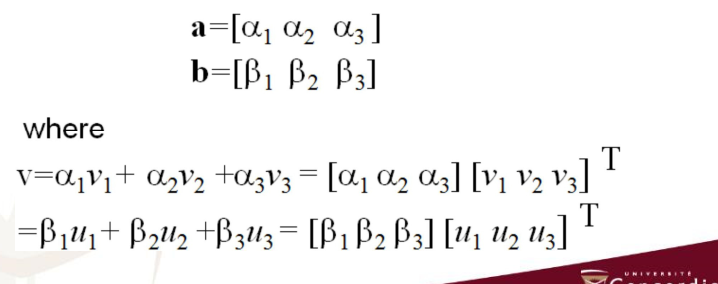
Frame

一个坐标系并不能够表达point

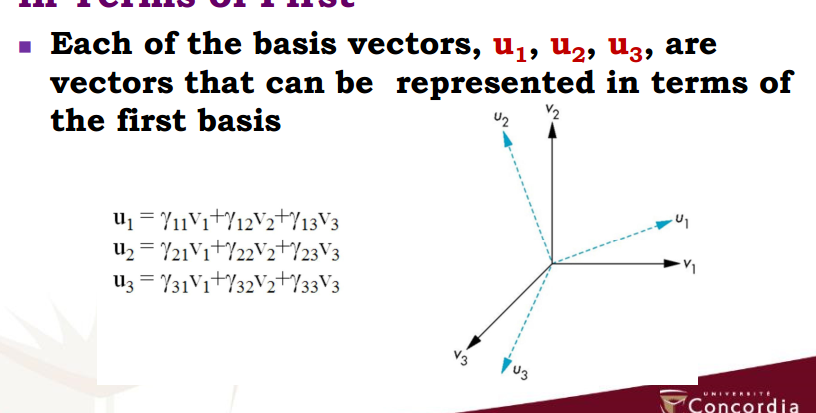
一个坐标系加上一个原点origin,才能形成一个FRAME

坐标系的转换

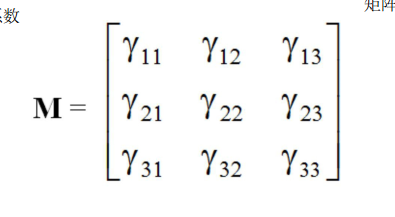
同样的一个向量，在不同的frame里有不同的表达



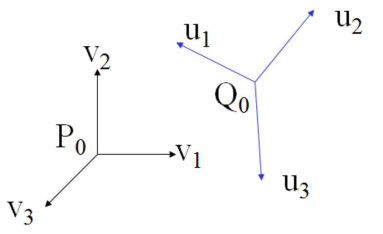
那么我们怎么从第一个coordinate转换到第二个coordinate呢

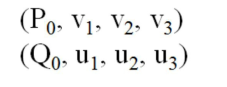


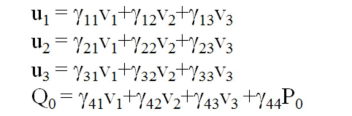
就是把每个坐标轴都当成向量，再原frame的基础上求



现在我们加上origin原点，怎么从一个frame表示另一个frame

