3D Polygon Rendering Pipeline

CS 4810: Graphics

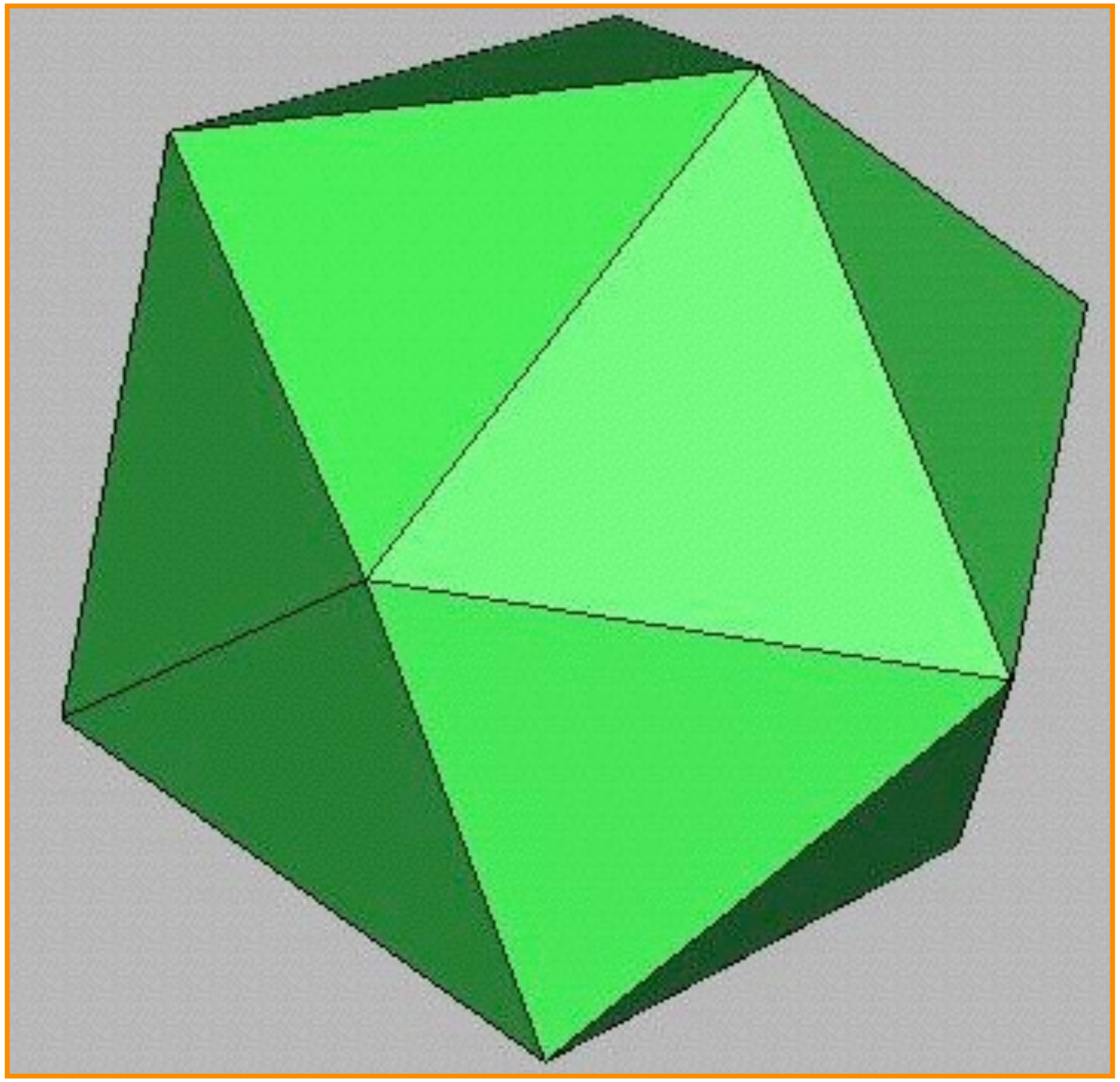
Acknowledgment: slides by Jason Lawrence, Misha Kazhdan, Allison Klein, Tom Funkhouser, Adam Finkelstein and David Dobkin

Road Map for Next Lectures

- Leaving ray-tracing
- Moving on to polygon-based rendering
  - Rendering pipeline (today)
  - Clipping
  - Scan conversion & shading
  - Texture-mapping
  - Hidden-surface removal
- Polygon-based rendering is what happens on your PC (think NVIDIA, etc.)

3D Polygon Rendering

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



3D Polygon Rendering

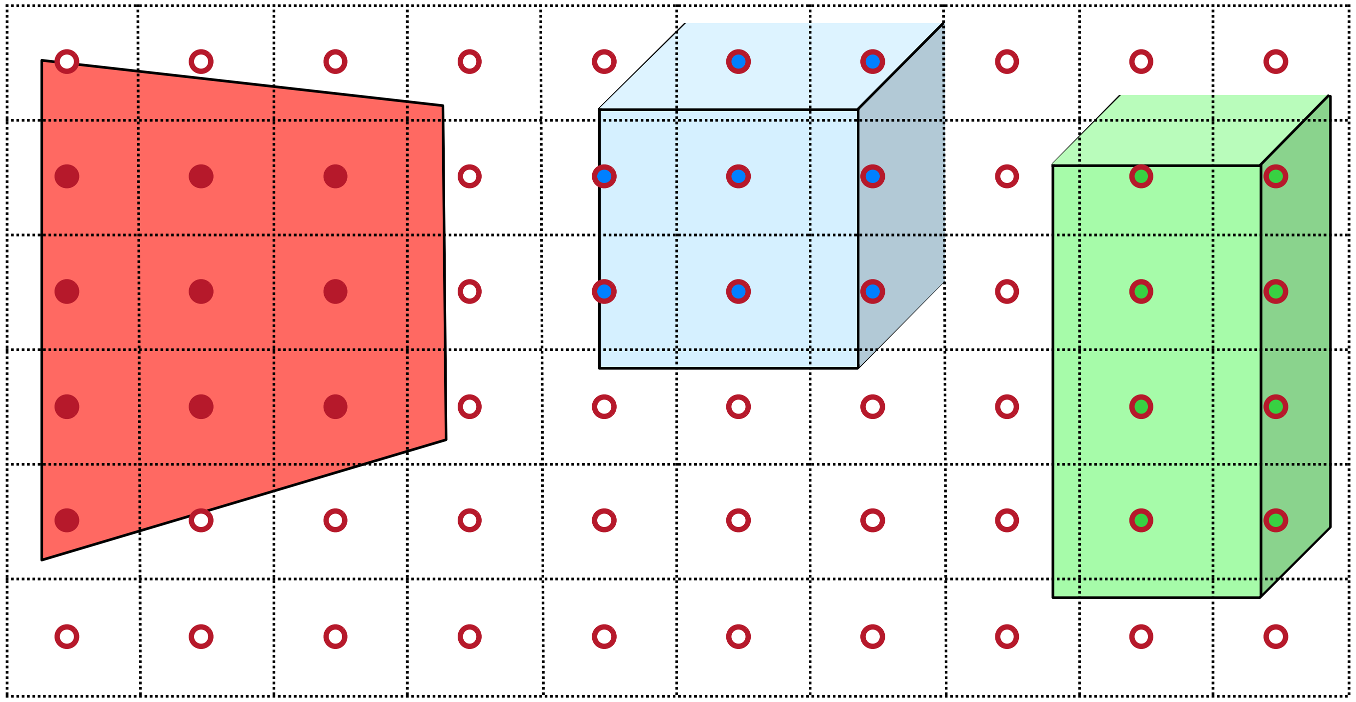
- Many applications use rendering of 3D polygons with direct illumination

3D Polygon Rendering

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

Ray Casting Revisited

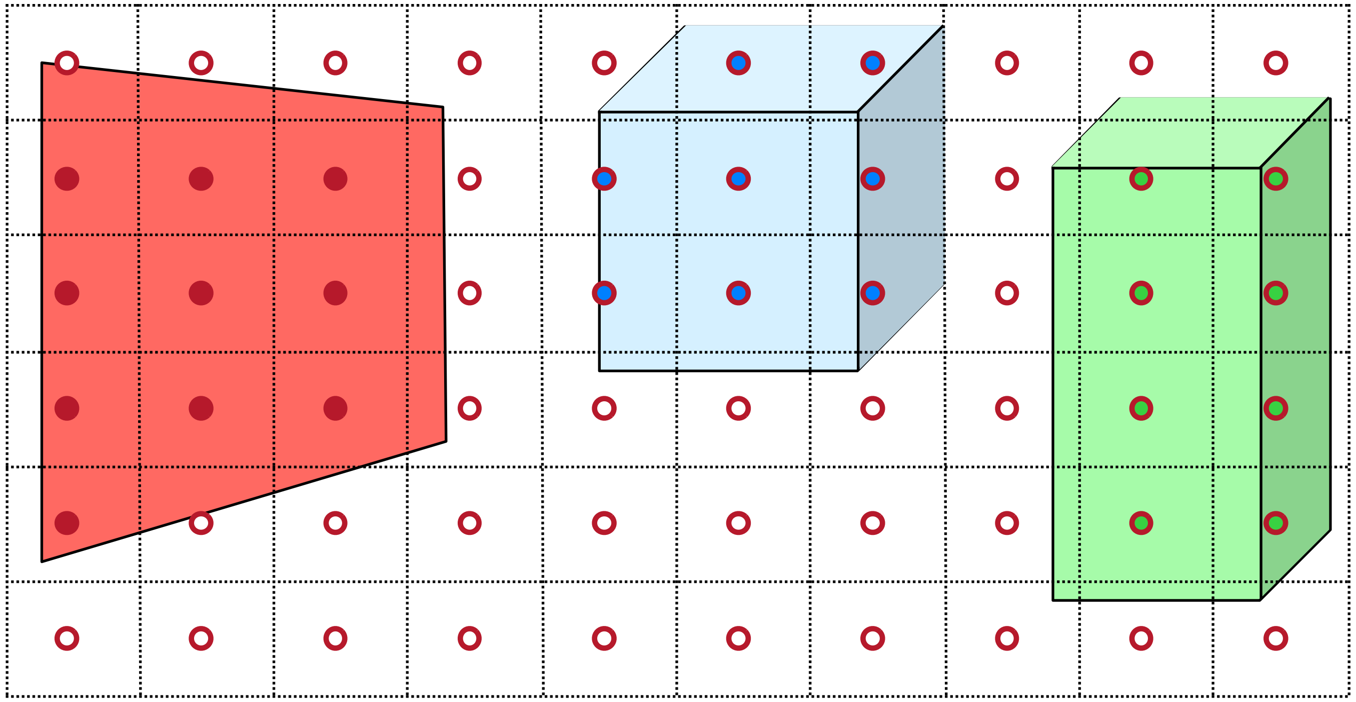
- For each sample …
  - Construct ray from eye position through view plane
  - Find first surface intersected by ray through pixel
  - Compute color of sample based on surface radiance



More efficient algorithms utilize spatial coherence!

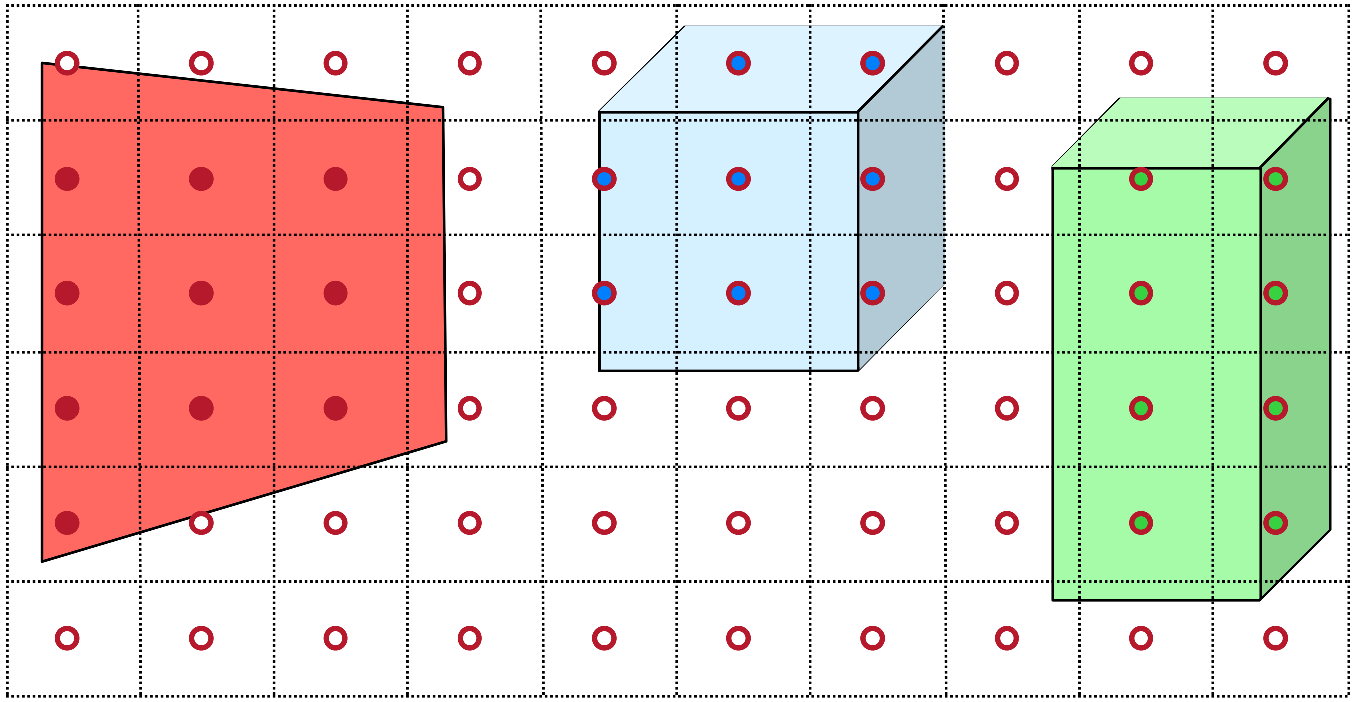
3D Polygon Rendering

- Logical inverse of ray casting
- Idea: Instead of sending rays from the camera into the scene, send rays from the scene into the camera.

