张吉祺 16340286

Basic:

1. 投影(Projection):

· 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

修改**HW4_v0**代码，将model矩阵的代码修改为

```
model = glm::translate(model, glm::vec3(-1.5f, 0.5f, -1.5f));
```

使cube移动到(-1.5, 0.5, -1.5)位置。

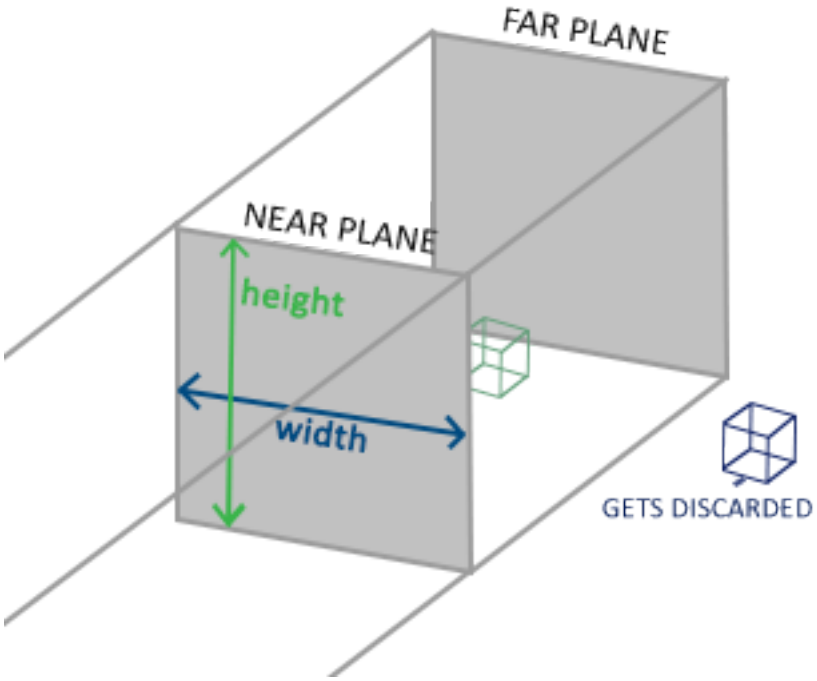
· 正交投影(orthographic projection):实现正交投影，使用多组(left, right, bottom, top, near, far)参数， 比较结果差异

修改projection矩阵为

```
projection = glm::ortho(left, right, bottom, top, zNear, zFar);
```

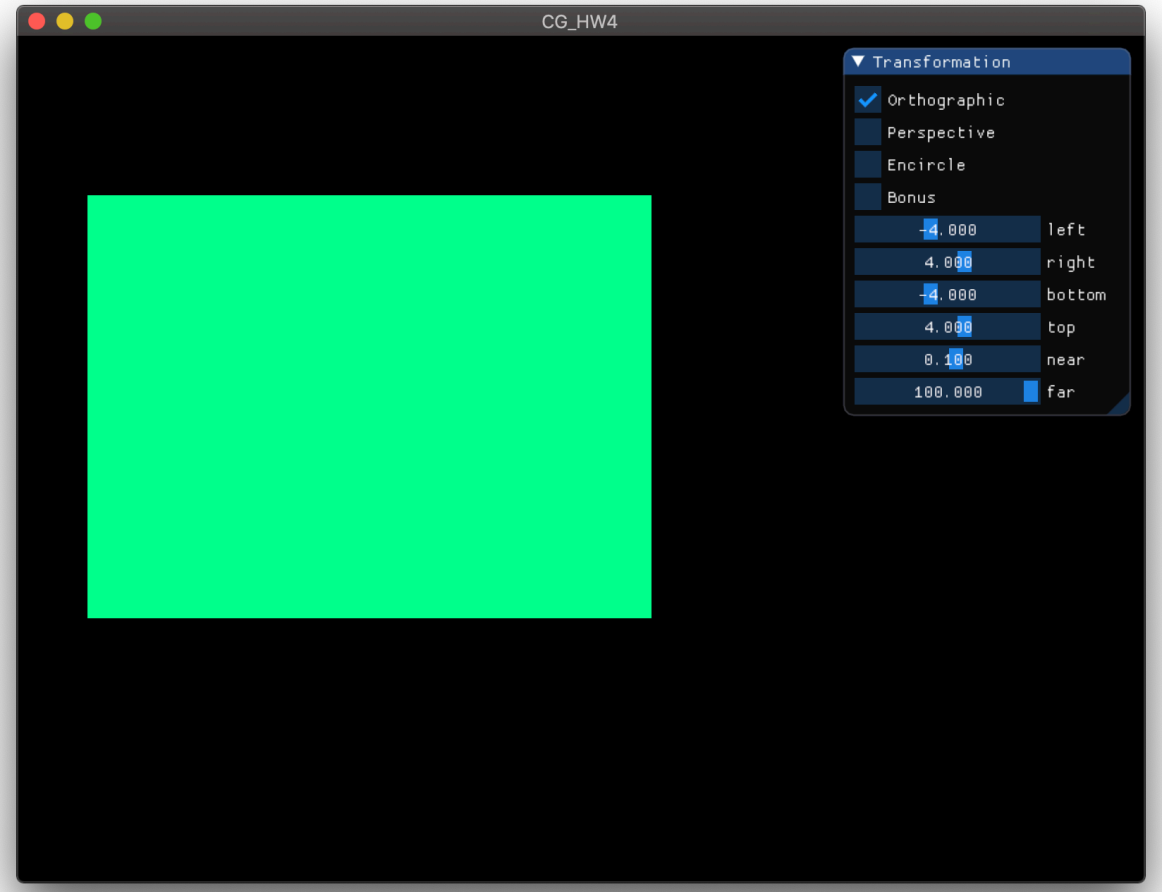
利用`ImGui::SliderFloat`组件调节left, right, bottom, top, zNear, zFar的值。

