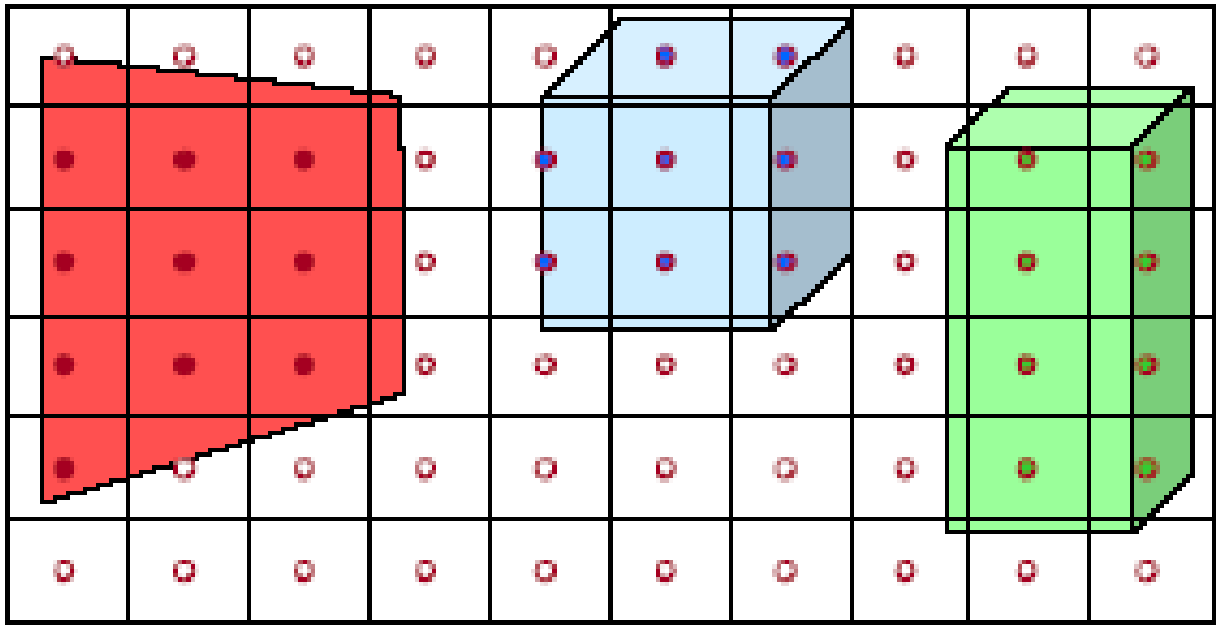
Rendering Pipeline

3D Polygon Rendering

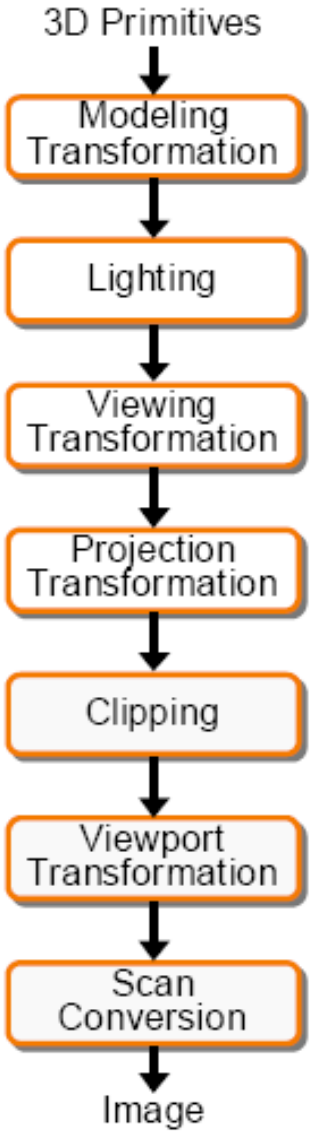
- **Many applications use rendering of 3D polygons with direct illumination**

3D Polygon Rendering

- **What steps are necessary to utilize spatial coherence while drawing these polygons into a 2D image?**



3D Rendering Pipeline (for direct illumination)



Transform into 3D world coordinate system

Illuminate according to lighting and reflection

Transform into 3D camera coordinate system

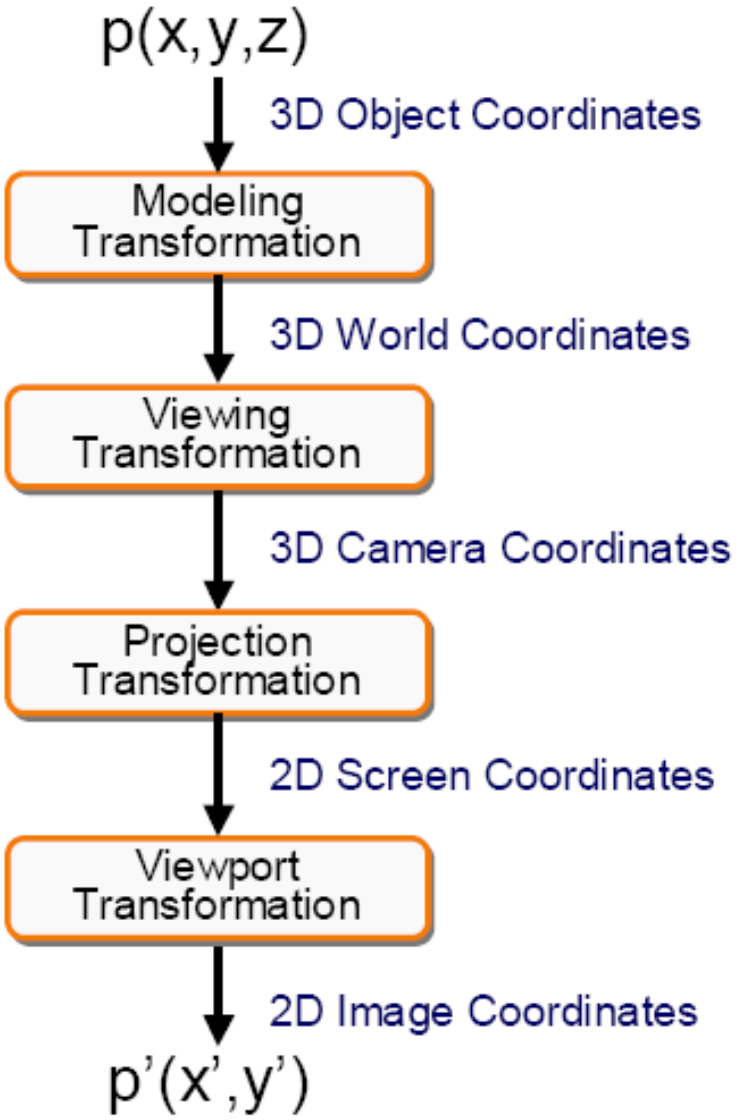
Transform into 2D camera coordinate system

Clip primitives outside camera's view

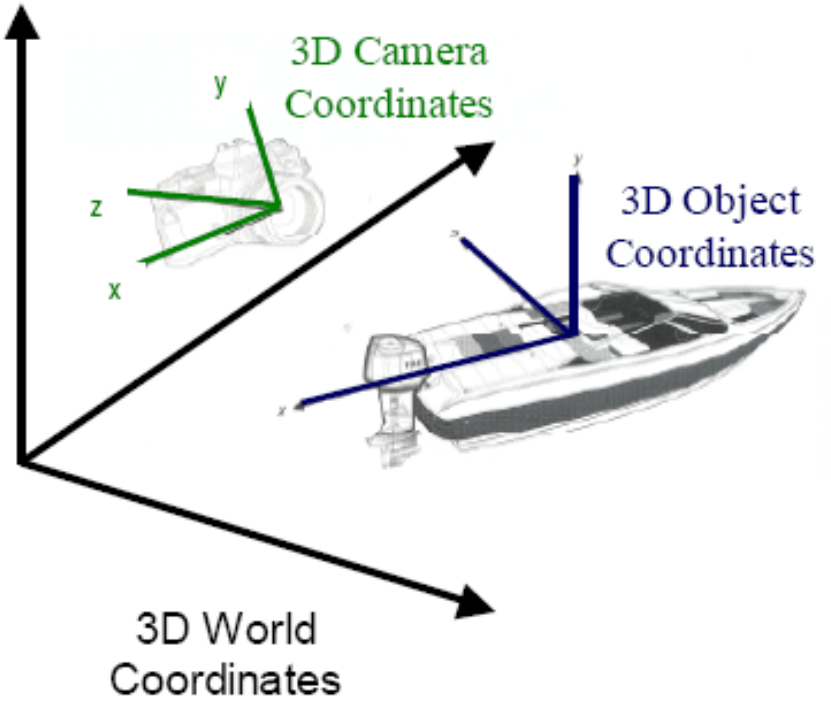
Transform into image coordinate system

Draw pixels (includes texturing, hidden surface, …)

