HW4 实验报告

实验要求

Basic:

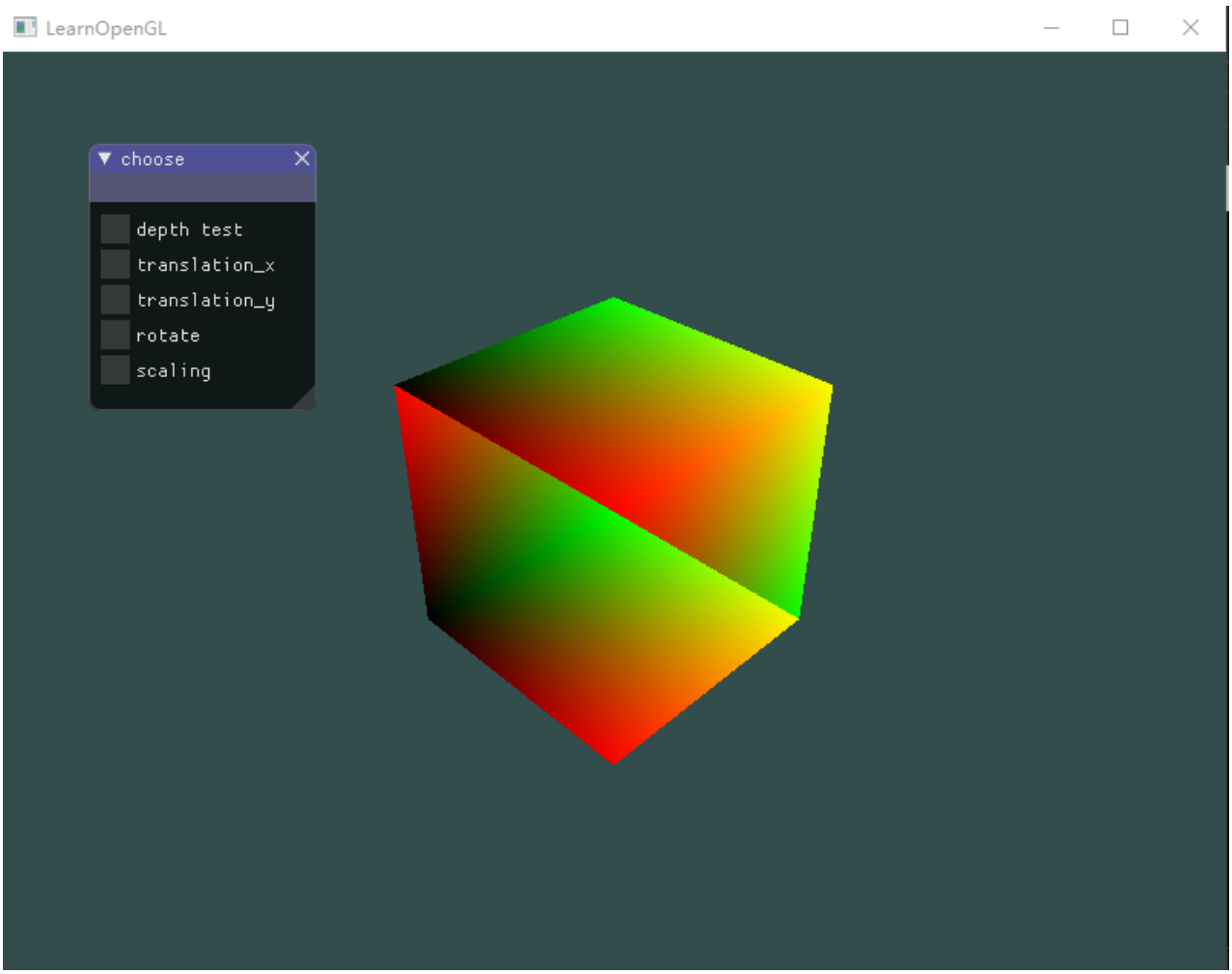
1. 画一个立方体(cube)：边长为4， 中心位置为(0, 0, 0)。分别启动和关闭深度测试 glEnable(GL_DEPTH_TEST) 、glDisable(GL_DEPTH_TEST) ，查看区别，并分析原因。
2. 平移(Translation)：使画好的cube沿着水平或垂直方向来回移动。
3. 旋转(Rotation)：使画好的cube沿着XoZ平面的x=z轴持续旋转。
4. 放缩(Scaling)：使画好的cube持续放大缩小。
5. 在GUI里添加菜单栏，可以选择各种变换。
6. 结合Shader谈谈对渲染管线的理解 Hint: 可以使用GLFW时间函数 glfwGetTime() ，或者 、 等获取不同的数值

Bonus:

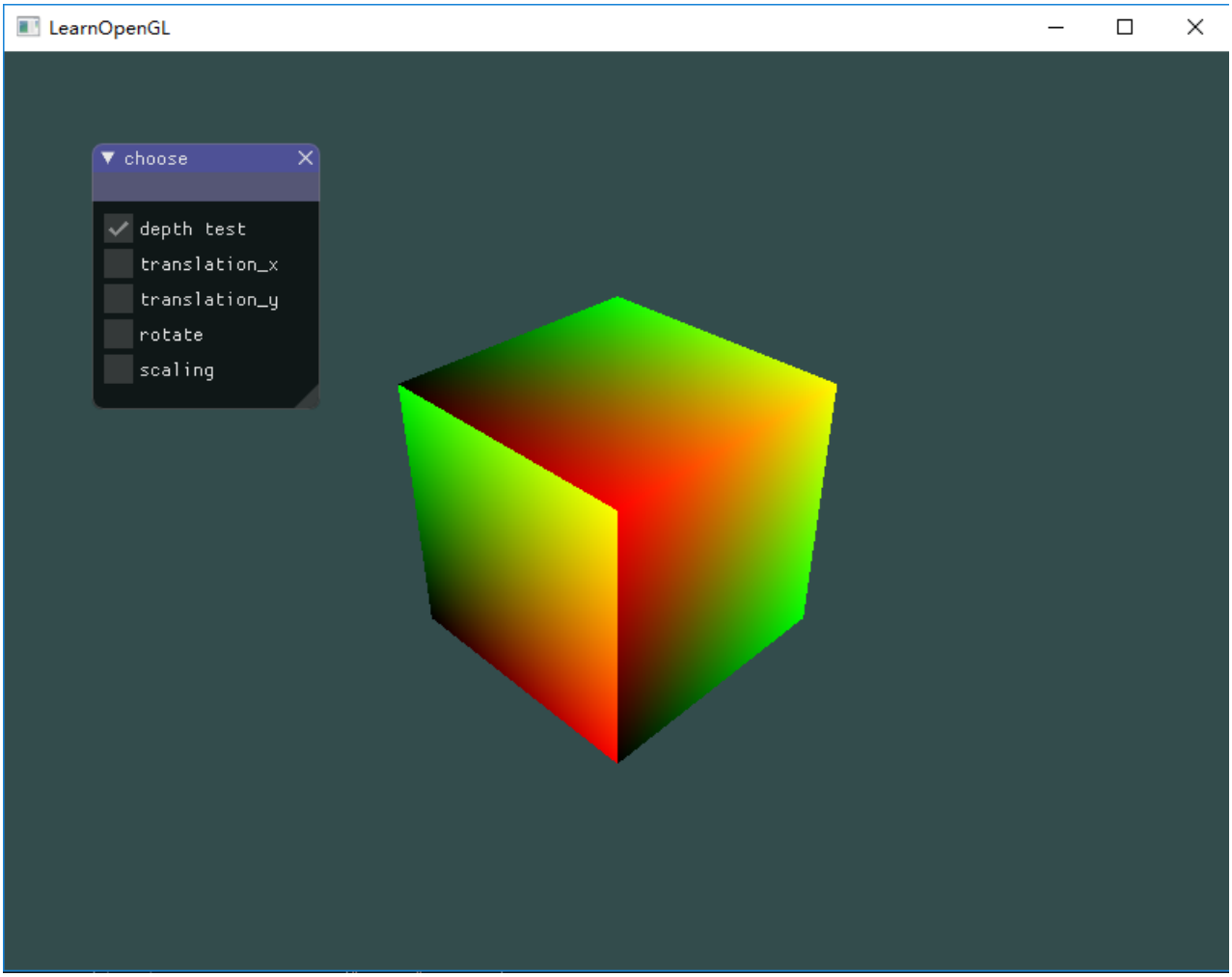
1. 将以上三种变换相结合，打开你们的脑洞，实现有创意的动画。比如：地球绕太阳转等。

