中国科学技术大学
University of Science and Technology of China

计算机图形学

计算机学院 黄章进
zhuang@ustc.edu.cn

第六章 观察

6.1 经典视图

6.2 计算机视图

6.3 投影矩阵

6.1 经典视图

- 平行投影
- 透视投影

为什么需要经典视图？

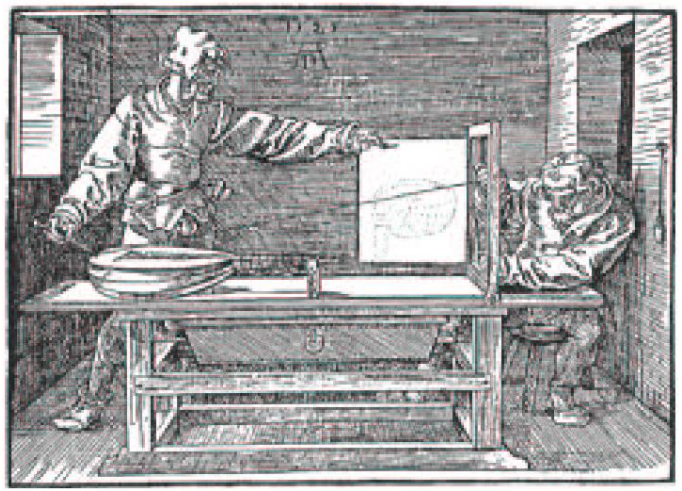
- 传统由手工操作的制图工作现在可以用计算机模拟
  - 电影中的动画，建筑图纸，机器零件图纸
  - 这些领域中需要不同的经典视图
    - 等轴测投影(isometrics)，正视图(elevation)，透视
- 考察经典视图和计算机视图之间的联系，可以更好地理解API所采用方法的优点和不足
  - 两种视图的基本元素是相同的：对象、观察者、投影线和投影平面
  - 都允许观察者相距对象无穷远
  - 经典照相机vs虚拟照相机

经典视图

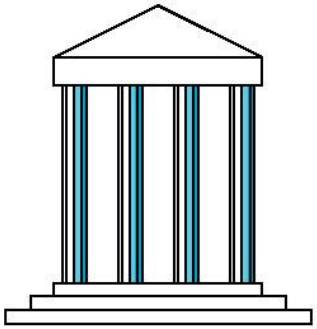
- 视图中需要三个基本要素
  - 一个或多个对象
  - 观察者，带有一个投影面
  - 从对象到投影平面的投影线
- 经典视图就是基于这些要素之间的关系的
  - 观察者选取一个对象，以期望看到的方位进行定向
- 每个对象都假定是用平面的基本多边形构造出来的
  - 如：建筑物、多面体、锻造物

平面几何投影

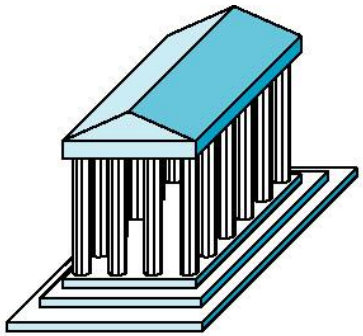
- 即投影到平面上的标准投影
- 投影线为直线，这些直线
  - 会聚于投影中心，或者
  - 彼此平行
- 这种投影保持共线性
  - 但不一定保角
- 在诸如地图绘制等应用中需要非平面投影



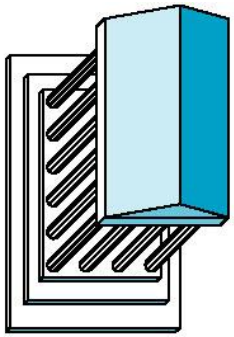
经典投影



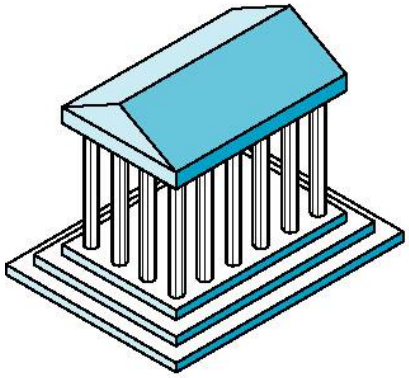
前视图
Front elevation



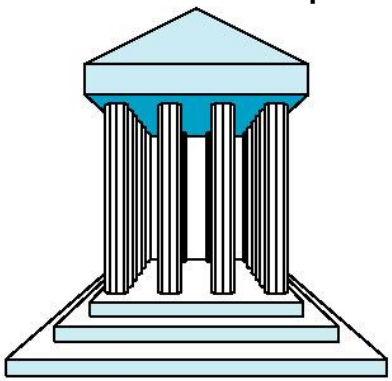
前斜视图
Elevation oblique



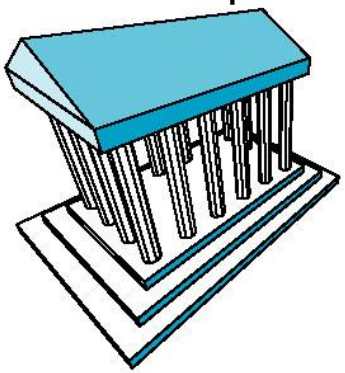
顶斜视图
Plan oblique



正等轴测图
Isometrics



单点透视
One-point perspective



三点透视
Three-point perspective

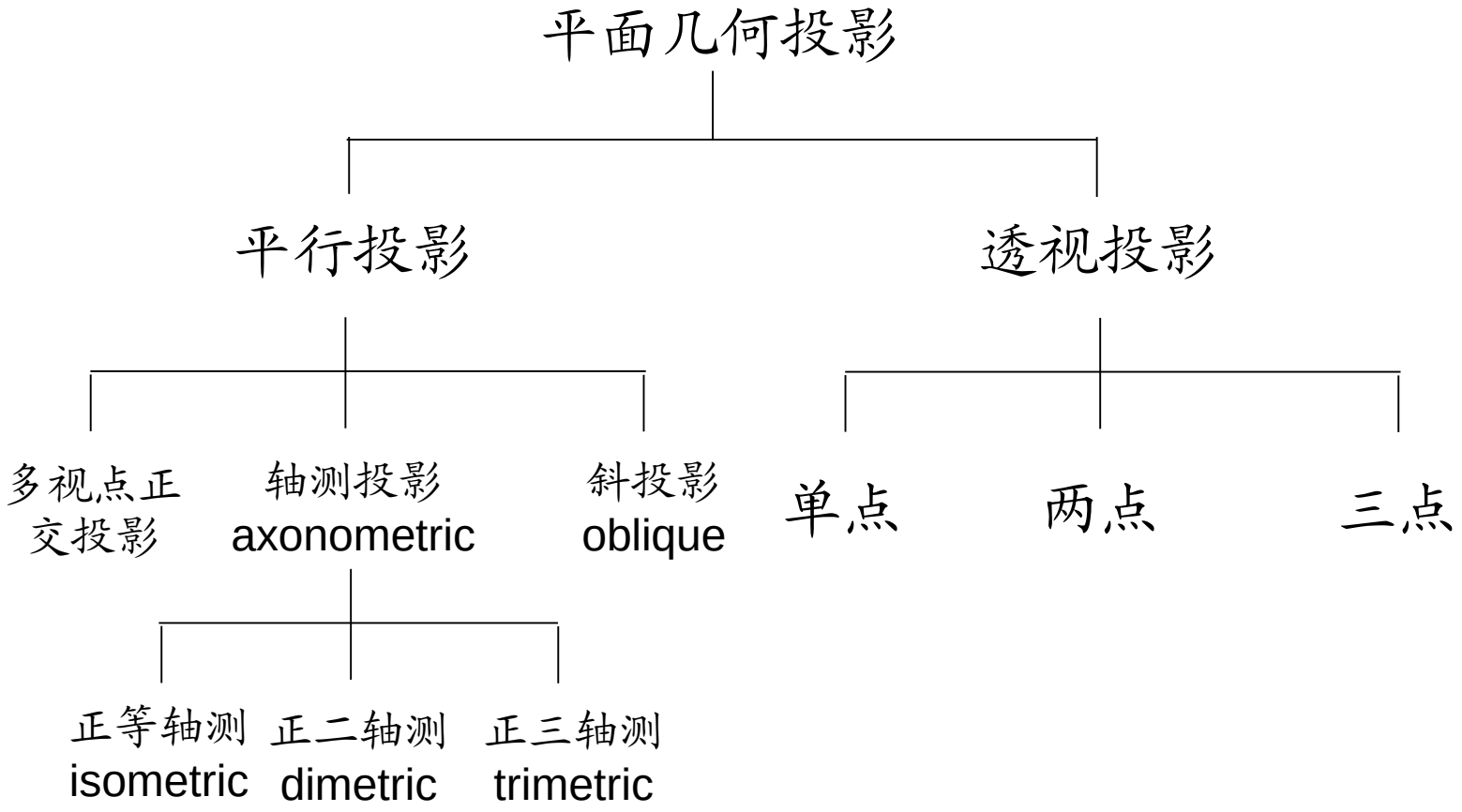
主视面

- 在诸如建筑业等实际应用中，所观察的对象通常由许多平坦面构成。
- 这些面中任一个都可以认为是一个主视面(principal faces)，从而进行定位
  - 对于规则物体，例如房屋，按照通常的方式可以定义前、后、左、右、顶、底等面
  - 许多对象上都有几个面相交于直角，从而可以得到三个正交的方向，称为主视方向

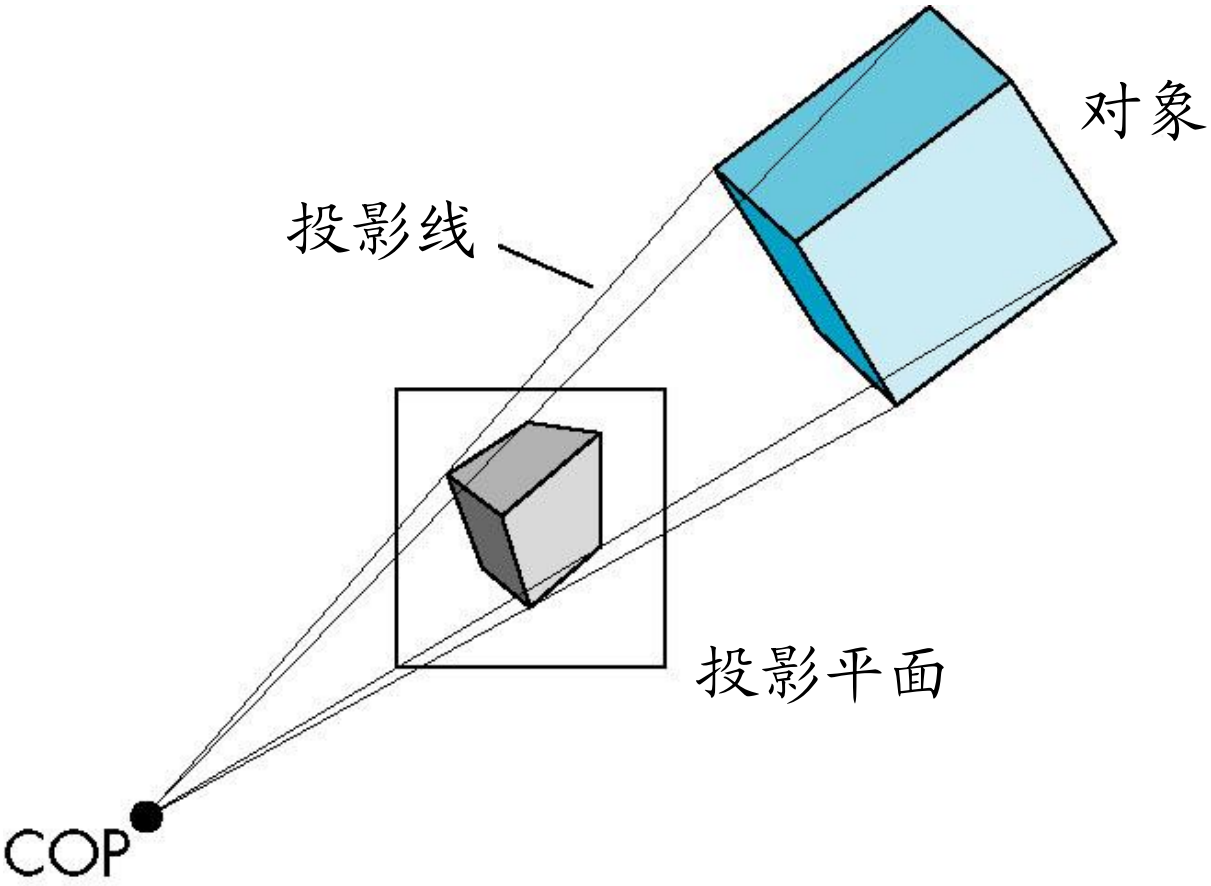
透视投影与平行投影

- 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

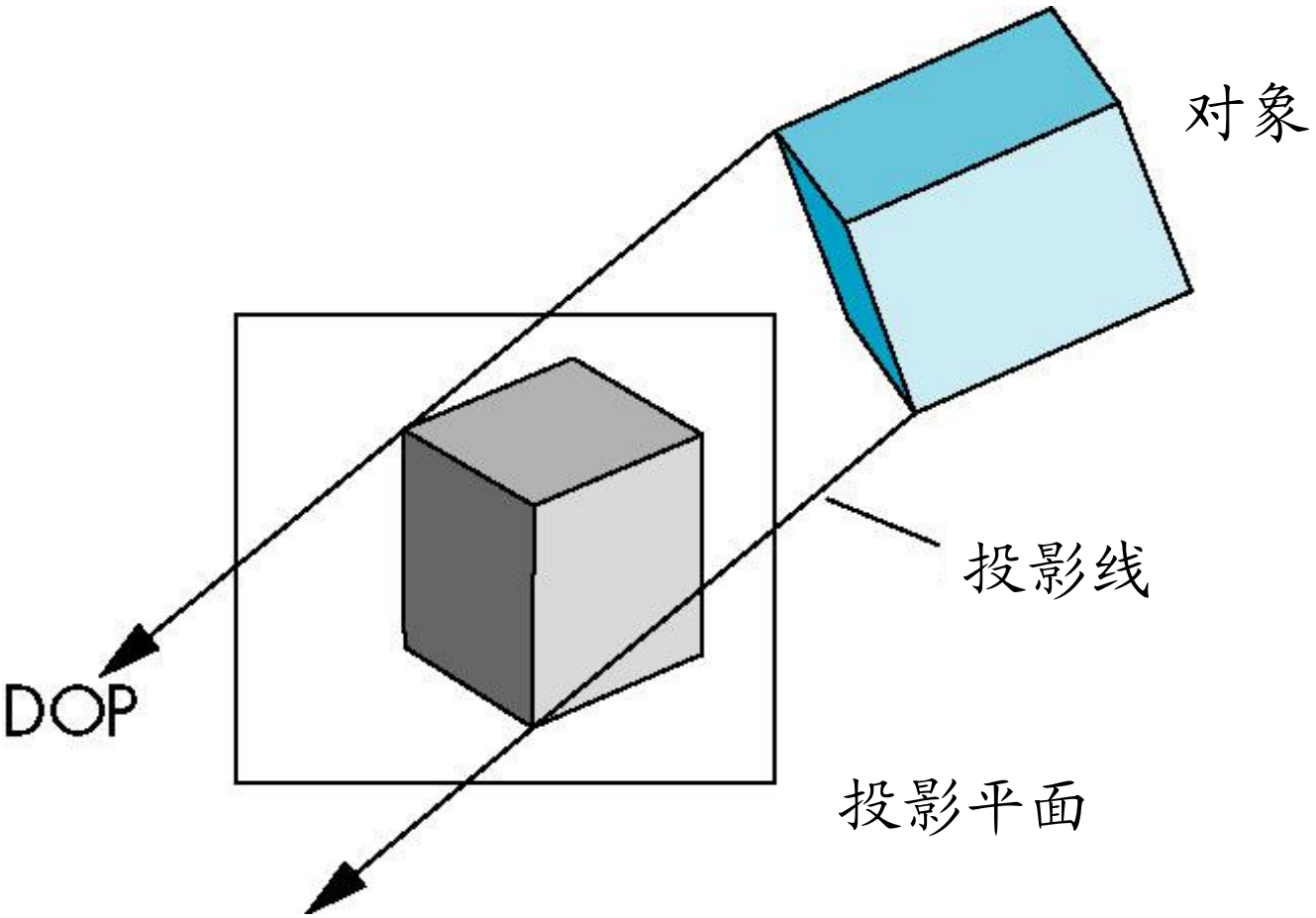
平面几何投影分类图


平面几何投影
平行投影
透视投影
多视点正交投影
轴测投影 axonometric
斜投影 oblique
单点
两点
三点
正等轴测 isometric
正二轴测 dimetric
正三轴测 trimetric

