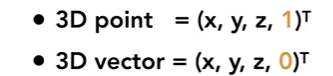
# 3D transformations

### 1、General Introduction

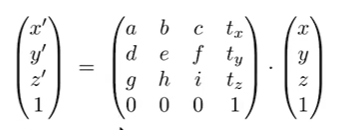
Use homogeneous coordinates again:



(x, y, z, w) (w != 0) 其实是3维空间中的点​ (x/w, y/w, z/w）

在3D里同理与上一节2D中所推导的方法，用齐次坐标来描述变换。在这里依然需要注意区分3D的点和向量。

Use 4×4 matrices for affine(仿射)transformations：

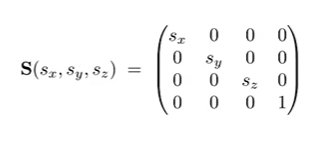




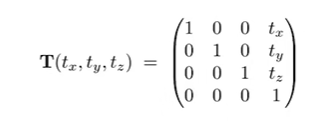
当然是先线性变换，再平移

### 2、Basic 3D Transformations

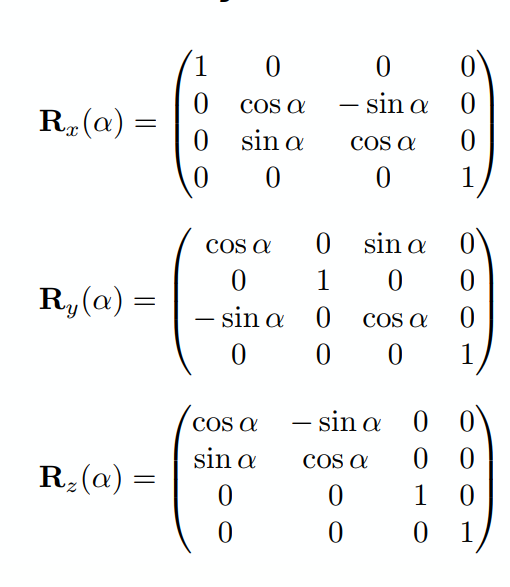
（1）、Scale



（2）、Translation



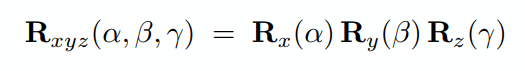
（3）、Rotation around x-, y-, or z-axis(以下是绕x y z轴旋转)



在这里，我们可以观察到沿y轴旋转的旋转矩阵中有两项（右上和左下的sinα）的符号被调换了，这是因为以右手螺旋定则来看的话，沿x轴（大拇指指向x轴）是从y转向z，沿z轴（大拇指指向z轴）是从x到y，而沿y轴是从z到x（而不是x到z），所以相应的符号也就反了。

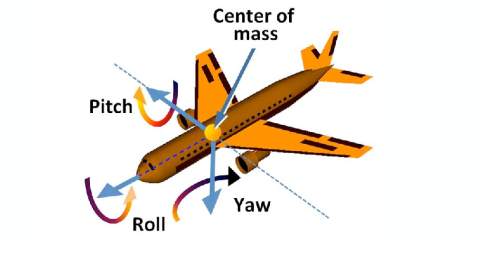
### 3、3D Rotations

任意一个旋转都可以被分解成由绕x，y，z轴的三个旋转合成的一个旋转，因此



So-called Euler angles（这也就是所说的欧拉角）

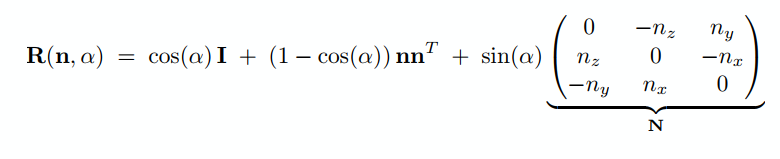
Often used in flight simulators: roll, pitch, yaw



但是使用欧拉角会产生万向锁的问题

### 4、Rodrigues’ Rotation Formula(罗德里格斯旋转公式)

Rotation by angle α around axis n(围绕任意轴n旋转α角度可写为以下式子)



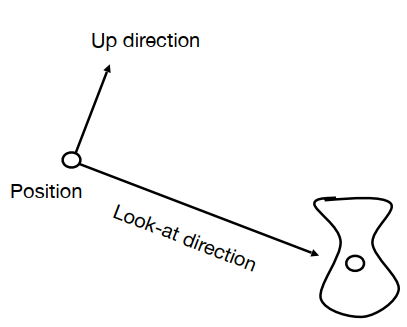
注意这个旋转轴n用向量表示，就是该向量过原点产生的射线。如果我们想让物体沿任意轴旋转(包括不过原点的轴)，可以和二维图像围绕任一点旋转一样，先平移到过原点轴进行旋转，再平移回去