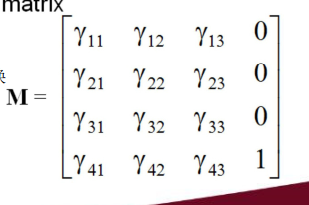






U1u2u3不变，因为是向量，我们并不考虑原点

然后让origin变化



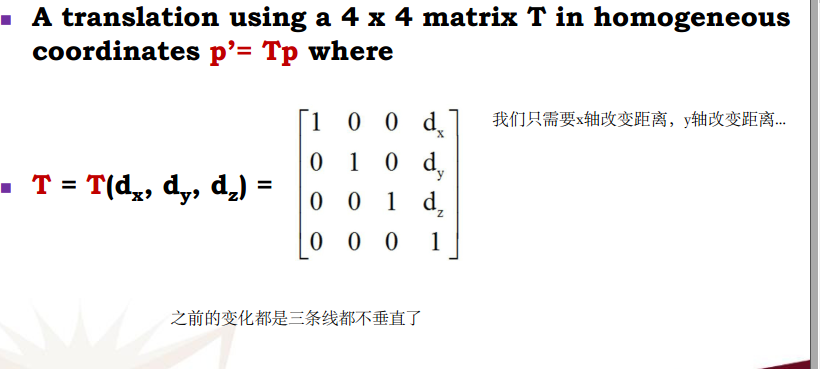
在OPENGL中，我们用world frame作为基础frame

最后我们会用camera frame来表达实体，这个camera frame就是通过world frame转换过来的

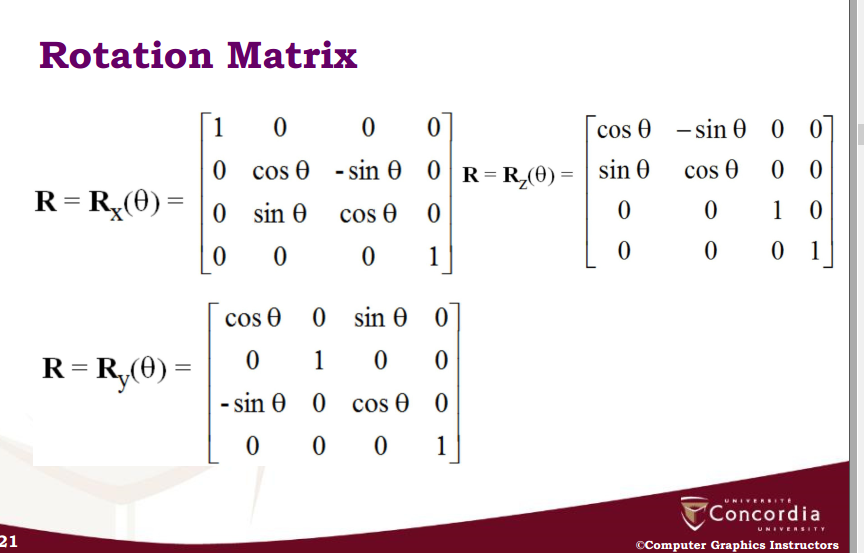
在一开始camera frame= world frame

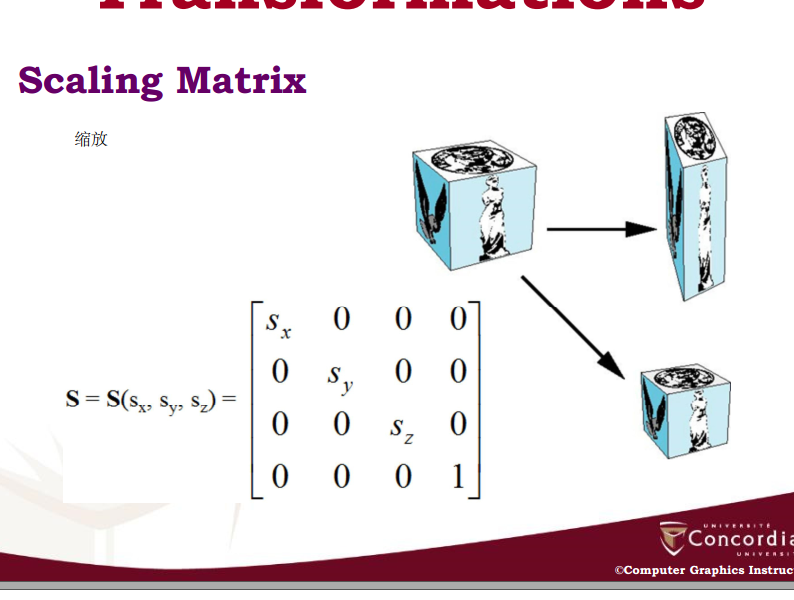
第一种转换

平移转换translate



旋转转换

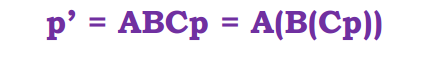




缩放转换