正射投影矩阵定义了一个类似立方体的平截头箱，它定义了一个裁剪空间，在这空间之外的顶点都会被裁剪掉。



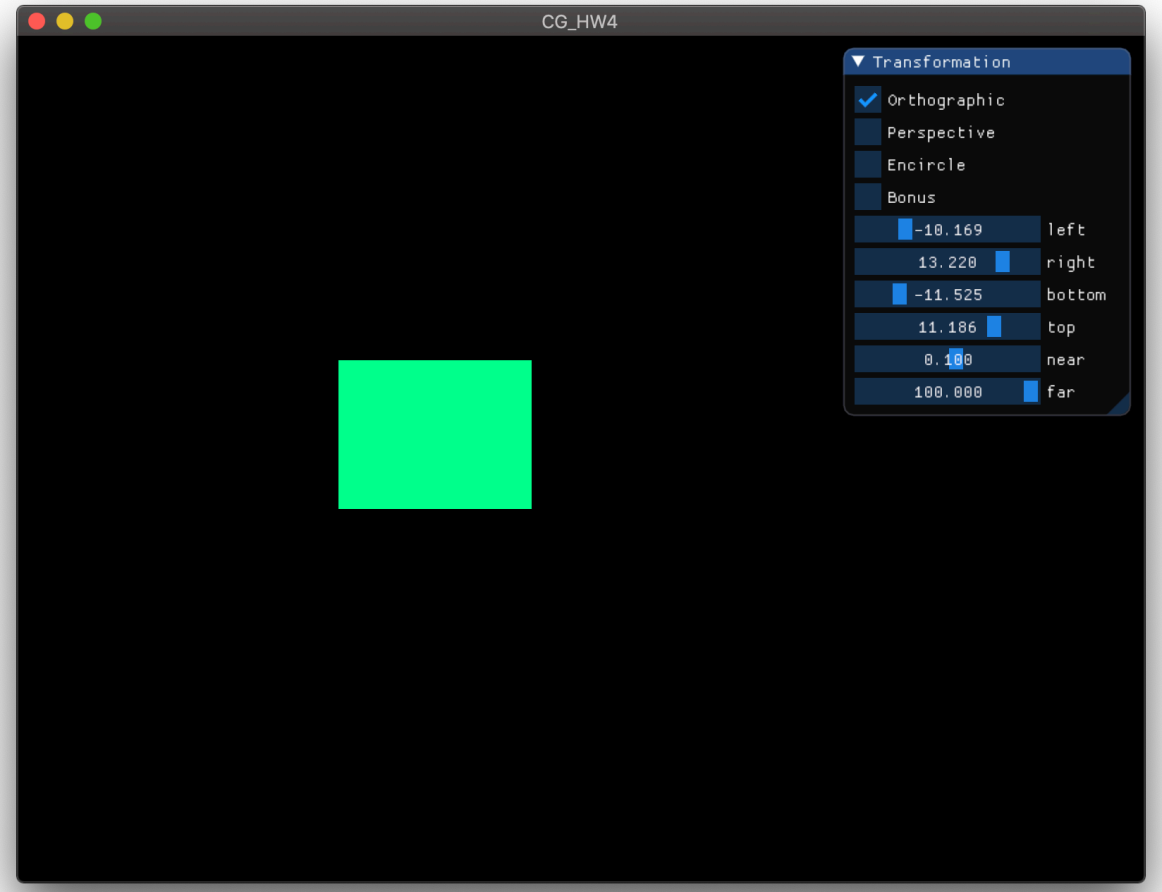
`glm::ortho`前两个参数指定了平截头体的左右坐标，第三和第四参数指定了平截头体的底部和顶部。通过这四个参数定义了近平面和远平面的大小，然后第五和第六个参数则定义了近平面和远平面的距离。这个投影矩阵会将处于这些x，y，z值范围内的坐标变换为标准化设备坐标。

