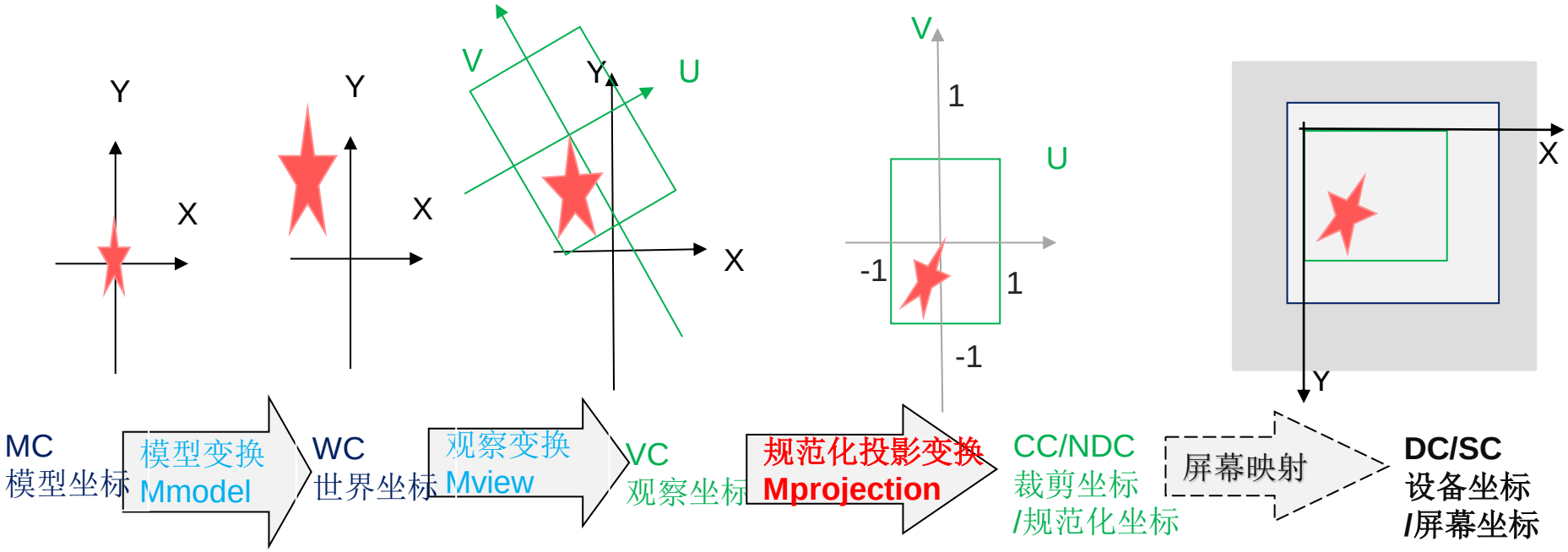
Recap

- Every “change of coordinates” is equivalent to “a matrix transformation” 坐标变换等价于一个矩阵变换



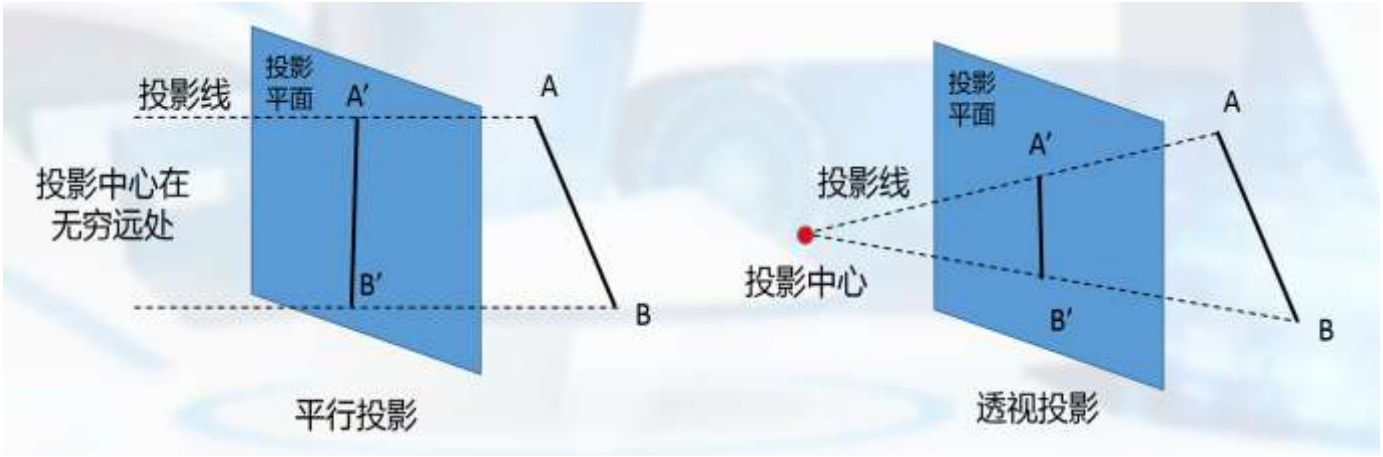
Outlines

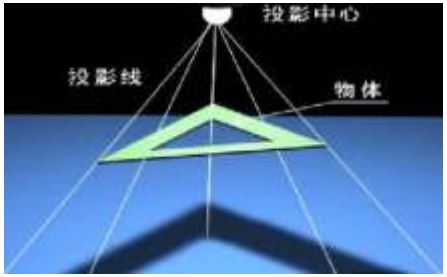
Computer Viewing Process 观察过程中的变换

- Model Transformation模型变换
- View Transformation观察变换
- **Projection Transformation投影变换**
  - Classical Projection 经典投影
    - Parallel Projection平行投影
    - Perspective Projection透视投影
  - **Computer Projection 计算机投影***
    - Normalization Transformation规范化变换
    - Visual Volume Transformation视见体变换
    - Normalized Projection规范化投影变换
- **Window-Viewport Trans. 窗区视区变换/屏幕映射**

Classical Projection经典投影

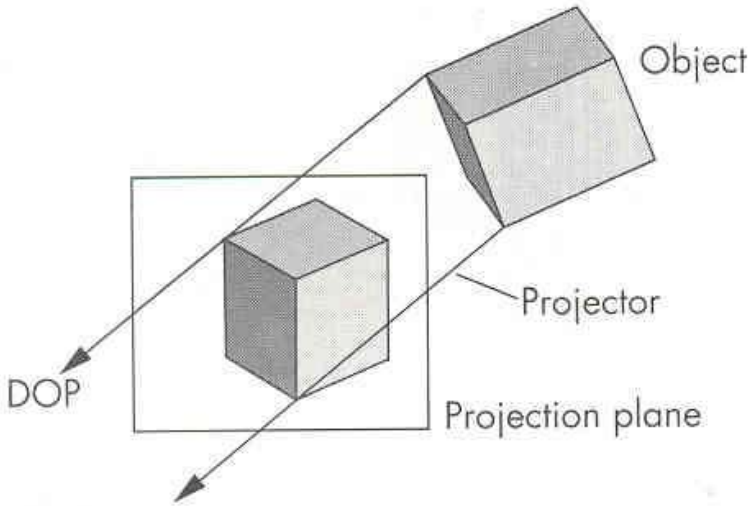
- 经典平面投影定义
  - 投影指的是用一组光线将物体的形状投射到一个平面上去，称为“投影”。 在该平面上得到的图像，也称为“投影”
  - **投影三要素(Three Elements)：投影中心COP，投影线Porjector，投影平面Projection plane**

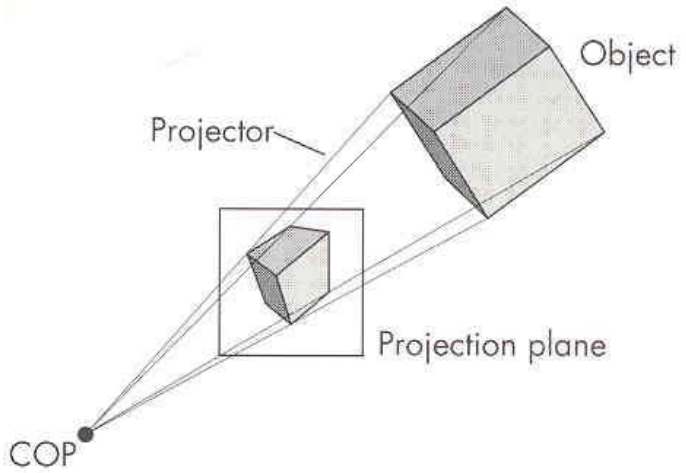




Classical Projection(cont.)

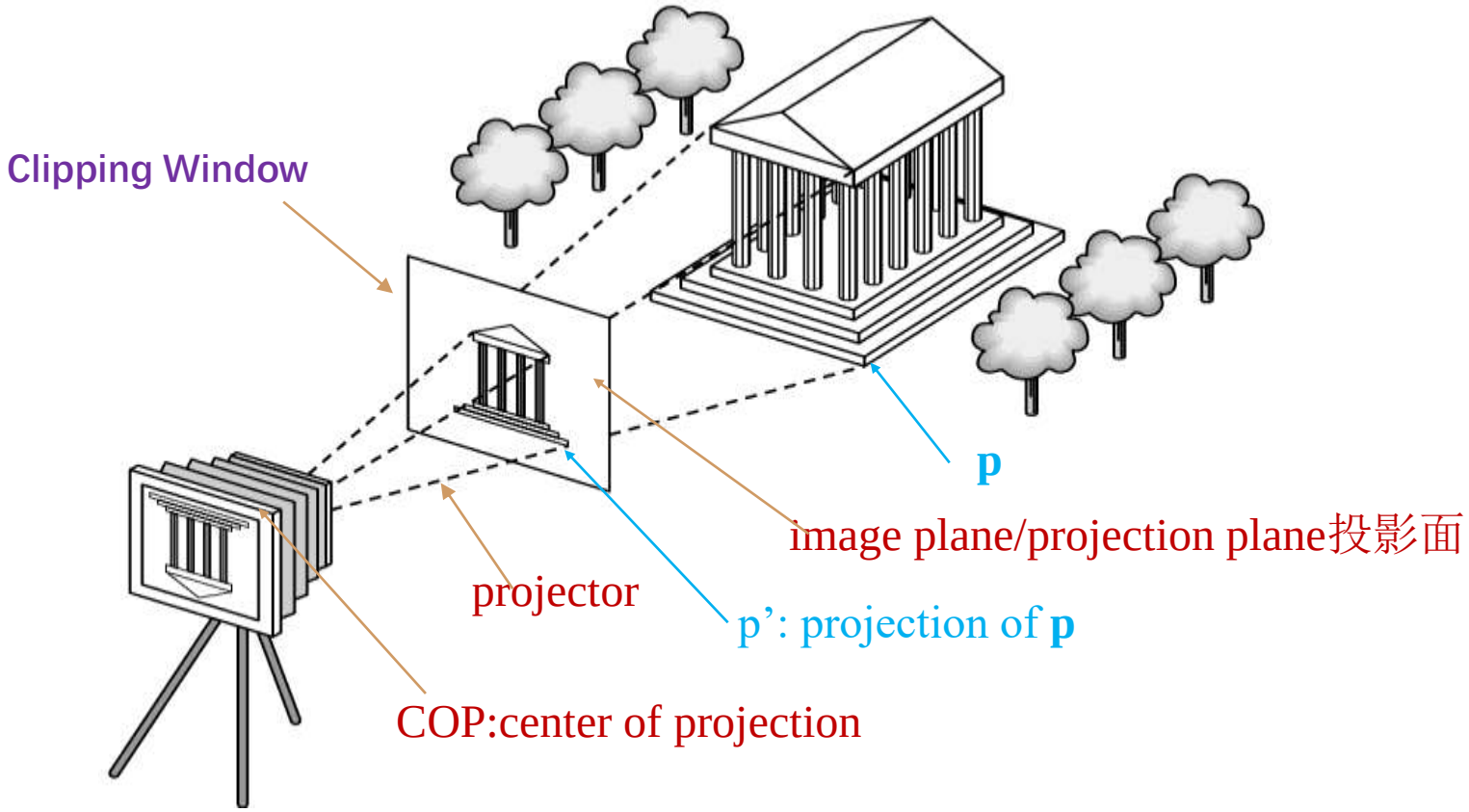
- 经典平面投影的两大类型
  - **看“投影中心”到“投影面”之间的距离是否有限，或者看投影线汇聚于投影中心点**
    - 距离有限的->**透视投影**
    - 距离无限的->**平行投影**
    - 平行投影可看作是透视投影的特殊情况





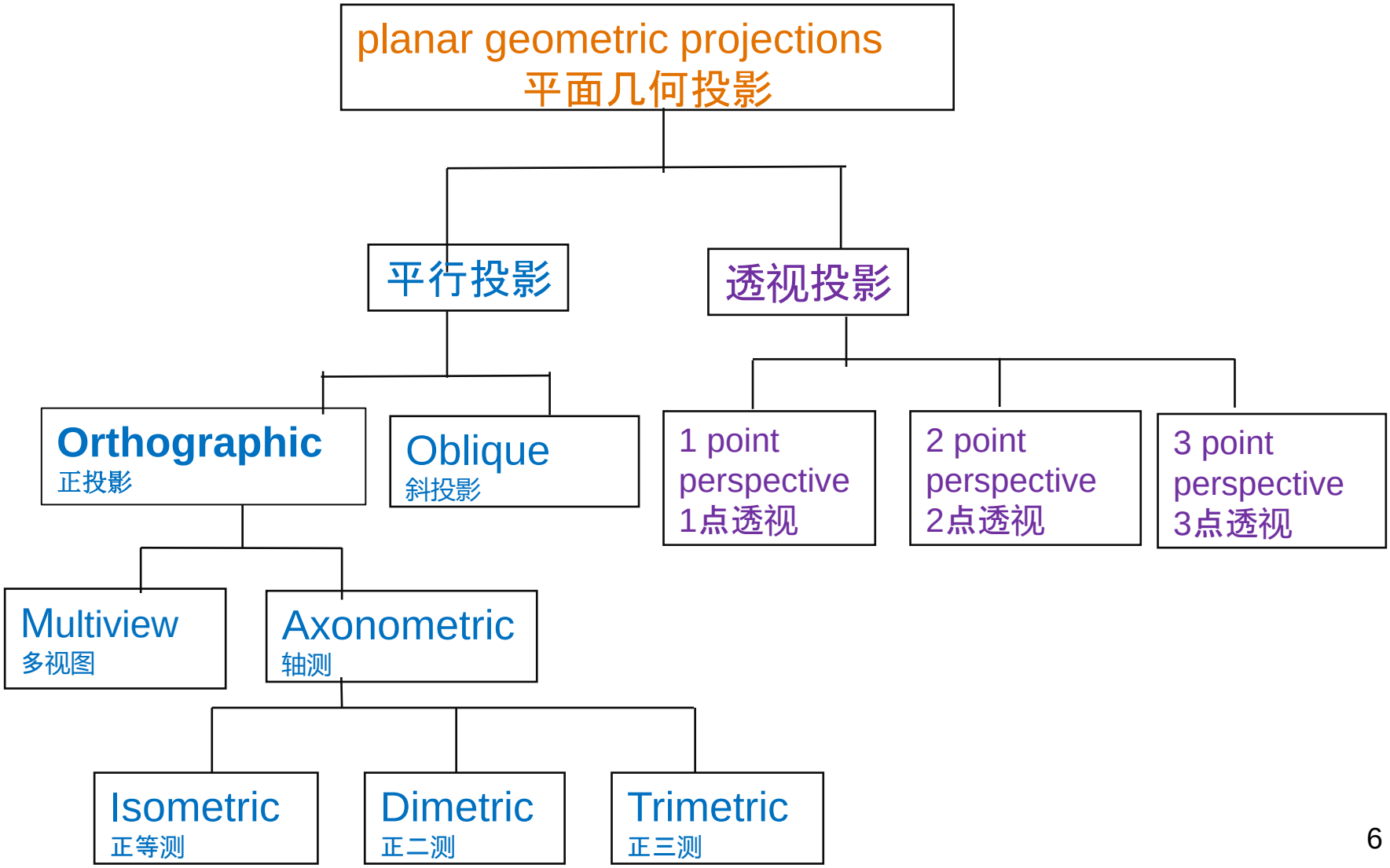
Classical Projection(cont.)

- 计算机虚拟相机模型就是透视的效果



Classical Projection(cont.)

- planar geometric projections平面几何投影分类图



Outlines

Computer Viewing Process 计算机观察流程

- Model Transformation模型变换
- View Transformation观察变换
- **Projection Transformation投影变换**
  - Classical Projection 经典投影
    - **Parallel Projection平行投影**
    - Perspective Projection透视投影
  - **Computer Projection 计算机投影***
    - Normalization Transformation规范化变换
    - Visual Volume Transformation视见体变换
    - Normalized Projection规范化投影变换
- **Window-Viewport Trans. 窗区视区变换/屏幕映射**

Classical Projection(cont.)

1.Parallel Projection平行投影

- 平行线保持：任何平行线投影后还是平行线，能**如实反映物体的尺寸和形状**
- 真实感差： 不是人眼成像的效果，所以真实感差

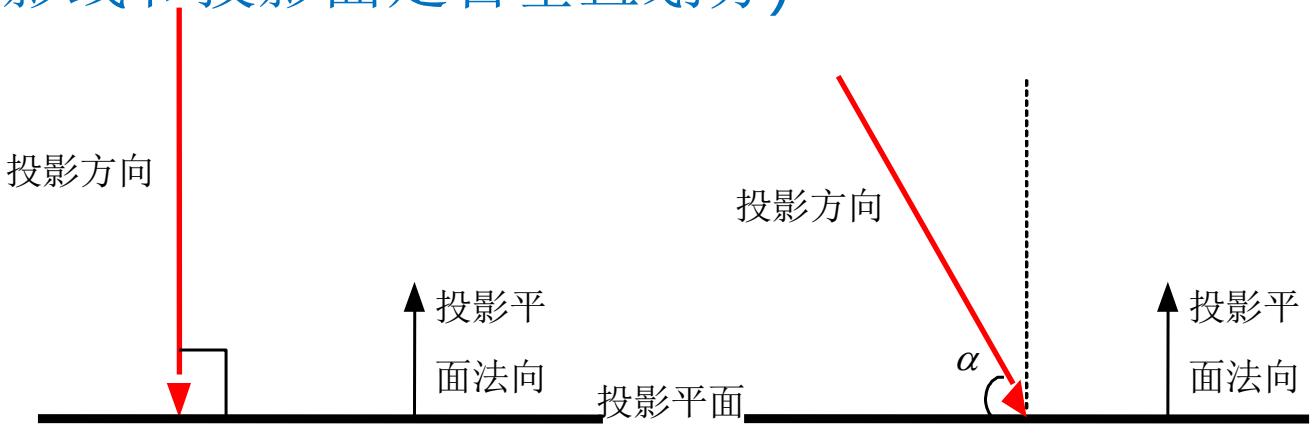
（中国）卷轴图

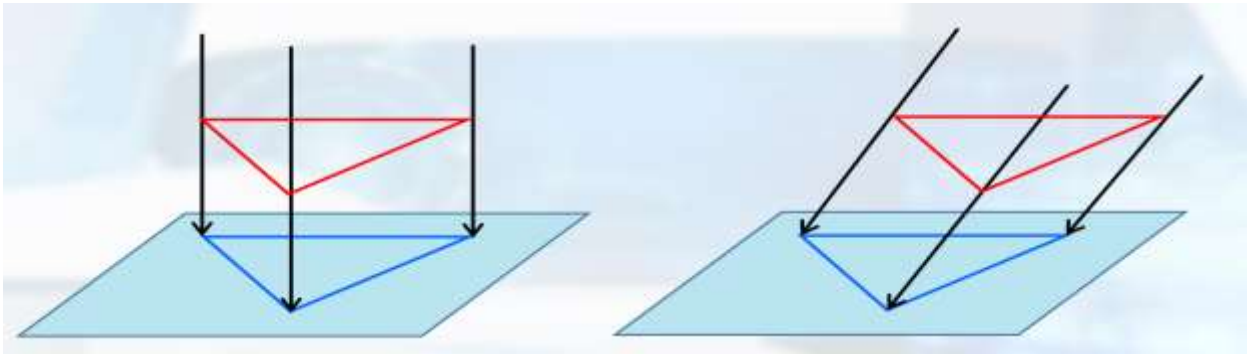
Classical Projection(cont.)

