**计算机图形学实验报告**

学号：16340054

姓名：戴馨乐

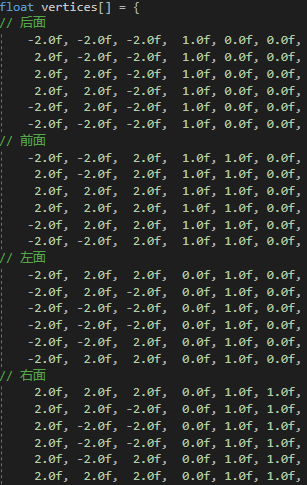
学院：数据科学与计算机学院

作业：第五次作业

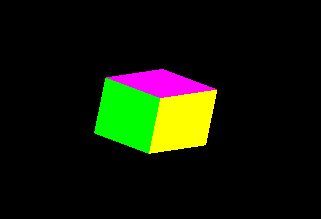
**Basic：**

1. **投影(Projection):**
   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**
      1. 实现思路

这里只需要修改各个面的颜色，然后在model矩阵加入一个平移就好了

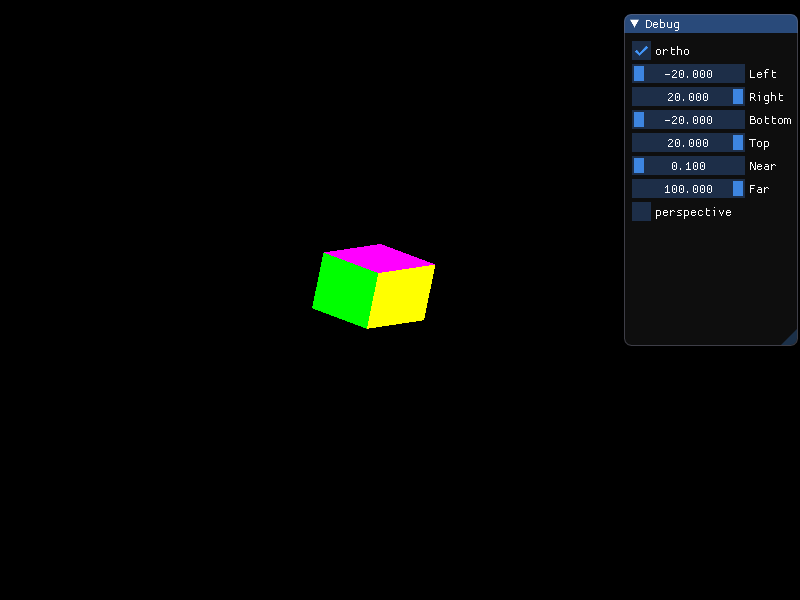
修改各个面的颜色 移动正方体到(-1.5, 0.5, -1.5)位置

* + 1. 运行效果截图



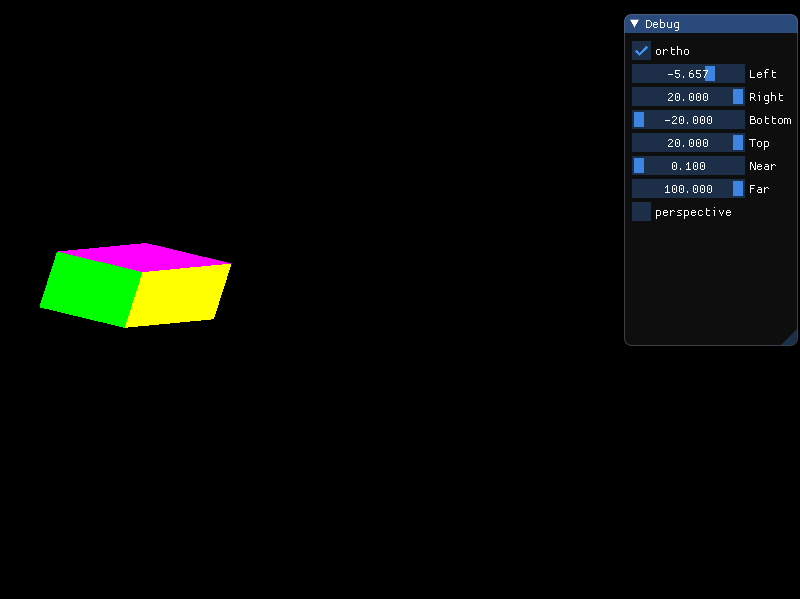
* 1. **正交投影(orthographic projection)：实现正交投影，使用多组(left, right, bottom, top, near, far)参数，比较结果差异**

