计算机图形学 Homework4

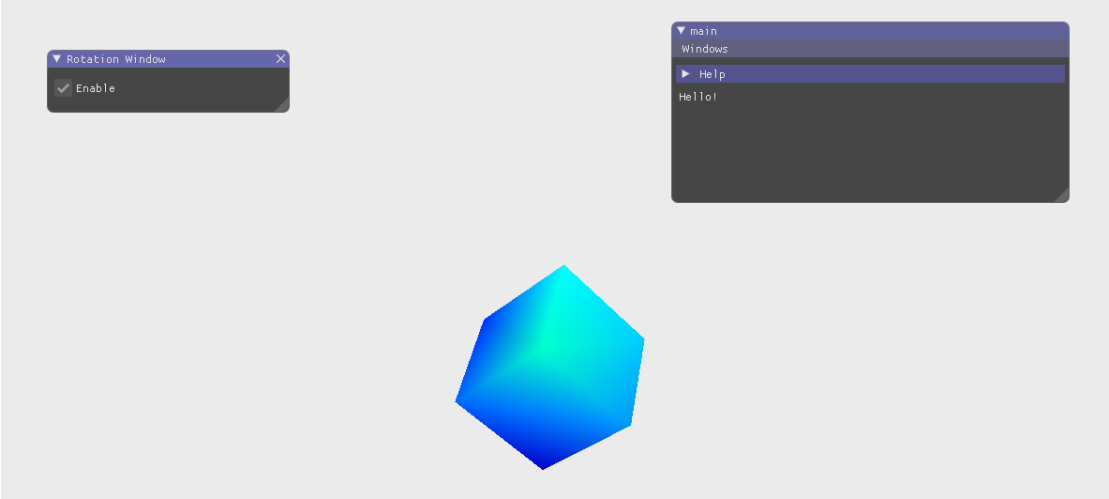
15331416 数字媒体技术方向 赵寒旭

1. 运行结果

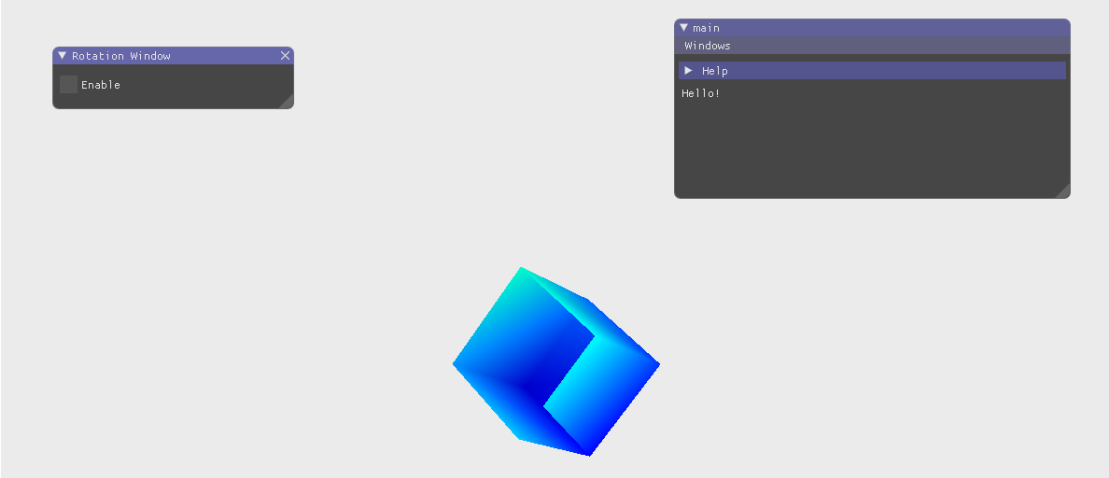
1） 深度测试与旋转

画一个立方体（cube）边长为 0.4，中心位置为（0，0，0）。分别启动和关闭深度测试，查看区别，并分析原因。同时旋转（Rotation）：使 cube 沿着 YOZ 屏幕的 y=z 轴持续旋转。