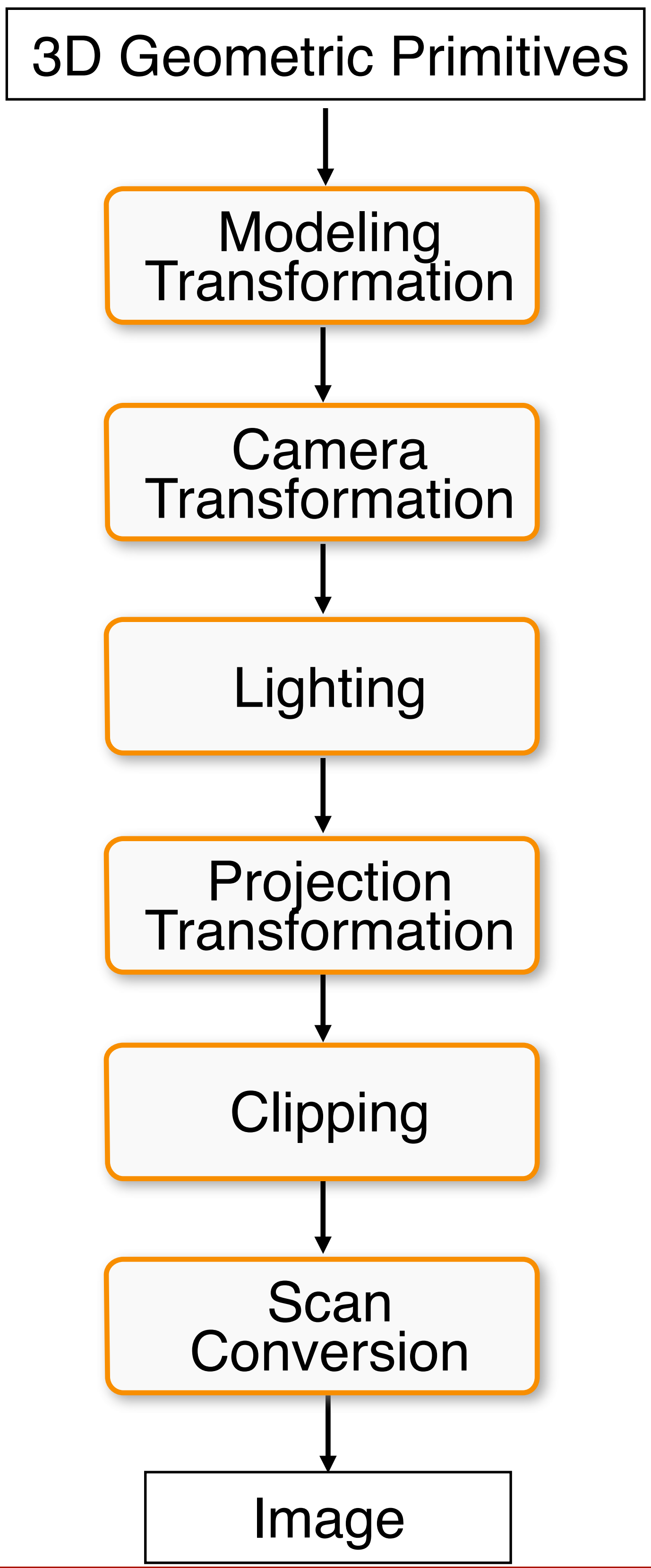


3D Polygon Rendering

- Ray casting: pick pixel and figure out what color it should be based on what object its ray hits
- Polygon rendering: pick polygon and figure out what pixels it should affect

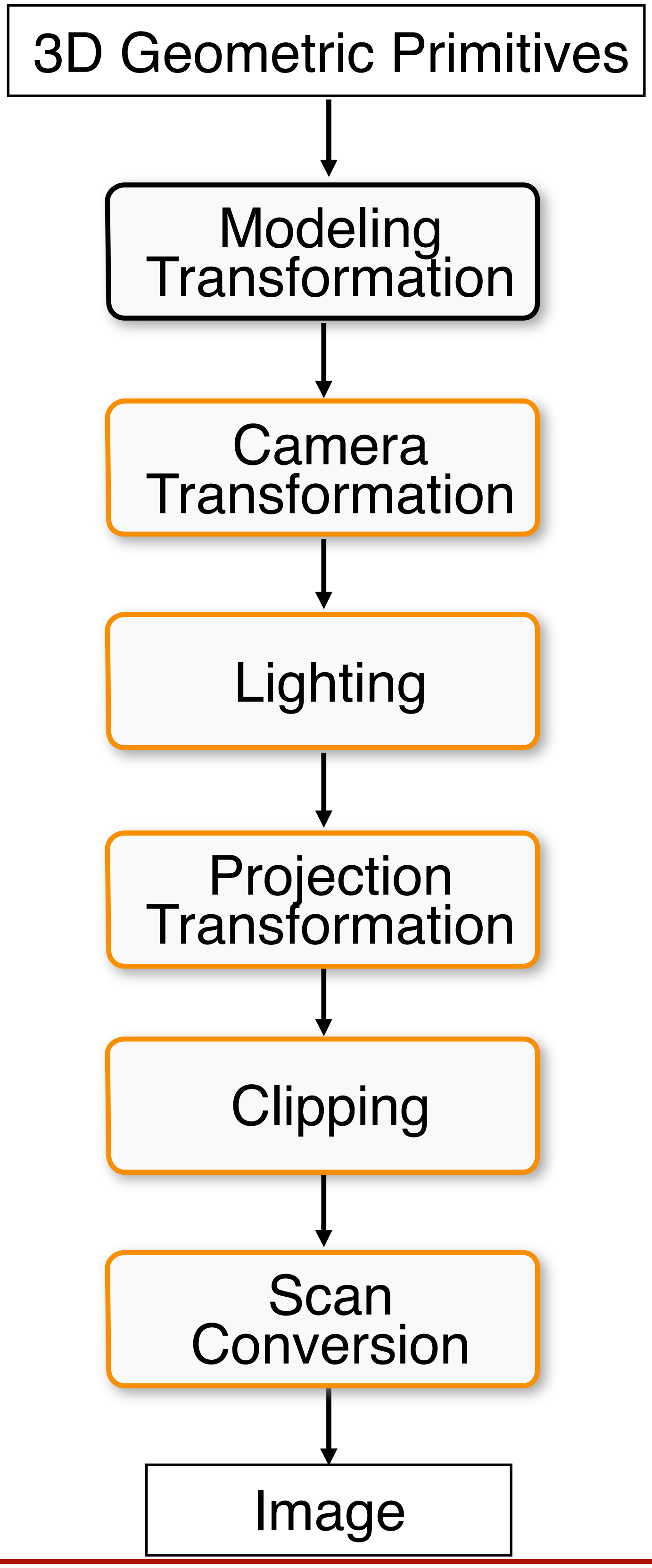


3D Rendering Pipeline (direct illumination)



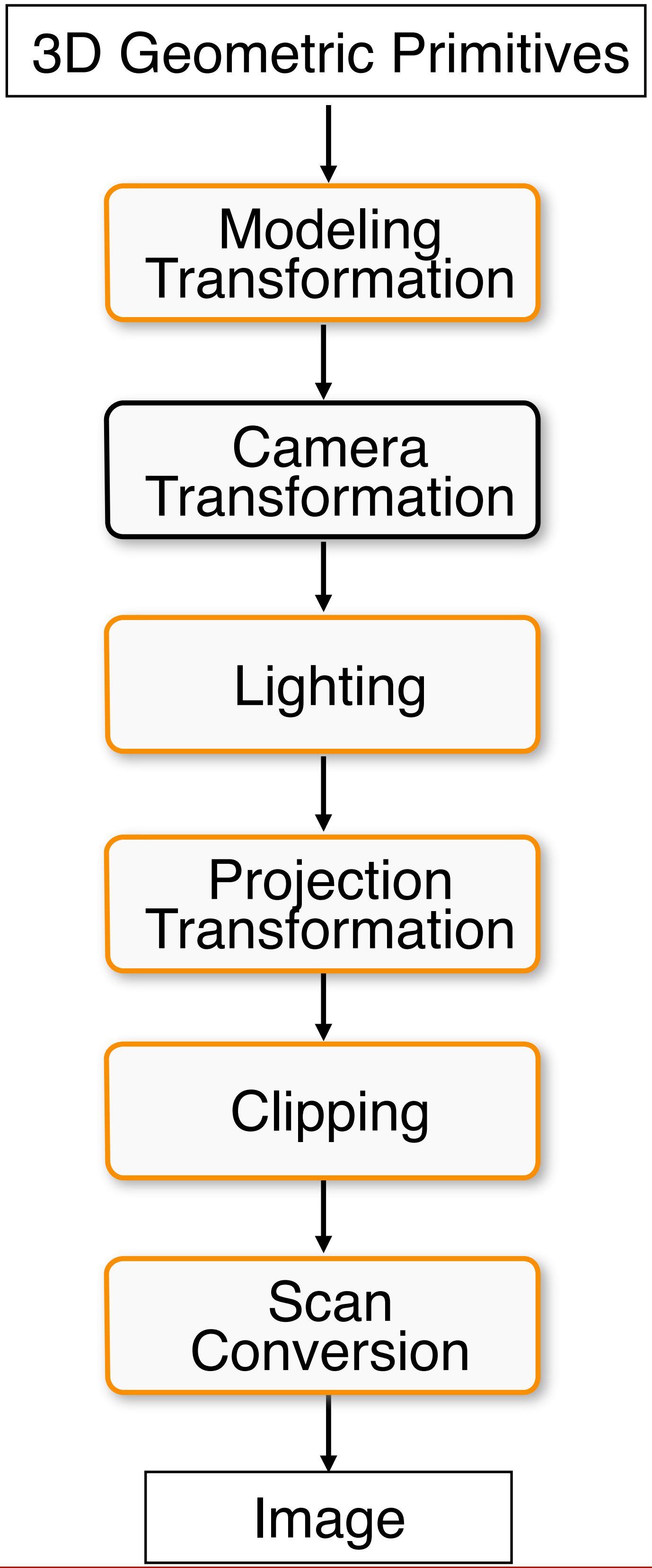
This is a pipelined sequence of operations to draw a 3D primitive into a 2D image

3D Rendering Pipeline (direct illumination)



Transform from current (local) coordinate system into 3D world coordinate system

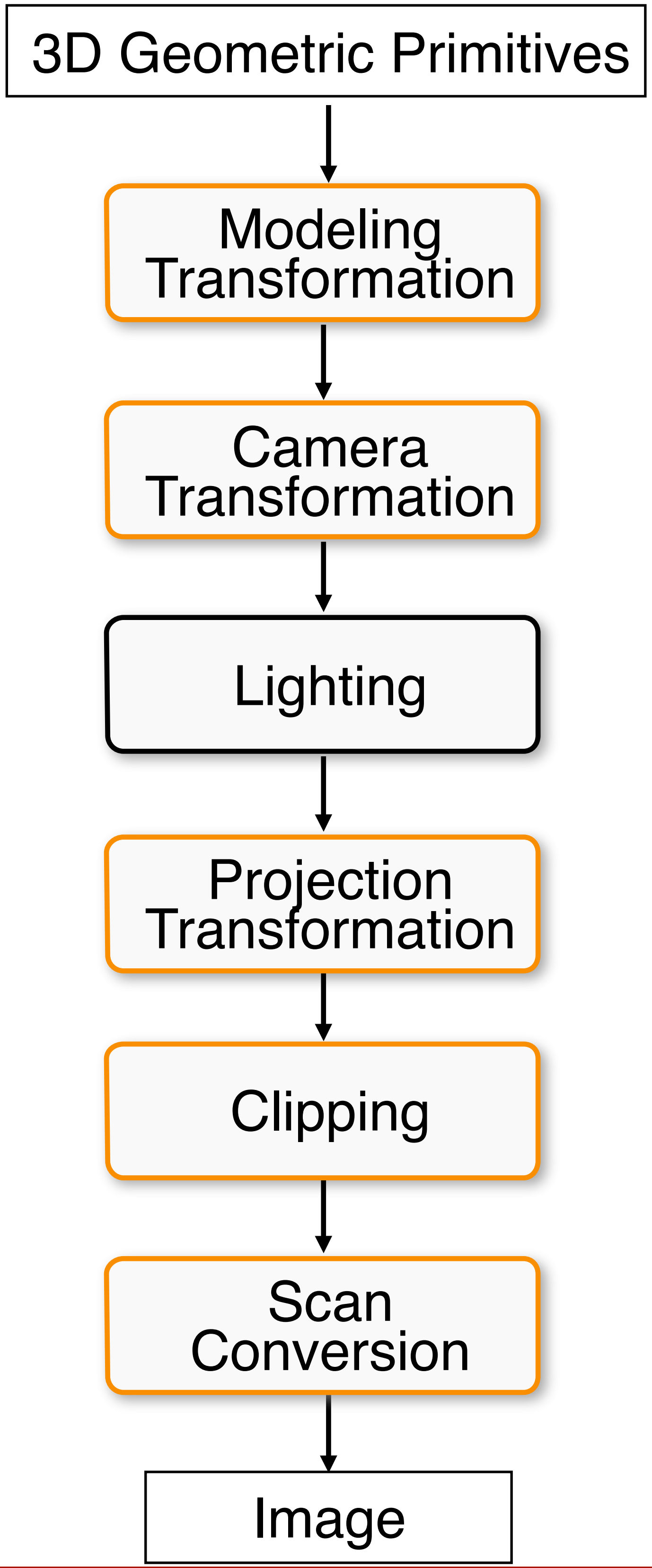
3D Rendering Pipeline (for direct illumination)



Transform into 3D world coordinate system

Transform into 3D camera coordinate system

3D Rendering Pipeline (for direct illumination)

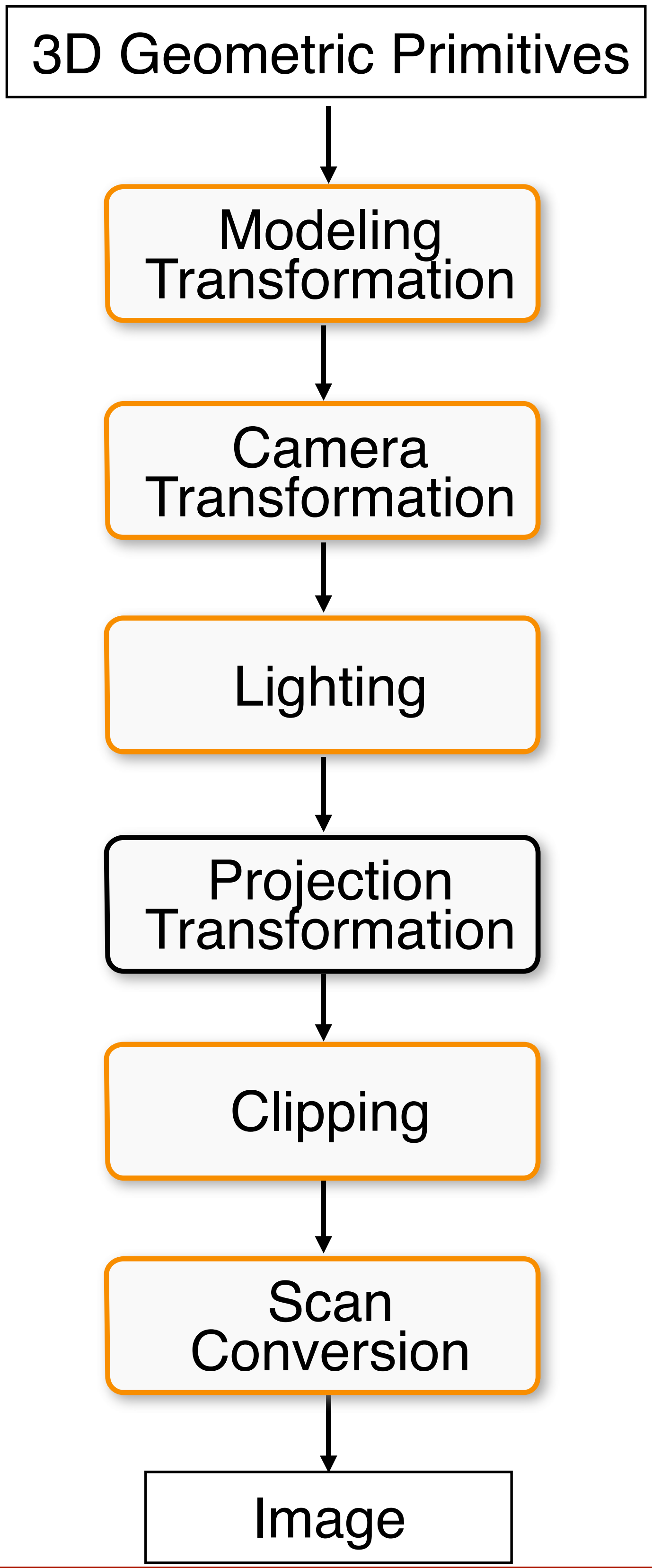


Transform into 3D world coordinate system

Transform into 3D camera coordinate system

Illuminate according to lighting and reflectance

3D Rendering Pipeline (for direct illumination)