初始参数为：



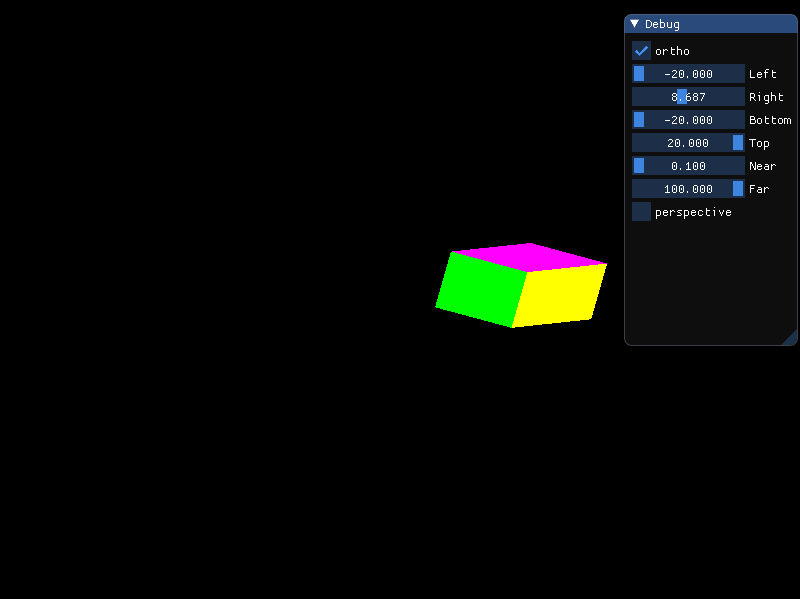
定义了投影平面，我们增大相应的参数，会有向相应方向拉伸的效果；反之，则有压缩的效果。

1. 变化参数，这里增大参数，将变化为，如下：



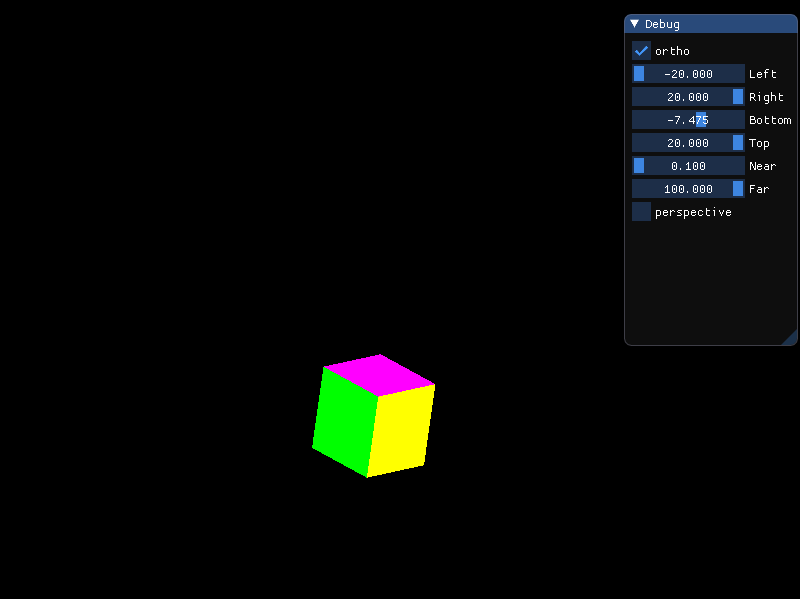
这是因为我们增大了参数，相当于说，定义的平截头体的左平面更加靠近我们的立方体。而且屏幕所展示的，就是平截头体之内的，所以，平截头体的左平面靠近立方体，而我们窗口的大小是不变的，看起来就是立方体更加靠近窗口的左边，就有了上面展示的，物体向左拉伸的效果了。

