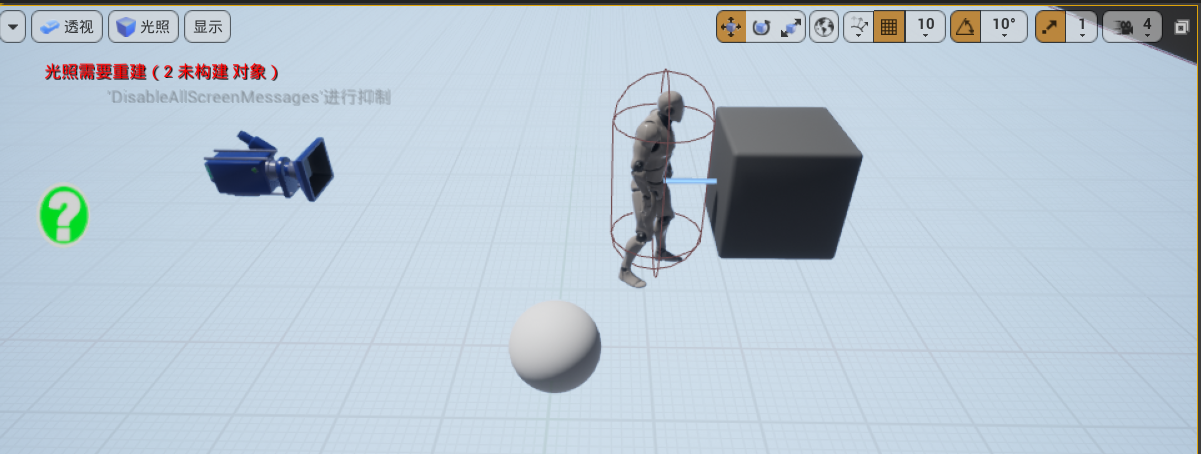
Model Matrix View Matrix Projection Matrix (/W) NDC与窗口坐标的线性关系

x, y, z(模型坐标) --->x,y,z(世界坐标)---> x, y, z(以相机为原点的坐标)---->x, y, z(剪辑坐标)--->x, y, z(标准化设备坐标)--------->x, y(屏幕坐标)

这是第三人称的相机位置，屁股后面一点，挪到眼睛那就是第一人称了。



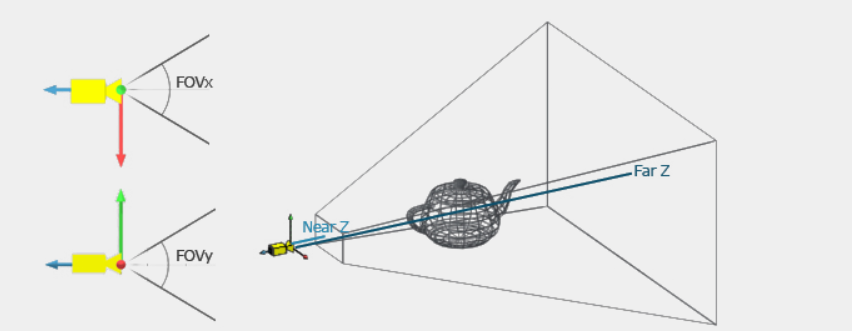
ModelMartix是用来将模型坐标转化为世界坐标的，一般来说没什么用，引擎会给你搞好的，游戏内的矩阵也不会将这个乘上。

实际上很多教程说的MVP矩阵(model-view-projection)中的Model Matrix是针对某一个点来说的，不具有任一点的通用性。

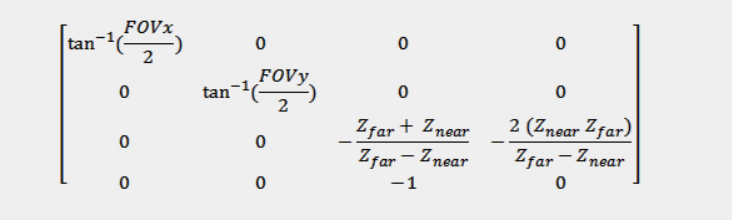
一般来说Matrix有两种，一种是正交矩阵，一种是投影矩阵，3D游戏肯定都是投影矩阵的。