实验截图

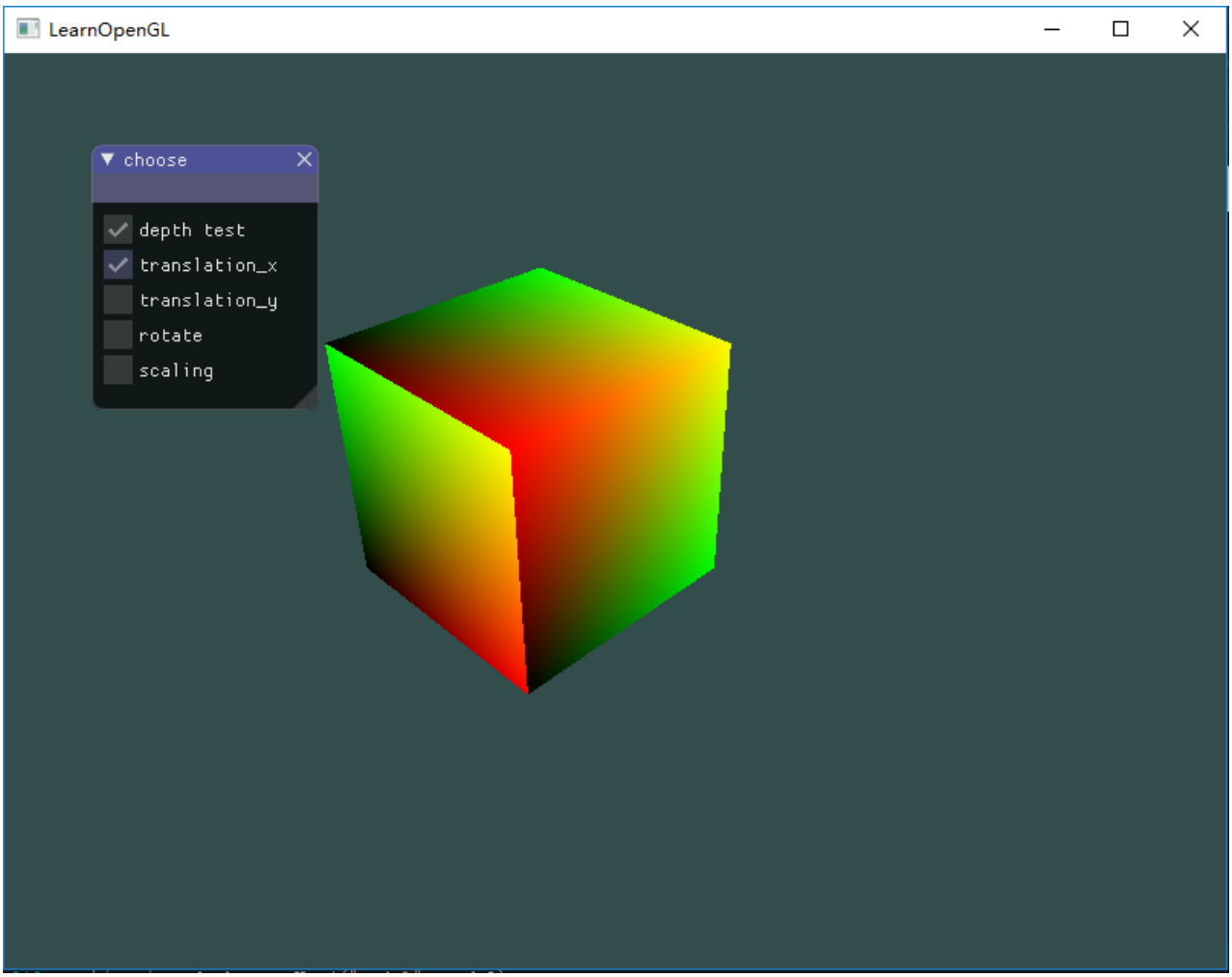
1. 画立方体，边长为4，中心位置为（0，0，0），关闭深度测试：

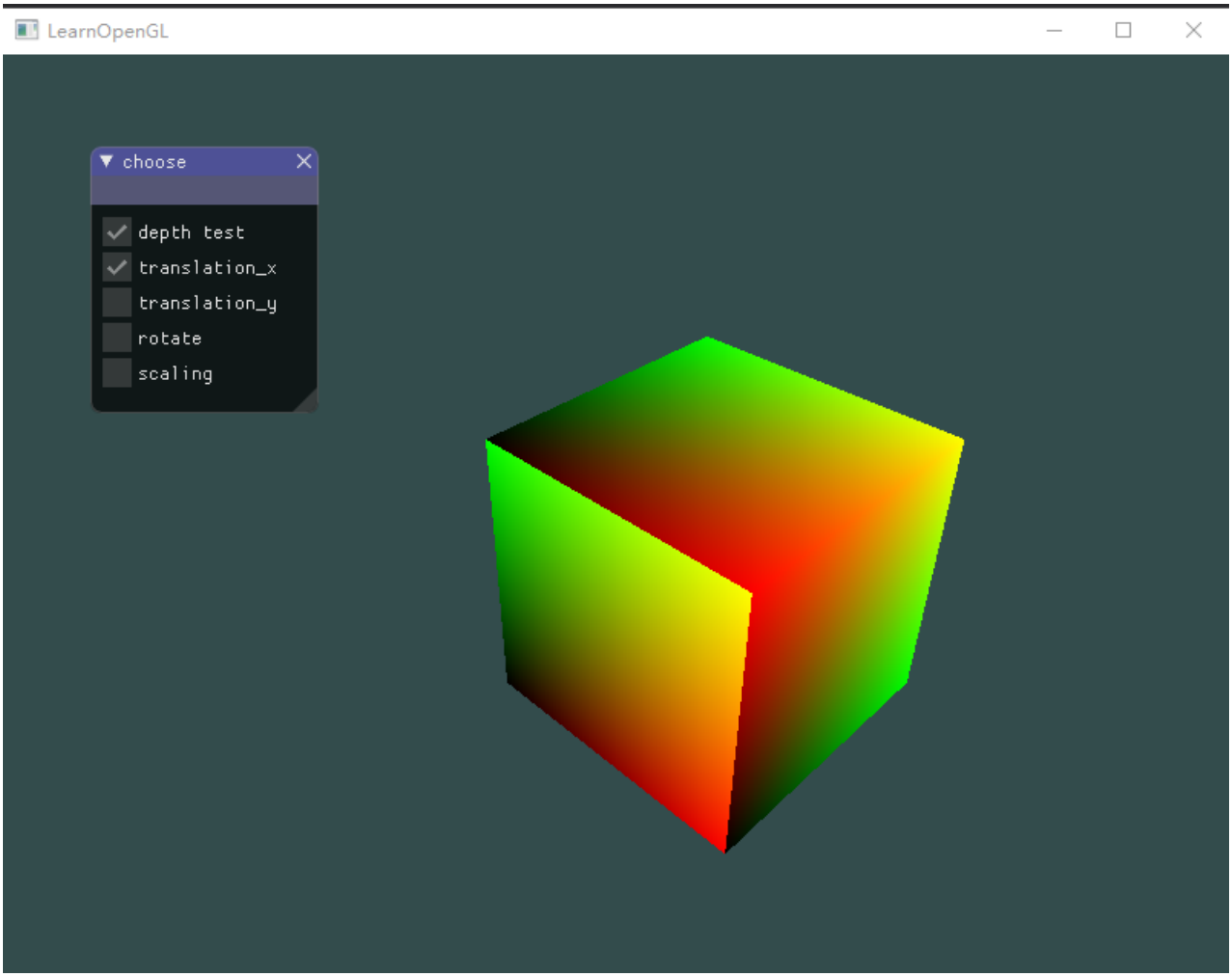


2. 立方体开启深度测试：

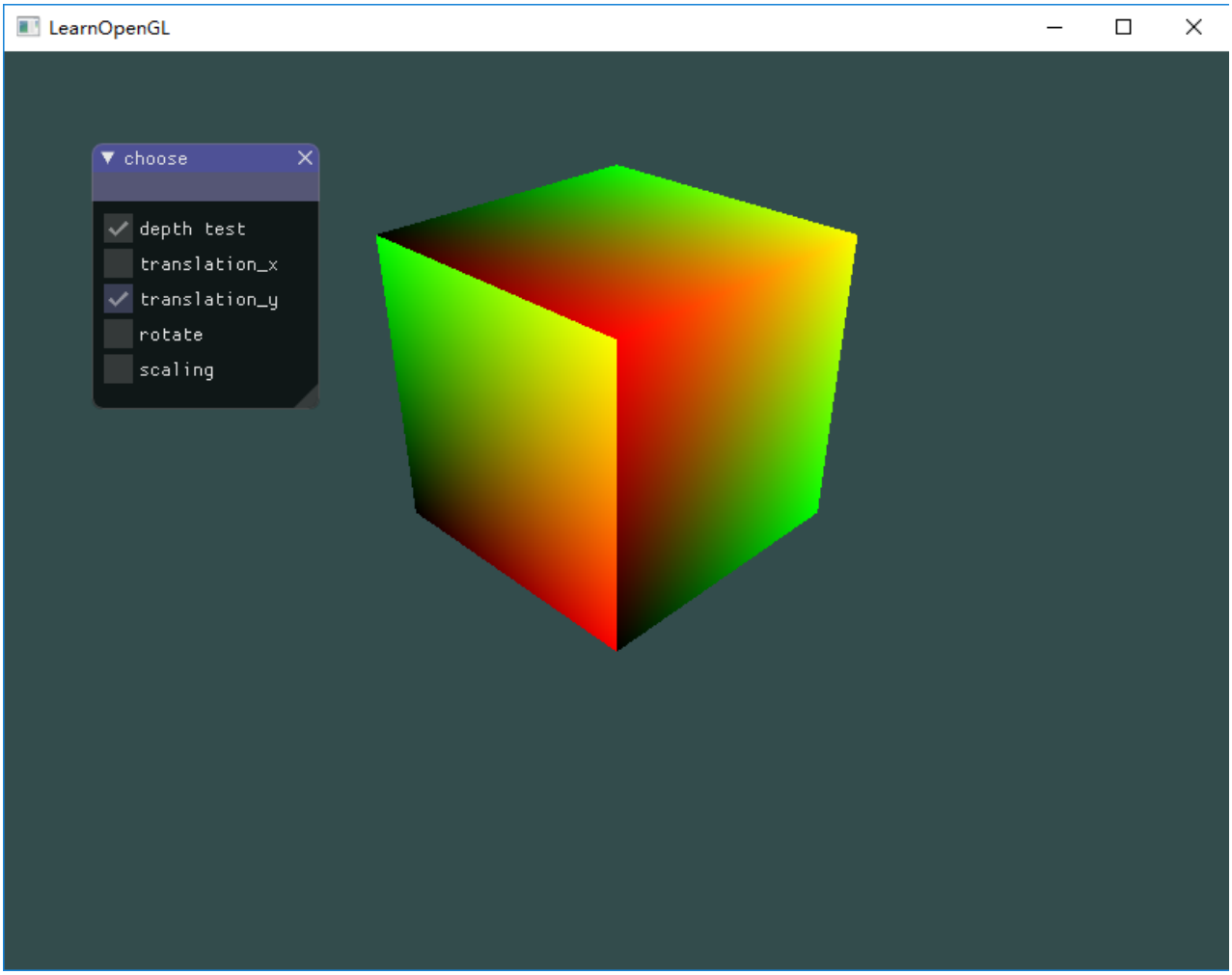


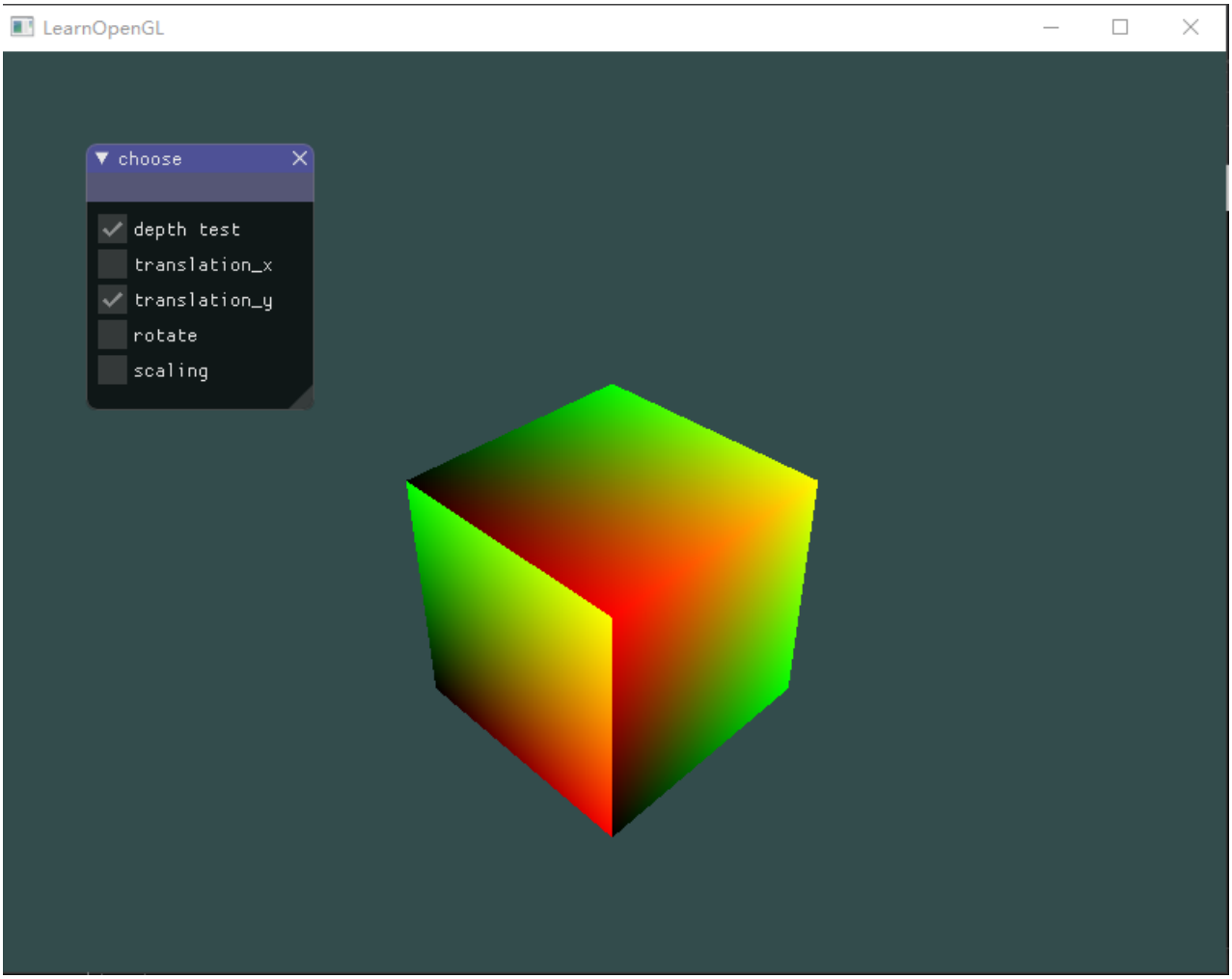
3. 平移：在X轴上来回移动：


LearnOpenGL
choose
depth test
translation_x
translation_y
rotate
scaling

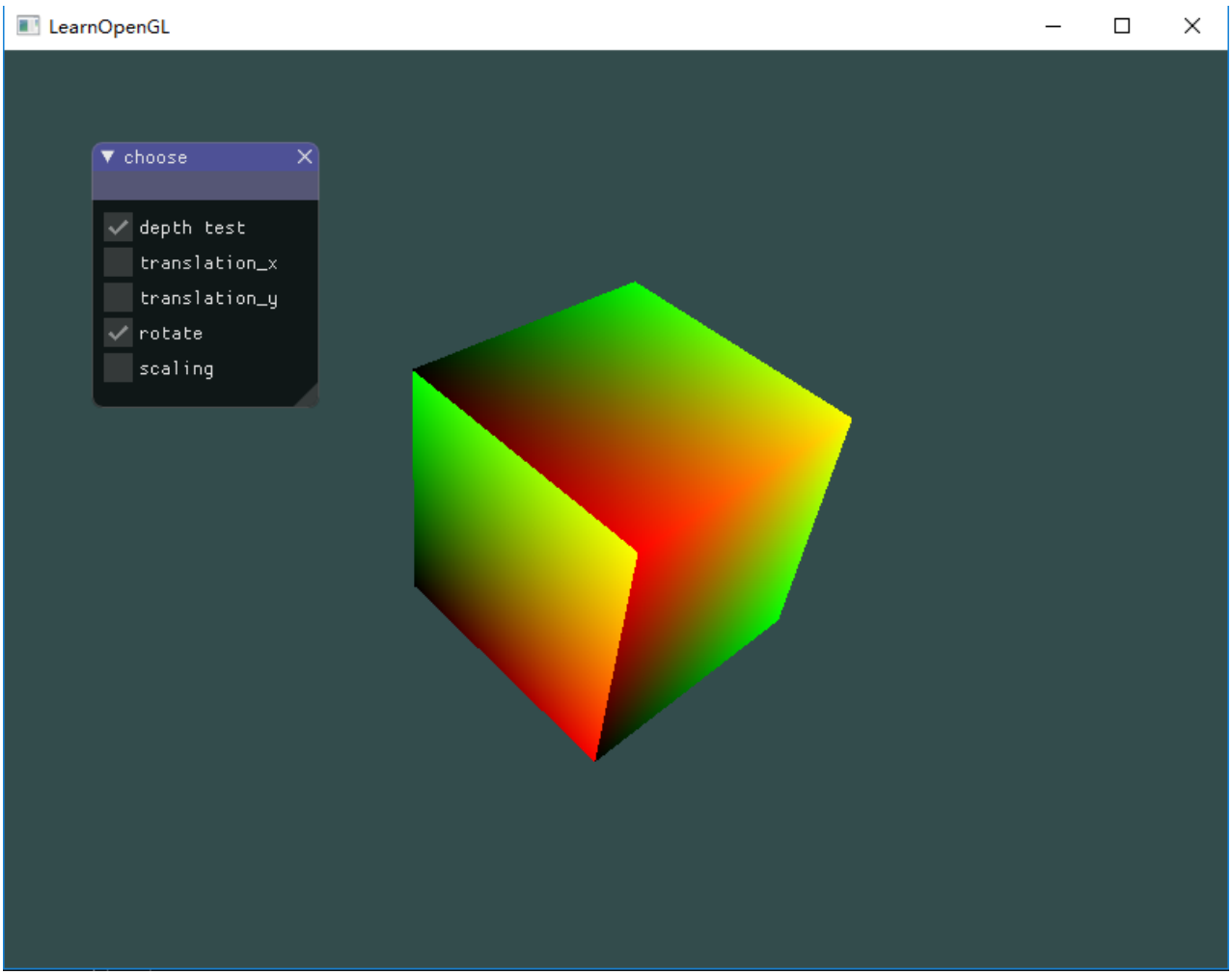


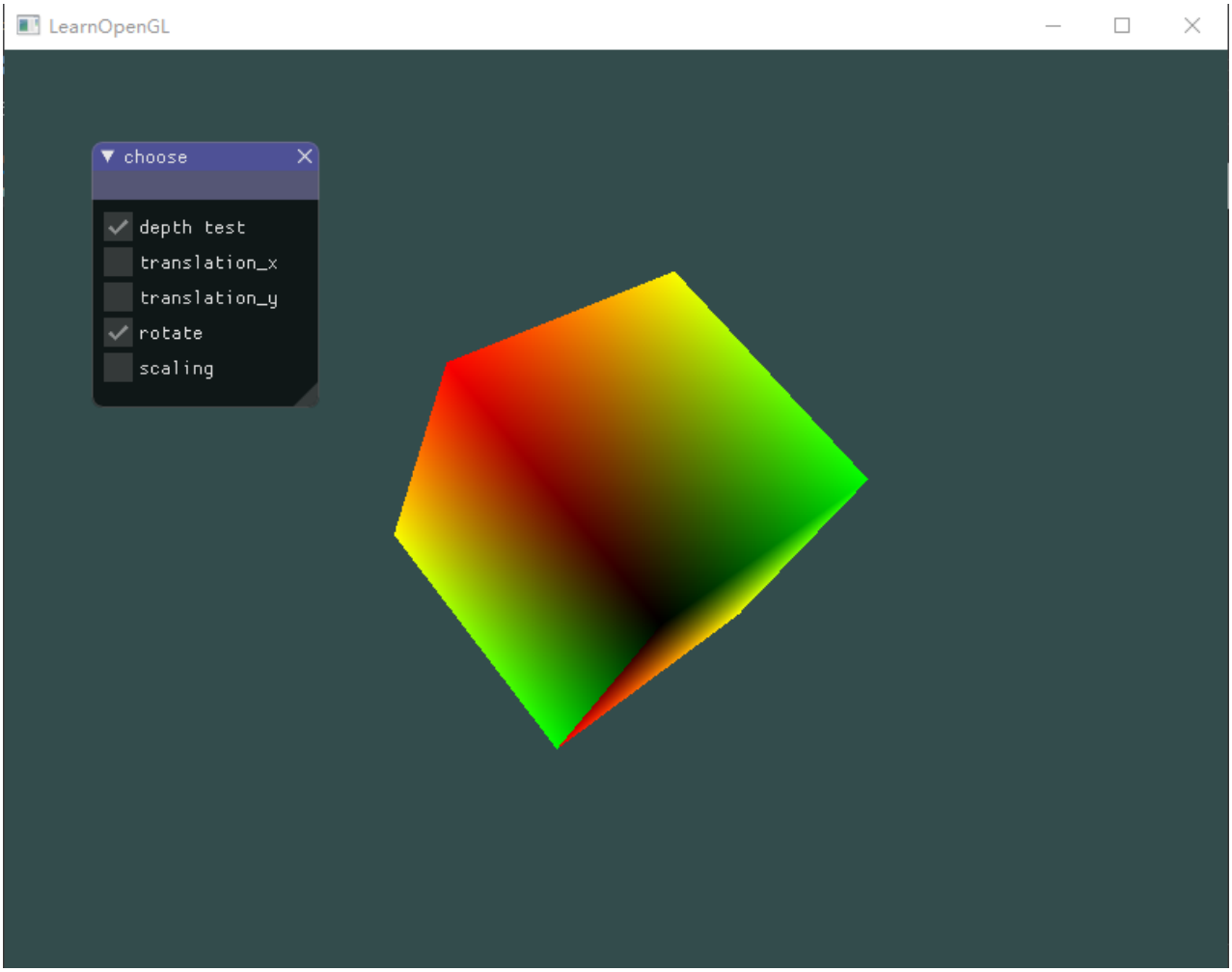
4. 平移：在Y轴上来回移动：


LearnOpenGL
choose
depth test
translation_x
translation_y
rotate
scaling

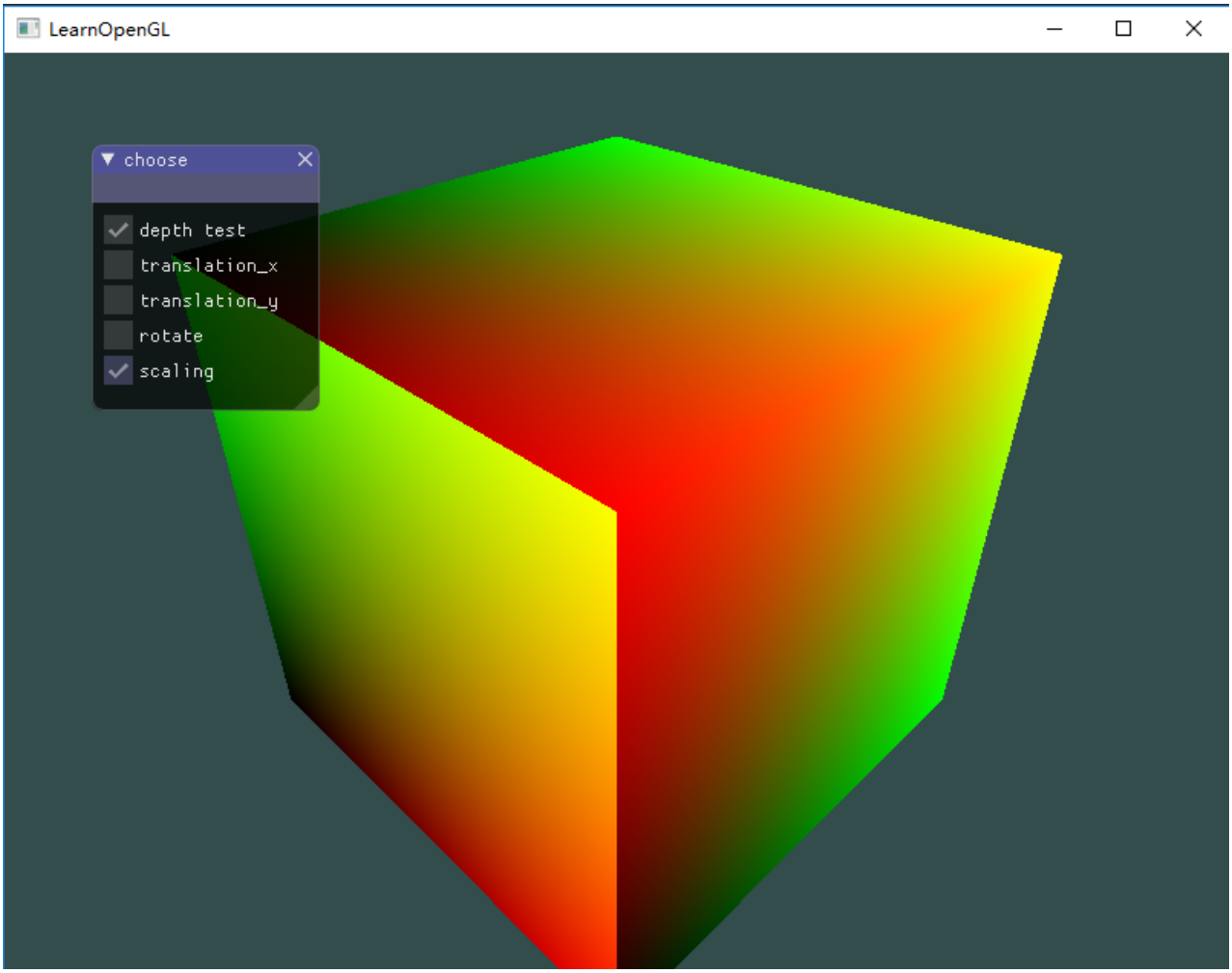


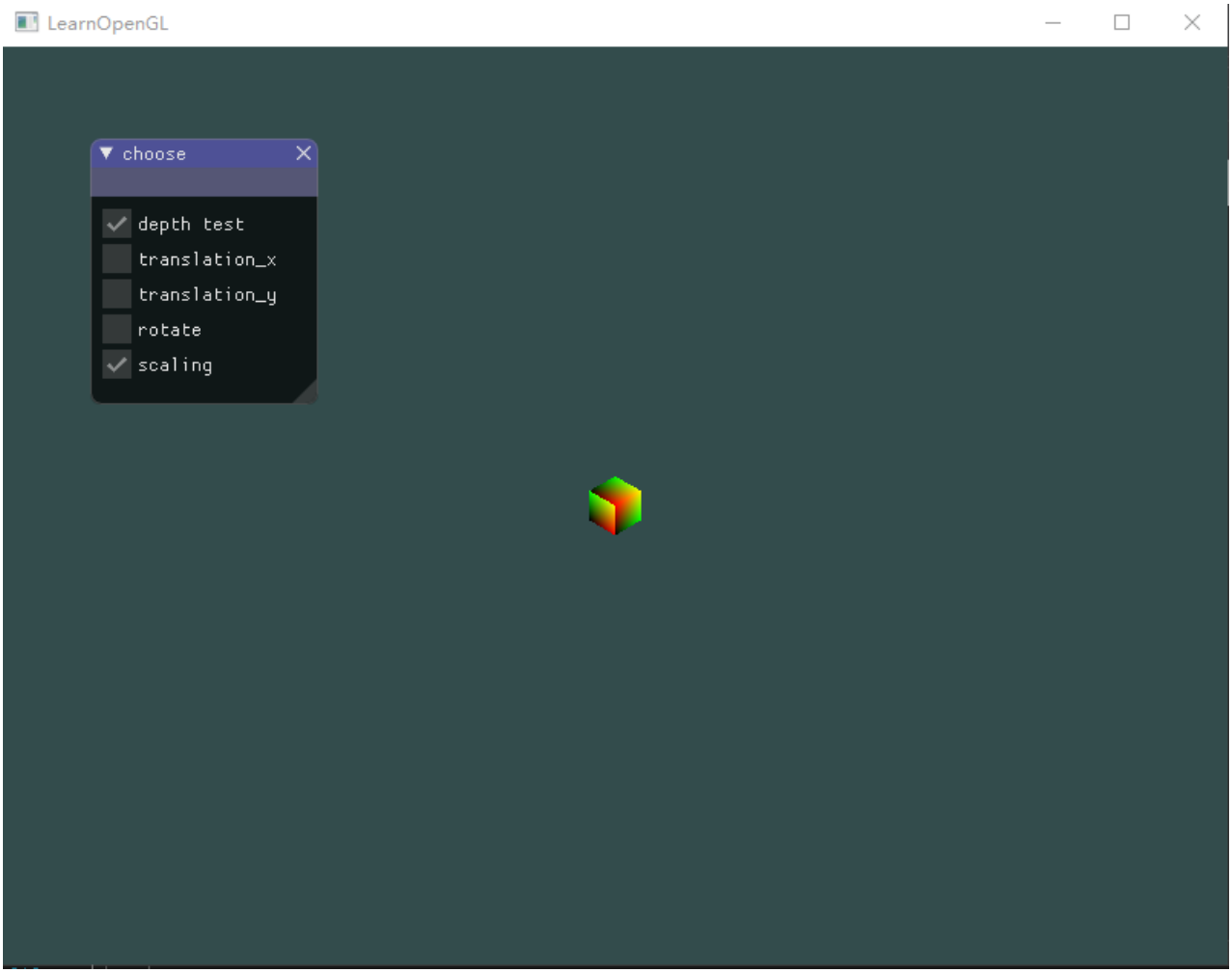
5. 使画好的cube沿着XoZ平面的x=z轴持续旋转：


LearnOpenGL
choose
depth test
translation_x
translation_y
rotate
scaling