首先设置left = -4.0f, right = 4.0f, bottom = -4.0f, top = 4.0f, zNear = 0.1f, zFar = 100.0f。

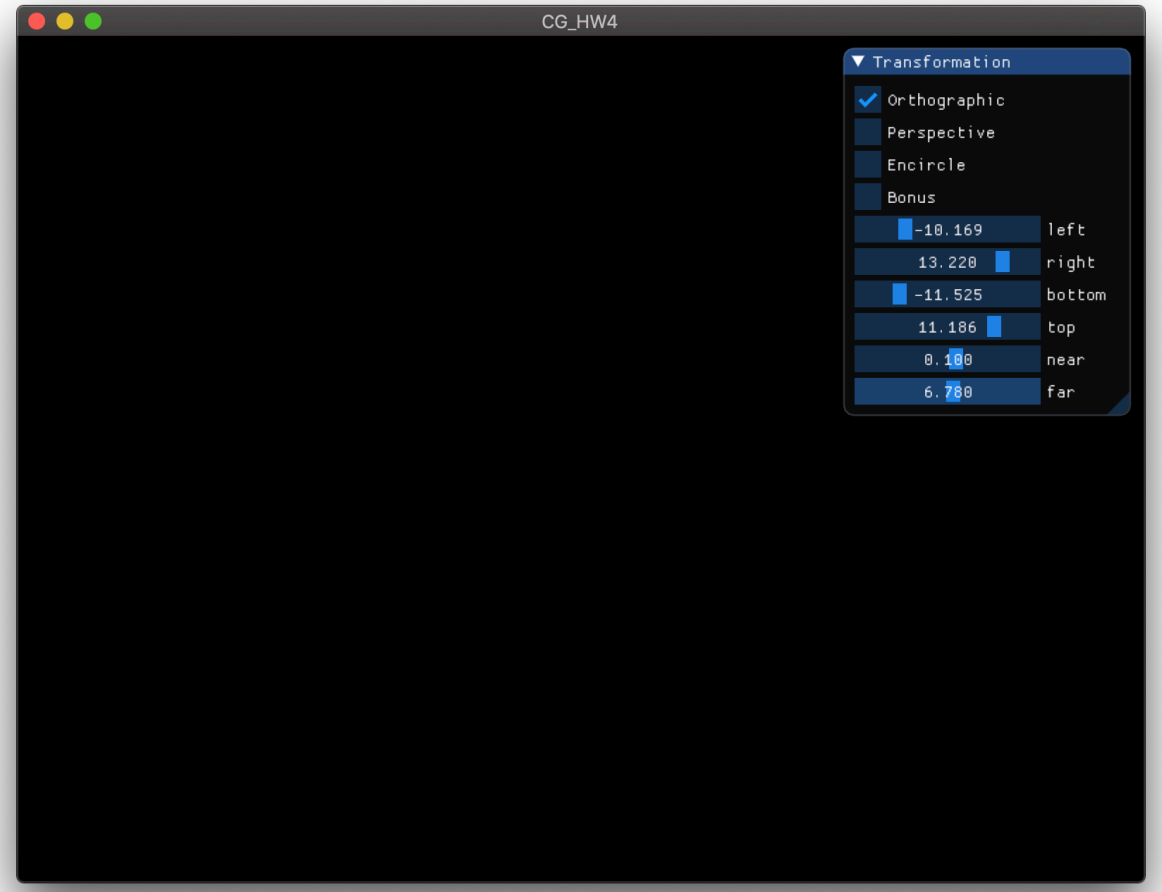


调节这四个值的效果是改变物体显示的大小和位置，如将left调小或right调大或bottom调小或top调大可以使物体的显示变小。

张吉祺 16340286



调节zNear和zFar的值可以使显示的物体消失。



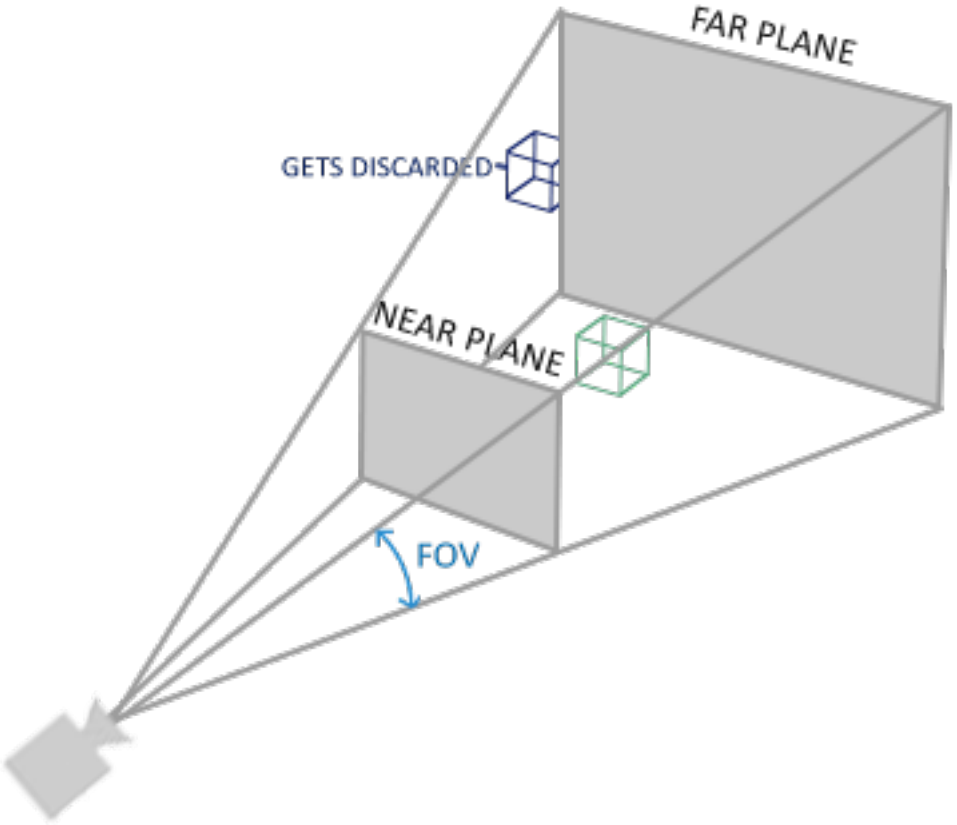
