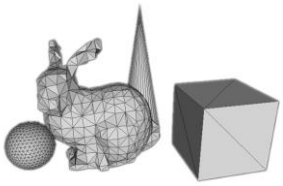


# Introduction to Computer Graphics


## Viewing

Ming Zeng

Software School

Xiamen University

Contact Information:

zengming@xmu.edu.cn

# 第四章 几何对象与变换
# Geometry Objects and Transformations

- 第一次课：几何及其表示 (更注重数学概念)
  - 与坐标无关的几何：点、标量、向量
    - 向量空间、仿射空间、欧氏空间
  - 几何的坐标表示
  - 几何在OpenGL中是如何表示的（OpenGL中各种不同标架）
- 第二次课：变换及OpenGL实现（更注重实际算法运作）
  - 几何变换的数学表示
  - OpenGL中的几何变换
  - 专题：虚拟轨迹球的实现

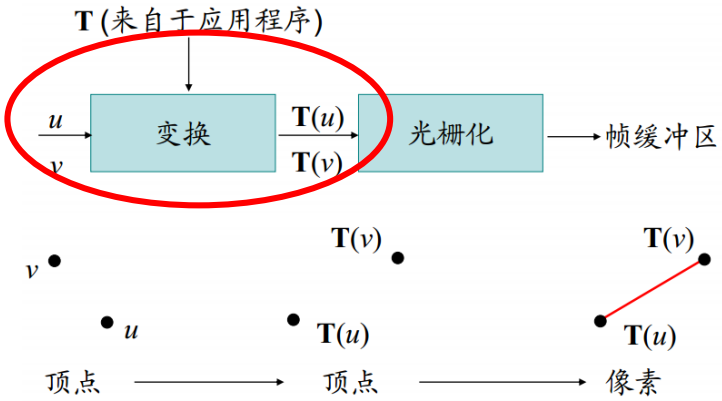
# 第四章 几何对象与变换
# Geometry Objects and Transformations

- 第二次课：变换及OpenGL实现（更注重实际算法运作）
  - 几何变换的数学表示
    - 平移、旋转、缩放的4*4矩阵表示（对应齐次坐标）
    - 变换的复合：复杂的旋转、旋转中心不在原点的旋转
  - OpenGL中的几何变换
    - gltranslatexx, glRotatexx, glScalexx等
    - glLoadIdentity等
    - 获取opengl矩阵的方法：glGetFloatv(GL_MODELVIEW_MATRIX, arr);
    - glPushMatrix, glPopMatrix等
  - 专题：虚拟轨迹球的实现
    - 鼠标在窗口的位置映射到球面上
    - 计算相邻两个轨迹点在球面上的张角，以及转轴
    - 当鼠标在轨迹球之外时，应该如何处理？

# 第五章 视图(Viewing)

本章节主要解决相机的设置问题

- 视图概念(了解)
- OpenGL中的视图设置 （掌握）
  - 描述相机方位的函数、矩阵
  - 描述相机投影的函数、矩阵
- 回顾、总结图形流水线的前半部分（顶点变换）（掌握）



# 第五章 视图(Viewing)

本章节主要解决相机的设置问题

- 视图概念
- OpenGL中的视图设置
  - 描述相机方位的函数、矩阵
  - 描述相机投影的函数、矩阵
- 回顾、总结图形流水线的前半部分（顶点变换）

# 视图概念

- 视图（这里特制相机的投影过程）主要包括：
  - 平行投影
  - 透视投影

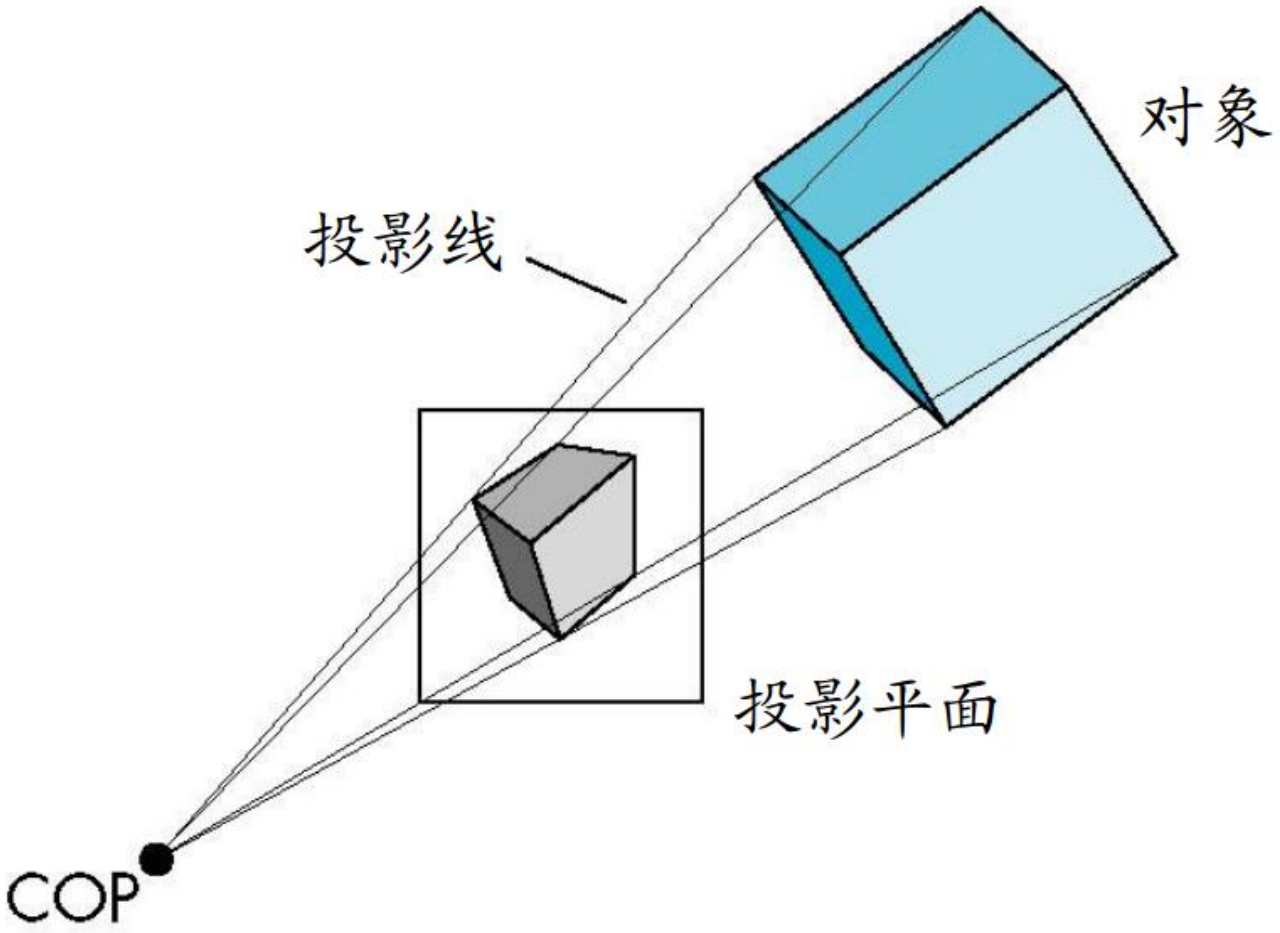
# 视图三要素

- 视图中需要三个基本要素
  - 一个或多个对象
  - 观察者，带有一个投影面
  - 从对象到投影平面的投影线
- 视图就是基于这些要素之间的关系的
  - 观察者选取一个对象，以期望看到的方位进行定向