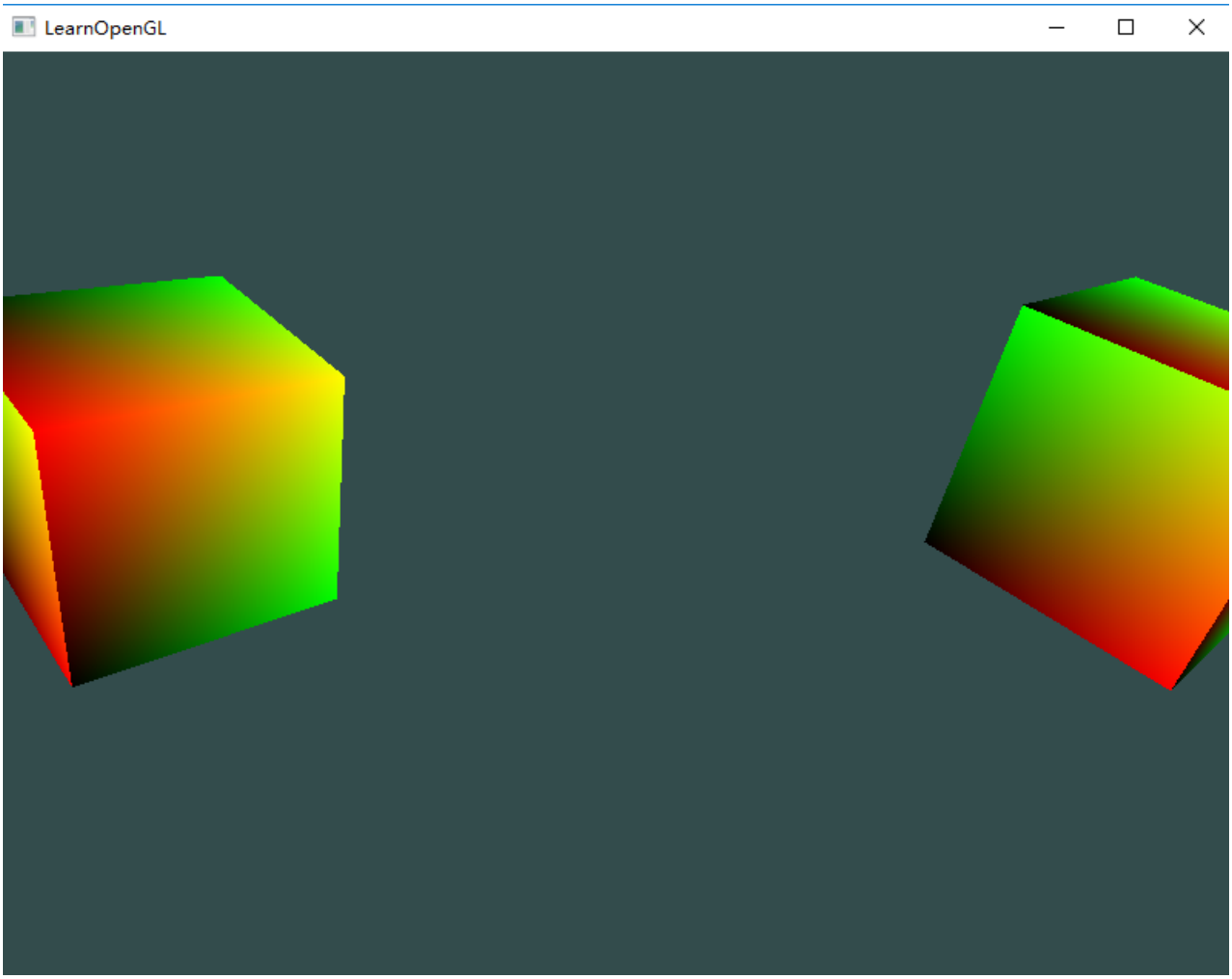


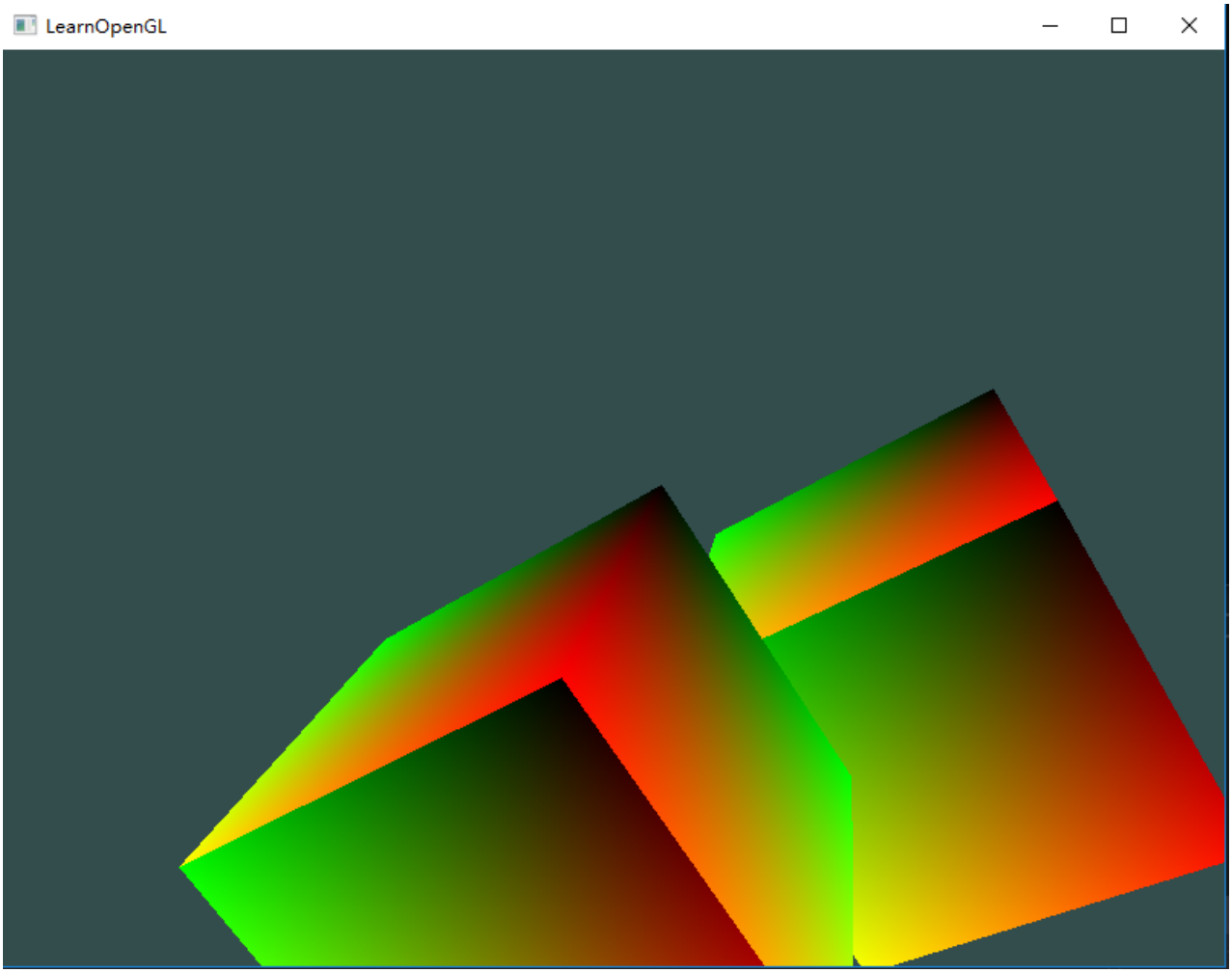
6. 放缩(Scaling)：使画好的cube持续放大缩小：

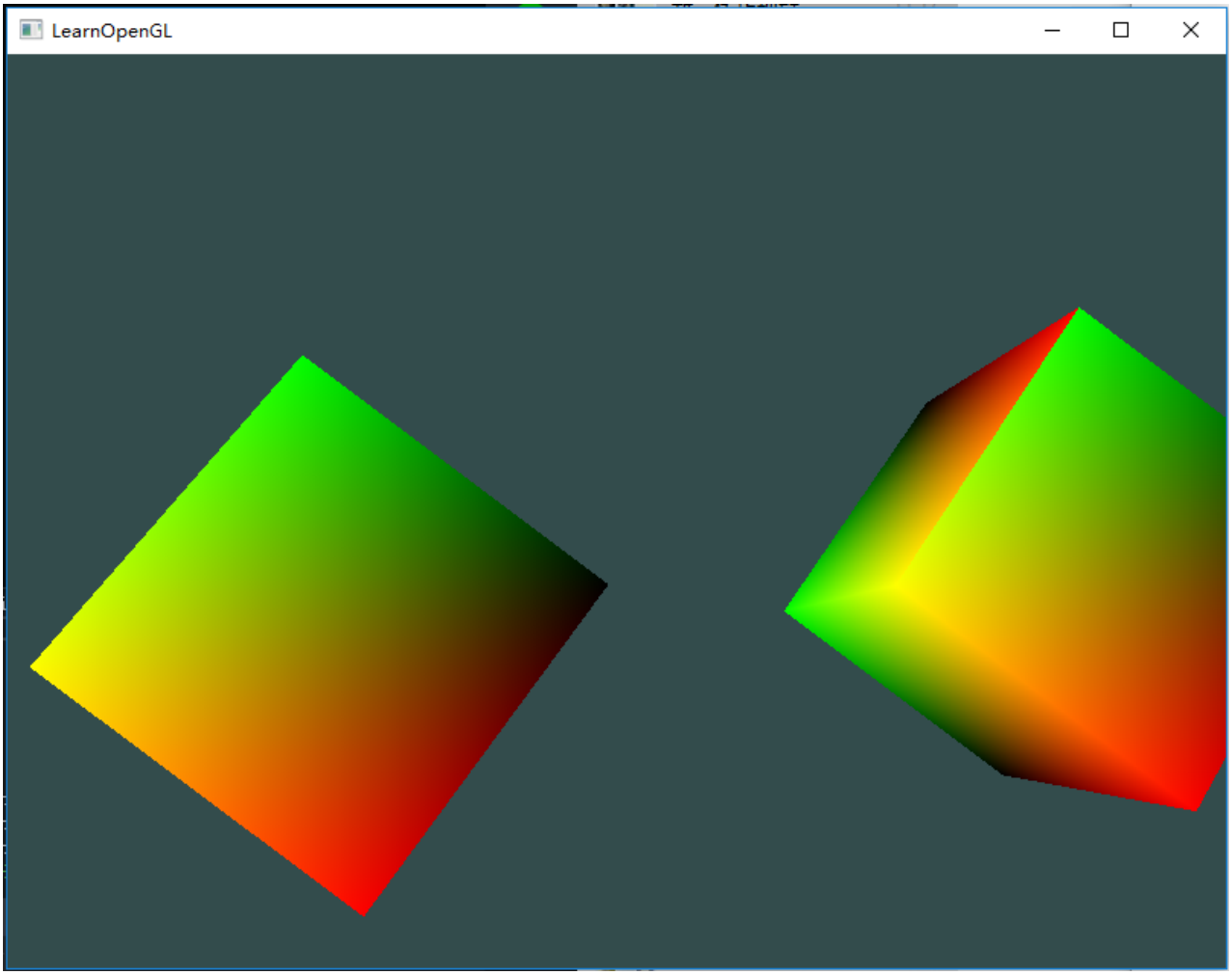

LearnOpenGL
choose
depth test
translation_x
translation_y
rotate
scaling



7. Bonus：两个不同位置的自旋立方体，互相碰撞后来回移动，同时通过放大调整方块大小。


LearnOpenGL


LearnOpenGL



关键代码

1. 立方体边长为4，左边设为+/-2.0f时，边长则为4。

```
// 立方体顶点坐标
float vertices[] = {
    //位置                  颜色
    -2.0f, -2.0f, -2.0f,  0.0f, 0.0f, 0.0f,
     2.0f, -2.0f, -2.0f,  1.0f, 0.0f, 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,  0.0f, 1.0f, 0.0f,
    -2.0f, -2.0f, -2.0f,  0.0f, 0.0f, 0.0f,

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

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