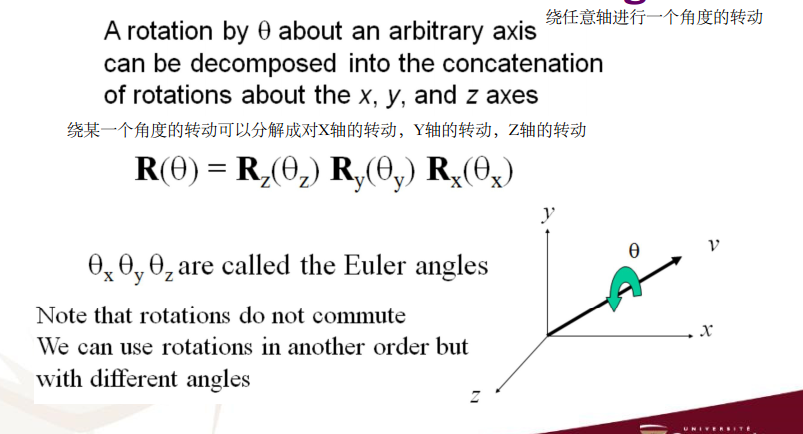
我们可以任意将旋转缩放等相乘

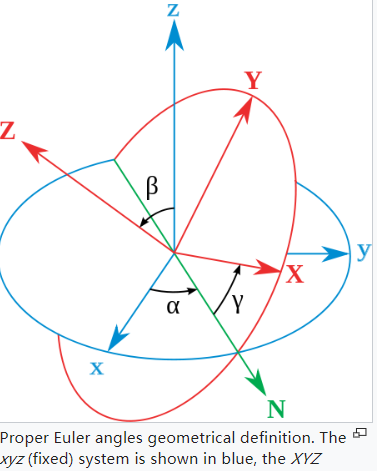




从右往左算

绕原点转动

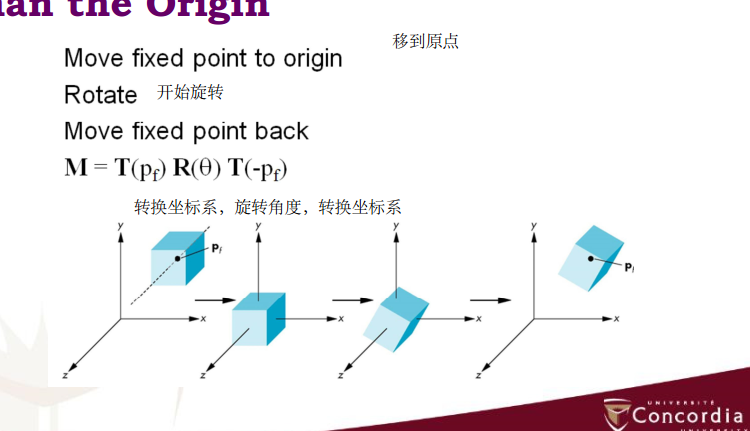




Euler角度代表着原轴到新轴的角度，大小是一样的

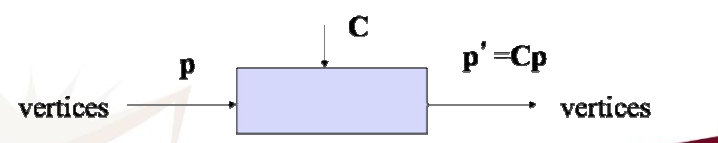
绕一个原点外的定点转动

1. 把定点移动到原点
2. 开始旋转
3. 把定点移动回去



Opengl 里的 CTM current transformation matrix：是一个4x4齐次转换matrix，所有的vertices都要经过这个Pipeine

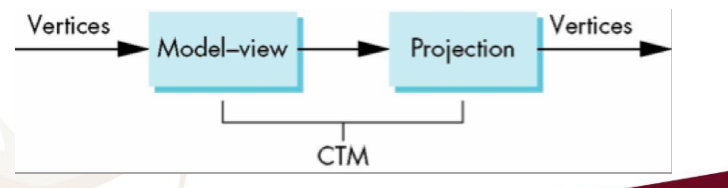
CTM是一个User-defined program并且要在transformation unit里所装载