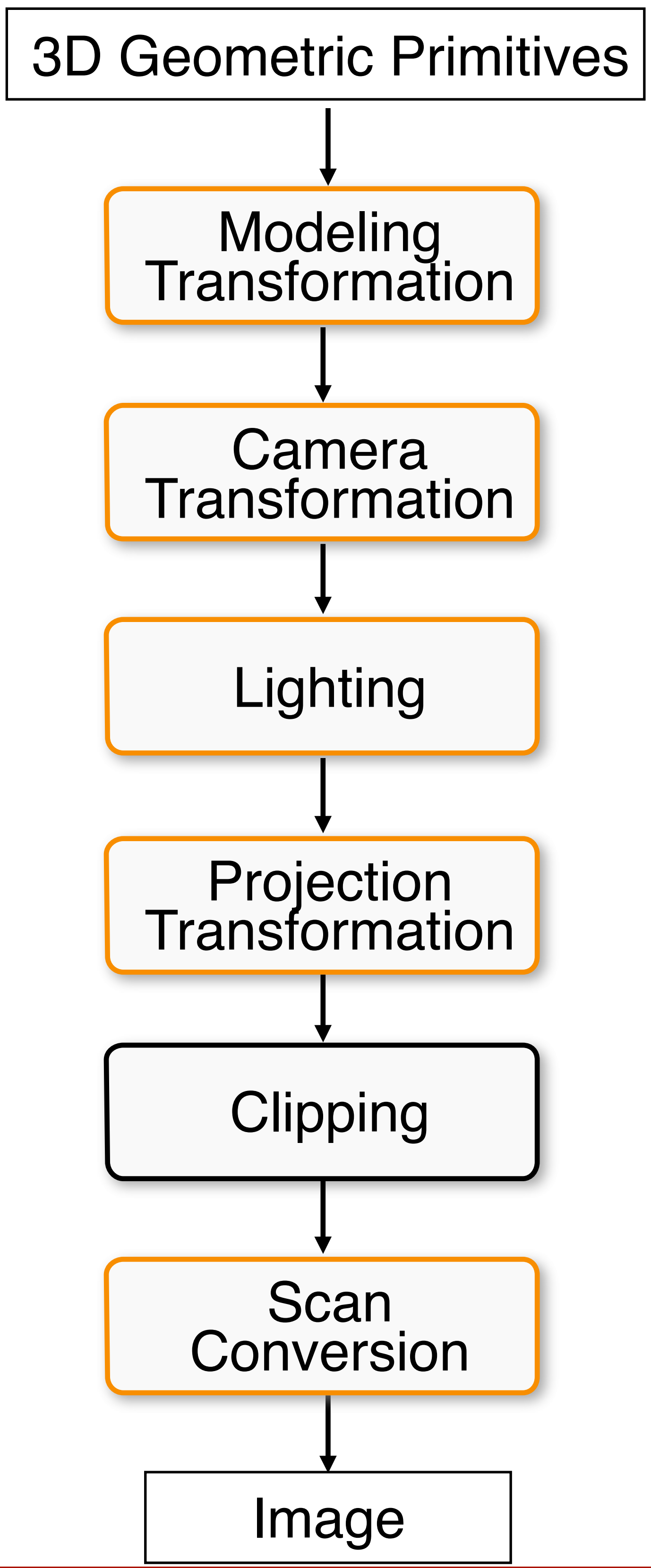
Transform into 3D world coordinate system

Transform into 3D camera coordinate system

Illuminate according to lighting and reflectance

Transform into 2D camera coordinate system

3D Rendering Pipeline (for direct illumination)



Transform into 3D world coordinate system

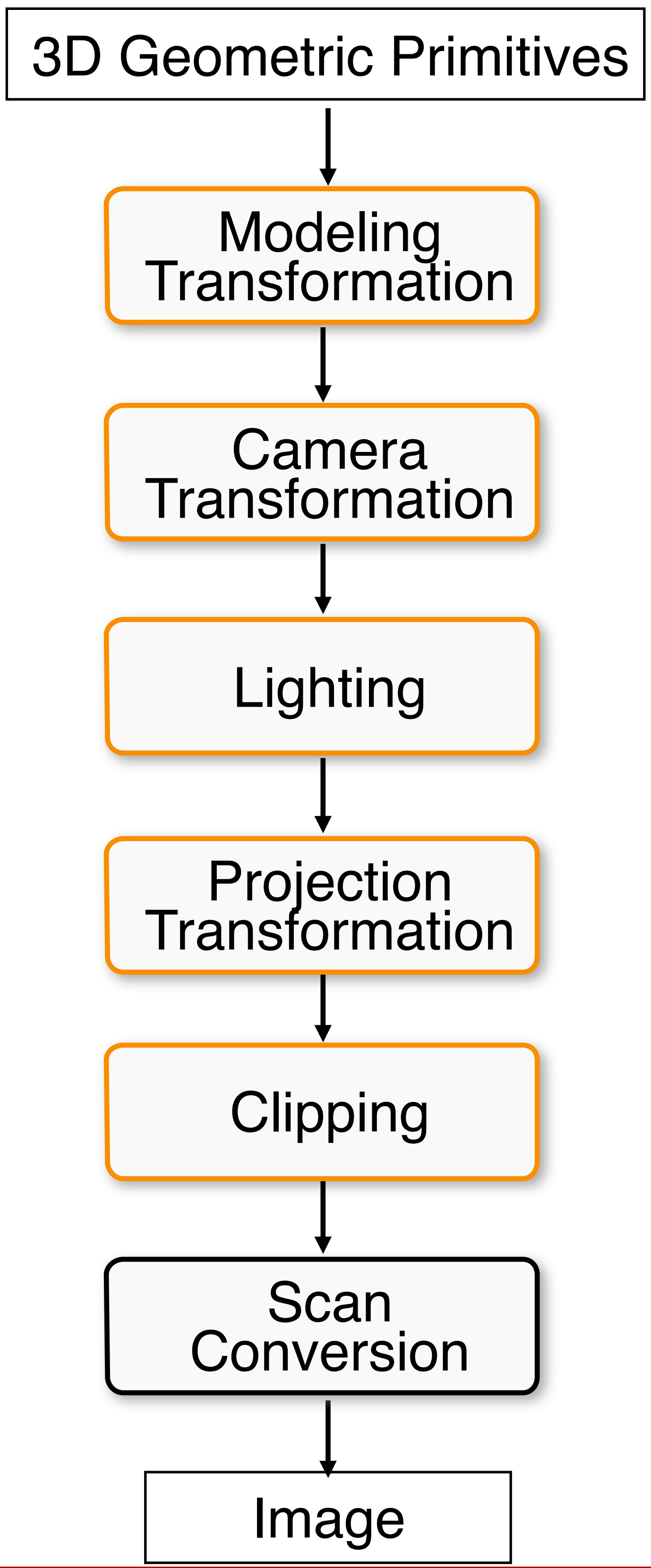
Transform into 3D camera coordinate system

Illuminate according to lighting and reflectance

Transform into 2D camera coordinate system

Clip (parts of) primitives outside camera's view

3D Rendering Pipeline (for direct illumination)



Transform into 3D world coordinate system

Transform into 3D camera coordinate system

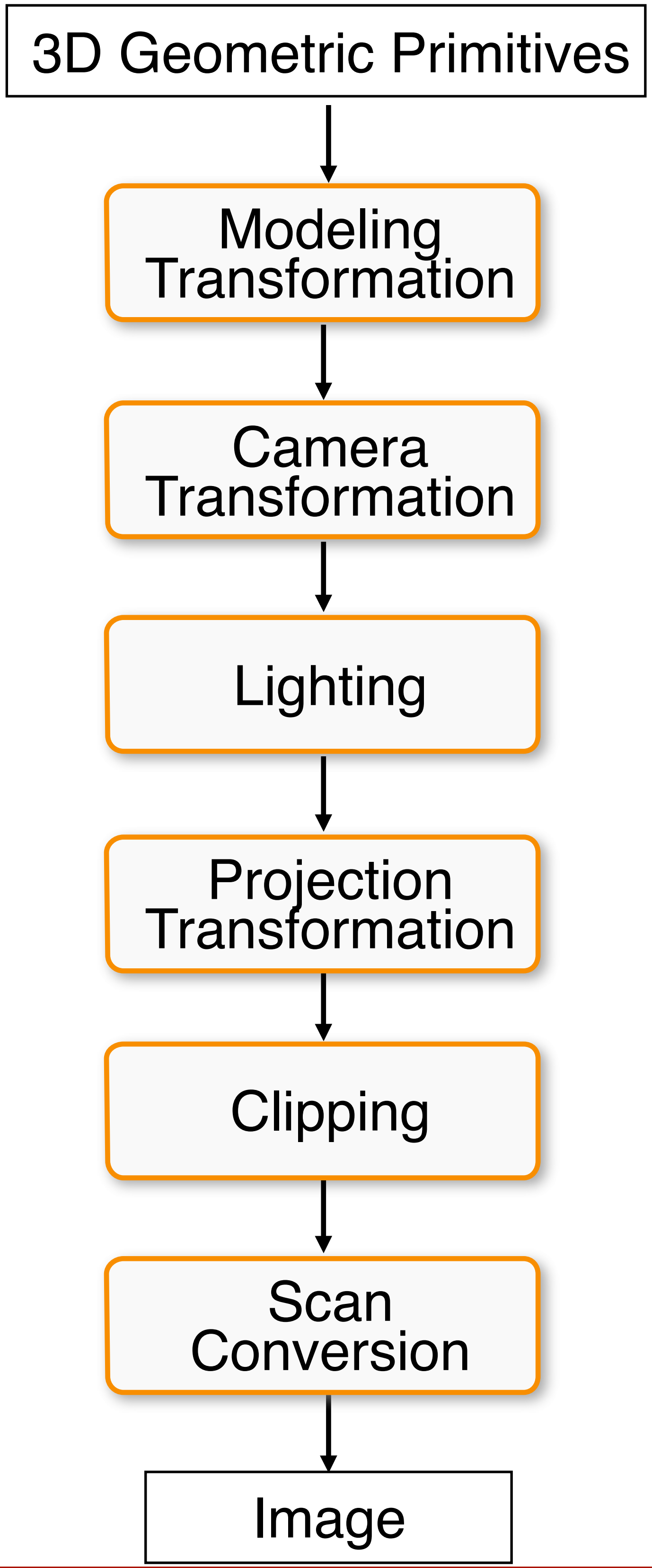
Illuminate according to lighting and reflectance

Transform into 2D camera coordinate system

Clip (parts of) primitives outside camera's view

Draw pixels (includes texturing, hidden surface, ...)

Transformations



Transform into 3D world coordinate system

Transform into 3D camera coordinate system

Illuminate according to lighting and reflectance

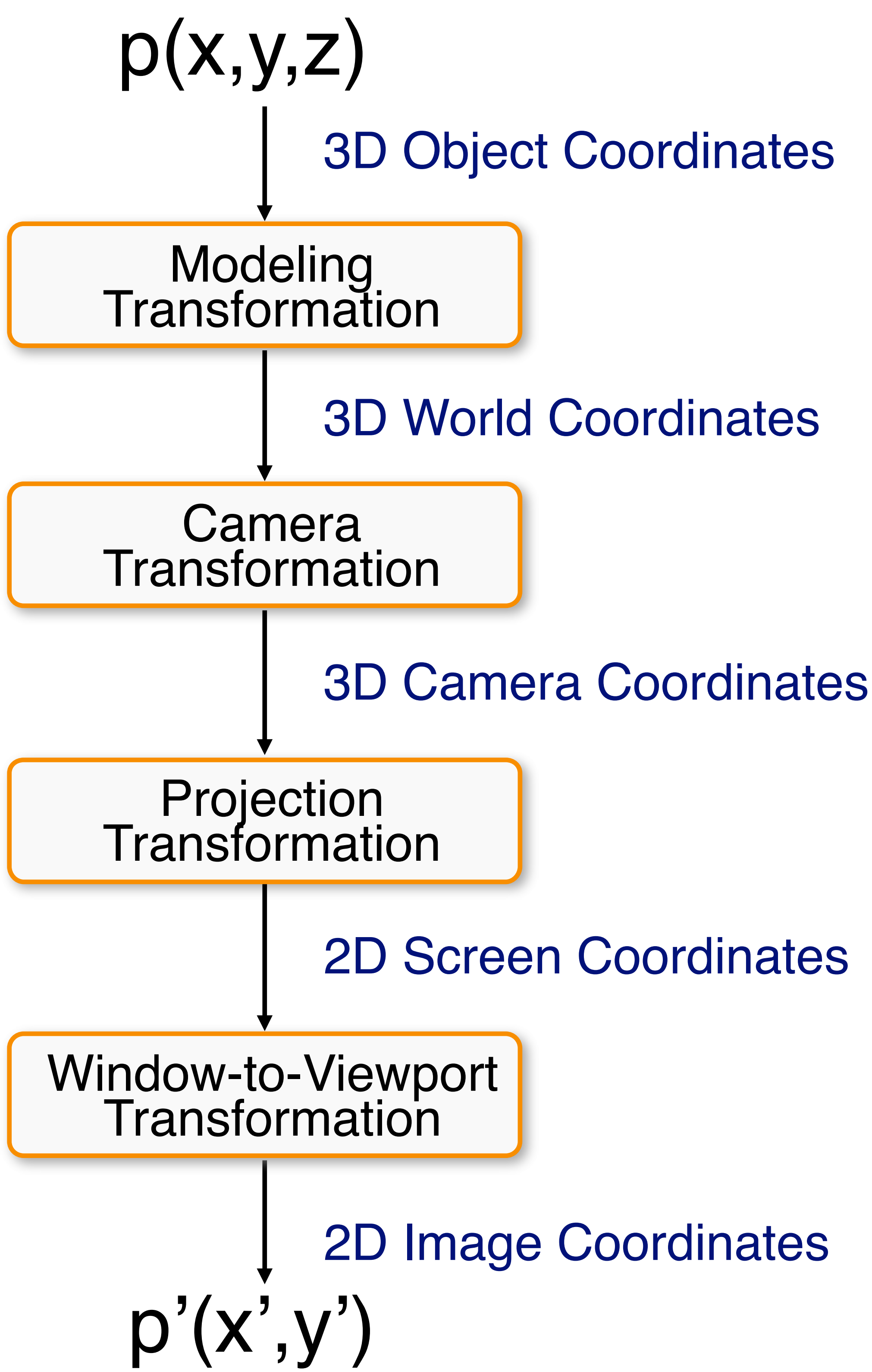
Transform into 2D camera coordinate system

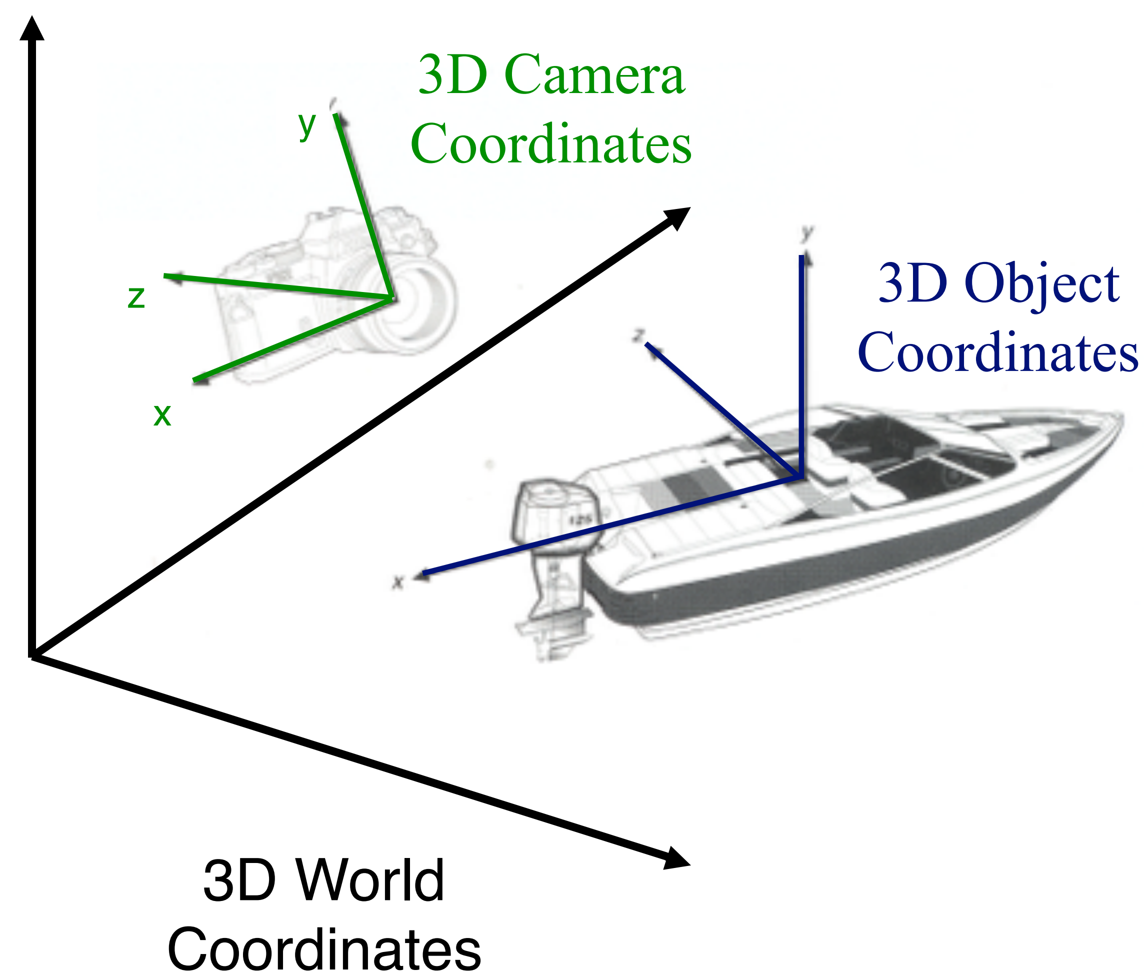
Clip primitives outside camera's view

Draw pixels (includes texturing, hidden surface, etc.)