· 透视投影(perspective projection):实现透视投影，使用多组参数，比较结果差异

修改projection矩阵为

```
projection = glm::perspective(fovy, aspect, zNear, zFar);
```

利用`ImGui::SliderFloat`组件调节fovy, aspect, zNear, zFar的值。

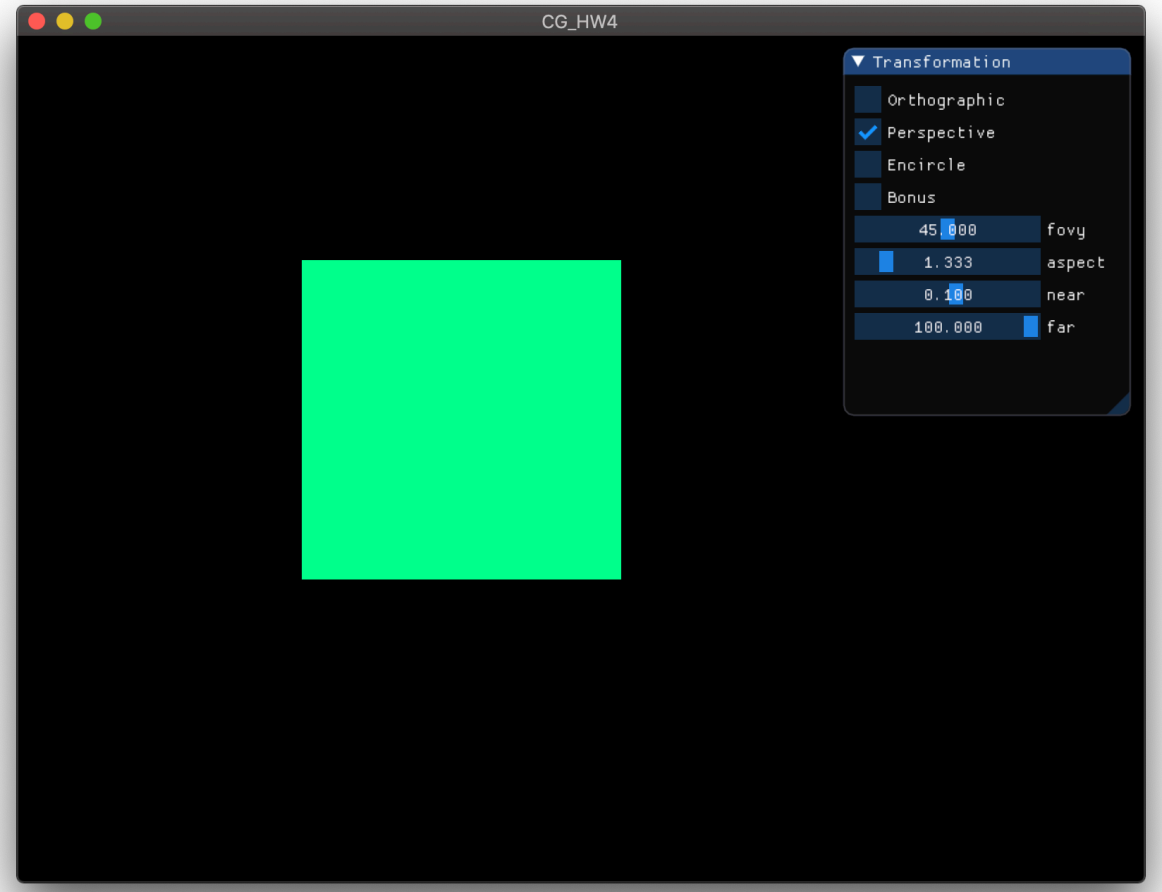
透视投影创建了一个定义了可视空间的大平截头体，任何在这个平截头体以外的东西最后都不会出现在裁剪空间体积内，并且将会受到裁剪。



