Homework 4 - Transformation

黄树凯

16340085

作业要求

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 <math.h> 、<time.h> 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。 作业

实现及结果

1. 画一个立方体

一个立方体每个面由两个三角形构成，即六个点坐标；初始化36个点坐标，同样为坐标数组生成VAO、VBO，并绑定，其中每个坐标包含6个元素分别为 3 个坐标值和 3 个 RGB值，设置位置属性和颜色属性，调用着色器进行渲染。

为了进行后续的变换操作，还要进行坐标变换，首先初始化三个坐标变换矩阵：模型矩阵、观察矩阵和投影矩阵。

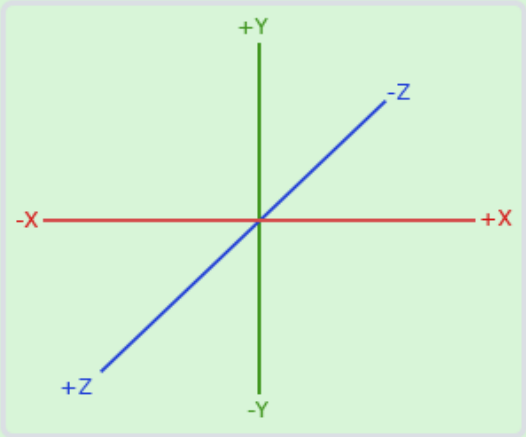
```
glm::mat4 model = glm::mat4(1.0f); // 初始化矩阵
glm::mat4 view = glm::mat4(1.0f);
glm::mat4 projection = glm::mat4(1.0f);
```

一个顶点坐标将会根据以下过程被变换到裁剪坐标：

$$V_{clip} = M_{projection} \cdot M_{view} \cdot M_{model} \cdot V_{local}$$

右手坐标系(Right-handed System)

按照惯例，OpenGL是一个右手坐标系。简单来说，就是正x轴在你的右手边，正y轴朝上，而正z轴是朝向后方的。想象你的屏幕处于三个轴的中心，则正z轴穿过你的屏幕朝向你。坐标系画起来如下：



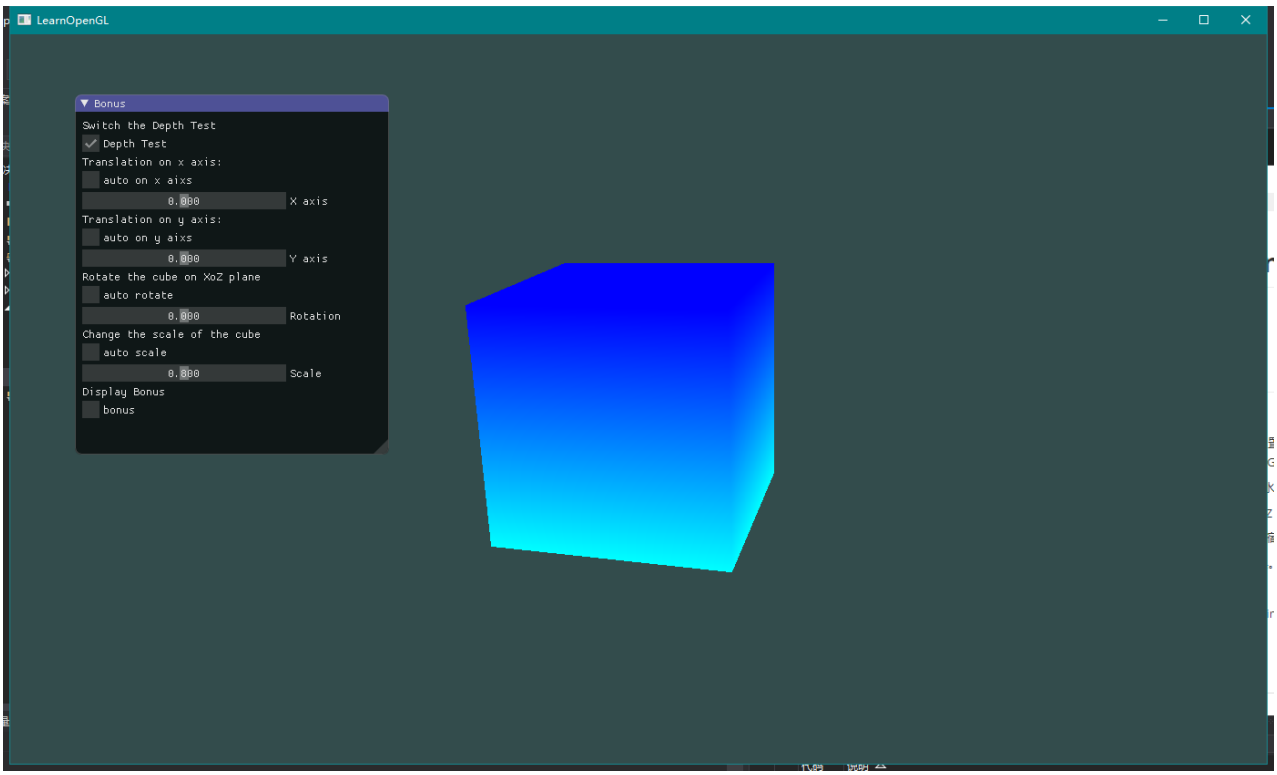
为了使立方体的立体效果更明显，预先作以下变换

```
// 模型矩阵沿 x 轴逆时针、沿 y 轴顺时针旋转一定角度
model = glm::rotate(model, glm::radians(25.0f), glm::vec3(1.0f, -1.0f, 0.0f));
// 将观察矩阵向要进行移动场景的反方向移动。
view = glm::translate(view, glm::vec3(0.0f, 0.0f, -3.0f));
// 用投影矩阵在场景中使用透视投影
projection = glm::perspective(glm::radians(45.0f), (float)SCR_WIDTH / (float)SCR_HEIGHT,
0.1f, 100.0f);
```