透视投影

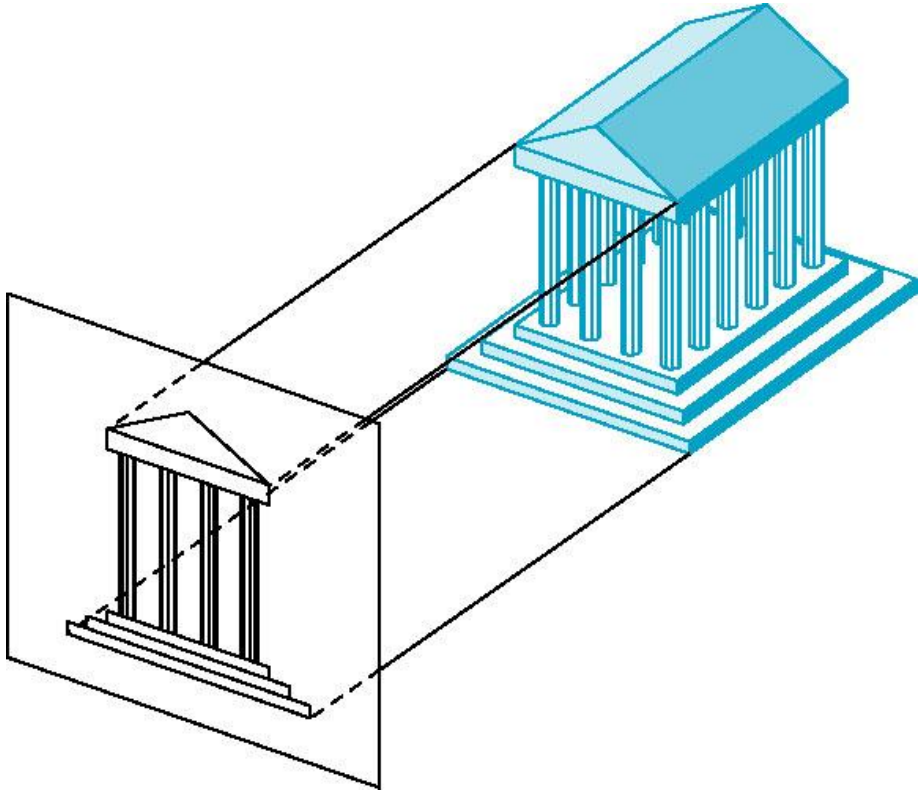


平行投影



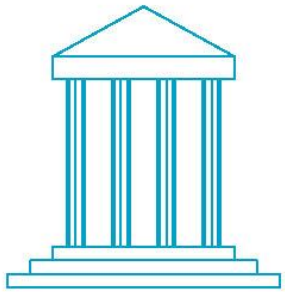
正交投影

- 投影线垂直于投影平面



多视点正交投影

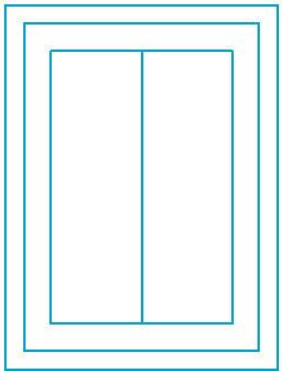
- 投影平面平行于某个主视面
- 通常从前面、顶部和侧面进行投影
  - 三视图
    - 正视图（主视图）
    - 俯视图
    - 侧视图（左视图）



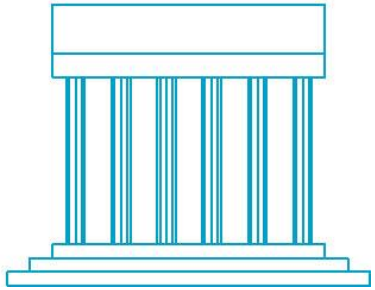
前面

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

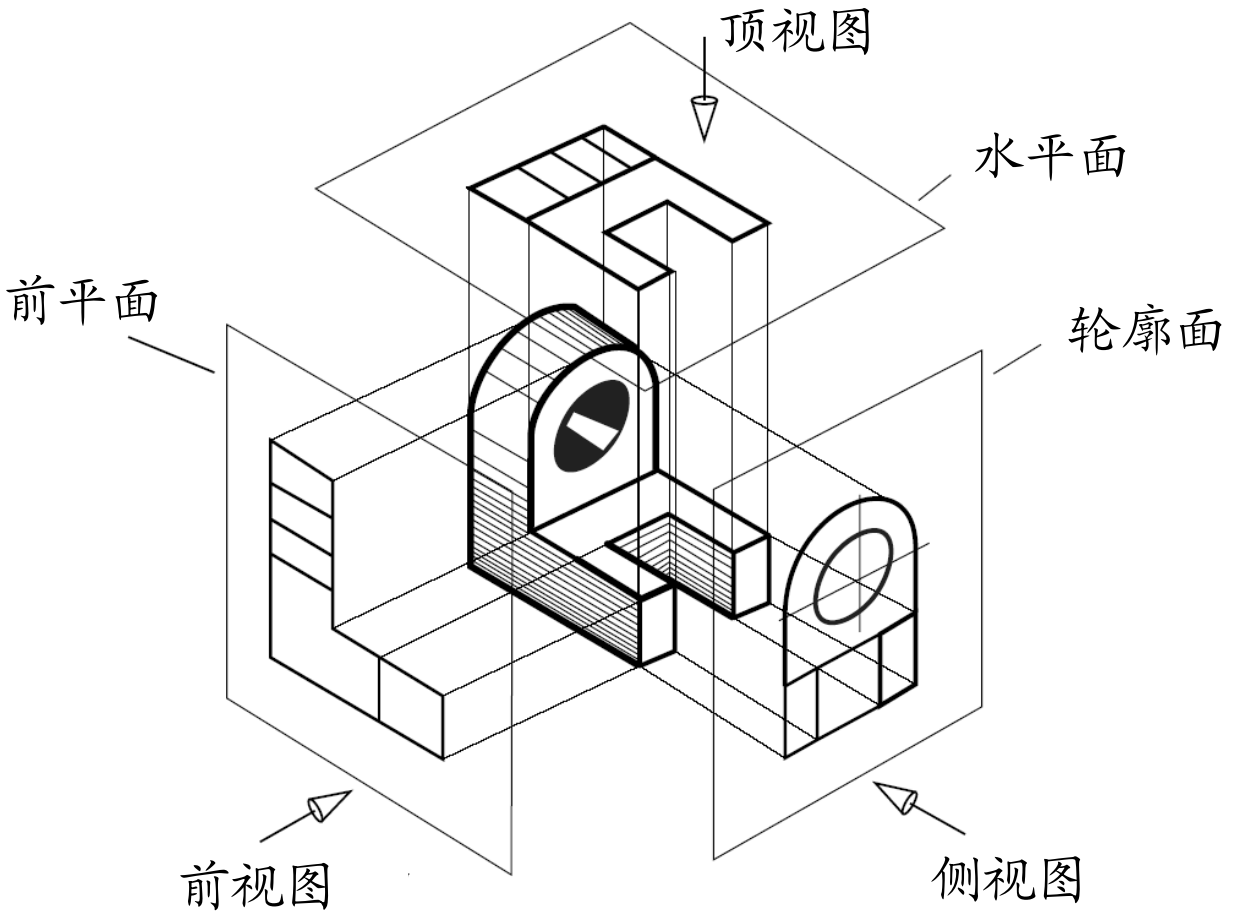


顶部



侧面

机器零件的多视点视图

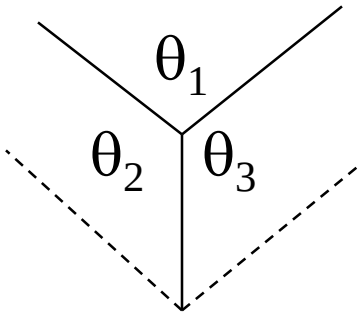


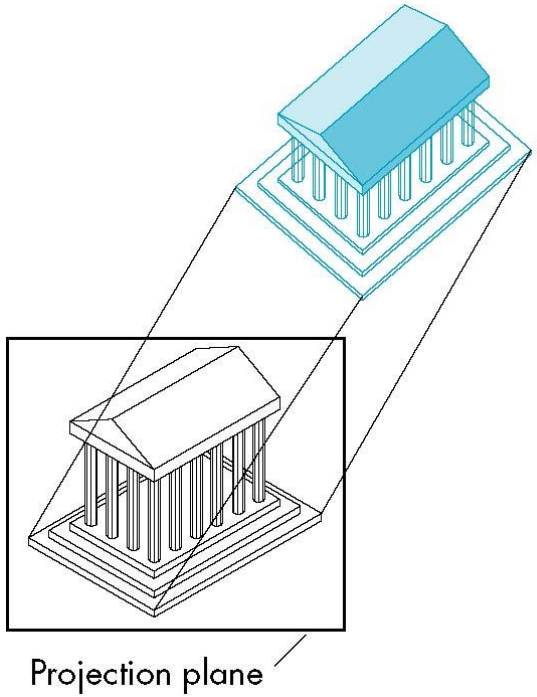
优缺点

- 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
- 不能看到对象真正的全局形状，因为许多面在视点中不可见
  - 经常加上正等轴测图

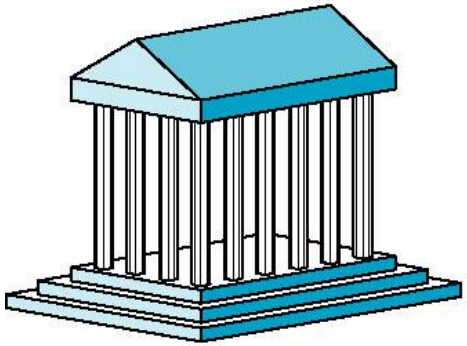
轴测投影

- 投影线垂直于投影平面，但投影面相对于对象的主视面有一定的夹角
  - 根据对立方体进行投影时一个角点处有多少个角相等进行分类
    - 没有：正三轴测
    - 两个：正二轴测
    - 三个：正等轴测

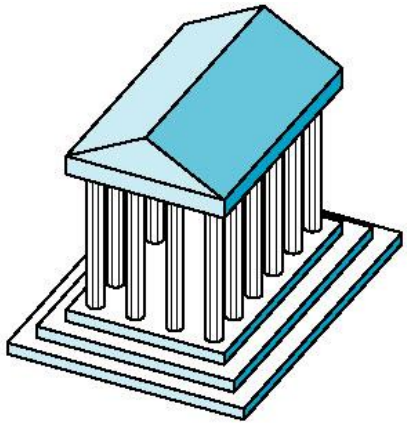




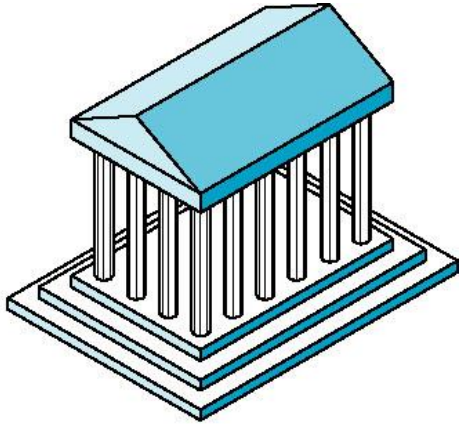
轴测投影的示例



正二轴测



正三轴测



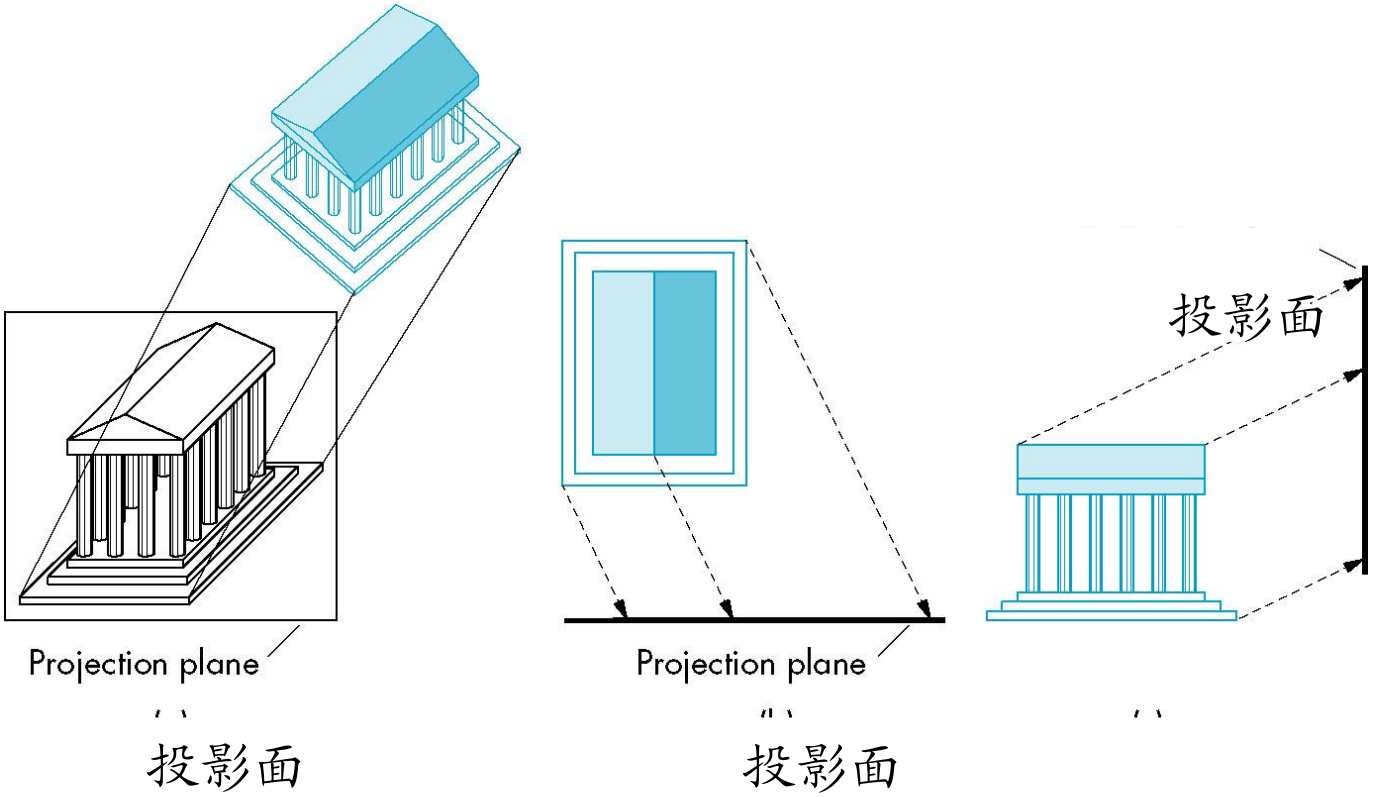
正等轴测

优缺点

- 直线段长度被缩短(foreshortened)，但可以求出收缩因子
- 保持直线但不保角
  - 圆所在平面如果不平行于投影面，它的投影为椭圆
- 可以见到盒状对象的三个主视面
- 会导致某些观察错觉
  - 平行线看起来不平行
- 不是很真实，因为远的对象与近的对象具有同样的收缩因子
- 在CAD应用中经常用到

