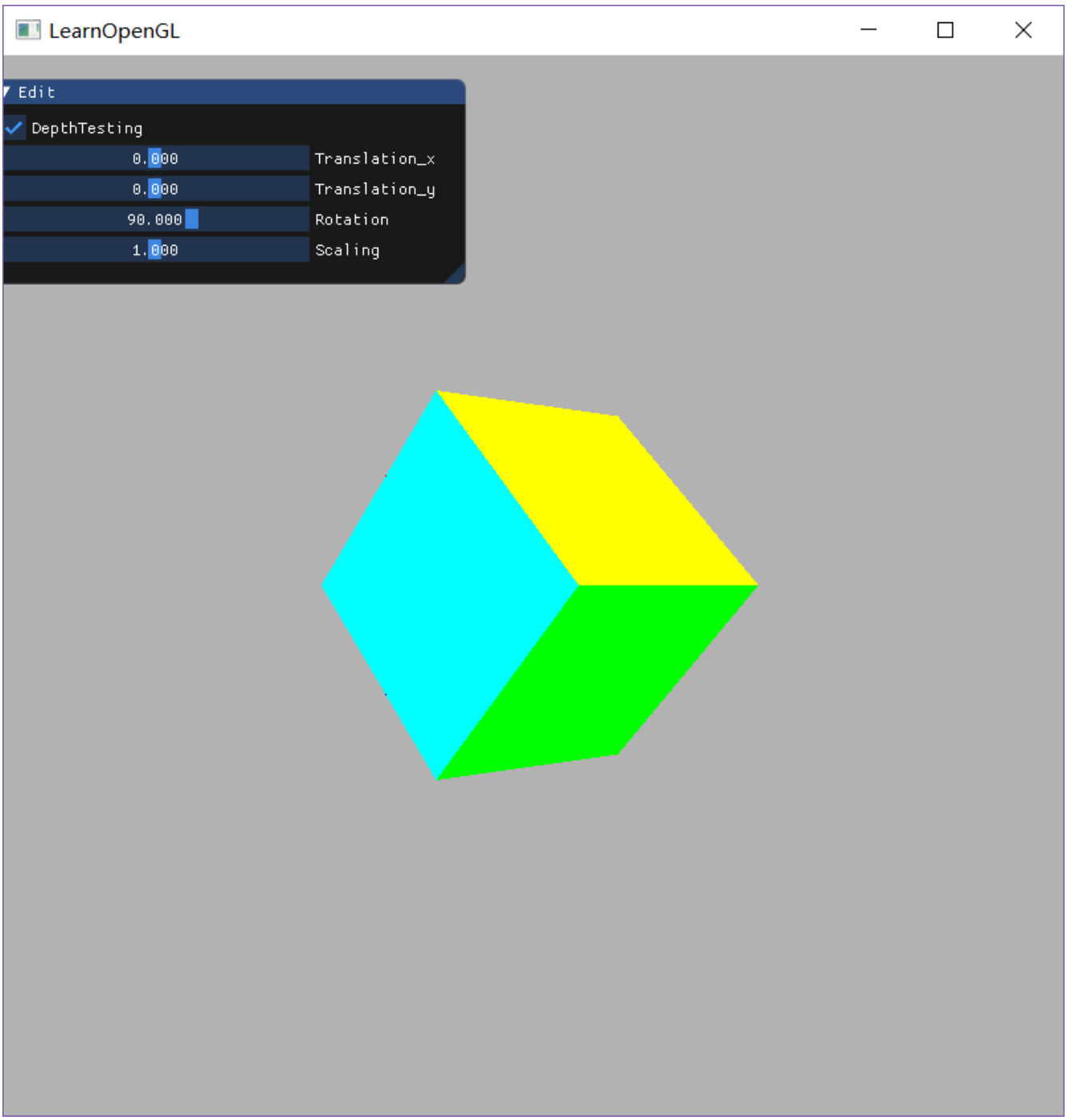
绘制6*2个三角形最终拼接成一个立方体，并将每一面填充一个颜色

```
//多个三角形最终拼接成一个立方体（6*2）
float vertices[] = {
        -2.0f, -2.0f, -2.0f, 1.0f, 0.0f, 0.0f, //triangle 1 : begin
         2.0f, -2.0f, -2.0f, 1.0f, 0.0f, 0.0f,
         2.0f,  2.0f, -2.0f, 1.0f, 0.0f, 0.0f, //triangle 1 : end
         2.0f,  2.0f, -2.0f, 1.0f, 0.0f, 0.0f, // triangle 2 : begin
        -2.0f,  2.0f, -2.0f, 1.0f, 0.0f, 0.0f,
        -2.0f, -2.0f, -2.0f, 1.0f, 0.0f, 0.0f, // triangle 2 : end

        -2.0f, -2.0f,  2.0f, 0.0f, 1.0f, 0.0f,
         2.0f, -2.0f,  2.0f, 0.0f, 1.0f, 0.0f,
         2.0f,  2.0f,  2.0f, 0.0f, 1.0f, 0.0f,
         2.0f,  2.0f,  2.0f, 0.0f, 1.0f, 0.0f,
        -2.0f,  2.0f,  2.0f, 0.0f, 1.0f, 0.0f,
        -2.0f, -2.0f,  2.0f, 0.0f, 1.0f, 0.0f,

        -2.0f,  2.0f,  2.0f, 0.0f, 0.0f, 1.0f,
        -2.0f,  2.0f, -2.0f, 0.0f, 0.0f, 1.0f,
```

```
    -2.0f, -2.0f, -2.0f, 0.0f, 0.0f, 1.0f,