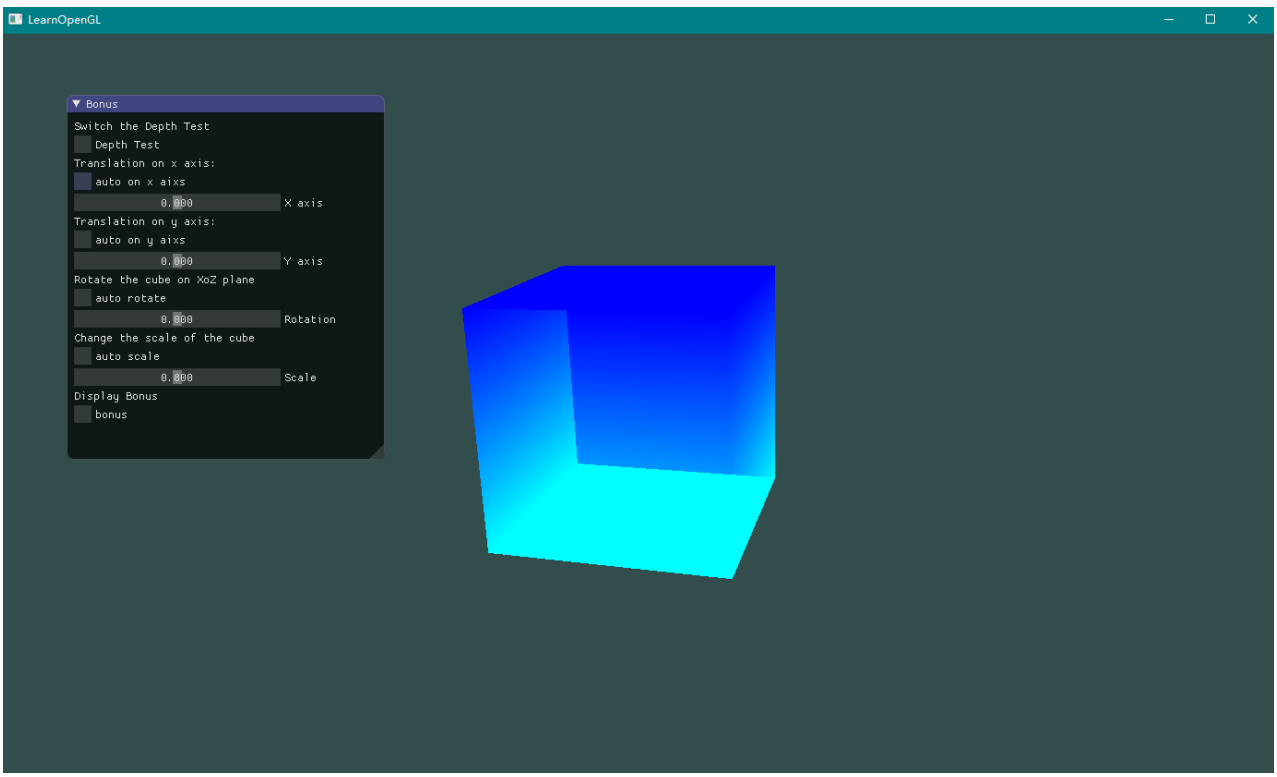
最后将矩阵传入着色器，并进行渲染

```
unsigned int modelLoc = glGetUniformLocation(myShader.ID, "model");
unsigned int viewLoc = glGetUniformLocation(myShader.ID, "view");
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(viewLoc, 1, GL_FALSE, &view[0][0]);
myShader.setMat4("projection", projection);
```

启动了深度测试的效果如下：

- 

关闭深度测试的效果如下：！

- 

区别及原因分析：

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之上；之所 以这样是因为OpenGL是一个三角形一个三角形地来绘制你的立方体的，所以即便之前那里 有东西它也会覆盖之前的像素。因为这个原因，有些三角形会被绘制在其它三角形上面；

启动深度测试之后，立方体中各面的显示按照给定坐标正常出现，面的前后遮蔽效果正常。

2. 平移 (Translation)

使画好的 cube 沿着水平或垂直方向来回移动。

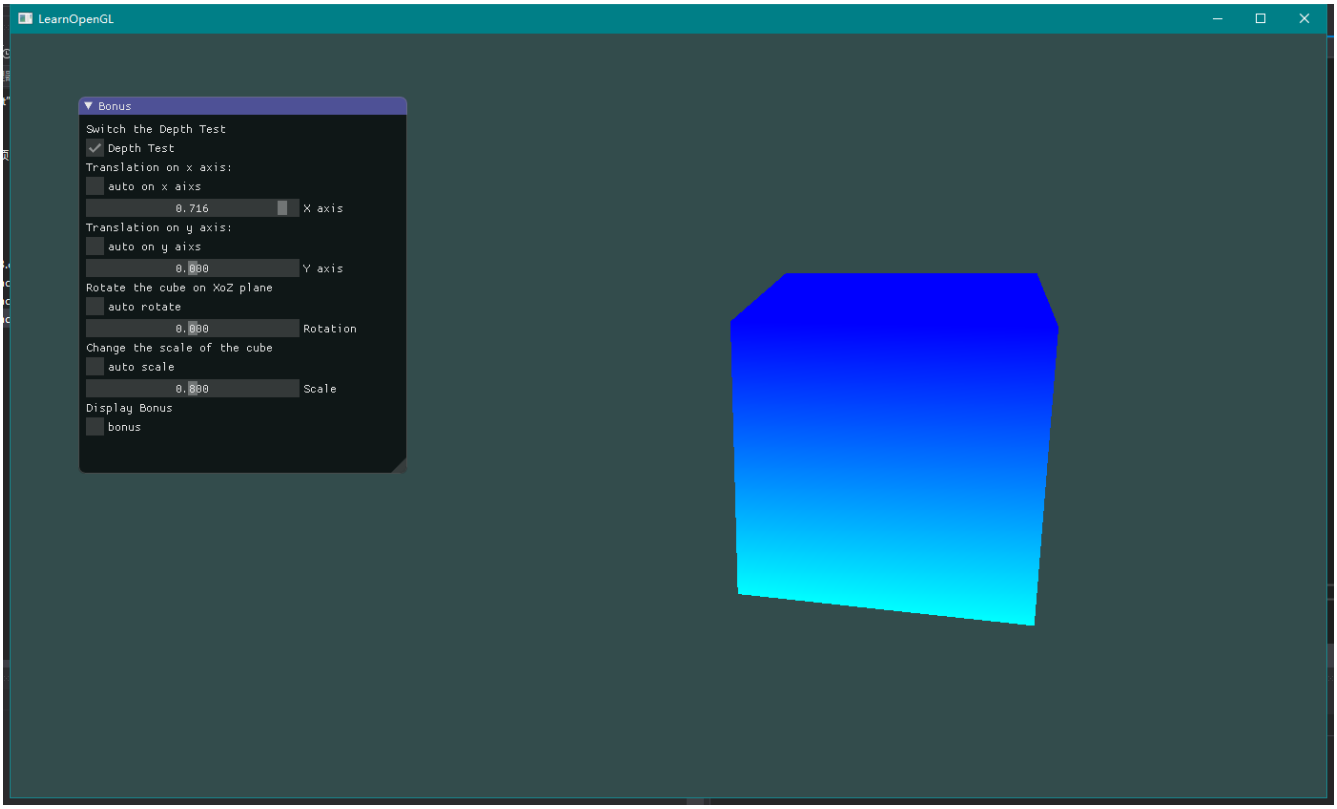
实现方式是通过调用 glm 的 translate 函数，第一个参数是原矩阵，第二个参数是一个三元坐标数组，其中每个坐标表示物体相对局部坐标中心的距离，因此实现水平方向移动或垂直方向移动的方法是传入相应的 x 轴、 y 轴坐标。