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

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

2. ImGui 选项设置

```
ImGui_ImplOpenGL3_NewFrame();
ImGui_ImplGlfw_NewFrame();
ImGui::NewFrame();
bool ImGui = true
ImGui::Begin("choose", &ImGui, ImGuiWindowFlags_MenuBar);

ImGui::Checkbox("depth test", &depth_test);
ImGui::Checkbox("translation_x", &translation_x);
ImGui::Checkbox("translation_y", &translation_y);
ImGui::Checkbox("rotate", &rotate);
ImGui::Checkbox("scaling", &scaling);

ImGui::End();
...
ImGui::Render();
ImGui_ImplOpenGL3_RenderDrawData(ImGui::GetDrawData());
```

3. 实现平移、旋转、缩放动画，通过改变model的设置来实现

```
float size = 0.0f;
bool add = true;
...

glm::mat4 model = glm::mat4(1.0f);
```

```
glm::mat4 view = glm::mat4(1.0f);
glm::mat4 projection = glm::mat4(1.0f);

// 调整size大小完成动画
if (add && size > 2.0f) {
    add = false;
}
if (!add && size < -2.0f) {
    add = true;
}

size += add ? 0.001f : -0.001f;

if (translation_x) {
    model = glm::translate(model, glm::vec3(size, 0.0f, 0.0f));
}
if (translation_y) {
    model = glm::translate(model, glm::vec3(0.0f, size, 0.0f));
}
if (rotate) {
    model = glm::rotate(model, (float)glfwGetTime(), glm::vec3(0.0f, 0.0f, 1.0f));
}
if (scaling) {
    model = glm::scale(model, glm::vec3(size, size, size));
}
// 设置view第一个参数为相机的位置，第二个参数为（0，0，0）将立方体置于世界坐标的（0，0，0）
// 第三个参数为相机视角
view = glm::lookAt(glm::vec3(8, 8, 8), glm::vec3(0, 0, 0), glm::vec3(0, 1, 0));