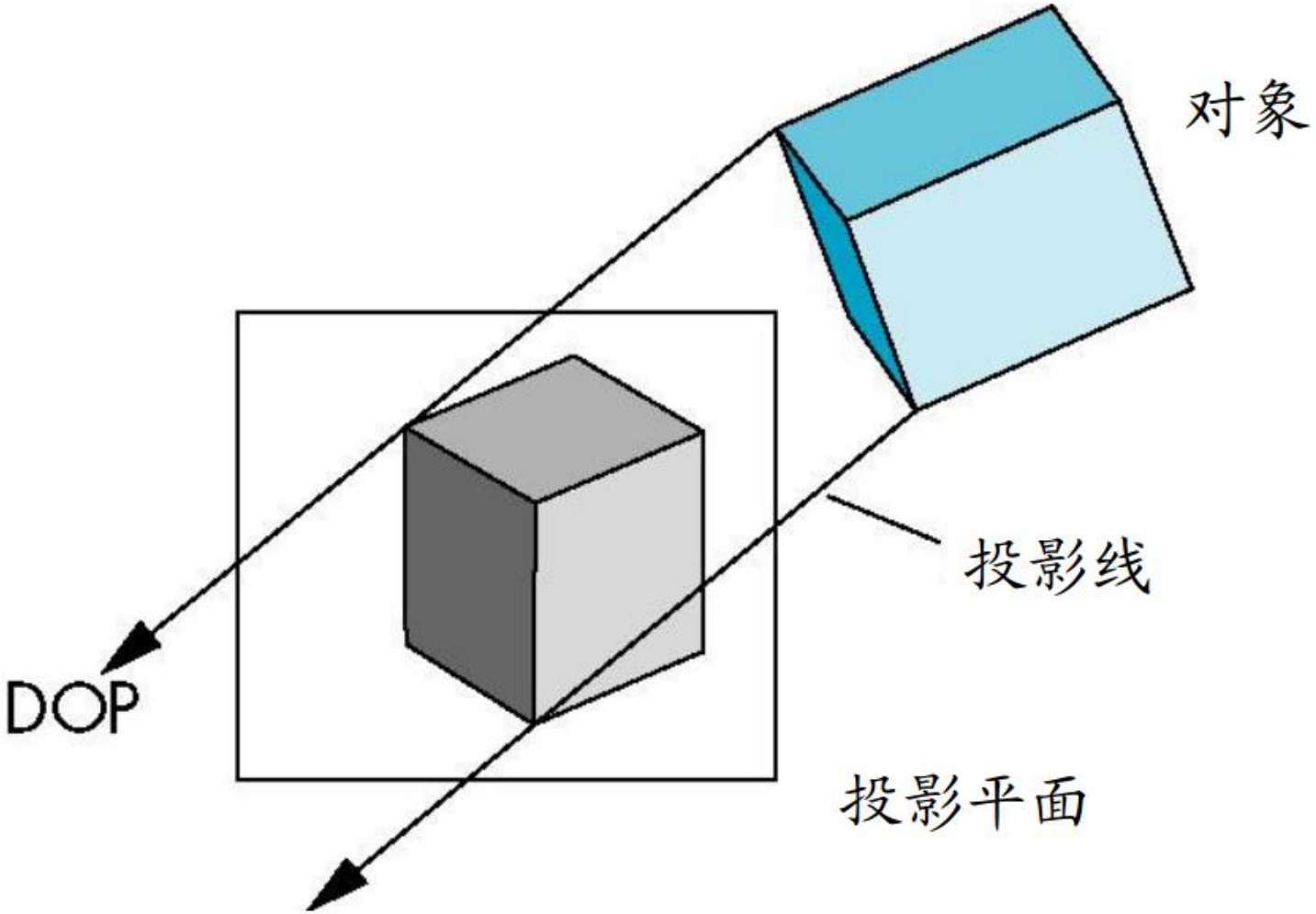
# 透视投影 vs 平行投影

- 计算机图形学中把所有的投影用同样的方法处理，用一个流水线体系实现它们
- 在经典视图中为了绘制不同类型的投影，发展出来不同的技术
- 平行投影和透视投影有基本区别，虽然从数学上说，平行投影是透视投影的极限状态