    -2.0f, -2.0f, -2.0f, 0.0f, 0.0f, 1.0f,
    -2.0f, -2.0f,  2.0f, 0.0f, 0.0f, 1.0f,
    -2.0f,  2.0f,  2.0f, 0.0f, 0.0f, 1.0f,

     2.0f,  2.0f,  2.0f, 1.0f, 1.0f, 0.0f,
     2.0f,  2.0f, -2.0f, 1.0f, 1.0f, 0.0f,
     2.0f, -2.0f, -2.0f, 1.0f, 1.0f, 0.0f,
     2.0f, -2.0f, -2.0f, 1.0f, 1.0f, 0.0f,
     2.0f, -2.0f,  2.0f, 1.0f, 1.0f, 0.0f,
     2.0f,  2.0f,  2.0f, 1.0f, 1.0f, 0.0f,

    -2.0f, -2.0f, -2.0f, 1.0f, 0.0f, 1.0f,
     2.0f, -2.0f, -2.0f, 1.0f, 0.0f, 1.0f,
     2.0f, -2.0f,  2.0f, 1.0f, 0.0f, 1.0f,
     2.0f, -2.0f,  2.0f, 1.0f, 0.0f, 1.0f,
    -2.0f, -2.0f,  2.0f, 1.0f, 0.0f, 1.0f,
    -2.0f, -2.0f, -2.0f, 1.0f, 0.0f, 1.0f,

    -2.0f,  2.0f, -2.0f, 0.0f, 1.0f, 1.0f,
     2.0f,  2.0f, -2.0f, 0.0f, 1.0f, 1.0f,
     2.0f,  2.0f,  2.0f, 0.0f, 1.0f, 1.0f,
     2.0f,  2.0f,  2.0f, 0.0f, 1.0f, 1.0f,
    -2.0f,  2.0f,  2.0f, 0.0f, 1.0f, 1.0f,
    -2.0f,  2.0f, -2.0f, 0.0f, 1.0f, 1.0f
};
```