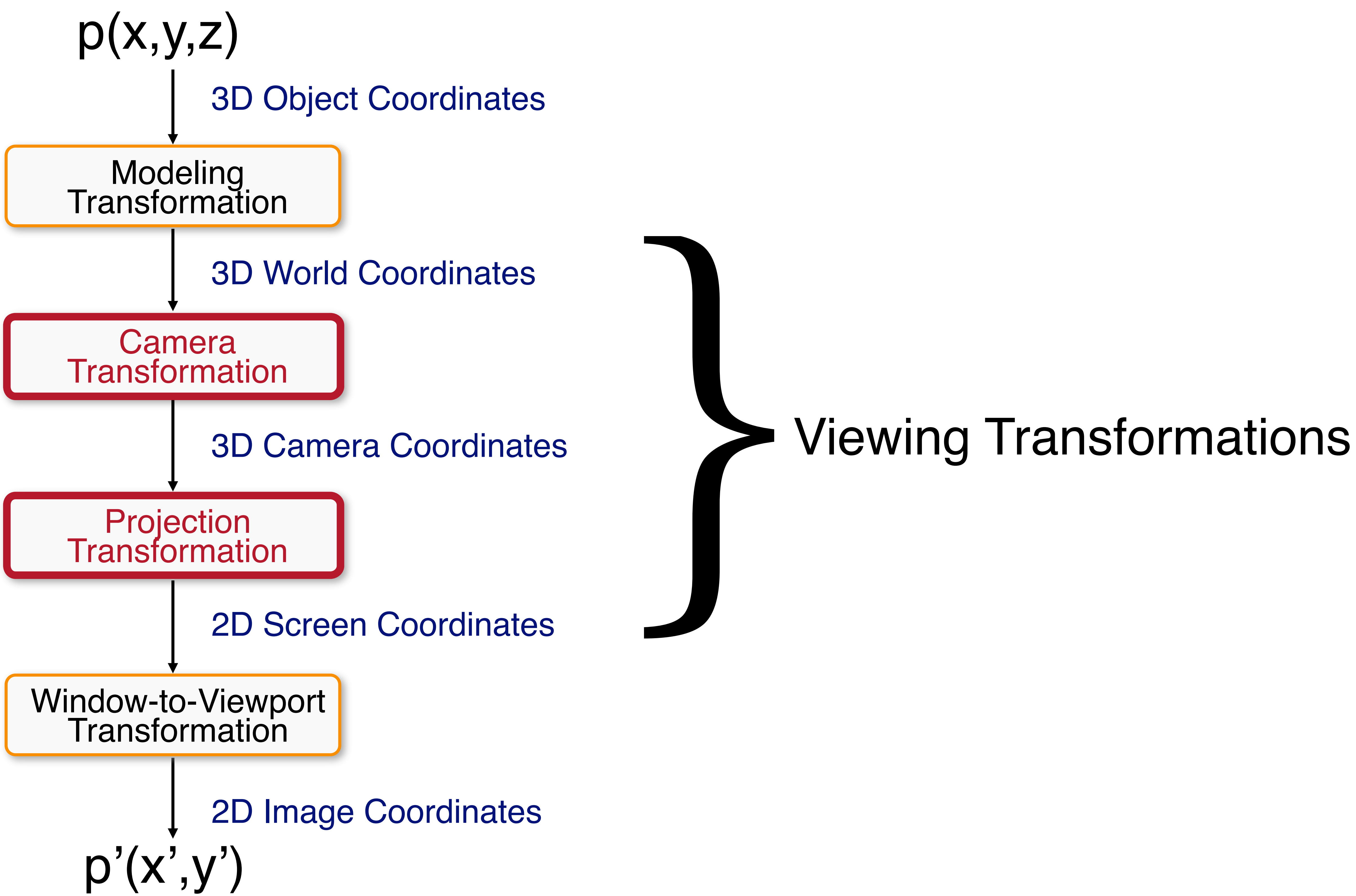
Transformations

Transformations map points from one coordinate system to another



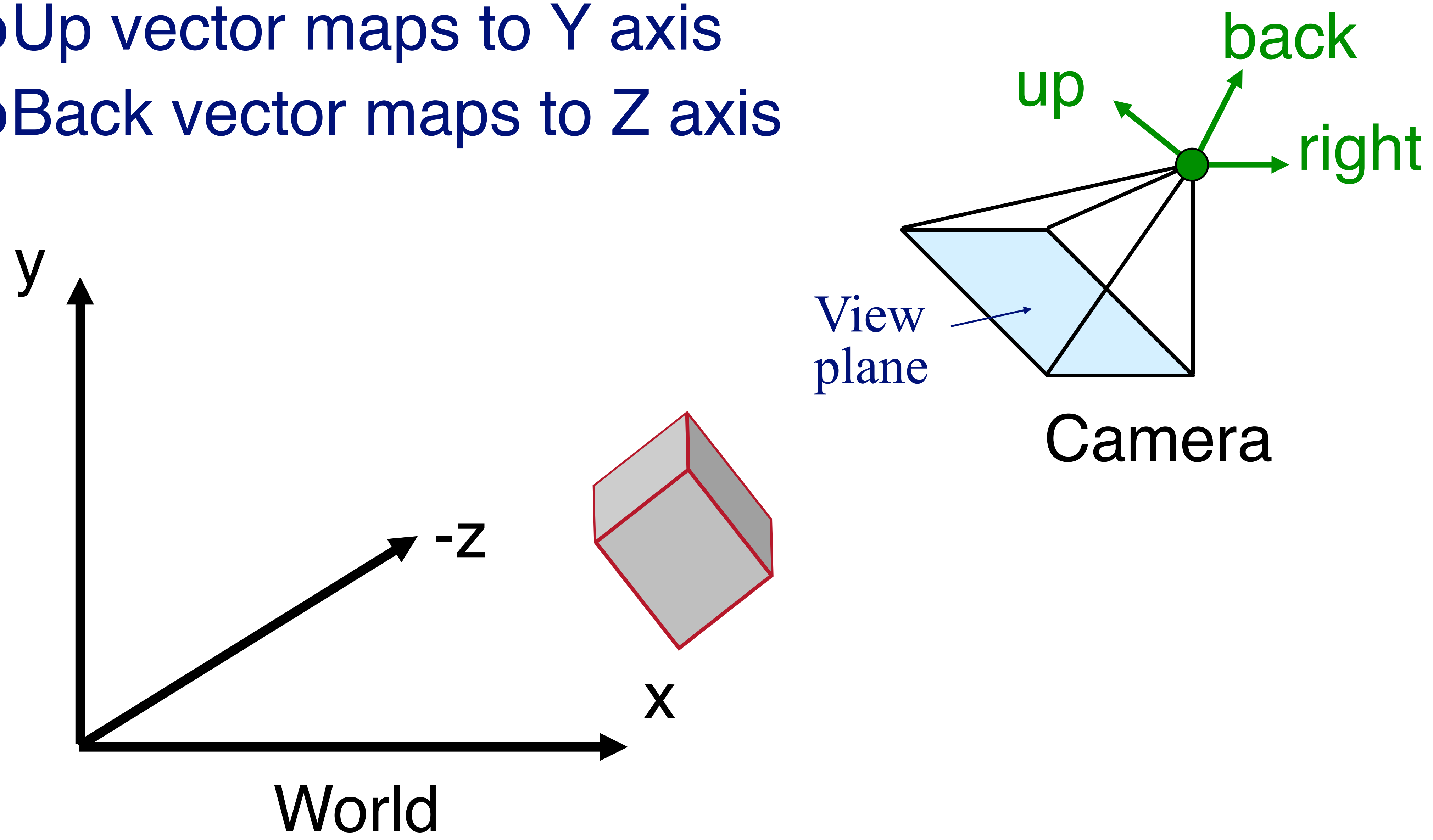


Viewing Transformations



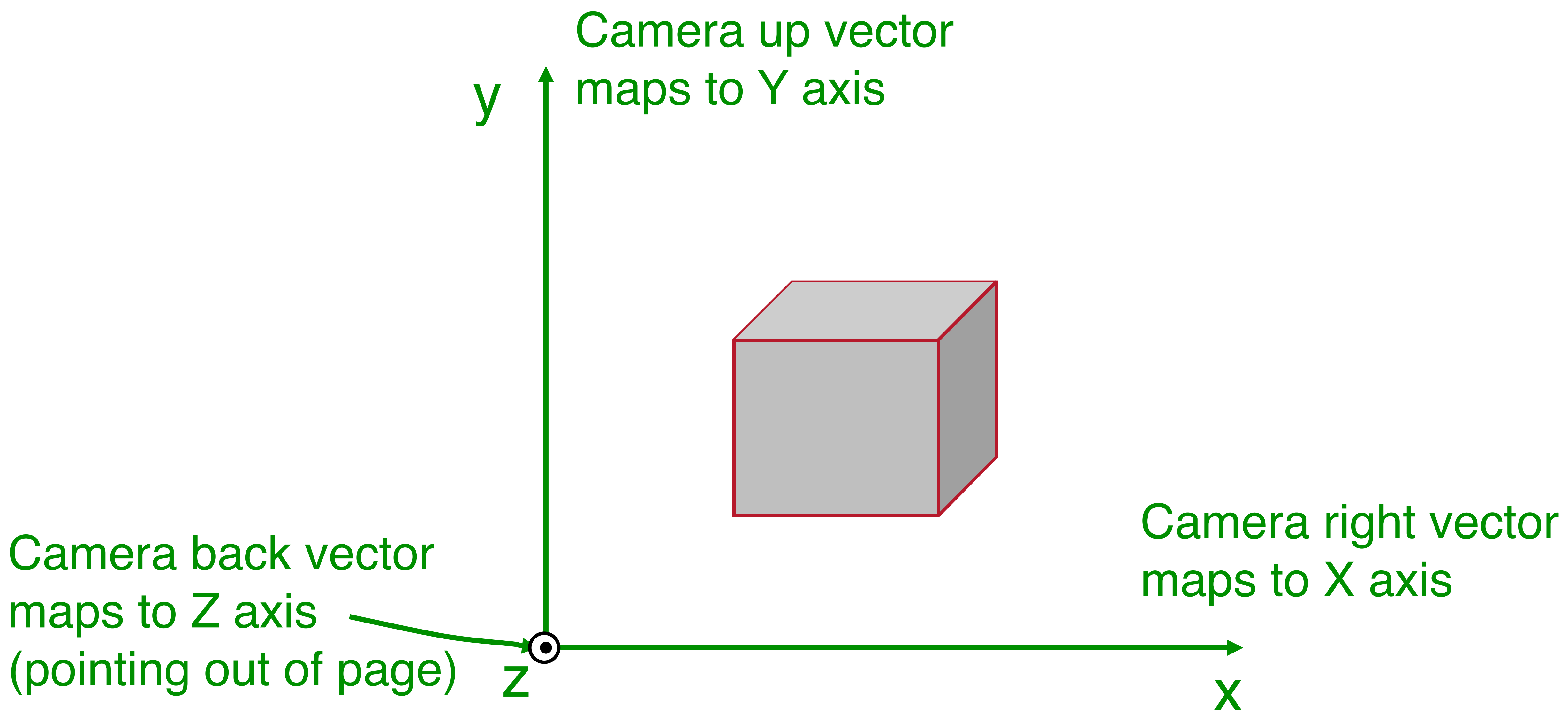
Viewing Transformation

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

$$p^c = T\ p^w$$

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

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

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

Finding the Viewing Transformation

- Trick: map from camera coordinates 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

$$p^W = T^{-1} p^C$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} R_x & U_x & B_x & E_x \\ R_y & U_y & B_y & E_y \\ R_z & U_z & B_z & E_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

- This matrix is T^{-1} so we invert it to get T … easy!

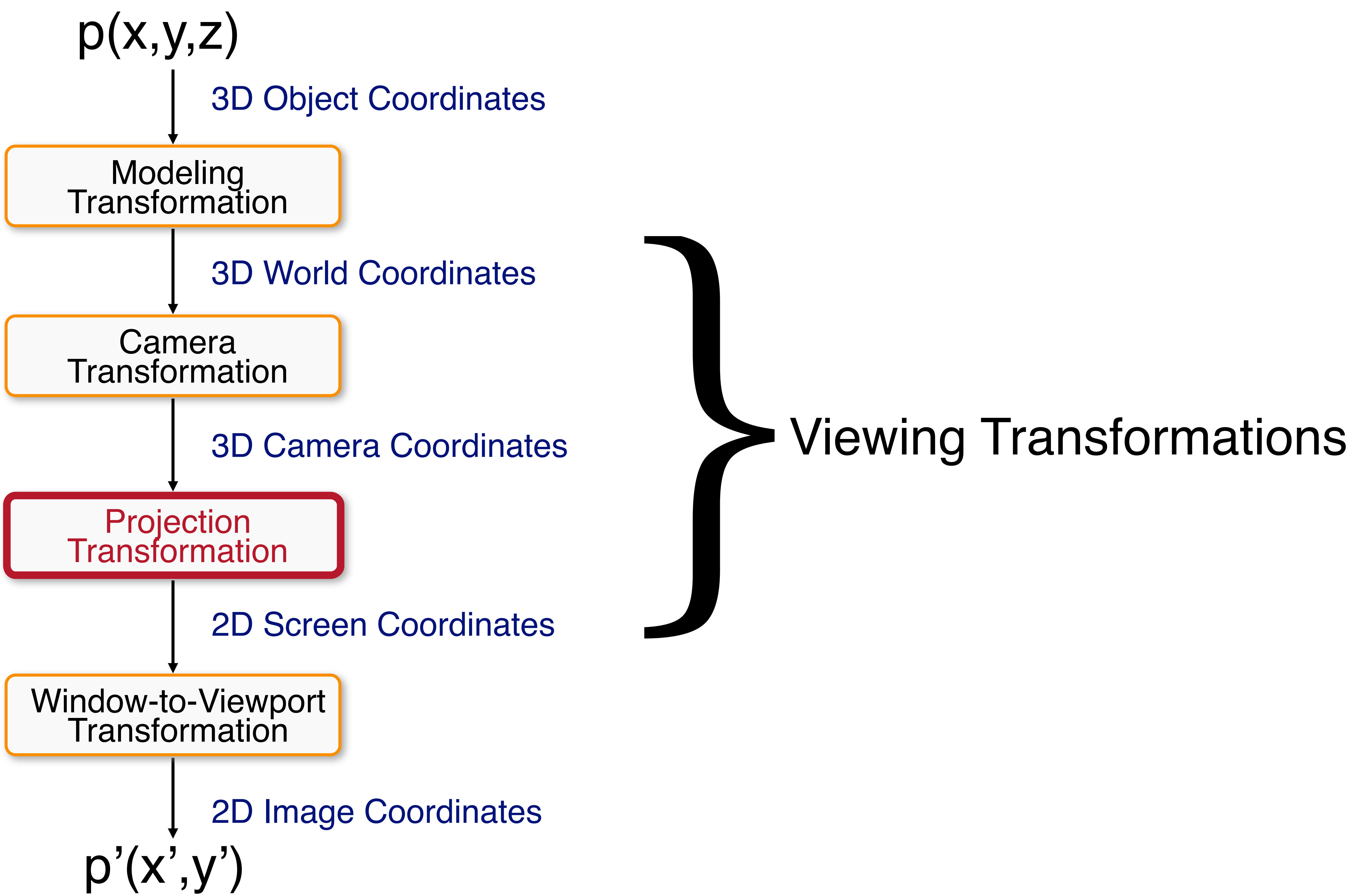
Finding the Viewing Transformation

- Trick: map from camera coordinates to world
  - Remember, with homogeneous coordinates, we divide through by *w* values…
  - So if we know *actual* point in 3D, $w = 1$
  - Easy to find code to invert a matrix

$$p^W = T^{-1} p^C \qquad \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} R_x & U_x & B_x & E_x \\ R_y & U_y & B_y & E_y \\ R_z & U_z & B_z & E_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