1. 变化参数，这里缩小参数，将变化为，如下：



这个变化的原因和上述的参数的变化的原因是一样的，参数缩小，则右平面靠近立方体，所以形成了这种物体向右拉伸的效果。

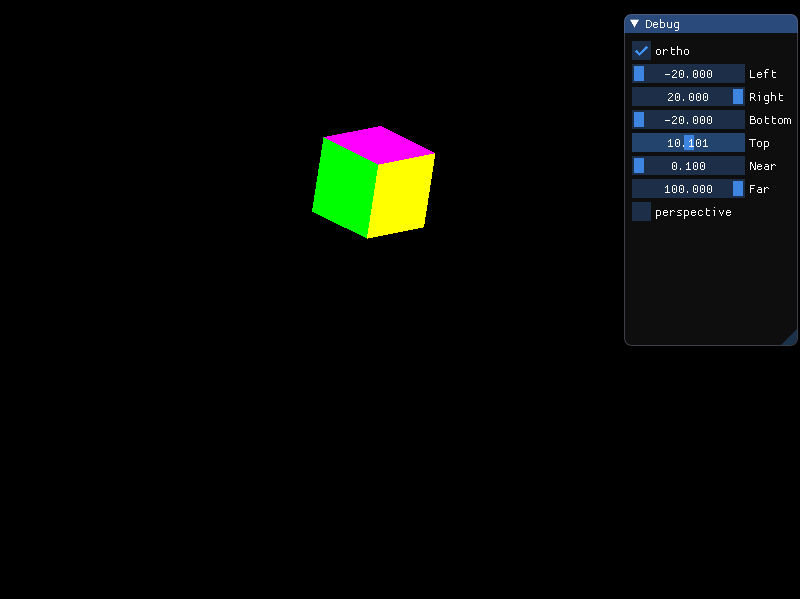
1. 变化参数，这里增大参数为，如下：



可以看到立方体向下拉伸了，理由和上面是一样的。

1. 变化参数，将参数缩小为，如下：

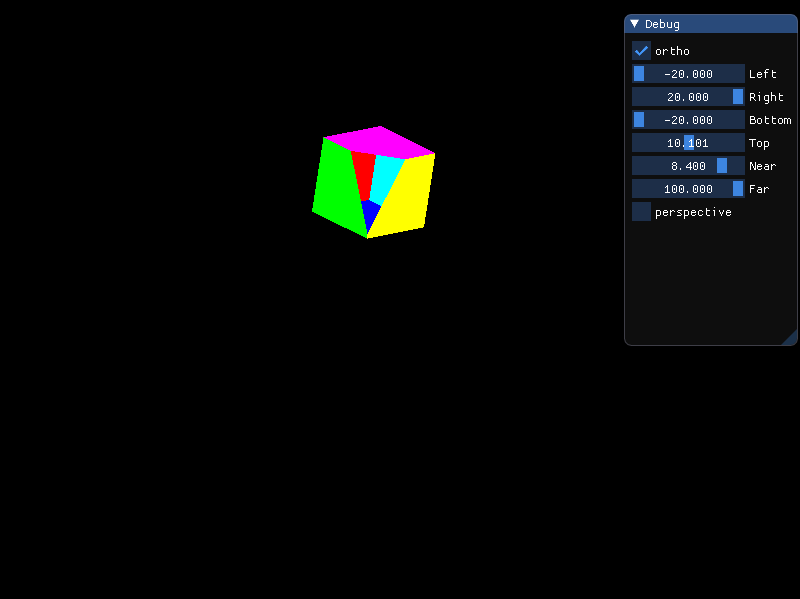
可以看到立方体向上拉伸了，原因也和上面描述是类似的。



1. 变化和参数

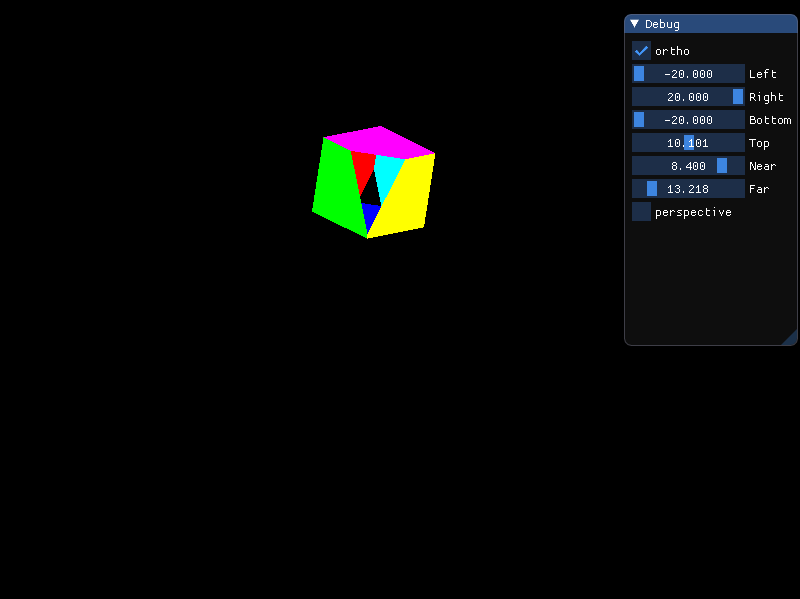
参数代表了平截头体的近平面，参数代表了平截头体的原平面

1. 这里我们先增大参数，将参数增大到，如下：



可以看到，仿佛就是切开了前部的一个角，可以看得到内部后面的3个平面。这是因为增大了参数，相当于将近平面后移。假如近平面后移过了立方体，那么立方体超出近平面的部分就会被丢弃掉，这是因为OpenGL仅仅只保留平截头体之内的部分，所以看起来就是被切掉了一个角这种效果。