# 透视投影

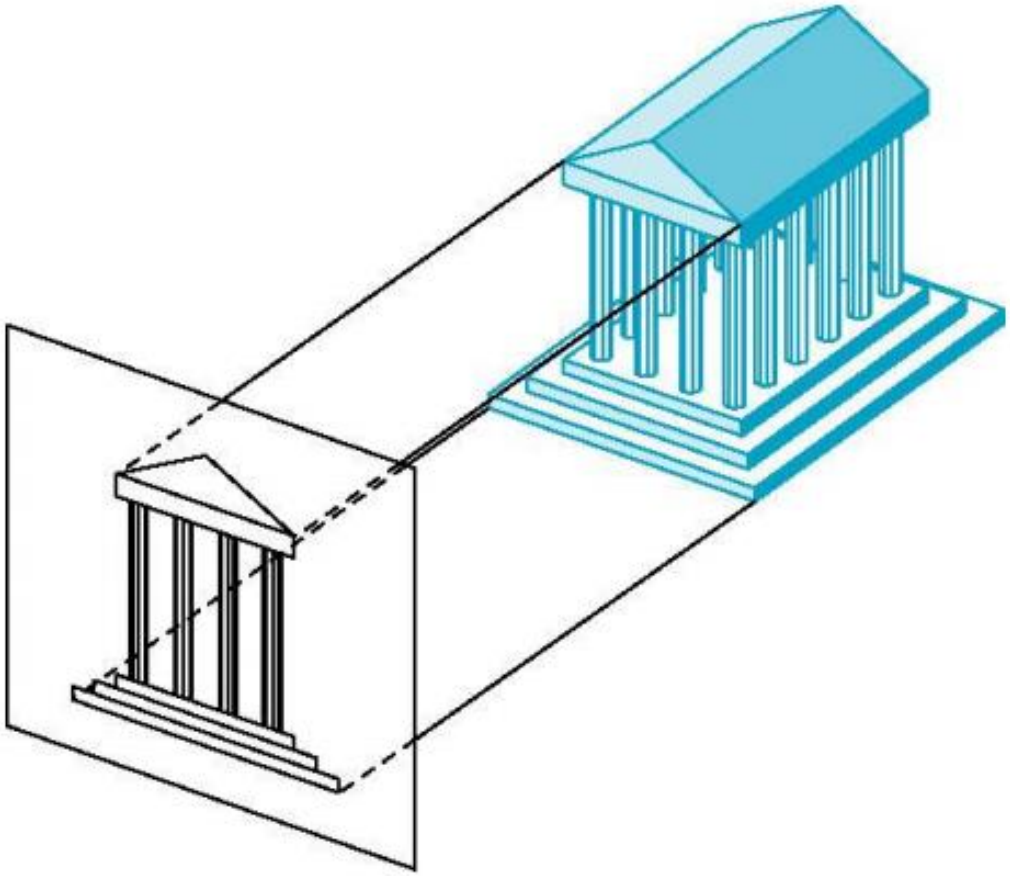


# 平行投影



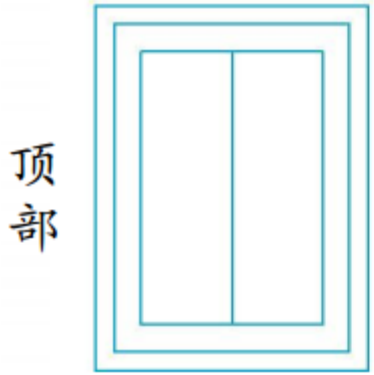
# 正交投影

- 投影线垂直于投影平面

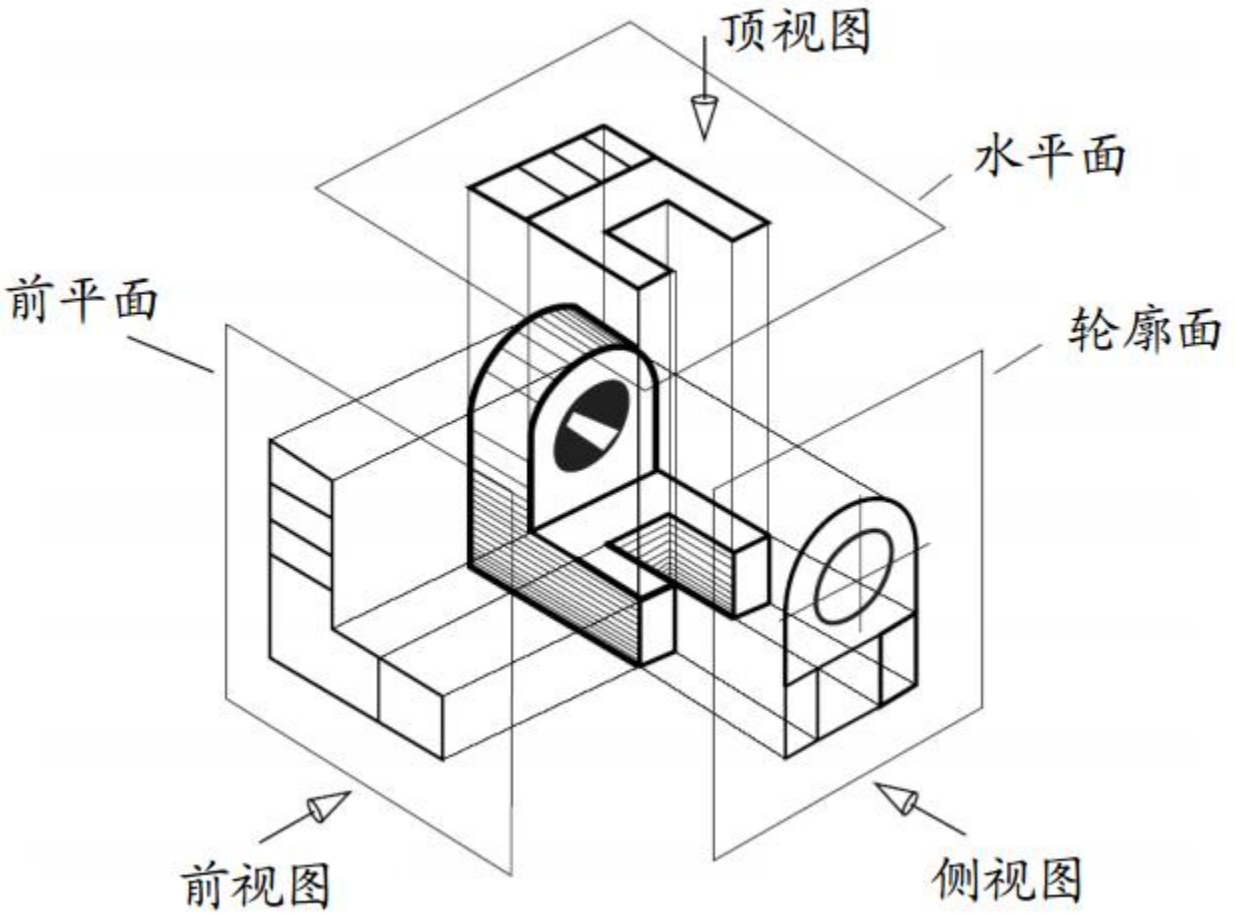


# 多视点正交投影

- 投影平面平行于某个主视面
- 通常从前面、和侧面进行投影
  - 三视图
  - 正视图（主视图）
  - 俯视图
  - 侧视图（左视图）
  - 正等轴测图（不是多视图正交视图中的一部分）
  - 在CAD和建筑行业中，通常显示出来三个视图以及正等轴测图