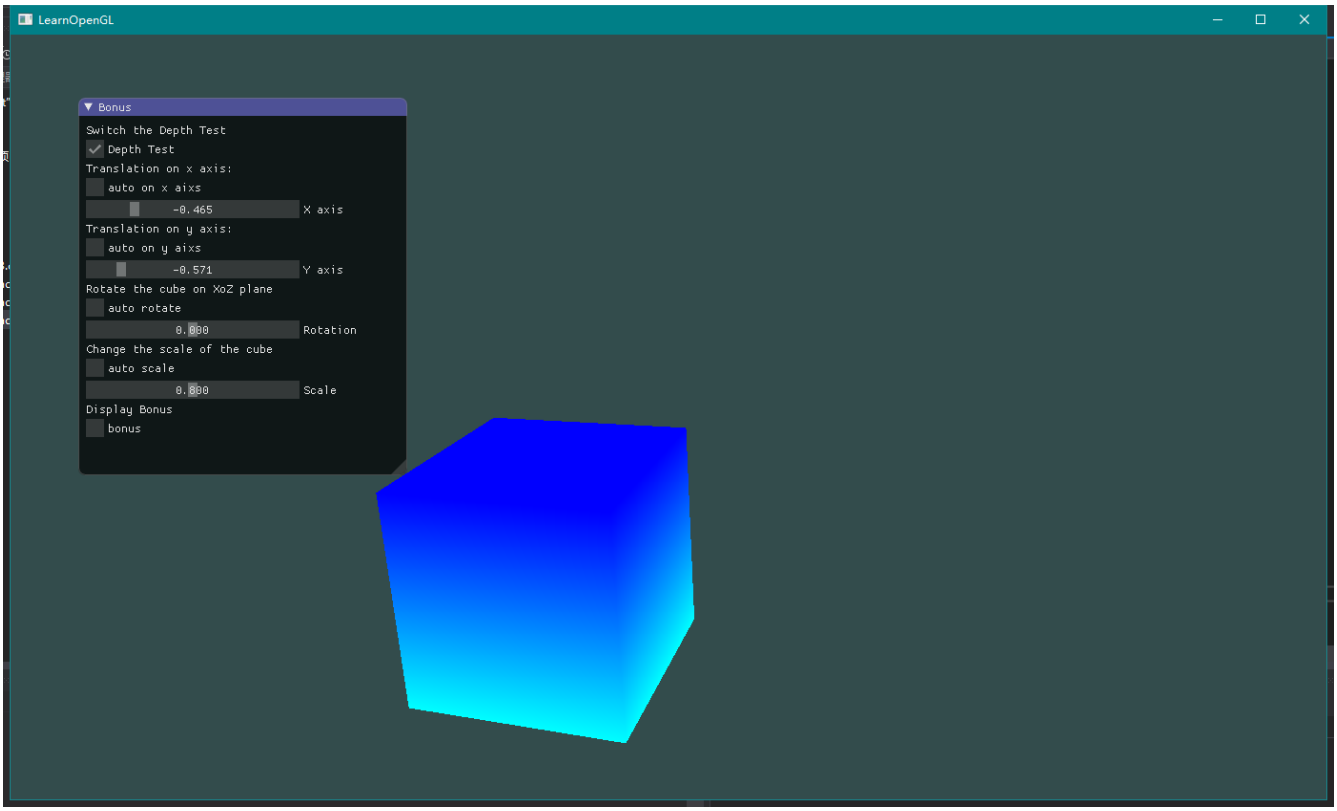
```
model = glm::translate(model, glm::vec3(x_axis, y_axis, 0.0f)); // 移动
```

实现来回移动的方法是定义一个变量，变量在给定范围内变化，当到达最大值时开始减小，当到达最小值时开始增大，实现代码如下：

```
// 水平方向移动
if (auto_translate_x) {
    x_axis += x_add ? 0.0025f : -0.0025f;
    if (x_axis <= -0.8f) {
        x_add = true;
    }
    else if (x_axis >= 0.8f) {
        x_add = false;
    }
}
// 垂直方向移动
if (auto_translate_y) {
    y_axis += y_add ? 0.0025f : -0.0025f;
    if (y_axis <= -0.8f) {
        y_add = true;
    }
    else if (y_axis >= 0.8f) {
        y_add = false;
    }
}
```

效果如下：





3. 旋转 (Rotation)

使画好的 cube 沿着 XoZ 平面的 x=z 轴持续旋转。

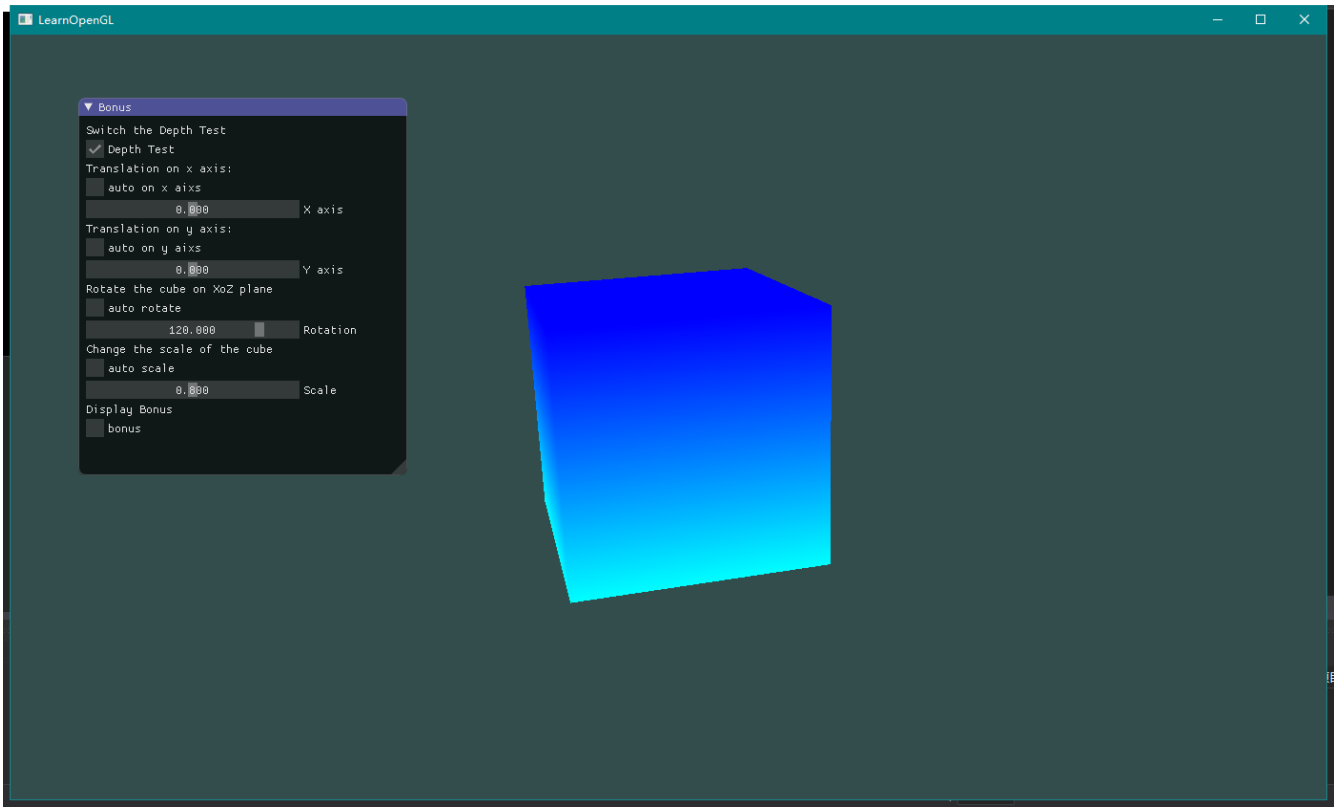
实现方式是通过调用 glm 的 rotate 函数，第一个参数是原矩阵，第二个参数是角度值，第三个参数是一个三元坐标数组，其中每个坐标表示物体在每个轴上旋转的系数，用该系数和传入的角度值进行运算得到绕每个轴的旋转角度。