下图分别为关闭和开启深度测试


LearnOpenGL
Edit
DepthTesting
0.000
Translation_x
0.000
Translation_y
90.000
Rotation
1.000
Scaling

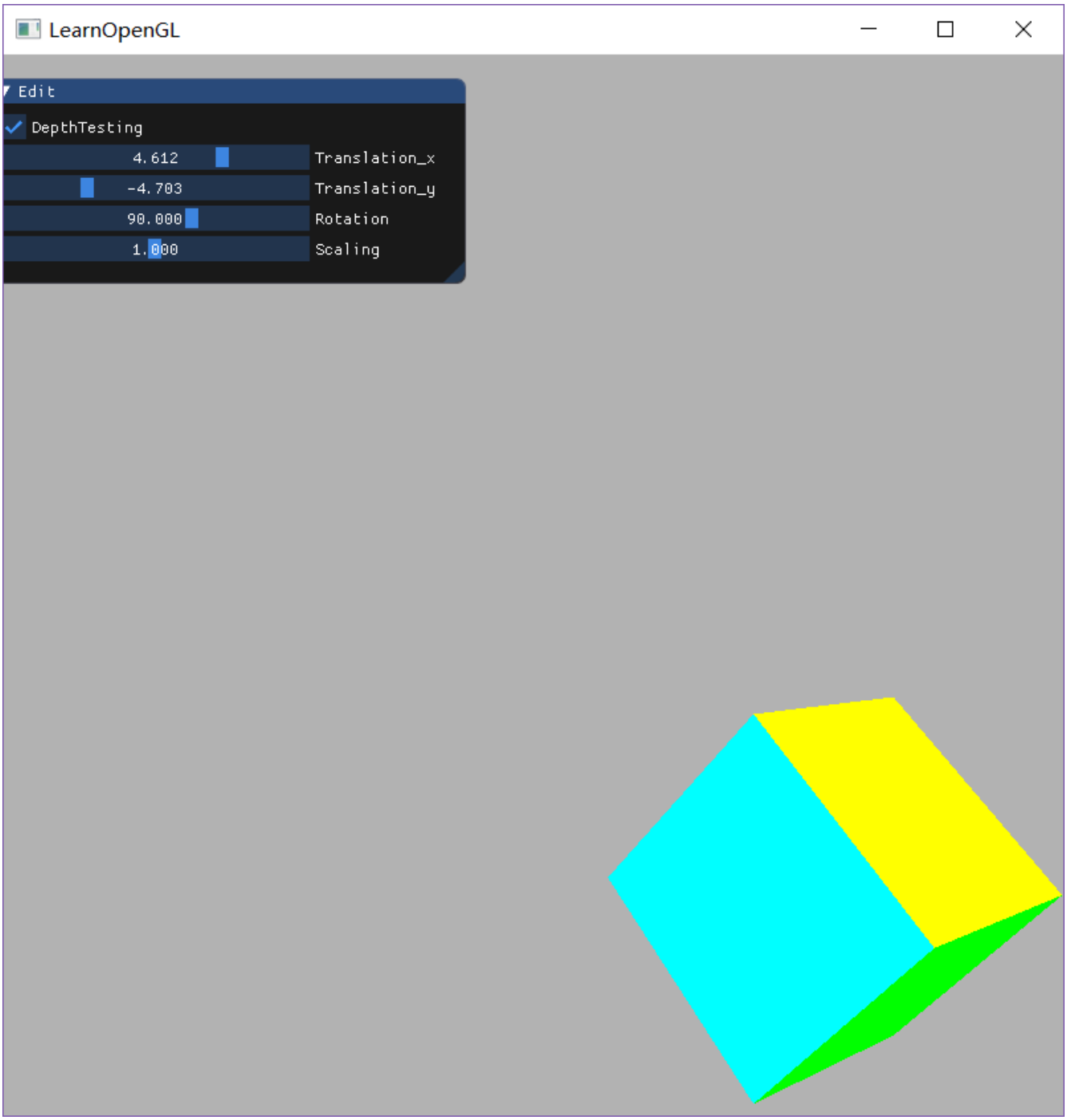