# 机器零件的三视图

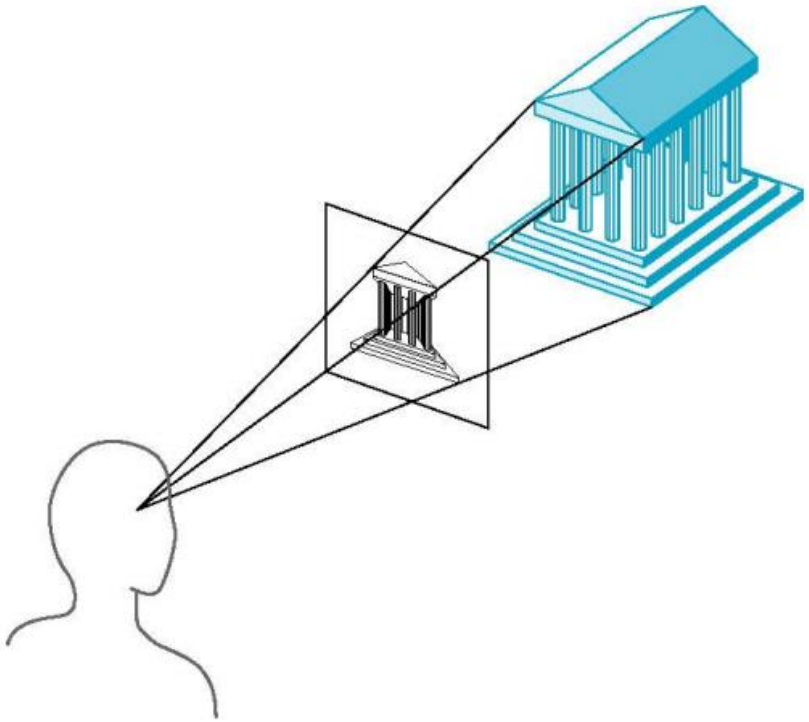


# 优缺点

- 保持了距离与角度
  - 保持形状
  - 可以用来测量
    - 建筑设计图
    - 手册
- 看不到对象真正的全局形状，因为许多面在视点中不可见

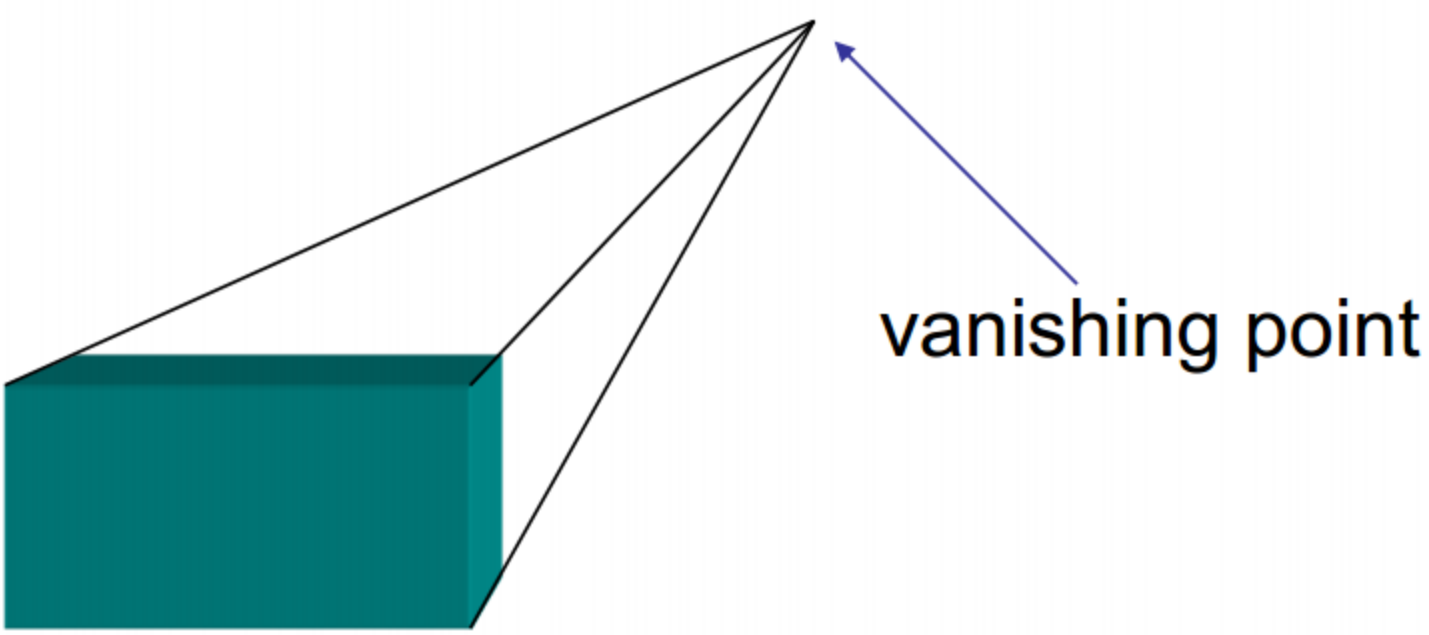
# 透视投影

- 投影线会聚于投影中心(COP)：尺寸缩小
- 经典视图中，观察者相对于投影平面对称
  – 投影中心和投影窗口确定一个对称的正棱锥

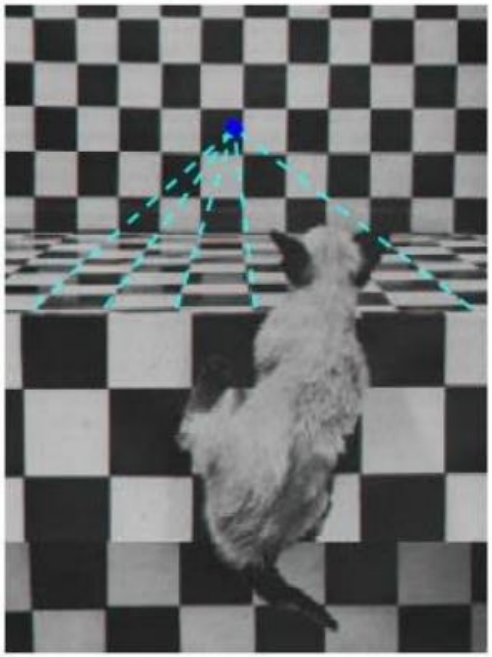


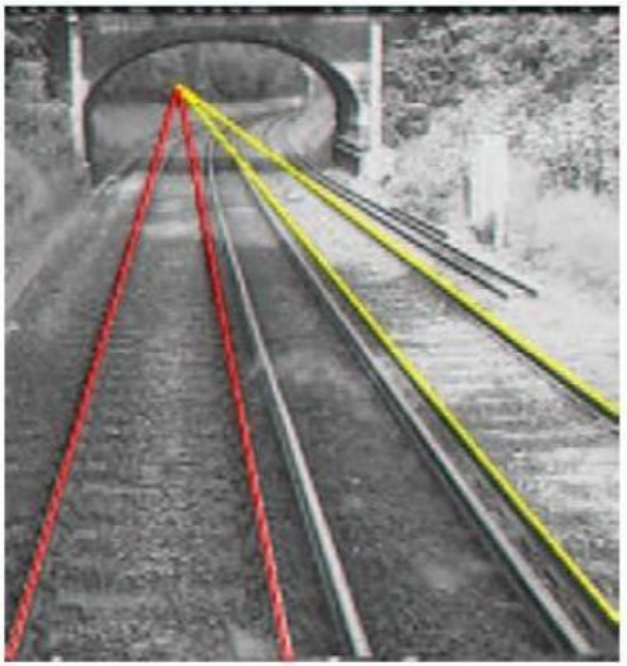
# 灭点（vanishing point）

- 对象上（不平行于投影面）的平行线在投影后交于一个灭点(vanishing point)
- 手工绘制简单透视投影时要利用这些灭点