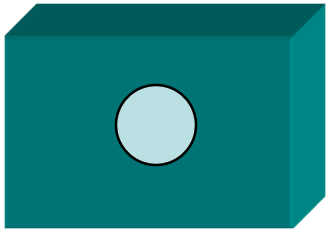
斜投影

- 投影线与投影面之间的关系任意



优缺点

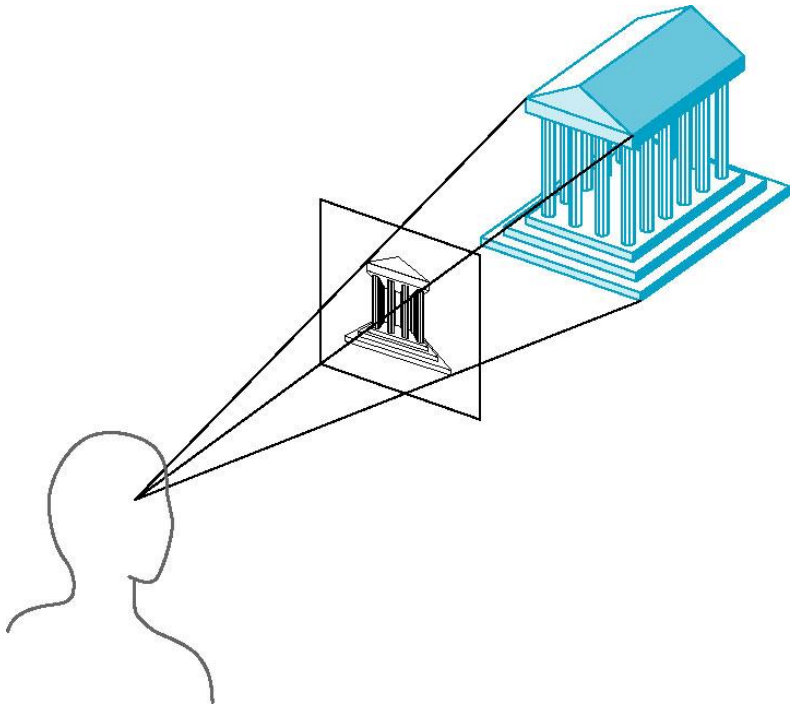
- 可以选取某个角度，以便强调特定面
  - 建筑行业：plan oblique, elevation oblique
- 在平行于投影面的面上的角是保持的，但我们仍然可以见到其它侧面



- 在实际世界中，只能利用特殊相机（风箱式照相机或特殊透镜）做到这一点

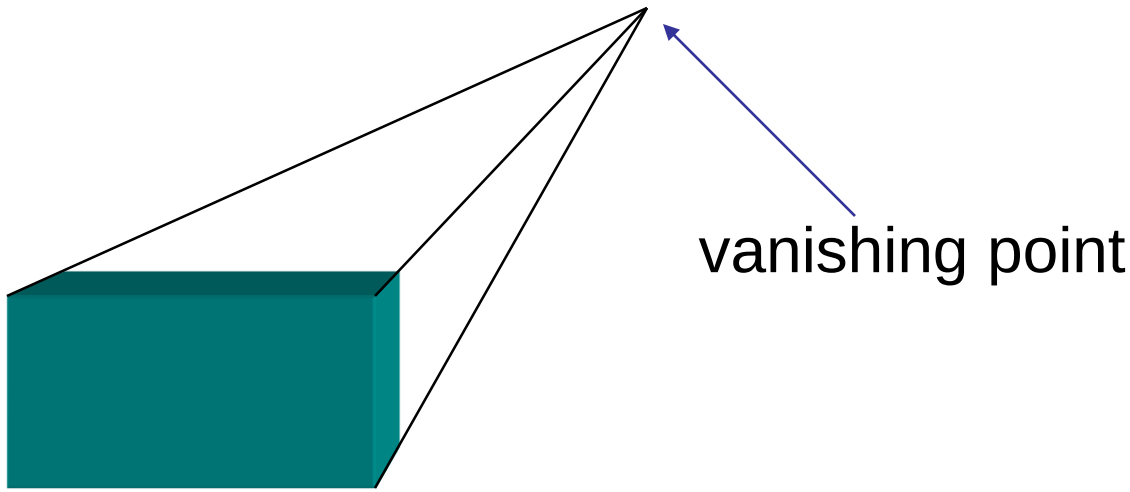
透视投影

- 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

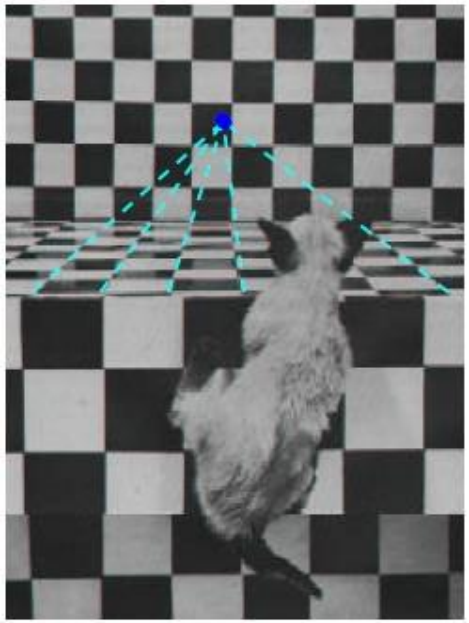


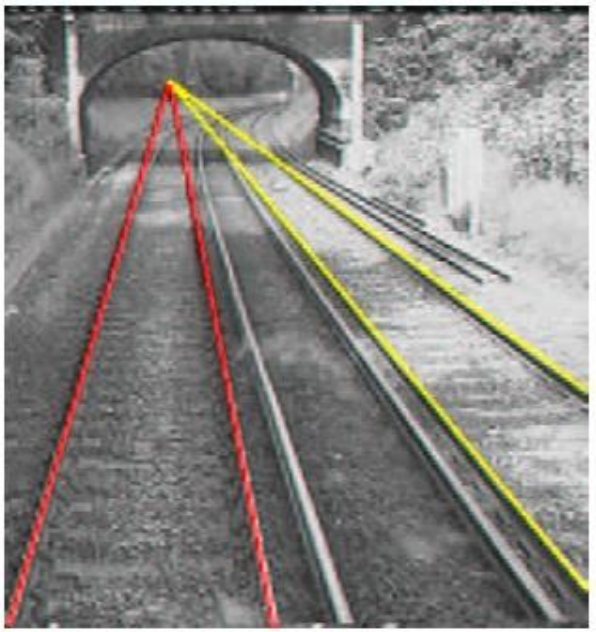
灭点

- 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



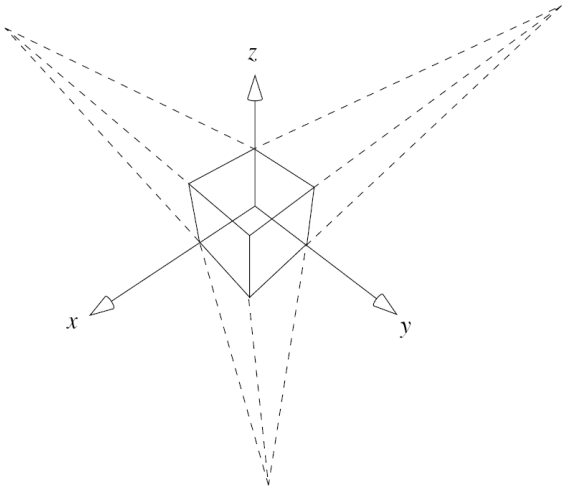
示例





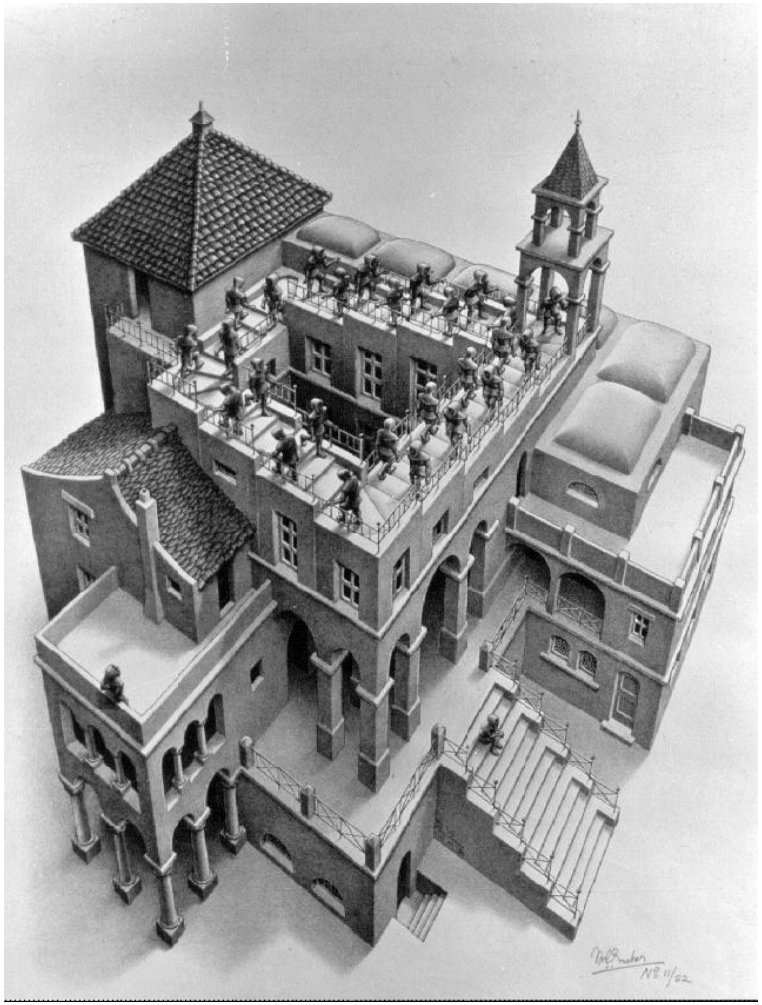
三点透视

- 没有主视面平行于投影面
- 立方体的投影中有三个灭点



(a)

错觉图



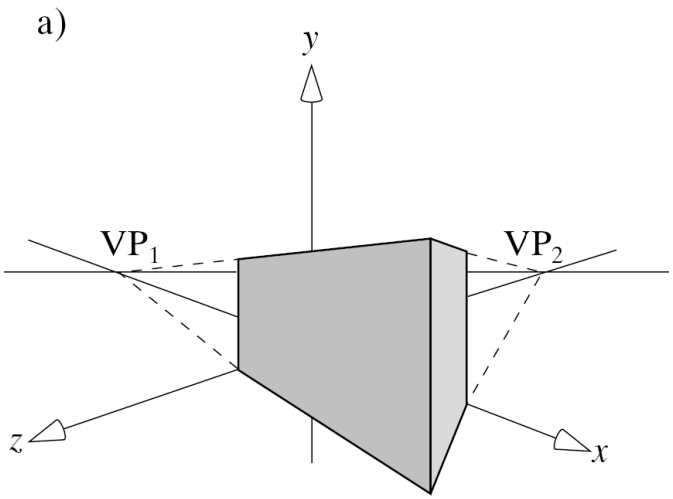
M.C. Esher(1898-1972)

http://www.mcescher.com/

Ascending and Descending, 1960（左图）

两点透视

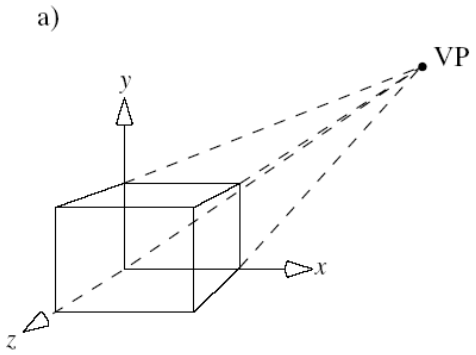
- 一个主视方向平行于投影面
- 立方体的投影中有两个灭点

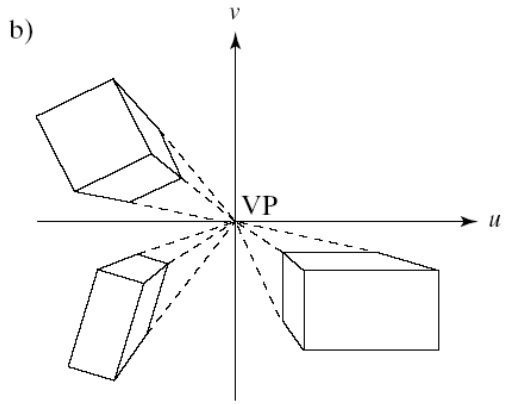


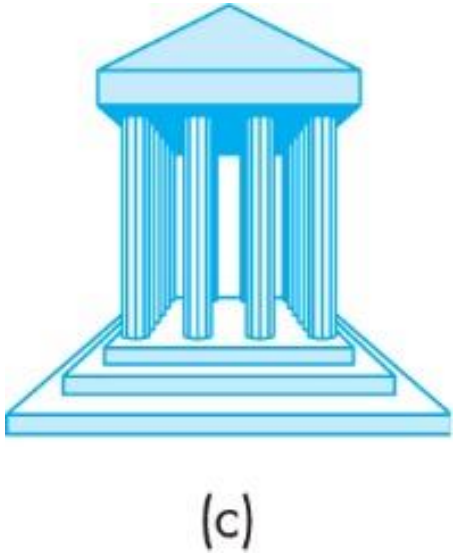
(b)

