在前面的课程中，我们通过简单地忘记z坐标来渲染我们的模型。今天的目标是学习如何透视画:



//: 本节主要过程

**ModelView 模型变换和视图变换(相机)， 分别处理模型和相机的位置和朝向**

**Projection 投影变换**

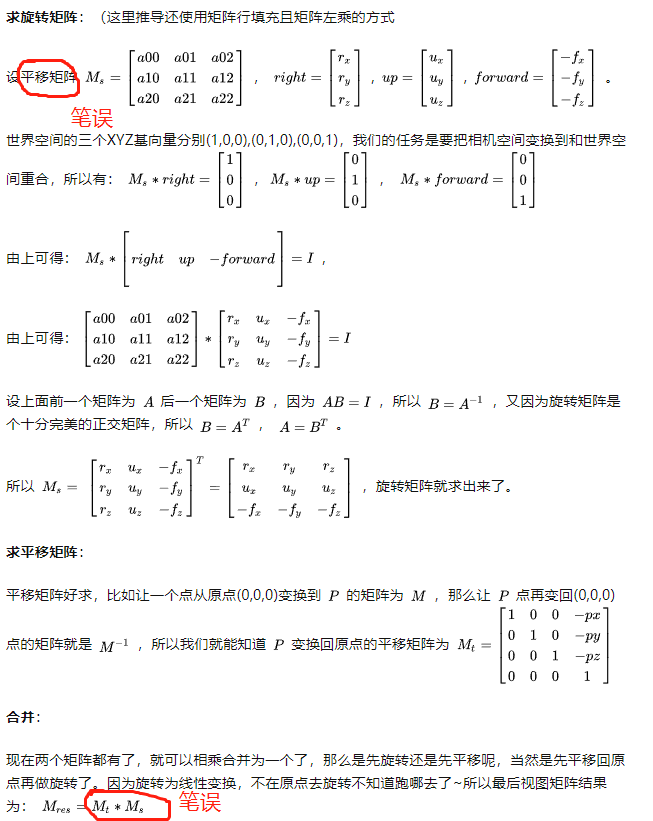
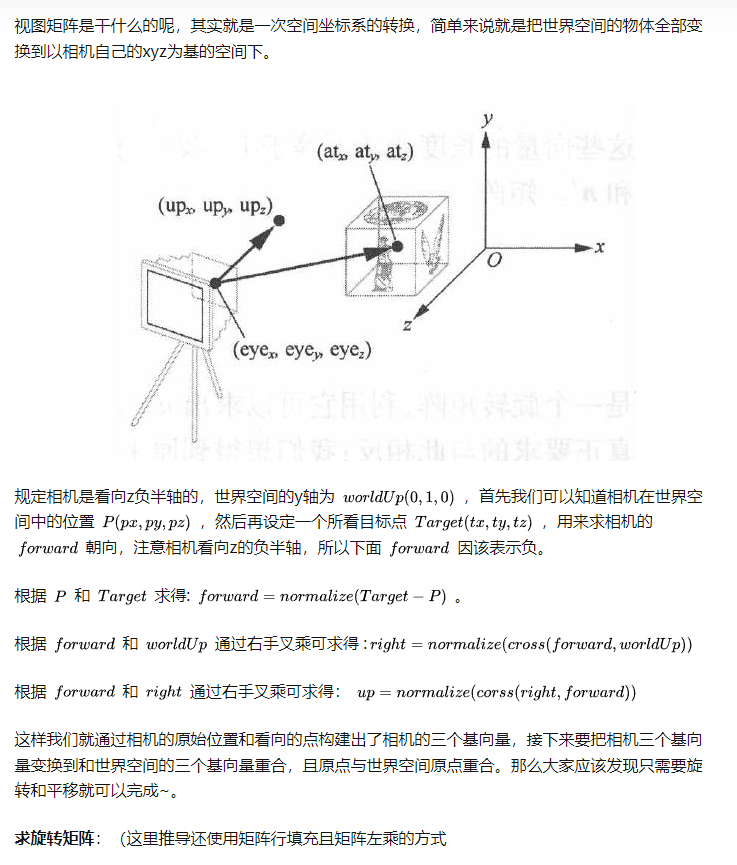
**Viewport 视口变换到屏幕空间**