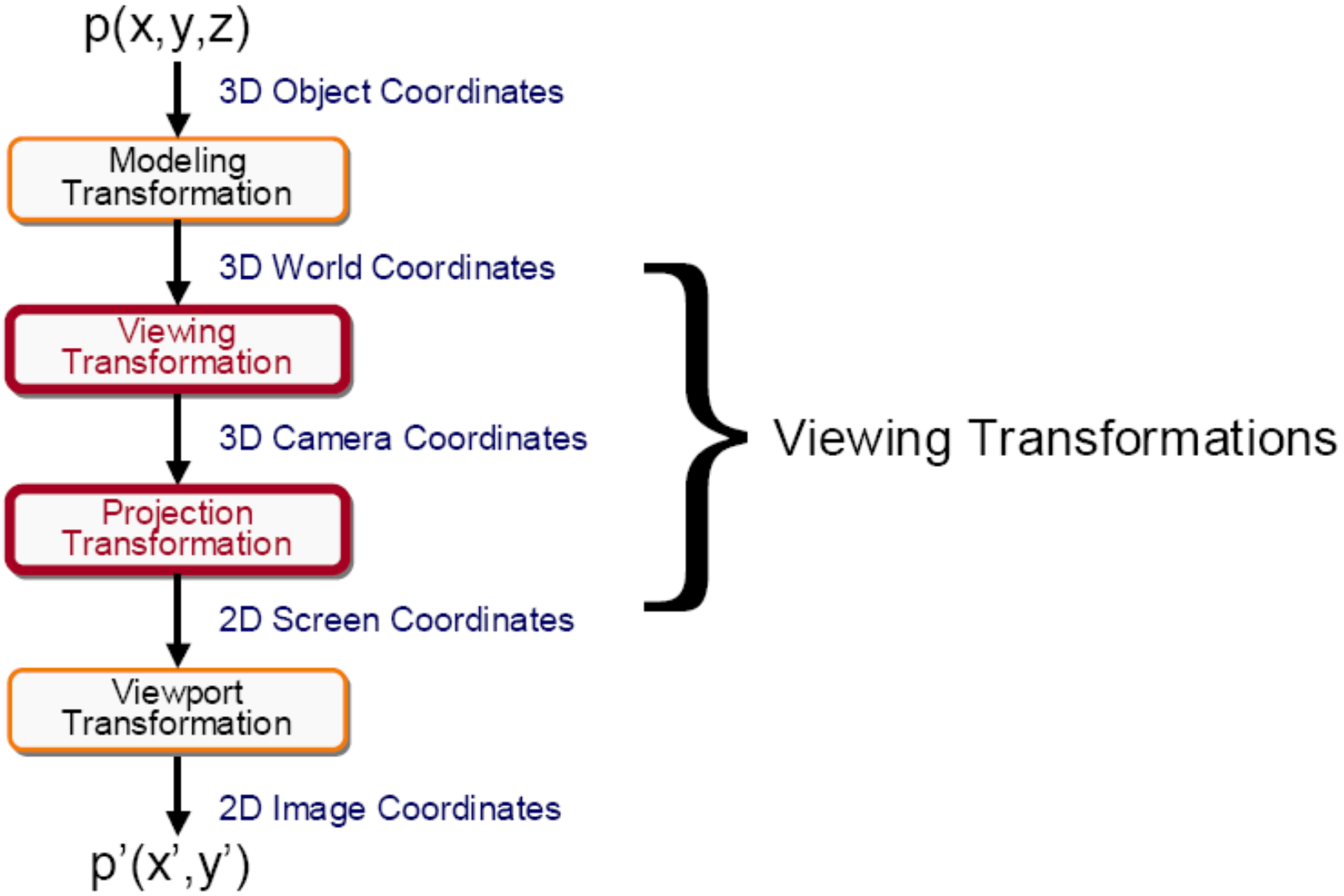
Transformations



Transformations map points from one coordinate system to another

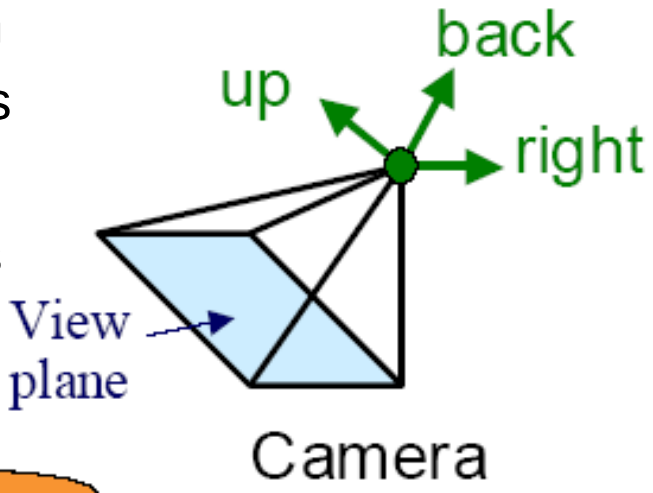


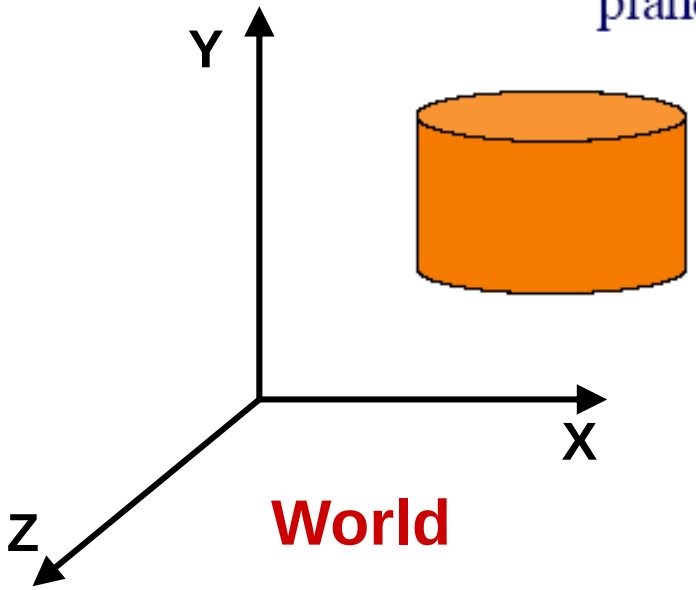
Viewing Transformations



Viewing Transformation

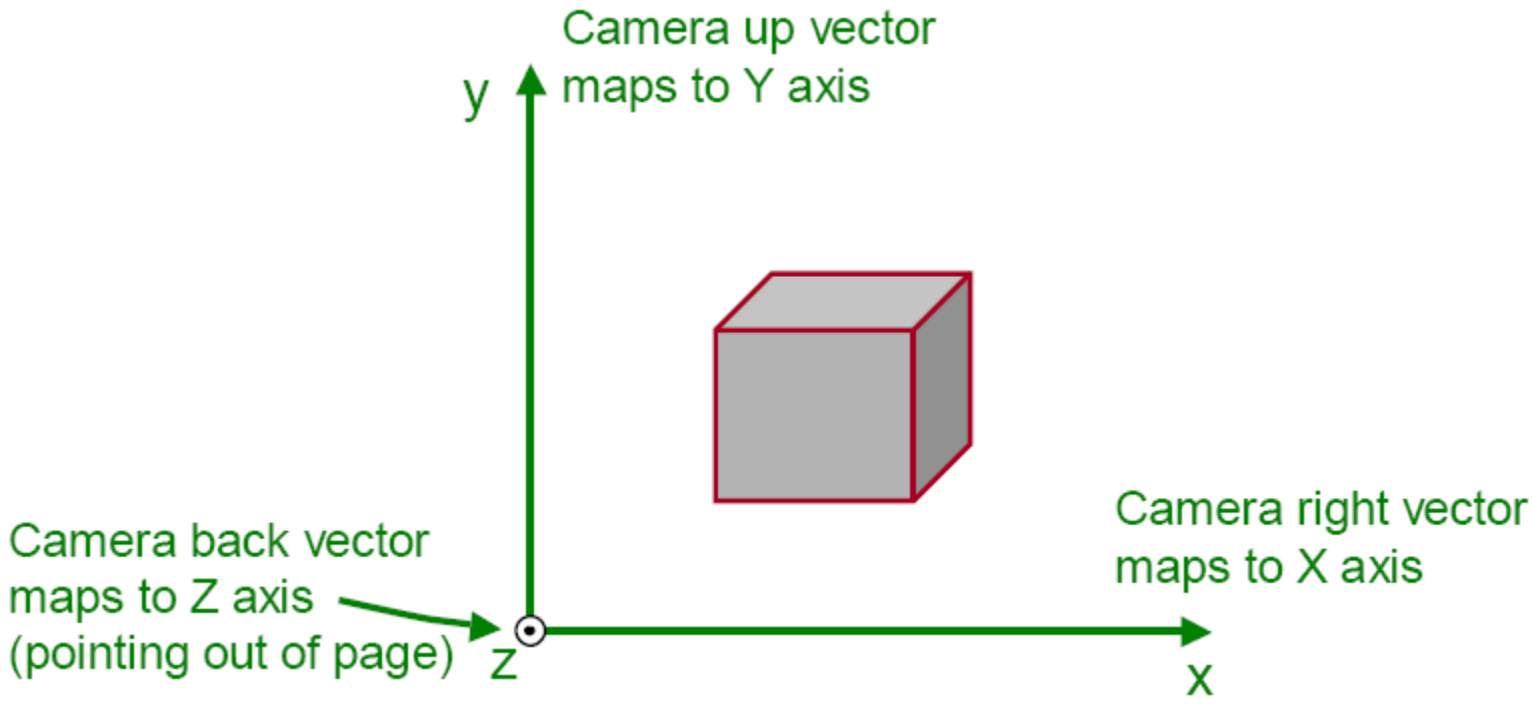
- **Mapping from world to camera coordinates**
  - Eye position maps to origin
  - Right vector maps to X axis
  - Up vector maps to Y axis
  - Back vector maps to Z axis





Camera Coordinates

- **Canonical coordinate system**
  - Convention is **right-handed** (looking down –z axis)
  - Convenient for projection, clipping, etc .



Finding the viewing transformation

- We have the camera (in world coordinates)
- We want T taking objects from world to camera

$$\mathbf{p}^c = \mathbf{T}\mathbf{p}^w$$

- Trick: find T^{-1} taking objects in camera to world

$$\mathbf{p}^w = \mathbf{T}^{-1}\mathbf{p}^c$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Finding the viewing transformation

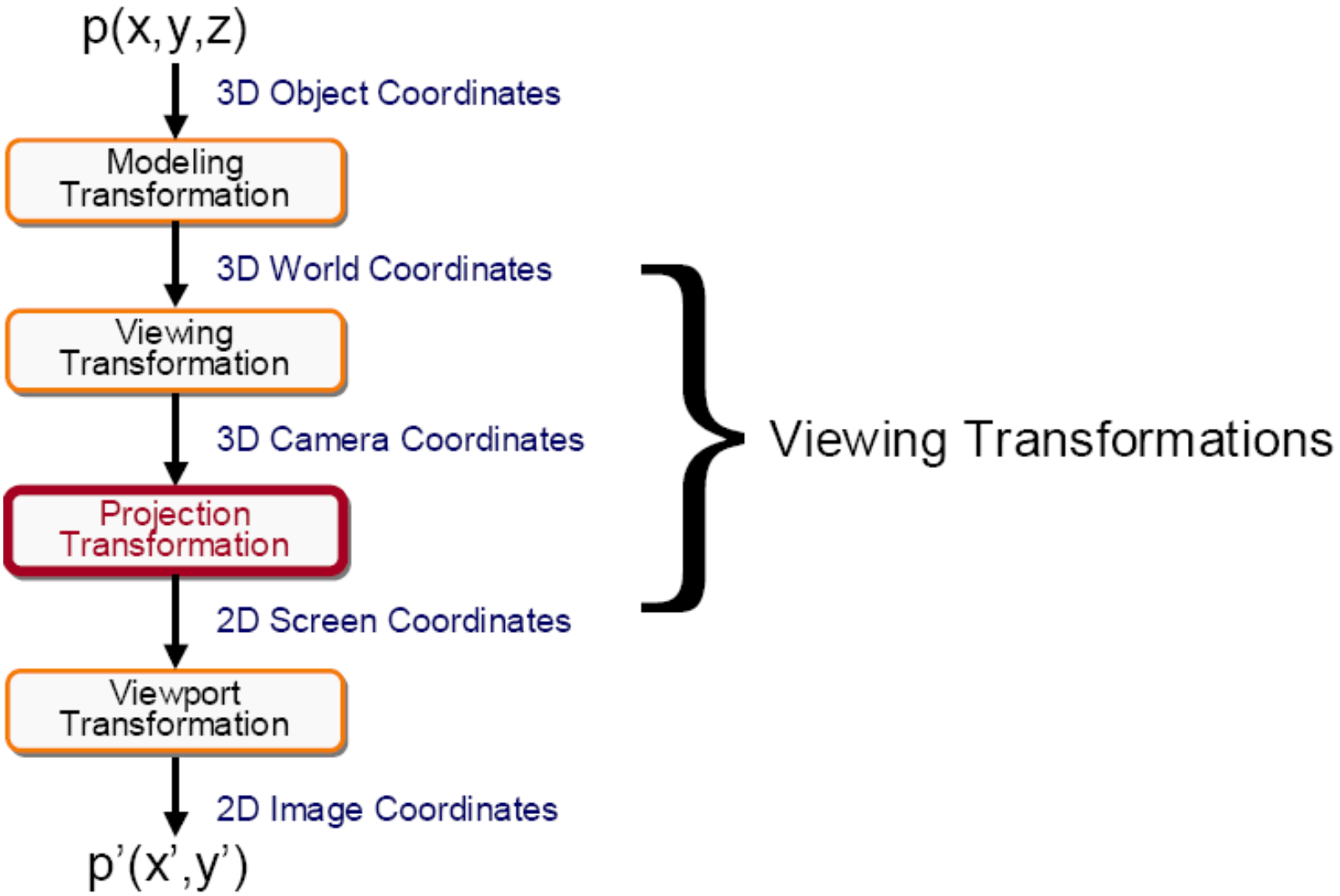
- **Trick: map from camera coordinate to world**
  - Origin maps to eye position
  - Z axis maps to Back vector
  - Y axis maps to Up vector
  - X axis maps to Right vector

$$\mathbf{p}^w = \mathbf{T}^{-1}\mathbf{p}^c$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} R_x & U_x & B_x & E_x \\ R_y & U_y & B_y & E_y \\ R_z & U_z & B_z & E_z \\ R_w & U_w & B_w & E_w \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

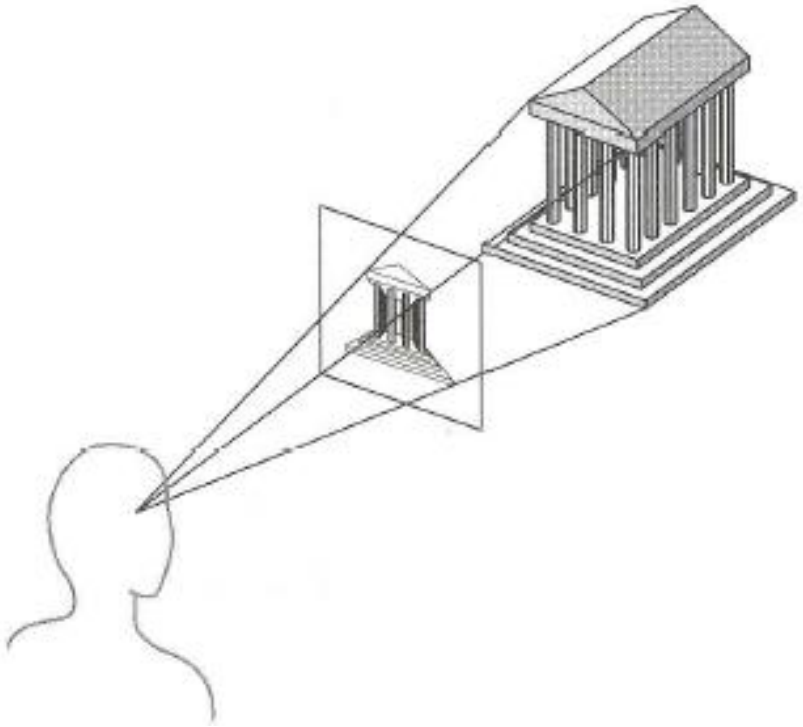
- **This matrix is T^{-1} so we invert in to get T (easy)**

Viewing Transformations

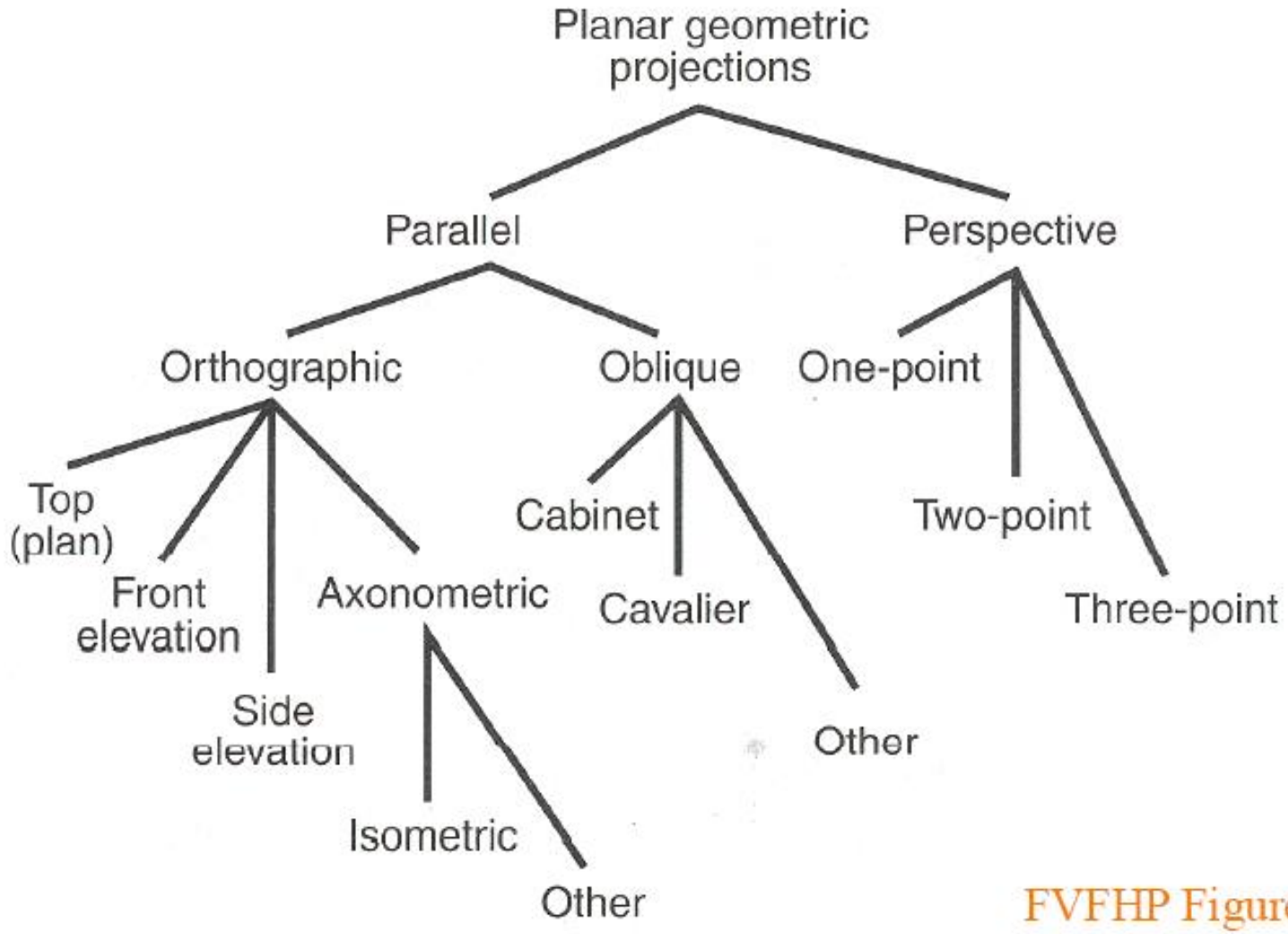


Projection

- **General definition:**
  - Transform points in n-space to m-space (m<n)
- **In computer Graphics:**
  - Map 3D camera coordinates to 2D screen coordinates



