计算机图形学 HW5

姓名：朱俊凯

学号：16340315

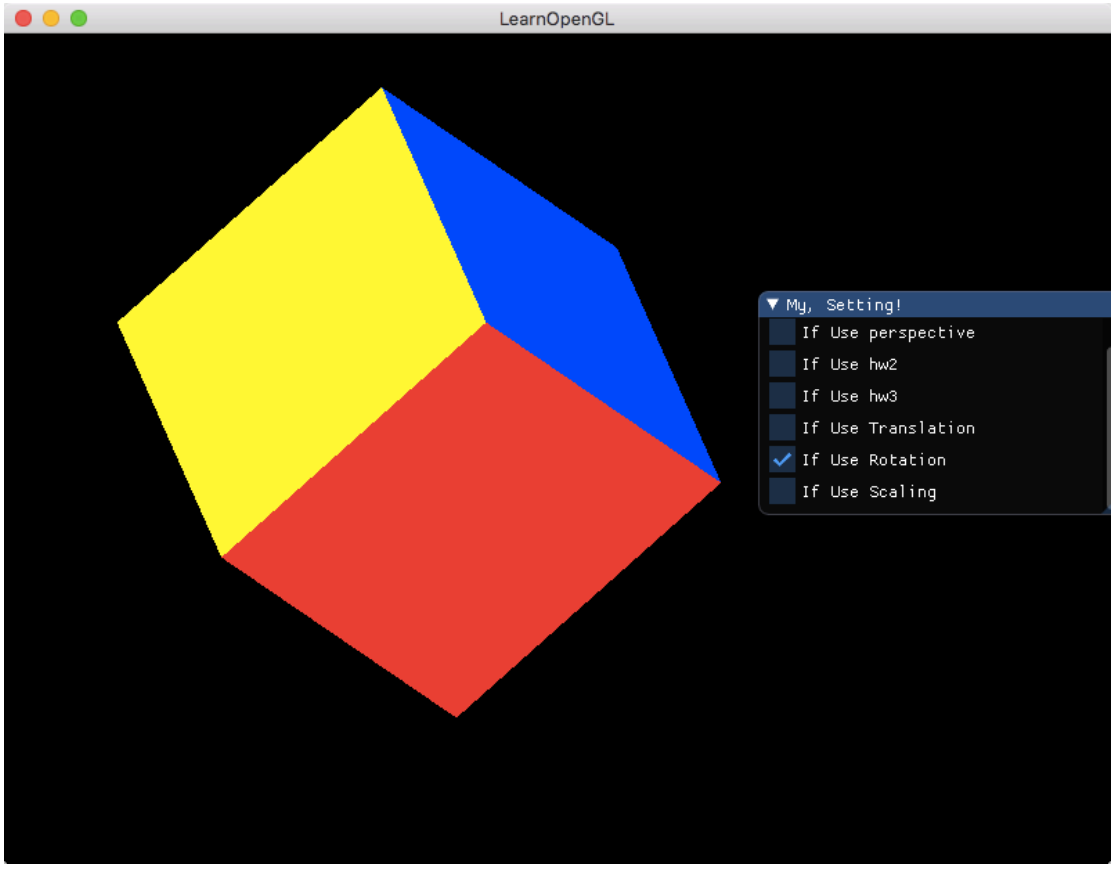
----详细运行截图请看 gif-----

一. 把运行结果截图贴到报告里，并回答作业里提出的问题。

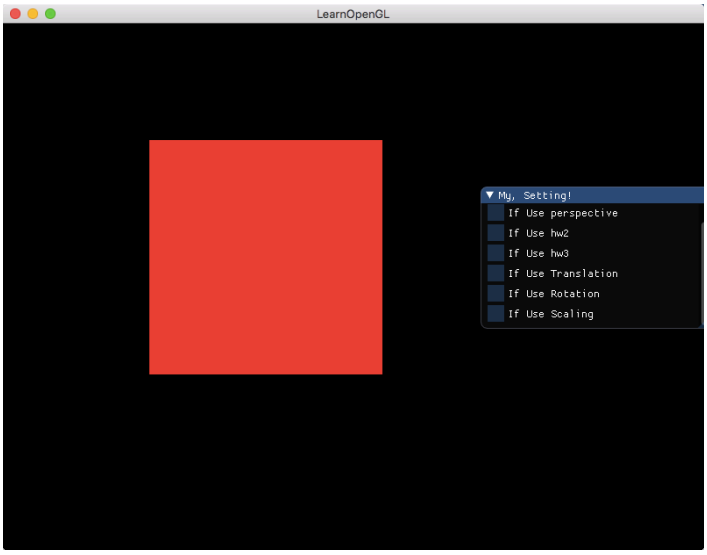
1. 投影(Projection):

下列测试均建立在 view = glm::translate(view, glm::vec3(0.0f, 0.0f, -10.0f))的前提下进行的。