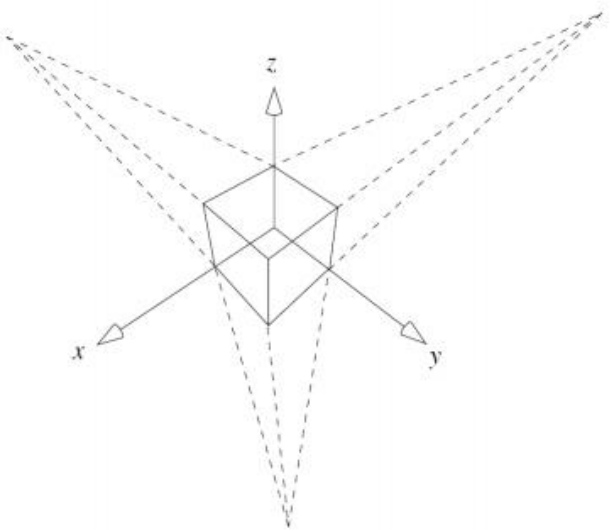
# 示例





# 三点透视

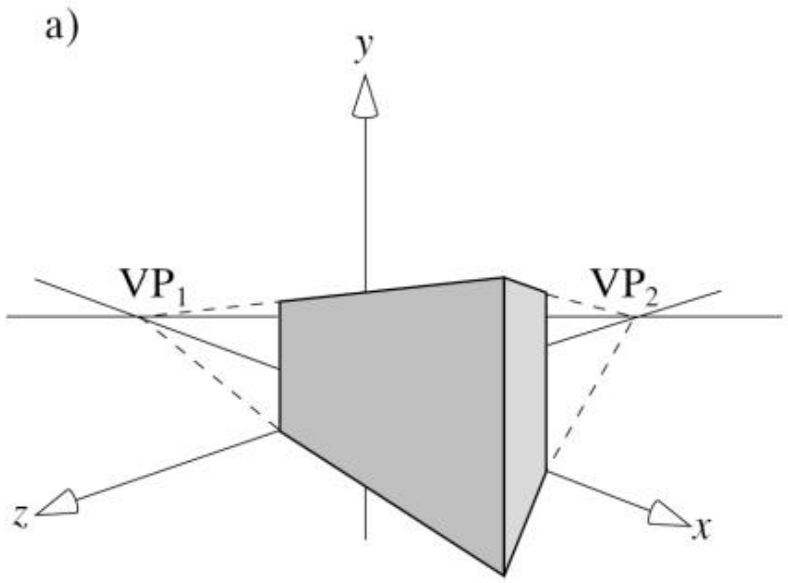
- 立方体的投影中有三个灭点




(a)

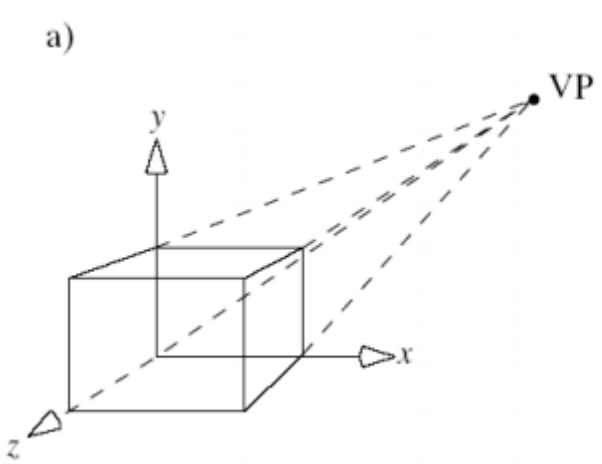
# 两点透视

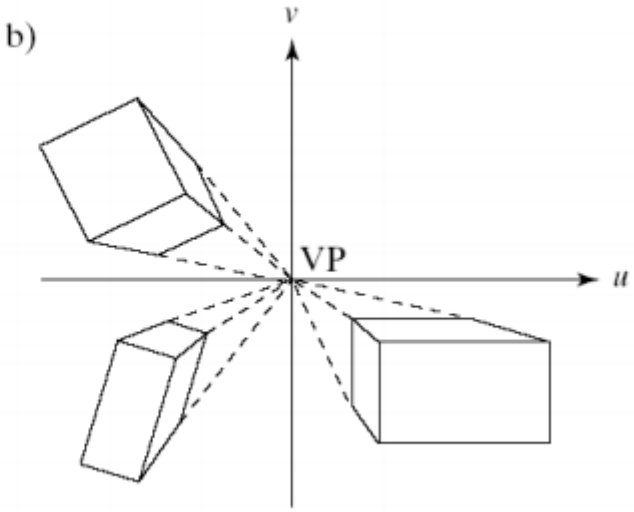
- 立方体的投影中有两个灭点




# 单点透视

- 立方体的投影中有一个灭点




# 优缺点

- 同样大小的对象，离视点越远，投影结果就越小(diminuition)
  - 看起来更自然
- 直线上等距的几点投影后不一定等距—非均匀缩短(nonuniform foreshortening)
  - 借助透视投影图测量尺寸较平行投影困难
- 只有在平行于投影面的平面上角度被保持
- 相对于平行投影而言，更难用手工进行绘制（但对计算机而言，没有增加更多的困难）
- 主要应用在动画等真实感图形领域