1.Parallel Projection平行投影(cont.)

平行投影可分成两类：正投影和斜投影

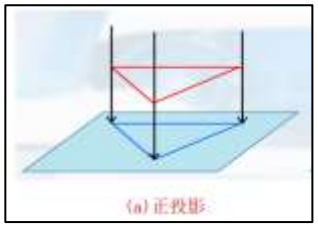
(按投影线和投影面是否垂直划分)





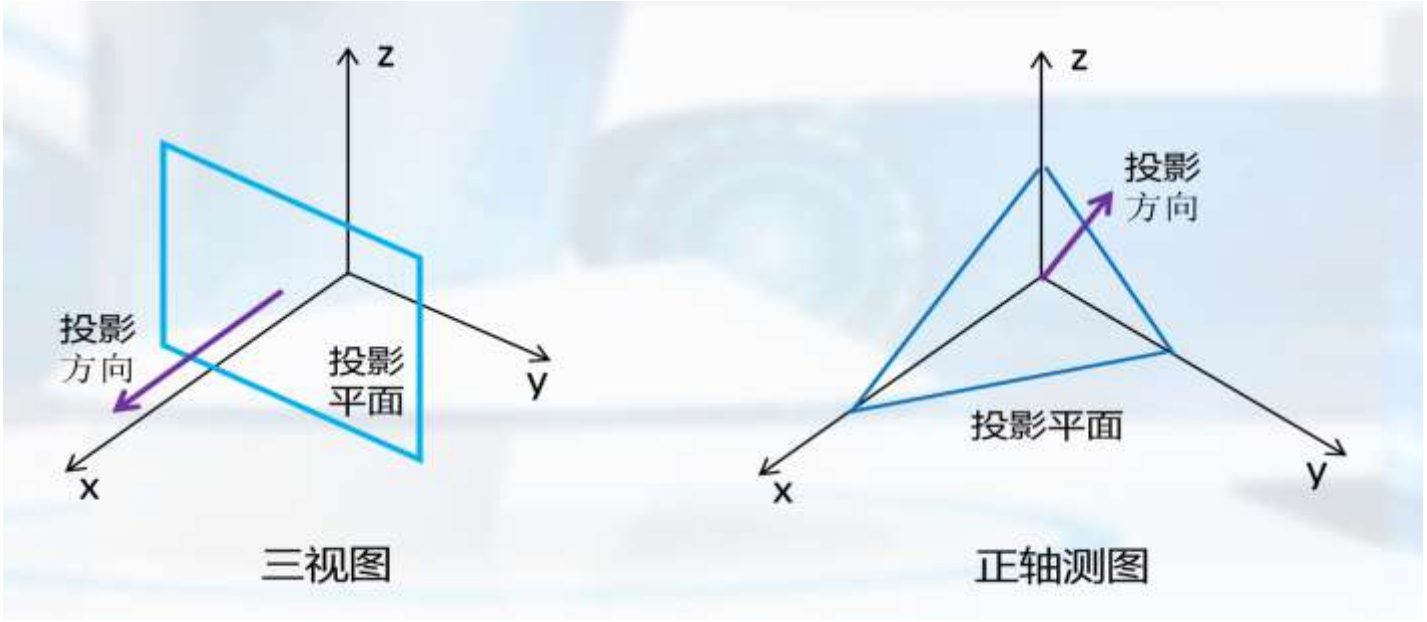
(a) 正投影　　(b) 斜投影

Classical Projection(cont.)



1.Parallel Projection平行投影(cont.)

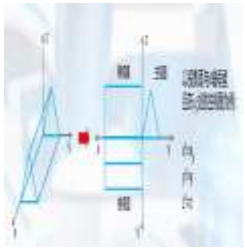
- **平行投影之”正投影”(orthographic)**
  - “正投影”又分为三视图和正轴测图: **根据投影面是否平行于某“主面”**
    - ◆主面：**与三维空间的坐标轴垂直的面:** XOY,YOZ,ZOX面
    - ◆主轴：三维空间的三个坐标轴方向： **X,Y,Z轴**

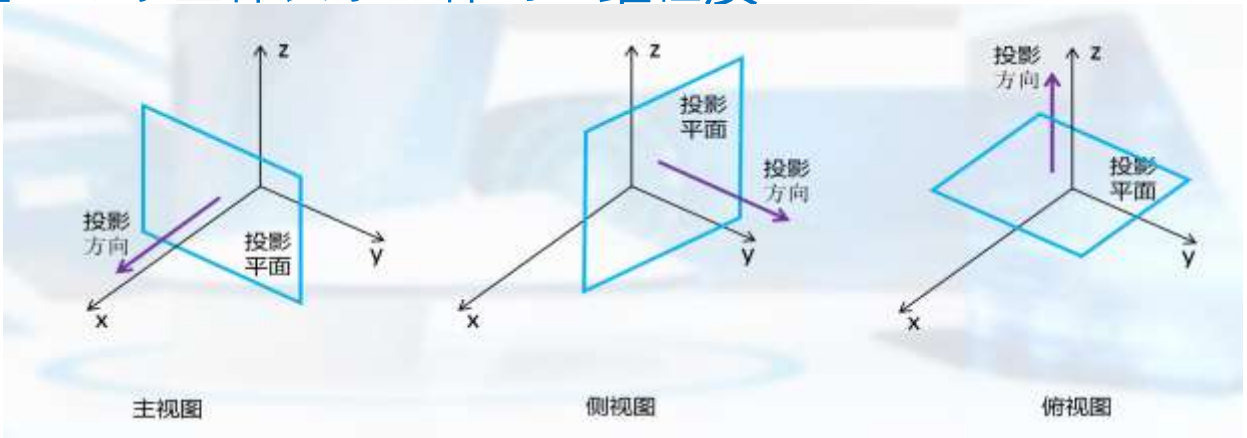


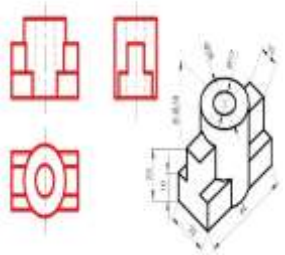
Classical Projection(cont.)

1.Parallel Projection平行投影(cont.)

- **平行投影之”正投影”(orthographic)**
  - **三视图：投影平面平行于某主面**
  - ◆优点：投影后不改变距离和角度,反映形体实际尺寸,适于施工图纸用
  - ◆缺点：难以形象立体表示形体的三维性质







$$T_{yoz}=\begin{bmatrix}0&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}$$

$$T_{zox}=\begin{bmatrix}1&0&0&0\\0&0&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}$$

$$T_{xoy}=\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&0\\0&0&0&1\end{bmatrix}$$

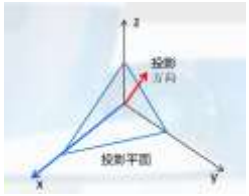
Classical Projection(cont.)

1.Parallel Projection平行投影(cont.)

- **平行投影之正投影**
  - **正**轴测：**投影面是不平行于某“主面”**

优点：有三维效果，轴测投影图广泛地应用于建筑设计和机械设计等领域。

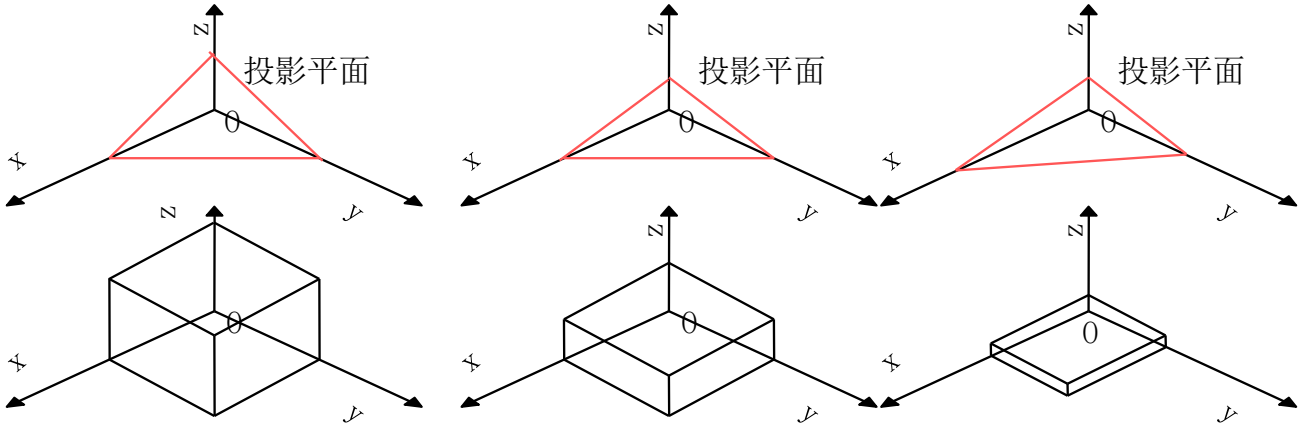
缺点：投影后角度关系可能会改变（如圆投影后变成椭圆）。




Isometric

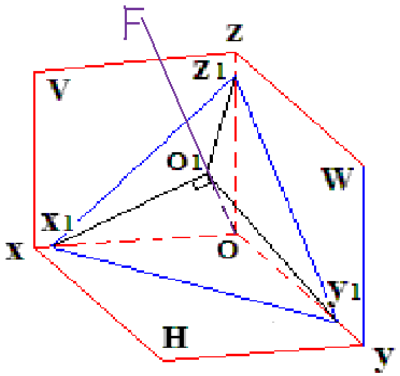

Dimetric

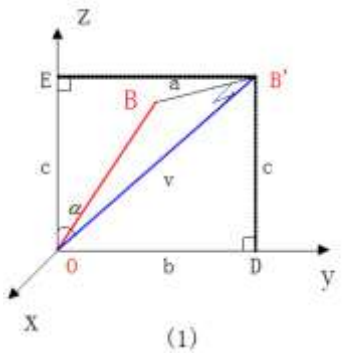

Trimetric



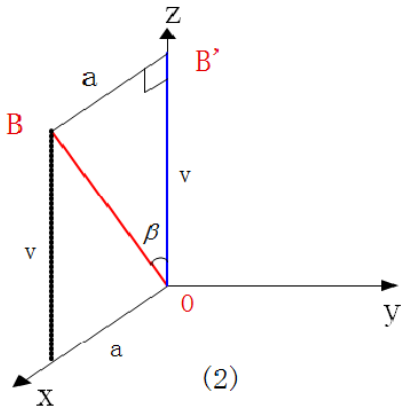
(a) 等轴测　(b) 正二测　(c) 正三测

正轴测的变换矩阵推导





(1)



(2)

思路: 将投影面的法向量F旋转到Z轴 ,即将投影面变换到XOY平面，再进行投影。

需要的串联变换为：

(1)先进行平移，将O1点移动到与O点重合,单位化为OB-图（1）中红线。Tt

(2)绕y轴顺时针旋转α角，使OBB'转动到yoz面上, 见图(1)。Try

(3)绕x轴逆时针旋转β角，使OB转动到OZ轴上, 见图(2)。Trx

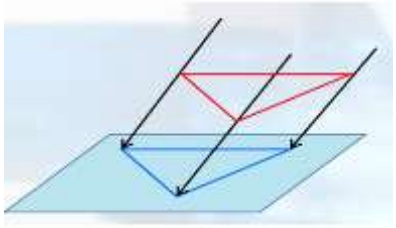
(4)将三维形体向xoy平面作正投影 Tp

$$Tt=\begin{bmatrix}1&0&0&-r\\0&1&0&-s\\0&0&1&-t\\0&0&0&1\end{bmatrix}\quad Try=\begin{bmatrix}\cos(-\alpha)&0&\sin(-\alpha)&0\\0&1&0&0\\-\sin(-\alpha)&0&\cos(-\alpha)&0\\0&0&0&1\end{bmatrix}\quad Trx=\begin{bmatrix}1&0&0&0\\0&\cos\beta&-\sin\beta&0\\0&\sin\beta&\cos\beta&0\\0&0&0&1\end{bmatrix}\quad Tp=\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&0\\0&0&0&1\end{bmatrix}$$

最后得到轴测图的投影变换矩阵为：

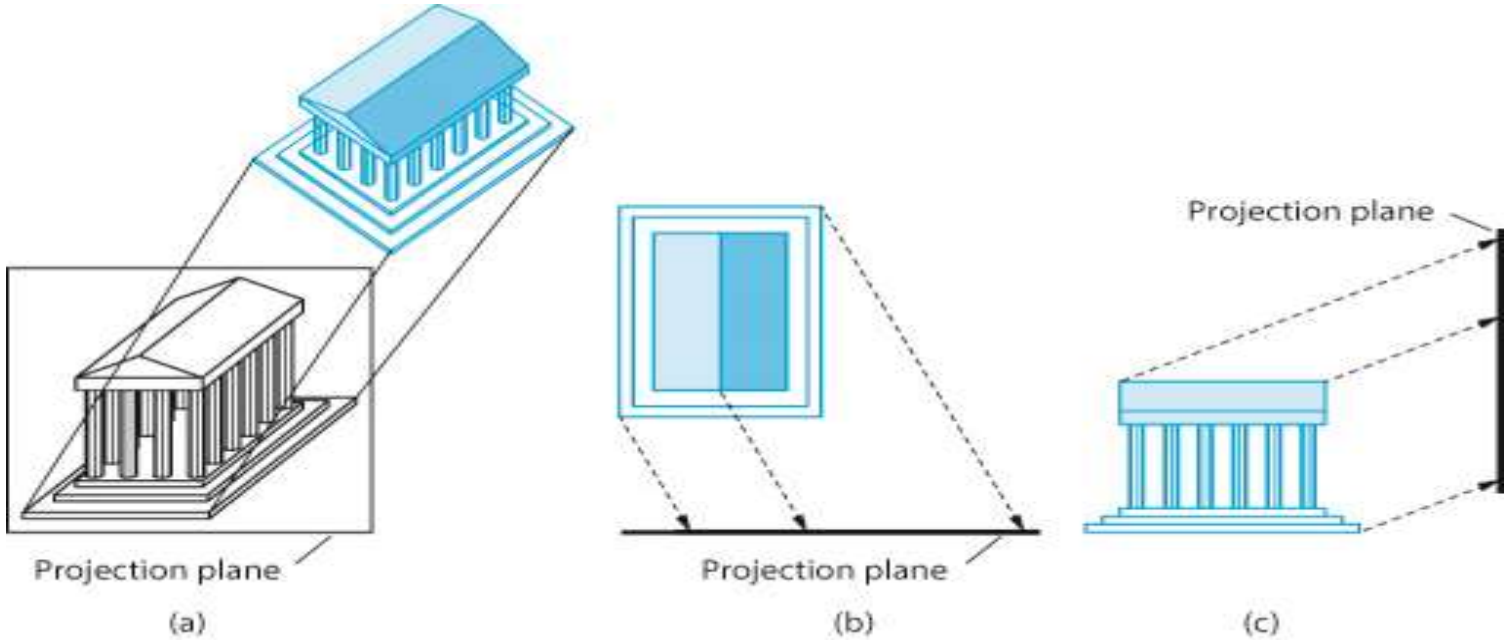
$$T=Tp\bullet Trx\bullet Try\bullet Tt$$

Classical Projection(cont.)



1.Parallel Projection平行投影(cont.)