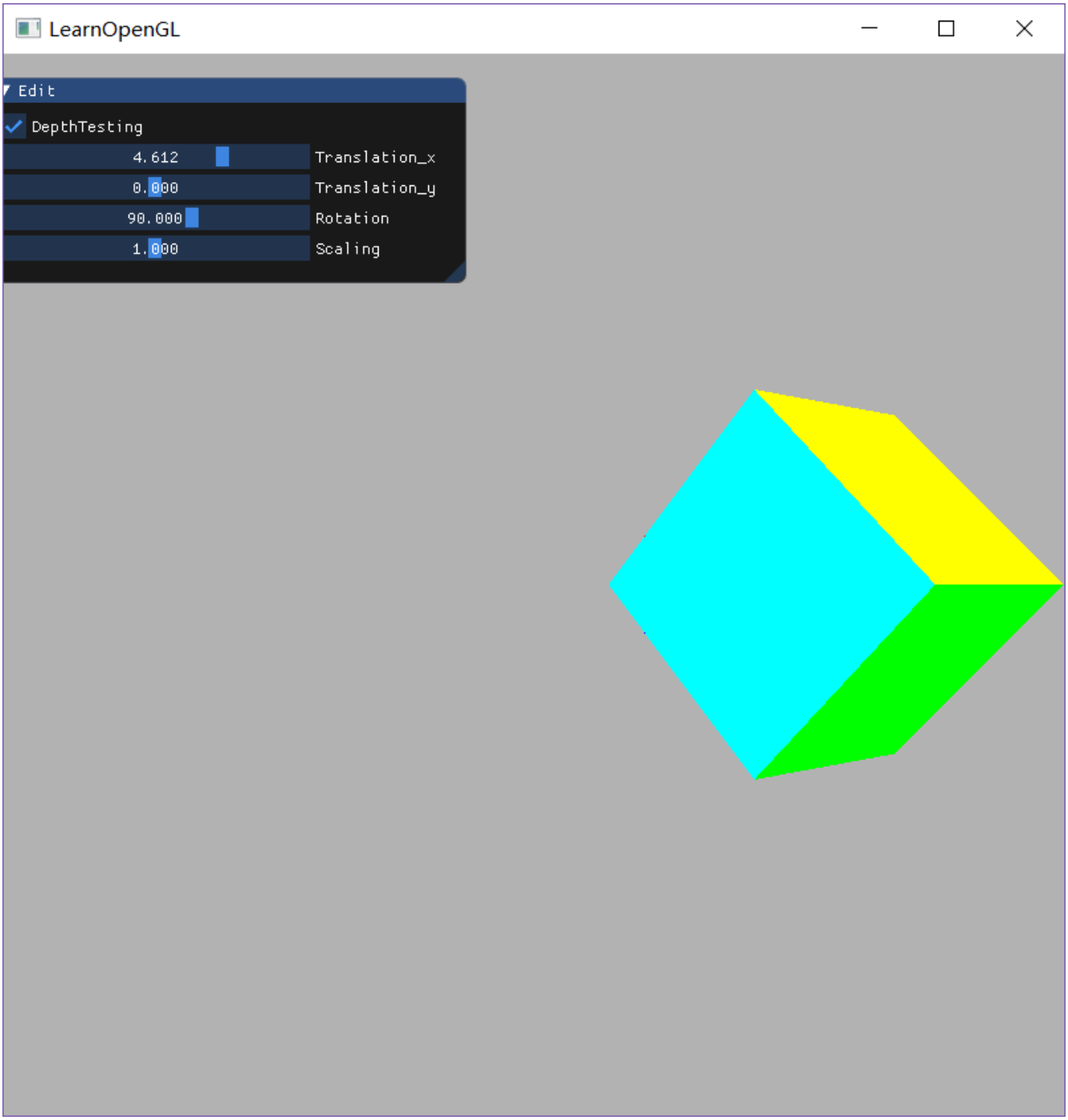
分析：立方体的某些本应被遮挡住的面被绘制在了这个立方体其他面之上，出现上面两图的原因是OpenGL是一个一个三角形来绘制立方体的，有些三角形本应被遮挡却被绘制到了其他三角形上面，所以当关闭深度测试时出现了第一张图的情况