`glm::perspective`的第一个参数值表示的是视野(Field of View)，并且设置了观察空间的大小。如果想要一个真实的观察效果，它的值通常设置为45.0f，但想要一个末日风格的结果可以将其设置一个更大的值。第二个参数设置了宽高比，由视口的宽除以高所得。第三和第四个参数设置了平截头体的近和远平面。我们通常设置近距离为0.1f，而远距离设为100.0f。所有在近平面和远平面内且处于平截头体内的顶点都会被渲染。

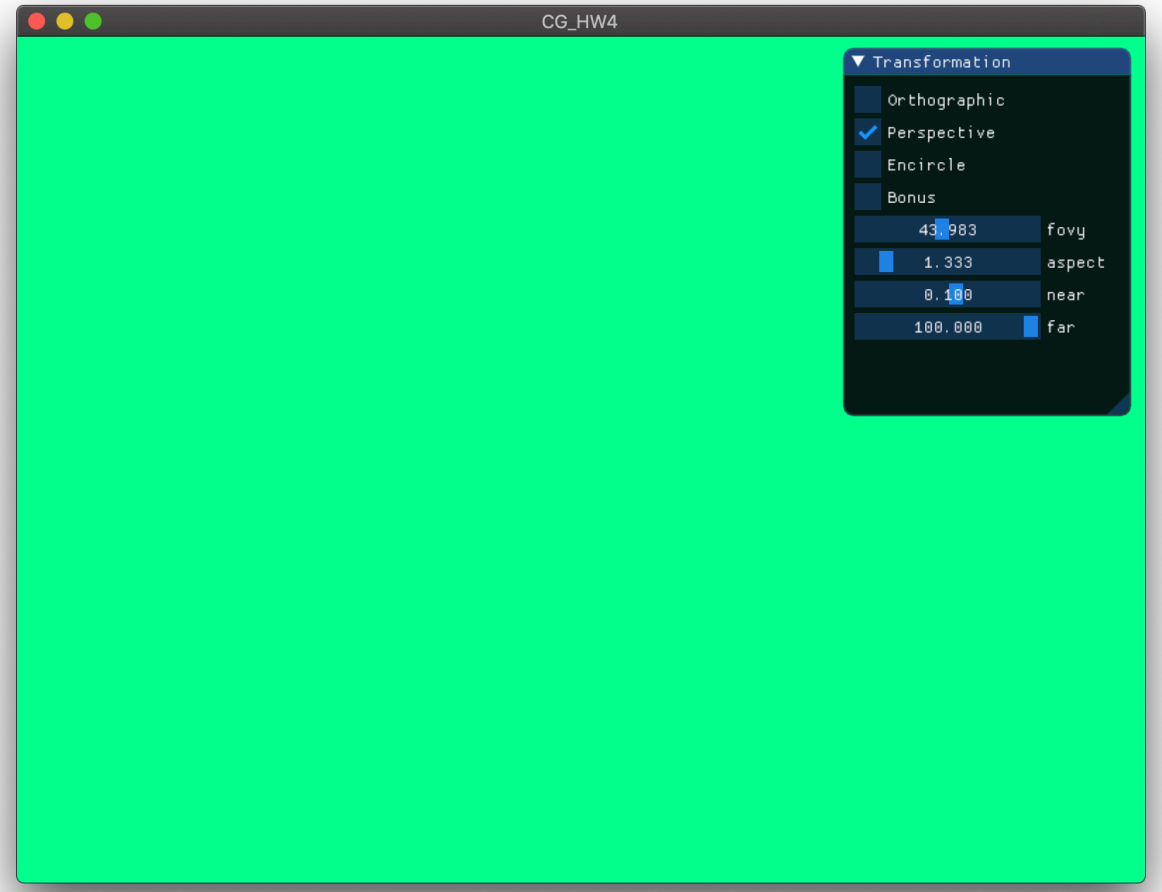
首先设置fovy = 45.0f, aspect = (float)SCR_WIDTH/(float)SCR_HEIGHT, bottom = -4.0f, top = 4.0f, zNear = 0.1f, zFar = 100.0f。