启动深度测试：



关闭深度测试：



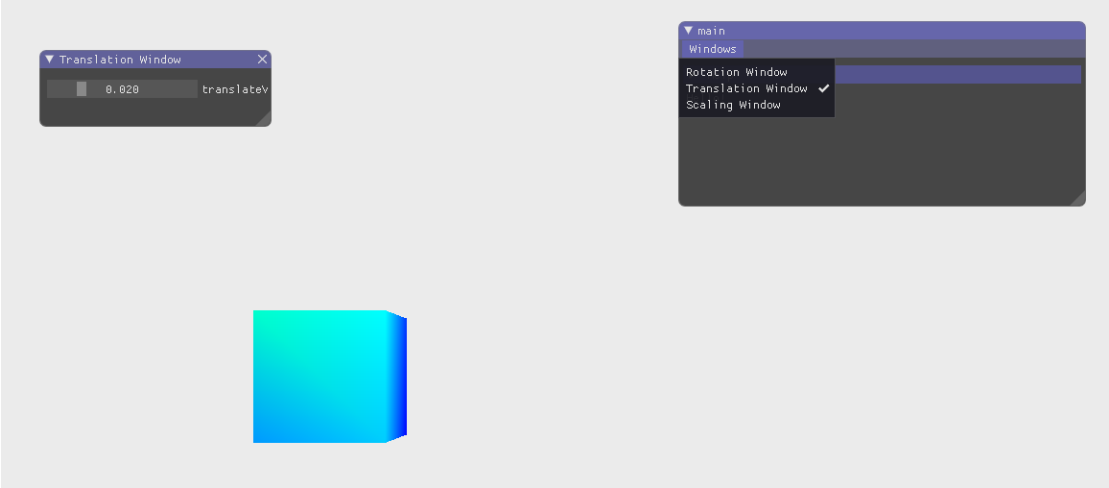
原因：

OpenGL 存储它的所有深度信息于 Z 缓冲区(Z-buffer)中，也被称为深度缓冲区(Depth Buffer)。GLFW 会自动为你生成这样一个缓冲区（就如它有一个颜色缓冲区来存储输出图像的颜色)。深度存储在每个片段里面(作为片段的 z 值)当片段像输出它的颜色时，OpenGL 会将它的深度值和 z 缓冲进行比较然后如果当前的片段在其它片段之后它将会被丢弃，然后重写。这个过程称为深度测试(Depth Testing)并且它是由 OpenGL 自动完成的。

如果我们想要确定 OpenGL 是否真的执行深度测试，首先我们要告诉 OpenGL 我们想要开启深度测试；而这通常是默认关闭的。我们通过 glEnable 函数来开启深度测试。glEnable 和 glDisable 函数允许我们开启或关闭某一个 OpenGL 的功能。该功能会一直是开启或关闭的状态直到另一个调用来关闭或开启它。我们想开启深度测试就需要开启 GL_DEPTH_TEST：glEnable(GL_DEPTH_TEST);