projection = glm::perspective(glm::radians(50.0f), (float)SCR_WIDTH /
(float)SCR_HEIGHT, 0.1f, 100.0f);
shader.setMat4("view", view);
shader.setMat4("model", model);
shader.setMat4("projection", projection);
```

4. 开启/关闭深度测试

```
if (depth_test) {
    glEnable(GL_DEPTH_TEST);
}
else {
    glDisable(GL_DEPTH_TEST);
}
...
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

5. bonus：创建2个立方体，添加一个平移矩阵数据，通过改变立方体位置实现多个立方体的放置。

```
glm::vec3 cubePositions[] = {
    glm::vec3(1.8f,  0.0f,  -2.0f),
    glm::vec3(0.0f,  1.0f, 2.0f)
};
...
glm::mat4 model = glm::mat4(1.0f);
model = glm::translate(model, cubePositions[i]);
```

6. bonus：实现立方体旋转、平移、缩放，

```
for (unsigned int i = 0; i < 2; i++)
{
    glm::mat4 model = glm::mat4(1.0f);
    model = glm::translate(model, cubePositions[i]);
    if (i == 0) {
        model = glm::translate(model, glm::vec3(0.0f, 0.0f, size));
        model = glm::scale(model, glm::vec3(1.5f, 1.5f, 1.5f));
    }
    else {
        model = glm::translate(model, glm::vec3(size, 0.0f, 0.0f));
    }
    model = glm::rotate(model, (float)glfwGetTime(), glm::vec3(1.0f, 0.3f, 0.5f));

    shader.setMat4("model", model);

    glDrawArrays(GL_TRIANGLES, 0, 36);
}
```

实验思考

1. 启动和关闭深度测试的区别和原因

   关闭深度测试时，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的上面。而打开深度测试时，则使得被遮住的面不会展示出来，显示的是一个正常的立方体。