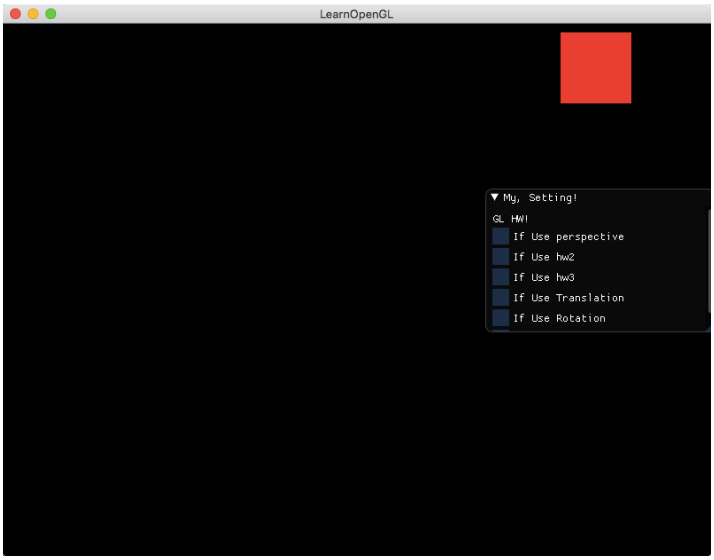
- 把上次作业绘制的 cube 放置在(-1.5, 0.5, -1.5)位置，要求 6 个面颜色不一致



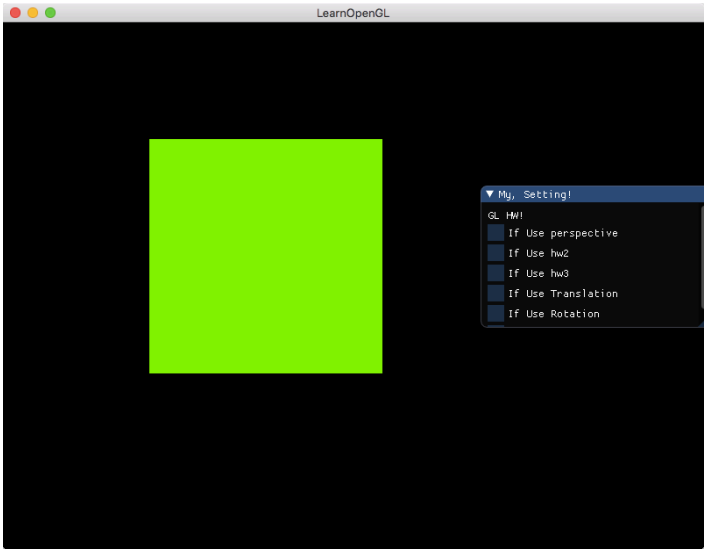
- 正交投影(orthographic projection):实现正交投影，使用多组(left, right, bottom, top, near, far)参数， 比较结果差异
  - 第一组 glm::ortho(-6.0f, 6.0f, -4.5f, 4.5f, 0.1f, 100.0f)