1. 然后我们缩小参数，将参数缩小到了，如下：



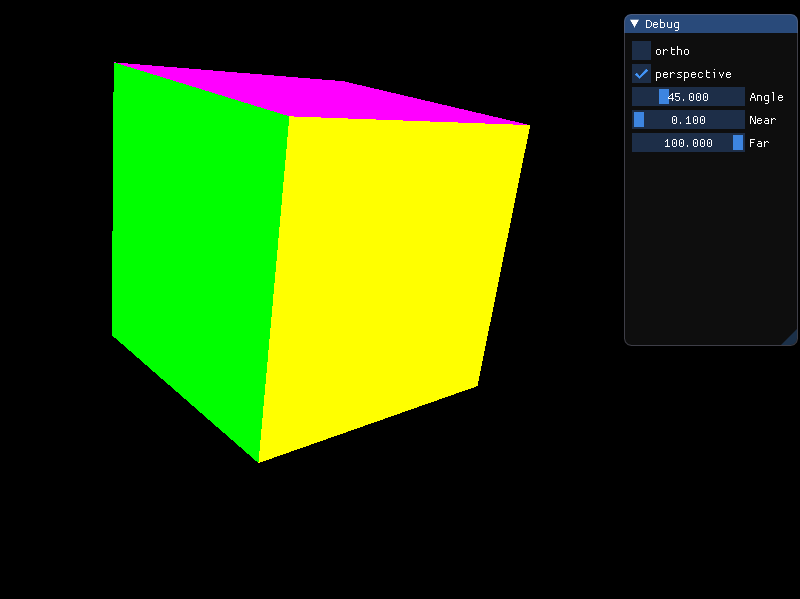
可以看到，在立方体的后部也被切掉了一个角。这是因为缩小参数，相当于远平面向前移，当移动过了立方体，那么在远平面之外的部分就会被丢弃，就有了这种切掉了一个角的效果。

可以看到，正射投影生成的立方体，每个面看起来都是一样近的，和我们平时看到的立方体不一样，它没有透视，点不分远近，就有了上面这种效果。

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

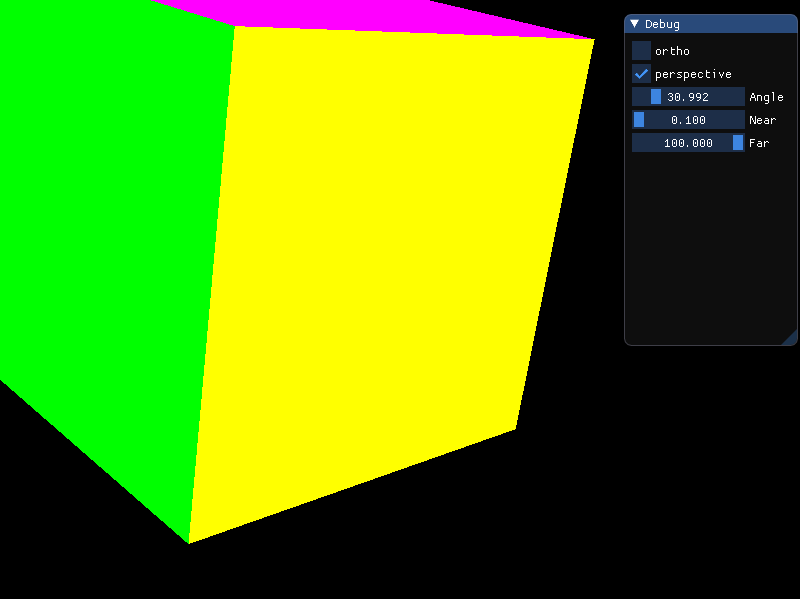
初始参数为：

效果如下：



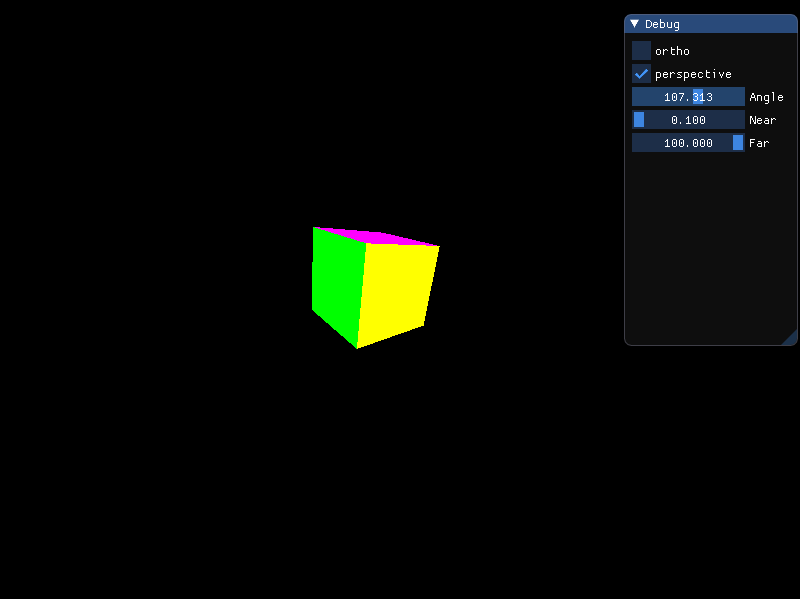
，表示视野，越大，则表示视野越大，那么物体看起来就会变小，反之则会变大。则是宽高比，由窗口的高和宽的比组成。和代表近平面和远平面，和正射投影是一样的。