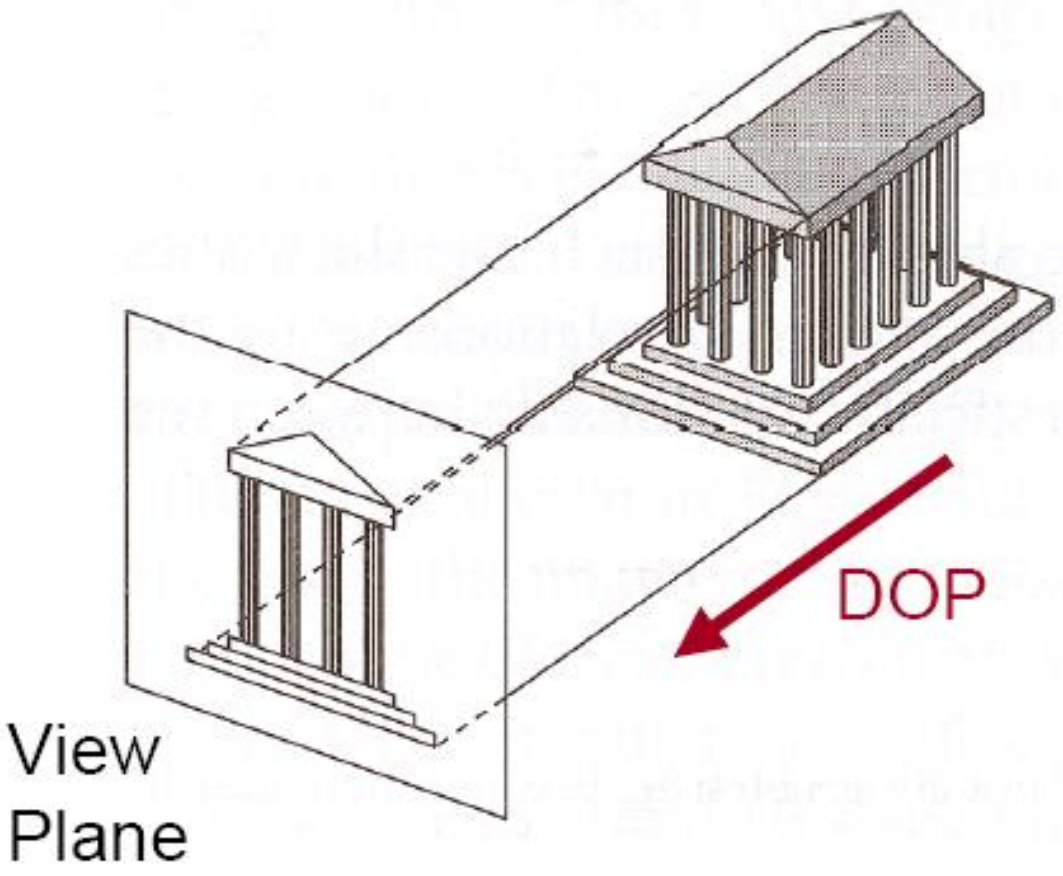
Taxonomy of Projections



FVFHP Figure 6.10

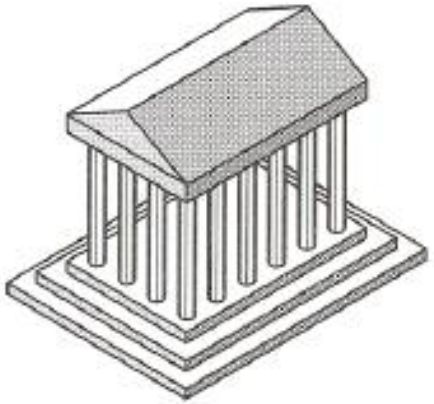
Parallel Projection

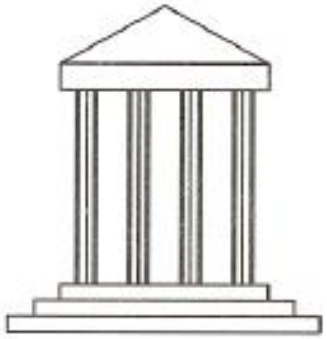
- **Center of projections is at infinity**
  - Direction of projection (DOP) same for all points



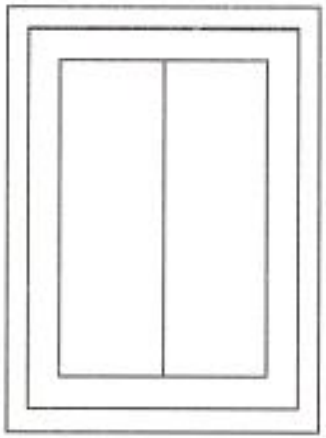
Orthogonal Projections

- **DOP perpendicular to view plane**

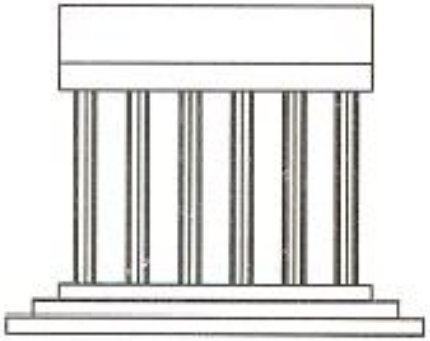




Front



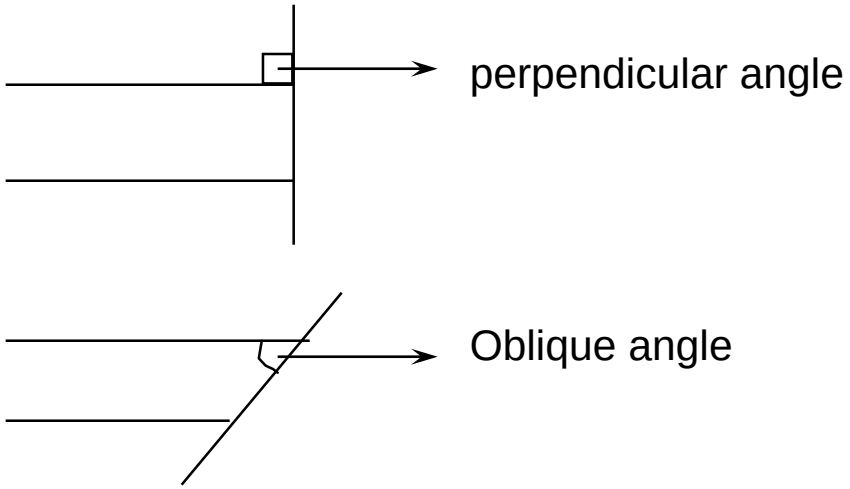
Top

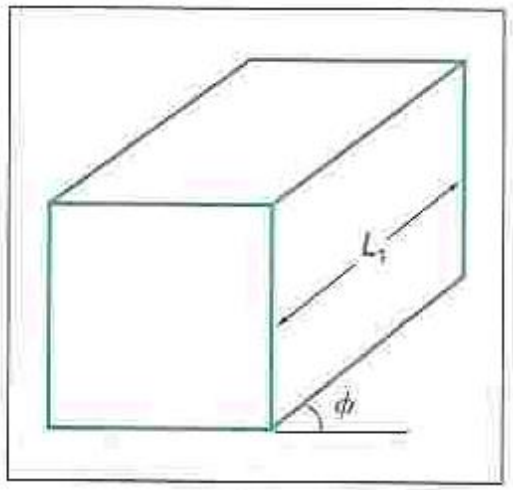


Side

Oblique Projections

- **DOP not perpendicular to view plane**
  1) Cavalier projection
     - when angle between projector and view plane is 45 degree
     - foreshortening ratio for all three principal axis are equal.
  2) Cabinet projection
     - the foreshortening ratio for edges that perpendicular to the plane of projection is one-half (angle 63.43 degree)

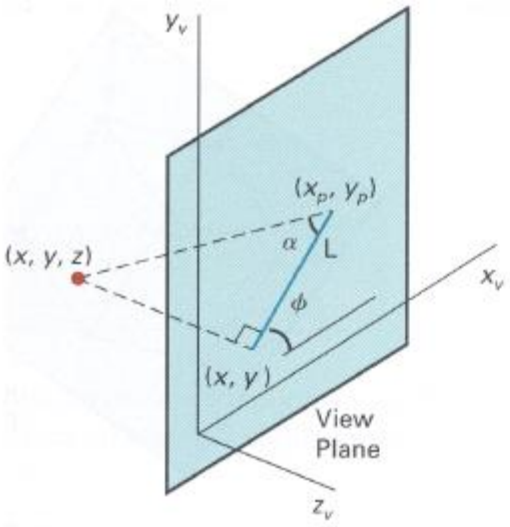


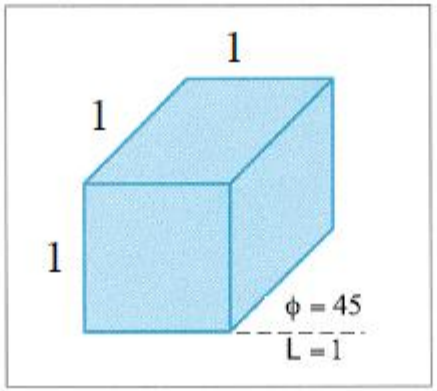


(b)
Projection on the
Viewing Plane

Oblique Projections

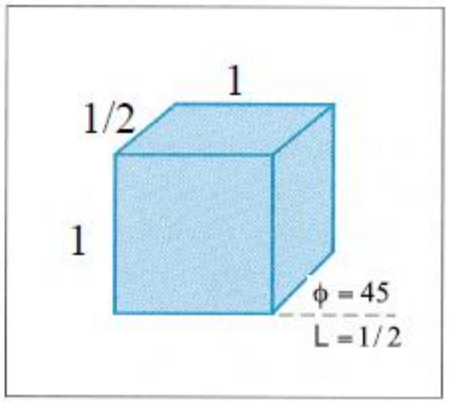
- **DOP not perpendicular to view plane**





Cavalier

(DOP $\alpha = 45^{\circ}$)

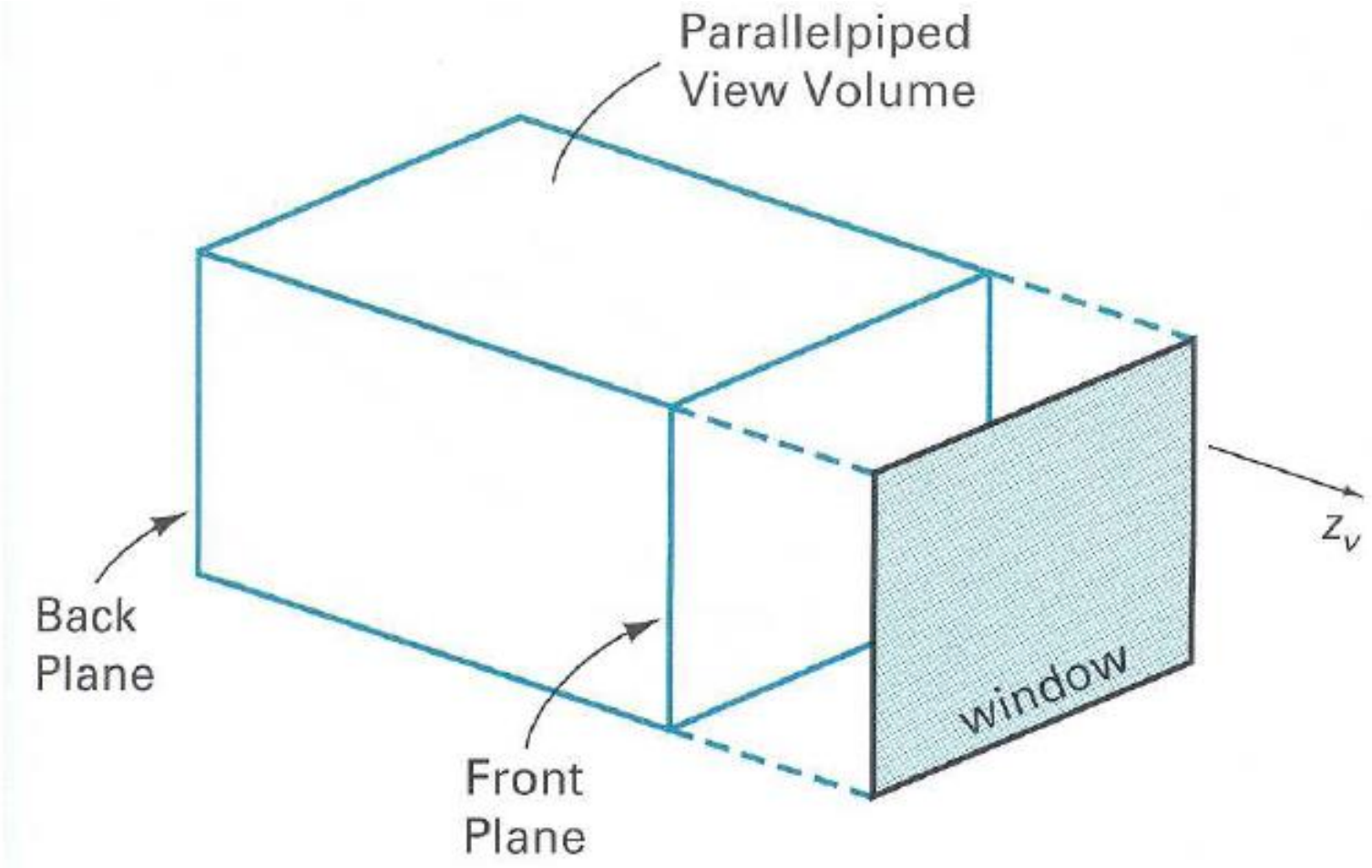


Cabinet

(DOP $\alpha = 63.4^{\circ}$)

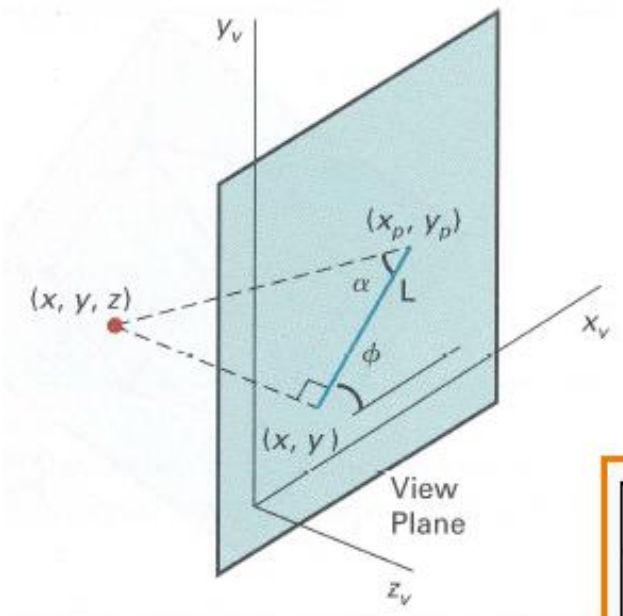
- φ describes the angle of the projection of the view plane's normal
- L represents the scale factor applied to the view plane's normal

Parallel Projection View Volume



Parallel Projection Matrix

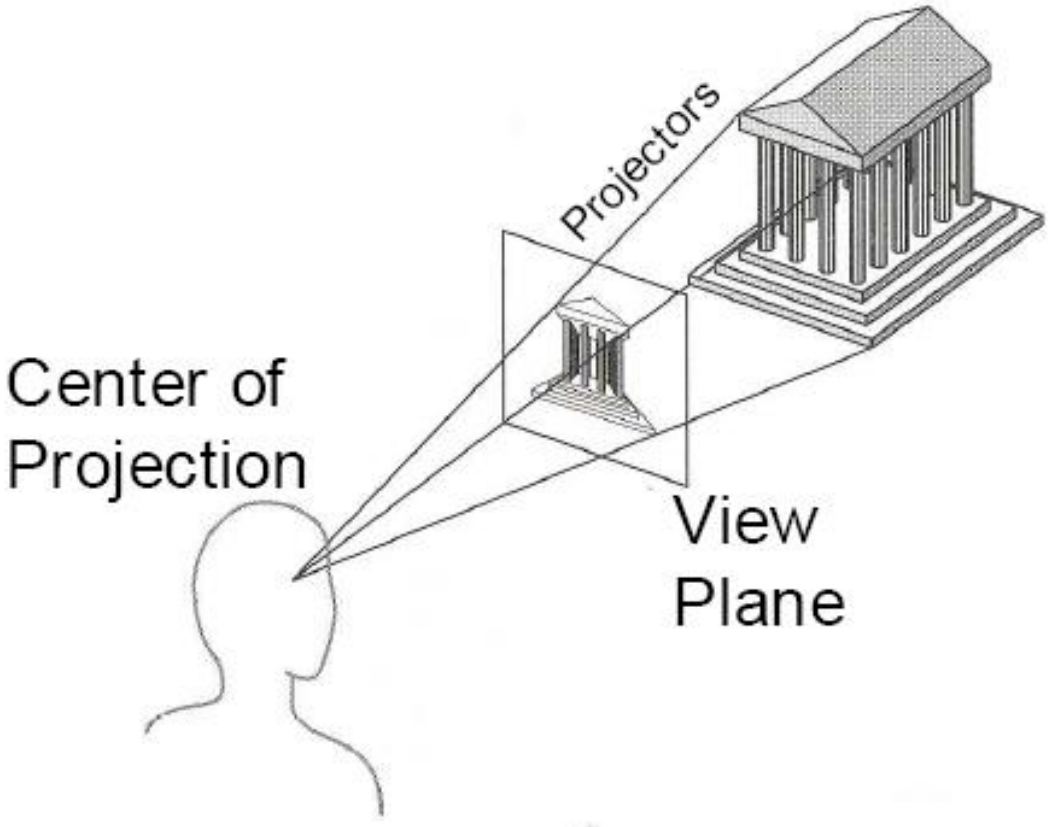
- **General parallel projection transformation**



$$\begin{bmatrix} x_s \\ y_s \\ z_s \\ w_s \end{bmatrix} = \begin{bmatrix} 1 & 0 & L_1 \cos\phi & 0 \\ 0 & 1 & L_1 \sin\phi & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix}$$