张吉祺 16340286

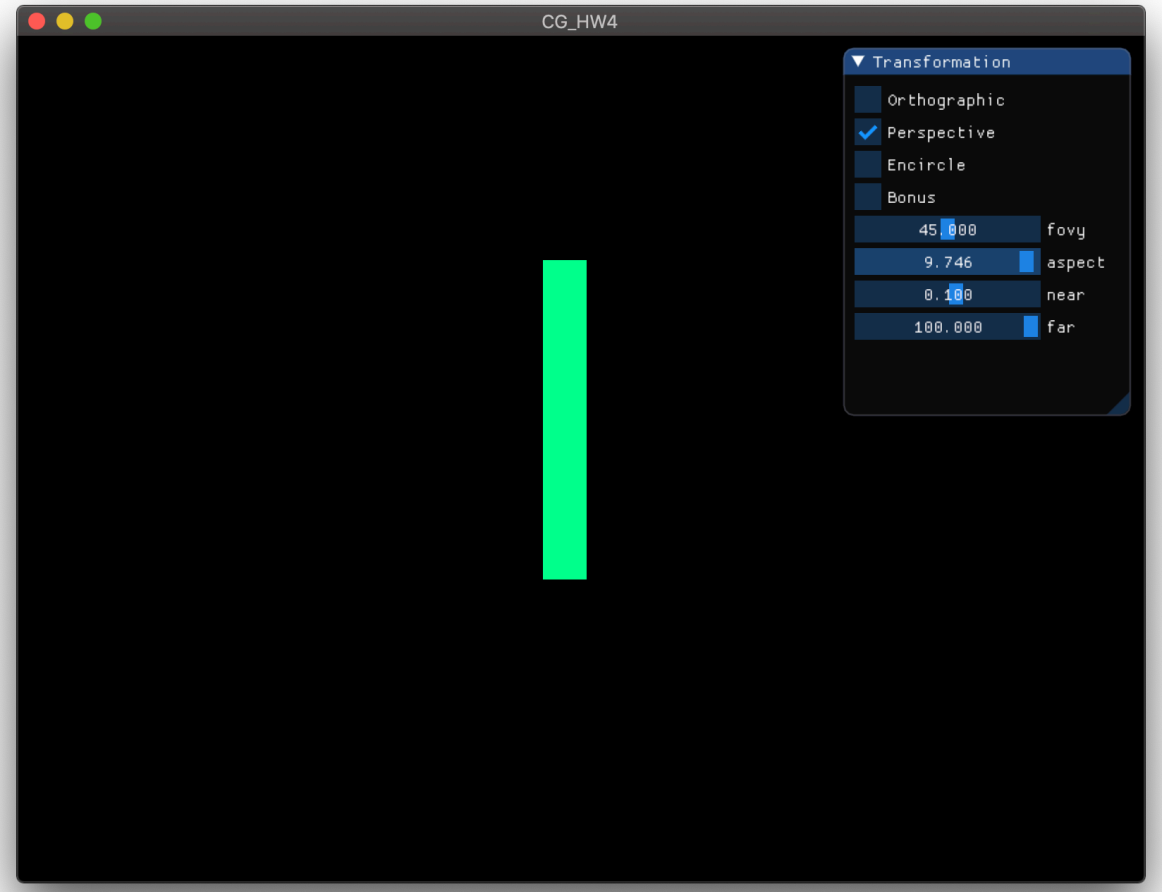


调节fovy的值的过程中物体显示的大小和位置会循环变化。

张吉祺 16340286



调节aspect的值的效果是改变物体显示的长宽比例。

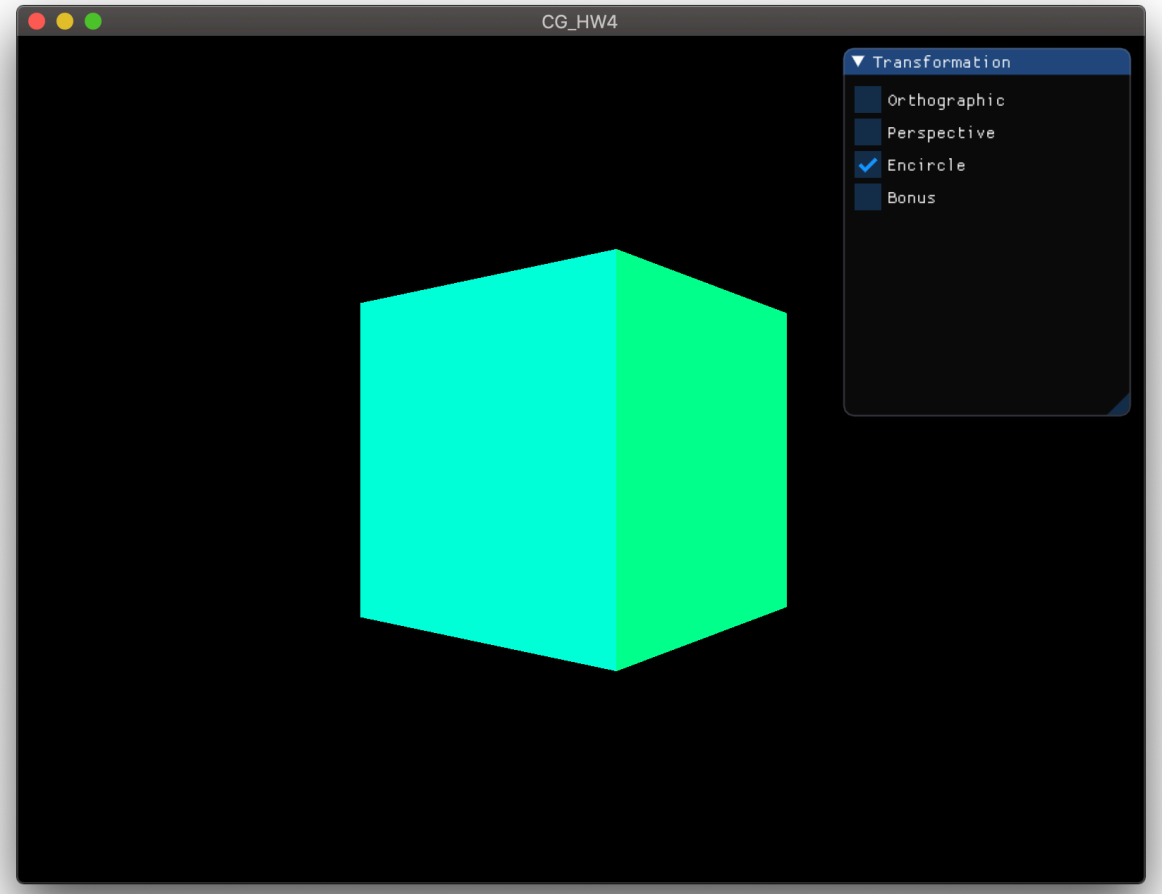


调节zNear和zFar的值效果同正交投影。

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

将model矩阵的代码修改为
```
model = glm::translate(model, glm::vec3(0.0f, 0.0f,0.0f));
```
使cube移动到(0, 0, 0)位置。
将view矩阵的代码修改为
```
float radius = 10.0f;
float camX = sin(glfwGetTime()) * radius;
float camZ = cos(glfwGetTime()) * radius;
view = glm::lookAt(glm::vec3(camX, 0.0f, camZ),
glm::vec3(0.0f, 0.0f, 0.0f), glm::vec3(0.0f, 1.0f, 0.0f));
```
`glm::lookAt`创建摄像机向量，输入为摄像机位置，目标位置和表示世界空间中的上向量的向量。将摄像机位置的y置为 0，x置为sin(time)*radius，y置为cos(time)*radius，time随时间变化，这样就使摄像机的轨道在X-Z平面上是一个