1. 变化参数
   1. 将变大，将变化为，如下：



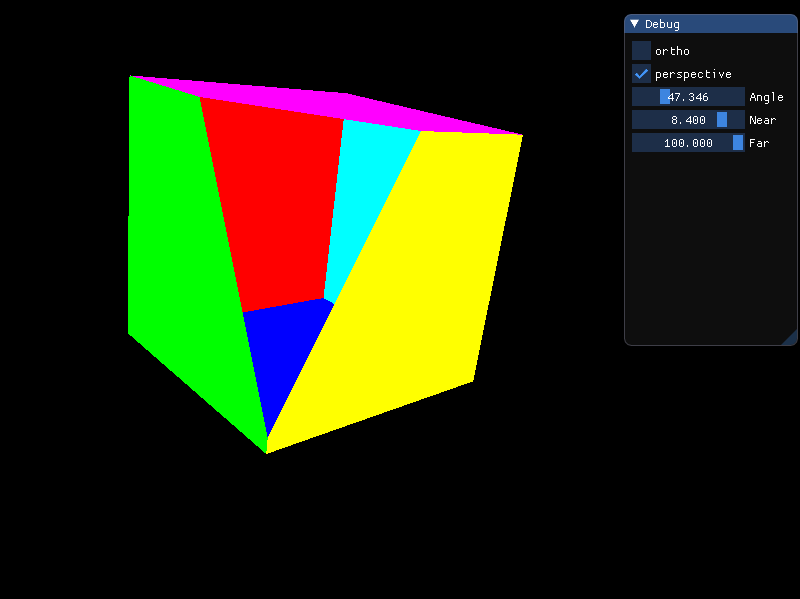
将缩小后，视野变小了，物体离观察则近了，物体看起来更大了

* 1. 将变小，将增大为187.313，如下：

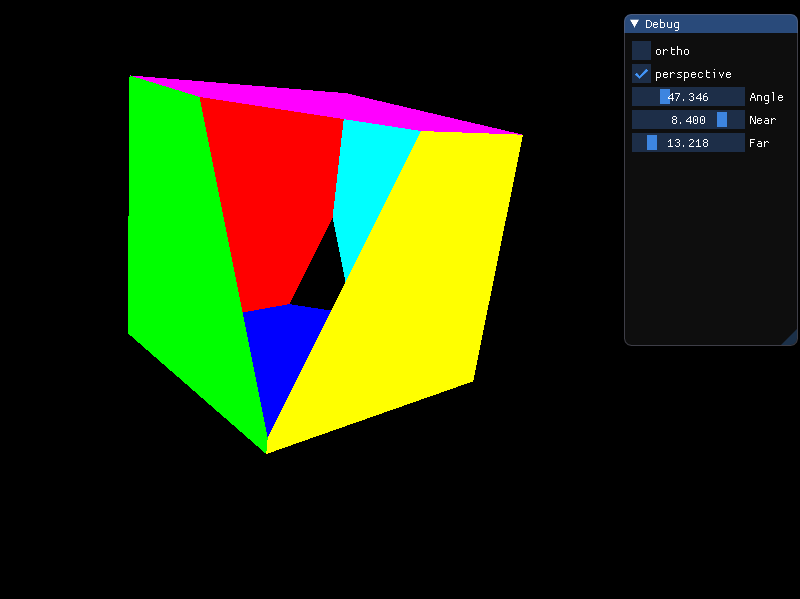


将增大之后，视野变大了，物体看起来变小了

1. 变化和参数
   1. 增大参数，将增大到，如下：



* 1. 缩小参数，将缩小到，如下：



可以看到，这个效果和上面的正射投影是一样的，原因也是一样的。

这里看得到，应用了透视的透视投影，立方体看起来和我们看到的一样了，因为各个点有了远近之分，远的点看起来小，近的点看起来大，更加符合我们视觉看到的。

1. **视角变换(View Changing):**
   1. **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**

摄像机，其实就是我们的观察者角度。为了定义摄像机，首先需要使用一个变化矩阵，可以用这个矩阵乘以任意的坐标向量就可以转换到观察空间。首先我们可以先定义摄像机的位置和摄像机观察的目标，然后由标准化得到的向量，就可以得到了摄像机坐标的轴；为了得到轴，先定义一个上向量，这个和平面一起形成的平面属于摄像机坐标的平面，然后得到正交于这两个向量的标准化向量，就得到了摄像机坐标的坐标；最后，正交于轴和轴的向量，就是轴。最后加上平移向量，就得到了需要的矩阵。可以看到生成矩阵，需要3个向量：