区别是3D游戏的摄像机是跟着人动的，2D的不会动，w分量(摄像机与物体的距离)不会变，自然物体大小不会随着人的移动距离而改变。

下面是投影示意图，



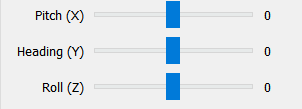
这是投影矩阵公式，怎么推的可以看最后面的链接，有篇相关矩阵公式的数学推导过程。



比如CSGO的zNear是7,zFar是28300左右好像（在源引擎代码里可以找到，你也可以到游戏里直接找）,FOVx一般人是90，我的电脑上是106.7,FOVy可以根据屏幕或者游戏窗口纵横比转换。

实际上游戏中存储的矩阵是View Matrix和Matrix Projection的乘积。因为这两个是对不同玩家来说最通用的，只跟摄像机的位置(也就是人物眼睛的位置)和游戏本身的fov还有投影距离有关，跟你具体的窗口大小无关。

OpenGl世界坐标系和摄像机坐标系都是右手坐标系，Direct3D都是左手坐标系，摄像机的Z轴是人物眼睛看的视角。

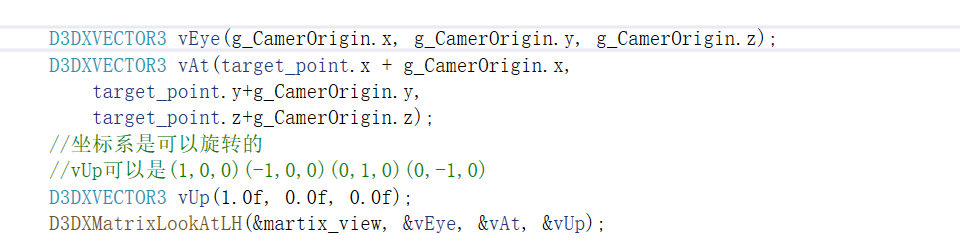


ViewAnlges x 是pitch ，y是yaw z是roll，OpenGl和D3d一般都这样，如果换了的话，计算的时候也要换一下。

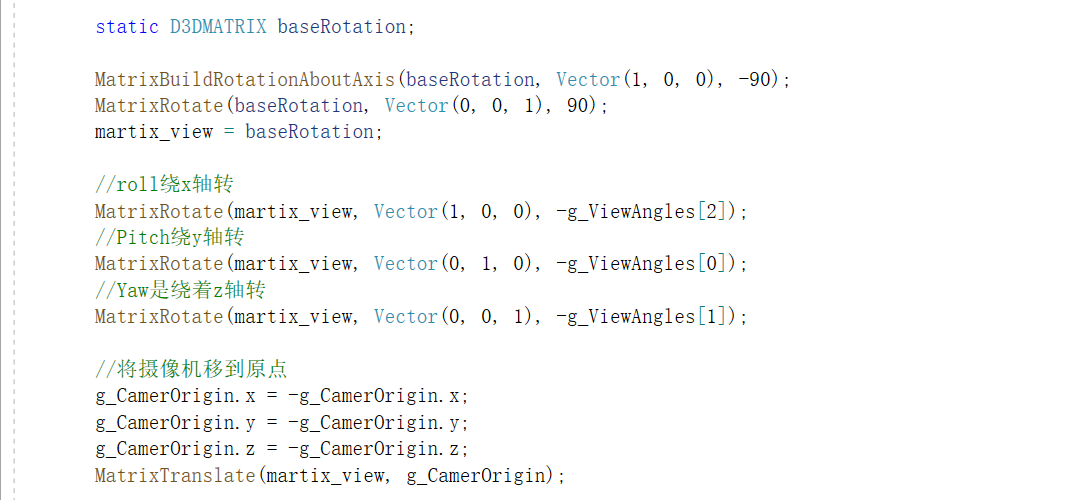
然后用代码验证一下，

设置View Martix

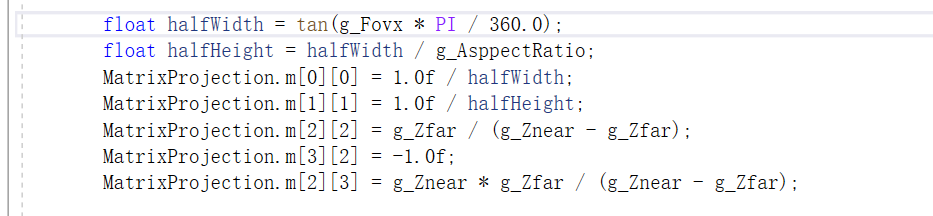
本来我是这样的，发现不行，不知道哪里有问题



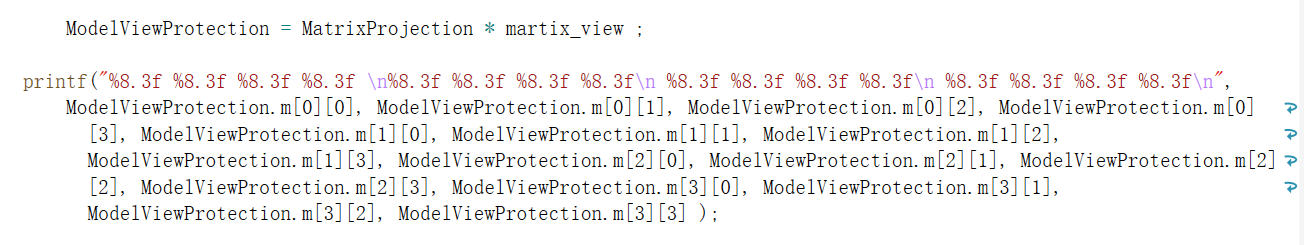
后来还是一步步来，



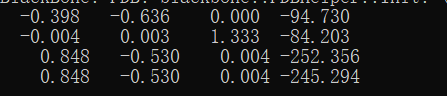