   原因是因为当关闭深度测试时，OpenGL是一个三角形一个三角形地来绘制立方体，不会检查要绘制的地方上是否已经有了其他像素的存在，即有东西会被覆盖，这样就使得有些三角形被绘制在了其他三角形上面。

   OpenGL存储所有深度信息在一个Z缓冲，深度值存储在每个片段里面，而当开启深度测试时，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。

2. 结合Shader谈谈对渲染管线的理解

   渲染管线主要的工作阶段有:

   1. 顶点变换
   2. 图元装配
   3. 纹理映射、计算
   4. 光栅化
   5. 测试与混合

首先渲染管线拿到顶点数据后，完成顶点位置变换、计算顶点观照、纹理坐标变换。

然后进入图元装配阶段，使用变换后的顶点以及连通性信息来形成一种原始的绘制数据，这个阶段还负责对视锥裁剪操作，背面剔除，光栅扫描确定的片段，和原始的像素位置。

之后进入纹理映射、计算阶段，这里会有插值片段信息的输入，它可以结合例如一个纹理像素。每个片段的本阶段的共同的最终结果是一个颜色值和深度的片段。

再进行光栅化阶段，涉及到像素处理，像素处理主要包括：对每个像素区域进行着色，对像素贴上贴图，最后形成最终的画面。从而渲染物体。

最后一个阶段时测试与混合，测试的方式有裁减测试、透明度测试、模板测试、深度测试，通过测试的片段信息，根据当前的混合模式，用于更新的像素的值。

在着几个阶段中，顶点shader发生在顶点变换阶段，通过顶点shader可以修改一些基本的图元属性，颜色，光照，发现等。片段shader则主要发生在纹理着色阶段，主要是对上一阶段输出的数据，进行再次加工。