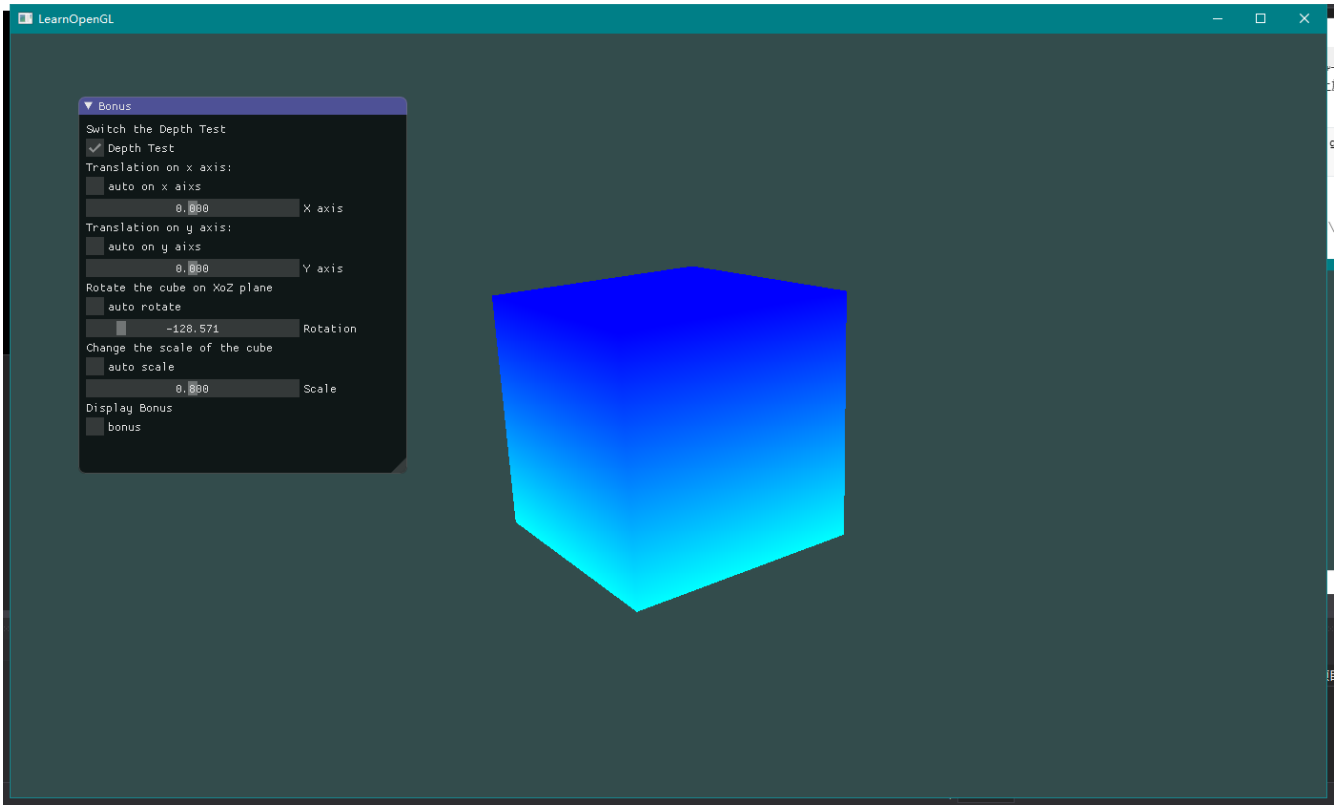
```
model = glm::rotate(model, glm::radians(ratio), glm::vec3(0.0f, 1.0f, 0.0f)); // 在 xoz 平面上旋转，即绕着 y 轴旋转
```

实现持续旋转的方法是定义一个变量，变量在给定范围内变化，当到达最大值时开始减小，当到达最小值时开始增大，（或单调增加，可实现立方体一直往同一个方向旋转，若交替增加减小会使立方体旋转到一定角度就往逆方向旋转）实现代码如下：

```
// 在 xoz 平面上旋转
if (auto_rotate) {
    ratio += rotate_add ? 0.5f : -0.5f;
    if (ratio <= -180.0f) {
        rotate_add = true;
    }
    else if (ratio >= 180.0f) {
        rotate_add = false;
    }
}
```

效果如下：





4. 放缩(Scaling)