- **Z缓冲**：OpenGL存储它的所有深度信息于一个Z缓冲(Z-buffer)中，也被称为深度缓冲(Depth Buffer)。GLFW会自动为你生成这样一个缓冲（就像它也有一个颜色缓冲来存储输出图像的颜色）。深度值存储在每个片段里面（作为片段的**z**值），当片段想要输出它的颜色时，OpenGL会将它的深度值和z缓冲进行比较，如果当前的片段在其它片段之后，它将会被丢弃，否则将会覆盖。这个过程称为深度测试(Depth Testing)，它是由OpenGL自动完成的。

2.平移(Translation)：使画好的cube沿着水平或垂直方向来回移动。

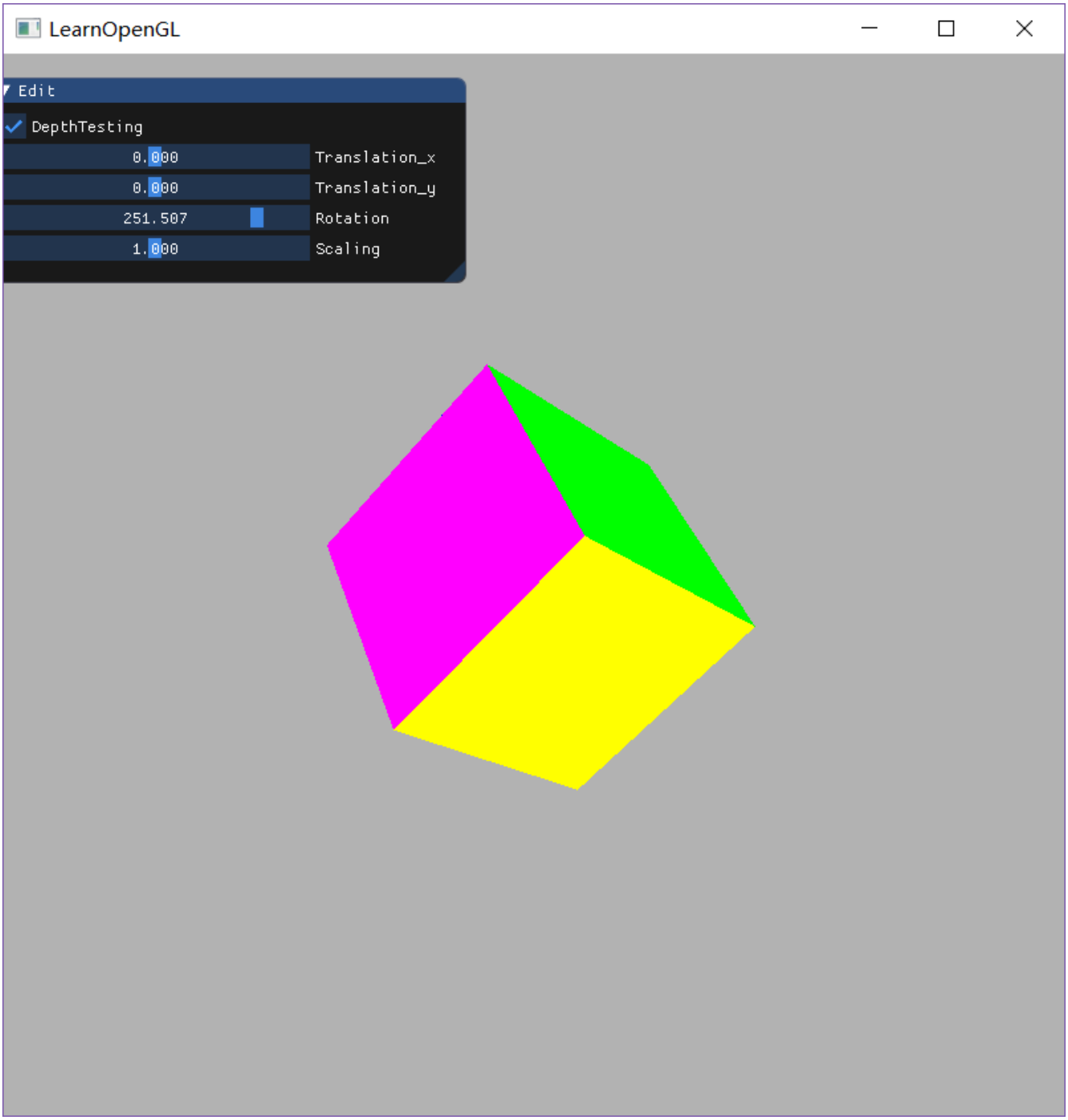
```
mat4 view = mat4(1.0f);
mat4 projection = mat4(1.0f);
view = translate(view, vec3(0.0f, 0.0f, -20.0f));
projection = perspective(radians(45.0f), 800.0f / 800.0f, 0.1f, 100.0f);

//使画好的cube沿着水平或垂直方向来回移动
model = translate(model, vec3(x, y, 0.0f));
```