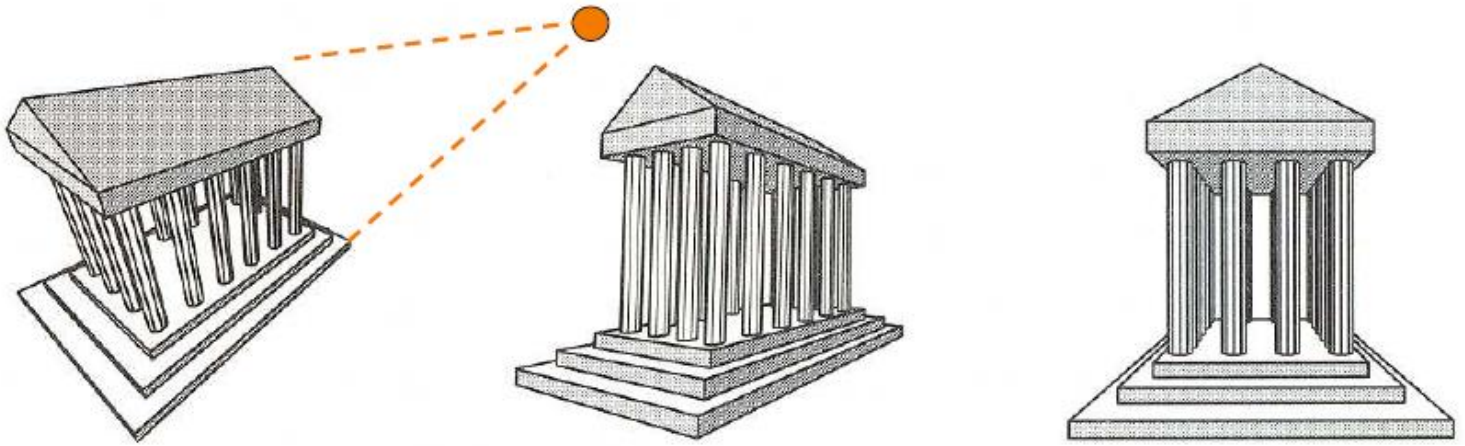
Perspective Projection

- **Map points onto “view plane” along “projectors” emanating from “center of projection” (COP)**



Perspective Projection

- **How many vanishing points?**

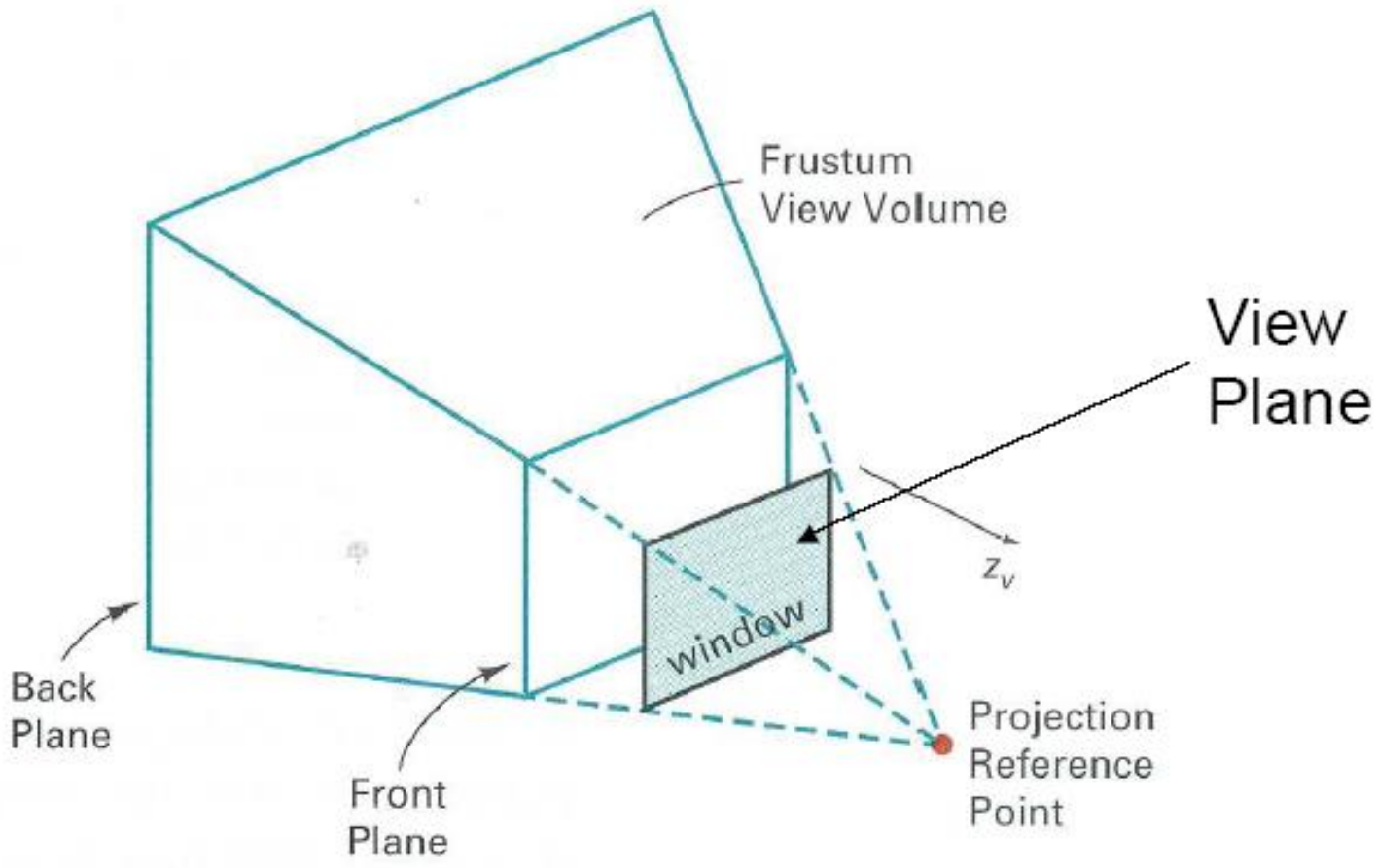


3-Point Perspective

2-Point Perspective

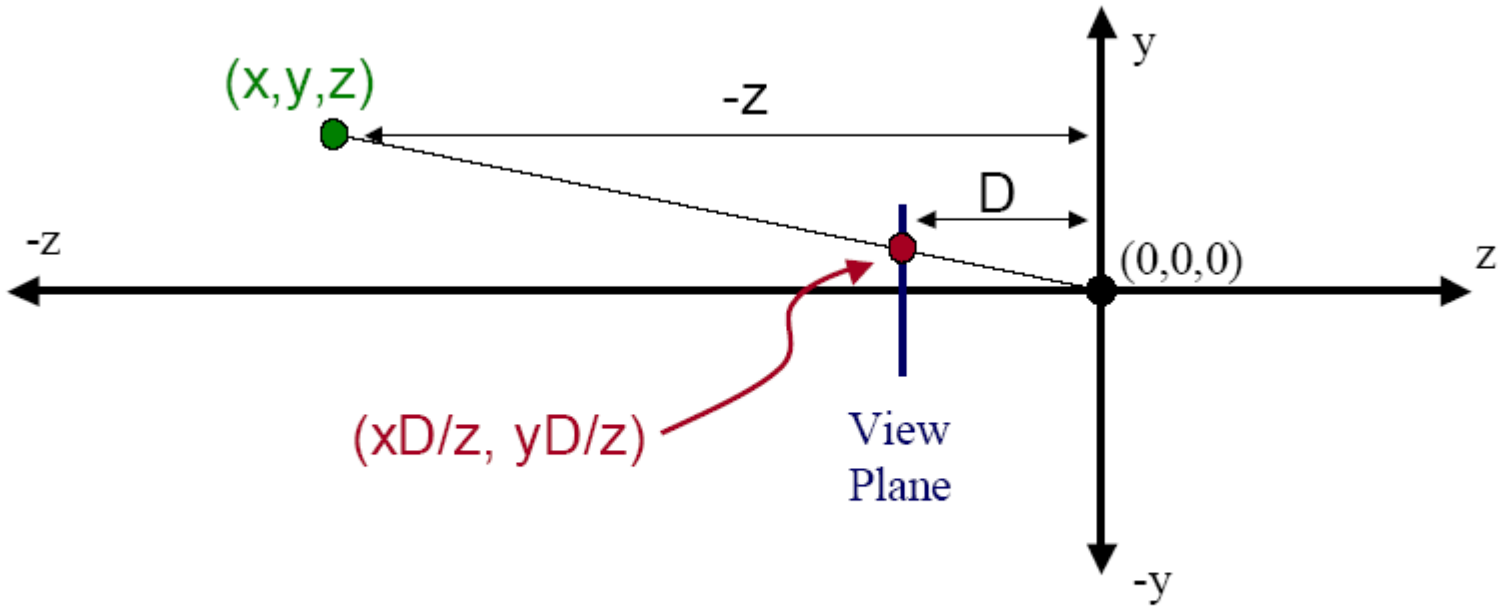
1-Point Perspective

Perspective Projection View Volume



Perspective Projection

- **Compute 2D coordinates from 3D coordinates with similar triangles**



Perspective Projection Matrix

- **4 x 4 matrix representation?**

$$x_s = x_c \, D / z_c$$

$$y_s = y_c \, D / z_c$$

$$z_s = D$$

$$w_s = 1$$

$$\begin{bmatrix} x_s \\ y_s \\ z_s \\ w_s \end{bmatrix} = \begin{bmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix}$$

Perspective Projection Matrix

- **4 x 4 matrix representation?**

$$x_s = x_c \, D/z_c$$

$$y_s = y_c \, D/z_c$$

$$z_s = D$$

$$w_s = 1$$

$$x' = x_c$$

$$y' = y_c$$

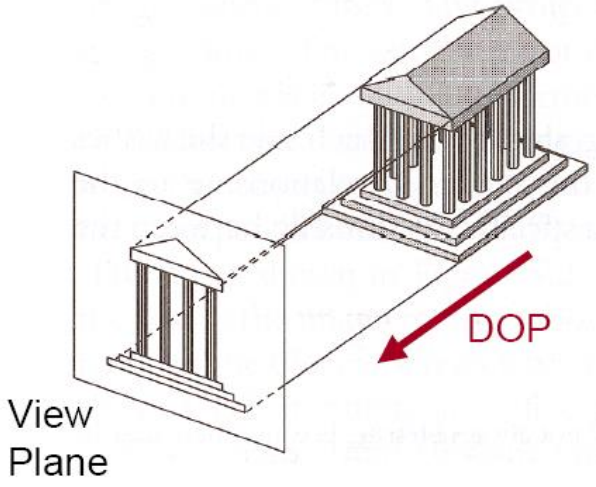
$$z' = z_c$$

$$w' = z_c/D$$

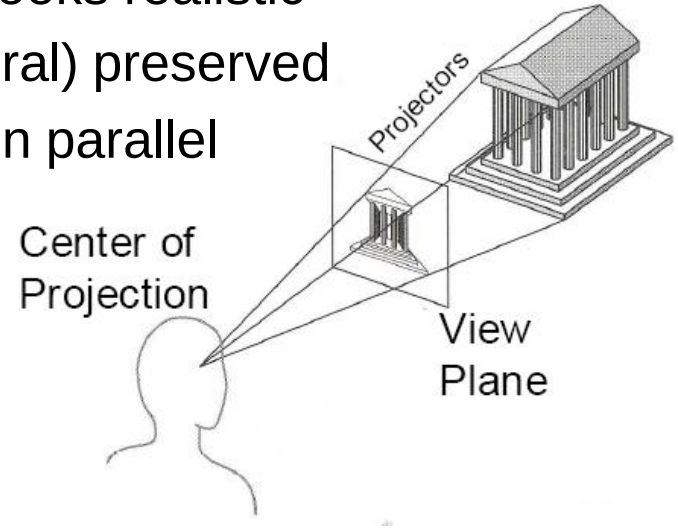
$$\begin{bmatrix} x_s \\ y_s \\ z_s \\ w_s \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/D & 0 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix}$$

Perspective vs. Parallel

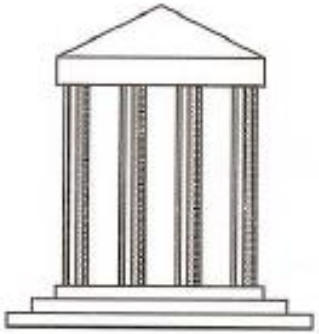
- **Parallel Projection**
  - + Good for exact measurements
  - + Parallel lines remain parallel
  - - Angles are not (in general) preserved
  - - Less realistic looking



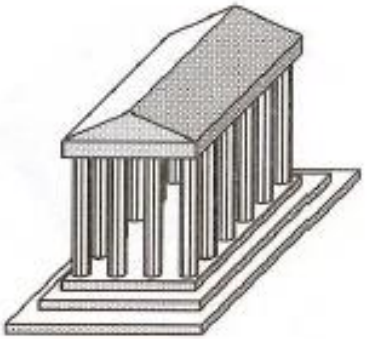
- **Perspective Projection**
  - + Size varies inversely with distance – looks realistic
  - - Distance and angles are not (in general) preserved
  - - Parallel line do not (in general) remain parallel