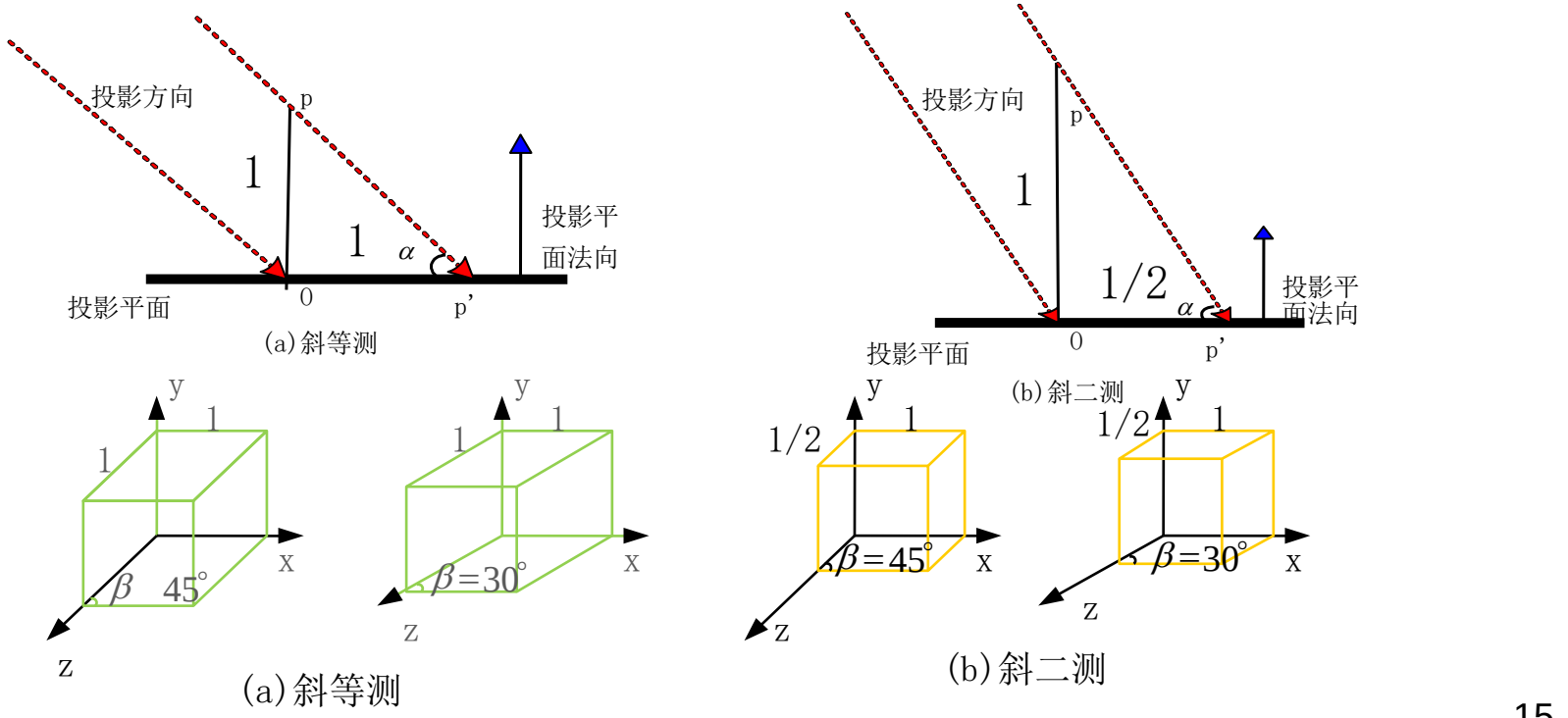
- 平行投影之斜投影Oblique
  - 投影方向不垂直于投影面，但通常投影面平行于某主面。
  - 优点：既能进行形体的尺寸，角度测量，又能展示三维效果（结合了三视图和正轴测图的特点）。注：数学书上的3D图大都是斜投影。



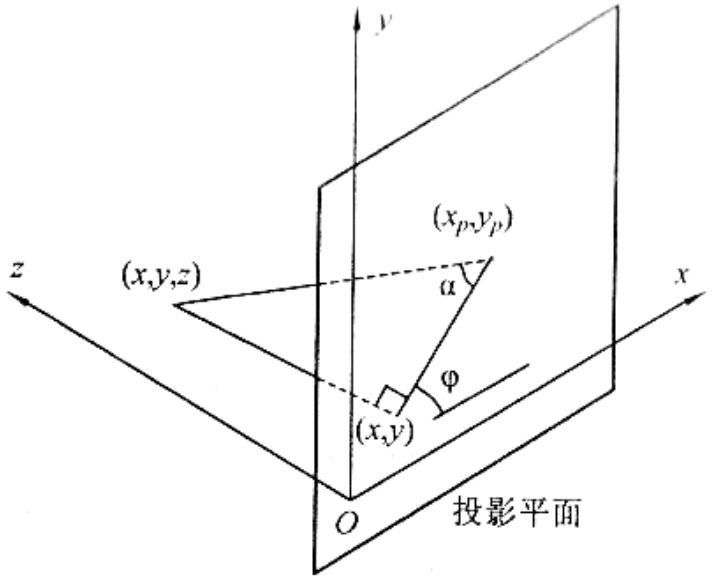
Classical Projection(cont.)

1.Parallel Projection平行投影(cont.)

- 平行投影之斜投影Oblique（cont.)
  - 斜等测 投影线和投影面夹角α=45˚=arctg(1)：投影后线段长度不变。
  - 斜二测 投影线和投影面夹角 α=arctg(2)：线段缩短1/2，但真实感强些



平行投影之斜投影变换矩阵-推导1



设L：(x,y)和(xp,yp)之间的连线长度

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

$$x_p = x + L\cos\varphi$$

$$y_p = y + L\sin\varphi$$

L1：当Z单位化时即Z=1时,L的长度

$$\tan\alpha = \frac{z}{L} = \frac{1}{L_1}$$

$$L = zL_1$$

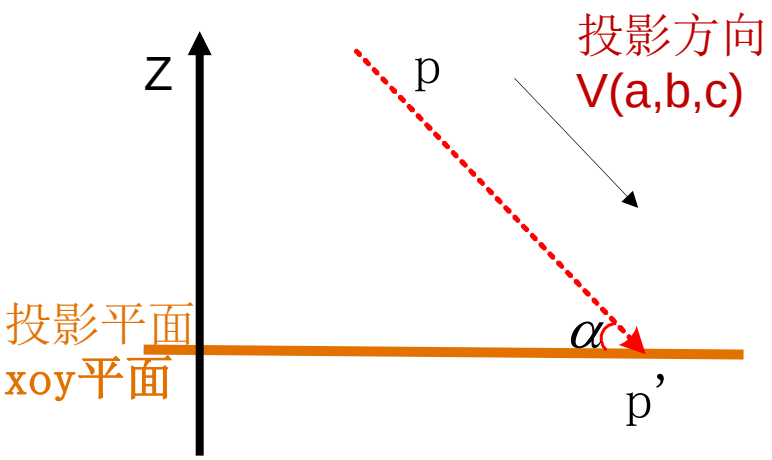
$$x_p = x + z(L_1\cos\varphi)$$

$$y_p = y + z(L_1\sin\varphi)$$

投影面是XOY平面时，
斜投影变换是图形对依赖轴Z轴的错切变换
X=X+g(z);
Y=Y+f(z);
Z=z;

$$M_{par} = \begin{bmatrix} 1 & 0 & L_1\cos\varphi & 0 \\ 0 & 1 & L_1\sin\varphi & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

平行投影之斜投影变换矩阵-推导2



$$\frac{Xp'-Xp}{a}=\frac{Yp'-Yp}{b}=\frac{Zp'-Zp}{c}=u$$

$$\begin{aligned} Xp' &= au+Xp \\ Yp' &= bu+Yp \\ Zp' &= cu+Zp \end{aligned} \Rightarrow u=(Zp'-Zp)/c \Rightarrow \begin{aligned} Xp' &= Xp+(Zp'-Zp)\cdot a/c \\ Yp' &= Yp+(Zp'-Zp)\cdot b/c \\ Zp' &= Zp' \end{aligned}$$

当Zp’=0时(投影面是xoy平面),得斜投影变换矩阵

投影面是XOY平面时，
斜投影变换是图形对依赖轴Z轴的错切变换

$$X=X+g(z);$$
$$Y=Y+f(z);$$
$$Z=z;$$

$$\begin{pmatrix} 1 & 0 & -a/c & 0 \\ 0 & 1 & -b/c & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Outlines

Computer Viewing Process 计算机观察流程

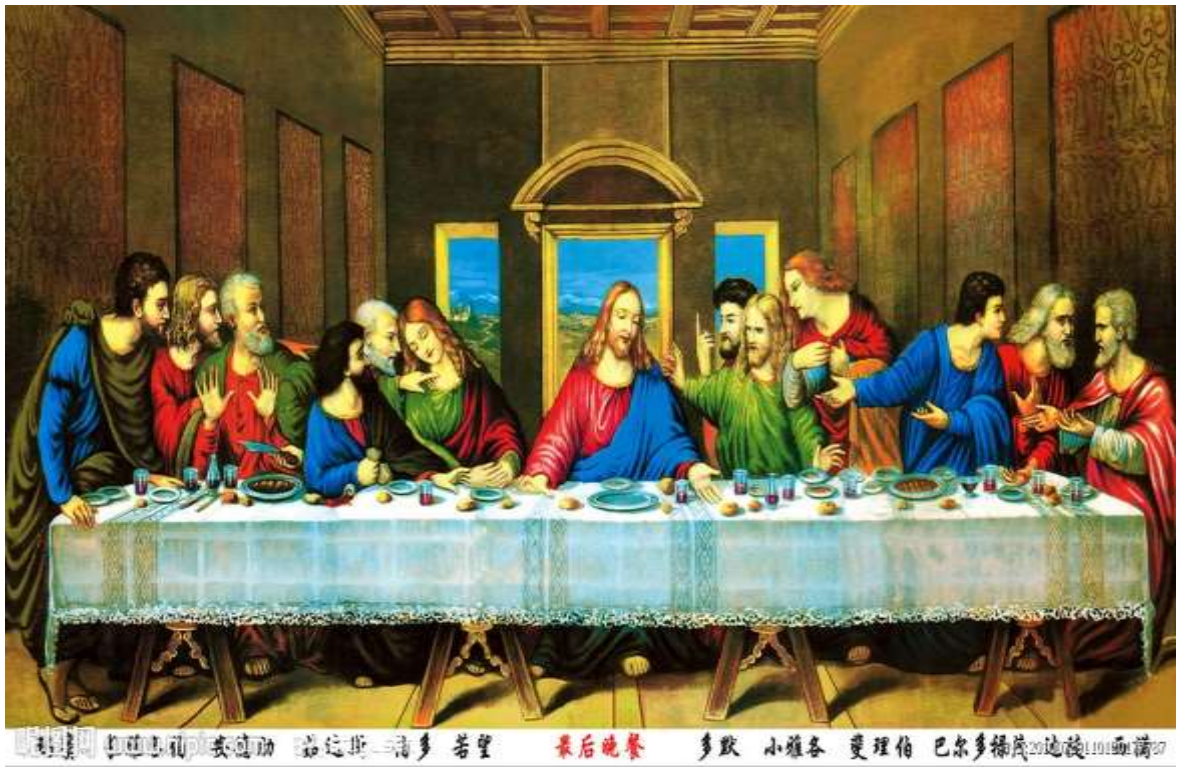
- Model Transformation模型变换
- View Transformation观察变换
- **Projection Transformation投影变换**
  - Classical Projection 经典投影
    - Parallel Projection平行投影
    - **Perspective Projection透视投影**
  - **Computer Projection 计算机投影***
    - Normalization Transformation规范化变换
    - Visual Volume Transformation视见体变换
    - Normalized Projection规范化投影变换
- **Window-Viewport Trans. 窗区视区变换/屏幕映射**

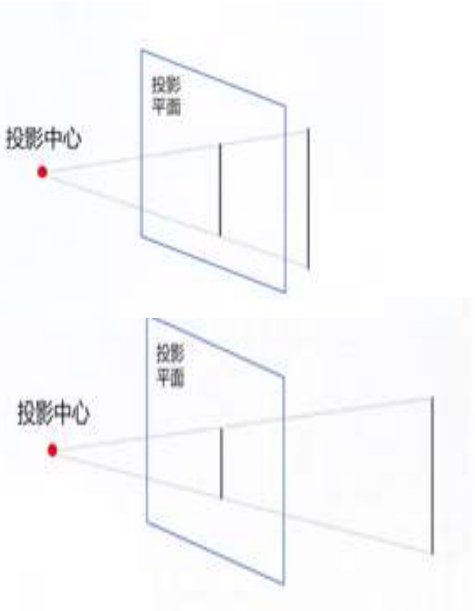
Classical Projection(cont.)

2.Perspective Projection透视投影

- 有真实感：成像原理类似于照相和人的视觉系统，深度感更强，更真实。
- 近大远小：不能如实地反映物体的精确尺寸和形状。

  透视缩小效应：三维形体透视投影后的大小，与形体到投影中心的距离成反比

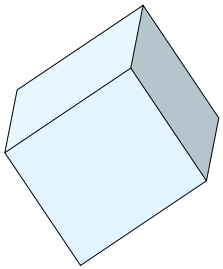




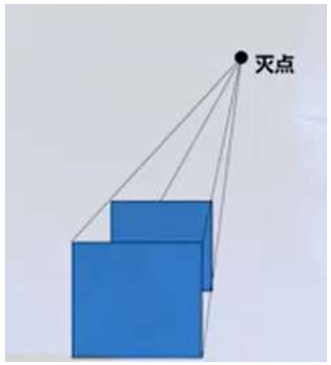
Classical Projection(cont.)

2.Perspective Projection透视投影（cont.)

- vanishing points 灭点
  - 不平行于投影面的一组平行线，在透视投影后，会汇聚成一个“灭点”
  - “灭点”可看作是无限远处的点在投影面上的投影。



平行投影：无灭点



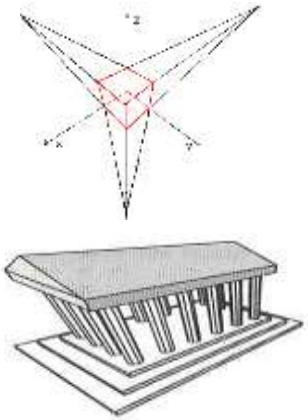
透视投影：有灭点

Classical Projection(cont.)

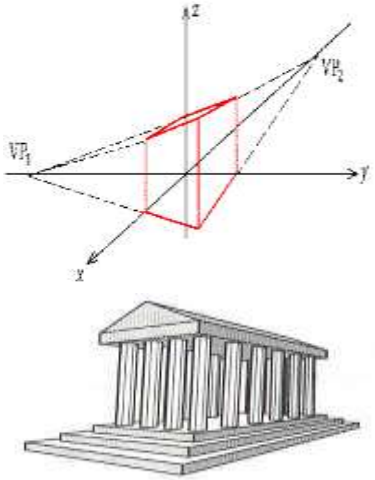
2.Perspective Projection透视投影（cont.)

主灭点：一组与主轴方向平行的平行线，在透视投影后所产生的灭点.

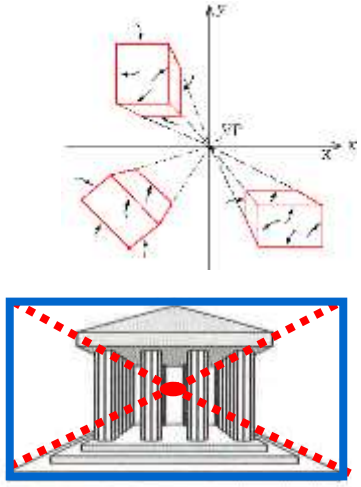
- 三维空间中只有三个主轴，所以最多只有三个“主灭点”，用来分类透视
  - 一点透视：产生1个主灭点，如Z轴方向平行线组投影后汇聚成1个主灭点
  - 二点透视：产生2个主灭点，如X轴，Y轴方向平行线组投影后，分别汇聚成2个主灭点，
  - 三点透视：产生3个主灭点,如X轴，Y轴，Z轴方向的平行线组投影后，分别汇聚成3个主灭点
    .



three-point perspective



two-point perspective



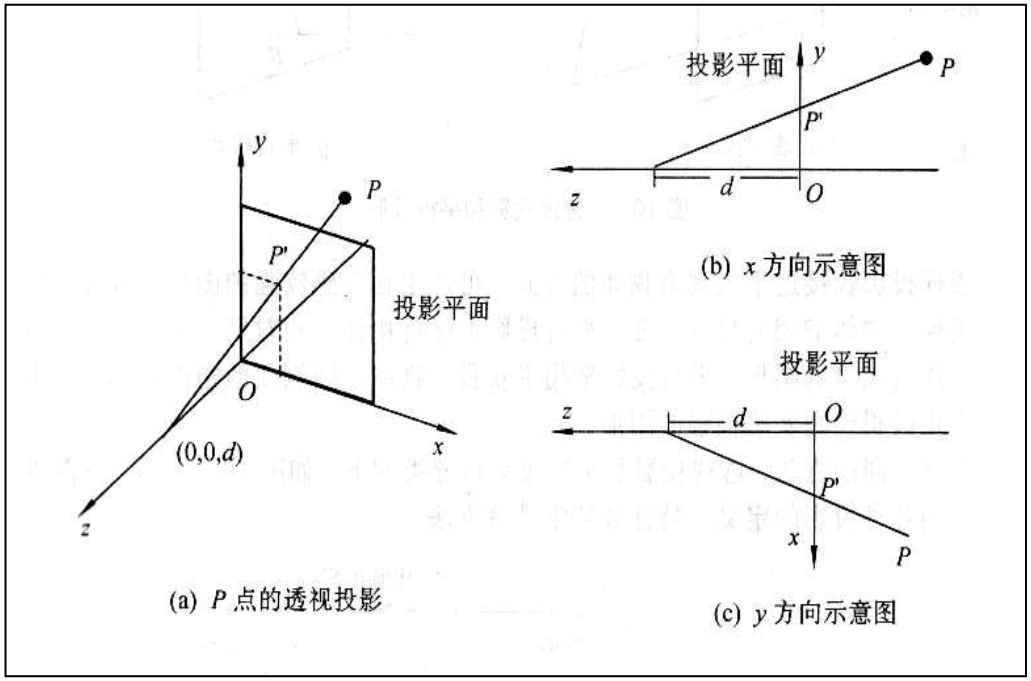
one-point perspective

Classical Projection(cont.)

- **2.Perspective Projection透视投影（cont.)**

一点透视矩阵的变换矩阵的推导

- 设投影中心（xc,yc,zc）为（0,0,d），投影面是XOY面,点P(x,y,z)投影后得到P’(x’,y’,z’) (这里z’=0)
- 据直线两点参数方程： (x’-xc)/(x-xc)=(y’-yc)/(y-yc)=(z’-zc)/(z-zc)=u
  - x’/x=y’/y=(z’-d)/(z-d)=u



$$\begin{cases} x' = xu \\ y' = yu \\ z' = (z-d)u + d \end{cases} \qquad (u \in [0,1])$$

$$\begin{cases} x' = x\left(\dfrac{d}{d-z}\right) = x\left(\dfrac{1}{1-z/d}\right) \\ y' = y\left(\dfrac{d}{d-z}\right) = y\left(\dfrac{1}{1-z/d}\right) \\ z' = 0 \end{cases}$$

Classical Projection(cont.)

2.Perspective Projection透视投影（cont.)

一点透视矩阵的推导(续)

令 **h=1-z/d**，可得投影变换矩阵如下**M**

经**M**投影变换后得到的坐标是齐次项为**h**的齐次坐标**(x'h,y'h,z'h,h)**，该齐次坐标需要除以齐次项**h**, 才能转换为真正的坐标**(x',y',z')**。

$$\begin{cases} x' = x(\dfrac{d}{d-z}) = x(\dfrac{1}{1-z/d}) \\ y' = y(\dfrac{d}{d-z}) = y(\dfrac{1}{1-z/d}) \\ z' = 0 \end{cases}$$

$$\begin{pmatrix} X'h \\ Y'h \\ Z'h \\ h \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -\dfrac{1}{d} & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

$$M = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -\dfrac{1}{d} & 1 \end{pmatrix}$$

Classical Projection(cont.)

2.Perspective Projection透视投影（cont.)