使画好的 cube 持续放大缩小。

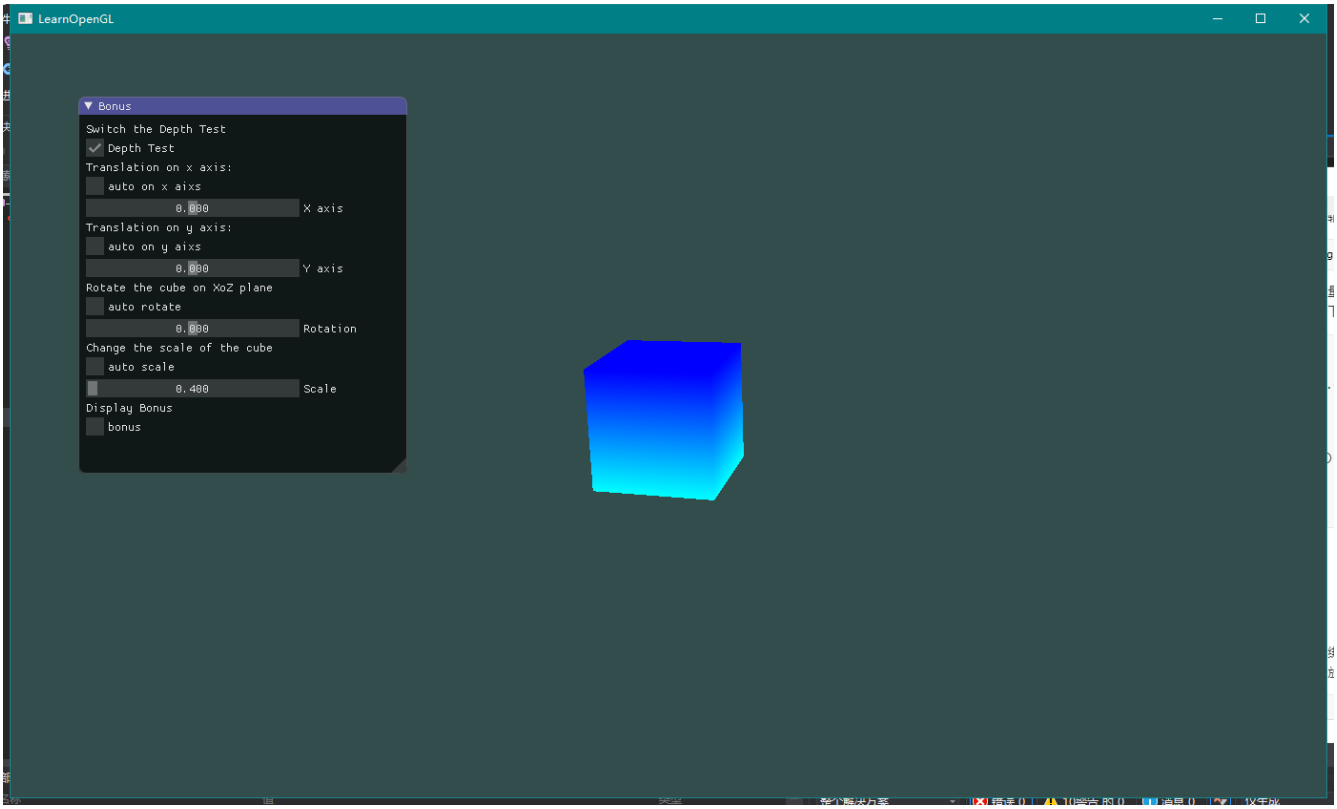
实现方式是通过调用 glm 的 scale 函数，第一个参数是原矩阵，第二个参数是一个三元坐标数组，其中每个坐标表示物体在每个轴上放大的倍数，若小于1则缩小。

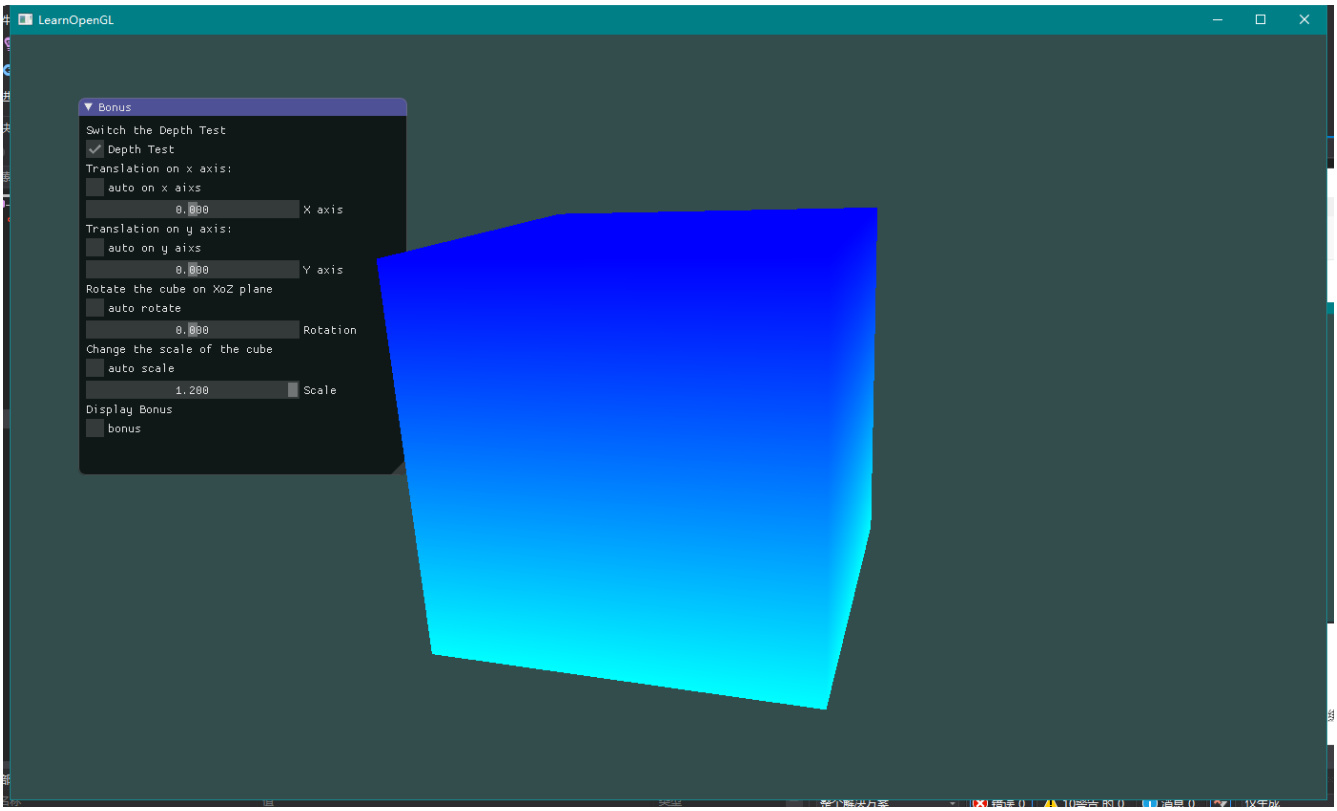
```
model = glm::scale(model, glm::vec3(scale, scale, scale));      // 放大缩小
```

实现持续旋转的方法是定义一个变量，变量在给定范围内变化，当到达最大值时开始减小，当到达最小值时开始增大，实现代码如下：

```
// 放大缩小
if (auto_scale) {
    scale += scale_add ? 0.001f : -0.001f;
    if (scale <= 0.4f) {
        scale_add = true;
    }
    else if (scale >= 1.2f) {
        scale_add = false;
    }
}
```