## Viewing transformation(观测变换)

观测变换=视图变换(摆相机)+模型变换(挪相对位置)+投影变换(透视投影+正交投影)

### （一）、View / Camera transformation（视图变换）

#### 1、如何实现视图变换(怎么摆摄像机)



指定相机位置

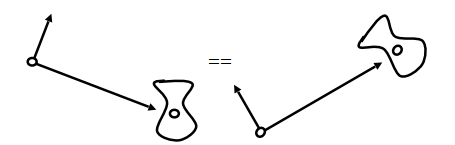
指定相机看的方向

指定向上方向

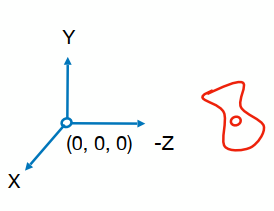
在此处我们要定义三个量，来确定唯一的摄像机变换。首先要确定摄像机在空间中所在的位置 然后确定摄像机朝向的方向 。当确定了这两个信息以后，我们可以发现摄像机仍然可以沿着所在的轴（左右）旋转（在建模软件中做摄像机动画时Z轴没锁），因此我们还需要确定一个向上的方向来锁住旋转轴这样就确定了唯一的一个摄像机信息

#### 2、Key observation

如果摄像机和场景中所有的东西一起移动的话，那么整个场景在摄像机中的画面还是相对保持不变。

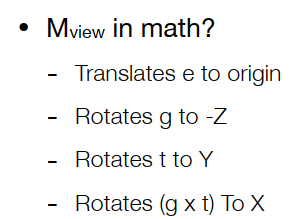


因此约定了将摄像机的位置 变换到坐标原点（0,0,0），将向上的方向 变换到Y轴，将摄像机朝向的方向变换到-Z。让场景中的其他所有物体也随着摄像机的变换而变换。

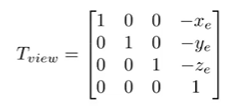


### （二）Transform the camera by Mview(模型变换)

如何实现Key observation所说的移动？遵从以下方法：



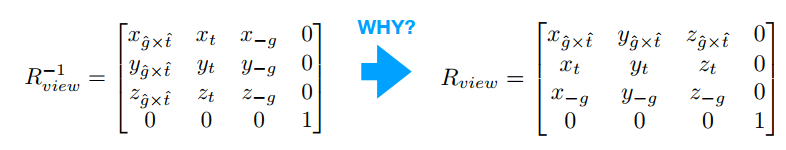
in math： (模型变换的仿射矩阵) = (如何写出其旋转矩阵与平移矩阵是重中之重)

这个很好写

我们希望把转到-Z，转到Y，叉乘为X

在写旋转矩阵的时候，从未知的，，和叉乘变换到-Z、Y、X可能很困难，但是从X、Y、Z变换到叉乘，和就显得容易了很多

所以可以先写出逆向变换的矩阵，再对其求逆，即为转到-Z，转到Y，叉乘为X的旋转矩阵



由于该旋转矩阵是正交矩阵，所以该矩阵的逆就是该矩阵的转置矩阵

**Summary**

Transform objects together with the camera

Until camera’s at the origin, up at Y, look at -Z

Also known as ModelView Transformation

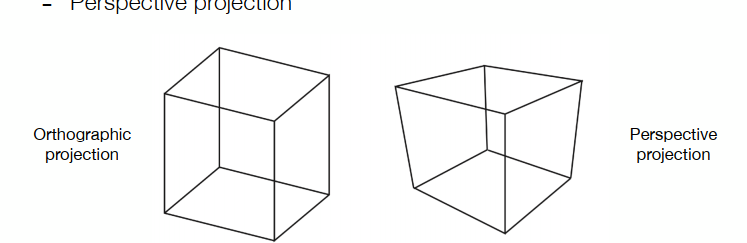
But why do we need this?