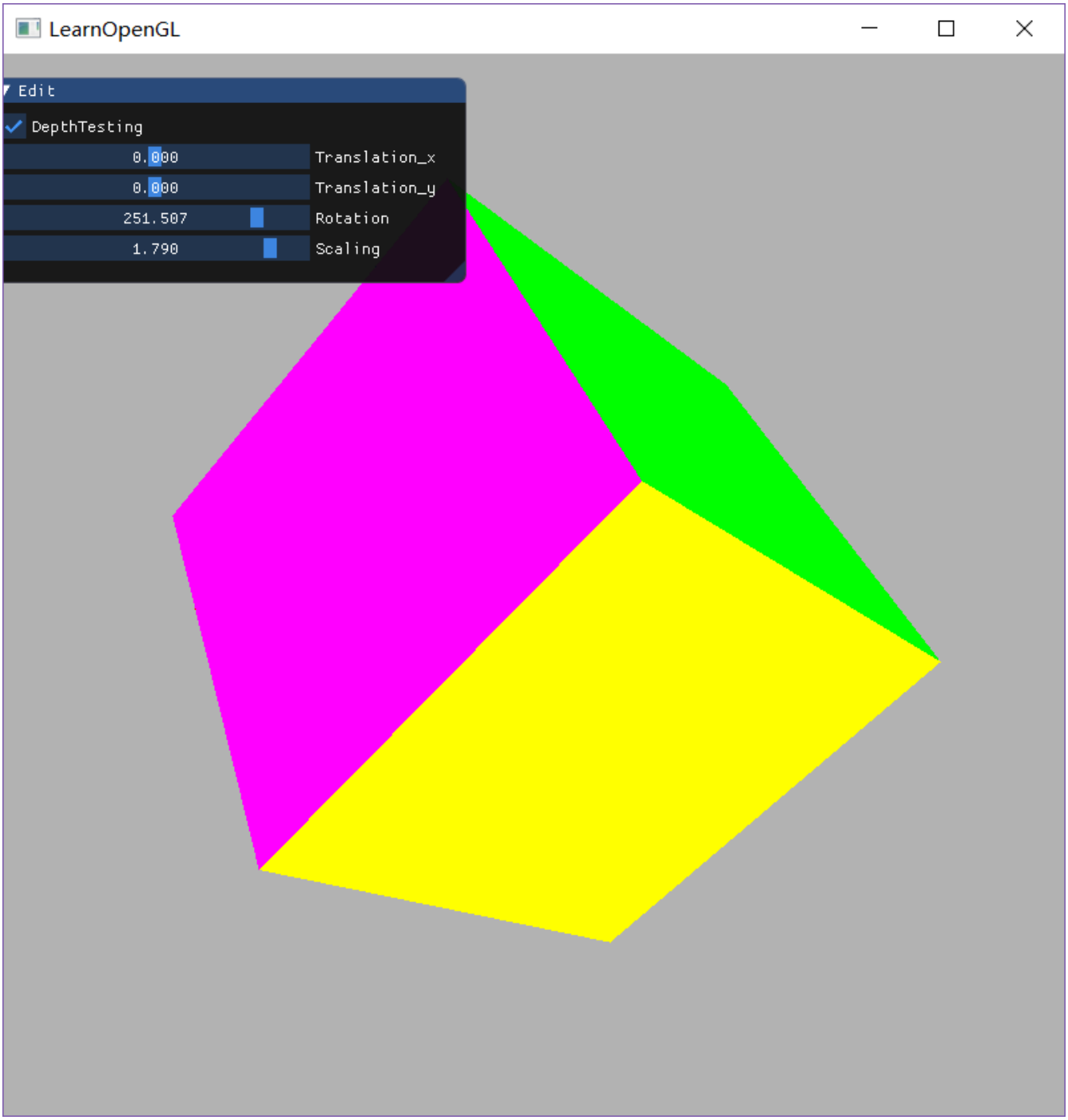
3.旋转(Rotation)：使画好的cube沿着XoZ平面的x=z轴持续旋转。