单点透视

- 一个主视面平行于投影平面
  - 两个主视方向平行于投影面
- 立方体的投影中有一个灭点





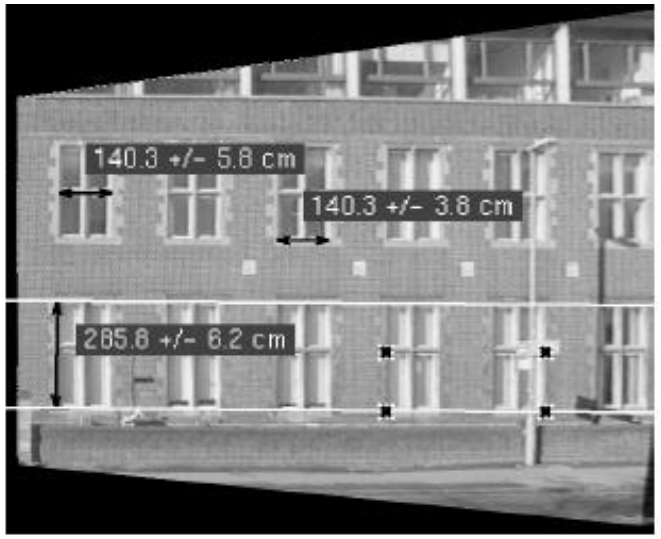


优缺点

- 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

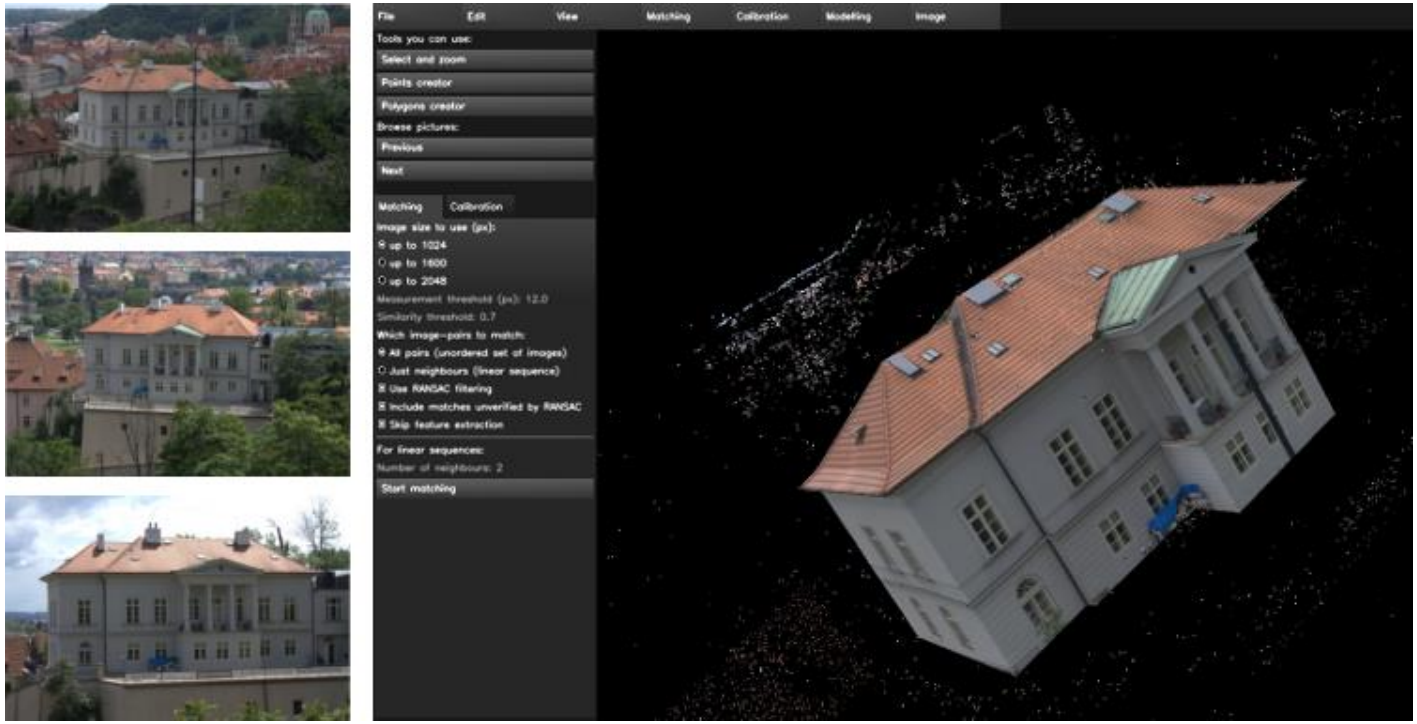
应用

- 把投影图像返回到真实尺寸



应用

- 基于图像的建模与渲染(image-based modeling and rendering)
  - http://insight3d.sourceforge.net/



6.1 经典视图

6.2 计算机视图

6.3 投影矩阵

6.2 计算机视图

- 视图API
- 投影的数学表示
- OpenGL中的投影

计算机视图

- 视图有三个功能，都在流水线结构中实现
  - 定位照相机
    - 设置模型－视图矩阵
  - 设置镜头
    - 设置投影矩阵
  - 裁剪
    - 设置视景体

虚拟照相机

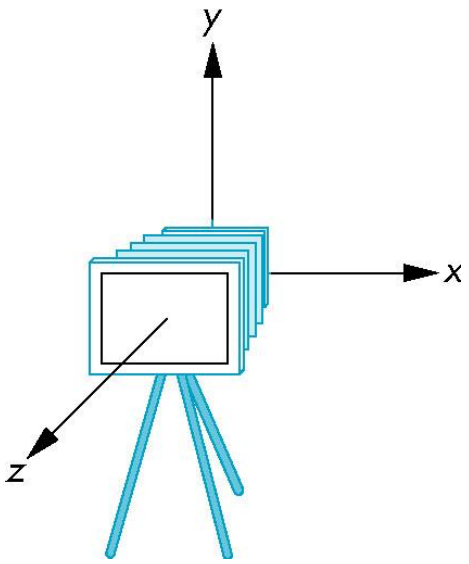
- 计算机视图是基于虚拟照相机模型的，原则上可以实现所有的经典视图
- 所有的经典视图都依赖于对象、观察者和投影线之间的特定位置关系；而在计算机图形学中，强调的则是对象定义与照相机参数设置两者之间的独立性
- 在OpenGL中可以指定透视投影或者正交投影，但OpenGL不能判断一个透视投影图是单点、两点还是三点透视
  - 这需要知道对象与照相机之间的位置关系

最终的选择

- 偏好对象定义与照相机设置之间的独立性
- 如果应用程序需要特定类型的视图，那么应当仔细确定照相机相对于对象的位置
  - 因为在透视投影中，从射影几何的角度来说，肯定有三个灭点，只是有些灭点在无穷远

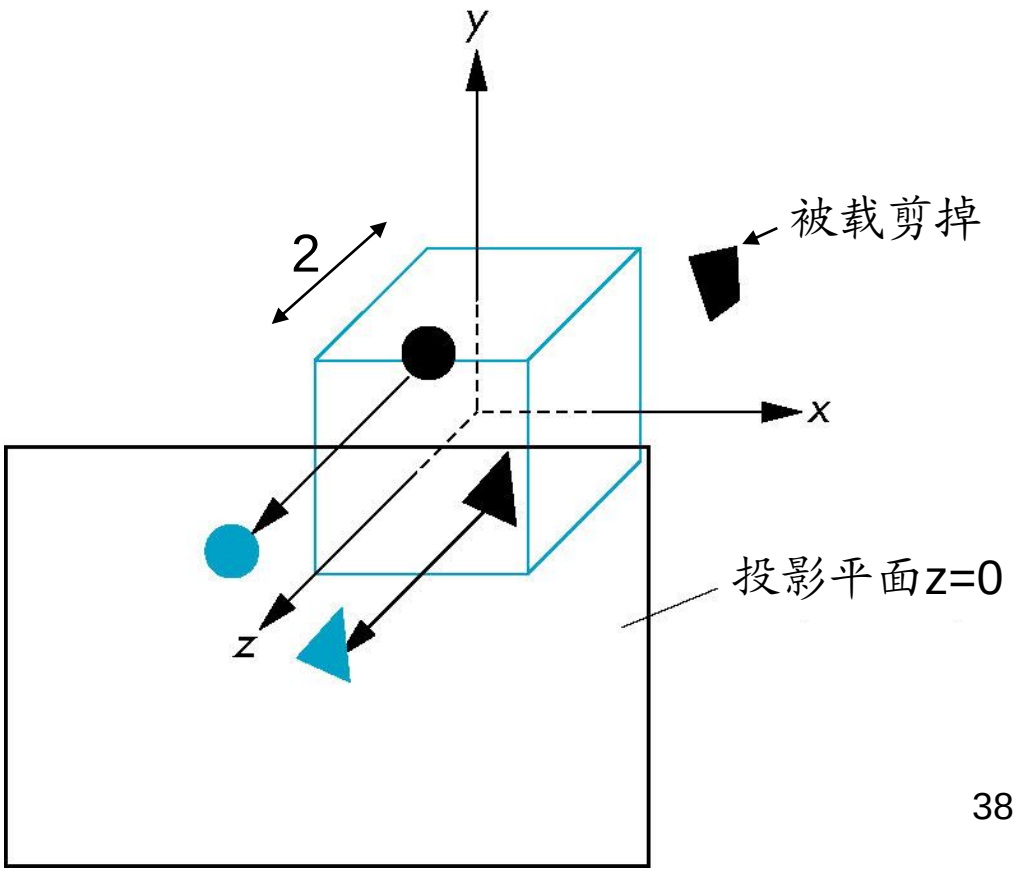
OpenGL中的照相机

- 在OpenGL中，初始的世界标架和照相机标架相同
  - 初始的模型－视图矩阵是单位阵
- 照相机位于原点，并指向z轴的负向
- 默认的视景体是一个中心在原点的边长为2的立方体
  - 初始的投影矩阵是单位阵



缺省投影

- 默认的投影是正交投影
  - 投影方向沿着z轴
  - 投影平面为z=0



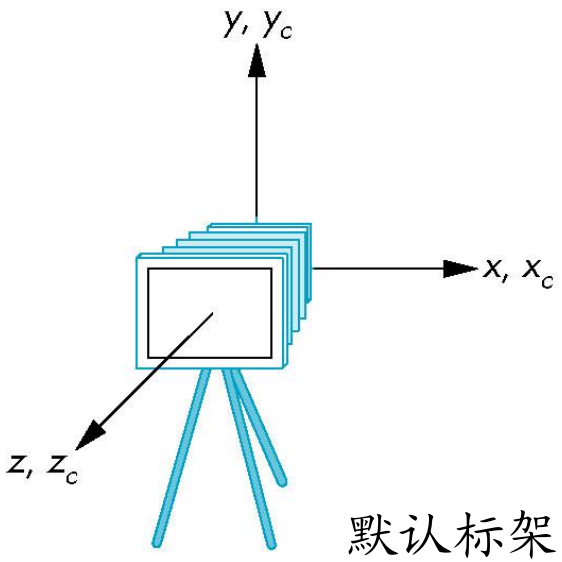
定义可见的对象

- 在缺省的照相机设置下，为了使定义的对象可见，必须使对象的位置和尺寸与默认视景体相匹配
  - 通常可以对数据进行适当的平移和各向同性放缩

移动照相机标架

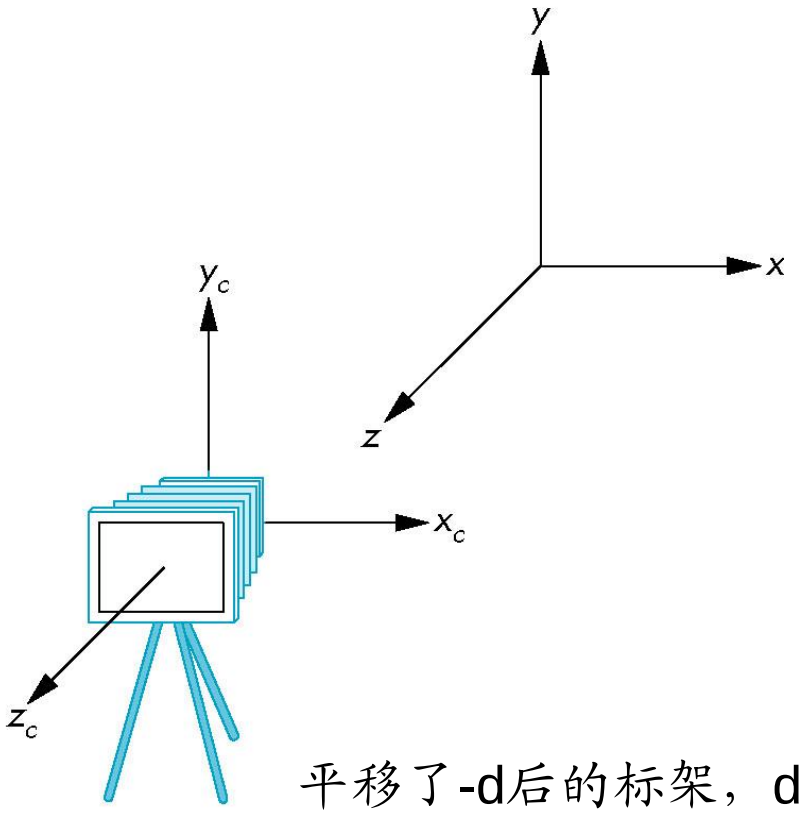
- 如果想看到具有更大的正z坐标的对象，我们可以
  - 把照相机沿z轴正向移动
    - 平移照相机标架
  - 把对象沿z轴负向移动
    - 移动世界标架
- 两者是完全等价的，都是由模型－视图矩阵确定的
  - 需要平移T(0.0, 0.0, -d);
  - 此处 d > 0

移动照相机标架



默认标架

(a)

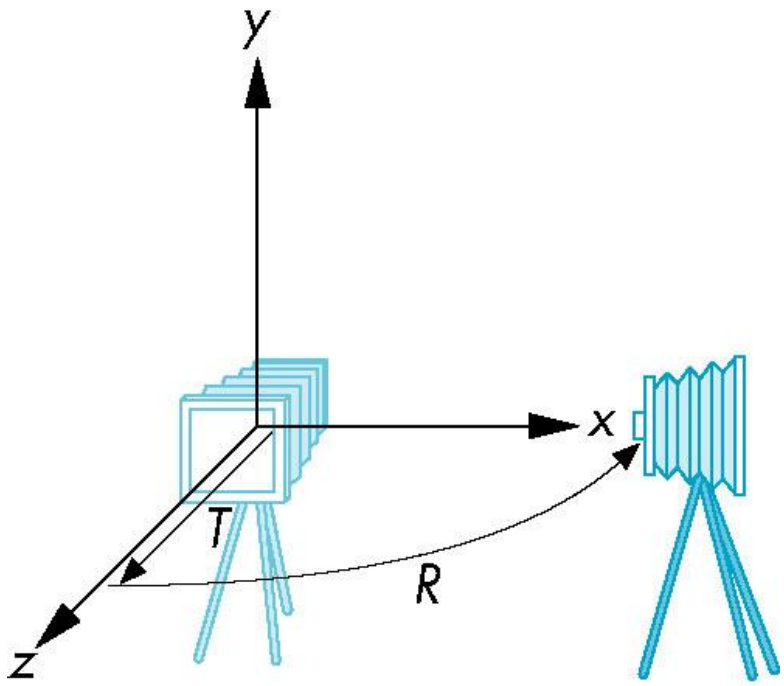


平移了-d后的标架，d>0

(b)