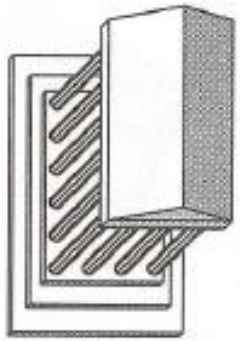
Classical Projections



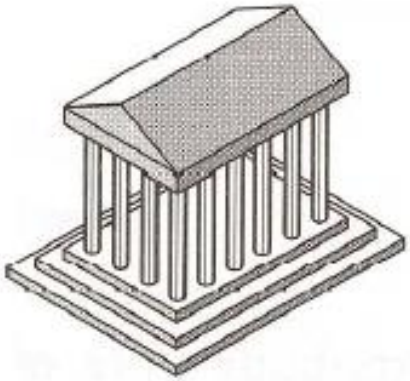
Front elevation



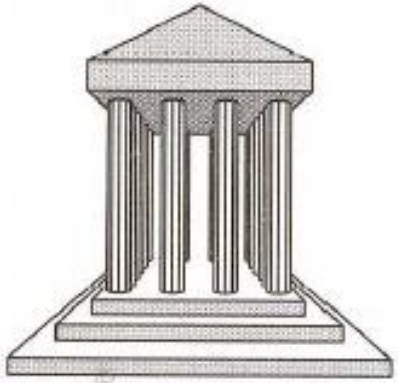
Elevation oblique



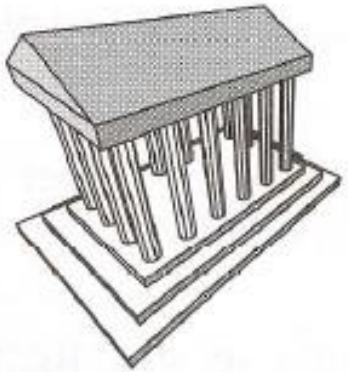
Plan oblique



Isometric

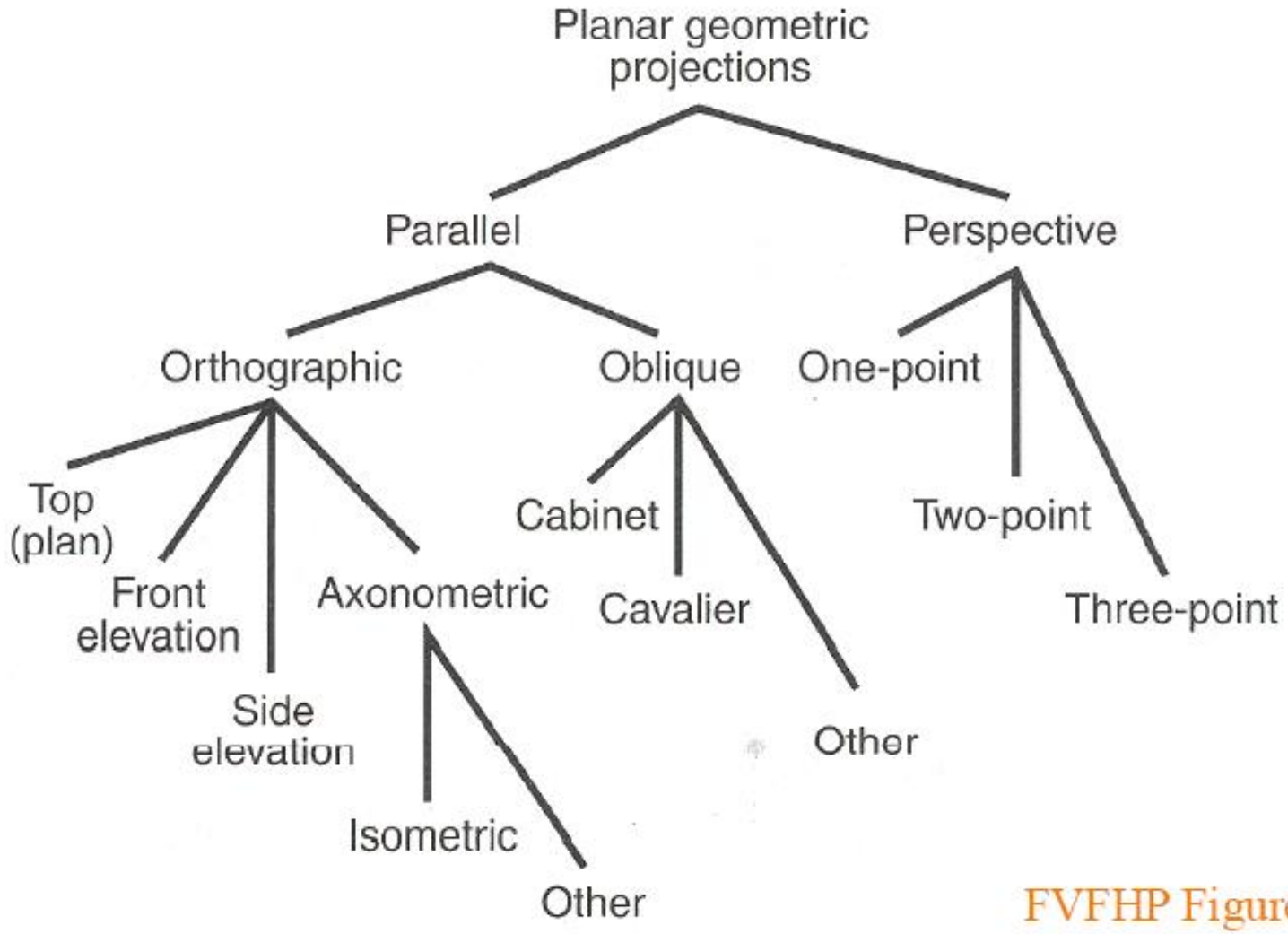


One-point perspective



Three-point perspective

Taxonomy of Projections

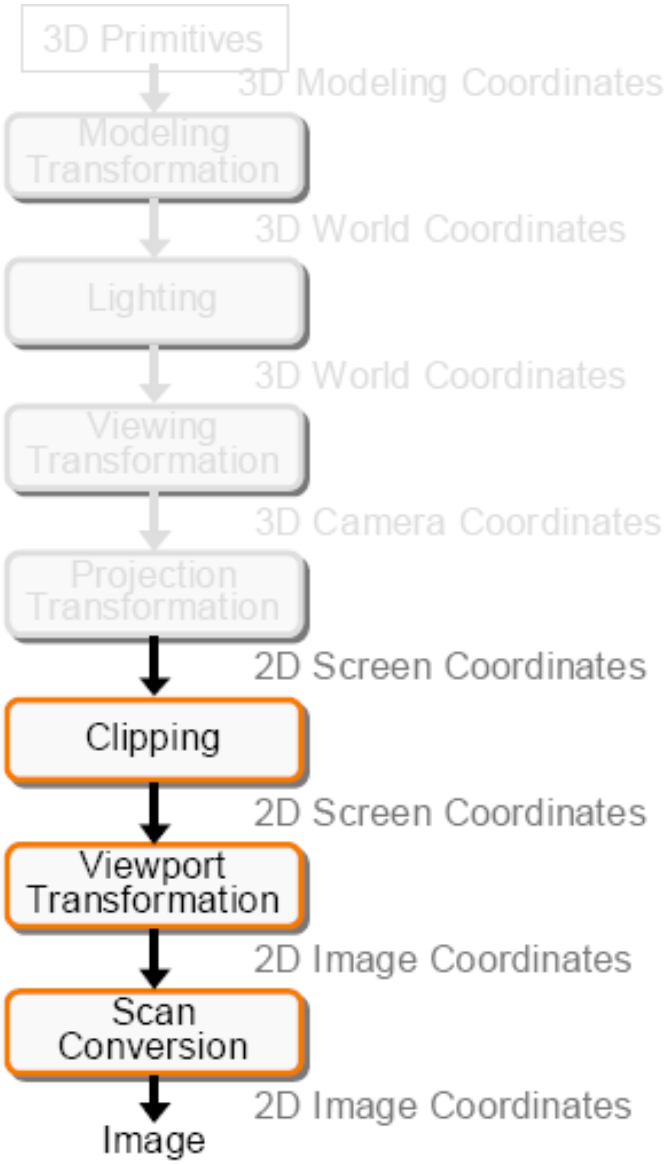


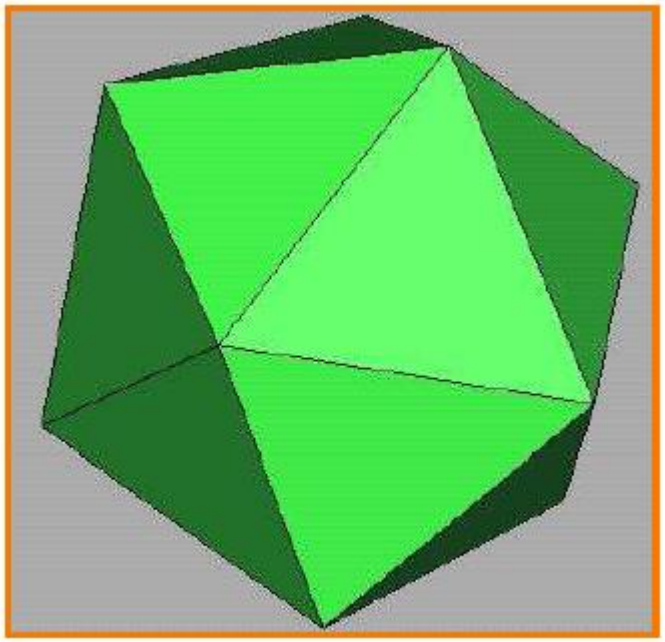
FVFHP Figure 6.10

Viewing Transformations Summary

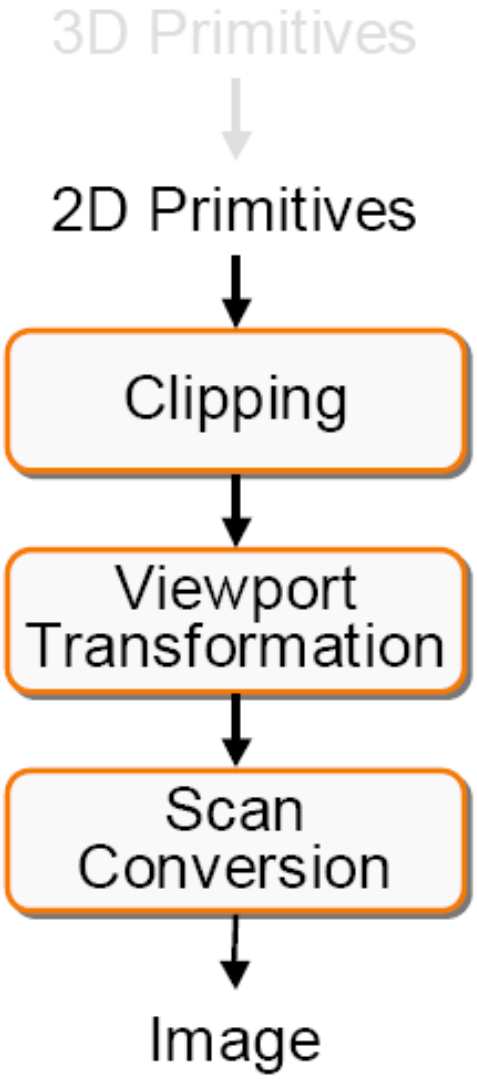
- **Camera Transformation**
  - Map 3D world coordinates to 3D camera coordinates
  - Matrix has camera vectors are rows
- **Projection Transformation**
  - Map 3D camera coordinates to 2D screen coordinates
  - Two type of projections
    - Parallel
    - Perspective

3D Rendering Pipeline (for direct illumination)





2D Rendering Pipeline



Clip portions of geometric primitives residing outside the window

Transform the clipped primitives From screen to image coordinates

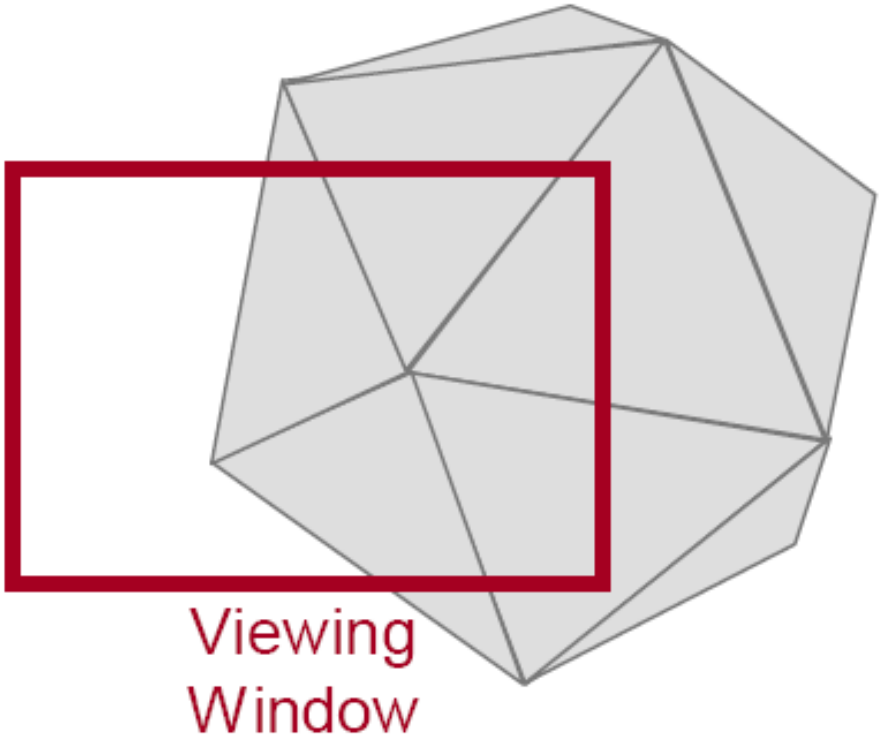
Fill pixels representing primitives In screen coordinates

Clipping

- **Avoid drawing parts of primitives outside window**
  - Window defines part of scene being viewed
  - Must draw geometric primitives only inside window

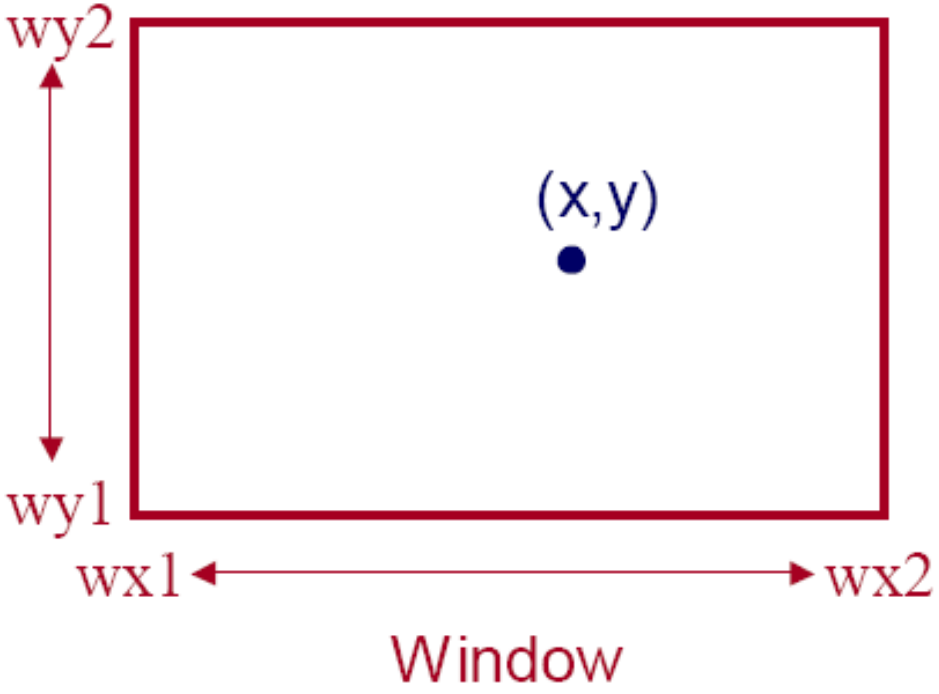
Clipping

- **Avoid drawing parts of primitives outside window**
  - Points
  - Lines
  - Polygons
  - Circles
  - etc.