所有原始vertices经过这个ctm pipeline变成新vertices

OPENGL 是怎么搭建CTM的

opengl用model-view matrix以及projection matrix这两个matrix来构建CTM

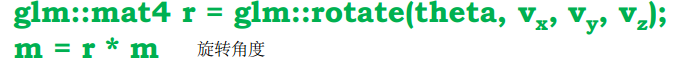


GLM的旋转，平移，拉伸公式

建立标准矩阵



旋转一个矩阵m

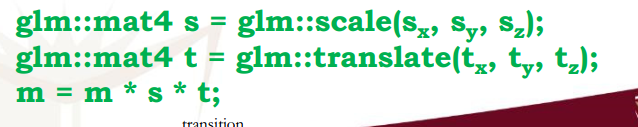


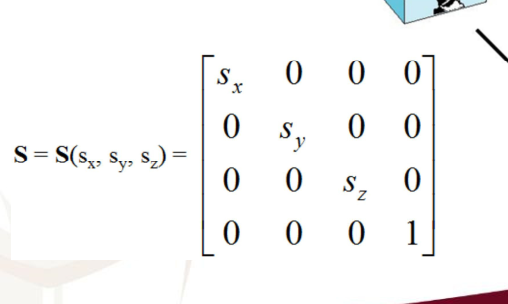
theta是旋转角度，vx vy vz是那条旋转轴（用向量表示，比如地球就是绕着一根轴公转，绕180°半圈，绕360°一年）

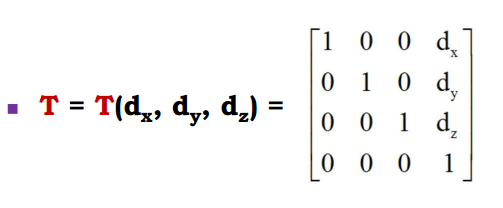
还有一种写法



translation 与scaling







Projection：

我们的眼睛把3D世界折叠到2D图像并放到我们视网膜上

CG里这个过程就叫做projection投射

由两部分组成：

视角转变：camera position与camera direction

orthographic tranformation:从3D到2D

使用4Dmatrix变换法 homogeneous transformation

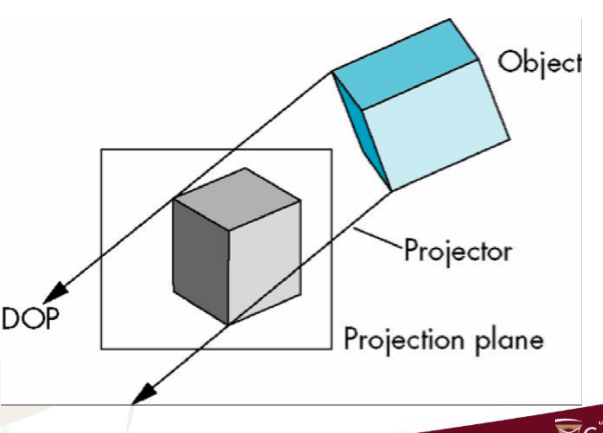
Geometric projections几何投射

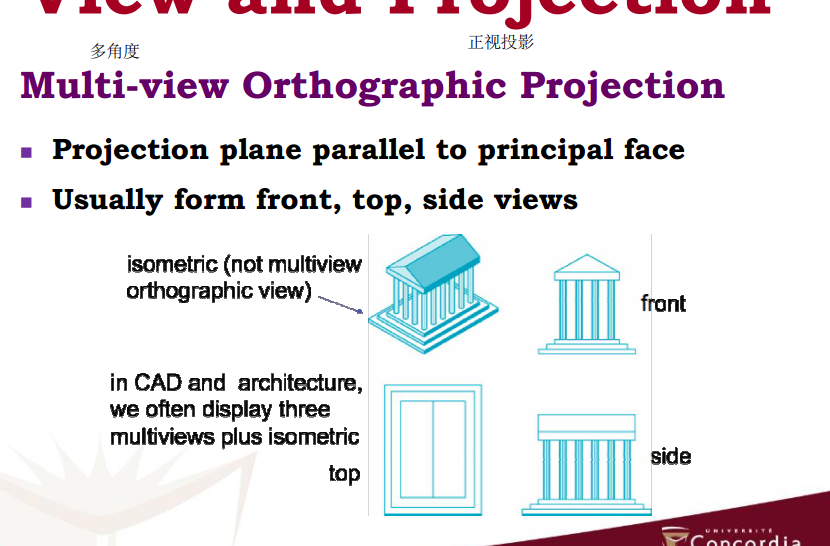
Projection的画法有许多不同的种类

最基本的区别就是parallel 与perspective viewing,即使在数学上parallel viewing是perspective viewing的极限