圆，使摄像机围绕cube旋转，将目标位置设置为(0, 0, 0)，使摄像机时刻看着cube中心。



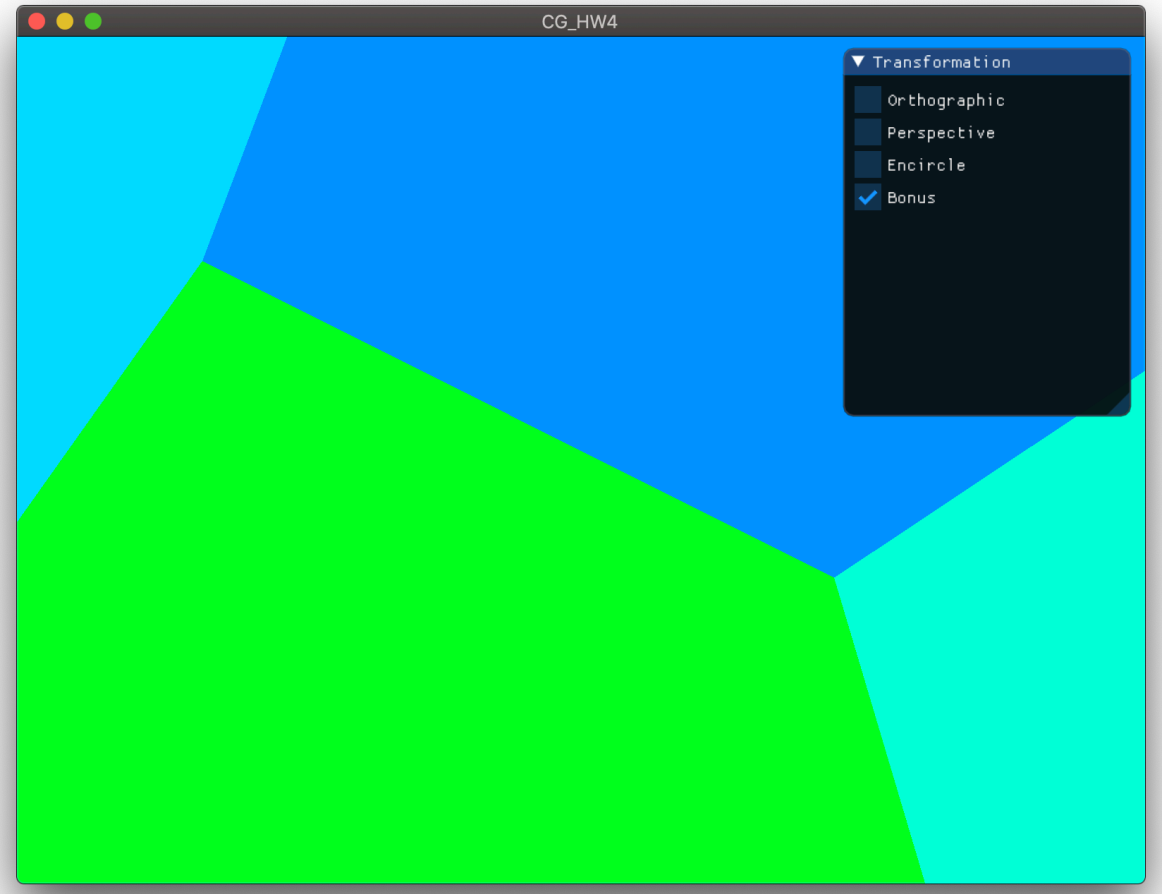
3. 在现实生活中，我们一般将摄像机摆放的空间View matrix和被拍摄的物体摆设的空间Model matrix分开，但是在OpenGL中却将两个合二为一设为ModelView matrix，通过上面的作业启发，你认为是为什么呢？在报告中写入。(Hints:你可能有不止一个摄像机)

在图形编程中，摄像机并不存在。它总是固定在(0, 0, 0)并且朝向(0, 0, -1)，照相机是人造的，它模仿人类观察物体的方式，如移动，转头等等。为了模仿这一点，OpenGL引入了相机的概念。将相机移到右边，还是将场景中的其他物体移到左边，这是一样的，于是将这种不变性转换到模型矩阵上，通过将对象上的所有转换组合在一个ModelView矩阵中。而视图和投影矩阵是分开的，因为这些矩阵做的变换非常不同。一个与模型矩阵非常相似，表示三维空间转换，另一个用于计算观察对象的角度。

张吉祺 16340286

Bonus:

1. 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



每个渲染循环中用`glfwGetKey`获取键盘输入，输入不同的键，调用`Camera`类的不同函数使摄像机的位置向特定方向移动，用`glfwSetCursorPosCallback`并定义回调函数获取鼠标位置，通过当前鼠标位置与上次的鼠标位置的x和y的差，乘以特定比例得到俯仰角和偏航角的角度，并调用`Camera`类的特定函数，改变摄像机的目标位置。再调用`Camera`类的特定函数获取view矩阵。

`Camera`类有属性：摄像机位置`cameraPos`，摄像机朝向`cameraFront`，摄像机上向量`cameraUp`，世界空间上向量`worldUp`，俯仰角`pitch`，偏航角`yaw`。

`Camera`类有方法：`moveForward`，`moveBack`，`moveRight`，`moveLeft`改变`cameraPos`的x轴或z轴。`rotate`方法通过俯仰角和偏航角计算出`cameraFront`，再将`cameraFront`与`worldUp`叉乘得到`cameraRight`，再将`cameraRight`与`cameraFront`叉乘得到`cameraUp`的值并将它更新。`getView`方法返回view矩阵`glm::lookAt(cameraPos, cameraPos + cameraFront, cameraUp)`。