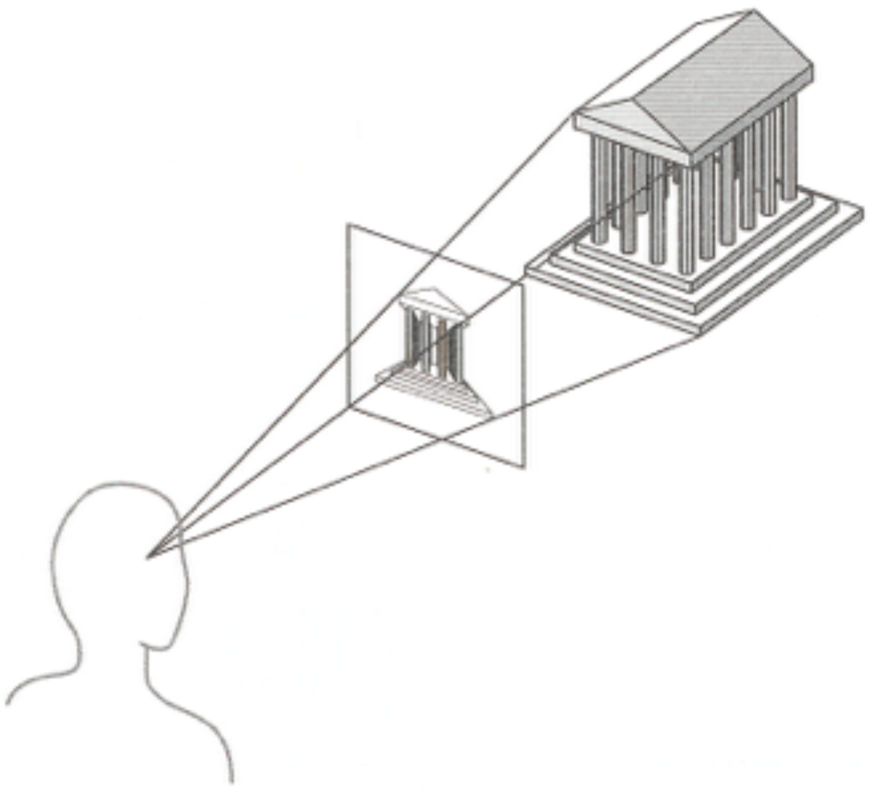
- This matrix is T^{-1} so we invert it to get T … easy!

Viewing Transformations

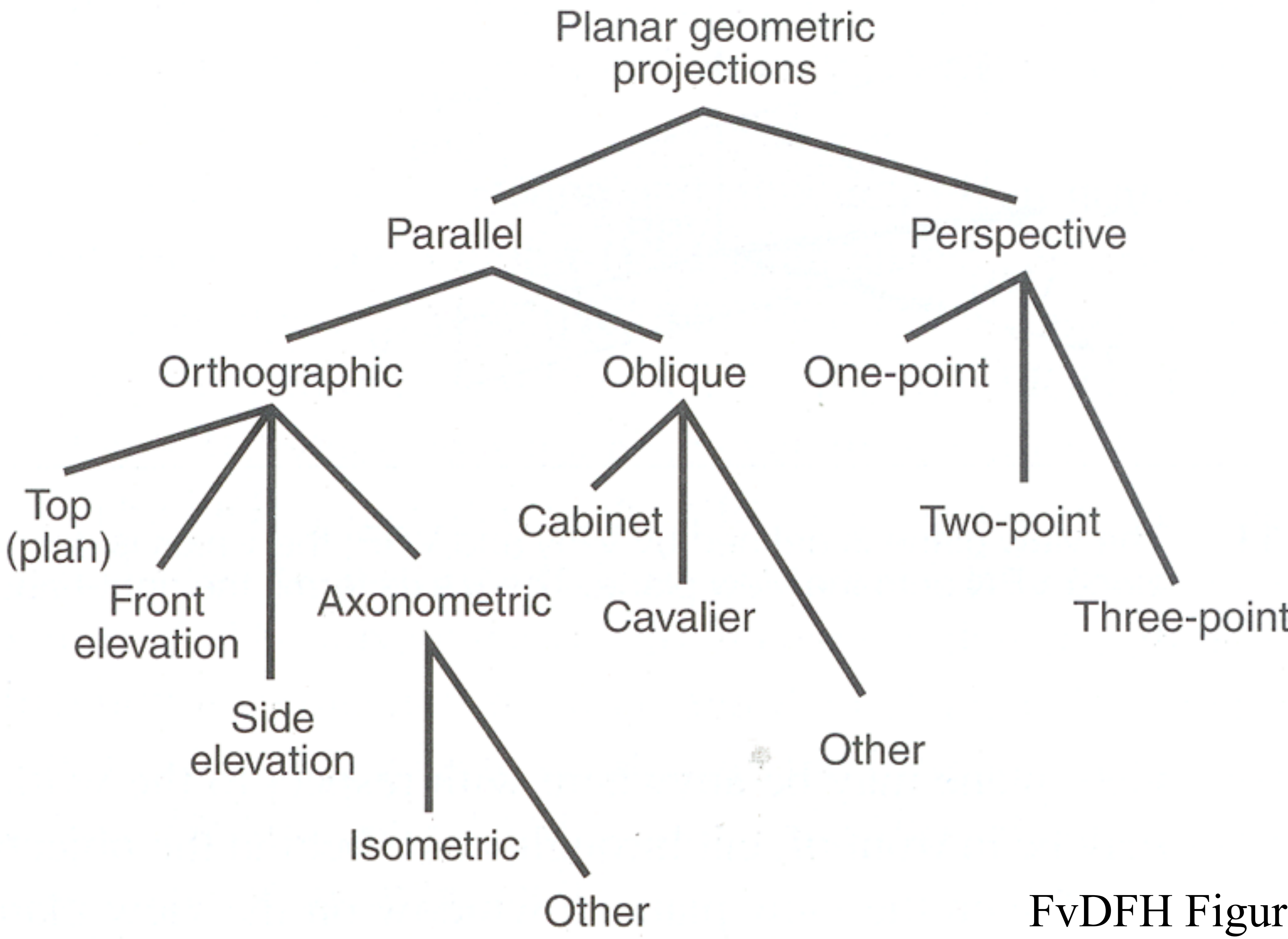


Projection

- General definition:
  - A linear transformation of points in *n*-space to *m*-space ($m<n$)
- In computer graphics:
  - Map 3D camera coordinates to 2D screen coordinates



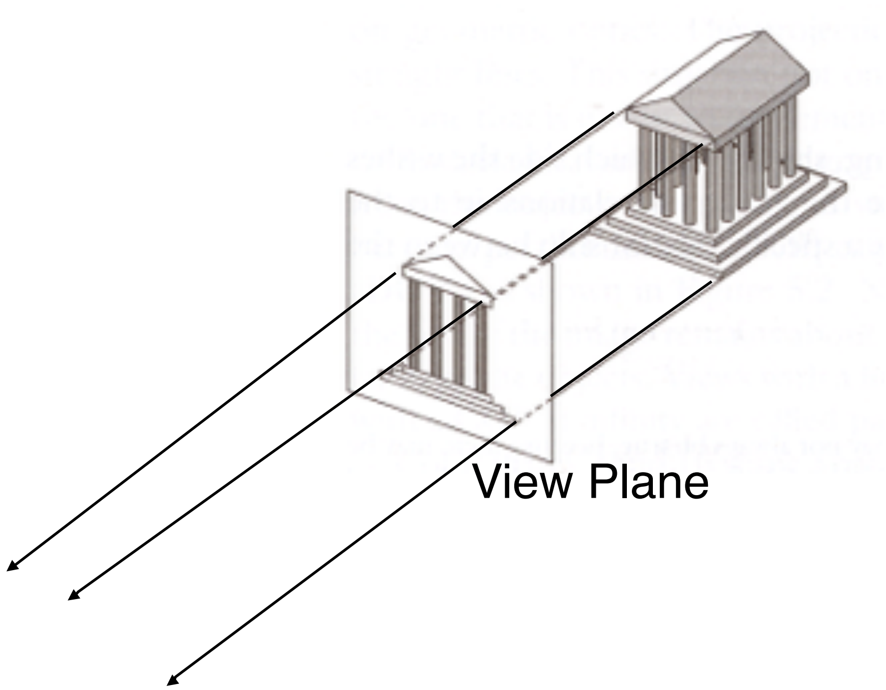
Taxonomy of Projections

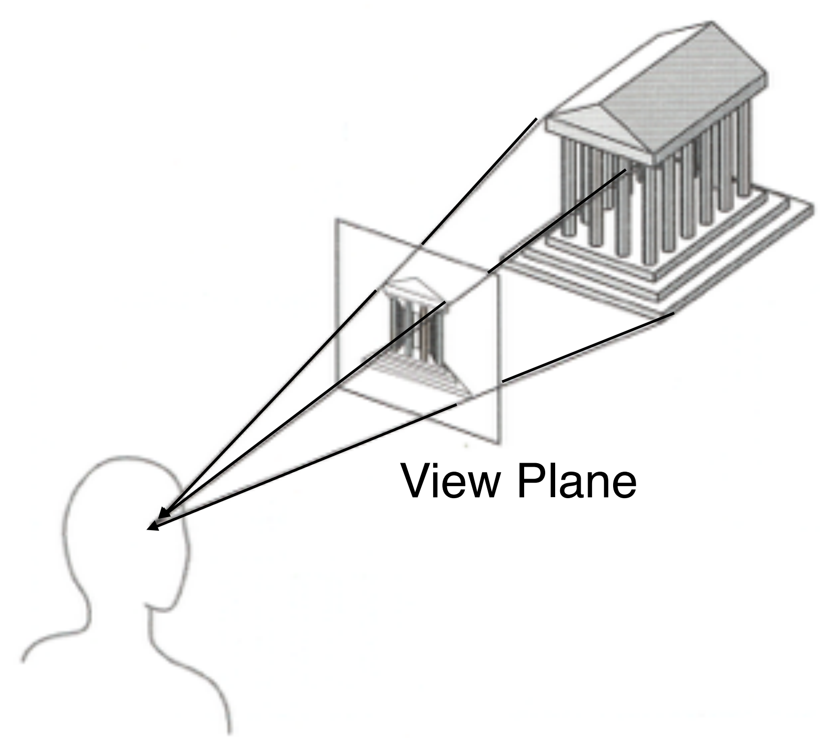


FvDFH Figure 6.13

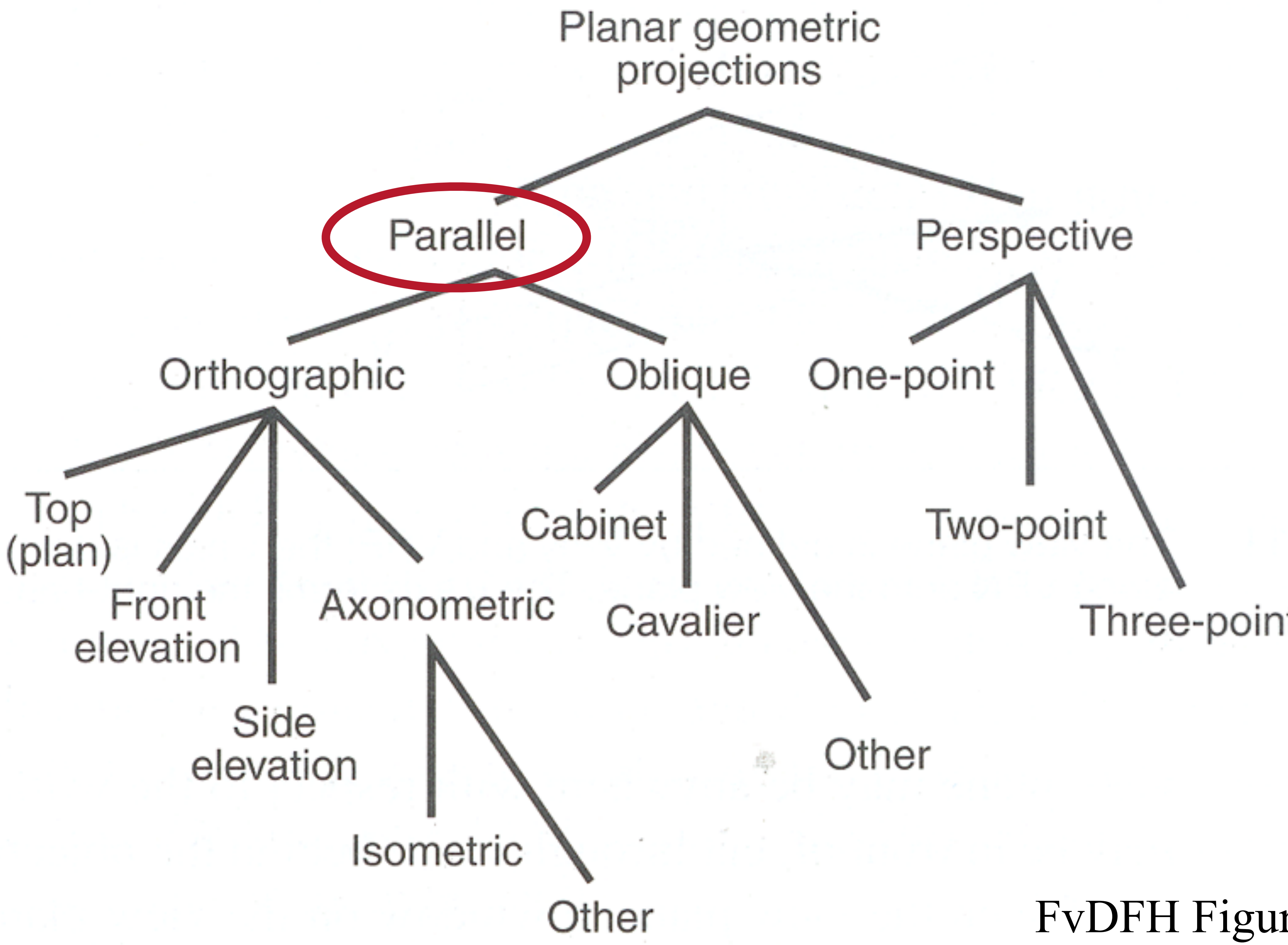
Projection

- Two general classes of projections, both of which shoot rays from the scene, through the view plane:
  - **o**Parallel Projection:
    - »Rays converge at a point at infinity and are parallel
  - **o**Perspective “Projection”:
    - »Rays converge at a finite point, giving rise to perspective distortion