- 第二组 glm::ortho(-35.0f, 5.0f, -27.0f, 3.0f, 0.1f, 100.0f);



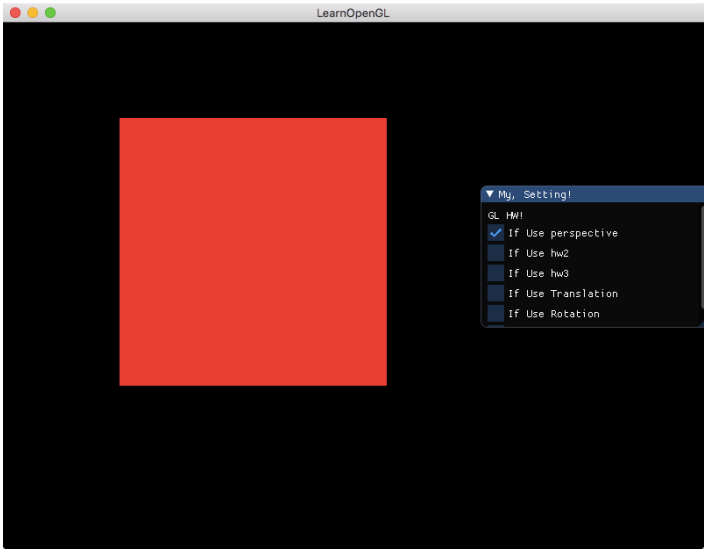
- 第三组 glm::ortho(-6.0f, 6.0f, -4.5f, 4.5f, 10.0f, 100.0f);



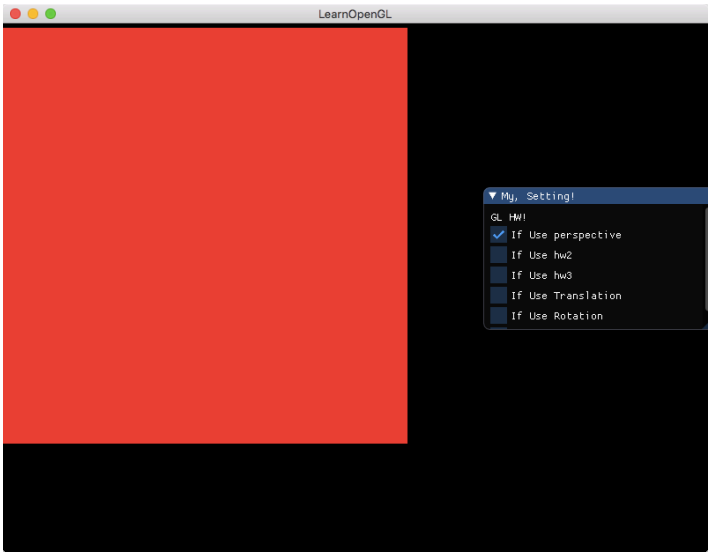
- **结果分析：**

- 关于 left,right, bottom, top 这一组，其实就是描述镜头的左右下上四个边界，并且将图形转化显示到屏幕上。由于显示大小不变，因此修改 left-right 或者 bottom- top 差值范围大小就会使得图像产生缩放的效果。另外同理，修改 left,right, bottom, top 的值，会导致图像相对于边界的距离占边界范围的比例发生改变，从而导致图像的位移效果。
- 关于 near 和 far 这一组,应该是修改能看到的深度距离范围，因此当我修改 near 值后，前面的红面就看不到了，显示出来的就是后面的那个绿面了。