# ==> ModelView[模型和视图变换两个写在一起]

--> model

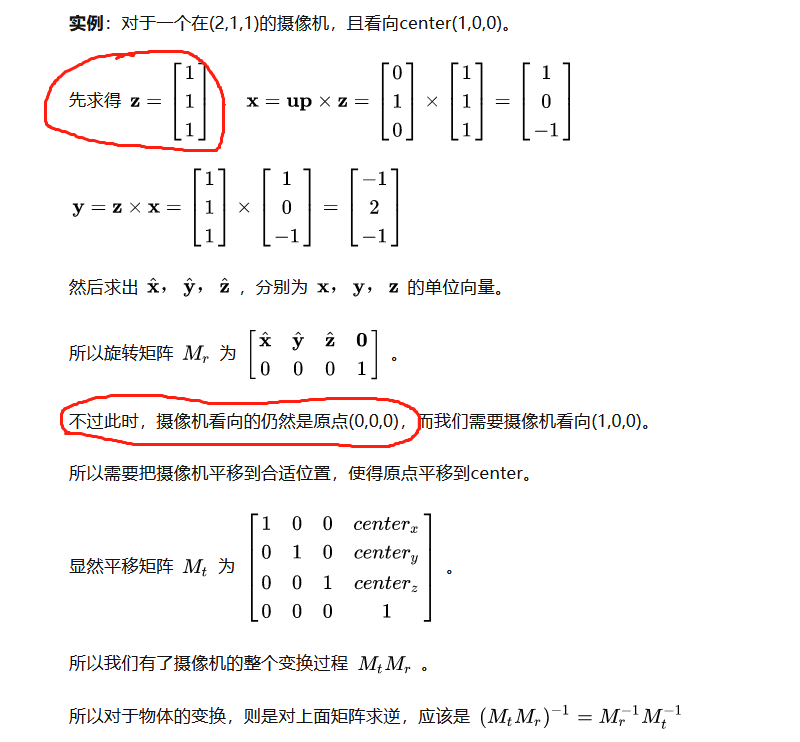
将模型放入世界空间中， 旋转平移矩阵的推导很简单， 模型本地空间的变换是， 将其移动到原点然后变换， 再移动回来. 推导过程待完成:

--> 重点看 view视图变换



Note: 我开始时的疑惑

为什么z = eye-center, x = up\*z, y = z\*x,(注意点乘的顺序,右手坐标系) 得到的一组正交基，用它变换后(这其实只是在原点的旋转变换)摄像机还是看向原点的 :

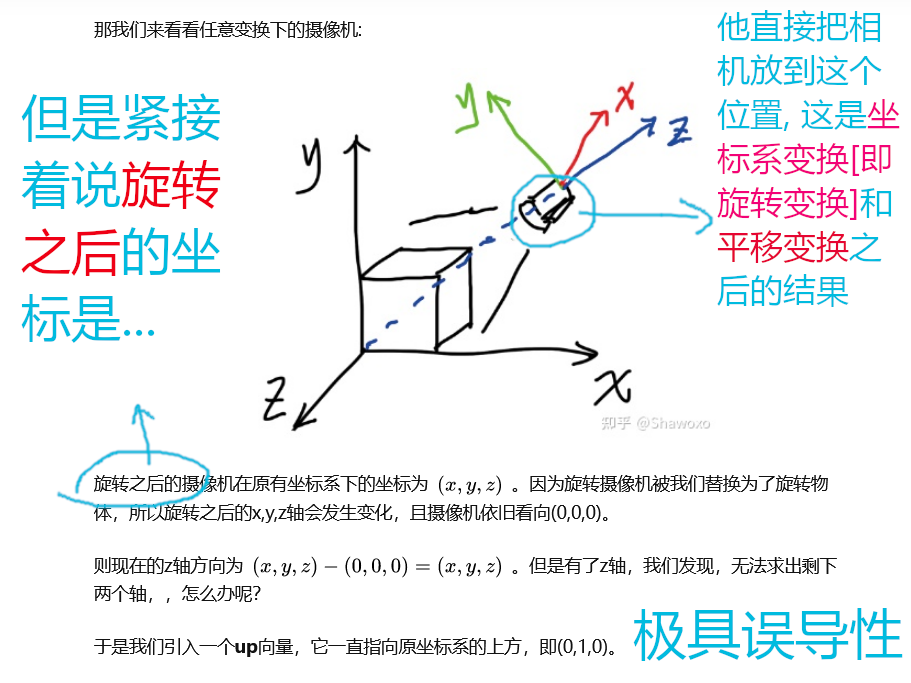


那是因为基变换分为旋转和平移, 上面的正交基只是将原坐标系的各个朝向和目标坐标系对齐， 应为旋转基变换后的坐标系原点还在原坐标系的原点[关键在于向量无论在哪他的起点默认为原点]参考https://www.zhihu.com/question/407603865(向量的起点重不重要)