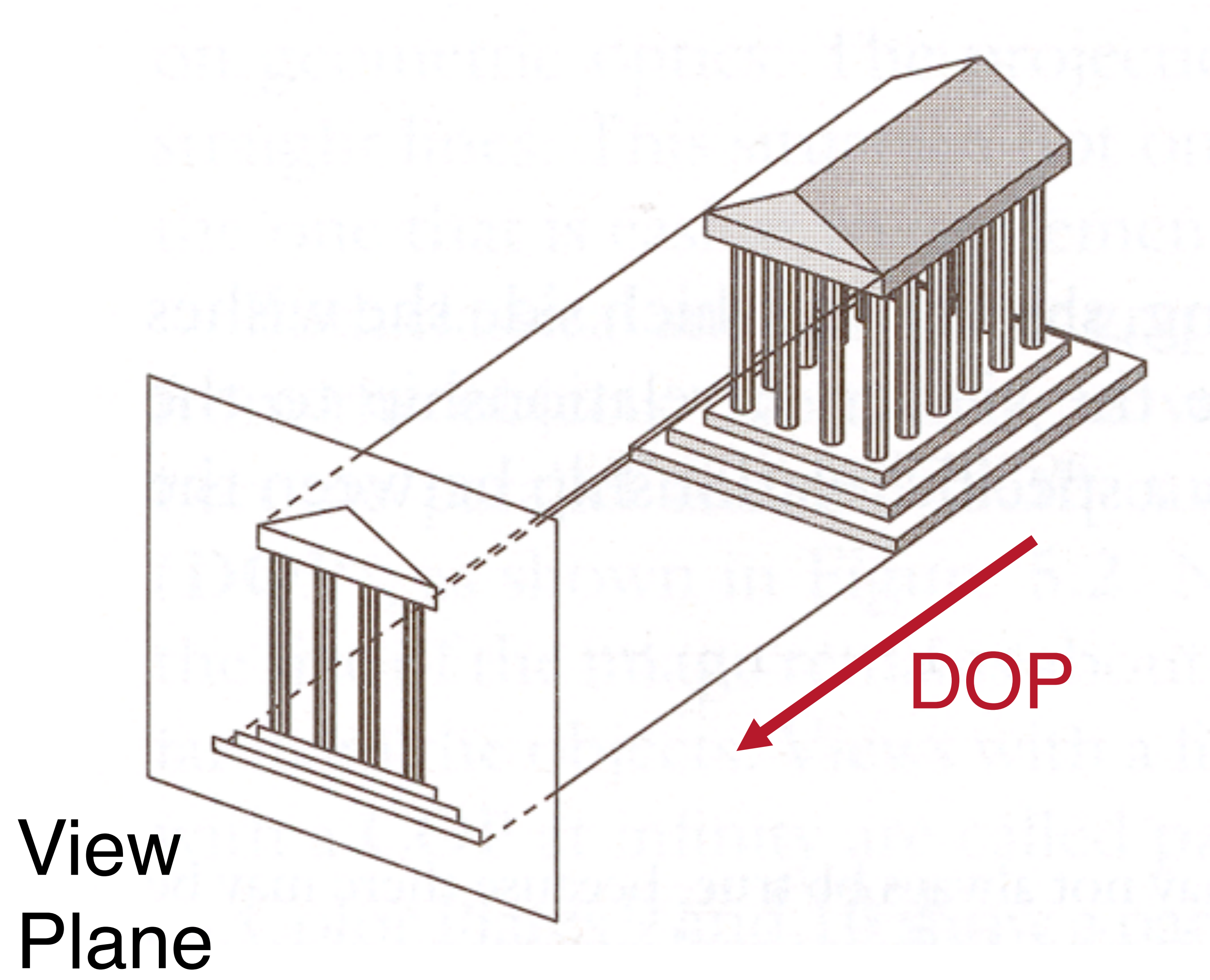
Taxonomy of Projections



FvDFH Figure 6.13

Parallel Projection

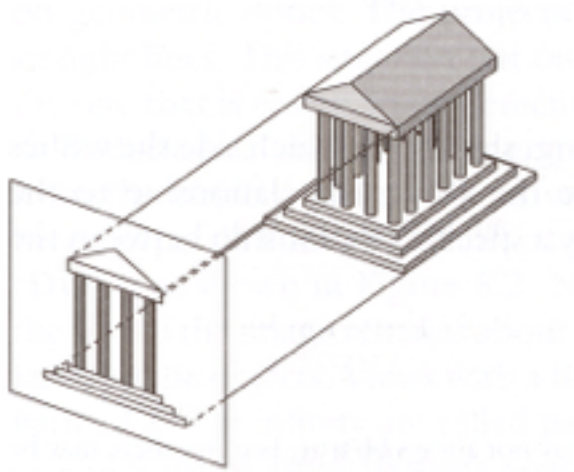
- Center of projection is at infinity
  - Direction of projection (DOP) same for all points



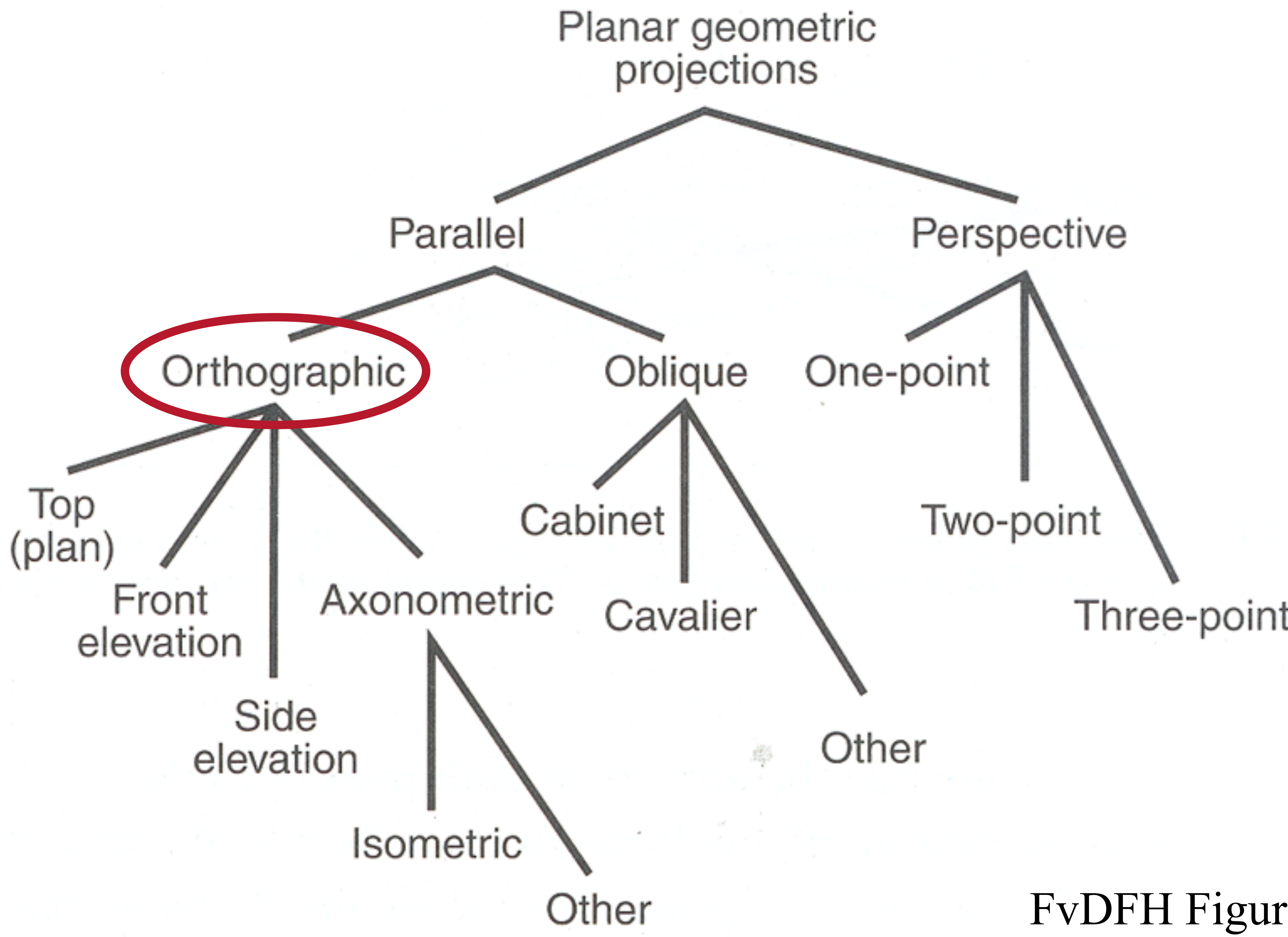
Angel Figure 5.4

Parallel Projection

- Parallel lines remain parallel
- Relative proportions of objects preserved
- Angles are not preserved
- Less realistic looking
    - Far away objects don’t get smaller