# 第五章 视图(Viewing)

本章节主要解决相机的设置问题

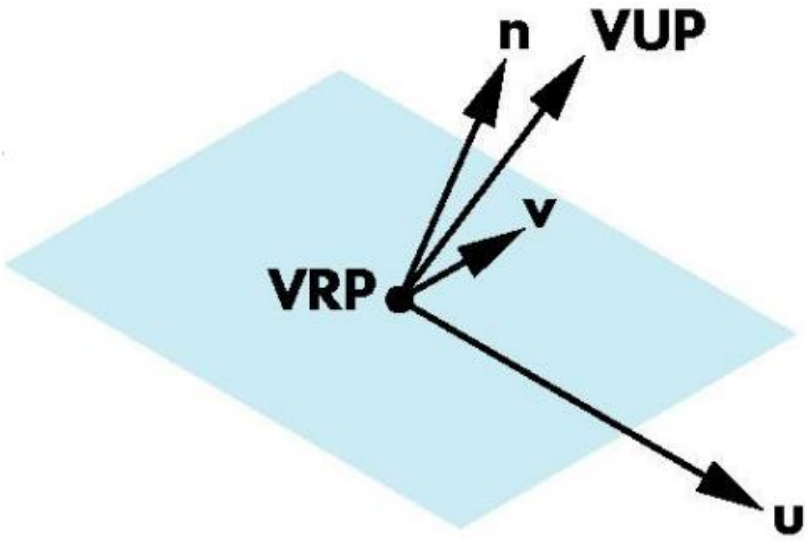
- 视图概念
- OpenGL中的视图设置
  - 描述相机方位的函数、矩阵
  - 描述相机投影的函数、矩阵
- 回顾、总结图形流水线的前半部分（顶点变换）

# gluLookAt

- 设置相机位置、方位
- 属于模型视图矩阵
  - gluLookAt设置相机的位置和方位
  - 其实可以等价于反方向设置物体的位置和方位
  - 因此：gluLookAt可以有glRotate和glTranslate等函数联合实现
- 如何求gluLookAt所对应的变换矩阵

# 先来回顾gluLookAt

- 照相机初始时位于世界标架的原点，方向指向z轴负方向
- gluLookAt在世界标架中描述照相机的方位
- 三组参数：相机中心,相机目标点,相机朝上上方位

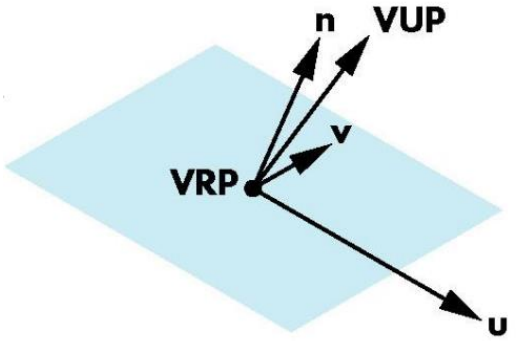


# VPN & VUP

- n给出投影面的方向，即平面的法向（视线方向）
- 只有平面的定向不能完全确定照相机的定向 –照相机还可以绕n方向旋转
- 只有给出了VUP，才完全确定了照相机的方向

# VUP如果不与平面平行呢？

- 不必要求VUP向量必定平行于投影面
- 把VUP投影到投影平面上得到向量**v**
  - **v**与**n**正交
  - 任何不平行与**v**的向量也可以指定为VUP



# 计算变换矩阵

设VRP点$\mathbf{p} = [x,y,z,1]^T$, 视平面法向 $\mathbf{n} = [n_x,n_y,n_z,0]^T$, 正向向量为 $\mathbf{v}_{up} = [v_{upx},v_{upy},v_{upz},0]^T$

$\mathbf{v} = \mathbf{v}_{up}-(\mathbf{v}_{up}\cdot\mathbf{n})/(\mathbf{n}\cdot\mathbf{n})\mathbf{n}$, 把$\mathbf{v}$, $\mathbf{n}$单位化

$\mathbf{u} = \mathbf{v} \times \mathbf{n}$

视图定位矩阵为

$$\begin{pmatrix} Ux & Uy & Uz & 0 \\ Vx & Vy & Vz & 0 \\ Nx & Ny & Nz & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & -Tx \\ 0 & 1 & 0 & -Ty \\ 0 & 0 & 1 & -Tz \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} Ux & Uy & Uz & -U\bullet T \\ Vx & Vy & Vz & -V\bullet T \\ Nx & Ny & Nz & -N\bullet T \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\mathbf{V} = \begin{bmatrix} u_x & u_y & u_z & -xu_x - yu_y - zu_z \\ v_x & v_y & v_z & -xv_x - yv_y - zv_z \\ n_x & n_y & n_z & -xn_x - yn_y - zn_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# 实例计算

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(0, 0, 0, 0, 0, -1, 0, 1, 0);
float arr[16];
glGetFloatv(GL_MODELVIEW_MATRIX, arr);
for(int i=0; i<4; ++i)
{
    for(int j=0; j<4; ++j)
    {
        printf("%f ", arr[j*4+i]);
    }
    printf("\n");
}
```

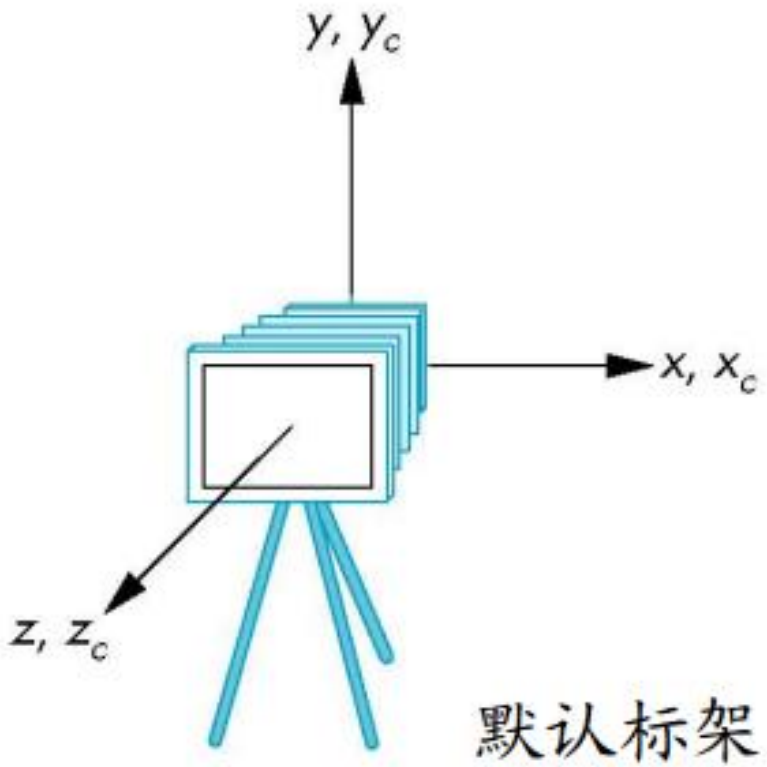
```
1.000000 0.000000 0.000000 0.000000
0.000000 1.000000 0.000000 0.000000
0.000000 0.000000 1.000000 0.000000
0.000000 0.000000 0.000000 1.000000
```

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(0, 0, 0, 0, 0, -1, 0, -1, 0);
float arr[16];
glGetFloatv(GL_MODELVIEW_MATRIX, arr);
for(int i=0; i<4; ++i)
{
    for(int j=0; j<4; ++j)
    {
        printf("%f ", arr[j*4+i]);
    }
    printf("\n");
}
```

```
-1.000000 0.000000 0.000000 0.000000
0.000000 -1.000000 0.000000 0.000000
0.000000 0.000000 1.000000 0.000000
0.000000 0.000000 0.000000 1.000000
```