**两点注意事项**

1. **相机的变换可以通过，对观察物体的逆变换实现**，子扬的思路就是假定相机已经变换过了， 我求出他的空间坐标系下的正交基，(注意这个正交基和位置没关系) 然后看如何将他变回原来的坐标系的正交基, 注意这只是方向 相同了， 在将它平移回原点， 这个过程就是相机变换的逆过程， 也就是物体要执行的过程。另外， 变换相机的时候一定是先旋转(求新坐标系正交基)， 在进行平移(不然新坐标系的正交基是平移后绕原来的原点变换的， 完全错乱), 所以逆变换就是先平移然后旋转。

**2.原来知乎的这个教程极具误导性**



至少相机上的坐标系图标， 应该画在原点， 如果他只是想表示旋转的话， 总之他这个可以帮你反面思考。

看看先不投影的结果

**代码**

mat4 modelview(vec3f eye, vec3f center, vec3f up){  
//-------------------model------------------  
  
//---------camera view (look at)------------  
 vec3f z = (eye - center).normalize();  
//note the order of cross v1 and v2.  
 vec3f x = cross(up, z).normalize();  
 vec3f y = cross(z, x).normalize();  
  
 mat4 view = mat4::identity();  
 for (int i = 0; i < 3; ++i) {  
 view[0][i] = x[i];  
 view[1][i] = y[i];  
 view[2][i] = z[i];  
  
 view[i][3] = -center[i];  
 }  
 return view;  
}

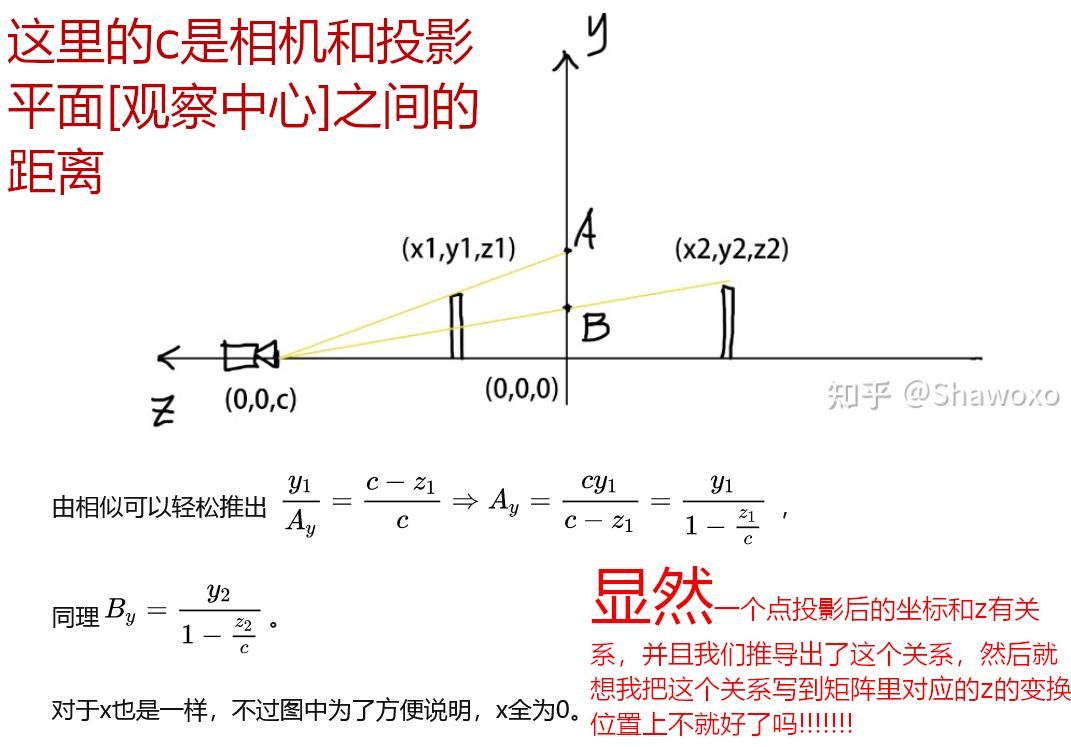
- bug分析

modelview(camera, center, vec3{0, 1, 0});