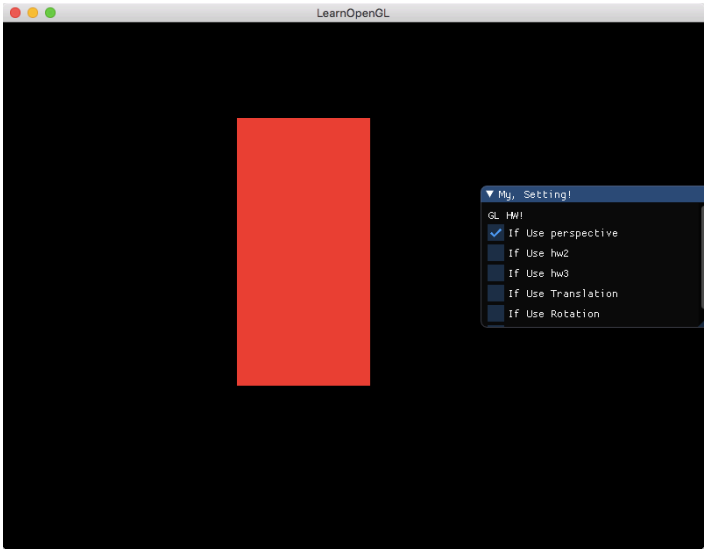
- 透视投影(perspective projection):实现透视投影，使用多组参数，比较结果差异
  - 第一组 glm::perspective(glm::radians(45.0f), (float)800/(float)600, 0.1f, 100.0f)



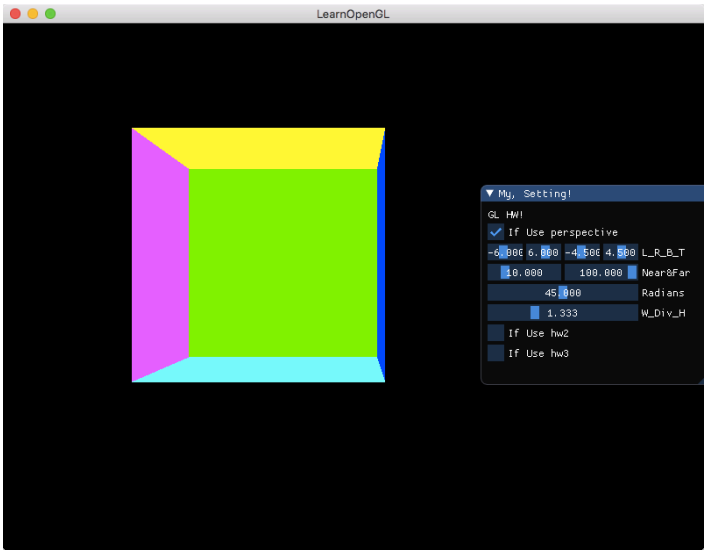
  - 第二组 glm::perspective(glm::radians(30.0f), (float)800/(float)600, 0.1f, 100.0f)



- 第三组 glm::perspective(glm::radians(45.0f), (float)800/(float)300, 0.1f, 100.0f)



- 第四组：glm::perspective(glm::radians(45.0f), (float)800/(float)600, 10.0f, 100.0f);