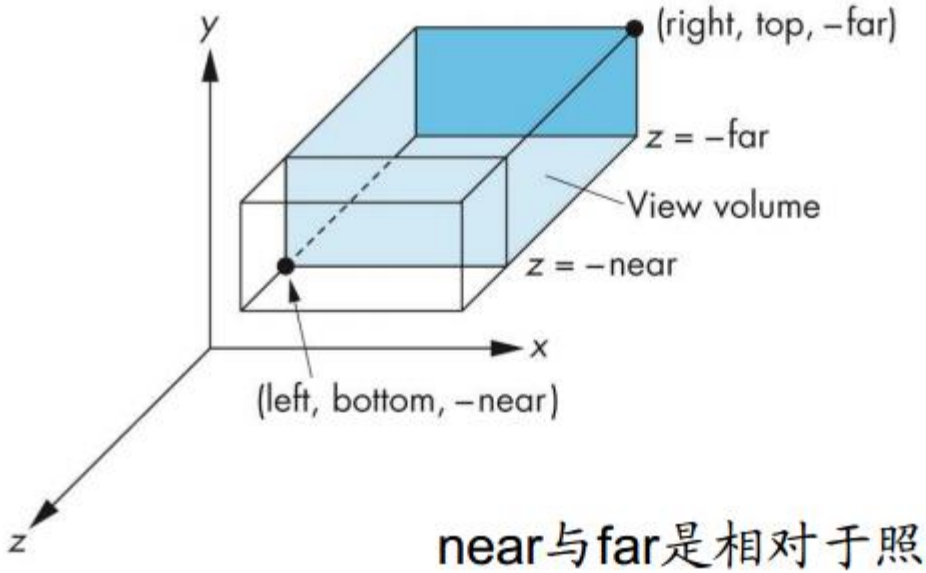
# 注意相机标架

- 默认相机标架是与世界标架重合
- 但是相机是朝z负方向看的



# 正交投影

- glMatrixMode(GL_PROJECTION)
- void **glOrtho**(GLdouble left, GLdouble right, Gldouble bottom, GLdouble top, GLdouble near, GLdouble far);
  – 视景体是与坐标轴平行的长方体
  – near和far可取正值、零或负值，但near和far不应相同




near与far是相对于照相机而言的

# 规范视景体

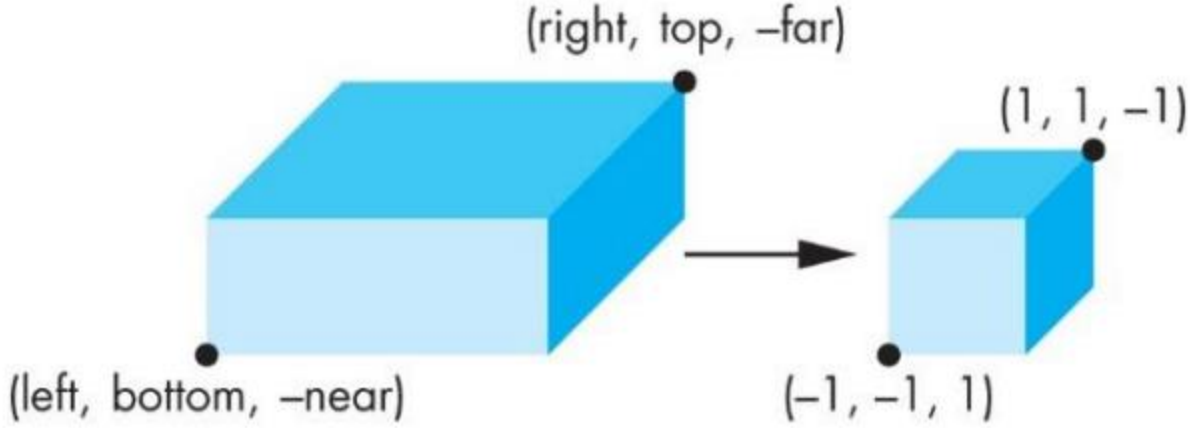
- OpenGL缺省的视景体是中心在原点，边长为2的立方体，相当于调用
  **glMatrixMode(GL_PROJECTION);**
  **glLoadIdentity();**
  **glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);**
  称这个视景体为规范视景体（canonical view volume）
- 采用规范视景体的优点：
  - 同一流水线支持平行投影和透视投影
  - 简化了裁剪过程

# 正交规范化

- 规范化 -> 求出把指定裁剪体转化为默认裁剪体的变换

  glOrtho(left, right, bottom, top, near, far)

- 作用：把视景体内的坐标规范化到-1和1之间




近裁剪面z=-near映射到z=-1平面，
远裁剪面z=-far映射到z=1平面

# 正交规范化矩阵

近裁剪面z=-near映射到z=-1平面，
远裁剪面z=-far映射到z=1平面

分两步

– 平移：把中心移到原点，对应的变换为

T(–(left+right)/2, –(bottom+top)/2, (near+far)/2))

– 缩放：进行放缩从而使视景体的边长为2

S(2/(right-left), 2/(top – bottom), -2/(far-near))

$$\mathbf{P}=\mathbf{ST}=\begin{bmatrix} \frac{2}{right-left} & 0 & 0 & -\frac{right+left}{right-left} \\ 0 & \frac{2}{top-bottom} & 0 & -\frac{top+bottom}{top-bottom} \\ 0 & 0 & \frac{-2}{far-near} & -\frac{far+near}{far-near} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# 规范化之后，再做投影