移动照相机标架

- 可以利用一系列旋转和平移把照相机定位到任意位置
- 例如，为了得到侧视图
  - 旋转照相机: R
  - 把照相机从原点移开: T
  - 模型－视图矩阵C = TR



模型—视图矩阵

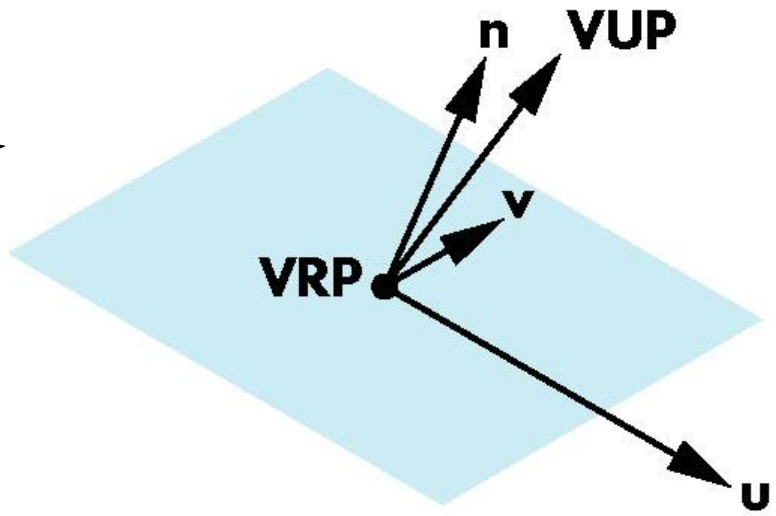
- 模型－视图矩阵是OpenGL状态的一部分
- 任何时刻的模型－视图矩阵包含了照相机标架与世界标架的位置关系
- 虽然表面上看把模型与视图矩阵结合为一个矩阵会导致一些混淆，但仔细体会这种流水线体系就会发现其中的优势
  - 如果把照相机也看作具有几何属性的对象，那么改变对象位置和定向的变换当然对照相机的位置和定向相对于其它对象也发生改变
- 通常希望在定义场景中的对象之前定位照相机
  - 应用到照相机的变换所生成的图像看起来和我们期望的正好相反

一种视图API

- 为了实现某种投影，需要经过复杂的计算得到变换的构成
- 可以采用PHIGS和GKS-3D中的方法来定位照相机
- 在世界标架中描述照相机的方位
- 照相机初始时位于世界标架的原点，方向指向z轴负方向

照相机参数

- 把照相机放到视图参考点(view-reference point, VRP)
  - VRP在世界标架中指定



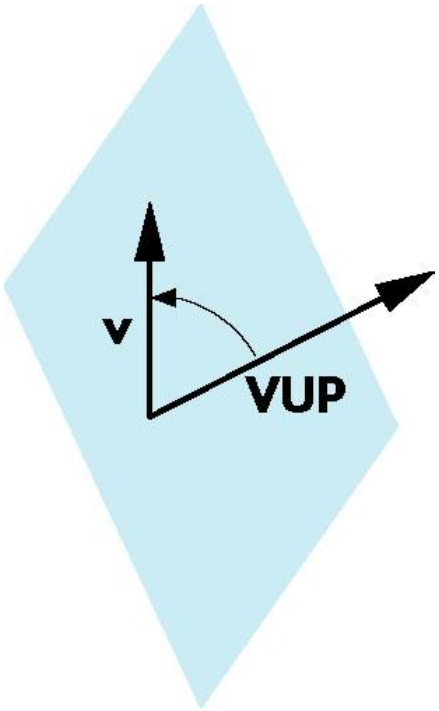
- 照相机定向
  - 视图平面法向**n** (view-plane normal, VPN)
  - 观察正向向量 (view-up vector, VUP)

VPN & VUP

- VPN给出投影面的方向，即平面的法向
- 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VPN方向旋转
- 只有给出了VUP，才完全确定了照相机的方向

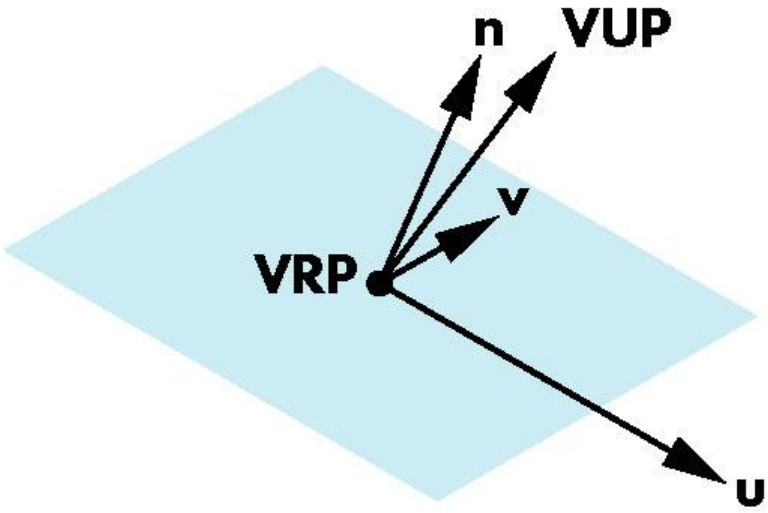
VUP

- 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



视图标架

- 假设**u = v x n**，由**(u,v,n)**构成的正交坐标系称为视图坐标系或**uvn**坐标系
- 视图坐标系加上VRP构成视图标架
- 从世界标架到视图标架的变换矩阵称为视图定位矩阵
  - 对应于模-视变换矩阵中的视图变换部分



视图定位矩阵

- 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

$\mathbf{v}\cdot\mathbf{n} = 0$

$\mathbf{v} = \alpha\,\mathbf{n} + \beta\,\mathbf{v}_{up}$，取$\beta=1$

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

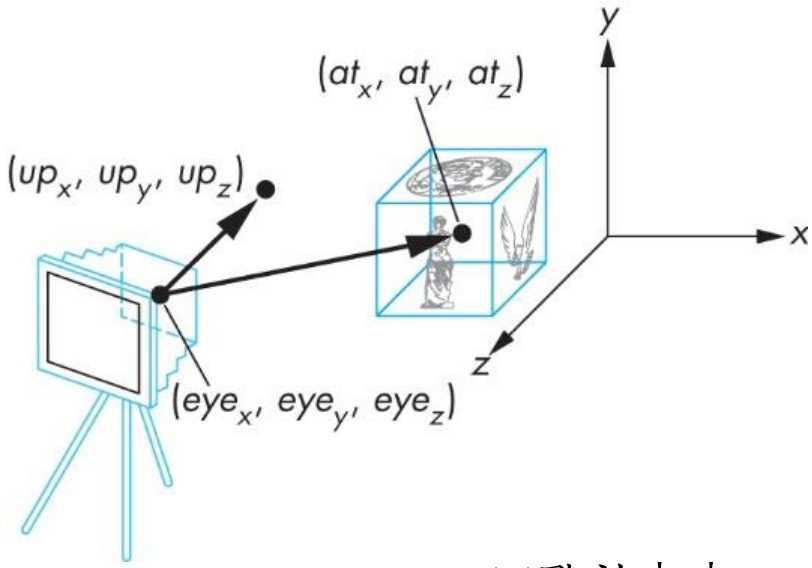
LookAt()函数

- 在GLU库中包含了函数gluLookAt()，提供了创建定位照相机所用的模型－视图矩阵的简单方法
- 注意在设置中需要一个向上的方向
- 需要初始化，即加载单位阵
  - 也可以与模型变换复合在一起
- 例如：轴对齐的立方体的正等轴测投影

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(1.,1.,1.,0.,0.,0.,0.,1.,0.);
```

gluLookAt()

- void **gluLookAt**(GLdouble eyex, GLdouble eyey, GLdouble eyez, GLdouble atx, GLdouble aty, GLdouble atz, GLdouble upx, GLdouble upy, GLdouble upz);
  - eye：视点，VRP
  - at：参考点，VPN = eye-at
  - up：观察正向VUP