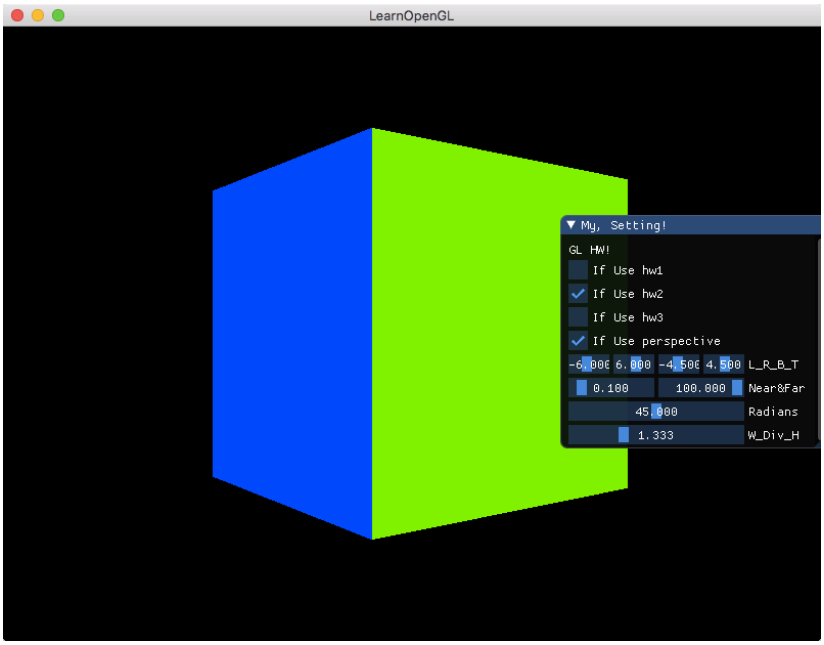


- **结果分析：**
- 关于 radians 这个参数，是一个角度值，主要是涉及视角范围，也就是值越大，看的视野越广，因此将值变小，就会导致图形在视野内占的比例变大，从而在显示上就感觉变大了。
- 关于那个 width/height 得到的宽高比值，就是就是描述这个视野的形状的宽高比，但是屏幕显示大小不变，因此被显示拉伸之后图形就会产生一些宽高的缩放效果，
- 关于 near 和 far 这一组，就和之前的一样，修改值后，前面这个面就看不到了，因此就显示了这个效果。

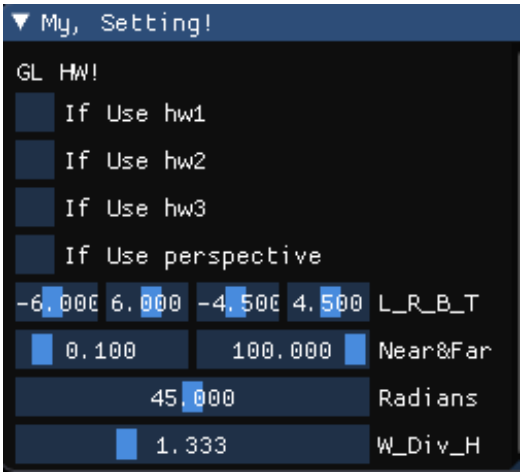
2. 视角变换(View Changing):

- 把 cube 放置在(0, 0, 0)处，做透视投影，使摄像机围绕 cube 旋转，并且时刻看着 cube 中心

  ----详细运行截图请看 gif-----



3. 在 GUI 里添加菜单栏，可以选择各种功能。



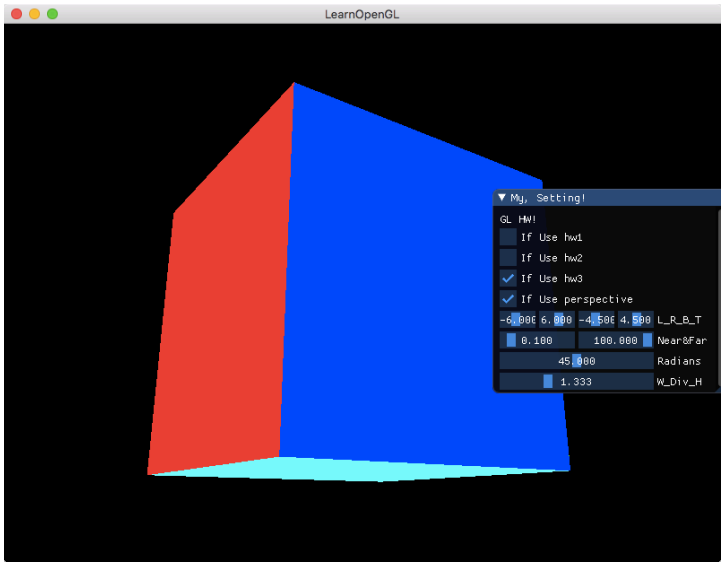
4. 在现实生活中，我们一般将摄像机摆放的空间 View matrix 和被拍摄的物体摆设的空间 Model matrix 分开，但是在 OpenGL 中却将两个合二为一设为 ModelView matrix，通过上面的作业启发，你认为是为什么呢?在报告中写入。(Hints:你可能有不止一个摄像机)