For projection transformation！

## 二、Projection transformation(投影变换)

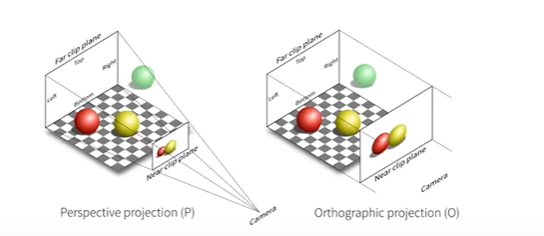
Projection in Computer Graphics（3D to 2D）

Perspective projection vs. orthographic projection(透视投影与正交投影)



正交投影：工程制图

透视投影：无限延长平行线看起来就好像相交，近大远小



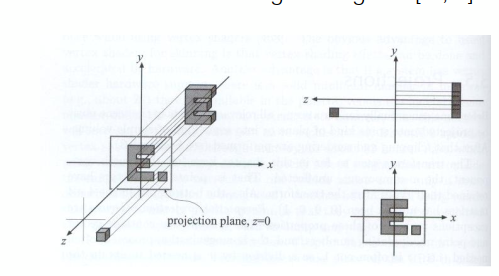
### 1、Orthographic projection

### （1）A simple way of understanding

Camera located at origin, looking at -Z, up at Y

Drop Z coordinate

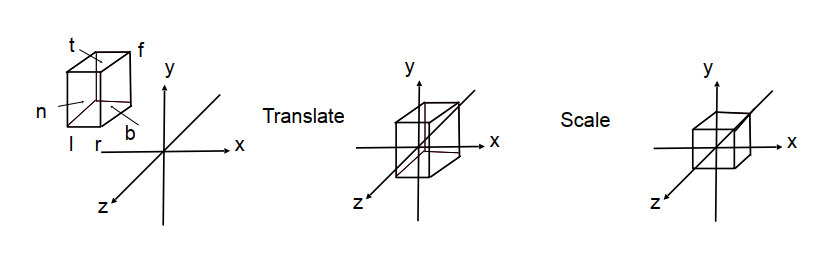
Translate and scale the resulting rectangle to



对于正交投影，最简单的理解方式就是将所有三维空间的物体挤压到一个二维的平面，也就是将所有物体的z坐标直接去掉，最后将图形压缩到[-1,1]的二维区域内。也就实现了从三维到二维平面的映射。

### （2）In general

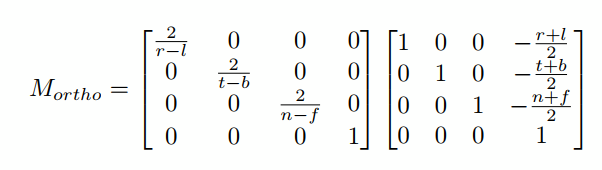
We want to map a cuboid [l, r] x [b, t] x [f, n] to the “canonical (正则、规范、标准)” cube [-1, 1]3



在这里，不管在空间中给一个什么样的长方体，都可以映射成一个中心在原点的立方体。

在X轴上，我们定义左和右（l和r），在Y轴上，我们定义下和上（b和t），在Z轴上，我们定义远和近（f和n）。通过这六个值，我们即可定义一个长方体。

Translate (center to origin) first, then scale (length/width/height to 2)



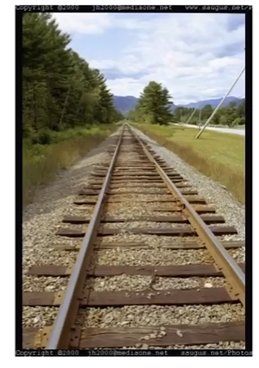
先通过一个位移矩阵将长方体中心移动到原点，再通过一个缩放矩阵将长方体压缩成一个立方体。

沿 -Z 看 /正在使近处和远处变得不直观（n > f），这就是为什么OpenGL（一种图形API）使用左手坐标的原因

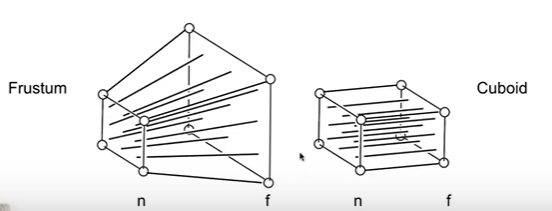
### 2、Perspective projection

对象更小(近大远小)

平行线不平行;收敛到单点(平行线相交)



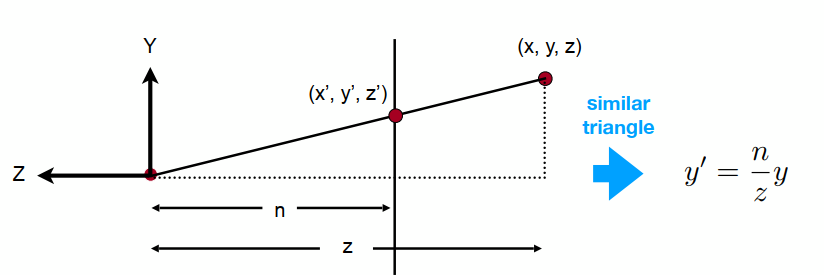
如何进行透视投影？



我们要通过变换将左侧这个视锥“挤”成右边这个长方体，再做一次正交投影，问题就解决了。在上面我们已经知道正交投影怎么做了，所以现在所有的问题集中在如何将这个视锥“挤”成右边这个长方体。