```
gluLookAt(0.,0.,0.,0.,0.,-100.,0.,1.,0.);//默认相机
gluLookAt(1.,1.,1.,0.,0.,0.,0.,1.,0.);//正等轴测投影
```

gluLookAt()与其它变换

- 用户可以自己定义模型－视图矩阵实现同样的功能
- 从概念上可以把gluLookAt()作为照相机的定位(视图变换)，而把后续的其它变换作为对象的定位(模型变换)

物理照相机

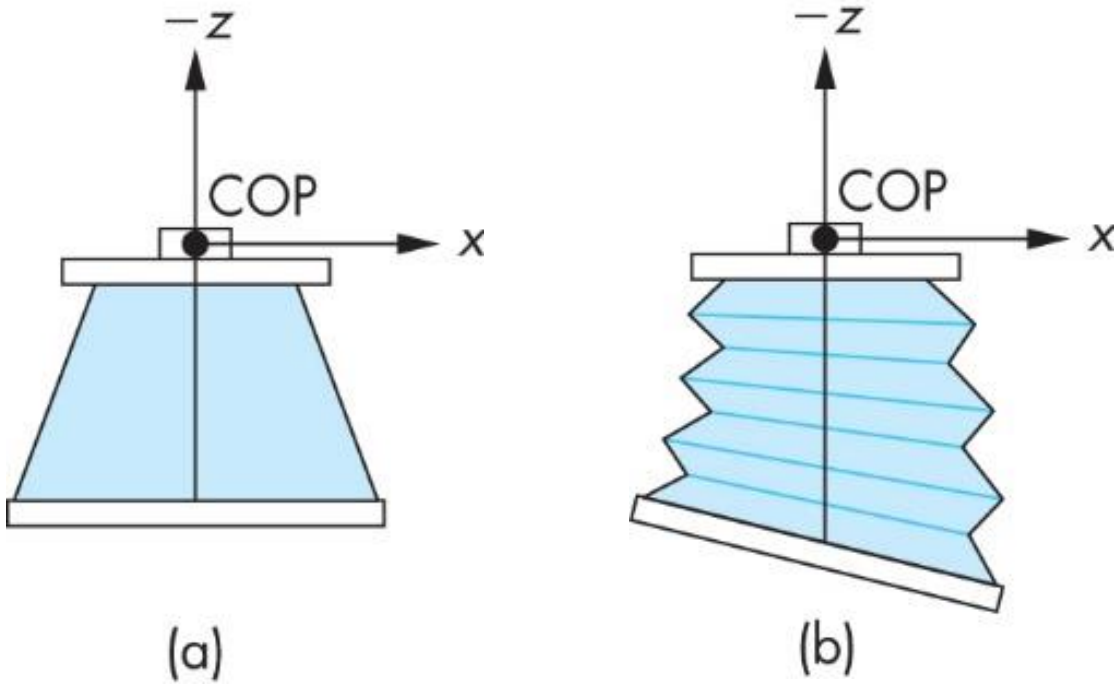
- 在定位后仍然可以选择镜头
- 镜头与底片的尺寸共同决定了在照相机前面多大范围的对象出现在最终的照片上
- 广角镜头可以使离照相机近的对象看起来比离照相机远的对象大，长焦镜头则会近似得到平行投影的效果

计算机中的照相机

- 可以选择投影的类型和视图参数
- 绝大多数API提供不同的函数用于定义平行投影和透视投影
- OpenGL也是如此，不过两者是在同一个流水线体系中实现
  - 可以用glLoadMatrix()设置投影矩阵

两种照相机模型

照相机位于视点标架的原点，指向z轴负方向



背面与镜头不平行

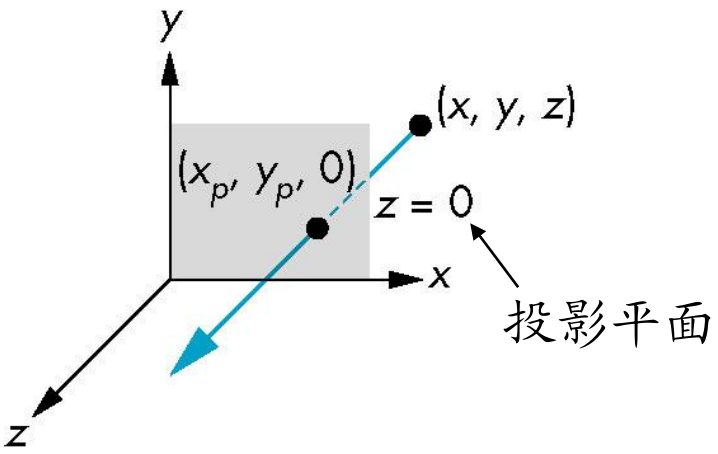
投影与规范化

- 在视点（照相机）标架中默认的投影是正交投影
- 对于在默认视景体内的点：

$$x_p = x,\ y_p = y,\ z_p = 0$$

- 大多数图形系统应用视图规范化的过程
  - 通过由投影矩阵确定的变换把所有其它的视图转化为默认视图
  - 从而可以对所有的视图采用同样的流水线体系

齐次坐标表示



$$x_p = x$$

$$y_p = y$$

$$z_p = 0$$

$$w_p = 1$$

默认的正交投影矩阵

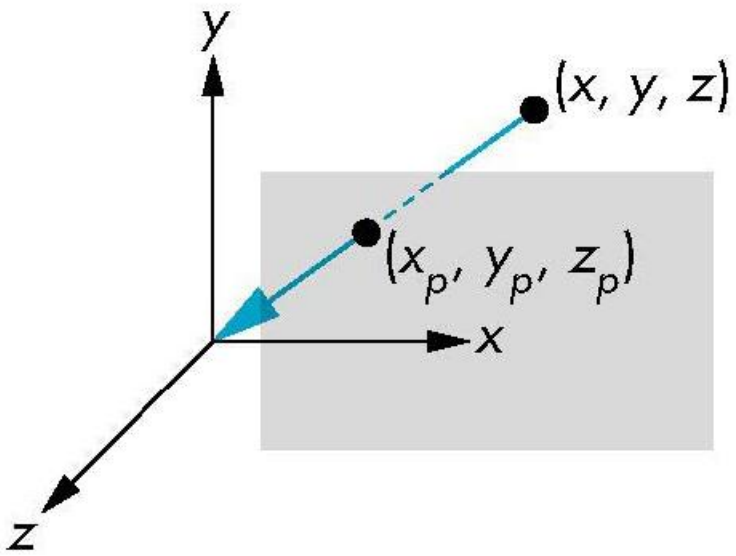
$$\mathbf{p}_p = \mathbf{M}\mathbf{p}$$

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

在实际应用中可以令$\mathbf{M} = \mathbf{I}$, 然后把对角线第三个元素置为零

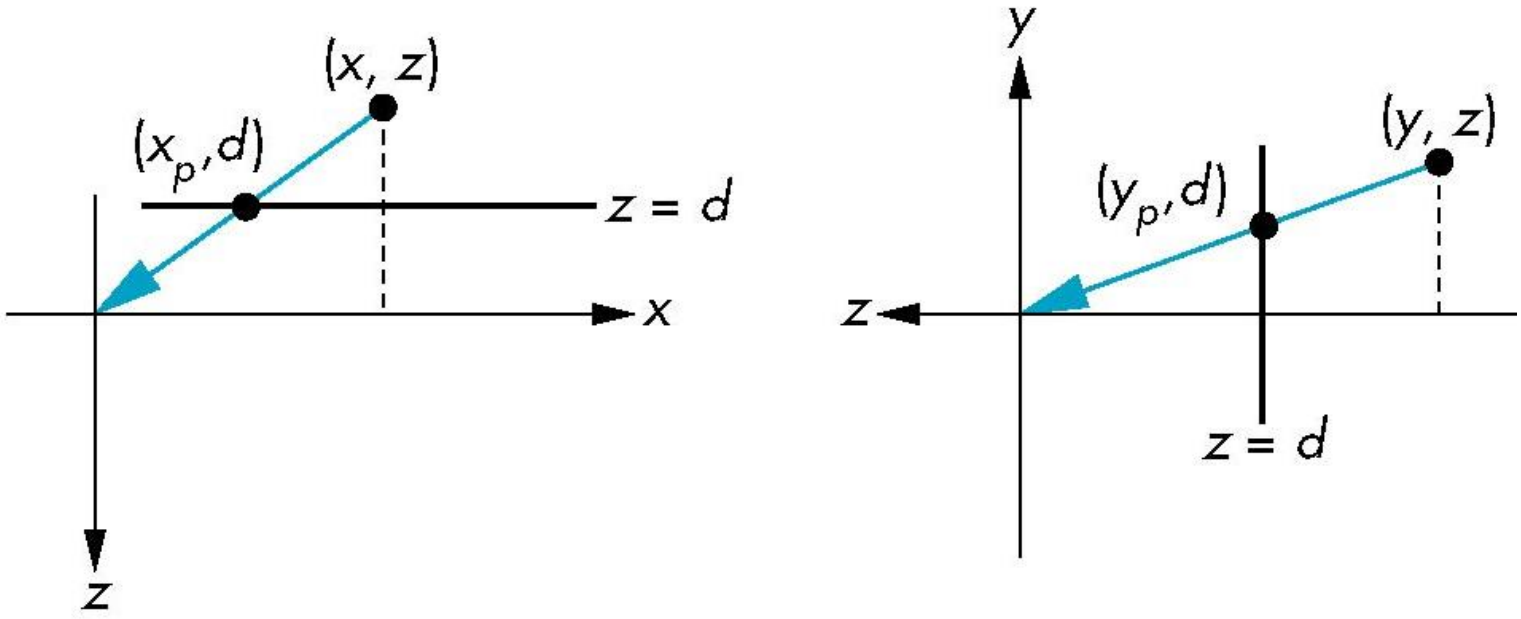
简单透视

- 只考虑两种照相机模型的第一种
- 投影中心(COP)在原点
- 投影平面为$z = d, d < 0$



透视方程

- 考虑俯视图和侧视图



$$x_p = \frac{x}{z/d}, y_p = \frac{y}{z/d}, z_p = d$$

齐次坐标形式

$$\mathbf{p} = \mathbf{M}\,\mathbf{q}, \text{ 这里 } \quad \mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix}$$

$$\mathbf{q} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad \Rightarrow \quad \mathbf{p} = \begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix}$$

透视除法

- 由于$w \neq 1$, 那么必须从四维齐次坐标中除以w来得到所表示的三维点
- 这就是透视除法(perspective division)，结果为

$$x_p = \frac{x}{z/d}, \quad y_p = \frac{y}{z/d}, \quad z_p = d$$

上述方程即为透视方程

- 后面会用OpenGL函数设置相应的裁剪体

透视变换

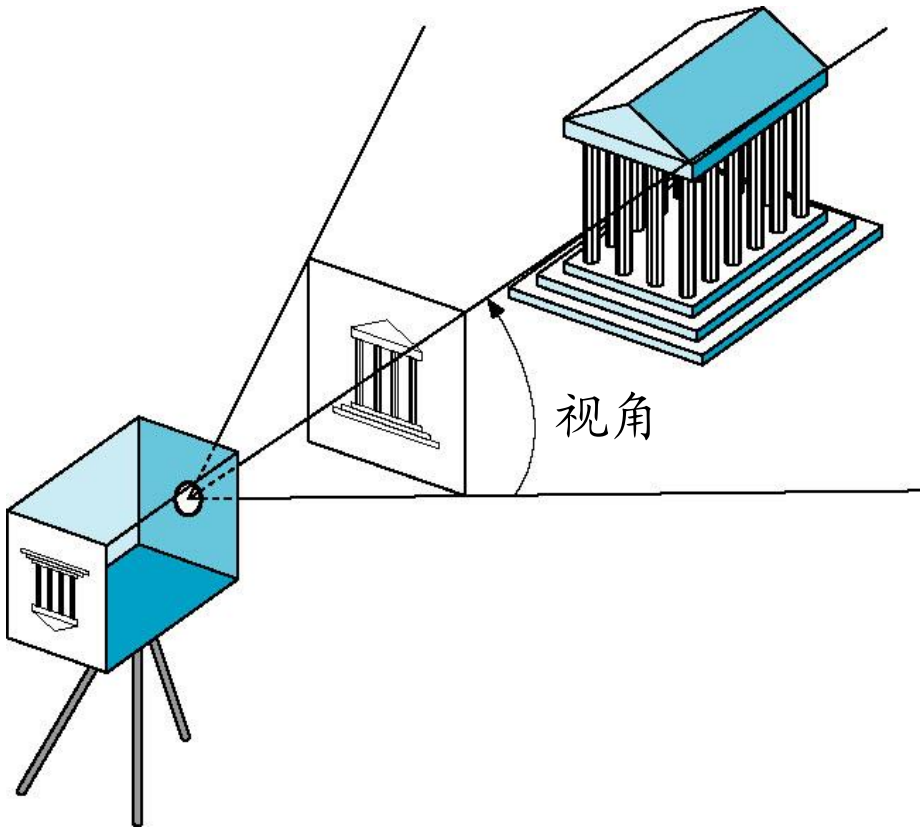
- 透视方程是非线性的，导致非均匀透视缩短(nonuniform foreshortening)
  - 离COP远的对象投影后尺寸缩短得比离COP近的对象大
- 透视变换是保直线的，但不是仿射变换
- 透视变换是不可逆的，因为沿一条投影直线上的所有点投影后的结果相同
  - 无法从投影点恢复原来的点

投影流水线

- 在模型－视图矩阵后应用4x4的投影矩阵可以实现简单的透视投影，但在最后需要进行透视除法
- 透视除法可以成为流水线的一部分

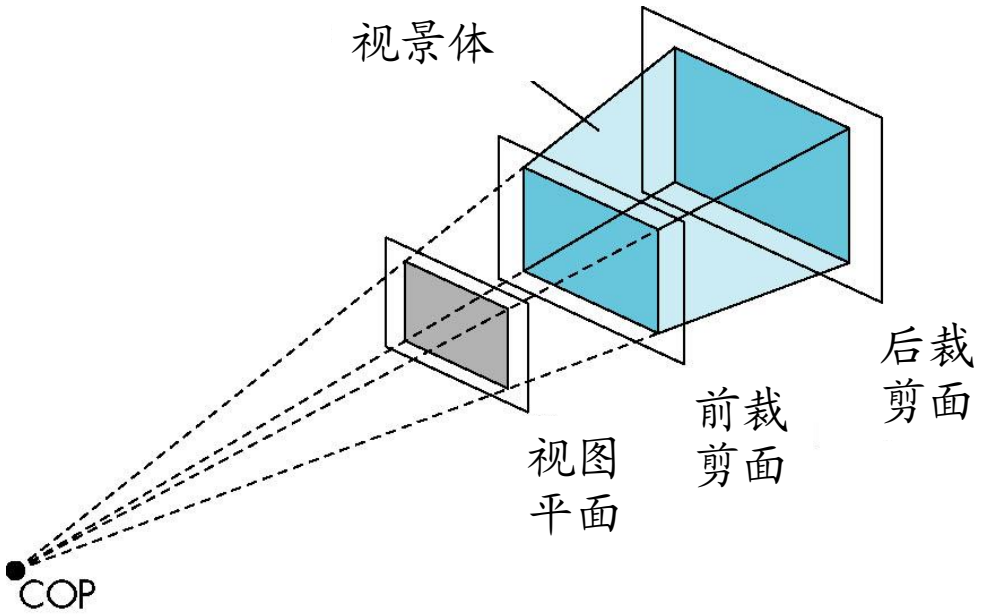
视角

- 只有位于视角(angle of view)范围内的对象才会出现在照片中



视景体

- 若底片是矩形的，那么由视角张成一个半无穷的棱椎体，称为视景体(view volume)
  - 视景体之外的对象从场景中裁剪掉
  - 顶点位于COP



棱台

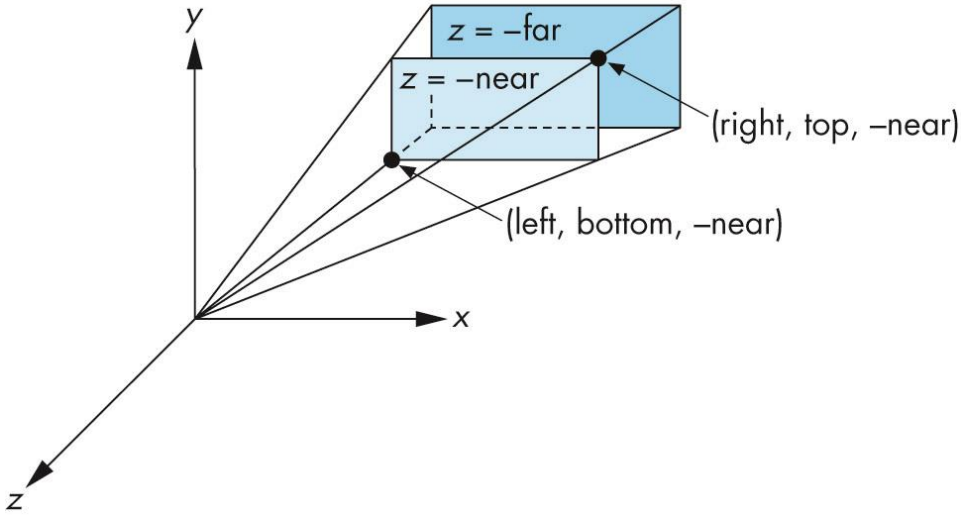
- 在大多数图形API中，指定投影函数的同时也指定裁剪参数
- 如果除了视角以外，再增加前后裁剪面，无穷棱锥变成一个棱台(frustum)
  - 有一个参数是固定的，即COP在原点
- 原则上应该可以任意指定棱台的6个面，但这样很难确定视图，而且很少需要这样的灵活性

OpenGL中的投影

- OpenGL提供了两个指定透视投影的函数和一个指定平行投影的函数
  - 参数用视点坐标给出
- 还可以通过加载4x4矩阵，或者通过对初始单位阵应用旋转、平移和缩放变换来直接生成投影变换矩阵

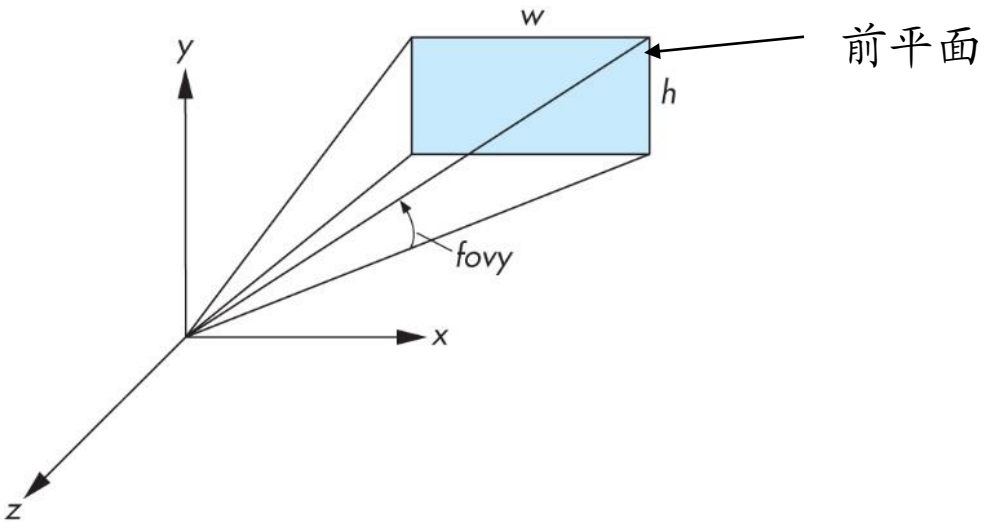
OpenGL的透视

- void **glFrustum**(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far);
  - near和far是从照相机（视点）到近裁剪面和远裁剪面的距离，必须是正值
  - left、right、top和bottom在近（前）裁剪面内指定
  - 视景体可以不对称



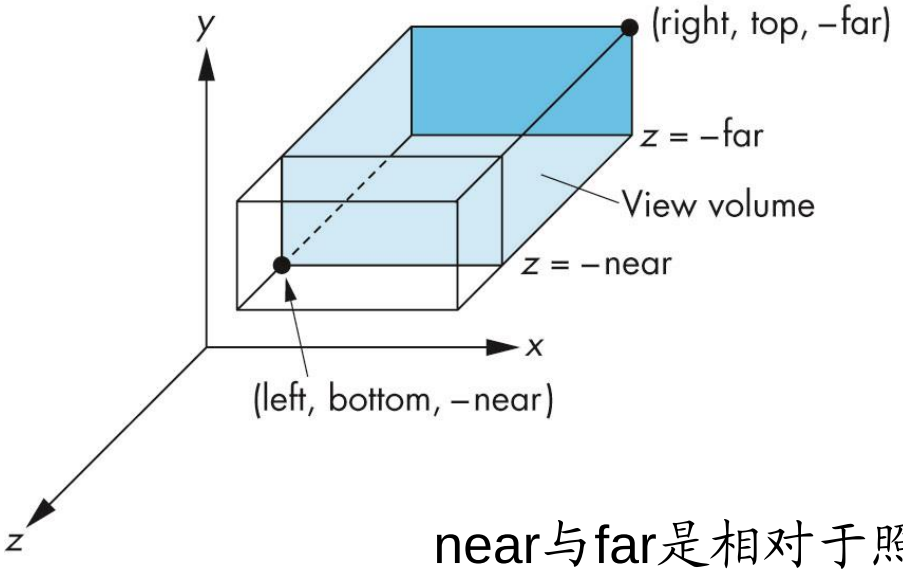
视角的应用

- 应用glFrustum有时很难得到所期望的结果
- void **gluPerspective**(GLdouble fovy, GLdouble aspect, GLdouble near, GLdouble far);
  - fovy：y方向上的视角，[0.0,180.0]
  - aspect：宽高比w/h
  - 视景体是关于z轴对称的四棱台



OpenGL的正交视图

- 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

6.1 经典视图

6.2 计算机视图

6.3 投影矩阵

6.3 投影矩阵

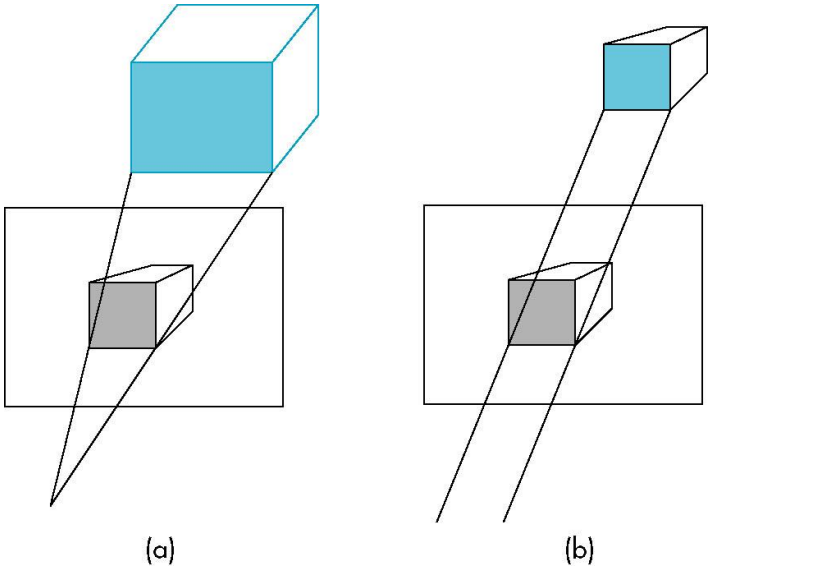
- 投影规范化
- 平行投影规范化变换矩阵
- 透视投影规范化变换矩阵

投影规范化

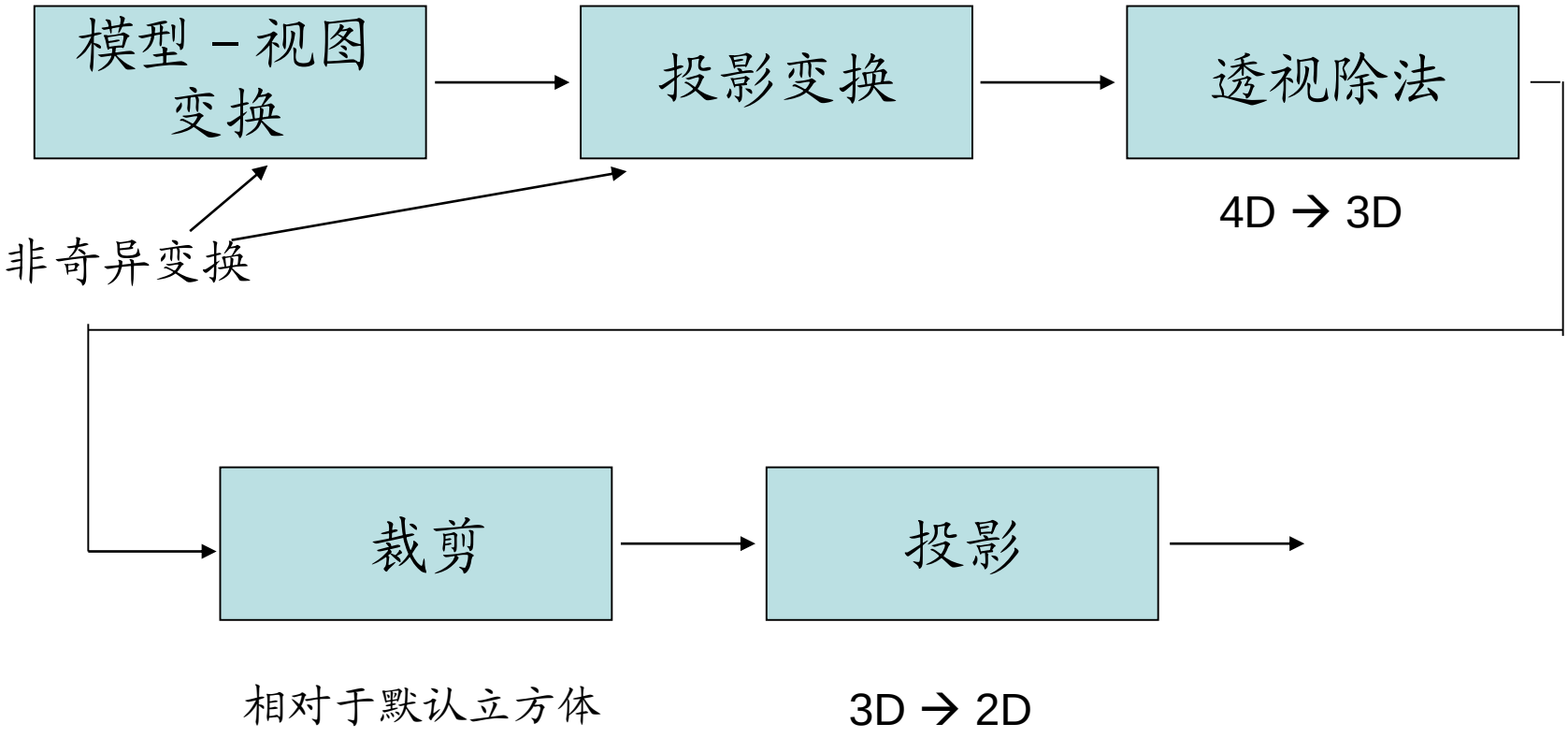
- 不想为每种类型的投影设计不同的投影矩阵，所以把所有的投影转化为具有默认视景体的正交投影
- 这种策略可以使我们在流水线中应用标准变换，并进行有效的裁剪

投影规范化

- 把对象进行变形，使得变形后的对象经简单正交投影后得到与原对象的理想投影一样的视图
- 扭曲变形由规范化矩阵来执行
  - 设计规范化矩阵，使视景体变形为规范视景体
  - OpenGL中，投影规范化由投影变换矩阵实现



流水线



注释

- 在模型－视图变换和投影变换的过程中，我们是一直在四维齐次坐标系中的
  - 这些变换都是非奇异的
  - 默认值为单位阵（正交视图）
- 规范化使得不管投影的类型是什么，都是相对于默认的简单立方体进行裁剪
- 投影直到最后时刻才进行
  - 从而可以尽可能的保留深度信息，这对隐藏面消除是非常重要的

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);
```