- 从矩阵分析一点，两点，三点透视投影变换
  - 列向量表示的4*4变换矩阵中，若p, q, r 不等于0，表示该变换是透视投影变换

$$M=\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&0&0\\0&0&r&1\end{pmatrix}\qquad M=\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&0&0\\0&q&r&1\end{pmatrix}\qquad M=\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&0&0\\p&q&r&1\end{pmatrix}$$

Outlines

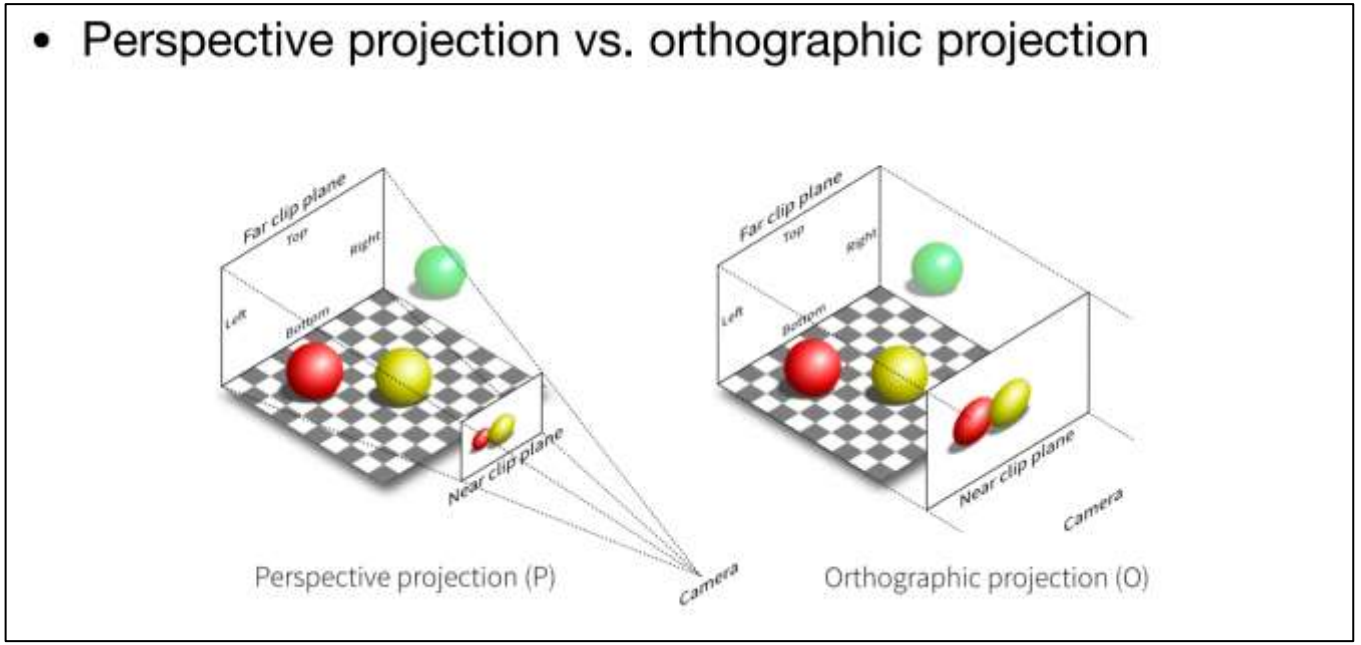
Computer Viewing Process 计算机观察流程

- Model Transformation模型变换
- View Transformation观察变换
- **Projection Transformation投影变换**
  - Classical Projection 经典投影
    - Parallel Projection平行投影
    - Perspective Projection透视投影
  - **Computer Projection 计算机投影***
    - Normalization Transformation规范化变换
    - Visual Volume Transformation视见体变换
    - Normalized Projection规范化投影变换
- **Window-Viewport Trans. 窗区视区变换/屏幕映射**

Computer Projection 计算机投影*

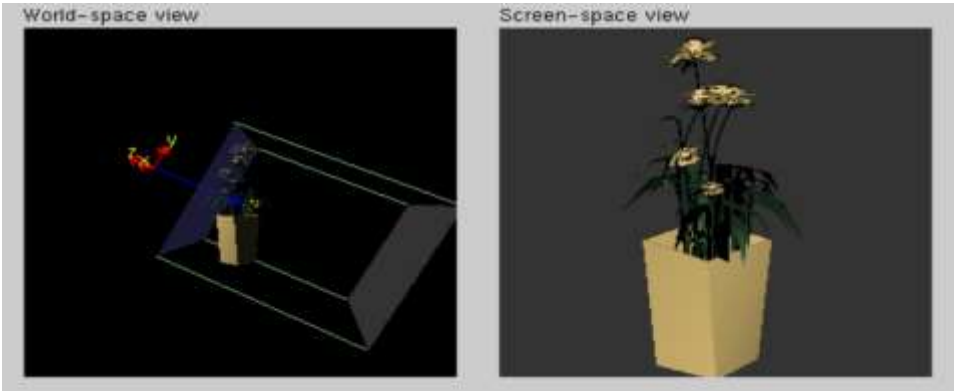
“计算机投影”并非是“经典投影”的照搬，具有以下特性：

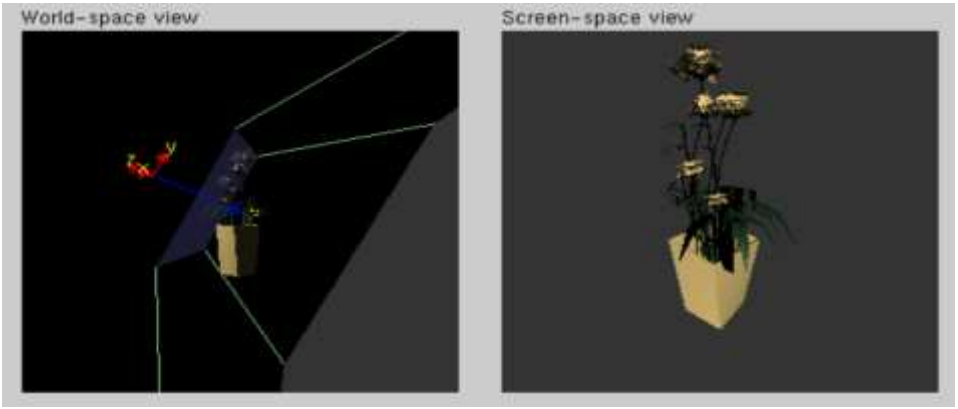
- **1）观察坐标：**投影在以观察点为原点,观察方向为Z向的VC坐标系下
- **2）观察方式：**可以平行投影或透视投影（一点）生成投影效果
- **3）取景范围：**场景中只有在“视见体”内的物体才会被投影显示，其外都被裁剪掉，并且对“视见体”进行了“规范化”（规约到$[-1,1]^3$)
- **4）保留Z值：**需要“保留深度信息”（后续消隐算法中用）



Classical Projection(cont.)

- 演示
  - 平行投影：保持平行线性，保持尺寸比例
  - 透视投影：类人眼，真实感强，不保持尺寸比例

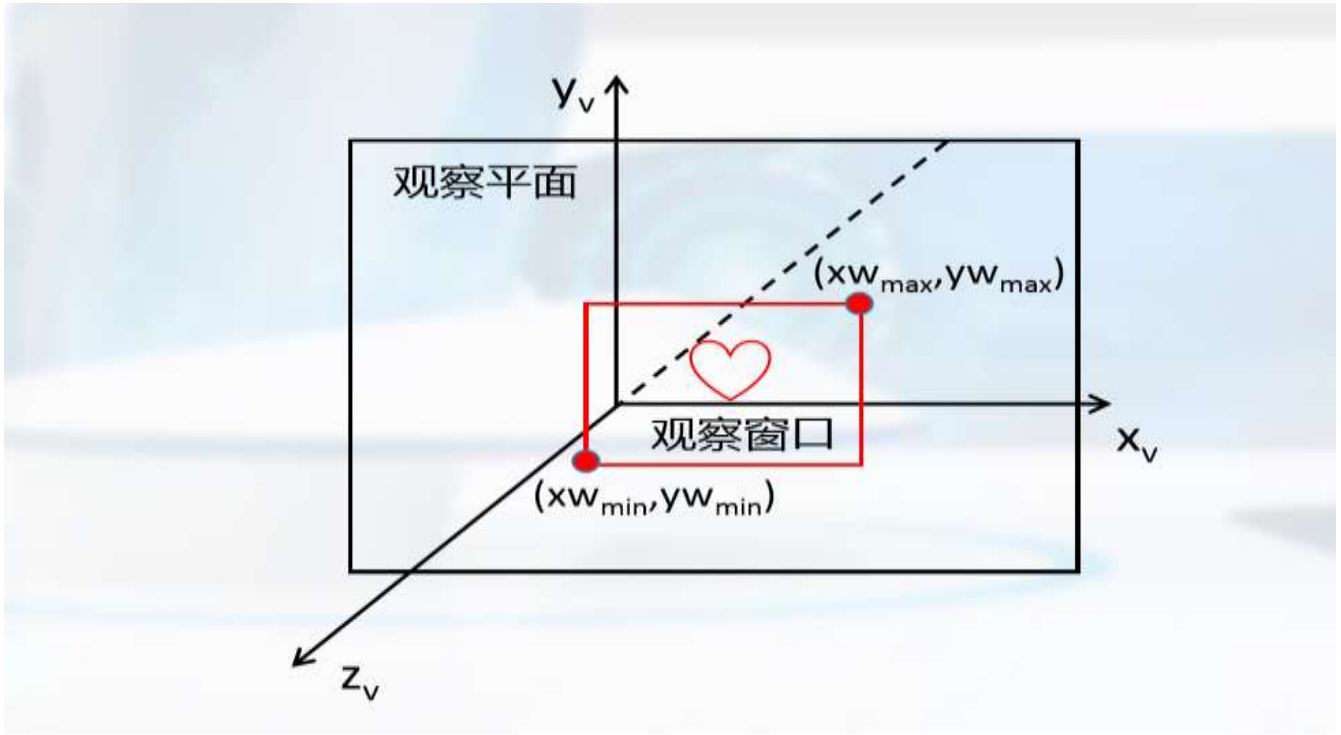




Computer Projection 计算机投影*

a.观察平面和观察窗口view window

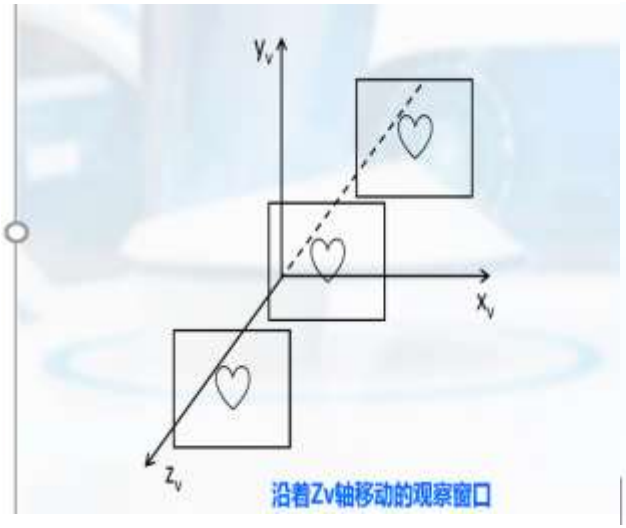
- 观察平面View plane：投影平面
- **观察窗口view window**：观察平面上的一个有限区域(矩形)

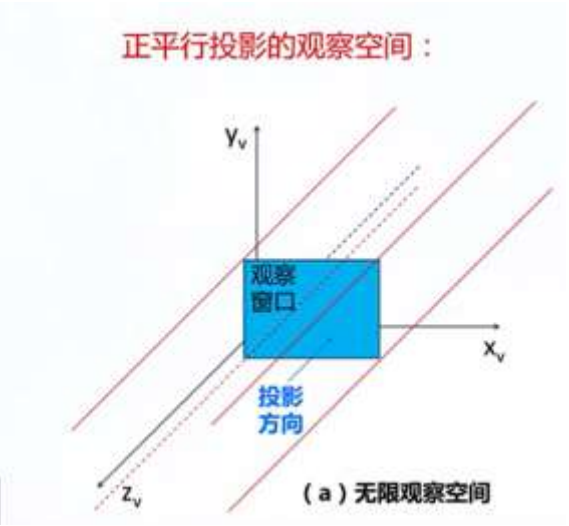


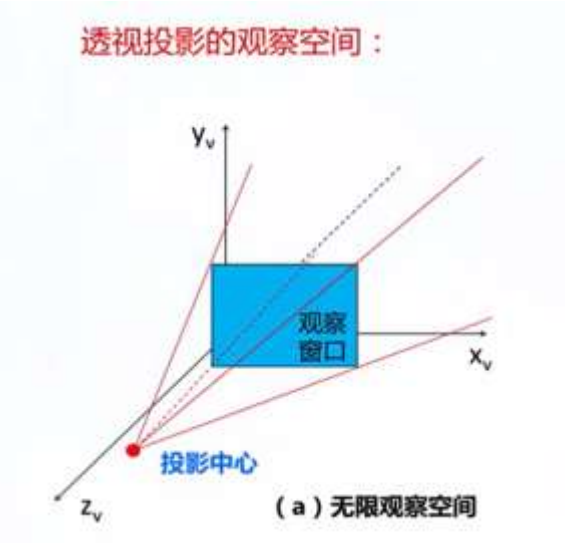
Computer Projection 计算机投影*

b.观察空间view space

- 观察窗口沿着观察方向运动得到的<u>无限的区域</u>，
  - 长方形管道（平行投影）或者棱锥（透视投影）



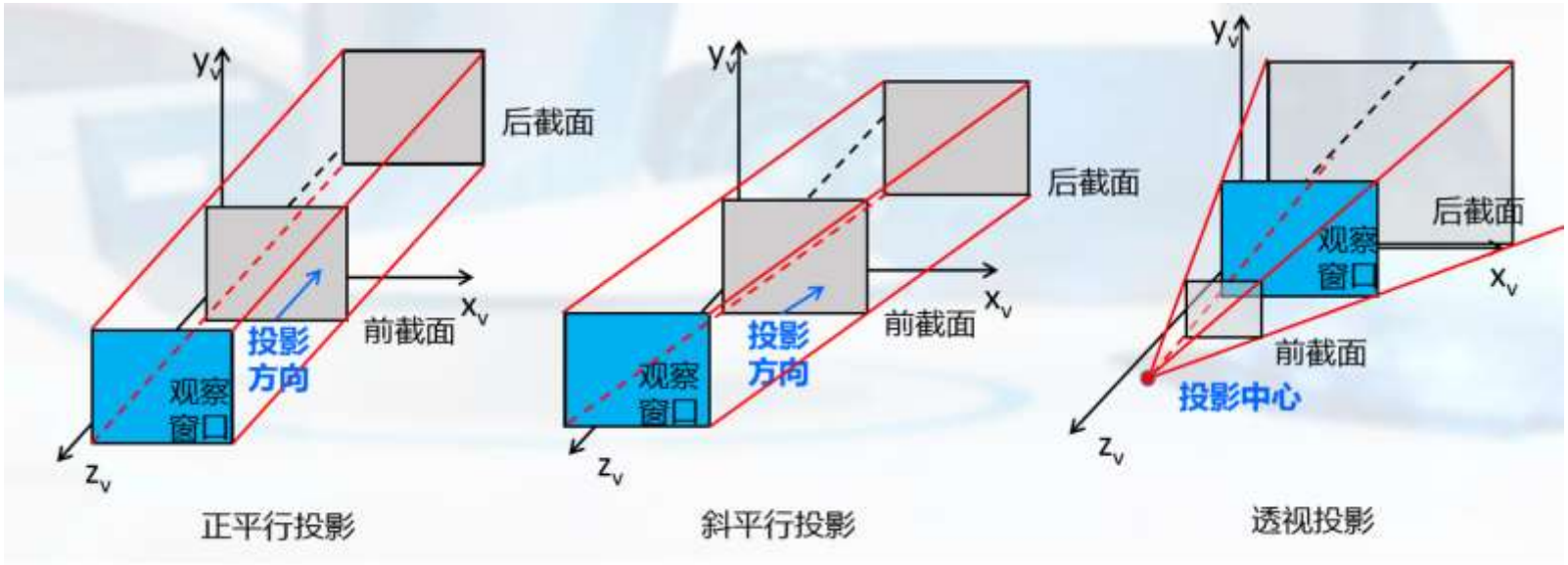




Computer Projection 计算机投影*

c. 视景体/视见体 View Frustum

- 定义了观察窗口和前后截面的三维有限空间
  - **正棱柱 or斜棱柱**
  - **正四棱台 or 斜四棱台**



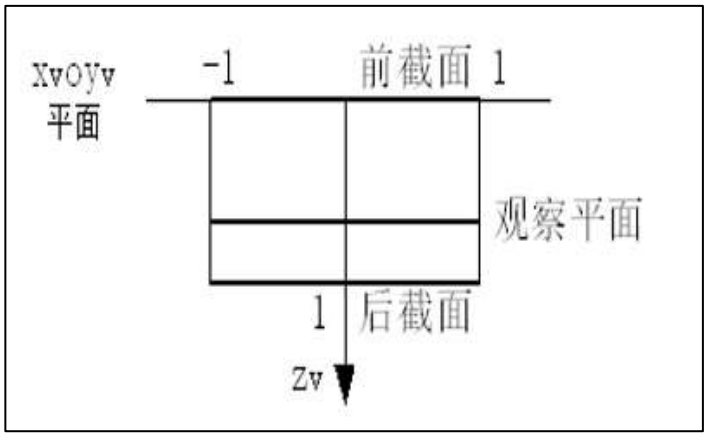
Outlines

Computer Viewing Process 计算机观察流程

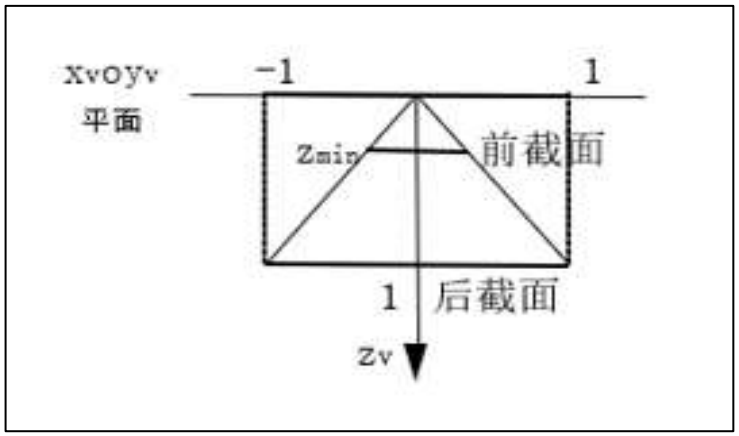
- Model Transformation模型变换
- View Transformation观察变换
- **Projection Transformation投影变换**
  - Classical Projection 经典投影
    - Parallel Projection平行投影
    - Perspective Projection透视投影
  - **Computer Projection 计算机投影***
    - Normalization Transformation规范化变换
    - Visual Volume Transformation视见体变换
    - Normalized Projection规范化投影变换
- **Window-Viewport Trans. 窗区视区变换/屏幕映射**