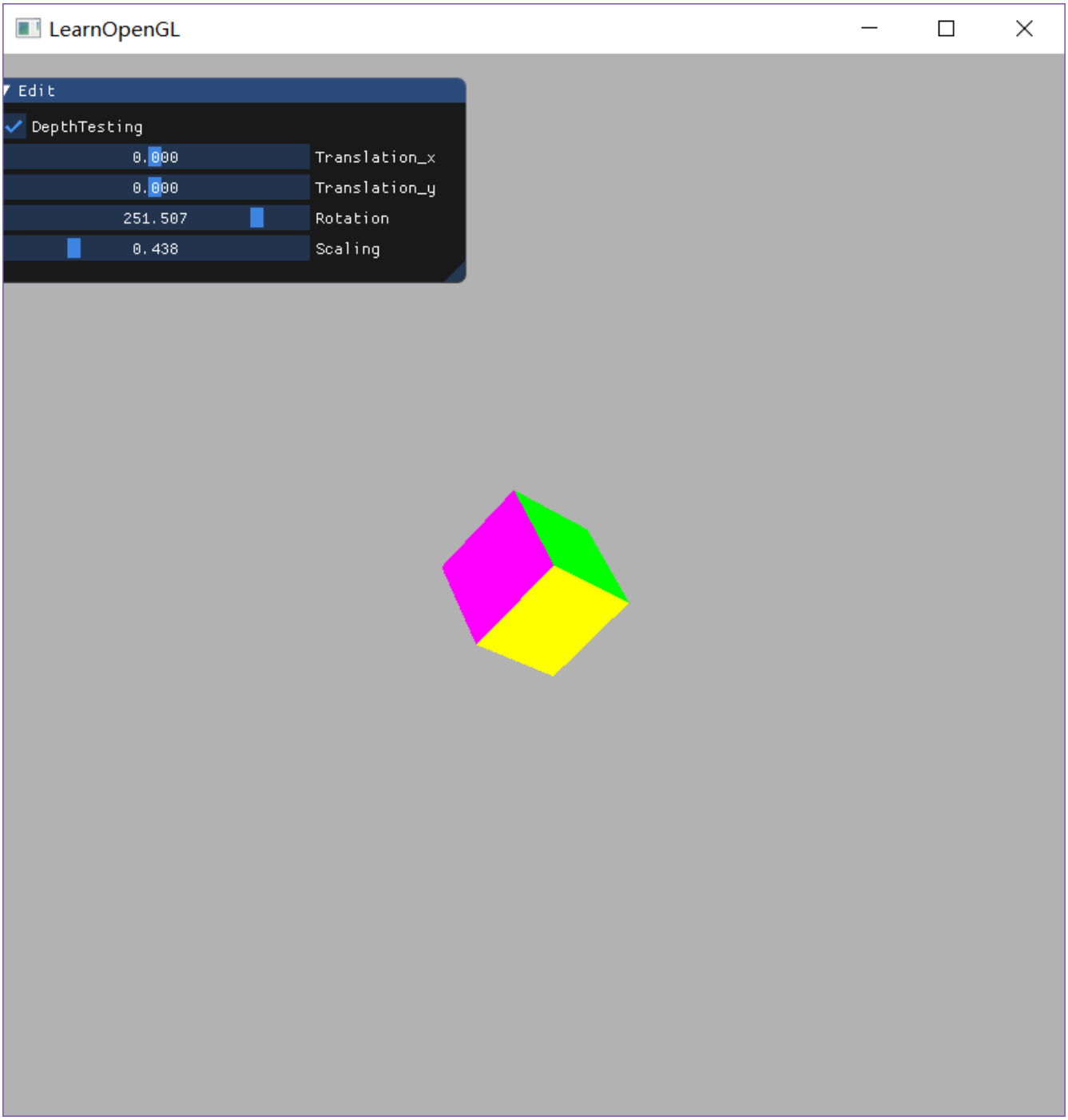
```
//使画好的cube沿着XoZ平面的x=z轴持续旋转
model = rotate(model, radians(angle), vec3(1.0f, 0.0f, 1.0f));
```