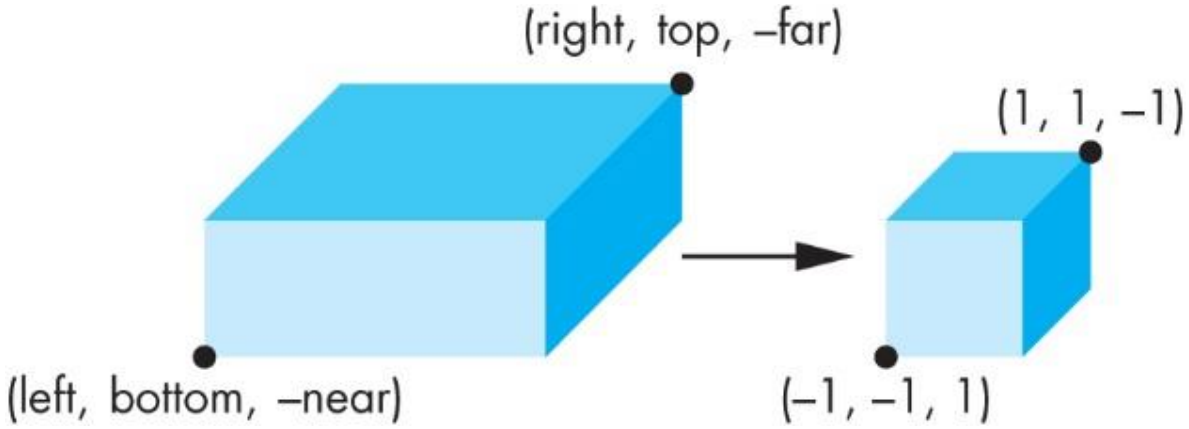
称这个视景体为规范视景体（canonical view volume）

- 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)



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

- 分两步

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

– 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))

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

最后的投影

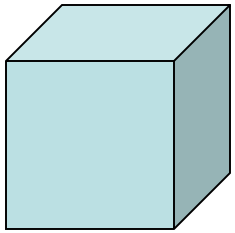
- 令z = 0
- 这等价于如下的齐次坐标变换

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

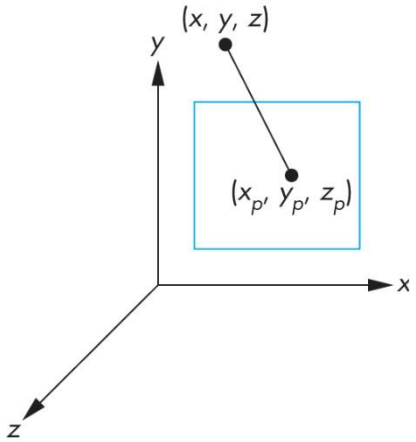
- 从而在4D中一般的正交投影为$\mathbf{P} = \mathbf{M}_{\text{orth}}\mathbf{ST}$

斜投影

- OpenGL的投影函数不支持一般的平行投影，例如立方体的如下图示

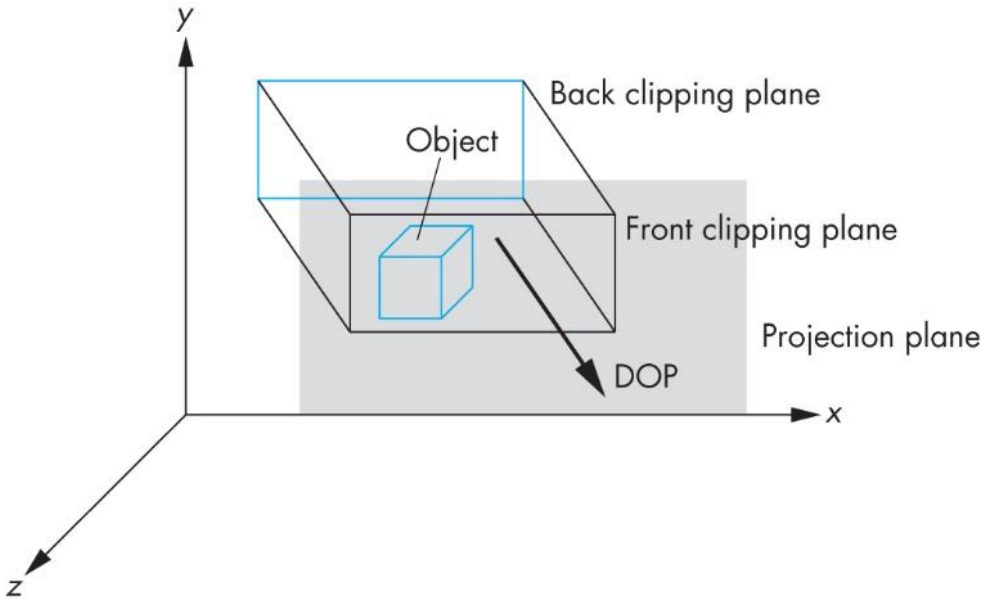


- 此时立方体好像发生了错切，然后再进行正交投影
- 斜投影 = 错切 + 正交投影
  - 斜投影由投影线和投影平面的夹角来表征



斜投影的视景体

- 斜投影的视景体由6个面确定
  - 远、近裁剪面平行于投影平面
  - 其他面平行于投影方向

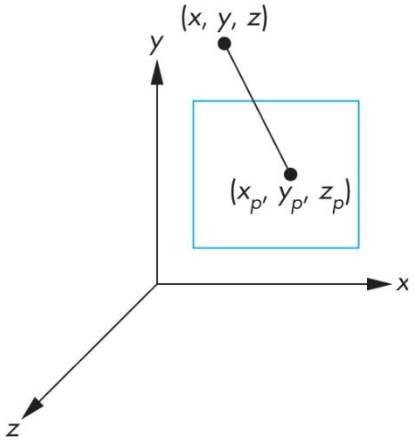


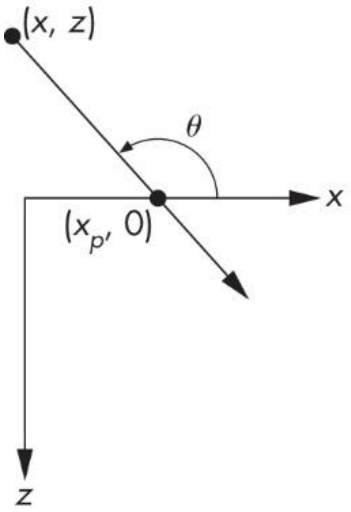
俯视图和侧视图

- 投影平面z=0

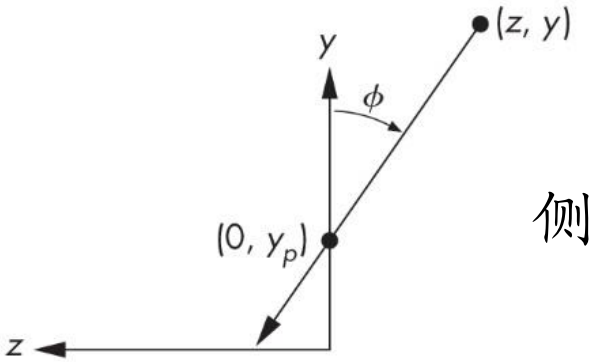
$$x_p = x + z \cot \theta$$

$$y_p = y + z \cot \phi$$





俯视图



侧视图

(a) (b)

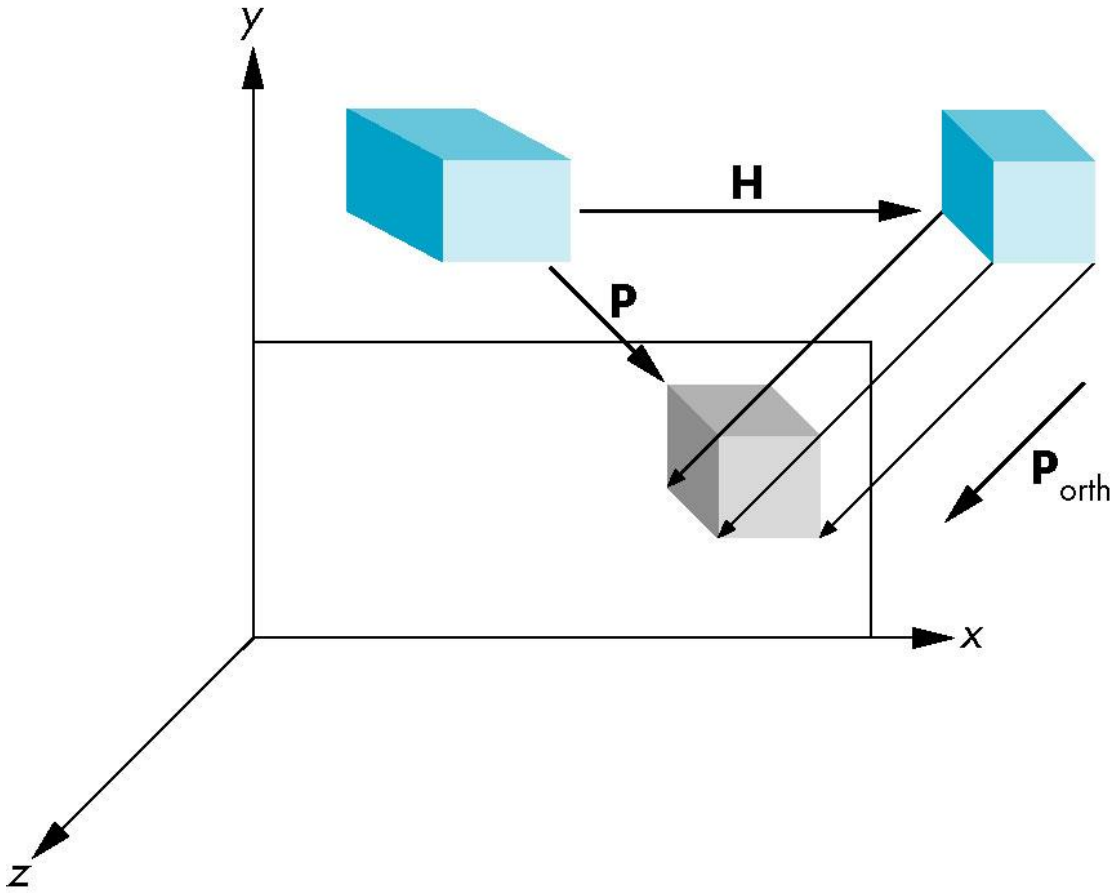
错切矩阵

- *xy* 错切（*z*值不变）

$$\mathbf{H}(\theta,\phi) = \begin{bmatrix} 1 & 0 & \cot\theta & 0 \\ 0 & 1 & \cot\phi & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