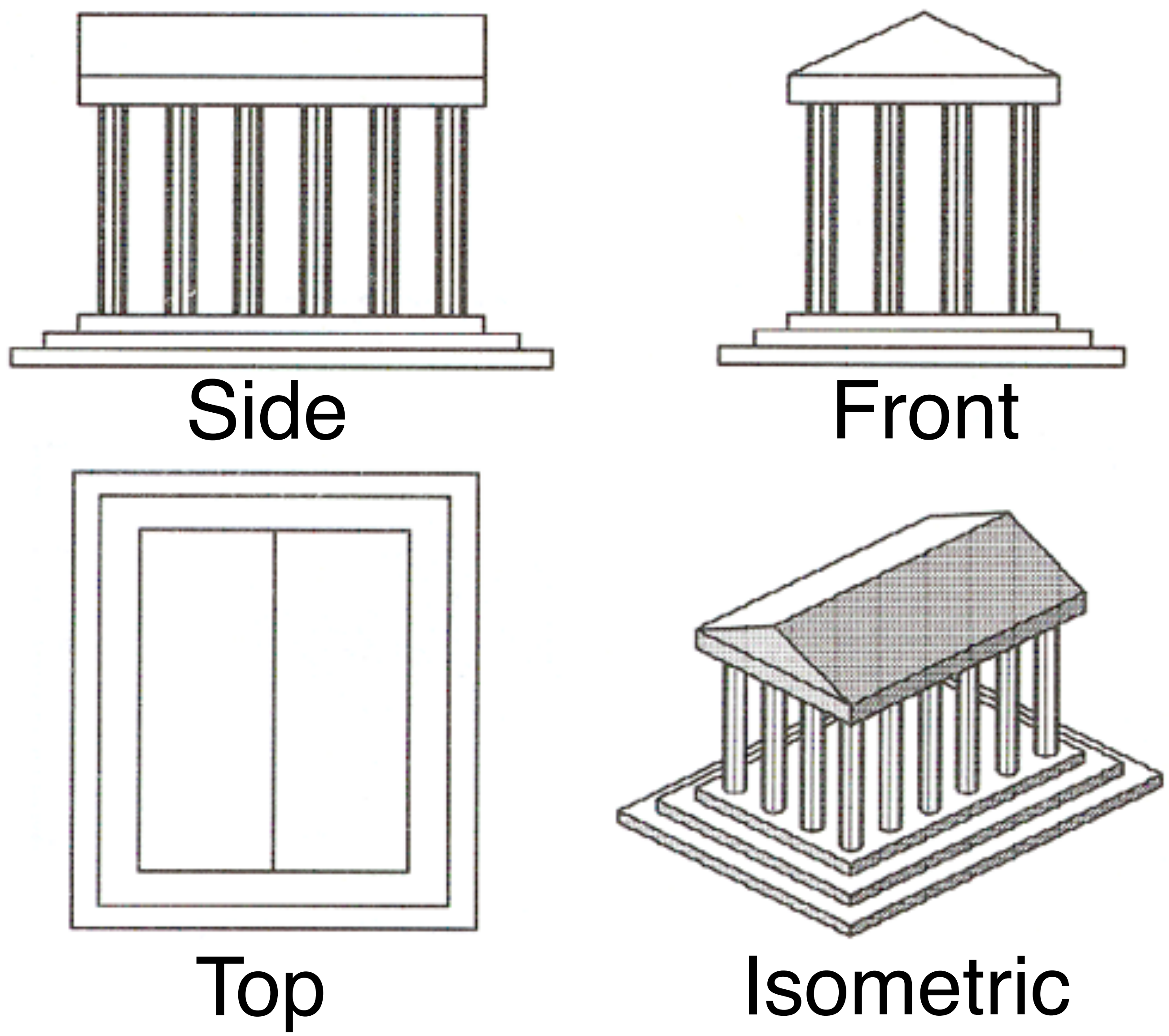
Taxonomy of Projections



FvDFH Figure 6.13

Orthographic Projections

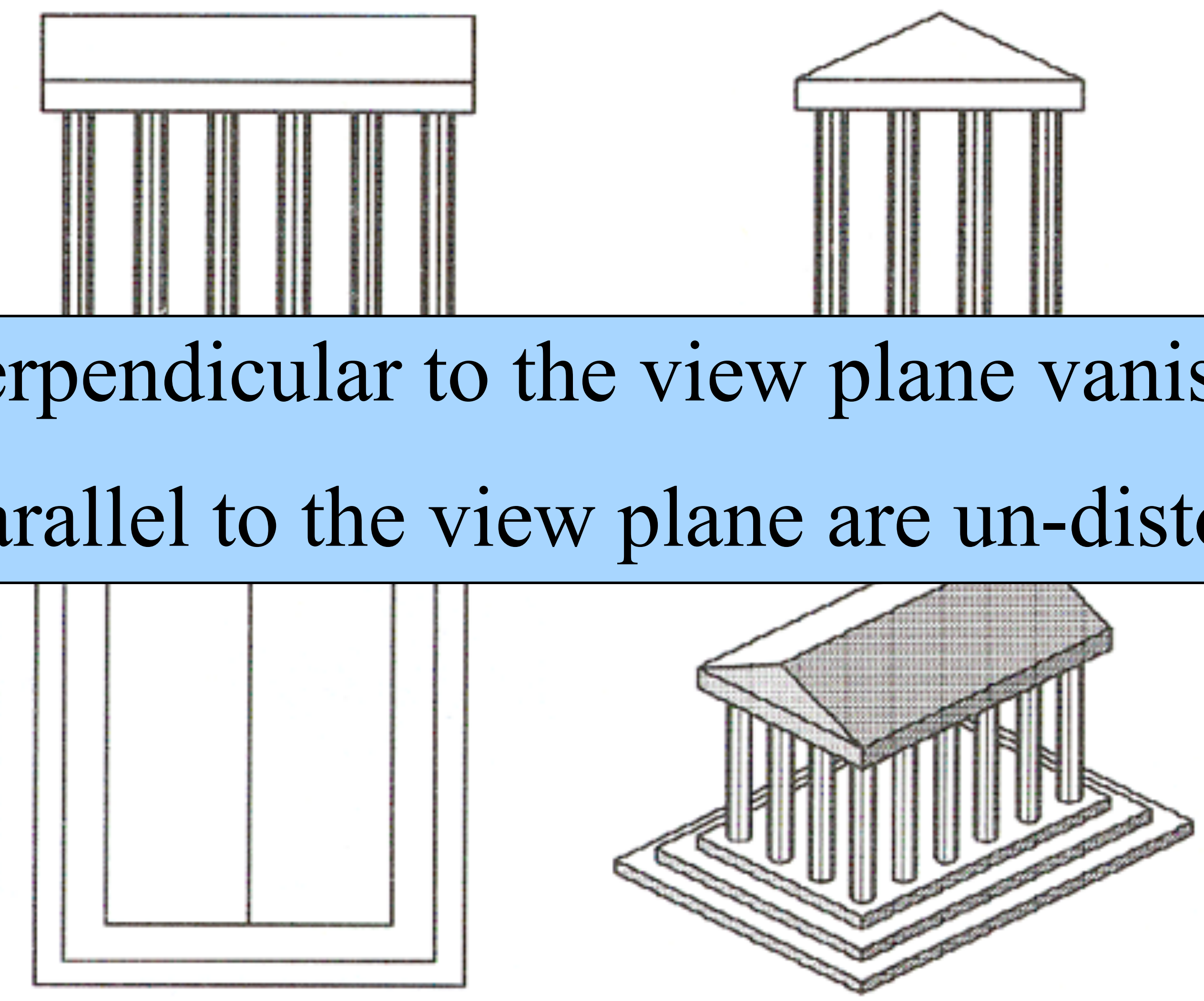
- DOP perpendicular to view plane



Angel Figure 5.5

Orthographic Projections

- DOP perpendicular to view plane

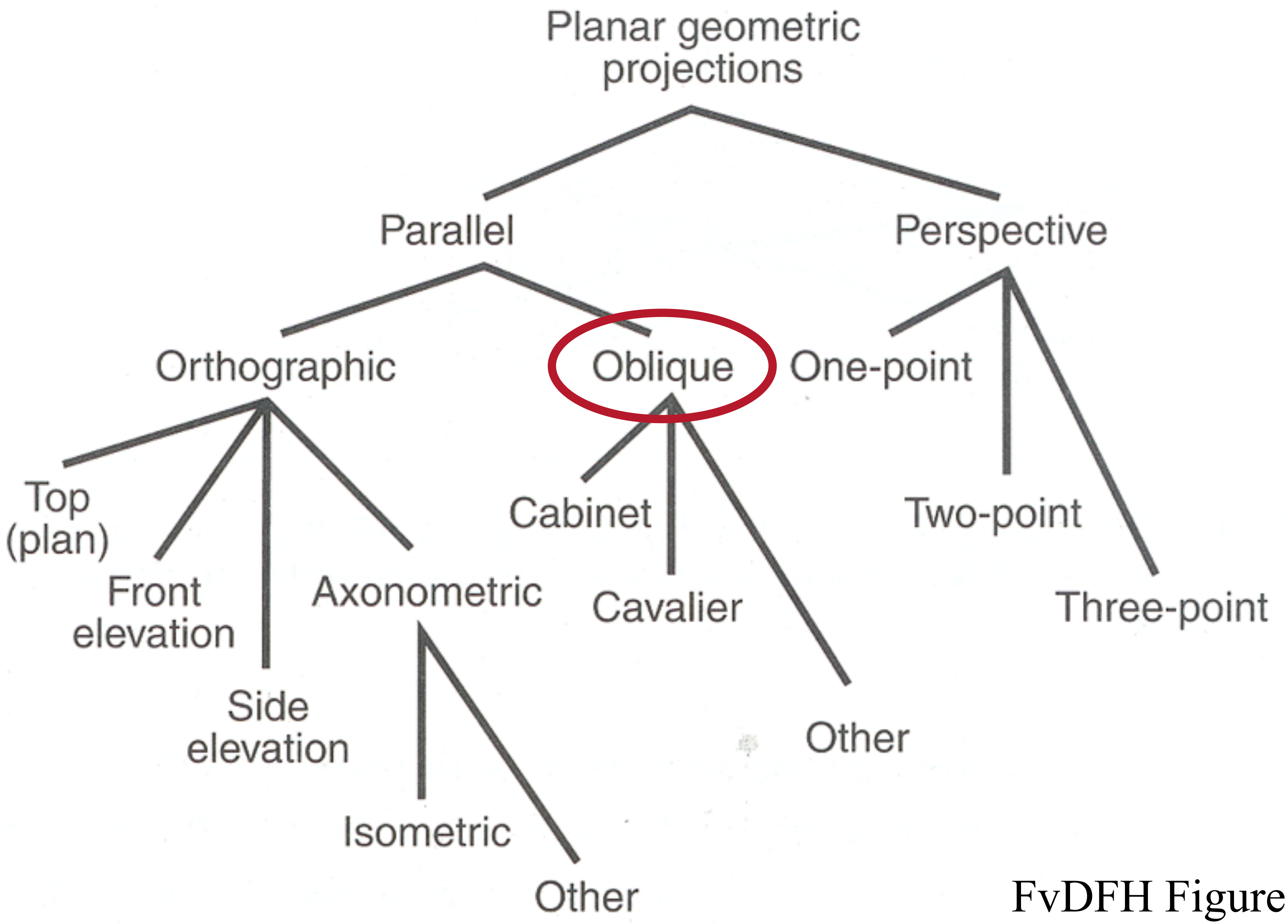


- Lines perpendicular to the view plane vanish
- Faces parallel to the view plane are un-distorted.

Top Isometric

Angel Figure 5.5

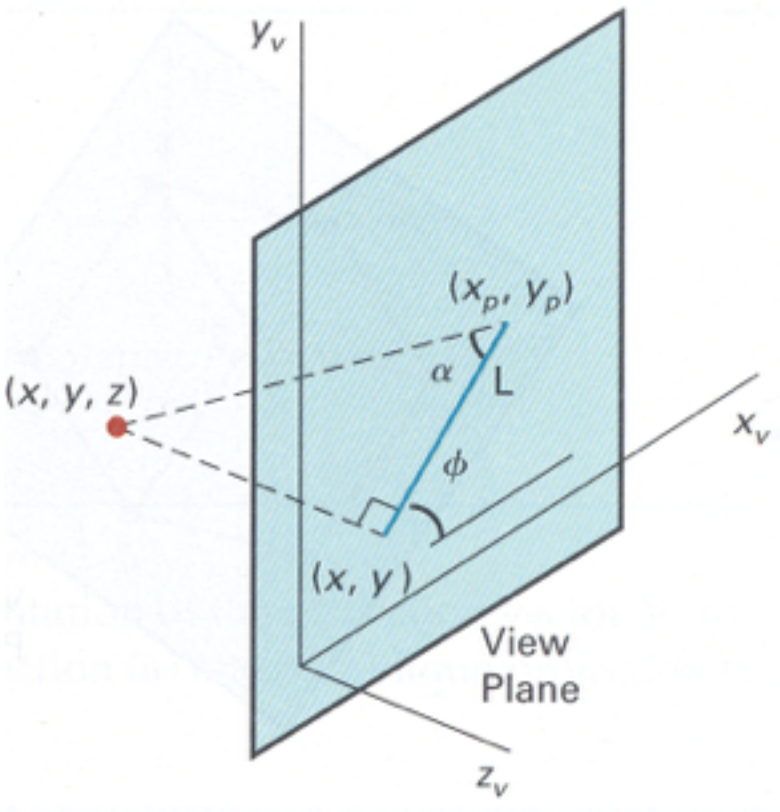
Taxonomy of Projections

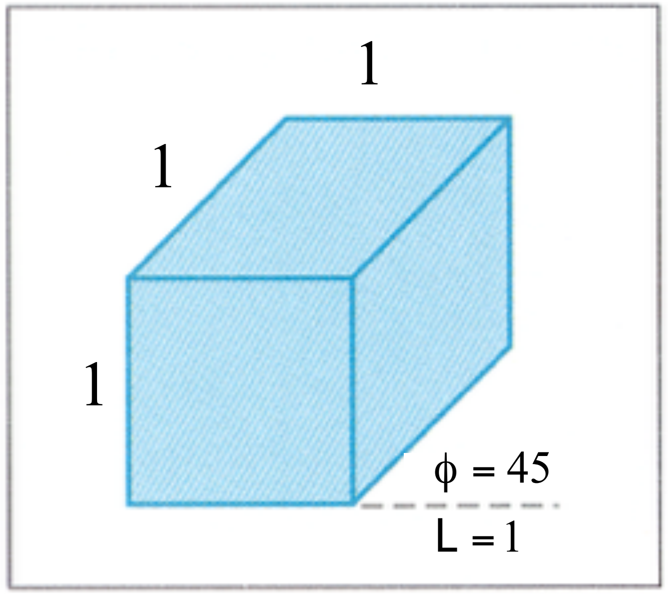


FvDFH Figure 6.13

Oblique Projections

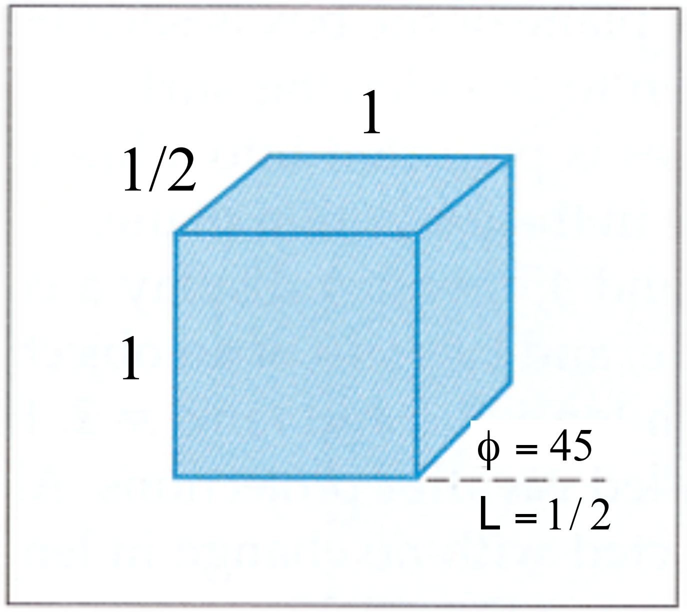
- DOP not perpendicular to view plane





Cavalier

(DOP $\alpha = 45^o$)



Cabinet

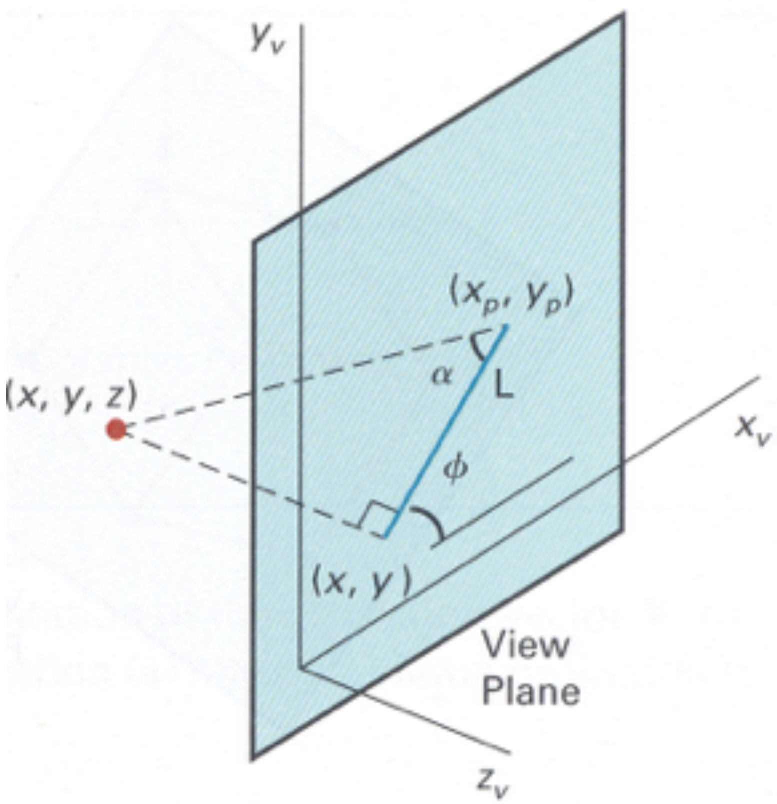
(DOP $\alpha = 63.4^o$)

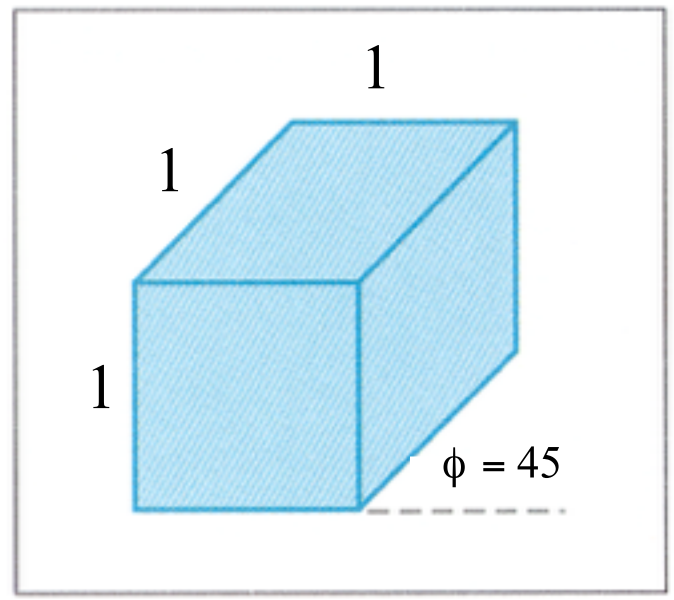
- ϕ describes the angle of the projection of the view plane's normal
- L represents the scale factor applied to the view plane's normal

H&B Figure 12.21

Parallel Projection Matrix

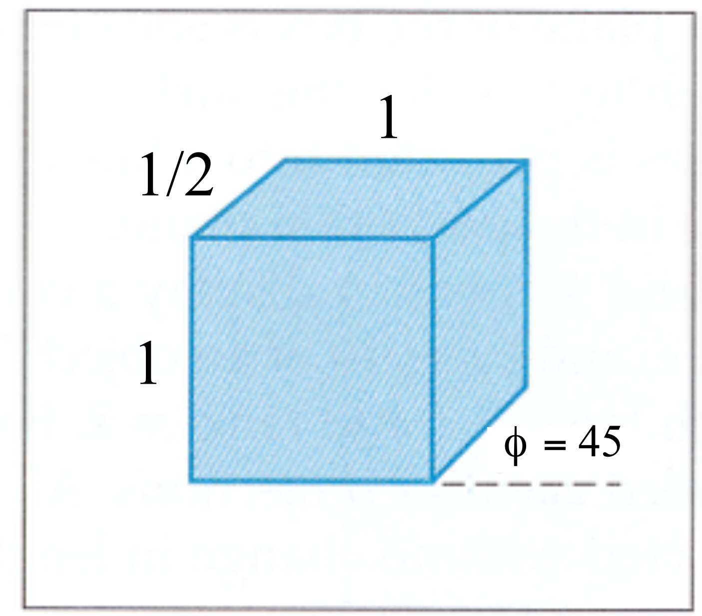
- General parallel projection transformation:





Cavalier

(DOP $\alpha = 45^o$)