* 因为现实生活中，可能被摄物体会涉及一些难以移动的背景，光线，地形之类的种种因素，因此难以实现用 View matrix 的变化来代替 Model matrix 的变化。但是在 opengl 里面就不太一样，物体和摄像机是完全可动的，而且 View matrix 的变化也可以代替 Model matrix 的变化，产生同样的效果，所以两者可以合二为一设为 ModelView matrix。

Bonus：

.实现一个 camera 类，当键盘输入 w,a,s,d ，能够前后左右移动;当移动鼠标，能够视角移动("look around")，即类似 FPS(First Person Shooting)的游戏场景

----详细运行截图请看 gif-----



二. 报告里简要说明实现思路，以及主要 function/algorithm 的解释。

1.把上次作业绘制的 cube 放置在(-1.5, 0.5, -1.5)位置，要求 6 个面颜色不一致

a.修改 fragmentFramshader，增加一个输入的颜色 uniform vec3 aColor;，使得我们可以动态修改这个颜色，使得每个面渲染时的颜色不一样。

b.之前 transform 位移(-1.5, 0.5, -1.5)即可位置发生改变。

2.正交投影(orthographic projection):实现正交投影，使用多组(left, right, bottom, top, near, far)参数， 比较结果差异

a.修改 vertexShaderSource，添加"uniform mat4 view;\n"和"uniform mat4 projection;\n"，即 view 和 projection 变量。

b.先将 glm::mat4(1.0f)平移到一个合适的位置作为 view 的 mat4, 用 glm::ortho 获得正交投影的 mat4.

c.用 glGetUniformLocation 找到对应字段的位置，用 glUniformMatrix4fv 通过变量 mat4 去修改着色器。

d.修改参数，测试。

3.透视投影(perspective projection):实现透视投影，使用多组参数，比较结果差异

a.这次的 projection 是用 glm::perspective 去生成的透视投影。

b.其他同上，修改参数，测试。

4.把 cube 放置在(0, 0, 0)处，做透视投影，使摄像机围绕 cube 旋转，并且时刻看着 cube 中心

a.根据 reference，使用 sin 和 cos 的方法获得相机位置。

b.使用 lookAt 的方法, 位置为 glm::vec3(camX, 0.0, camZ), 目标为 glm::vec3(0.0, 0.0, 0.0)，即可实现绕着转，看 cube 了。

5.在 GUI 里添加菜单栏，可以选择各种功能。

a.把之前用到的参数改用 ImGUI 修改，每次循环修改即可。

6.实现一个 camera 类，当键盘输入 w,a,s,d ，能够前后左右移动;当移动鼠标，能够视角移动("look around")，即类似 FPS(First Person Shooting)的游戏场景

a.先创造一个类, 主要要有 cameraPos、cameraFront、cameraRight、cameraUp、WorldUp、Yaw、Pitch 这几个变量。

b.构造函数里，要获取 cameraPos，WorldUp，yaw，pitch 等参数，然后，通过这些参数可以生产补全出 cameraFront、cameraRight、cameraUp。

c.move…… （）系列函数就让 cameraPos 加上 cameraFront 或者 cameraRight 乘以一个系数即可。

d.rotate 函数修改 Yaw 和 Pitch, 通过函数生成新的 cameraFront、cameraRight、cameraUp

e.检测键盘输入，并调用对应函数

f.创建鼠标移动回调函数，计算鼠标移动偏移量，再乘上一个系数，xoffset 为 yaw，yoffset 为 pitch，glfwSetCursorPosCallback 函数来绑定这个函数。