效果如下：





5. 添加GUI

为各个变换定义一个布尔型变量，绑定到 checkbox上，判断布尔值为真时则执行选中变换的代码，为平移的距离、旋转的角度、放缩的倍数定义变量并绑定到滑动条，实现手动的调整变换

```
// ImGui 菜单
{
```

```
    ImGui::Begin("Bonus");

    ImGui::Text("Switch the Depth Test");
    ImGui::Checkbox("Depth Test", &open_depth_test);

    ImGui::Text("Translation on x axis:");
    ImGui::Checkbox("auto on x aixs", &auto_translate_x);
    ImGui::SliderFloat("X axis", &x_axis, -0.8, 0.8);
    ImGui::Text("Translation on y axis:");
    ImGui::Checkbox("auto on y aixs", &auto_translate_y);
    ImGui::SliderFloat("Y axis", &y_axis, -0.8, 0.8);

    ImGui::Text("Rotate the cube on XoZ plane");
    ImGui::Checkbox("auto rotate", &auto_rotate);
    ImGui::SliderFloat("Rotation", &ratio, -180.f, 180.0f);

    ImGui::Text("Change the scale of the cube");
    ImGui::Checkbox("auto scale", &auto_scale);
    ImGui::SliderFloat("Scale", &scale, 0.4f, 1.2f);

    ImGui::Text("Display Bonus");
    ImGui::Checkbox("bonus", &mode);

    ImGui::End();
}
```

6. Shader 和渲染管线

结合 Shader 谈谈对渲染管线的理解

渲染管线是 OpenGL 渲染场景的一整个过程，是将 3D 对象渲染到平面上的流程。目前最新的可编程的方式是使用 Shader 实现，可编程管线相比固定管线，将固定管线中的一部分内容可以用编程的方式实现，相对比较灵活。和传统的固定管线相比主体没有明显变化，但是固定管线中的几何变换和光照计算被 VertexShader (顶点 Shader)取代了，固定管线中的纹理映射、雾效处理等被 Fragments Header取代了。

在 opengl 中编写shader程序，主要有以下步骤：首先创建一个 shader 程序对象，然后分别创建相应的 shader 对象，比如顶点 shader 对象, 片元 shader 对象，并把它们连接到 shader 程序对象。在创建 shader 对象时候，需要装入 shader 源码，编译 shader，链接到 shader 程序对象几个步骤。

shader 管理类似于创建 C/C++ 程序，首先写 shader 代码，把代码放在一个文本文件或者一个字符串中，然后编译该 shader 代码，把编译后的 shader 代码放到各个 shader 对象中，接着把shader 对象链接到程序中，最后把 shader 送到 GPU 中去。

7. Bonus

一段模拟史莱姆跳跃的动画：主要包括四个过程（下落、挤压、回弹、上升）

- 下落状态：沿 y 轴方向的下落速度会逐渐加快，x 轴方向速度不变，围绕 y 轴进行旋转，降落到一定位置时视为触碰地面进入挤压状态
- 挤压状态：y 轴方向缩小，x 和 z 轴方向增大， y 轴小于一定大小时进入回弹状态，此状态时不做平移和旋转

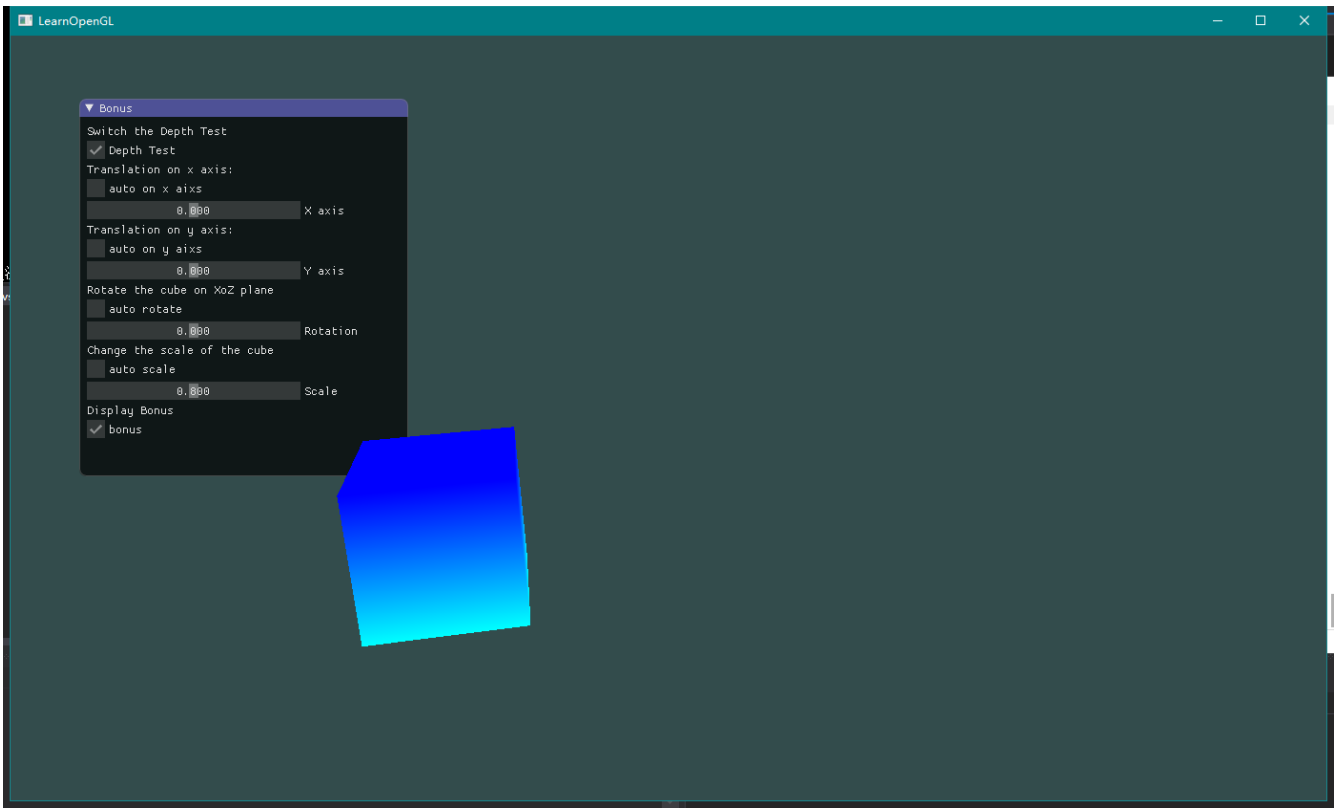
- 回弹状态：y 轴方向增大，x 和 z 轴方向减小，y 轴恢复原大小时进入上升状态，并具有一定初速度（初速度过小，则不上升，而是停止动作），此状态时不做平移和旋转
- 上升状态：沿 y 轴方向的上升速度会逐渐减小，x 轴方向速度不变，围绕 y 轴进行旋转，上升到 y 轴方向为 0 之后会进入下落状态。