1. 向量，摄像机的位置
2. 向量，摄像机的目标
3. 上向量

库提供了相关的函数：

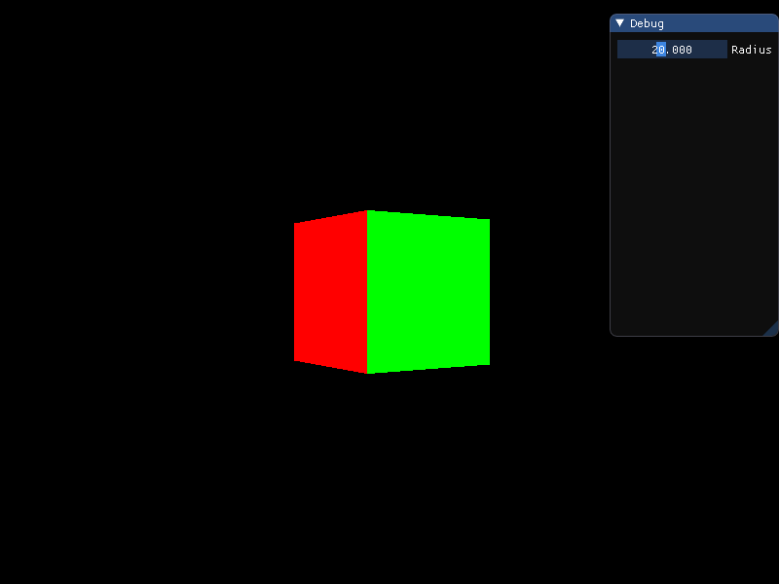
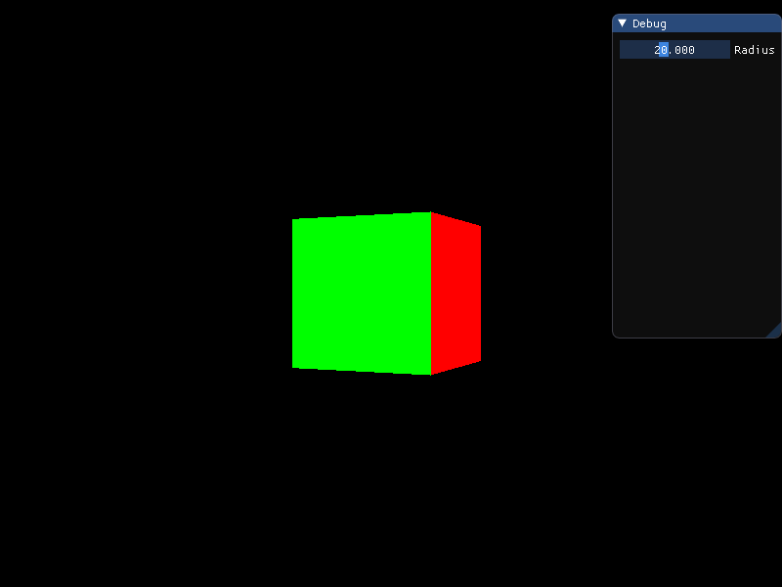


然后，我们需要摄像机沿着一个以立方体的中心，在平面，以一定半径地移动。我们记立方体地中心位置为。

对于摄像机的移动位置，我们用以下公式：

然后需要对进行改变，随着时间进行变化，所以的值，是应该和电脑的时钟得到的，在中，为。



效果截图：

1. **在GUI里添加菜单栏，可以选择各种功能**
2. **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？**

Model矩阵表示物体的摆放位置，View矩阵表示摄像机的观察位置，在固定管线的OpenGL中，将两个矩阵合起来变成一个ModelView矩阵，因为视图其实可以看作物体的摆放改变，视图的改变是物体变换的逆方向变换的，所以将其合二为一。另外，假如存在另外一个摄像机，那么View矩阵可以用于新的摄像机，实现多视角观察。

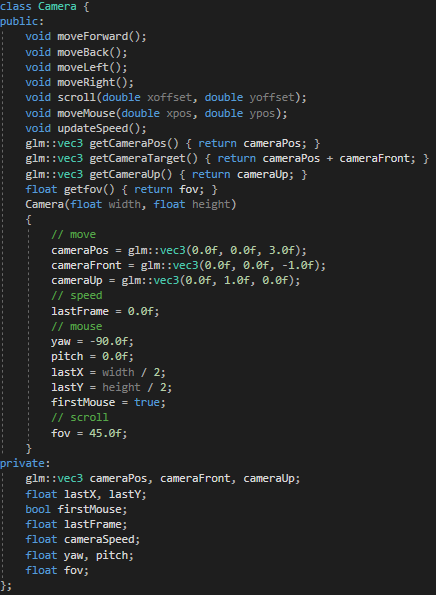
**Bonus：**

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**

首先，对于摄像机的移动，可以封装起来。摄像机主要有以下几种移动：

1. 上下左右移动
2. 视角上下左右变化
3. 对目标物体使用滚轮缩放

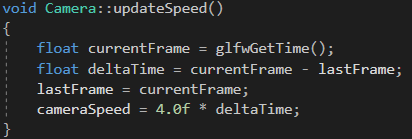
然后，摄像机的移动，其实就是改变上面说的3个向量，和。其中，向量是固定的，就是，然后摄像机的位置之后是可以改变的，所以初始化的值不重要，放在要看的立方体前方比较人性化，这里初始化为。这里摄像机的目标，并不是固定的了，是随着摄像机的移动而变化的，但是以人为例，摄像机看向的目标，就是人的正前方，相对于人的正前方，我们用这样一个移动向量来表示，初始化为。