依据公式设置Projection Matrix ,g\_AsppectRation是窗口纵横比，我是全屏游戏的，并且显示器是1920\*1080，纵横比就为1.777

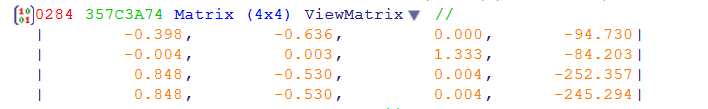


将两矩阵相乘并输出，



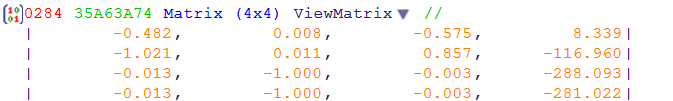
然后我们可以计算出跟游戏内一摸一样的矩阵，



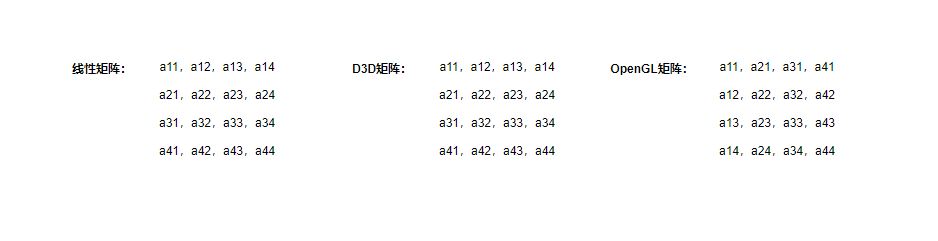


将你游戏中的3D坐标乘矩阵后是获得剪辑（clip）坐标，根据w分量（正交矩阵算出来的w分量一直是1）和NDC坐标的线性关系就能获得屏幕坐标。

矩阵的第三个一般都是0的，如果你能上下转起来（歪头不知道行不行），就不是0了。



Direct3D和OpenGl存储矩阵的方式不一样，OpenGl和Matlab一样列存储，转置一下就行了。



对理解矩阵和找矩阵有点帮助，实际作用应该不大。

如果一些名词不知道的话看下面4个链接。推导公式和具体细节基本都在里面。

希望能帮到各位挂哥

https://www.songho.ca/opengl/gl\_transform.html

https://zhuanlan.zhihu.com/p/63275902

http://www.codinglabs.net/article\_world\_view\_projection\_matrix.aspx

https://www.cnblogs.com/meteoric\_cry/p/7987548.html

https://www.andre-gaschler.com/rotationconverter/