令z = 0

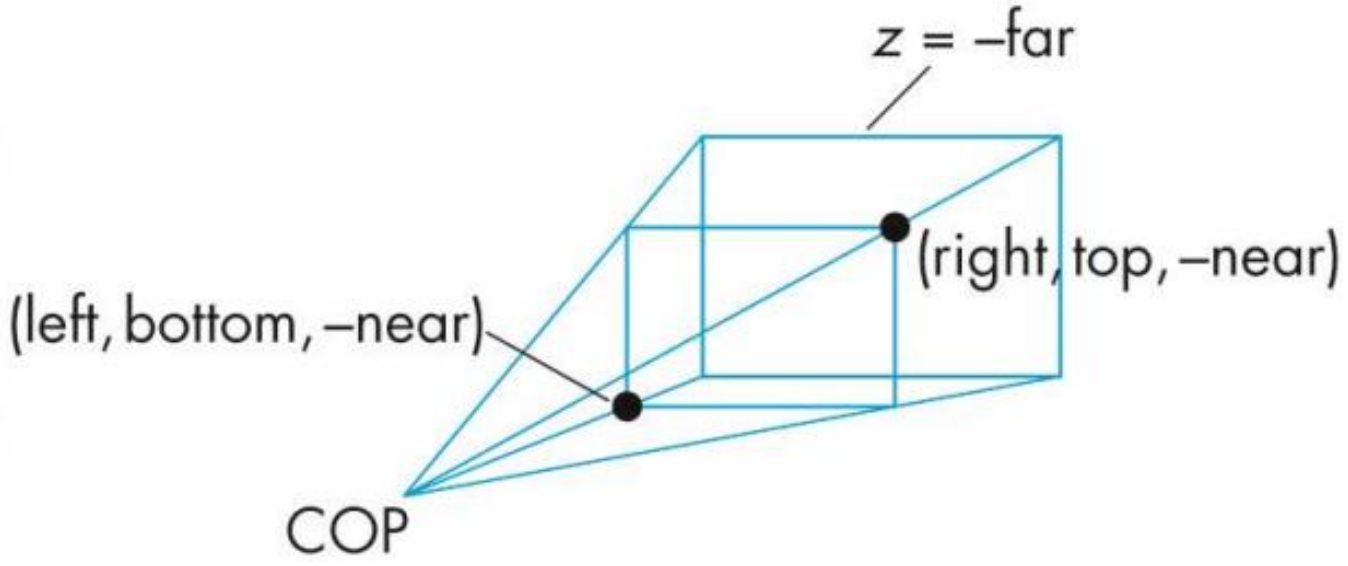
这等价于如下的齐次坐标变换

$$\mathbf{M}_{orth} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

从而在4D中一般的正交投影为$\mathbf{P} = \mathbf{M}_{orth}\mathbf{ST}$

# 透视投影

- glFrustum可以定义非对称视景体，而gluPerspective只能定义对称视景体



# glFrustum对应的矩阵(推导详见教材)

$$\mathbf{P}=\mathbf{NSH}=\begin{bmatrix} \frac{2near}{right-left} & 0 & \frac{right+left}{right-left} & 0 \\ 0 & \frac{2near}{top-bottom} & \frac{top+bottom}{top-bottom} & 0 \\ 0 & 0 & -\frac{far+near}{far-near} & -\frac{2far*near}{far-near} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

# glPerspective对应的矩阵(推导详见教材)

- gluPerspective(fovy, aspect, near, far)
  - 对称性： left = -right, bottom = -top
  - 三角学： top = near * tan(fovy)
  - 宽高比： right = top * aspect

$$\mathbf{P} = \mathbf{NSH} = \begin{bmatrix} \frac{near}{right} & 0 & 0 & 0 \\ 0 & \frac{near}{top} & 0 & 0 \\ 0 & 0 & -\frac{far + near}{far - near} & -\frac{2far * near}{far - near} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

# 思考？

- 经过透视投影之后？得到什么？
- 直接得到规范化后的投影位置？三个维度上的范围都在[-1,1]之间
- 齐次坐标计算完之后，用第四个分量w去除前三个分量 （透视除法）
- 需要在这个时候做透视除法么？

# 第五章 视图(Viewing)

本章节主要解决相机的设置问题

- 视图概念
- OpenGL中的视图设置
  - 描述相机方位的函数、矩阵
  - 描述相机投影的函数、矩阵
- 回顾、总结图形流水线的前半部分（顶点变换）

# 图形管线

- 输入
  - 图元类型 三角形
  - 三个顶点位置 [世界坐标系]
- 分别对三个顶点进行模型视图变换 [由世界坐标系变换到视点坐标系]
- 分别对三个顶点进行投影变换 [由视点坐标系变换到裁剪坐标系]
  - 对每个点：得到的是？
  - 4维的齐次坐标
- 分别对每个顶点计算颜色（光照） [下一节讲]

# 图形管线

- 对超出视景体的部分进行裁剪
  - 对每个点的裁剪区域是[-1,1] x [-1,1] x [-1,1] 么 ？
  - 裁剪时需要考虑什么？ [增加顶点，为增加出来的顶点计算颜色]
- 透视除法
  - 为何在这里做透视除法，而不是前面？
  - [由裁剪坐标系变换到规范化坐标系]
- glViewport设置透视除法之后的点应该投影到窗口中的什么位置
  - [由规范化坐标系到窗口坐标系]

- 至此，我们都是对每个顶点单独计算

- 光栅化 (确定三角形占据那些像素)
  - 像素填充

- 片段处理
  - 哪些图元被遮挡，哪些保留？
  - 像素颜色计算

# 从世界坐标到视口坐标

- 输入
  - 图元类型 三角形
  - 三个顶点位置 [世界坐标系]
- 分别对三个顶点进行模型视图变换 [由世界坐标系变换到视点坐标系]
- 分别对三个顶点进行投影变换 [由视点坐标系变换到裁剪坐标系]
  - 对每个点：得到的是？
  - 4维的齐次坐标
- 分别对每个顶点计算颜色（光照） [下一节讲]

# 图形管线

- 对超出视景体的部分进行裁剪
  - 对每个点的裁剪区域是[-1,1] x [-1,1] x [-1,1] 么 ？
  - 裁剪时需要考虑什么？ [增加顶点，为增加出来的顶点计算颜色]
- 透视除法
  - 为何在这里做透视除法，而不是前面？
  - [由裁剪坐标系变换到规范化坐标系]
- glViewport设置透视除法之后的点应该投影到窗口中的什么位置
  - [由规范化坐标系到窗口坐标系]

# glViewport的作用

- glViewport(Ox, Oy, width, height)
  - 把规范化视景体中的内容“填入”指定的范围内

  - 规范化视景体坐标(x,y,z)
  - WinX = Ox + (x+1)/2*width
  - WinY = Oy + (y+1)/2*height
  - WinZ = (z+1)/2 (规范化到[0,1]之间)

  - 注意：这里的WinY坐标和窗口系统的坐标（例如鼠标获取）朝向是反的

# gluProject

- 学会此函数的用法
- 要求自行实现顶点变换的结果应与此函数结果完全一致

# 本节任务

- 实现myLookAt （选）
- 实现myOrtho
- 实现myFrustum （选）
- 实现myPerspective （选）
- 实现完整的顶点变换管线