然后可以大致定义出下面这样一个类：

另外，其余变量是后续几个移动函数用到的。

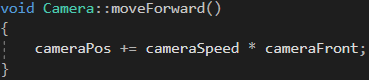
1. 上下左右移动

我们先定义一个摄像机的速度。为了使得摄像机在各个系统下，移动速度都一样，我们通过上下帧的时间差来求出移动速度。记上一帧为，当前帧为，那么可以得到，最后让乘上一个常量，就得到了移动的速度了。这里我定义这个常量C为。



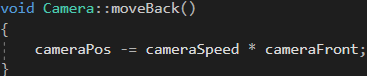
* 1. 向上移动摄像机

向上移动，就是让摄像机的位置加上移动的距离，记移动速度乘上移动向量



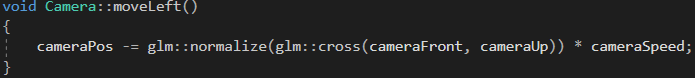
* 1. 向下移动计算机

向下移动，和向上移动类似，只不过是减去移动的距离表示后退



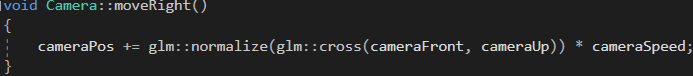
* 1. 向左移动计算机

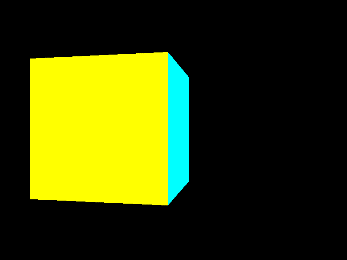
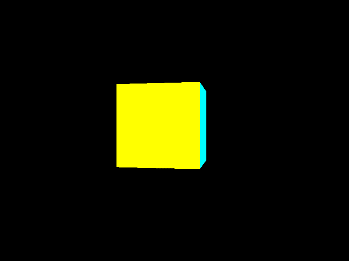
向左移动的移动向量，可以通过求向前的移动向量和向量的正交向量来求得。同样，也是求出移动的距离之后，摄像机的位置就减去移动的距离。



* 1. 向右移动计算机

向右移动和向左移动一样，只不过是加上移动的距离。



 效果如下：

1. 移动摄像机的视角

可以将视角看作一个飞机，可以上下抬头，这就是**俯仰角**；可以左右摆动，这就是**偏航角**；可以翻滚，这就是**滚转角**。目前只需要考虑的是俯仰角和偏航角。

通过三角函数推导，计算方向向量可以通过下面的公式得到：

为了计算俯仰角和偏航角，我们需要记录：

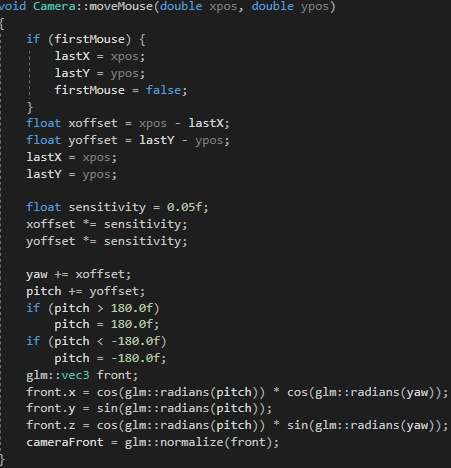
1. 当前鼠标的位置
2. 上一次鼠标的位置

然后计算方向的偏移：

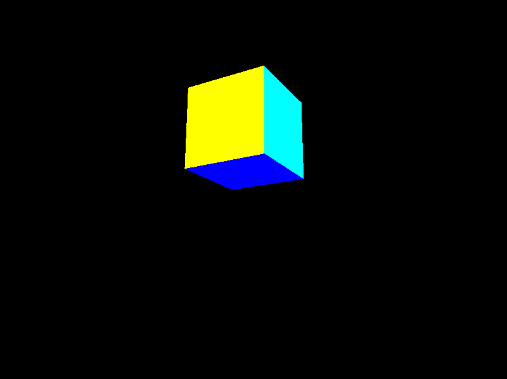
为了使得移动不要过于敏感，这个偏移量常常需要乘以一个常数

然后，俯仰角是在平面左右移动，变化的是，所以，偏航角在平面上下移动，变化的是，所以

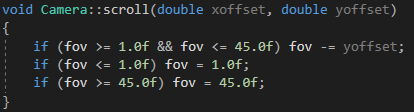
最后使用计算得到的和来计算方向向量就可以了，计算完成后，将这个新的方向向量赋值给原先的移动向量，这样就改变了视角的方向。