Point Clipping

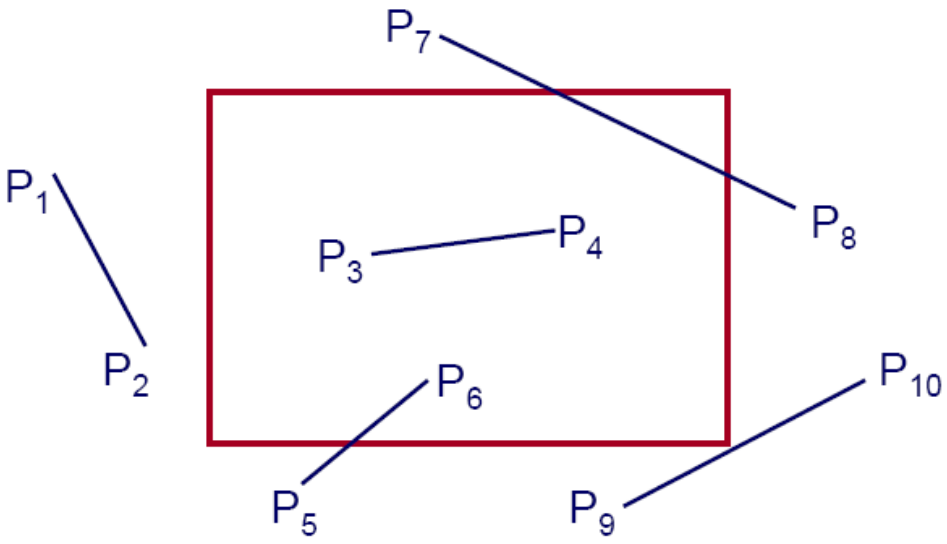
- Is point (x, y) inside window?



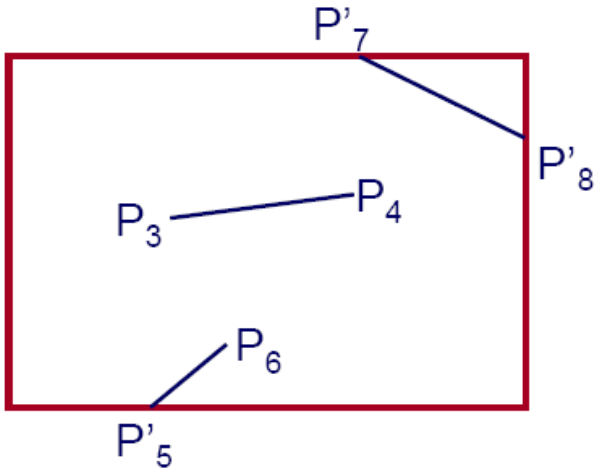

```
inside =
   (x >= wx1)  &&
   (x <= wx2)  &&
   (y >= wy1)  &&
   (y <= wy2);
```

Line Clipping

- **Find the part of a line inside the clip window**



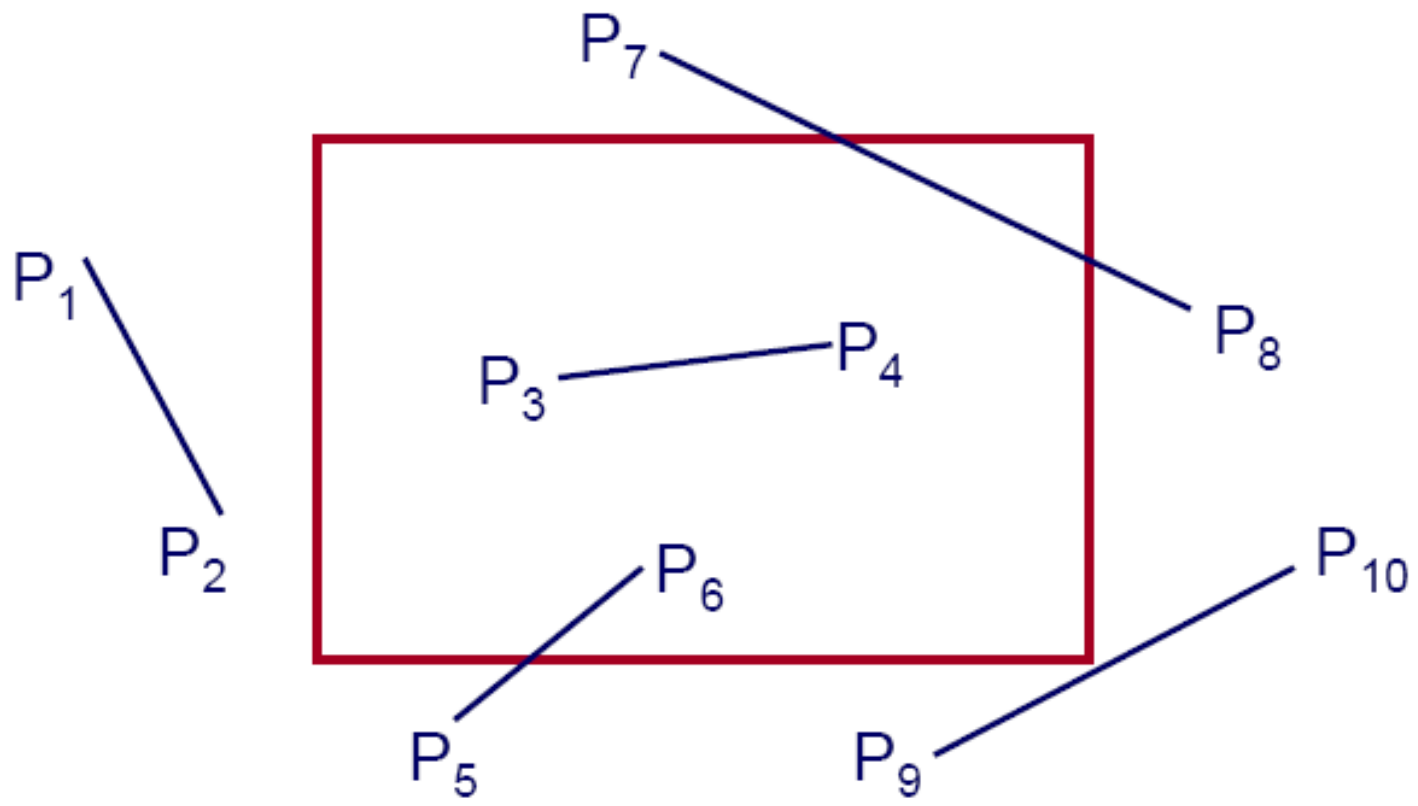
Before Clipping



After Clipping

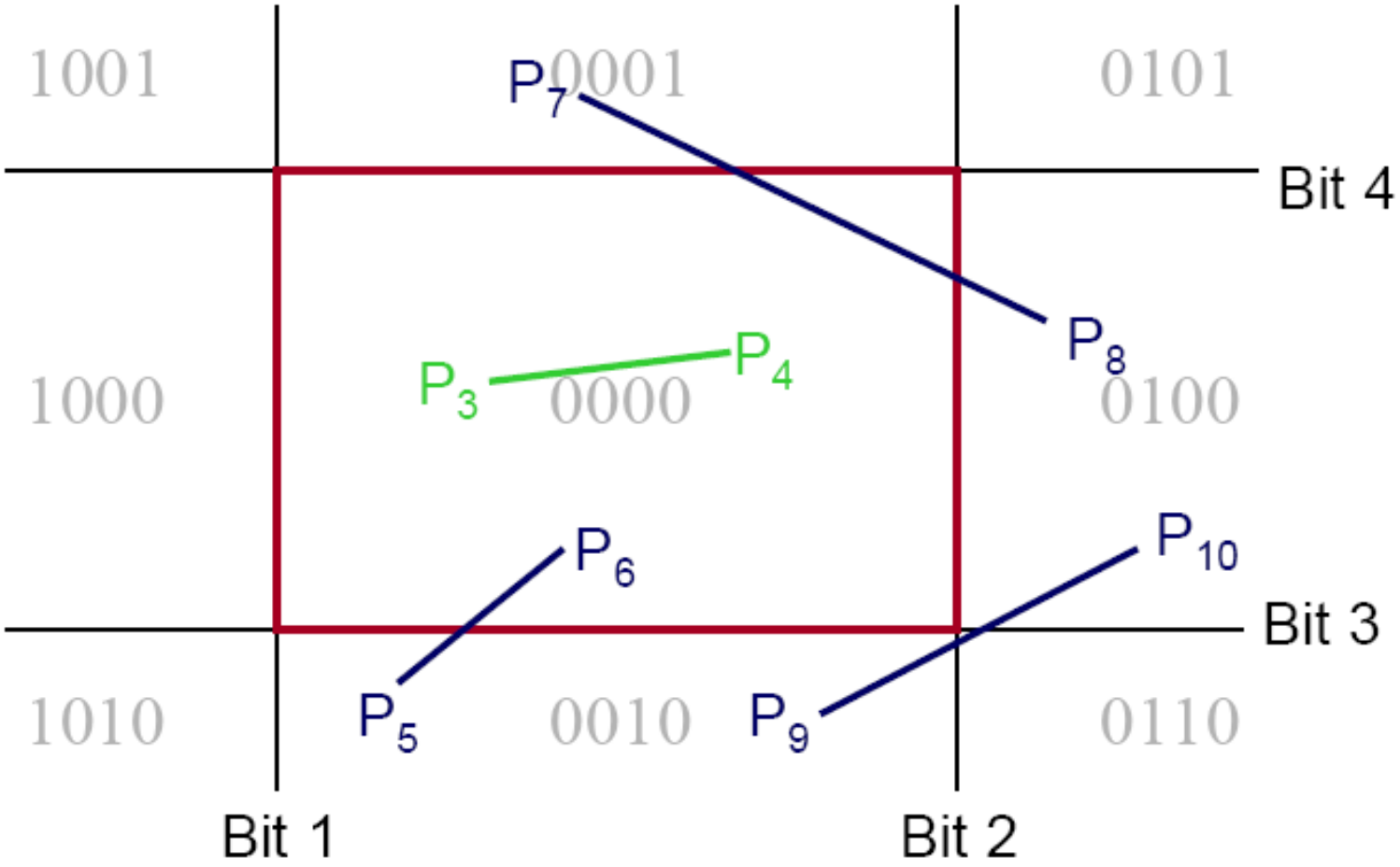
Cohen Sutherland Line Clipping

- **Use simple tests to classify easy cases first**



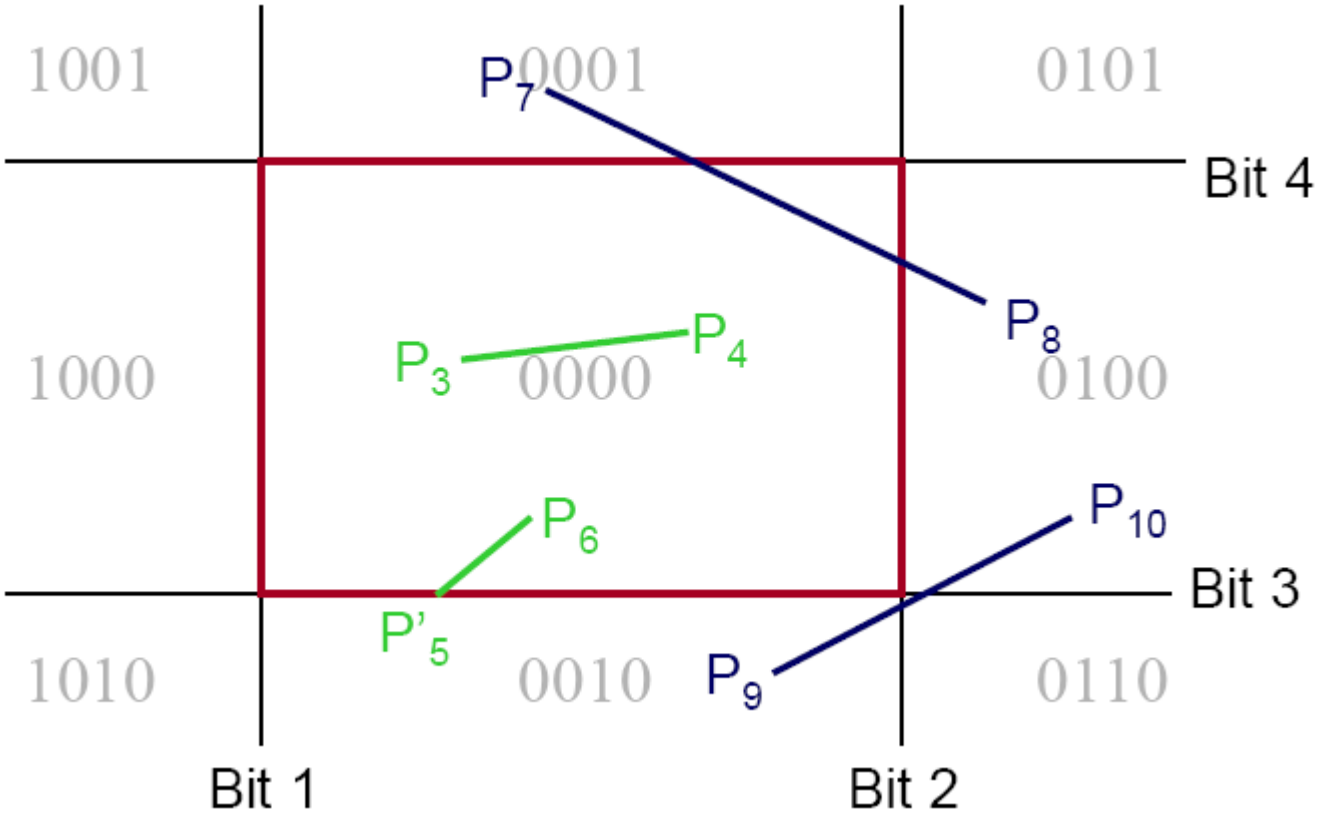
Cohen Sutherland Line Clipping

- **Classify some lines quickly AND of bit codes representing regions of two end points (must be 0)**



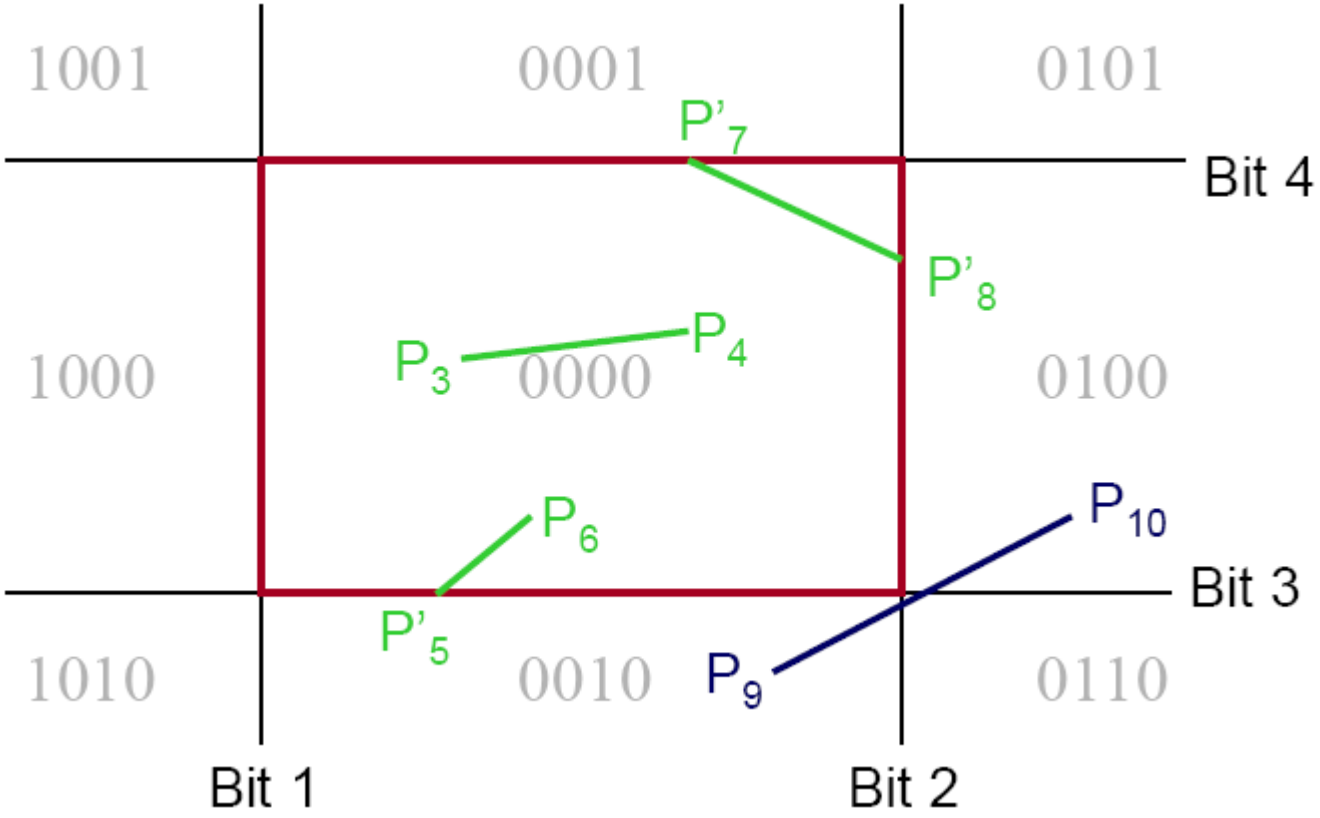
Cohen Sutherland Line Clipping

- **Compute intersections with window boundary for lines that can't be classified quickly**