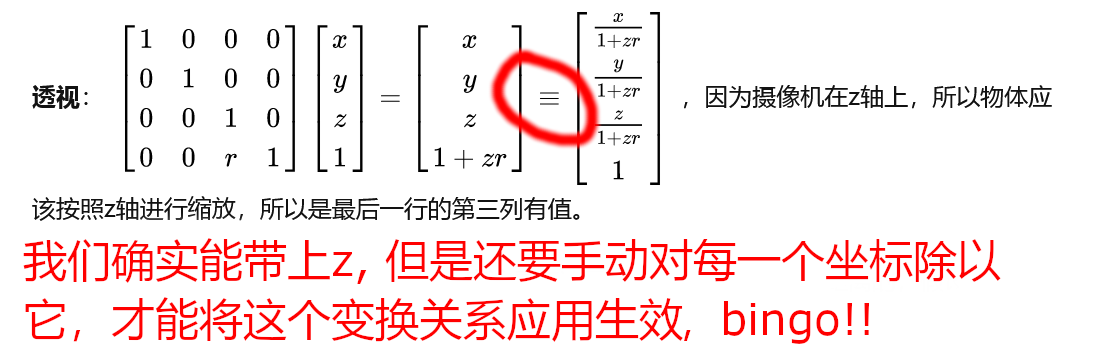
up向量不是{0,0,1}， 你要避免多少这要的失误，见编程法则！！

# ==> Projection

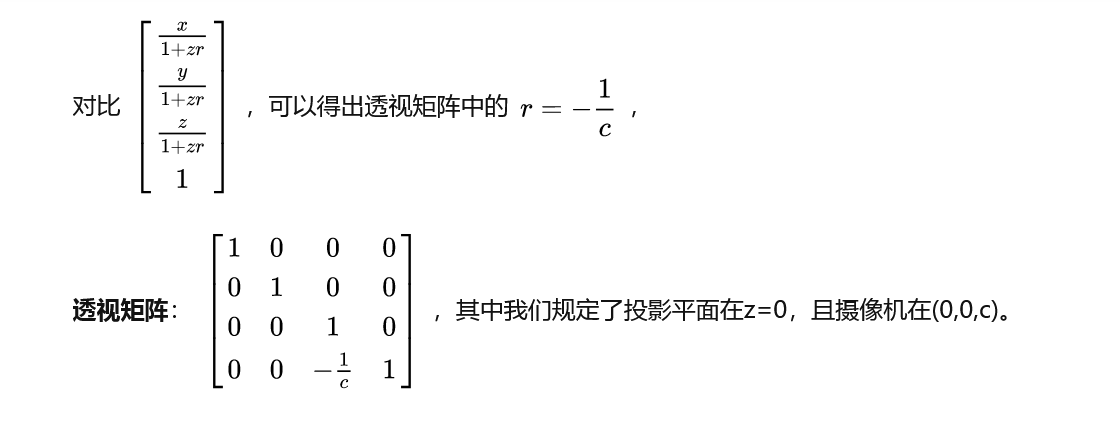
**透视投影的原理**



**然后我们先试着在矩阵对应的位置写一个factor看看**



**然后就是对比确定具体的factor值是多少:**



代码

mat4 projection()

{

mat4 matrix = mat4::identity();

matrix[3][2] = -1.f / (eye - center).norm();

return matrix;

}

NOTE!!!!!!!!!!

别忘了应用过后除以最后一个维度

# ==> viewport

视口矩阵就是把我们之前手动将(-1,1)映射到(0,width)用一个矩阵操作来代替了，通过缩放和平移来实现。

这里的xy参数是相对与中心的偏移， 默认为0, 可以设置为其他看看

viewport(width/8, height/8, width\*3/4, height\*3/4);

渲染结果（render target ）尺寸为width\* 3/4, 还剩1/4， 然后将结果居中， 就是 xy各移动1/8;

注意这里我们没有变换z的值

mat4 viewport(int dx, int dy, int rt\_w, int rt\_h){  
 mat4 m = mat4::identity();  
 m[0][0] = rt\_w / 2.;  
 m[1][1] = rt\_h / 2.;  
 m[2][2] = 1.0;  
  
 m[0][3] = dx + rt\_w / 2.;  
 m[1][3] = dy + rt\_h / 2.;  
 m[2][3] = 0.0;  
  
 return m;  
}