Normalization Transformation

- 规范化变换
  - 观察流程中，视景体/视见体 View Frustum需要进行规范化，使得景物表示从VC坐标转换为NDC坐标



平行投影的是规范化视见体



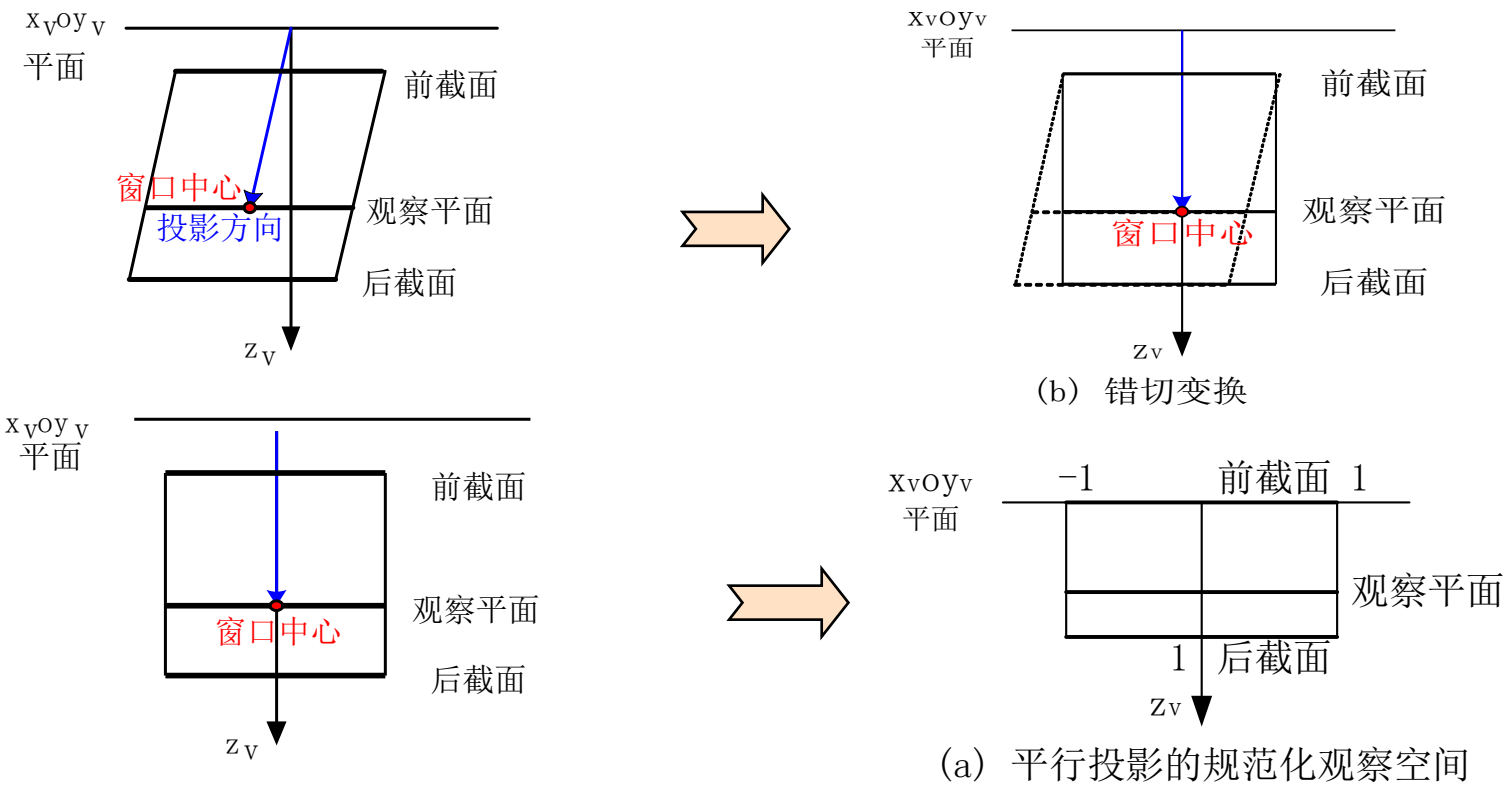
透视投影的规范化视见体

Normalization Transformation(cont.)

1.平行视见体的规范化变换

1)错切变换：将视见体中心线（观察方向）与Zv重合，得到正六面体

2)平移和缩放变换：得到平行投影的规范化正棱柱

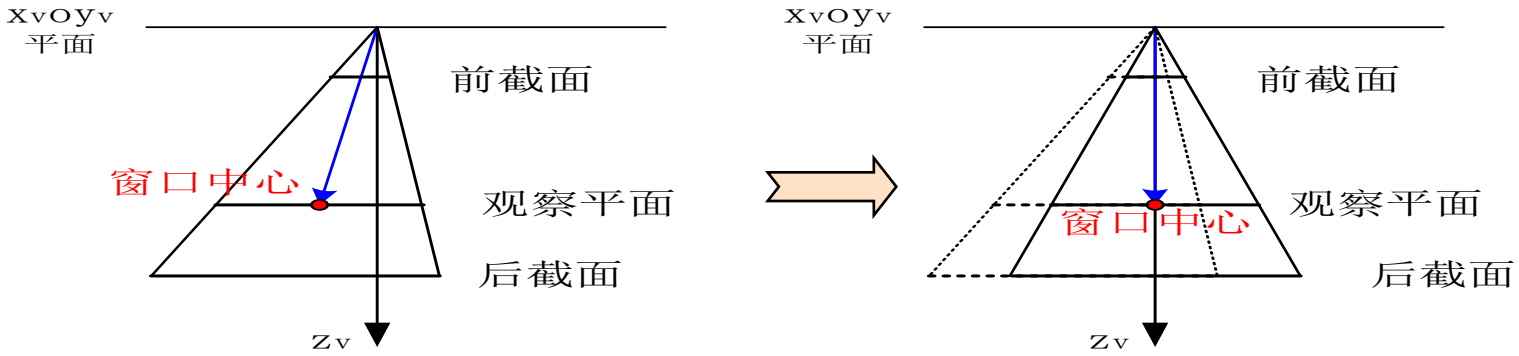


Normalization Transformation(cont.)

2.透视投影视见体的规范化变换

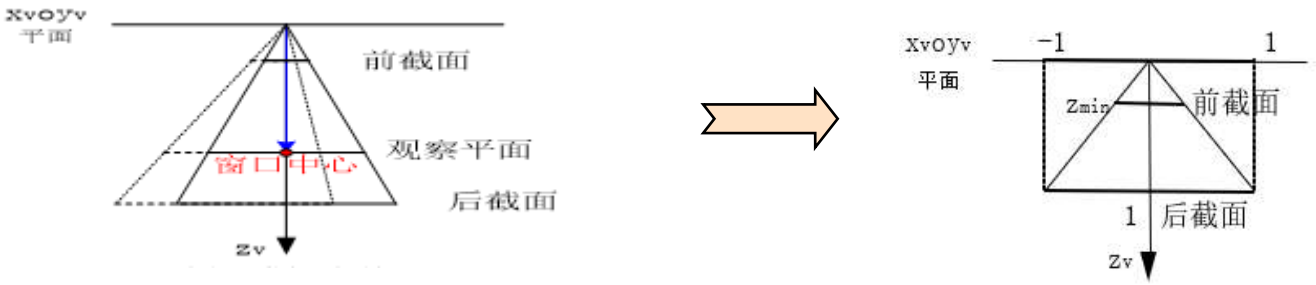
1)错切变换：若观察中心线方向不平行于Z轴，进行错切变换与Z轴重合。

2)缩放变换：再缩放，得到透视投影的规范化正棱台。



（a） 平移变换　　（b） 错切变换

图　透视投影的规范化投影变换步骤（1）（2）



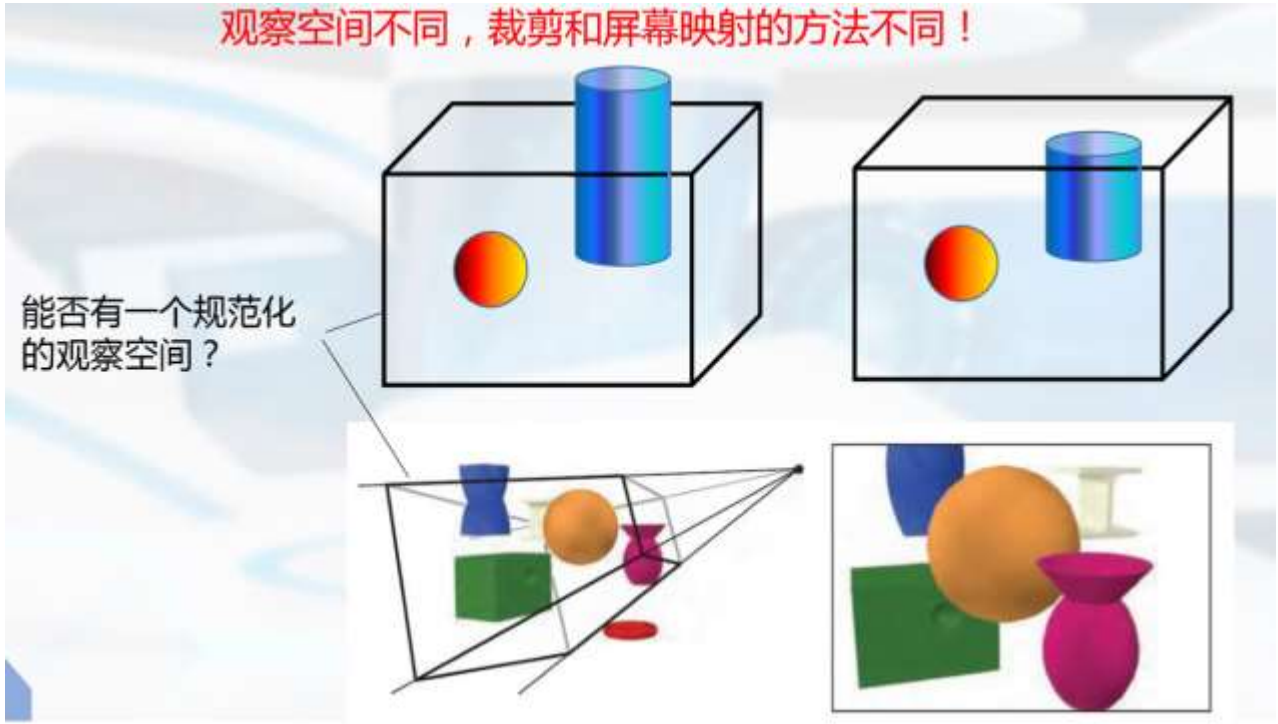
Outlines

Computer Viewing Process 计算机观察流程

- Model Transformation模型变换
- View Transformation观察变换
- **Projection Transformation投影变换**
  - Classical Projection 经典投影
    - Parallel Projection平行投影
    - Perspective Projection透视投影
  - **Computer Projection 计算机投影***
    - Normalization Transformation规范化变换
    - VisualVolume Transformation视见体变换
    - Normalized Projection规范化投影变换
- **Window-Viewport Trans. 窗区视区变换/屏幕映射**

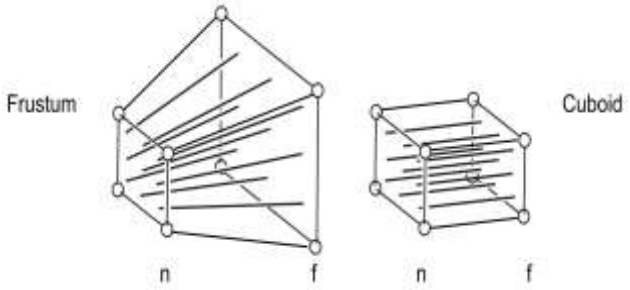
Visual Volume Transformation

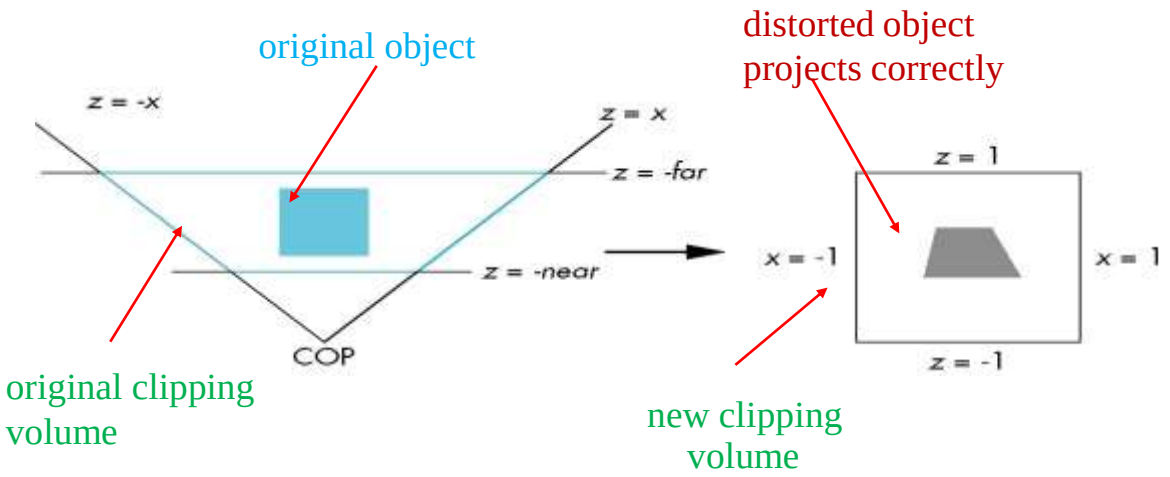
- 视见体变换
  - Why need Visual Volume Transformation?
    - 平行视见体（正棱柱）和透视视见体(正棱台）的不同，会导致后续的裁剪和屏幕映射的的算法不同！



Visual Volume Transformation(cont.)

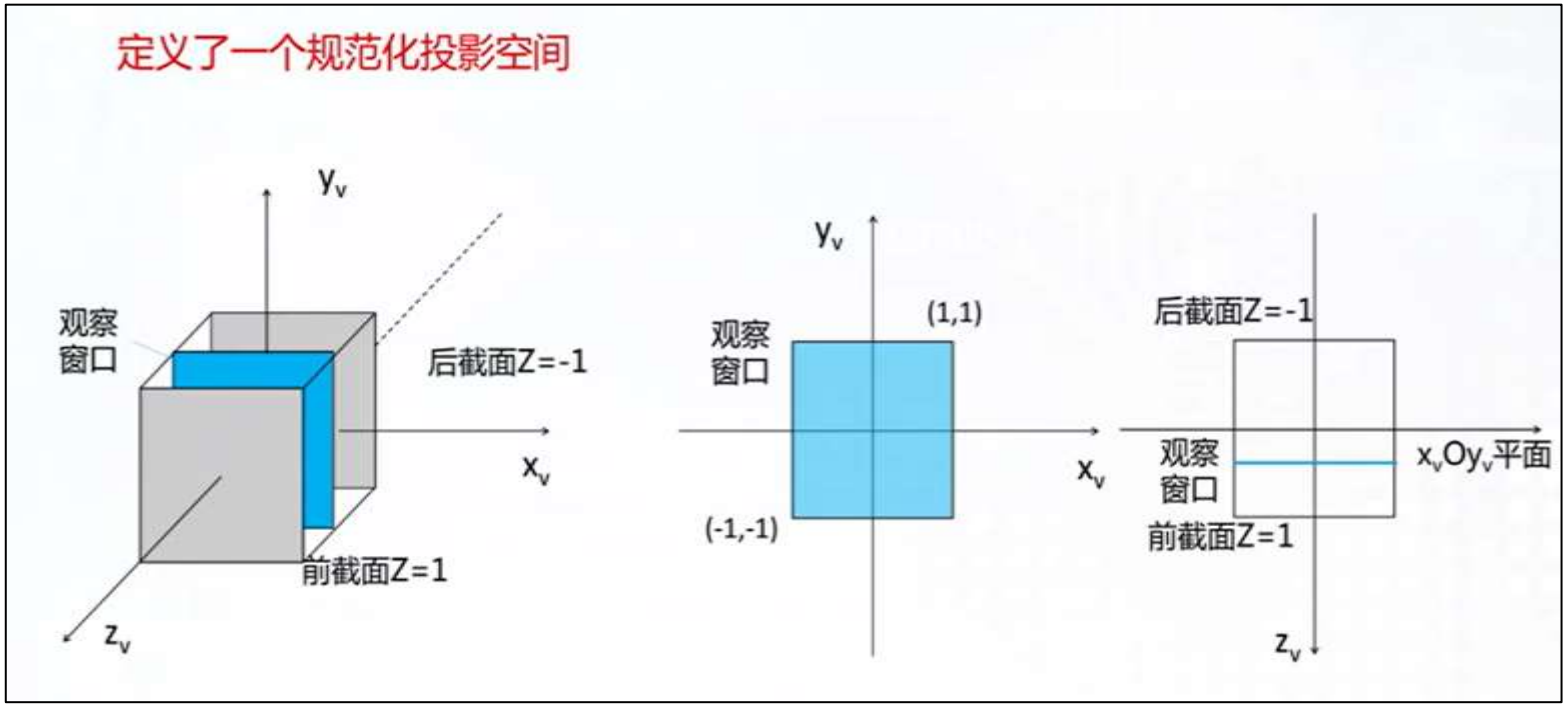
- 视见体变换：正棱台➔正棱柱
  - ✓ 变换矩阵推导参考：“虎书”以及**GAMES101** 闫老师讲解
  - **Projects correctly:**保留透视效果：“近大远小”的效果
  - **Distorted object:** 物体有变形，但是在观察方向上的深度关系不会改变！





Visual Volume Transformation(cont.)