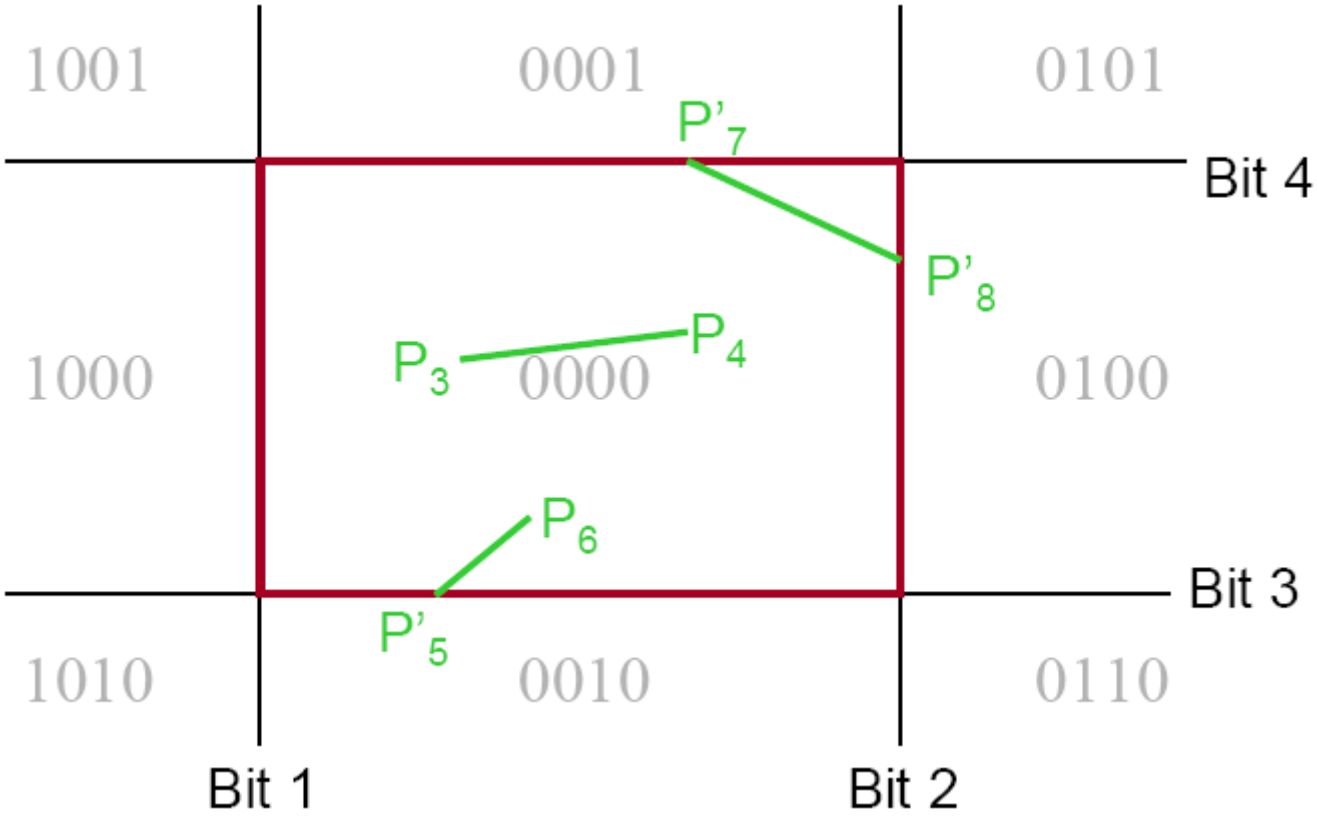
Cohen Sutherland Line Clipping

- **Compute intersections with window boundary for lines that can't be classified quickly**



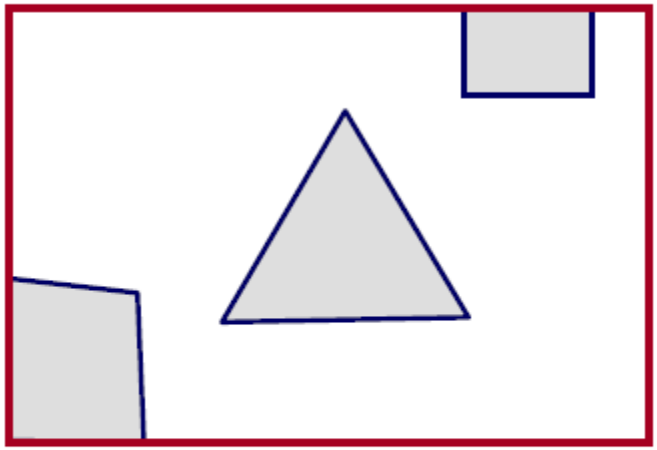
Cohen Sutherland Line Clipping

- **Compute intersections with window boundary for lines that can't be classified quickly**



Polygon Clipping

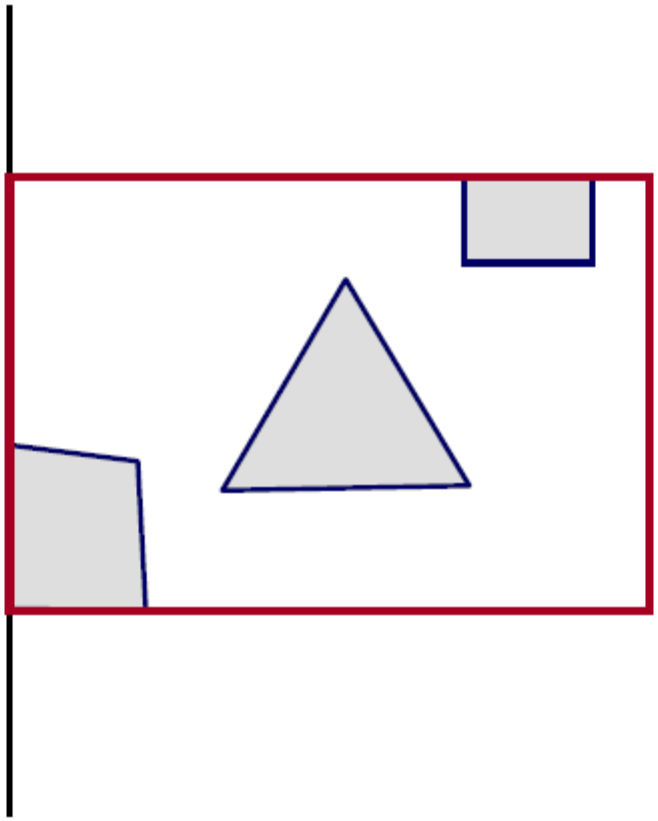
- **Find the part of a polygon inside the clip window?**



After Clipping

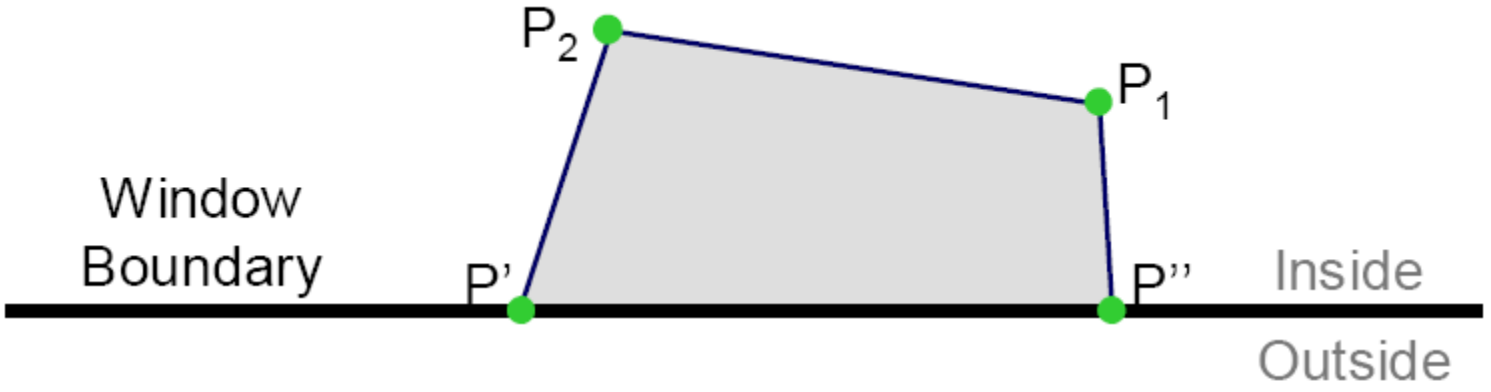
Sutherland Hodgeman Clipping

- **Clip each window boundary one at a time**

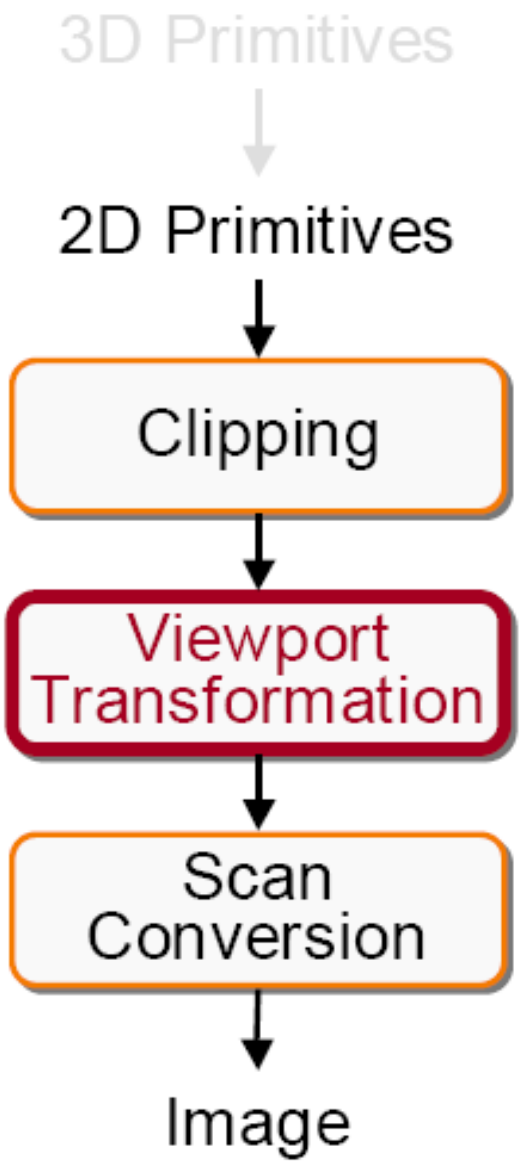


Clipping to a Boundary

1. **Do inside test for each point in sequence,**
2. **Insert new points when cross window boundary,**
3. **Remove Points outside window boundary**



2D Rendering Pipeline



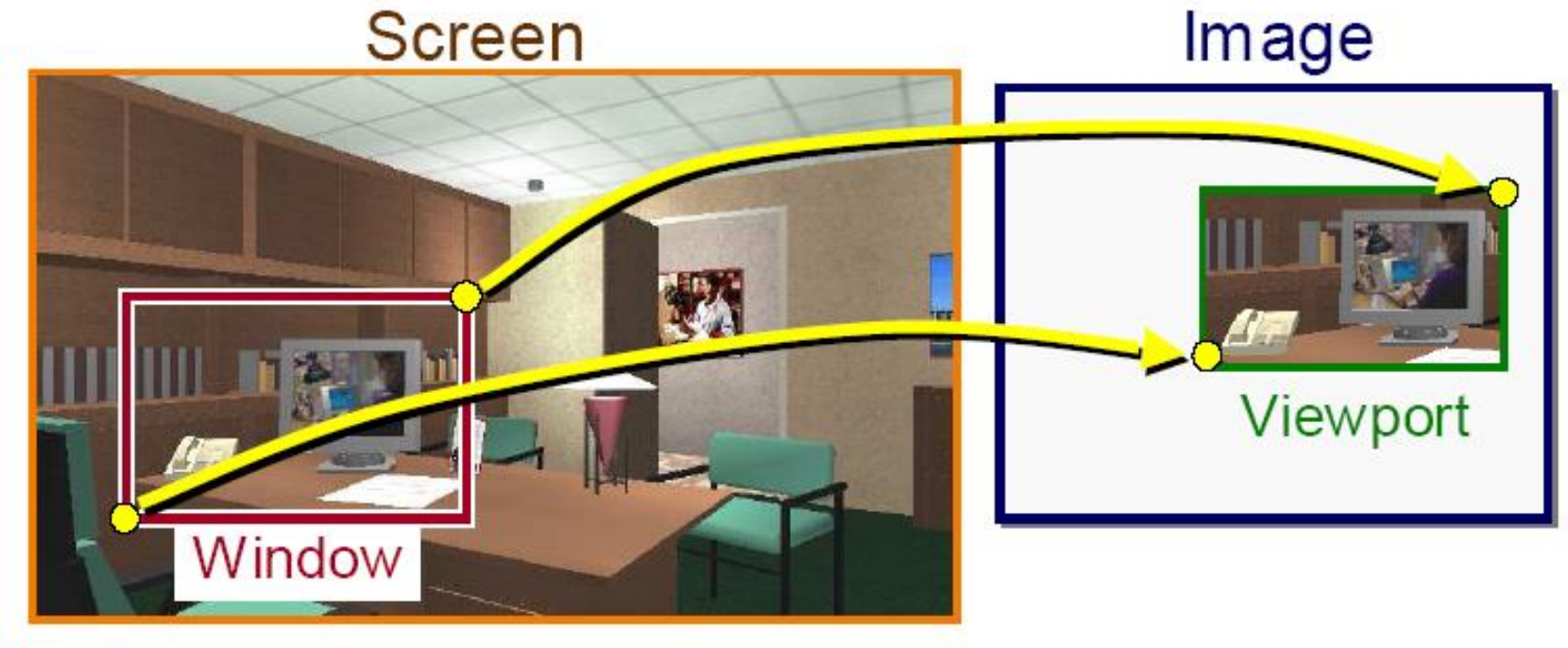
Clip portions of geometric primitives residing outside the window