4.放缩(Scaling)：使画好的cube持续放大缩小。

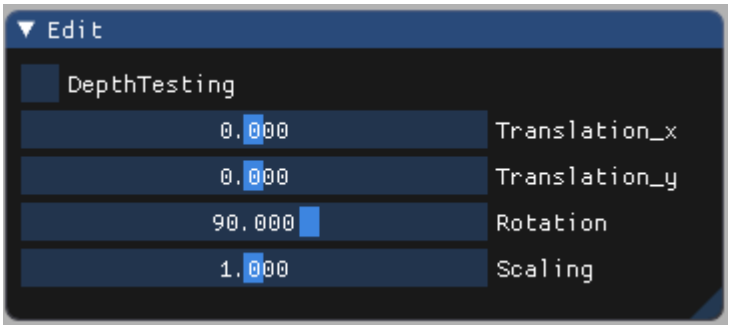
```
//使画好的cube持续放大缩小
model = glm::scale(model, vec3(scale, scale, scale));
```


LearnOpenGL
Edit
DepthTesting
0.000
Translation_x
0.000
Translation_y
251.507
Rotation
1.790
Scaling



5.在GUI里添加菜单栏，可以选择各种变换。

```
//创建ImGui
ImGui_ImplOpenGL3_NewFrame();
ImGui_ImplGlfw_NewFrame();
NewFrame();

Begin("Edit");
Checkbox("DepthTest", &depth);
SliderFloat("Translation_x", &x, -10, 10);
SliderFloat("Translation_y", &y, -10, 10);
SliderFloat("Rotation", &angle, -360.0f, 360.0f);
SliderFloat("Scaling", &scale, 0, 2.0f);
End();
```



6.结合Shader谈谈对渲染管线的理解

图形渲染管线可以被划分为几个高度专门化的阶段，很容易并行执行，大多数显卡都有成千上万的小处理核心，在GPU上为每一个阶段运行各自的小程序即Shader，从而在图形渲染管线中快速处理数据。

渲染管线步骤：

1.Vertex Processor：把一个单独的顶点作为输入

2.Primitive Assembly：图元配置，将顶点着色器输出的所有顶点作为输入，并将所有的点装配成指定图元的形状

3.Geometry Processor：把图元形式的一系列顶点的集合作为输入，可以通过产生新顶点构造出新的或是其它的图元来生成其他形状

4.Clipper：裁切

5.Rasterizer：光栅化，把图元映射为最终屏幕上相应的像素

6.Fragment Shader:计算一个像素的最终颜色

编写Shader程序过程：

1.声明Shader对象

2.指定Shader源代码

3.编译Shader

4.将Shader对象与程序对象绑定

5.链接

6.释放中间Shader对象