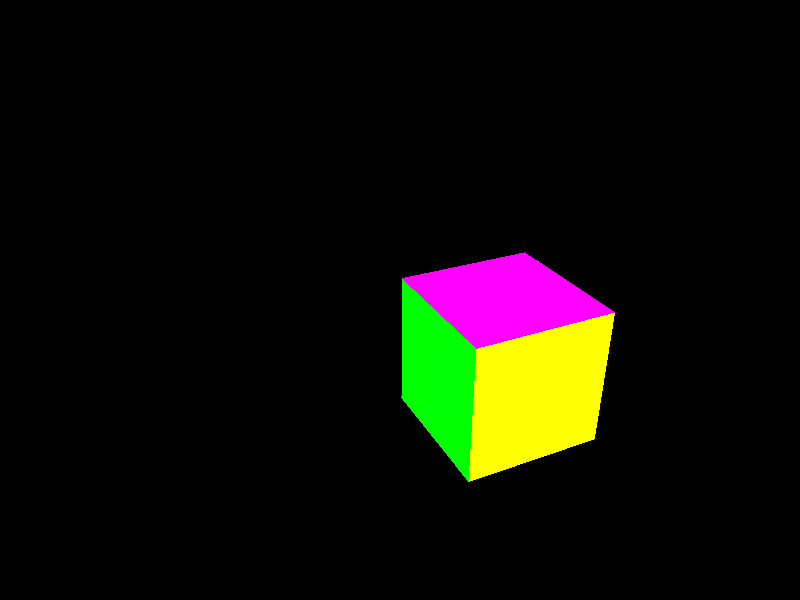
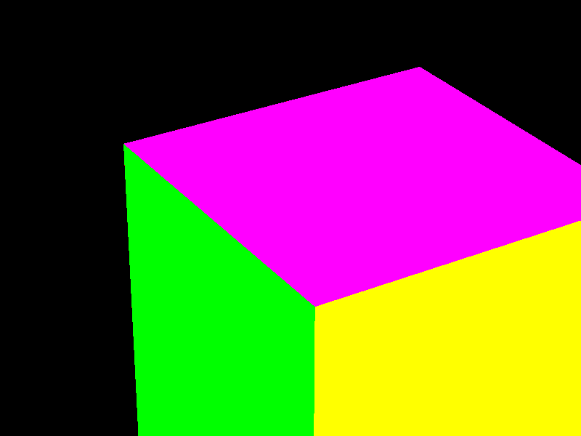


效果如下：

1. 缩放

缩放调整的是投影矩阵的视野，缩小即增大，视野变大；放大即减小，视野变小。



效果如下：

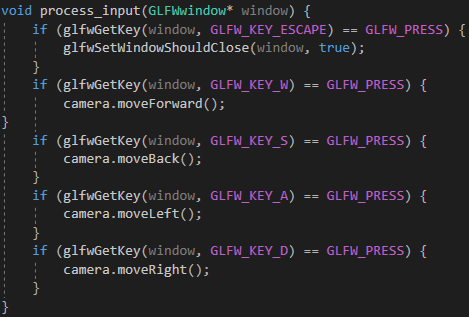
当然，上面仅仅只是实现了类，还需要应用在程序中才可以有上面的效果

1. 首先，新建一个类



然后，对于上下左右移动，是通过输入wsad来进行移动的，那么，需要OpenGL来监听来自键盘的输入，然后根据这4个键是否按下，来决定相应的移动。

监听函数：

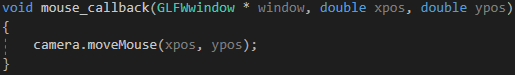


在渲染循环中，不断调用该函数实现监听

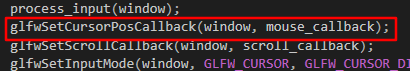


1. 对于鼠标移动的监听函数以及该函数的注册

对于鼠标的监听函数：



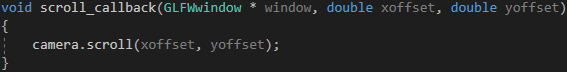
对该函数的注册：

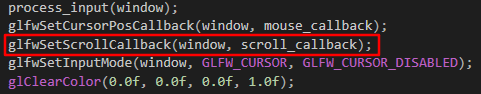


为了实现更加真实的FPS效果，可以将窗口模式设置为鼠标隐藏模式。



1. 滚轮来控制缩放的监听函数和注册函数也是一样的





1. 这里的确是修改了摄像机的相关向量，但是还需要让观察矩阵和投影矩阵来使用，才可以达到修改了摄像机和物体的效果。