- 最后平行和透视视见体都转成了规范化视见体
  - Normalized Visual Volume规范化视见体 $[-1,1]^3$



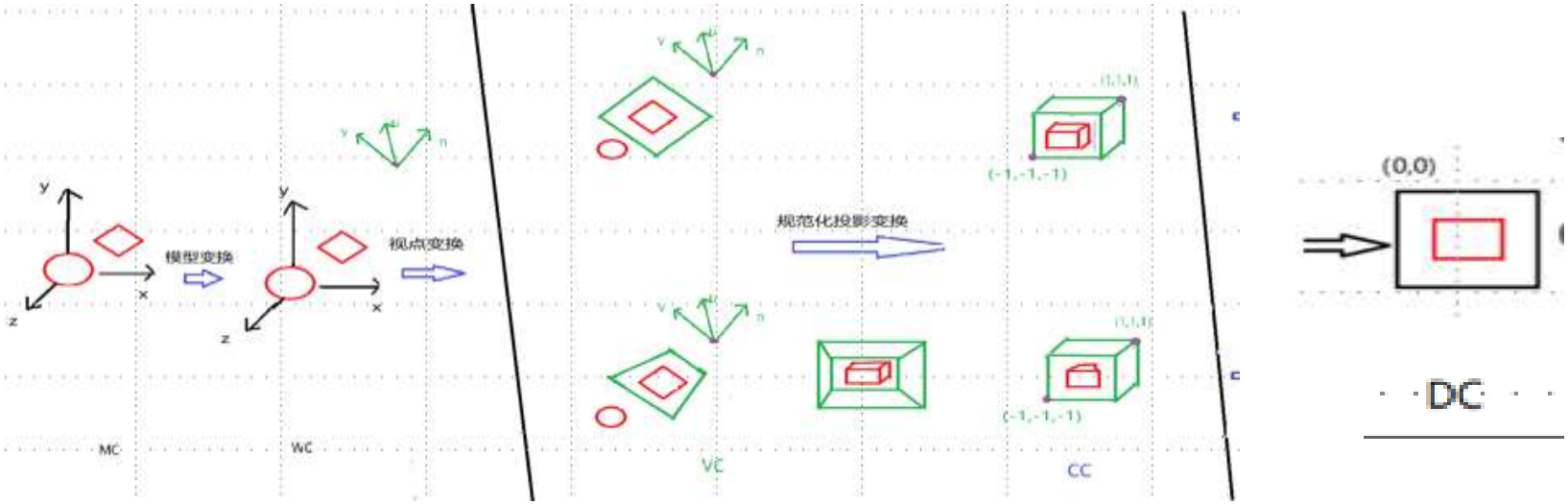
Outlines

Computer Viewing Process 计算机观察流程

- Model Transformation模型变换
- View Transformation观察变换
- **Projection Transformation投影变换**
  - Classical Projection 经典投影
    - Parallel Projection平行投影
    - Perspective Projection透视投影
  - **Computer Projection 计算机投影***
    - Normalization Transformation规范化变换
    - Visual Volume Transformation视见体变换
    - Normalized Projection规范化投影
- **Window-Viewport Trans. 窗区视区变换/屏幕映射**

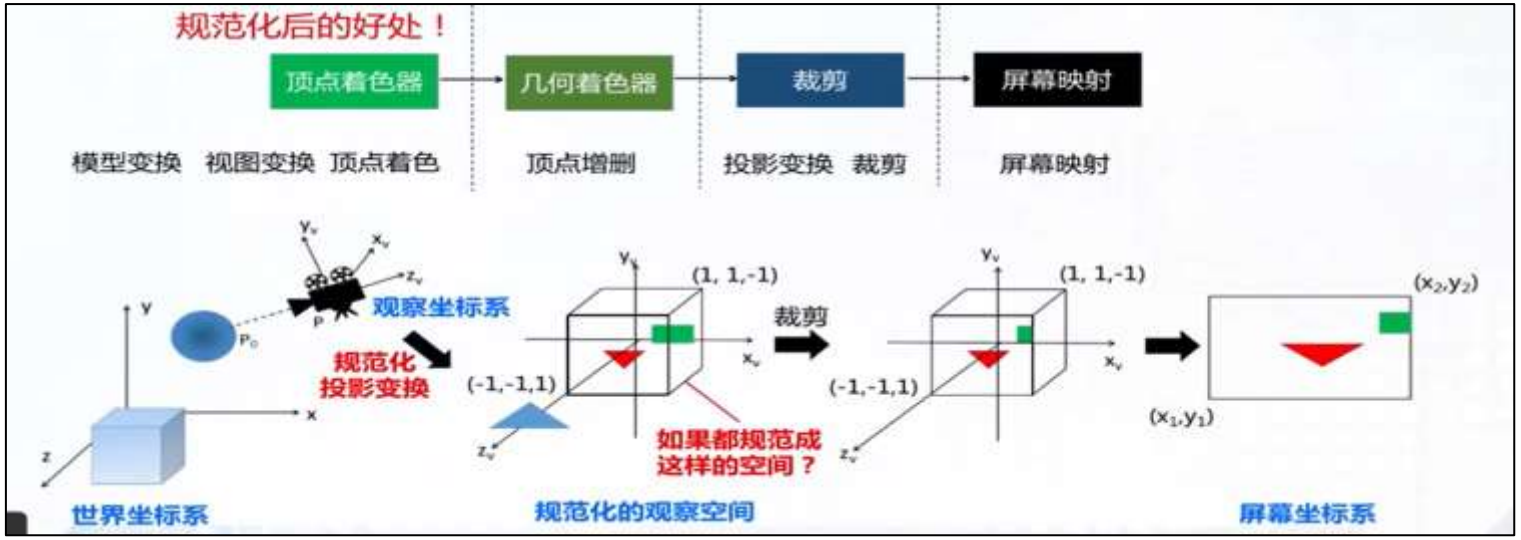
Normalized Projection

- **Normalized Projection规范化投影**
  - 平行视见体，经过规范化变换，转化为**规范化视见体**$[-1,1]^3$
    - 平行投影的规范化投影=规范化变换
  - 透视视见体，经过规范化变换及视见体变换后，转化为规范化视见体$[-1,1]^3$
    - 透视投影的规范化投影=规范化变换+视见体变换



Normalized Projection (cont.)

- Why Normalized Projection？
  - 采用规范化投影的好处：
    - 将平行和透视两种视见体都转为规范化视见体，可统一简化后续的“裁剪”及“屏幕映射”操作。 (Allows for a single pipeline for both perspective and orthogonal viewing)
    - 仍保留了Z深度信息，可为后续“消隐”等处理提供判定依据。（Stay in four dimensional homogeneous coordinates as long as possible to retain three-dimensional information needed）



Normalized Projection (cont.)

- 一般图形包提供实现两种规范化投影

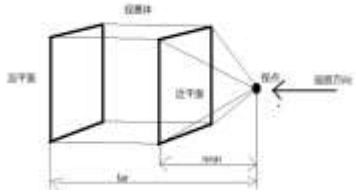
1. 规范化正投影
2. 规范化一点透视投影

Normalized Projection(cont.)

- “规范化正投影”

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

- 注1.VC下参数，left,right,bottom,top是观察窗口坐标
- 注2.far>near>0，near，far是前后截面到视点的距离值
- 注3.可逆的变换，保留了深度(z)信息
- 下面是正投影时（正六面体）规范化变换过程：

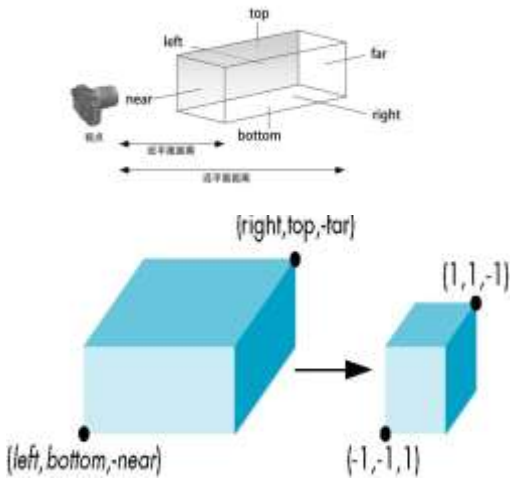


  - T:Move center to origin

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

  - S:Scale to have sides of length 2

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

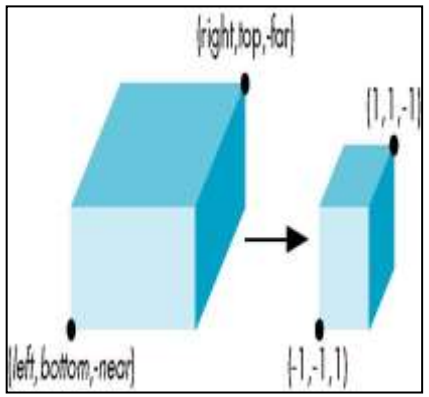


$$\mathbf{P} = \mathbf{ST} = \begin{bmatrix} \frac{2}{right-left} & 0 & 0 & -\frac{right+left}{right-left} \\ 0 & \frac{2}{top-bottom} & 0 & -\frac{top+bottom}{top-bottom} \\ 0 & 0 & \frac{2}{near-far} & \frac{far+near}{far-near} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Normalized Projection(cont.)

- “规范化正投影”（续）

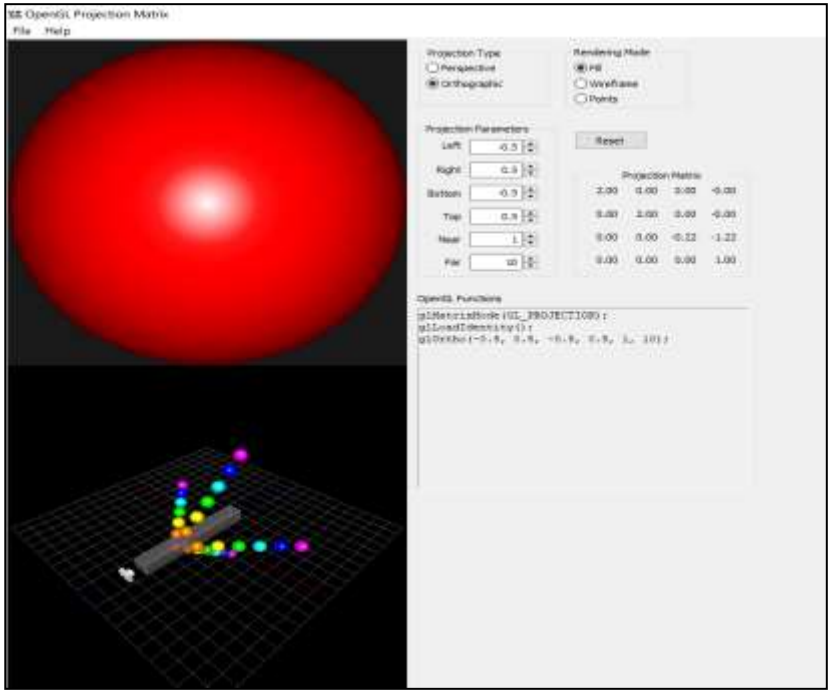
// AngleCode 提供Ortho函数 in MV.Js 如下：

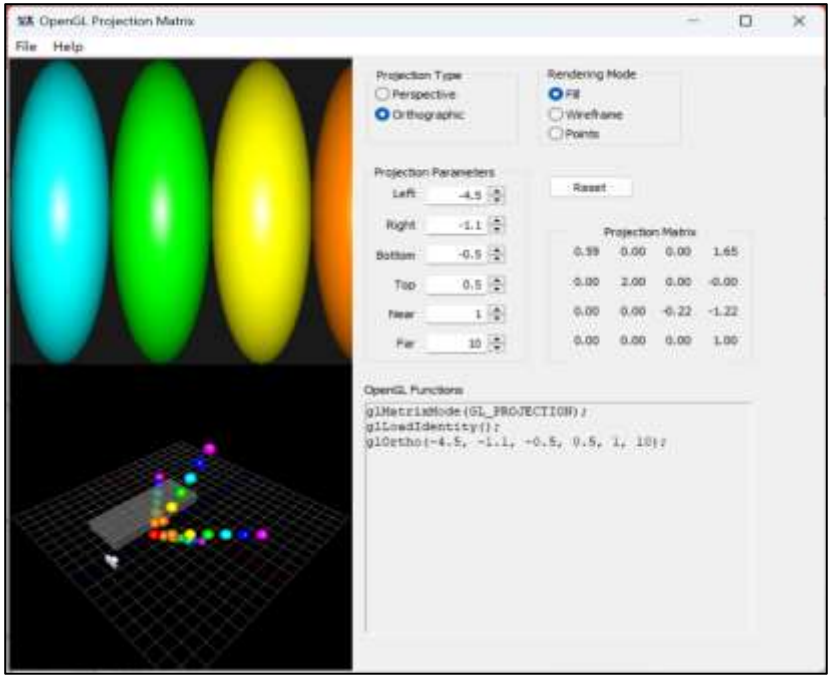



```
function ortho( left, right, bottom, top, near, far )
{
    if ( left == right ) { throw "ortho(): left and right are equal"; }
    if ( bottom == top ) { throw "ortho(): bottom and top are equal"; }
    if ( near == far )   { throw "ortho(): near and far are equal"; }

    var w = right - left;
    var h = top - bottom;
    var d = far - near;

    var result = mat4();
    result[0][0] = 2.0 / w;
    result[1][1] = 2.0 / h;
    result[2][2] = -2.0 / d;
    result[0][3] = -(left + right) / w;
    result[1][3] = -(top + bottom) / h;
    result[2][3] = -(near + far) / d;

    return result;
}
```

Normalized Projection(cont.)

- “规范化正投影”(续) 演示



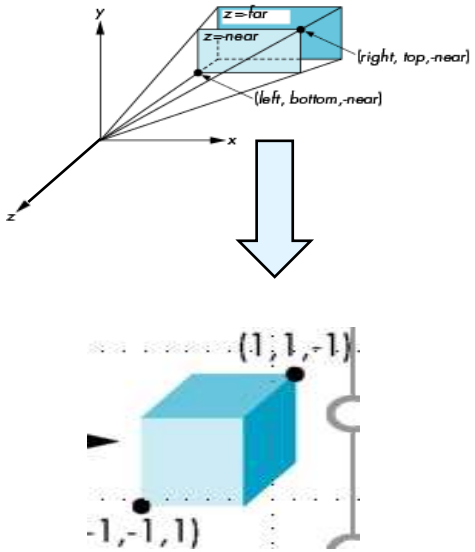


Normalized Projection(cont.)

- **规范化1点透视投影**

第1种： glFrustum(left,right,bottom,top,near,far)

- 注1.VC下参数，left,right,bottom,top是观察窗口坐标
- 注2：near，far是前截面和后截面距离视点的距离，且far>near>0
- 注3：是可逆的变换，保留了深度(z)信息，一点透视效果
- 注4：该矩阵是从“正/斜棱台”到“单位立方体”的变换矩阵



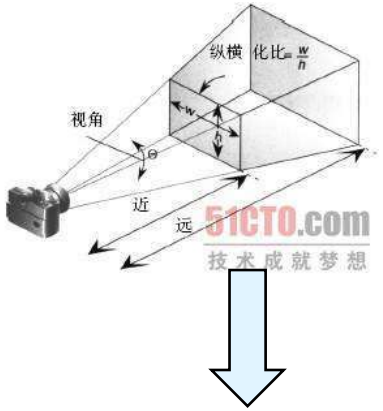
$$\begin{bmatrix} \frac{2*near}{right-left} & 0 & \frac{right-left}{right-left} & 0 \\ 0 & \frac{2*near}{top-bottom} & \frac{top+bottom}{top-bottom} & 0 \\ 0 & 0 & -\frac{far+near}{far-near} & -\frac{2*far*near}{far-near} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

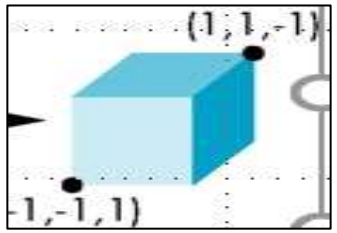
Normalized Projection(cont.)

- 规范化透视投影（续）

第2种： perspective(fovy, aspect, near, far)

- 注1：fovy是视角，aspect是横纵比w/h,四棱台是对称的
- 注2：far>near>0，far，near是后截面，前截面到视点的距离值
- 注3：该变换是可逆的变换，保留了深度(z)信息，一点透视效果，
- 注4： 该变换是“正棱台”到“规范化视见体”的变换矩阵。





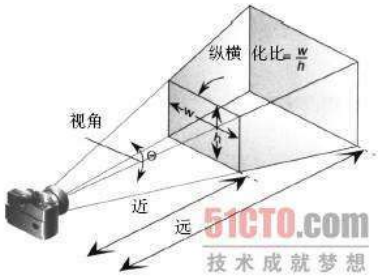
$$\begin{bmatrix} \frac{1}{\tan(fovy)*aspect} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan(fovy)} & 0 & 0 \\ 0 & 0 & -\frac{far+near}{far-near} & -\frac{2*far*near}{far-near} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

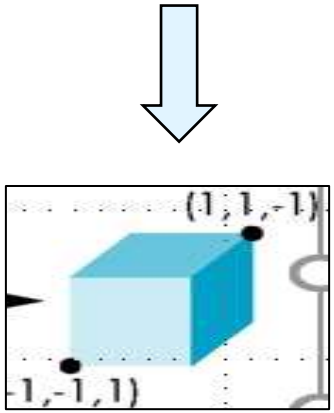
Normalized Projection(cont.)