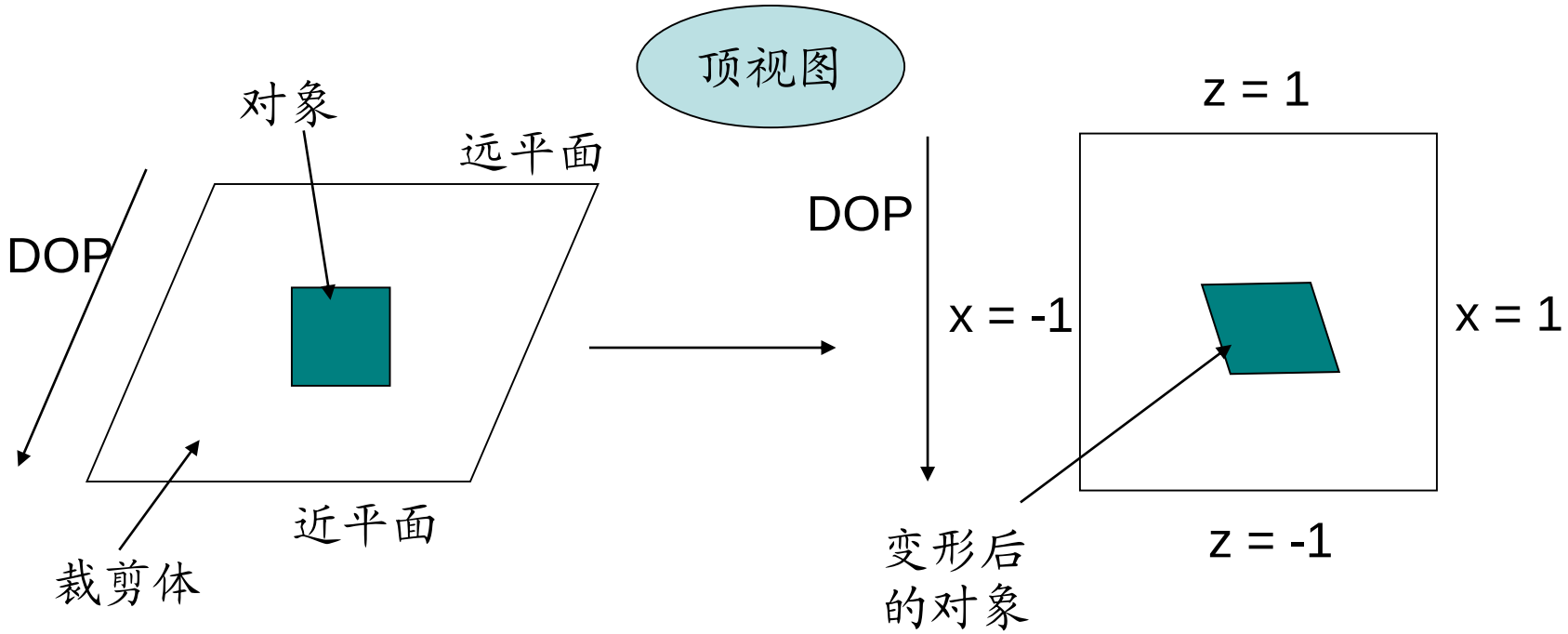
- 投影矩阵$\mathbf{P} = \mathbf{M}_{\mathbf{orth}}\,\mathbf{H}(\theta, \phi)$
- 一般情形：　$\mathbf{P} = \mathbf{M}_{\mathbf{orth}}\,\mathbf{S}\mathbf{T}\,\mathbf{H}(\theta, \phi)$

等价性



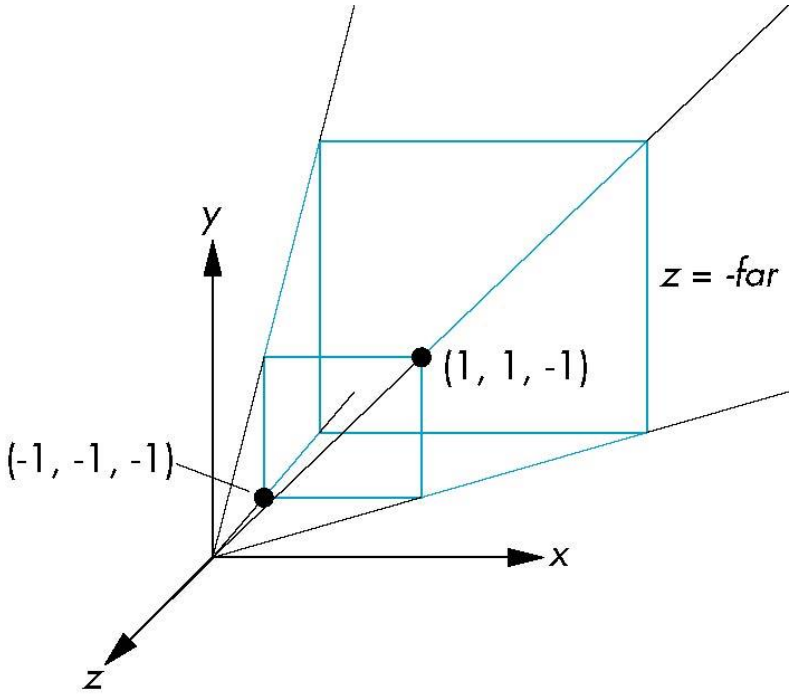
对裁剪体的影响

- 投影矩阵**P = STH**把原来的裁剪体变换为默认的裁剪体



简单透视

- 考虑简单透视：COP在原点，投影面在z = -1，由平面 x = ±z, y = ±z确定的有90度的视角



简单透视矩阵

- 齐次坐标下的简单透视投影矩阵
  - 投影平面z = d = -1

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

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix}$$

  - **M**为奇异矩阵
  - 注意这个矩阵和远裁剪面无关

推广

- 透视规范化矩阵

$$\mathbf{N}=\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & \alpha & \beta\\ 0 & 0 & -1 & 0\end{bmatrix}$$

$$(x, y, z, 1) \rightarrow (x, y, \alpha z+\beta, -z)$$

- 在透视除法后，点(x,y,z,1)变成

$$x' = -x/z,\ y' = -y/z,\ z' = -(\alpha+\beta/z)$$

- 无论α, β的值是什么，在正交投影后就得到所期望的点。此时，矩阵**N**非奇异

α与β的选取

如果取 $z' = -(\alpha+\beta/z)$

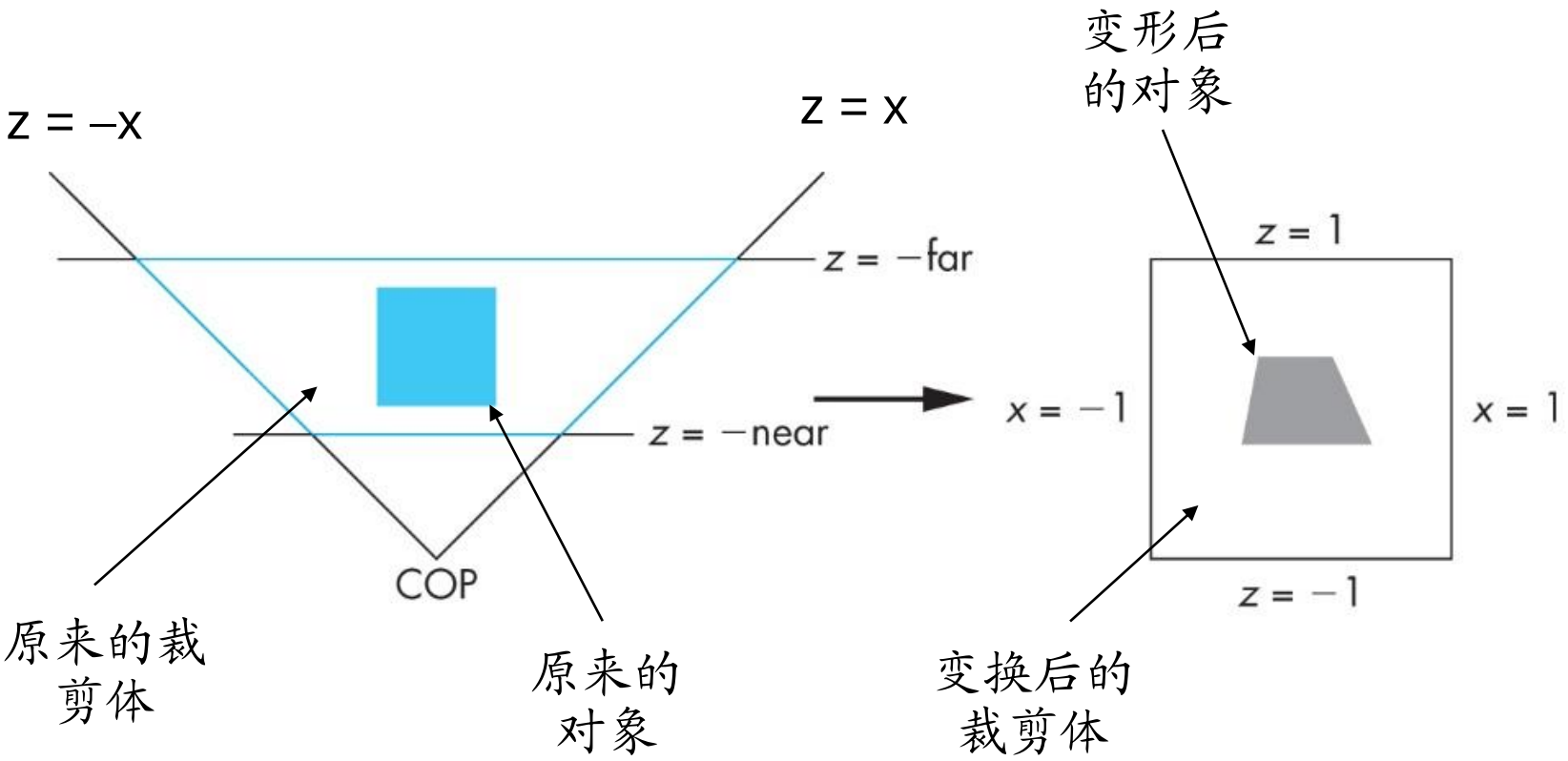
$$\alpha = (near + far) / (near - far)$$

$$\beta = 2\ near * far / (near - far)$$

那么

- 近平面z = -near映射到z = -1
- 远平面z = -far映射到z = 1
- 各侧边映射到 x = ±1, y = ±1
- 新的裁剪体就是缺省裁剪体

透视规范化变换

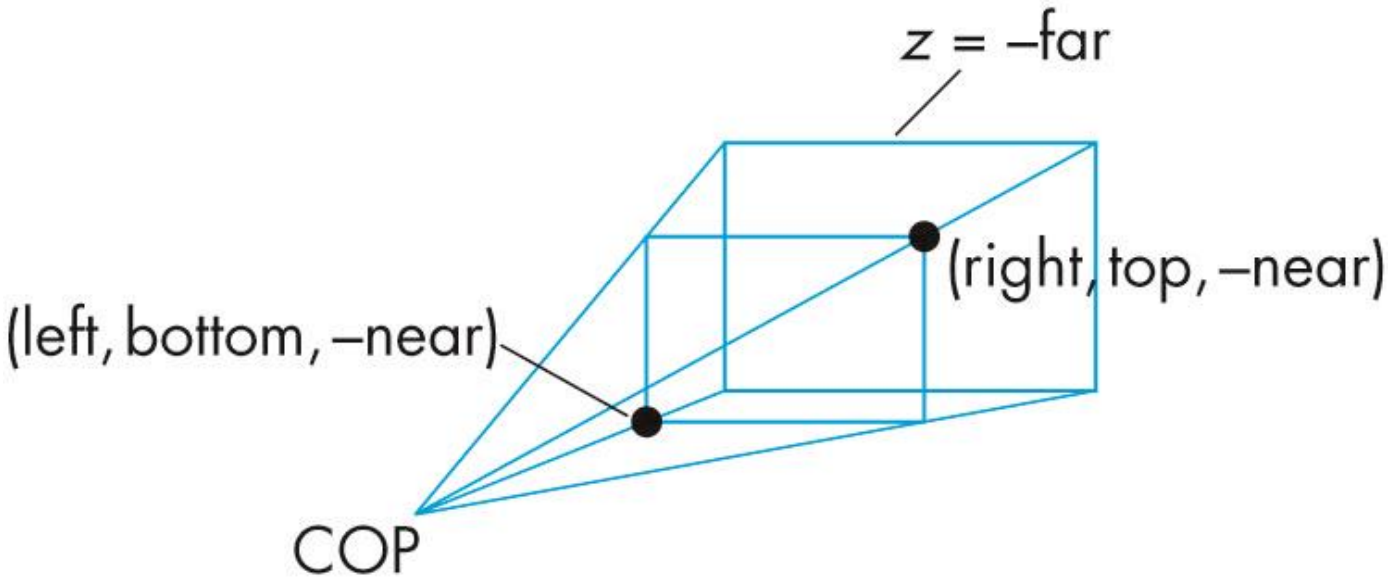


规范化与隐藏面消除

- 虽然这里选择的透视矩阵形式上看起来有点任意，但这种选择保证如果在原来的裁剪体内$z_1>z_2$, 那么变换后的点满足$z_1'>z_2'$
- 因此如果首先应用规范化变换，隐藏面消除算法仍有效
- 然而，公式$z' = -(\alpha+\beta/z)$意味着规范化引起距离发生了改变，这可能导致数值问题，特别是当近距离非常小的时候

OpenGL的透视变换

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



OpenGL透视矩阵

- glFrustum的规范化
  - 错切变换**H**：把非对称的棱台视景变换为对称的正棱台，远近平面不变
  - 缩放变换**S**：把四棱台的侧面变换为x= ±z和y= ±z，远近平面不变
  - 透视规范化变换**N**： x = ±z变换为x = ±1， y = ±z变换为y = ±1，近平面z = -near变换为z = -1，远平面z = -far变换为z = 1

$$\mathbf{P = NSH}$$

透视投影矩阵

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

gluPersective投影矩阵

- 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

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

为何规范化？

- 规范化使得只需要一个流水线体系就可以进行透视投影和正交投影
- 尽可能位于四维齐次空间中，以便保持隐藏面消除和明暗处理所需要的三维信息
- 简化了裁剪的操作