在此我们定义几个问题。首先近平面n在挤压之后，任何一个点都不会变，远平面f在挤压过程中，Z值不会变，也就是说挤压之后远平面的Z值依然是f；远平面的中心点在挤压之后也不会发生变化，x、y依然是0，Z=f。

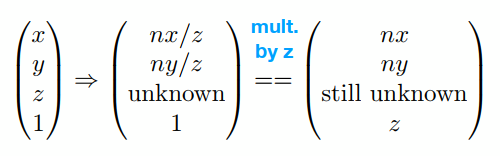
通过侧视图，对于视锥上任意一个点（x，y，z）和近平面（x’，y‘，z’），我们会发现两个相似三角形



近平面上的点的z‘=n，根据相似三角形对应边成比例我们可以求出

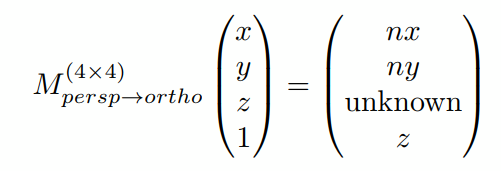
,

​那么我们将刚刚的变换过程写成齐次坐标的形式：

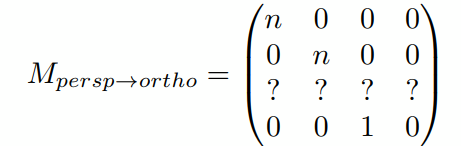


此处我们根据齐次坐标的性质，将中间的式子同乘一个z，即得到最右边的式子，这就是原来的点通过变换后得到的新的点的坐标。此时z’我们依然不知道。

我们希望算出来到底是什么样的一个变换矩阵使得这个点的坐标变换成最右边的样子，也就是我们要求出矩阵Mpersp->ortho（4×4）

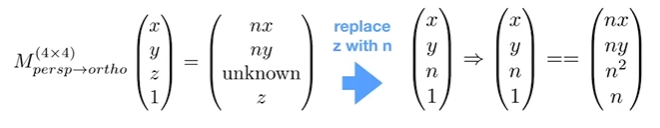


此时可以先通过刚刚的已知量将这个矩阵的部分写出来：

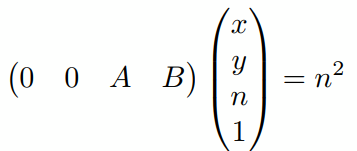


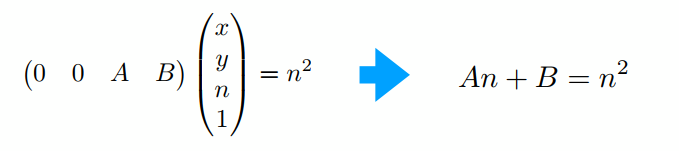
下面我们来取两个特殊的点。刚刚在一开始已经提到，近平面在挤压之后，任何一个点都不会变；远平面在挤压过程中，Z值不会变，也就是说挤压之后远平面的Z值依然是f（Z=f）。

首先我们取近平面上任意一点，由于近平面在挤压之后，任何一个点都不会变，(x,y,n,1)→(x,y,n,1)所以此时z就可以用n来替代，得到以下式子：

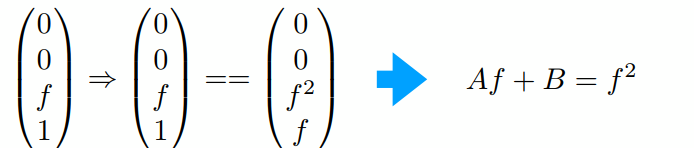


所以这个矩阵与近平面上的任意一点相乘的结果第三行是，而第一、二行经过齐次坐标的变换后，能将x变为nx，y变为ny，与一开始的通用写法相同，此时第三行的显然跟前两行的数x、y没关系，所以这个转移矩阵的第三行(0,0,A,B)和近平面上的任意点(x,y,n,1)存在以下关系：：





我们再选取远平面的中心点，由于远平面的中心点在挤压之后也不会发生变化(0,0,f,1)→(0,0,f,1)。x、y依然是0，Z=f，转移矩阵的第三行(0,0,A,B)和远平面上的中点(0,0,f,1)存在以下关系，可以写出矩阵：



联立刚刚的两个式子：



我们可以解出A和B。

至此，我们把整个变换的矩阵全部填完了。

接下来做正交投影，Mpersp(透视投影矩阵)= MorthoMpersp->ortho