- 规范化透视投影（续）

第2种： perspective(fovy, aspect, near, far)

//Angel Code提供了Perspective函数 in MV.Js

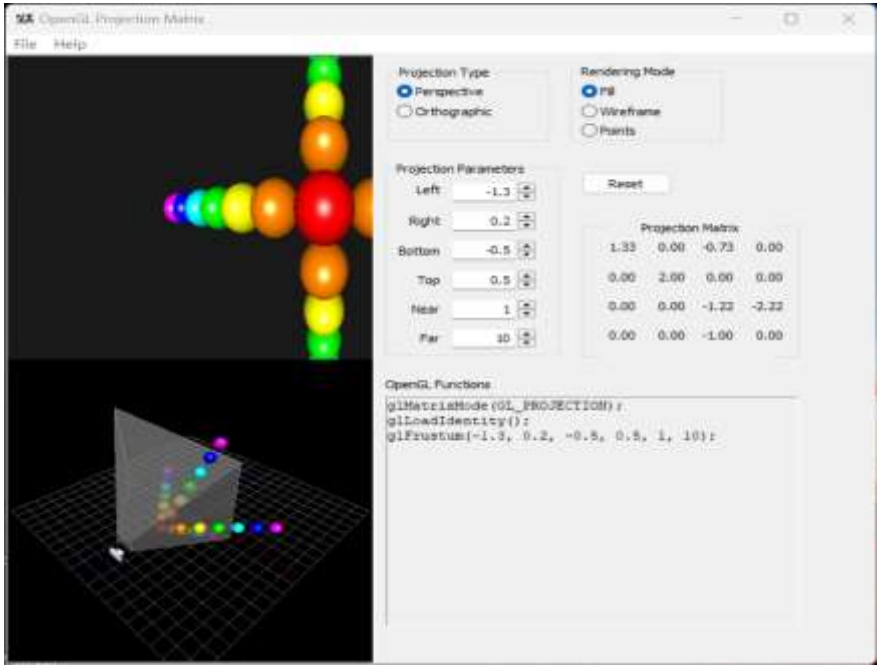


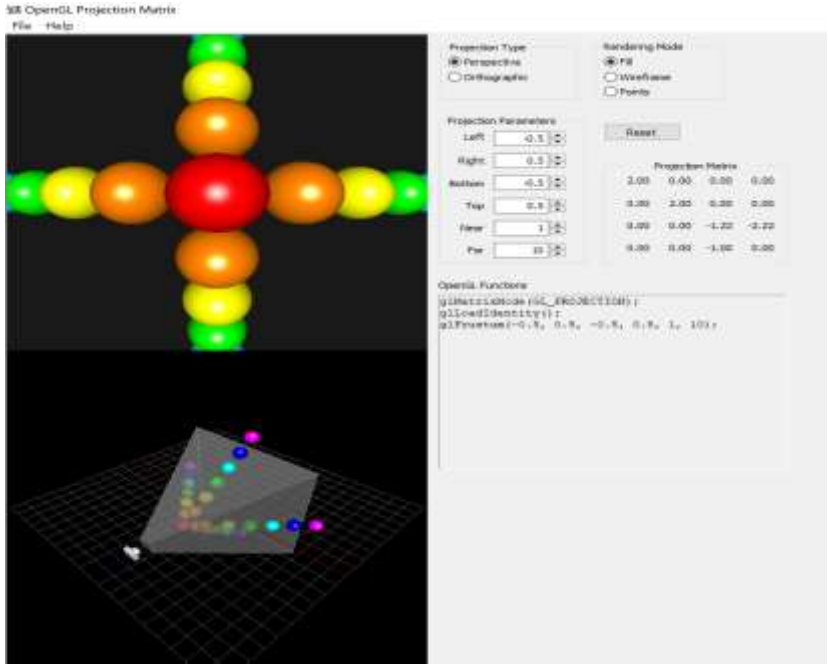



```
/*生成透视投影矩阵：*/
function perspective( fovy, aspect, near, far )
{
    var f = 1.0 / Math.tan( radians(fovy) / 2 );
    var d = far - near;

    var result = mat4();
    result[0][0] = f / aspect;
    result[1][1] = f;
    result[2][2] = -(near + far) / d;
    result[2][3] = -2 * near * far / d;
    result[3][2] = -1;
    result[3][3] = 0.0;

    return result;
}
```

Normalized Projection(cont.)

- 规范化透视投影：演示



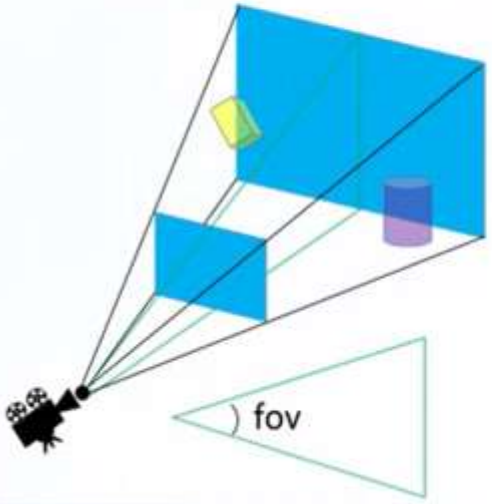


Normalized Projection(cont.)

- OpenGL提供了“规范化投影”的API函数
  - glm::ortho()　　//规范化正投影
  - glm::perspective() //规范化透视投影

规范化投影空间的过程：矩阵运算

OpenGL中的投影矩阵




```
glm::mat4 proj =glm::perspective(45.0f, 1.3f, 0.1f, 100.0f);
```

- 第一个参数定义了fov的值，它表示的是视野(Field of View)，并且设置了观察空间的大小。对于一个真实的观察效果，它的值经常设置为45.0，但想要看到更多结果你可以设置一个更大的值。
- 第二个参数设置了宽高比，就是宽度和高度的比例。
- 第三和第四个参数设置了近截面和远截面的位置。我们经常设置近距离为0.1而远距离设为100.0。所有在近平面和远平面的顶点且处于平截头体内的顶点都会被渲染。

Normalized Projection(cont.)

- 规范化投影实例
  - \Angle8E Code\05\ortho*
  - \Angle8E Code\05\perspective*

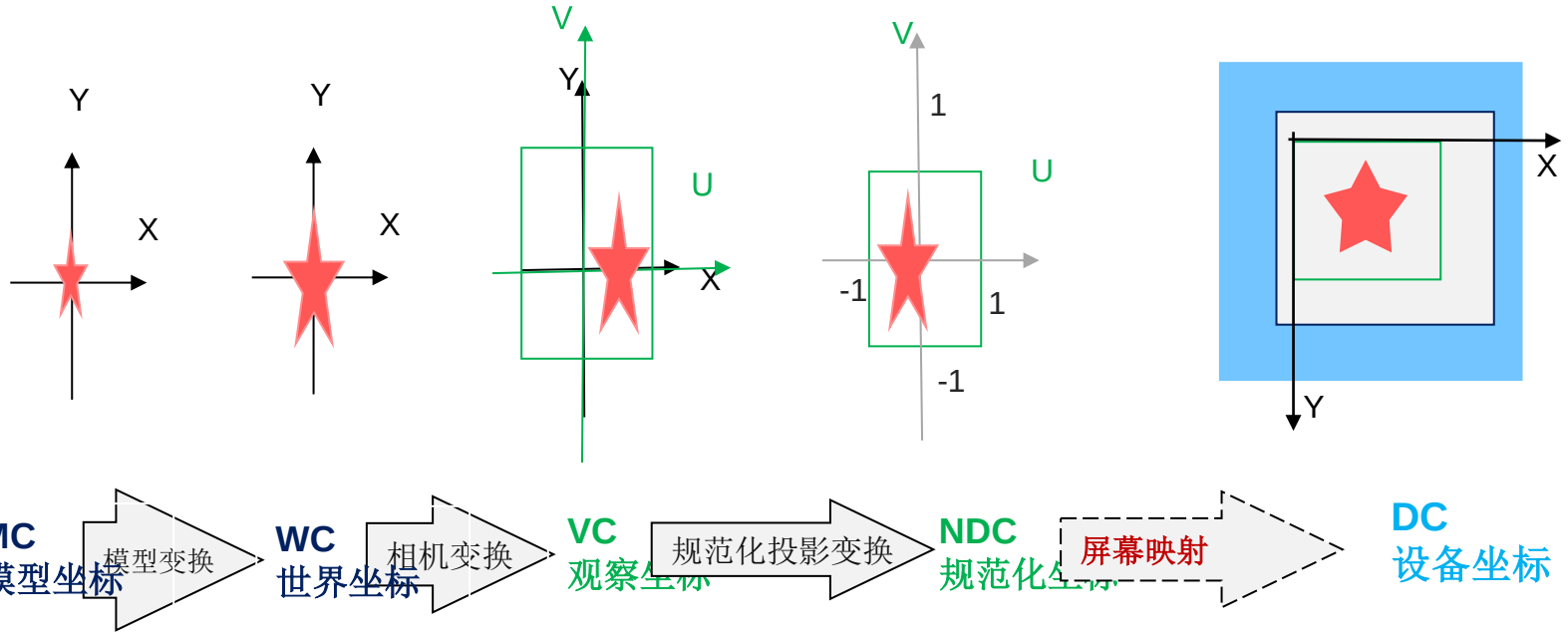
Outlines

Computer Viewing Process 计算机观察流程

- Model Transformation模型变换
- View Transformation观察变换
- **Normalized Projection Transformation规范化投影变换***
  - Classical Projection 经典投影
    - Parallel Projection平行投影
    - Perspective Projection透视投影
  - **Computer Projection 计算机投影***
    - Normalization Transformation规范化变换
    - Visual Volume Transformation视见体变换
    - **Normalized Projection规范化投影变换**
- **Window-Viewport Transformation 窗区视区变换**

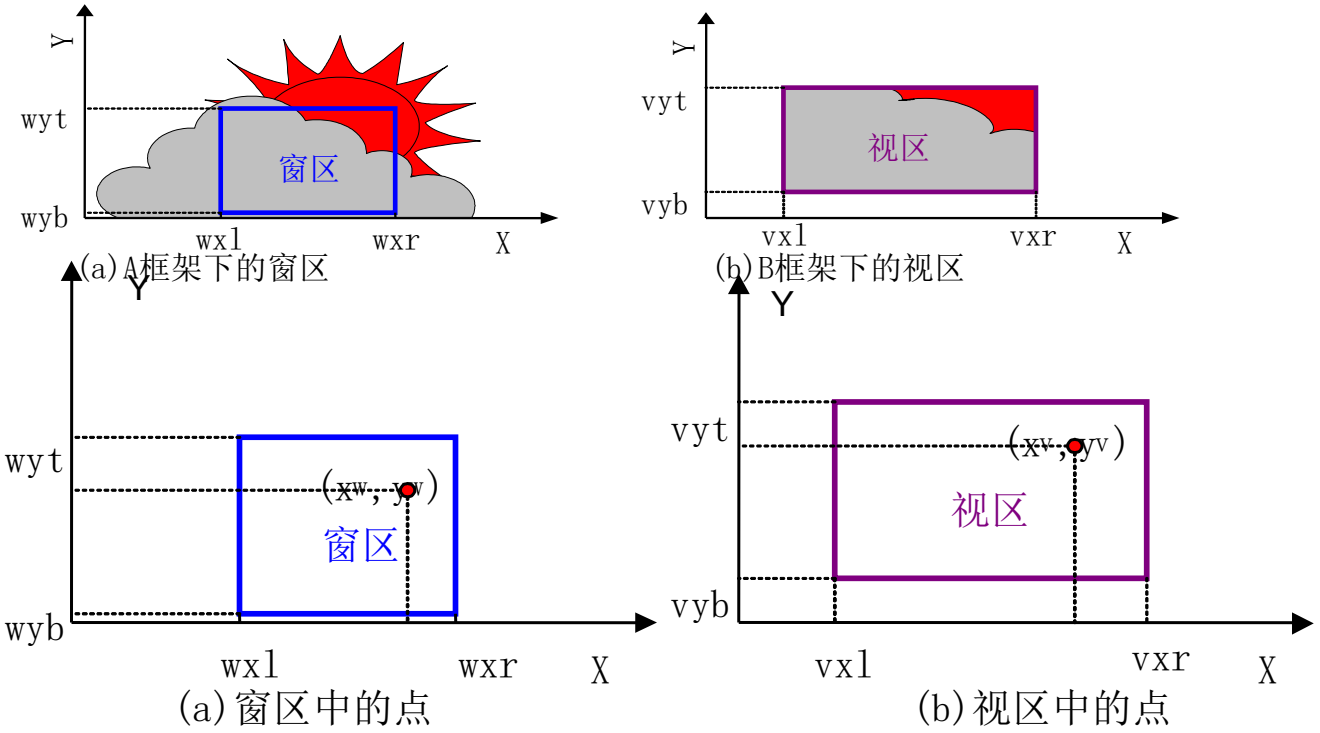
Window-Viewport Transformation

- 屏幕映射如何实现？
  - 实际上采用的是“窗区视区映射”变换**Window-Viewport Transformation**



Window-Viewport Trans.(cont.)

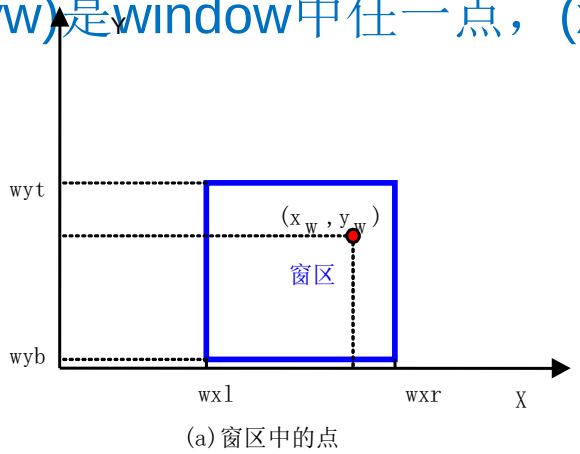
- 窗区视区变换
  - 从一个区域”窗区”到另一个区域”视区”的坐标映射！



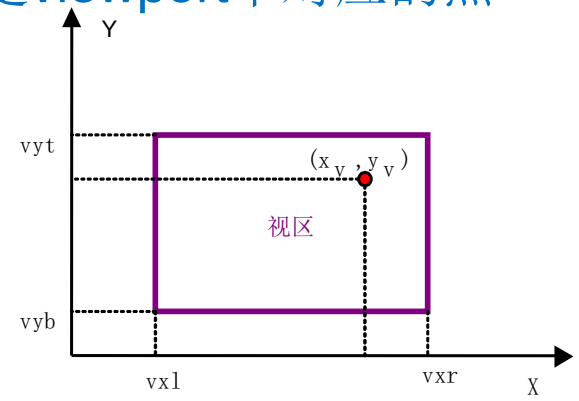
Window-Viewport Trans.(cont.)

- 窗区视区变换（继续）
  - 方法1找不变量：点到边界的距离保持 “对应成比例”

设：(xw,yw)是window中任一点，(xv,yv)是viewport中对应的点



(a) 窗区中的点



(b) 视区中的点

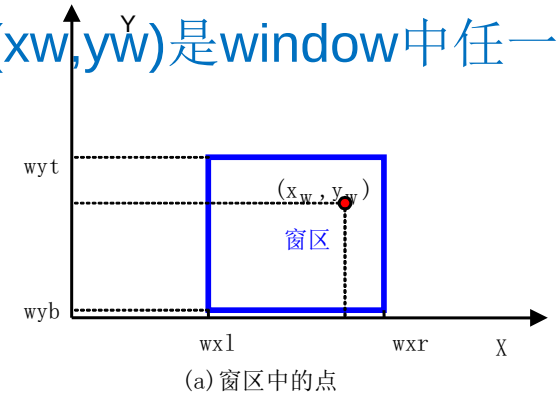
$$\frac{x_w - X_{w\text{-}min}}{X_{w\text{-}max} - X_{w\text{-}min}} = \frac{x_v - X_{v\text{-}min}}{X_{v\text{-}max} - X_{v\text{-}min}}$$

$$\frac{y_w - Y_{w\text{-}min}}{Y_{w\text{-}max} - Y_{w\text{-}min}} = \frac{y_v - Y_{v\text{-}min}}{Y_{v\text{-}max} - Y_{v\text{-}min}}$$

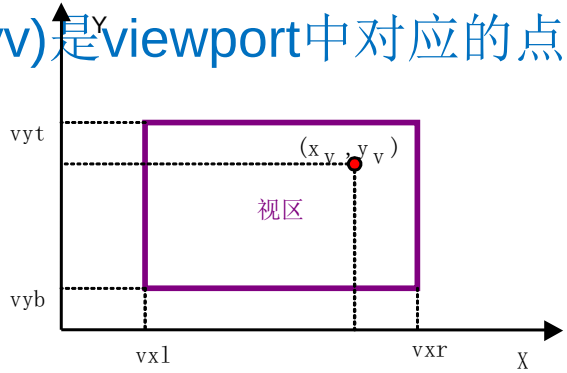
Window-Viewport Trans.(cont.)

- 窗区视区变换（继续）
  - 方法2：推导串联变换

设：(xw,yw)是window中任一点，(xv,yv)是viewport中对应的点



(a) 窗区中的点



(b) 视区中的点

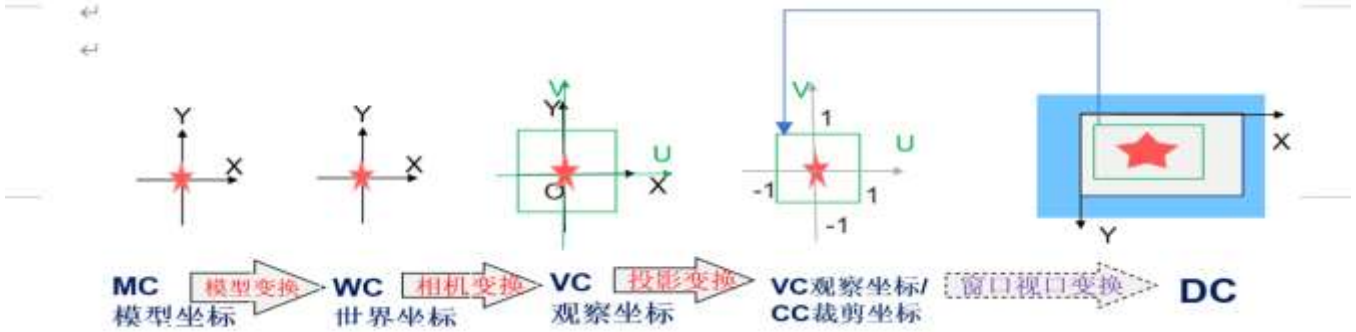
$M=T(X_{v\text{-}min}, Y_{v\text{-}min})\ S(S_X, S_Y)\ T(-X_{w\text{-}min}, -Y_{w\text{-}min})$

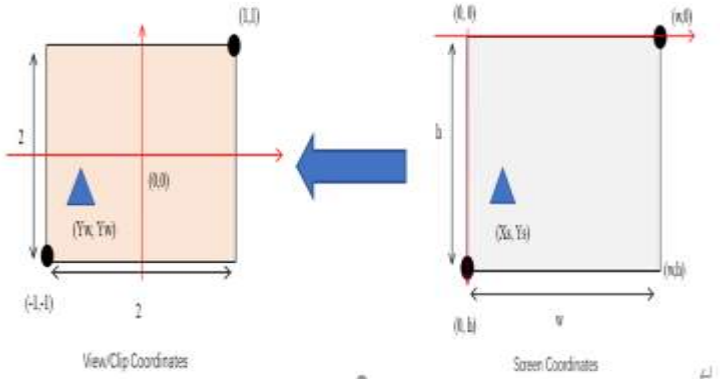
$//S_X = (x_{V\text{-}MAX}-x_{V\text{-}min})/(x_{W\text{-}MAX}-x_{W\text{-}min})$，$S_Y = (y_{V\text{-}MAX}-y_{V\text{-}min})/(y_{W\text{-}MAX}-y_{W\text{-}min})$

- $x_v=S_x(x_w-X_{w\text{-}min})+X_{v\text{-}min}$
- $y_v=S_y(y_w-Y_{w\text{-}min})+Y_{v\text{-}min}$

Window-Viewport Trans.(cont.)