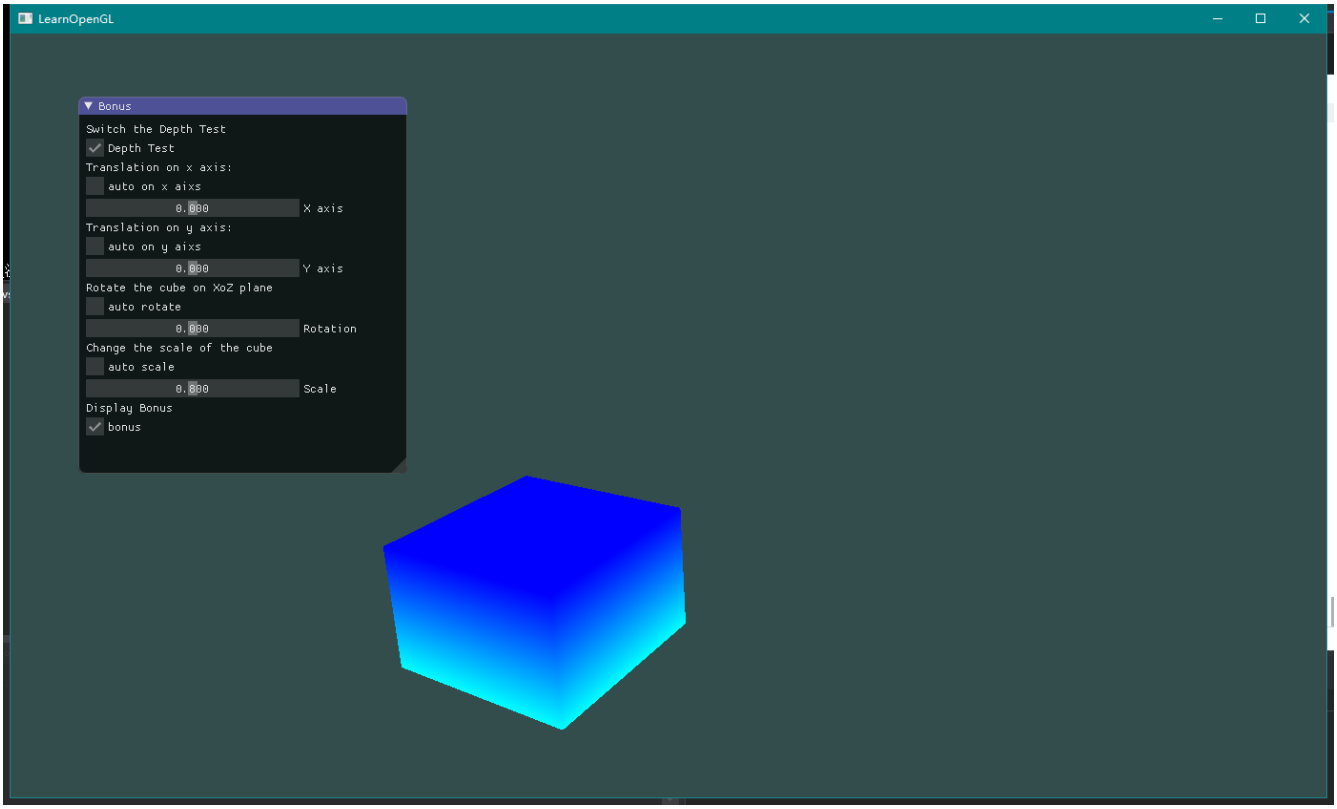
实现代码如下：

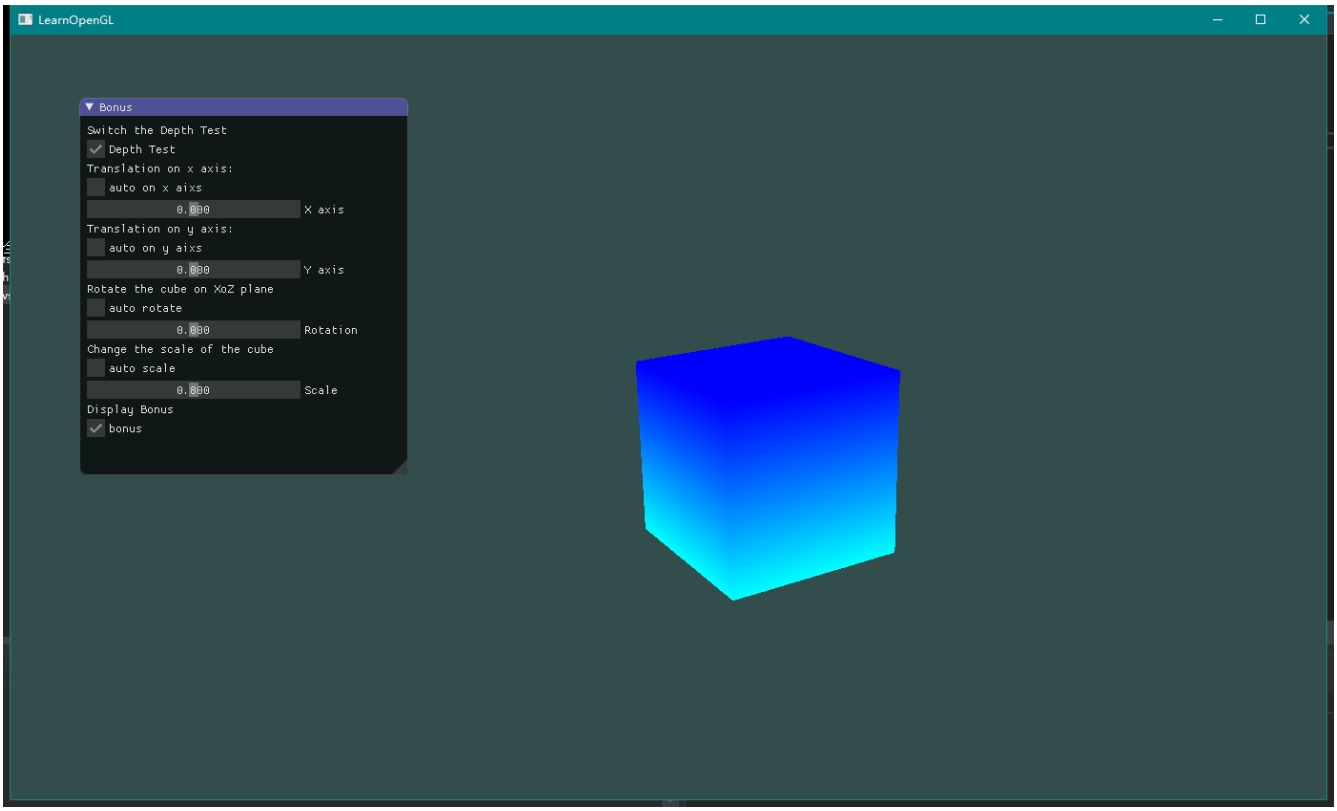
```
if (state != ending) {
    if (state == falling) {
        // 下落速度变化
        bonus_speed += 0.00002f * 9.8;
        bonus_y_axis -= bonus_speed;
        bonus_x_axis += 0.0005f;
        bonus_ratio += 0.3f;
        // 状态变换
        if (bonus_y_axis <= -0.3f) {
            state = squeezing;
            last_speed = bonus_speed;
        }
    }
    else if (state == squeezing) {
        bonus_speed = 0;
        // 形状变化
        y_scale -= 0.004;
        x_scale += 0.002;
        z_scale += 0.002;
        // 状态变换
        if (y_scale <= 0.3f) {
            state = springback;
        }
    }
    else if (state == springback) {
        bonus_speed = 0;
        // 形状变化
        y_scale += 0.004;
        x_scale -= 0.002;
        z_scale -= 0.002;
        // 状态变换
        if (y_scale >= 0.5f) {
            y_scale = x_scale = z_scale = 0.5f;
            state = rising;
            bonus_speed = last_speed * 0.88f;
            if (bonus_speed < 0.005f) {
                state = ending;
            }
        }
     }
     else if (state == rising) {
        // 上升速度变化
        bonus_speed -= 0.00002f * 9.8;
        bonus_y_axis += bonus_speed;
        bonus_x_axis += 0.0005f;
```