Transform the clipped primitives From screen to image coordinates

Fill pixels representing primitives In screen coordinates

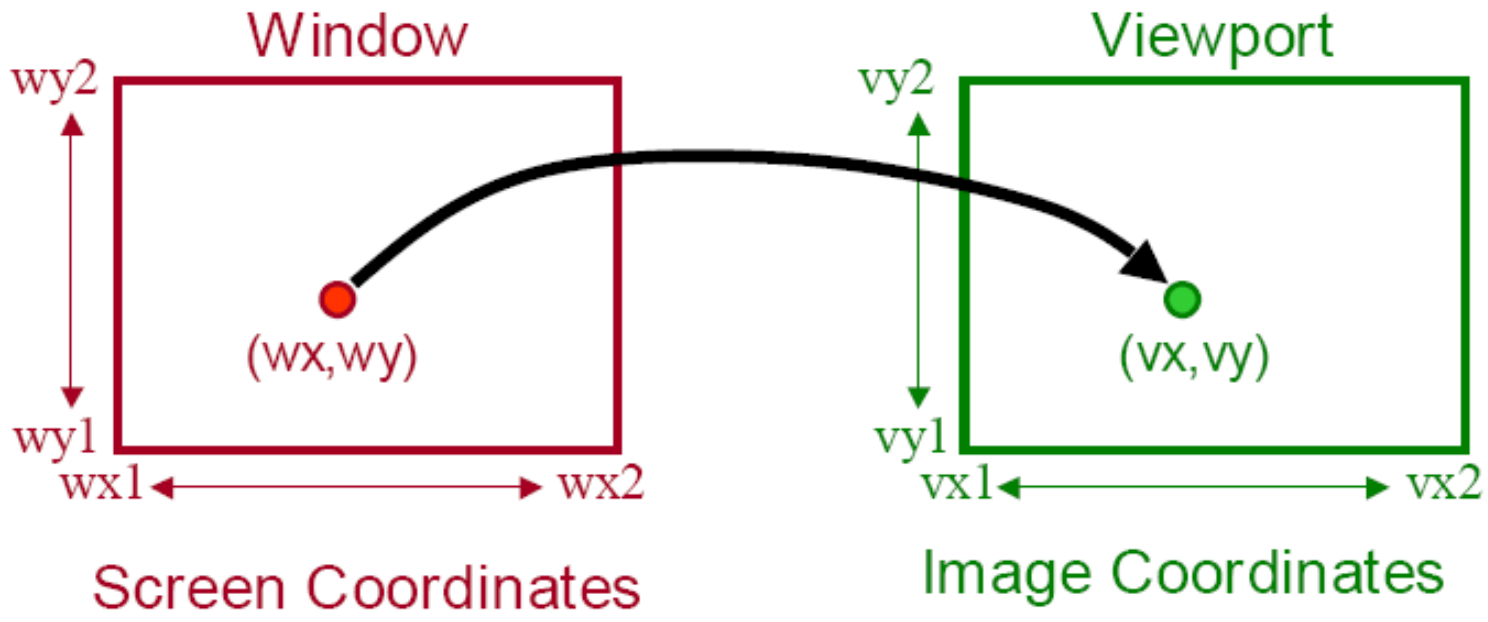
Viewport Transformation

- **Transformation 2D geometric primitives from screen coordinate system (normalized device coordinates) to image coordinate system (pixels)**



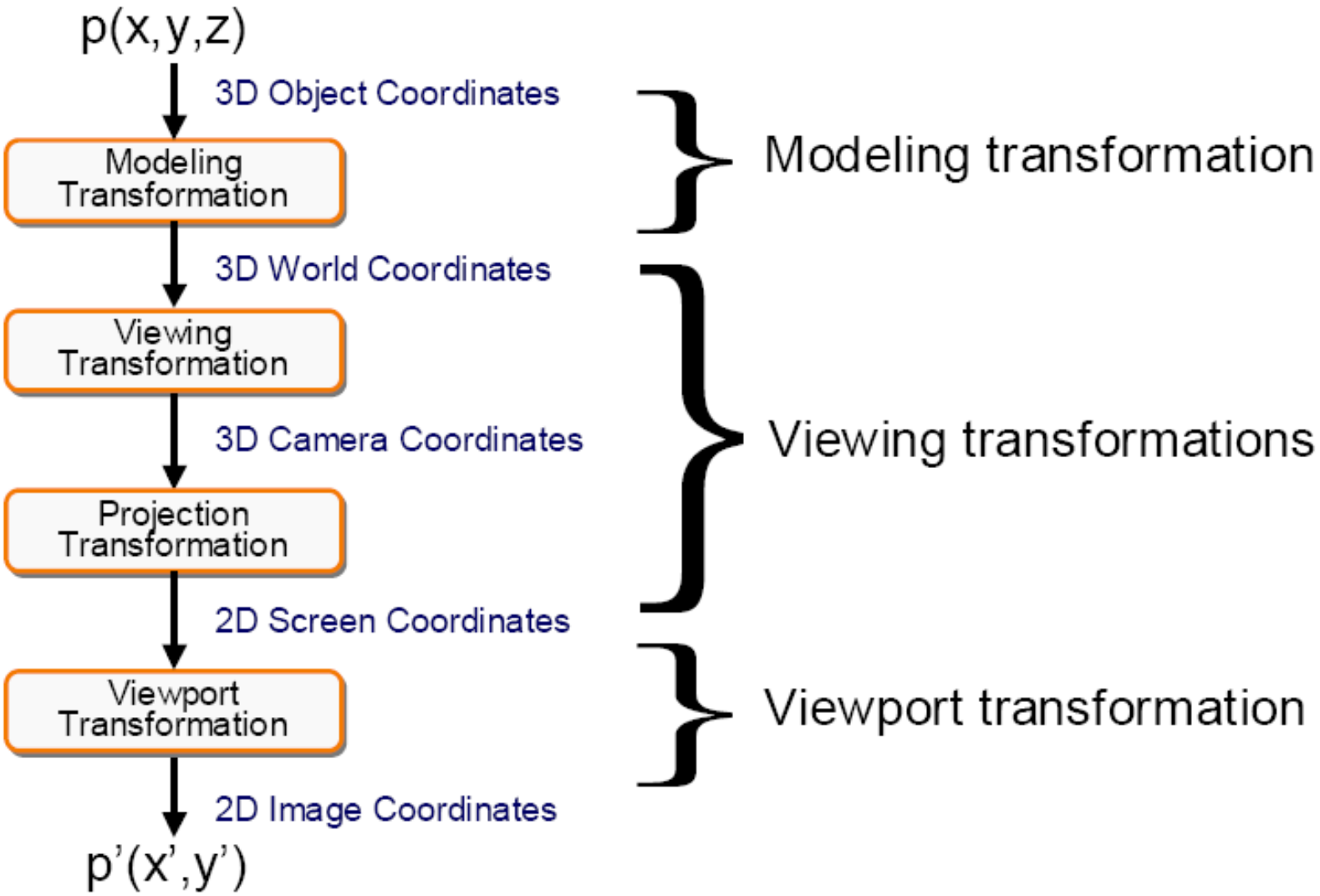
Viewport Transformation

- **Window-to-Viewport mapping**

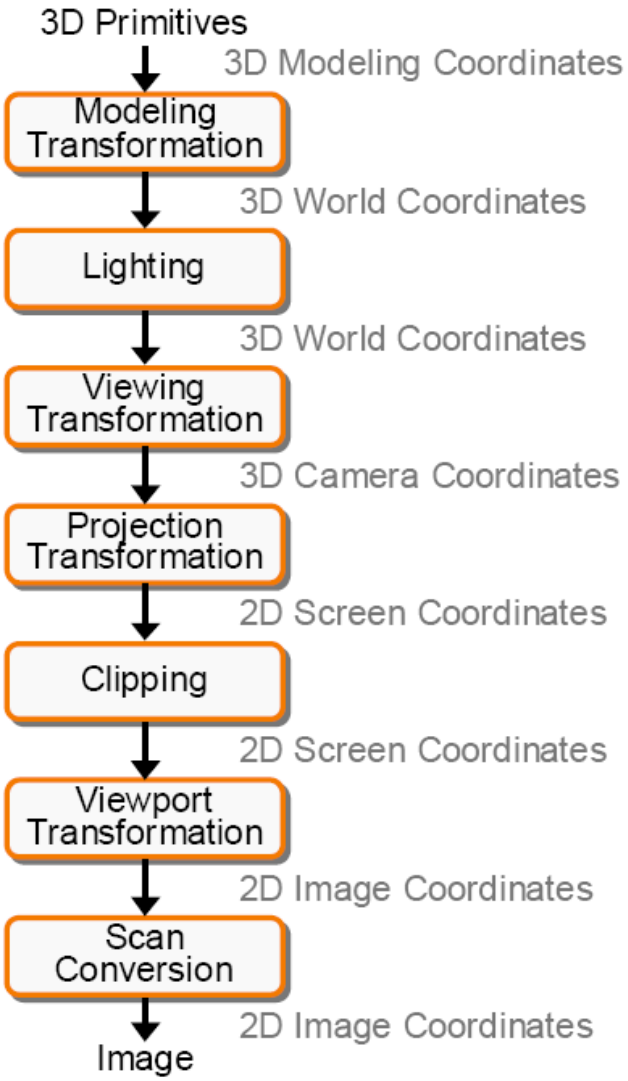


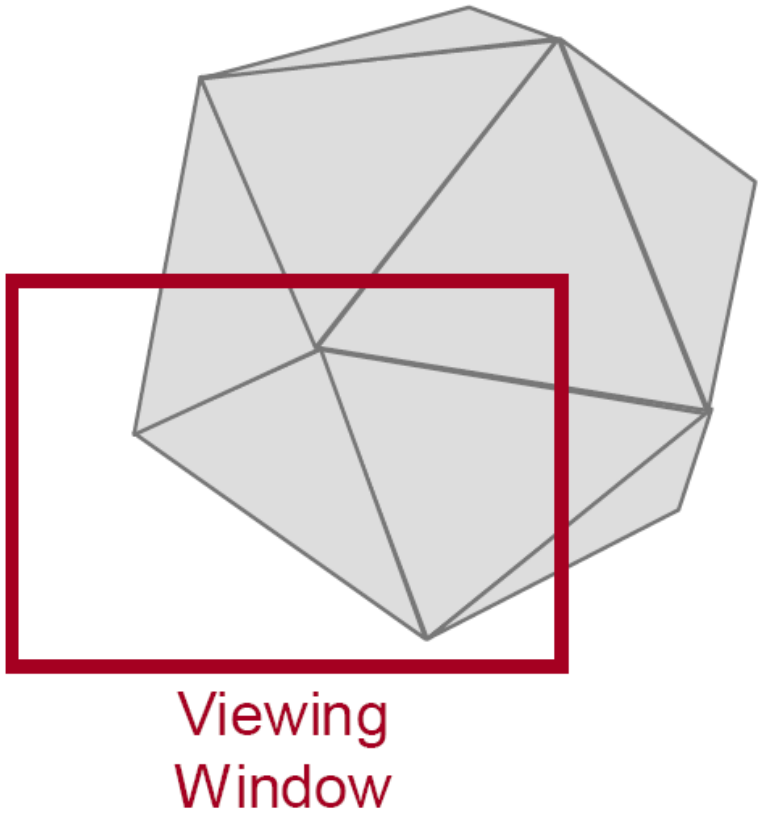

```
vx = vx1 + (wx - wx1) * (vx2 - vx1) / (wx2 - wx1);
vy = vy1 + (wy - wy1) * (vy2 - vy1) / (wy2 - wy1);
```

Summary of Transformations

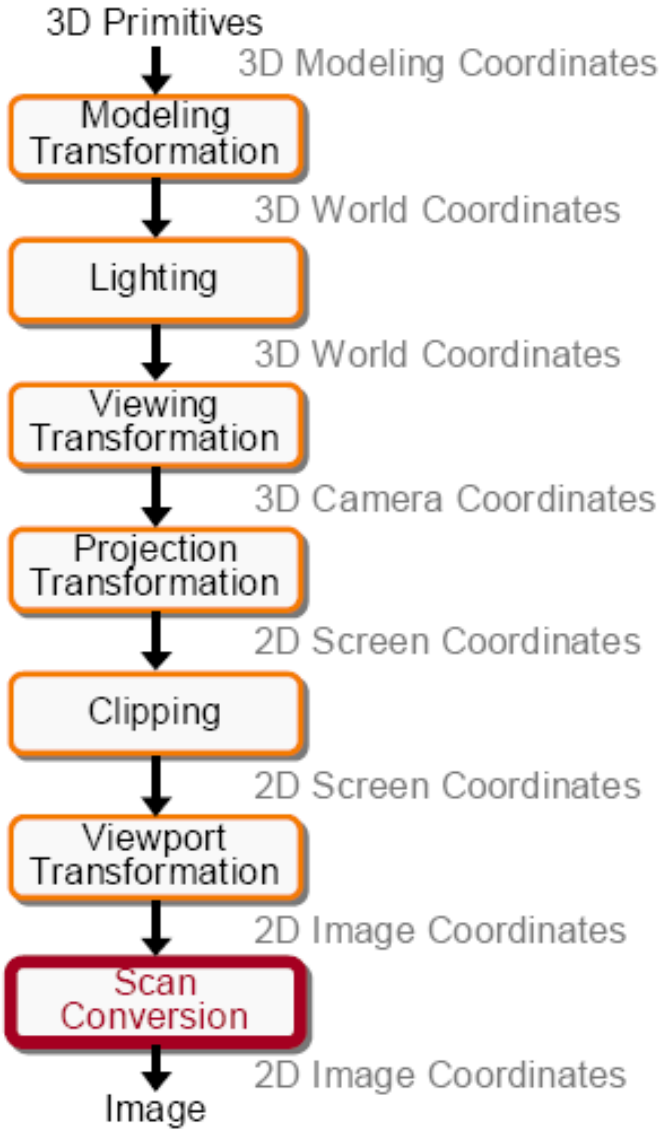


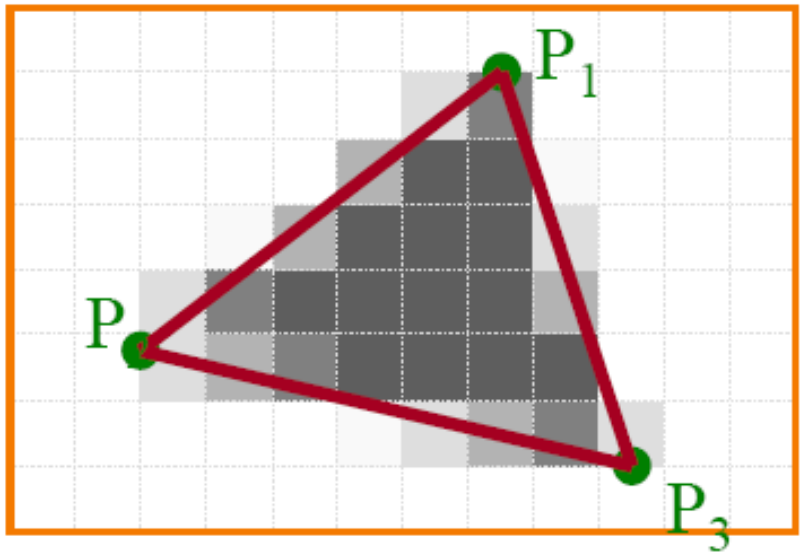
Summary


3D Primitives
3D Modeling Coordinates
Modeling Transformation
3D World Coordinates
Lighting
3D World Coordinates
Viewing Transformation
3D Camera Coordinates
Projection Transformation
2D Screen Coordinates
Clipping
2D Screen Coordinates
Viewport Transformation
2D Image Coordinates
Scan Conversion
2D Image Coordinates
Image


Viewing Window

Next Time





Scan Conversion!