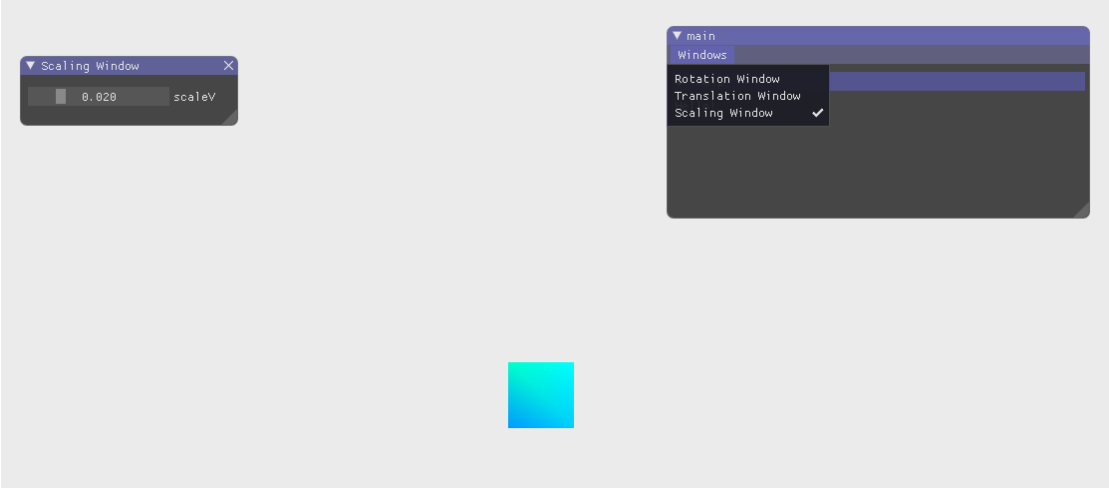
2）平移（Translation）

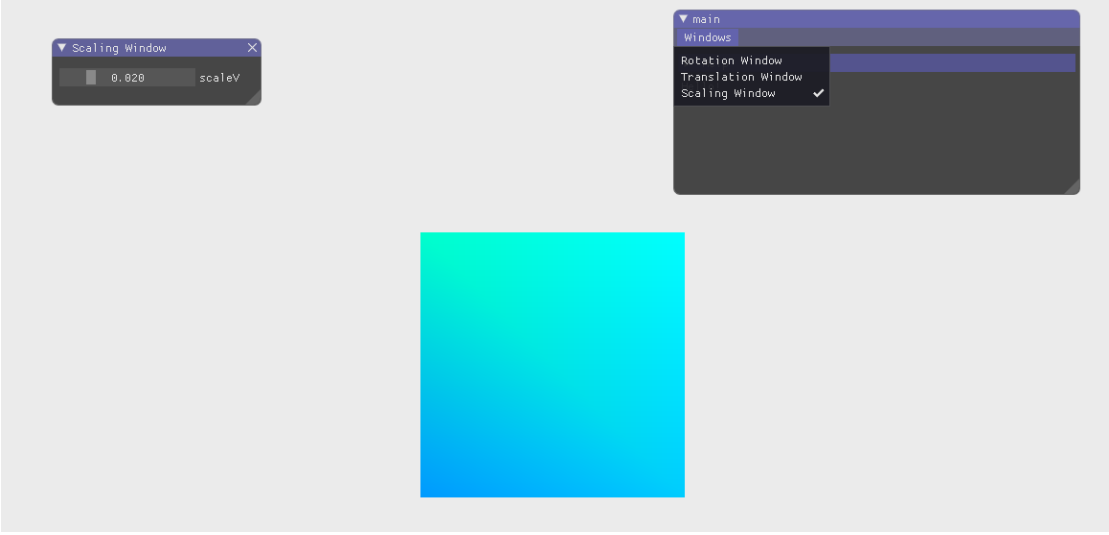
使画好的 cube 沿着水平或垂直方向来回移动。



3）放缩（Scaling）

使画好的 cube 持续放大缩小。





另有视频在/doc 文件夹下。

2. 实现思路

1）顶点坐标设置

组成一个立方体需要 8 个顶点。

实际绘制时：共 6 个面，12 个三角形，36 个顶点。

以下依次是 8 个顶点坐标和颜色值。（选用了 8 个不同的蓝色调颜色）

```
// 8个顶点坐标
    float vertices[] = {
        -0.2f,  0.2f,  0.2f,  0.0f, 1.0f, 0.8f,
        0.2f,  0.2f,  0.2f,  0.0f, 1.0f, 1.0f,
        0.2f, -0.2f,  0.2f,  0.0f, 0.8f, 1.0f,
        -0.2f, -0.2f,  0.2f,  0.0f, 0.6f, 1.0f,
        -0.2f,  0.2f,  -0.2f,  0.0f, 0.4f, 1.0f,
        0.2f,  0.2f,  -0.2f,  0.0f, 0.2f, 1.0f,
        0.2f, -0.2f,  -0.2f,  0.0f, 0.0f, 1.0f,
        -0.2f, -0.2f,  -0.2f,  0.0f, 0.0f, 0.8f


    };
```