Projection种类

Parallel projection



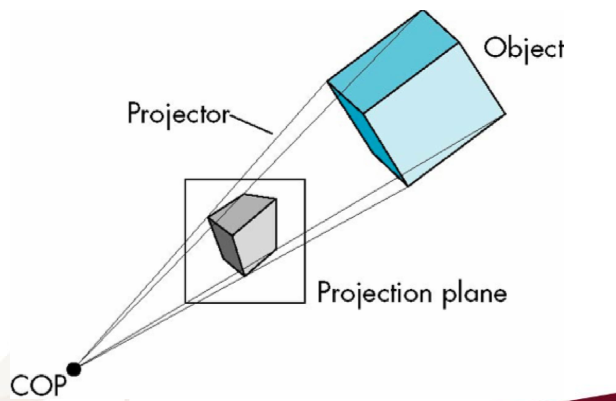


Multi-view orthographic projection的好处

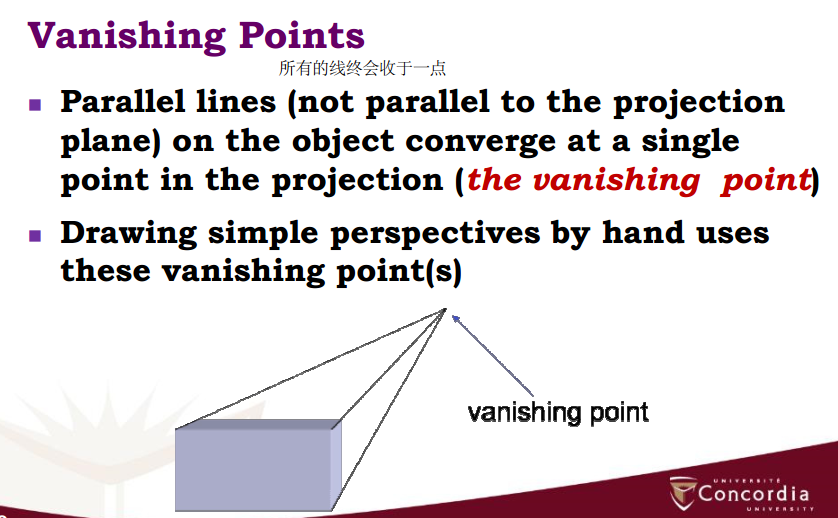
形状确定，保存了长度与角度，可以用做测量，建造计划等

但是不能直观地看见这个物体形象因为有太多面，通常我们会加isometric

Perspective projection,透视投影，中心投影法



所有的实际平行线 最后收为一点（the vanishing point）



离你远的透射出来就小，离你近的投射大，看起来更真实（diminution缩减）

等长的线段投射出来的长度不一定相同

只有平行于projection面的 平面才能保持原有角度

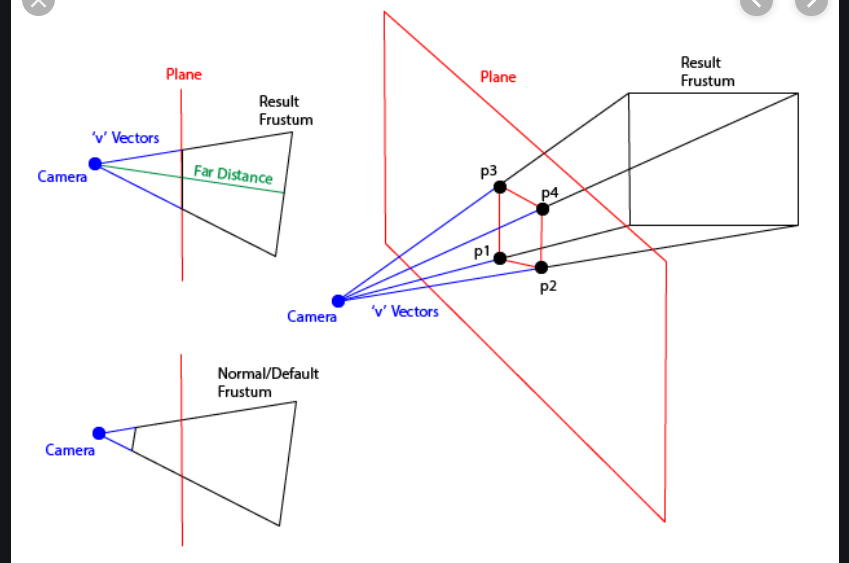
你很难用手构建Parallel projection（但是电脑很简单）

Computer Viewing，计算机视角

一共需要三步来Implement这个pipeline

1. 定位camera ：设置一个model-view matrix
2. 设置一个底：设置一个projection matrix投影矩阵 ，底就是result frustum
3. 剪裁：设置观看容积,观看角度