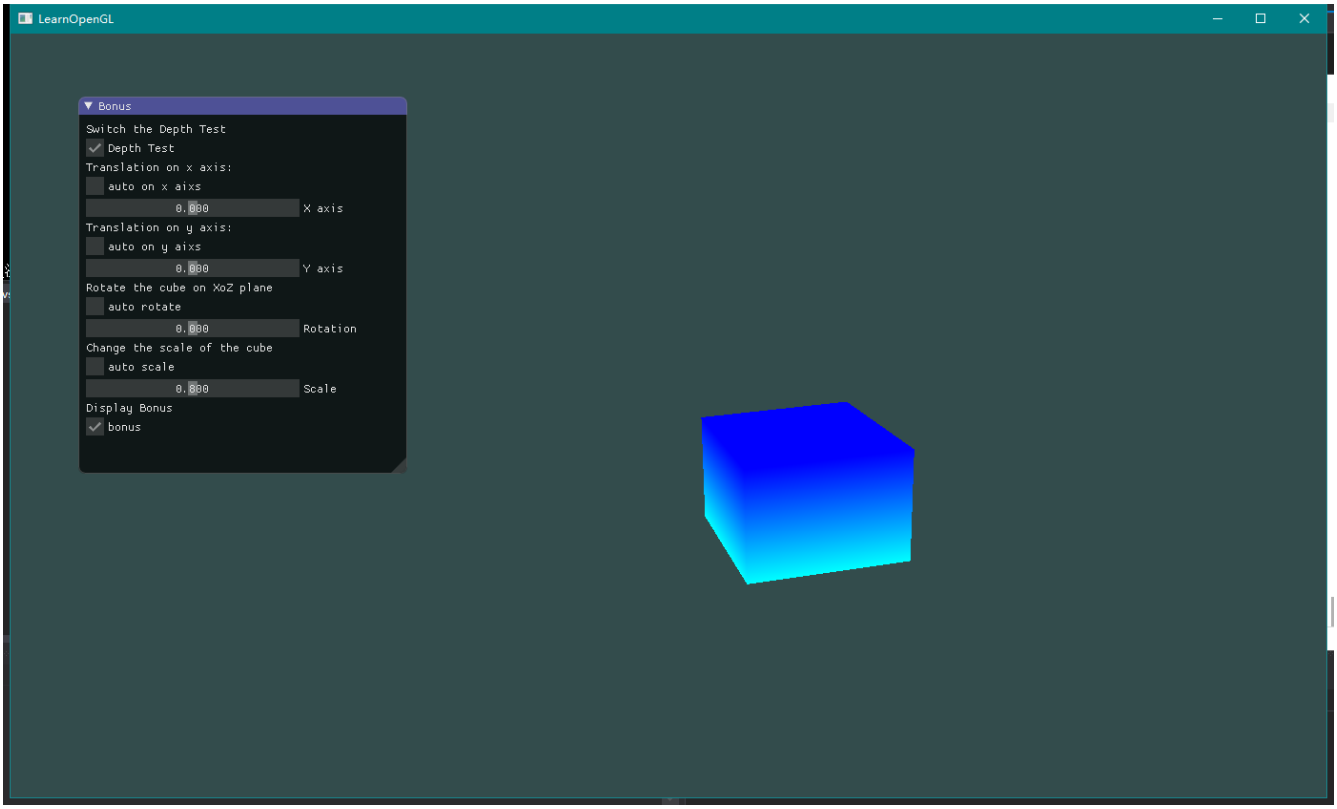
```
            bonus_ratio += 0.3f;
            // 状态变换
            if (bonus_speed <= 0) {
                state = falling;
            }
        }
    }
    model = glm::rotate(model, glm::radians(bonus_ratio), glm::vec3(0.0f, 1.0f, 0.0f)); // 在 XoZ 平面上旋转
    model = glm::translate(model, glm::vec3(bonus_x_axis, bonus_y_axis, 0.0f)); // 平移移动
    model = glm::scale(model, glm::vec3(x_scale, y_scale, z_scale));// 放大缩小
```

效果如下（详见附件动图）：




LearnOpenGL
Bonus
Switch the Depth Test
Depth Test
Translation on x axis:
auto on x aixs
0.000
X axis
Translation on y axis:
auto on y aixs
0.000
Y axis
Rotate the cube on XoZ plane
auto rotate
0.000
Rotation
Change the scale of the cube
auto scale
0.800
Scale
Display Bonus
bonus


LearnOpenGL
Bonus
Switch the Depth Test
Depth Test
Translation on x axis:
auto on x aixs
0.000
X axis
Translation on y axis:
auto on y aixs
0.000
Y axis
Rotate the cube on XoZ plane
auto rotate
0.000
Rotation
Change the scale of the cube
auto scale
0.800
Scale
Display Bonus
bonus


LearnOpenGL
Bonus
Switch the Depth Test
Depth Test
Translation on x axis:
auto on x aixs
0.000
X axis
Translation on y axis:
auto on y aixs
0.000
Y axis
Rotate the cube on XoZ plane
auto rotate
0.000
Rotation
Change the scale of the cube
auto scale
0.800
Scale
Display Bonus
bonus