以下是 EBO 对应索引值。

```
// 索引
unsigned int indices[] = {
    0, 1, 2,
    2, 3, 0,
    4, 5, 7,
    5, 6, 7,
    0, 4, 3,
    4, 7, 3,
    1, 5, 2,
    5, 6, 2,
    4, 5, 0,
    5, 1, 0,
    7, 6, 3,
    6, 2, 3
};
```

EBO 具体绑定：

```
glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices), indices, GL_STATIC_DRAW);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_INT, 0);
glBindVertexArray(EBO);
```

2）创建模型矩阵

在开始进行三维画图时，我们首先创建一个模型矩阵。这个模型矩阵包含了平移、缩放与旋转，我们将会运用它来将对象的顶点转换到全局世界空间。

（1）旋转

使 cube 沿着 YOZ 屏幕的 y=z 轴持续旋转：

```
model = glm::rotate(model, (GLfloat)glfwGetTime() * 50.0f, glm::vec3(0.0f, 1.0f, 1.0f));
```

（2）平移

使 cube 沿着水平方向来回移动。

设置水平方向偏移距离为 xDistance，在渲染循环中不断更新偏移值，实现在水平方向上的持续移动。

```
if (toRight == true) {
    xDistance += translateV;
    if (xDistance > 0.8f) {
        toRight = false;
    }
} else {
    xDistance -= translateV;
    if (xDistance < -0.8f) {
        toRight = true;
    }
}
model = glm::translate(model, glm::vec3(xDistance, 0.0f, 0.0f));
```

（3）缩放

使 cube 持续放大缩小。

设置放大倍数为 scaleSize，初始化为 1.0f，在循环中不断更新，实现随时间持续放缩。

```
if (toBigger == true) {
    scaleSize += scaleV;
    if (scaleSize > 2.0f) {
        toBigger = false;
    }
} else {
    scaleSize -= scaleV;
    if (scaleSize < 0.2f) {
        toBigger = true;
    }
}
model = glm::scale(model, glm::vec3(scaleSize, scaleSize, scaleSize));
```

3）创建观察矩阵

我们想要在场景里面稍微往后移动以使得对象变成可见的(当在世界空间时，我们位于原点(0,0,0))。要想在场景里面移动， 将摄像机往后移动跟将整个场景往前移是一样的。

这就是观察空间所做的，我们以相反于移动摄像机的方向移动整个场景。因为我们想要往后移动，并且 OpenGL 是一个右手坐标系(Right-handed System)所以我们沿着 z 轴的负方向移动。我们会通过将场景沿着 z 轴正方向平移来实现这个。它会给我们一种我们在往后移动的感觉。

```
view = glm::translate(view, glm::vec3(0.0f, 0.0f, -3.0f));
```

4）定义投影矩阵

我们想要在我们的场景中使用透视投影所以我们声明的投影矩阵是像这样的：

```
projection = glm::perspective(45.0f, 1.5f, 0.1f, 100.0f);
```

第二个参数用窗口长度/窗口宽度得到，因为选用 1200*800 作为窗口大小，得此处 1.5f。

5）着色器设置

我们创建了转换矩阵，我们应该将它们传入着色器。首先，让我们在顶点着色器中声明

一个单位转换矩阵然后将它乘以顶点坐标：

```
// 顶点着色器源码
#version 440 core
layout (location = 0) in vec3 Position;
layout (location = 1) in vec3 Color;
out vec3 vetexColor;
uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;
void main()
{
   gl_Position =  projection * view * model * vec4(Position, 1.0);
   vetexColor = Color;
};
```

```
// 片段着色器源码
#version 440 core
out vec4 FragColor;
in vec3 vetexColor;
void main()
{
   FragColor = vec4(vetexColor, 1.0f);
};
```

我们应该将矩阵传入着色器(这通常在每次渲染的时候即转换矩阵将要改变的时候完成)：

```
GLint modelLoc = glGetUniformLocation(shaderProgram, "model");
GLint viewLoc = glGetUniformLocation(shaderProgram, "view");
GLint projLoc = glGetUniformLocation(shaderProgram, "projection");
```