­­­­­­­MATRIX并不是一个面或者立方体，他是一个转换过程，他是一个变形方式，原来的object经过model-view matrix,从世界坐标系转到了摄像头坐标系，然后再投射到我们的投影上

­

opengl camera

一开始object frame与camera frame共享框架，都在原点

默认model-view matrix是一个单位矩阵

Camera在原点并且朝着Z轴反方向

Opengl也指定了默认view voume，那就是一个边长为2CM的立方体，中心在原点

默认projection matrix是一个单位矩阵//即原坐标系·通过投影矩阵变化没区别

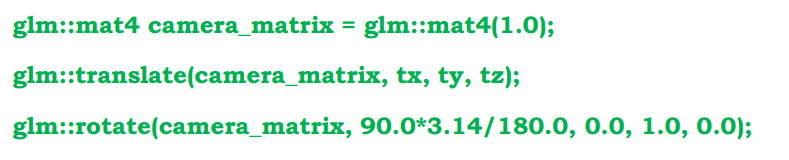
如果我们想看一个Object的背面

1. 移动camera frame，让camera指向正Z轴
2. 把object指向负z轴，//更改world frame

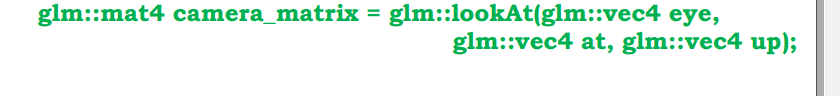
对于model-view matrix来说这两种都一样

我们如何设置我们想要的camera

1. 设置单位矩阵//经过这个变换还是原坐标系
2. 平移缩放
3. 最后一步旋转



GLM library也提供了一步到位的设置camera\_matrix方法即lookat

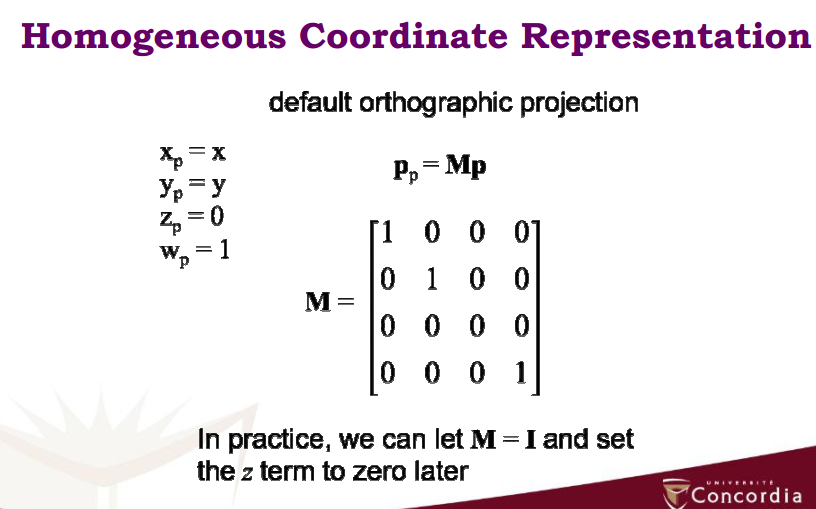


Eye是摄像机位置，up是upwards，是你这个摄像机视角朝上的方向(vectr),一般来说都是010，at就是你看着的点（position），也就是视角中心

Projection and normalization 设置一个底

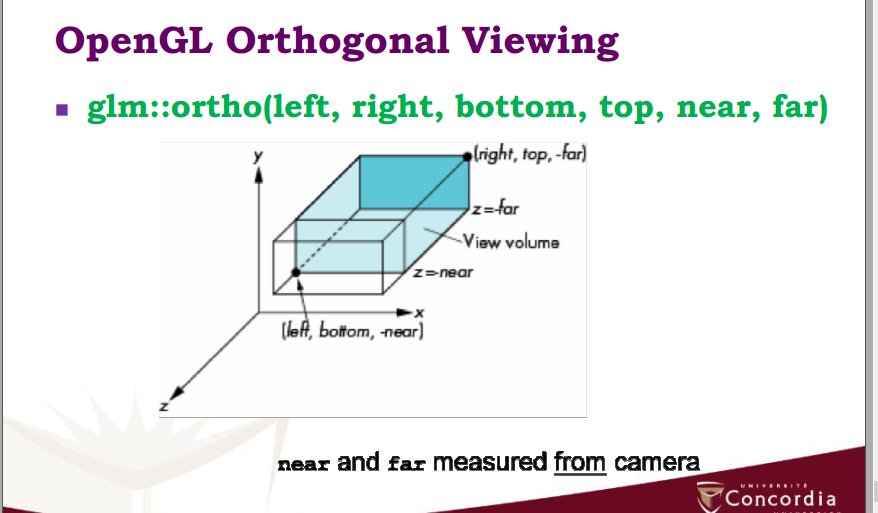
默认的底就是这个camera frame框架的XY那个竖着的面