- 窗区视区变换（继续）
  - 应用1：从2D屏幕坐标到2D裁剪坐标/NDC坐标





1.直接根据对应成比例进行推导：

$$\frac{Xw-Xwleft}{Xwright-Xwleft}=\frac{Xs-Xsleft}{Xsritht-Xsleft}$$

$$\frac{Yw-Ywbottom}{Ywtop-Ywbottom}=\frac{Ys-Ysbottom}{Ystop-Ysbottom}$$

视区左上角（xsleft，ysbottom）= (0，0) 时代入数据：

$$\frac{Xw-(-1)}{1-(-1)}=\frac{Xs-0}{w-0}$$ 得到 $Xw=-1+\frac{2Xs}{w}$

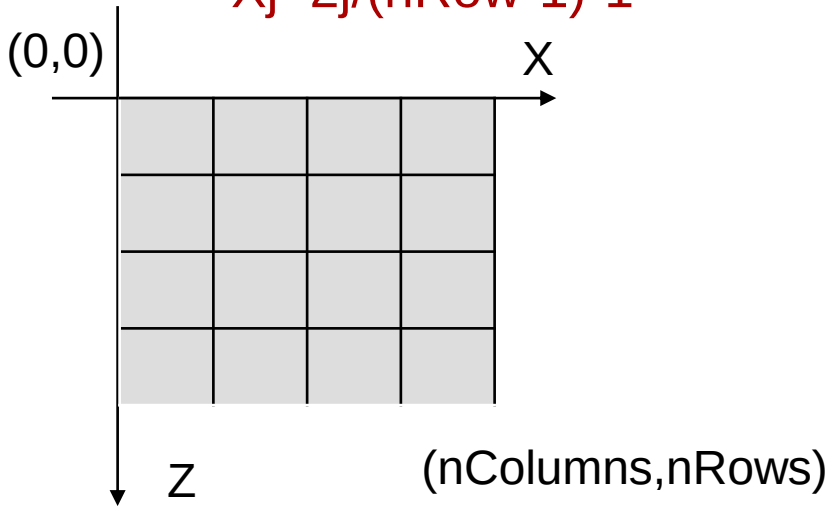
$$\frac{Yw-(-1)}{1-(-1)}=\frac{Ys-h}{0-h}$$ 得到 $Yw=-1+\frac{2(Ys-h)}{-h}=-1+\frac{2(h-Ys)}{h}=1-\frac{2Ys}{h}$

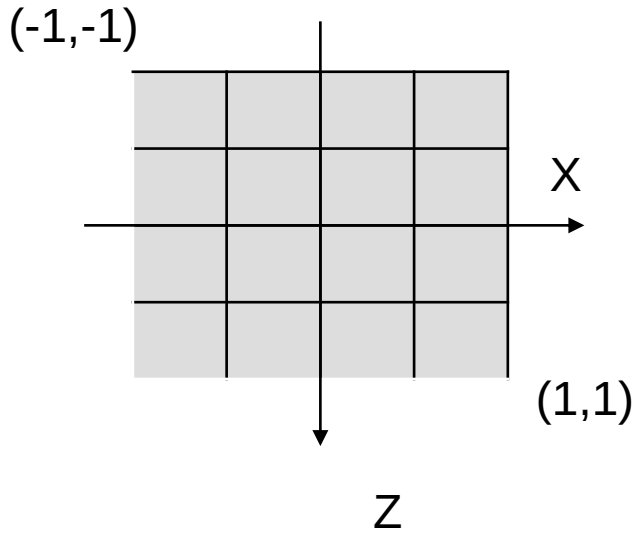
Window-Viewport Trans.(cont.)

- 窗区视区变换（继续）
  - 应用2：造型中初始化网格坐标

造型墨西哥帽子时，顶点坐标Z,X的计算

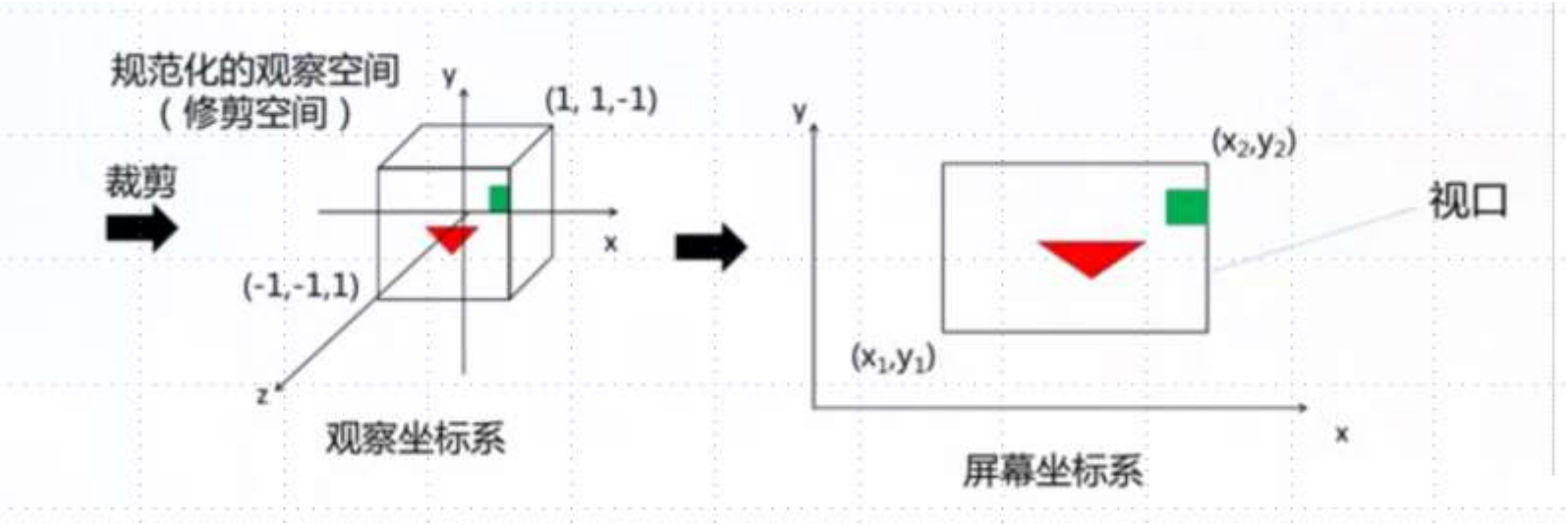
- (0,0),(nRow,nColomn)规约到(-1,-1)(1,1)正方形中
- 变换矩阵：T(-1,-1)*S(2/(nColumns-1), 2/(nRows-1)),
  - Zi=2*i/(nColumns-1)-1,
  - Xj=2j/(nRow-1)-1





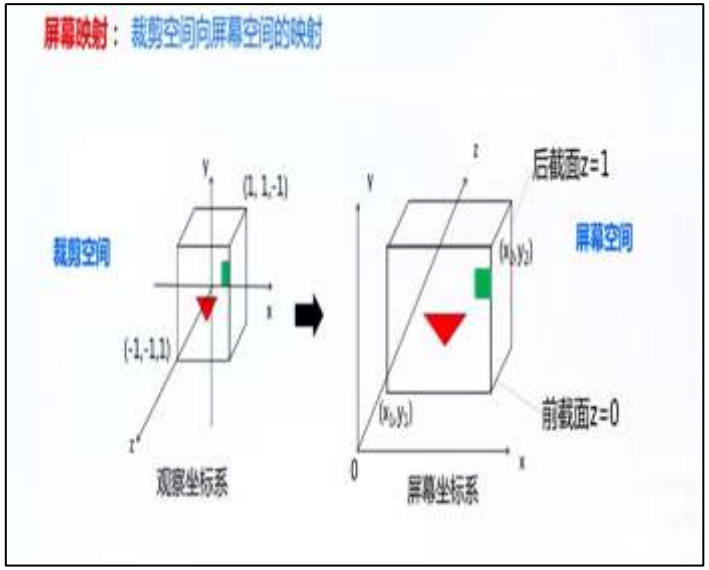
Window-Viewport Trans.(cont.)

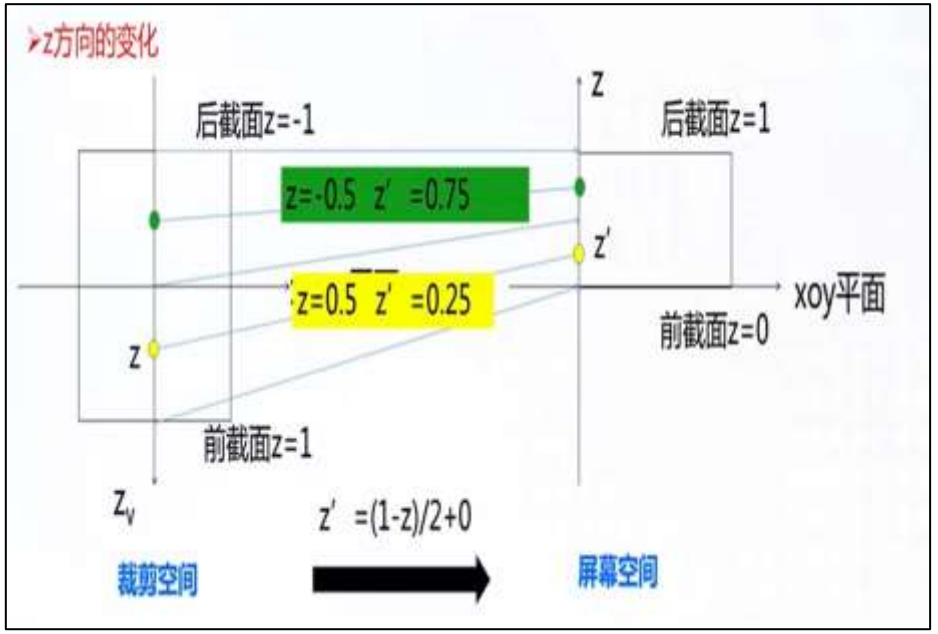
- 窗区视区变换（继续）
  - **应用3：屏幕映射，**由图形管线系统自动完成
  - “窗区”：在投影函数指定3D视见体
    gluOrtho(Xwmin,Xwmax,Ywmin,Ywmax,near,far)
  - “视区”：在绘制区域，指定的2D视口
    glViewport(Xvmin,Yvmin,Vwidth,Vheight);



Window-Viewport Trans.(cont.)

- 窗区视区变换（继续）
  - **应用3：屏幕映射 (续)**
    - OpenGL屏幕坐标是左手系！**Z方向变换Z=(1-Z)/2**





CC:右手坐标系　　DC:左手坐标系

Window-Viewport Trans.(cont.)

- 窗区视区变换（继续）

  - **应用3：屏幕映射 (续)**
    - webGL默认的裁剪坐标和屏幕坐标都采用左手坐标系

      参”webGL编程指南” 附录D

总结

总之，我们知道 WebGL 并不强制使用右手或左手坐标系。我们了解了，大部分 WebGL 库和 WebGL 程序都采用了传统的右手坐标系，本书也是这样做的。但是 WebGL 的默认行为（比如，在裁剪空间中使用左手坐标系）却与此冲突。为了解决这个冲突，我们可以通过翻转 z 坐标值进行补偿，这样就能够继续使用传统的右手坐标系了。但是，如前所述，这只是一个传统，只是大多数人遵守而已。如果你不了解 WebGL 的默认行为，以及处理它的过程，搞不好什么时候这个问题就会出来难为你了。

具体编程实现时，可以在MC,WC,VC下都统一采用右手坐标系进行计算，在顶点作色器程序最后进行Z反向处理即可。如果用系统提供的API函数（如投影函数）已经进行了左手处理，就不用进行反Z操作了（如angel框架中）

```
mat4 M_InverseZ = mat4(
            1.0,    0.0,    0.0,    0.0,
            0.0,    1.0,    0.0,    0.0,
            0.0,    0.0,    -1.0,   0.0,
            0.0,    0.0,    0.0,    1.0
            );
```

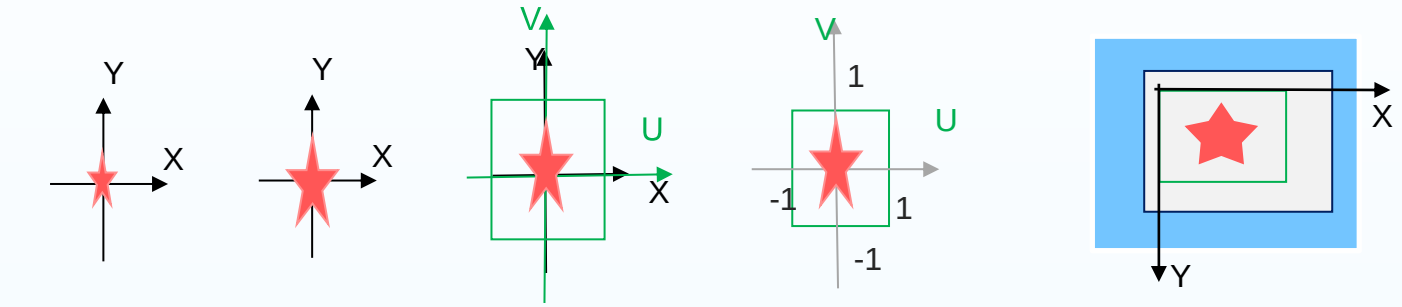
```
gl_Position = M_InverseZ*vPosition;
```

Summary(cont.)

- 规范化投影变换
  - Pcc= Mproject * Mview * Mmodel * Pmc
  - Mproject：就是“规范化投影变换矩阵”

    //注：如果CG API没有提供，需要自己推导和编写
    - 平行正投影的规范化变换矩阵
      - `Ortho(left,right,bottom,top,near,far)`
    - 一点透视的规范化投影变换矩阵
      - `perspective( fovy, aspect, near, far )`

Summary



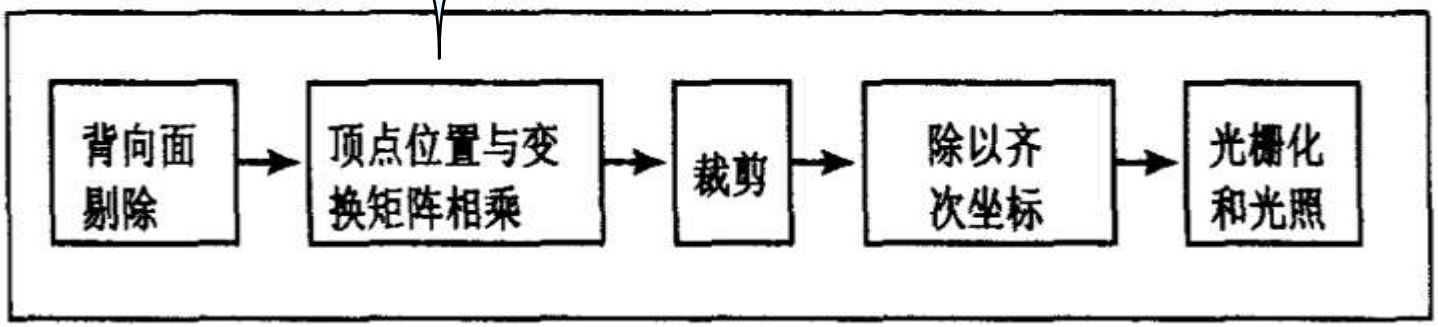
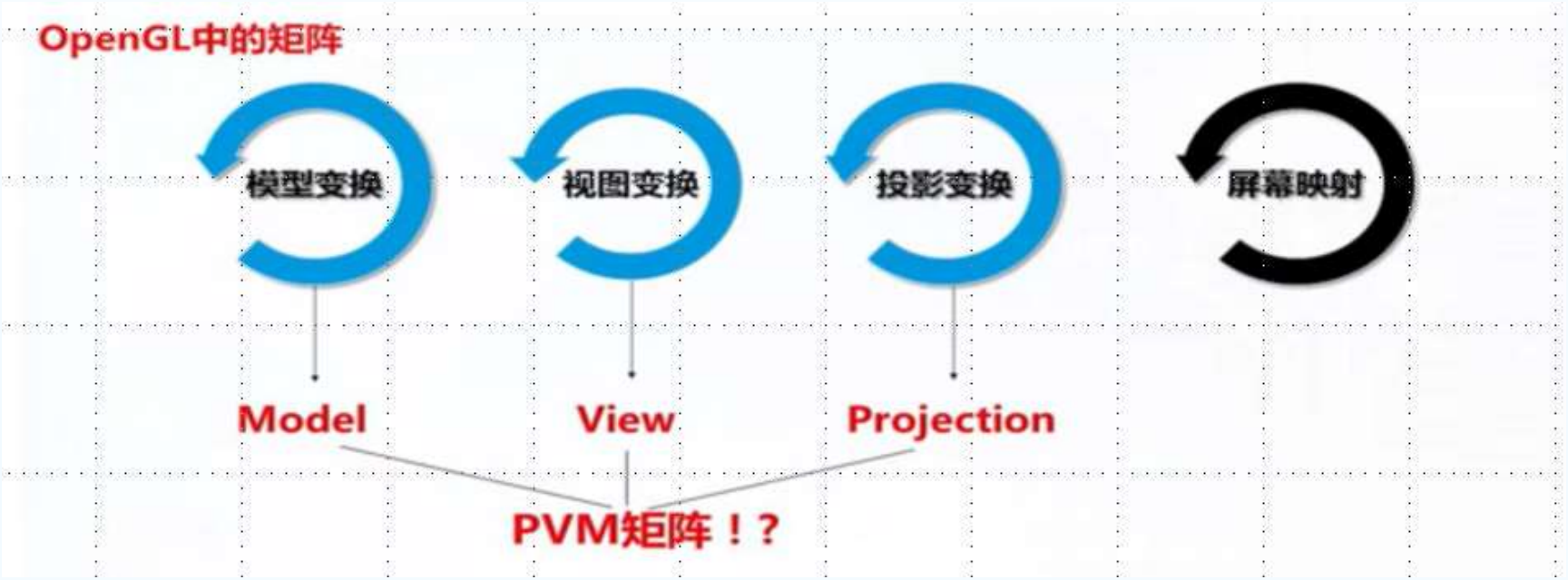


图12-6 一个完整的图形流水线。多边形从左边输入，逐次通过流水线的各个阶段

Summary(cont.)

- 顶点着色器中
  - 若采用列向量表示顶点，通常为P*V*M
  - P_{cc}= **Mproject * Mview * Mmodel * P_{mc}**



Summary(cont.)

1. **规范化投影变换：**是“可逆”的非仿射变换
   - 经典透视变换理论上讲是不可逆的（降维），因为一条投影线上的所有的点都会投影到同一个点（即不能从一个点的投影中恢复出这个点)。
   - 而计算机投影采用透视变换的可逆形式，即没有从3D降维到2D，而采用保留了深度值（Z值）以便作后续的处理。
2. 透视除法：由系统自动完成的操作
   - 图形顶点经过规范化投影变换后得到的齐次坐标，需要进一步做透视除法，才能得到其真正的笛卡尔坐标。
   - 透视除法在管线中由系统自动实现（顶点着色器只要计算出规范化视景体中的顶点的齐次坐标即可，不用做透视除法）

Summary(cont.)

- 作业：