对于在默认view volume内所有点，XY投射到projection上，xy等于原值，z=0

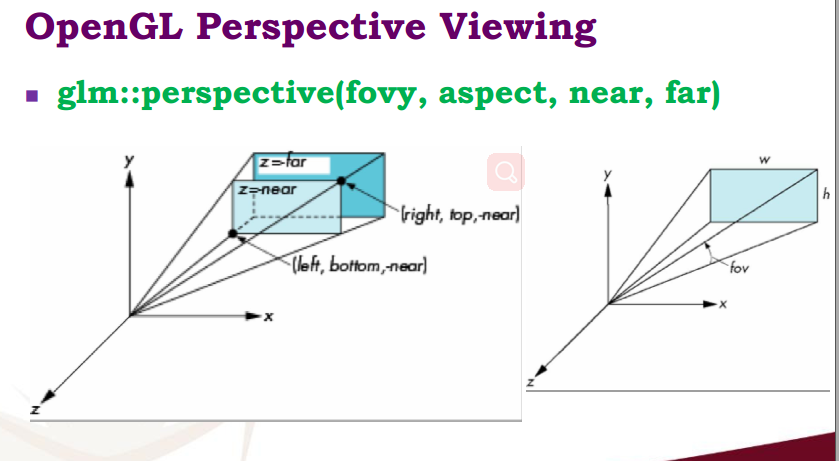


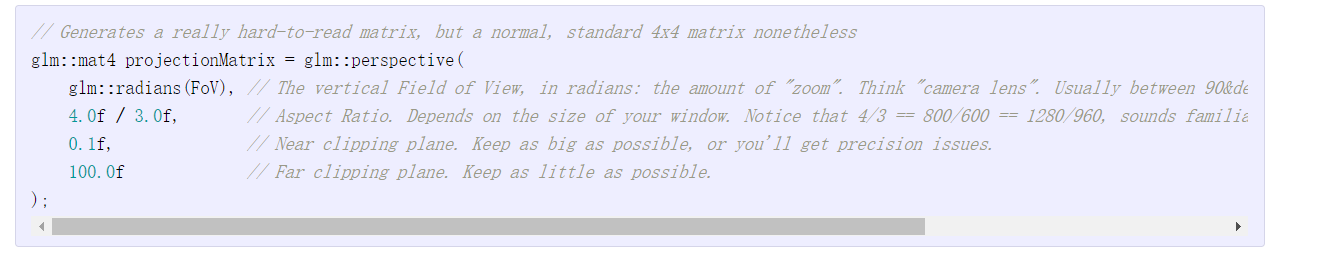
默认投影

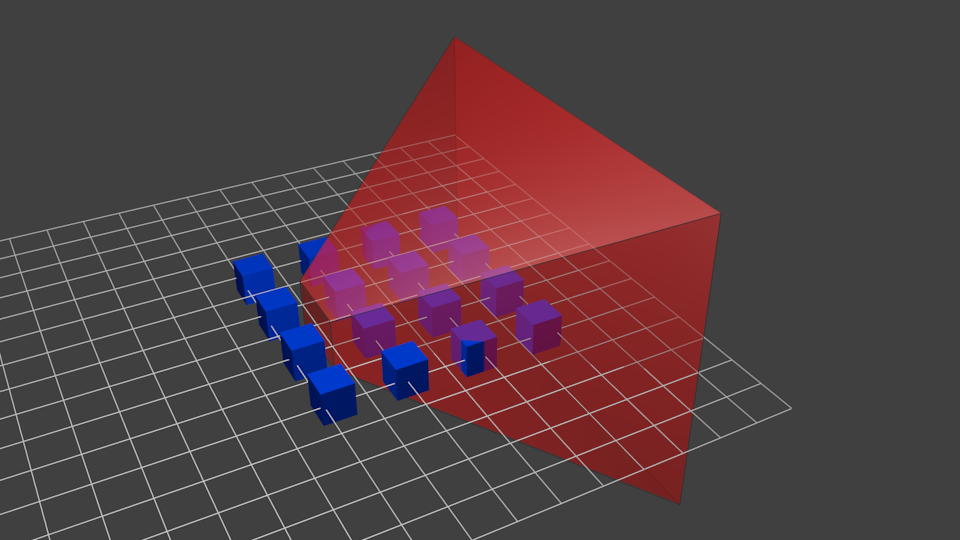
Glm怎么创造orthogonal viewing



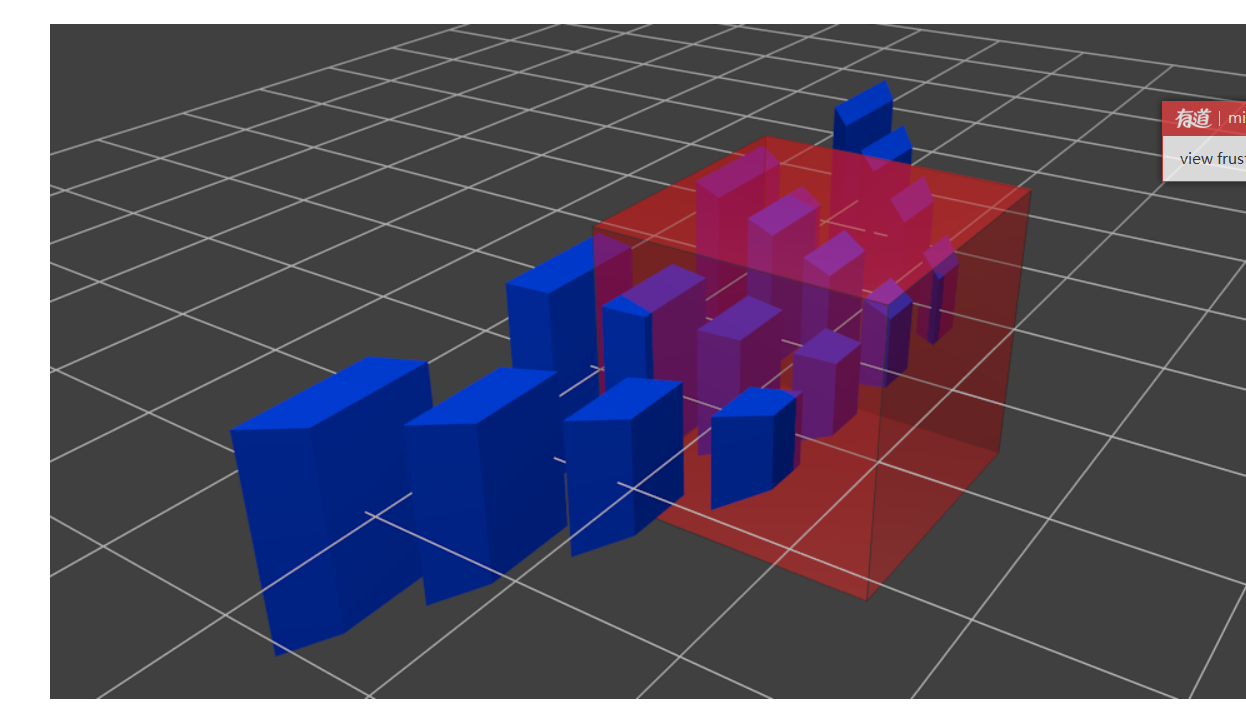
Near就是我们要的投影面，far就是我们视角的尽头，然后这个volume将把我们要描述的object包含在内







可以看见小的那个面是我们投影面，大的面是实际视角能看到的面



我们实际能看到的结果，在经过投影matrix变形后，我们的视角volume变成了方体，物体或多或少变形了，近的大远的小