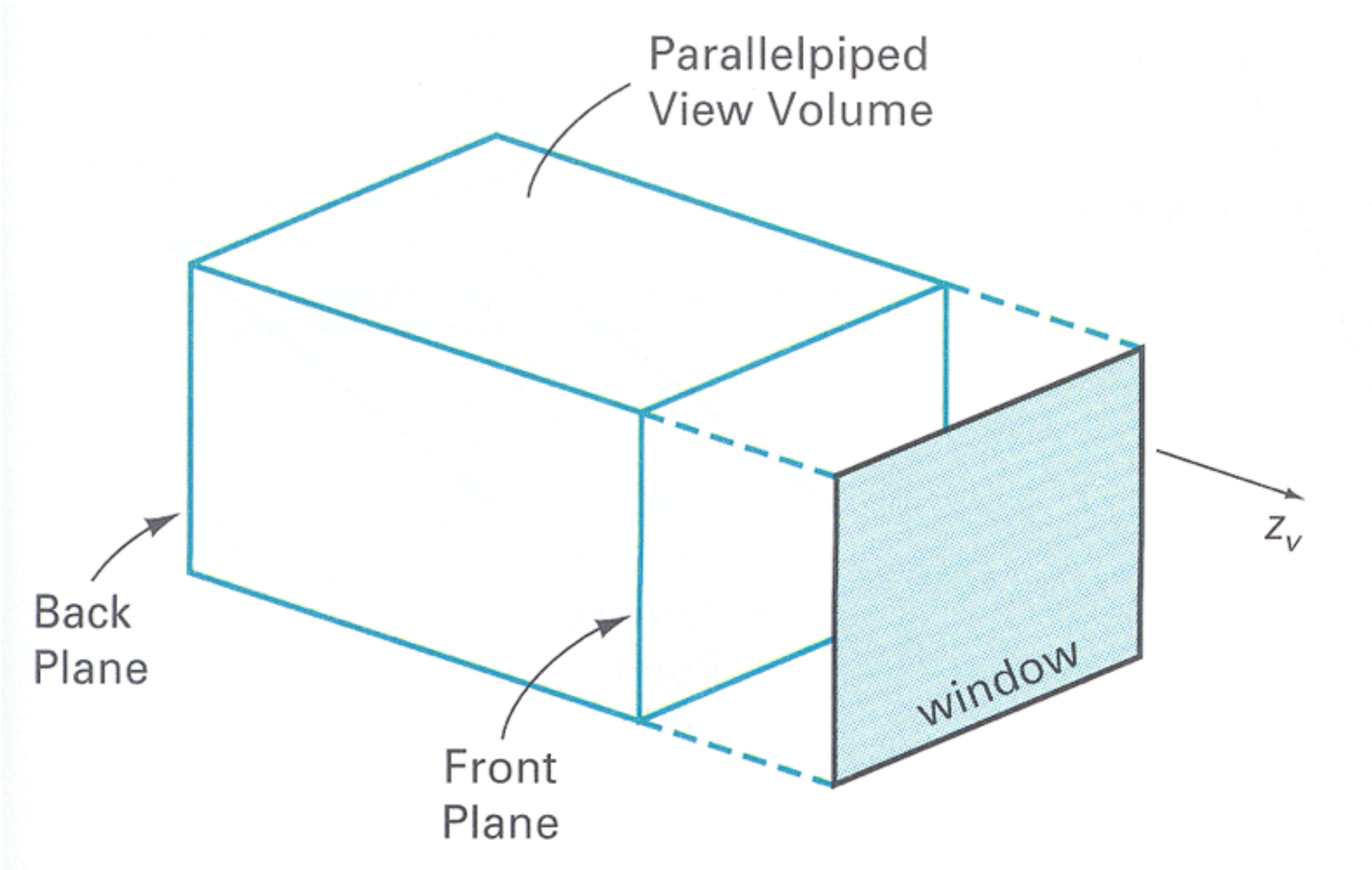
Cabinet

(DOP $\alpha = 63.4^o$)

$$\begin{bmatrix} x_p \\ y_p \end{bmatrix} = \begin{bmatrix} 1 & 0 & L\cos\phi \\ 0 & 1 & L\sin\phi \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

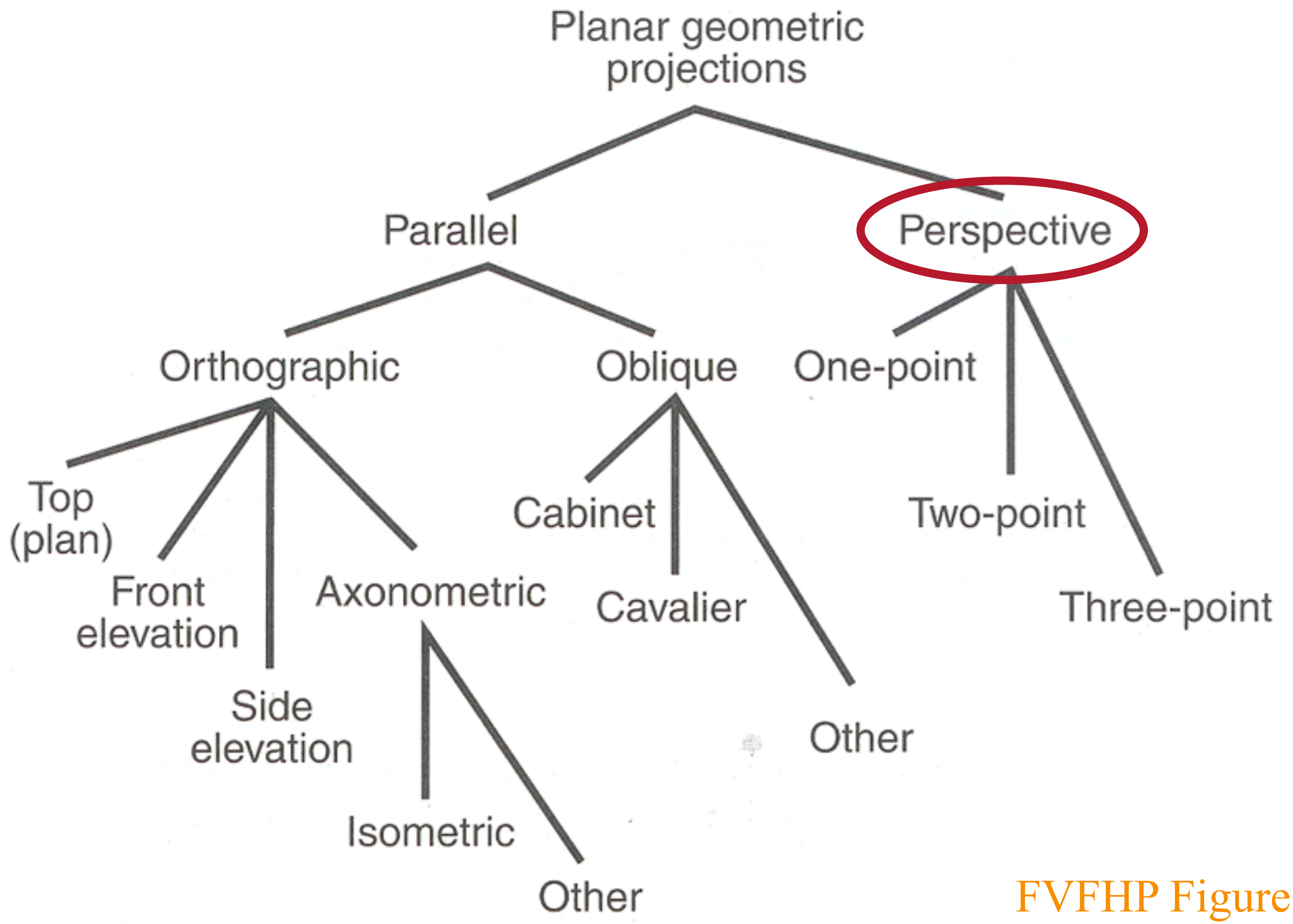
H&B Figure 12.21

Parallel Projection View Volume



H&B Figure 12.30

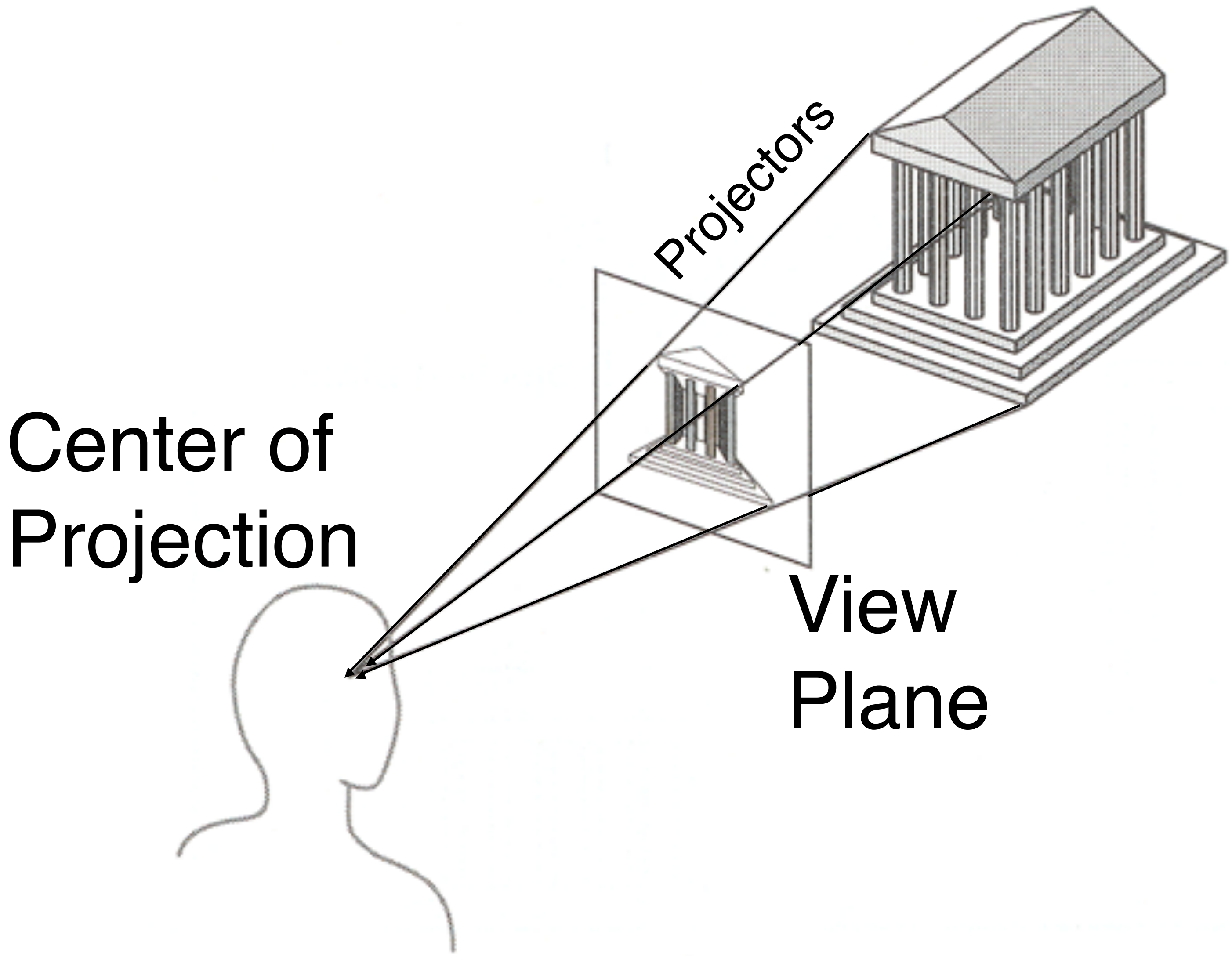
Taxonomy of Projections



FVFHP Figure 6.10

Perspective "Projection"

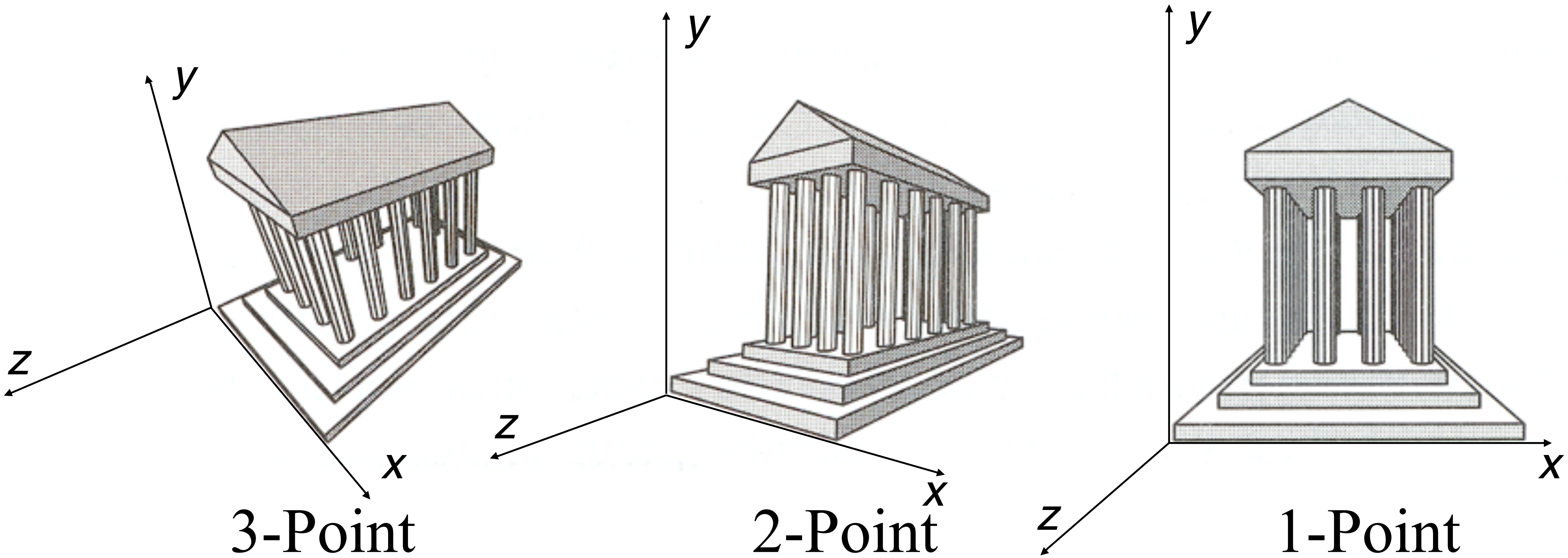
- Map points onto "view plane" along "projectors" emanating from "center of projection" (COP)



Angel Figure 5.9

Perspective Projection

- How many vanishing points?



3-Point Perspective

2-Point Perspective

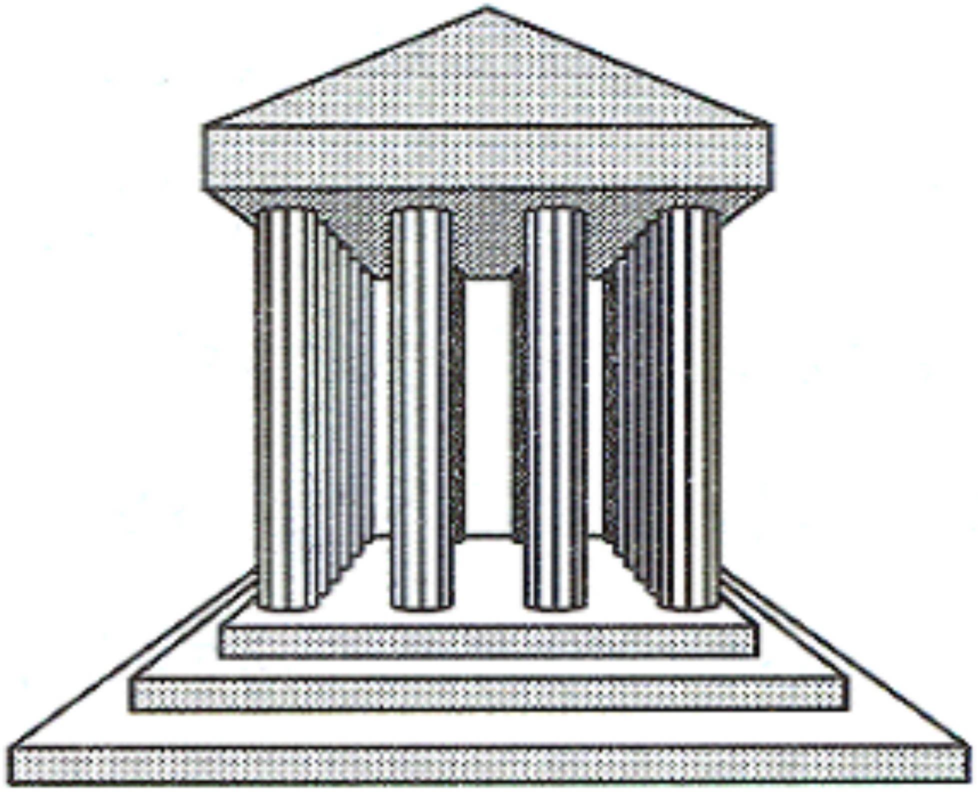
1-Point Perspective

Number of vanishing points determined by number of axes parallel to the view plane

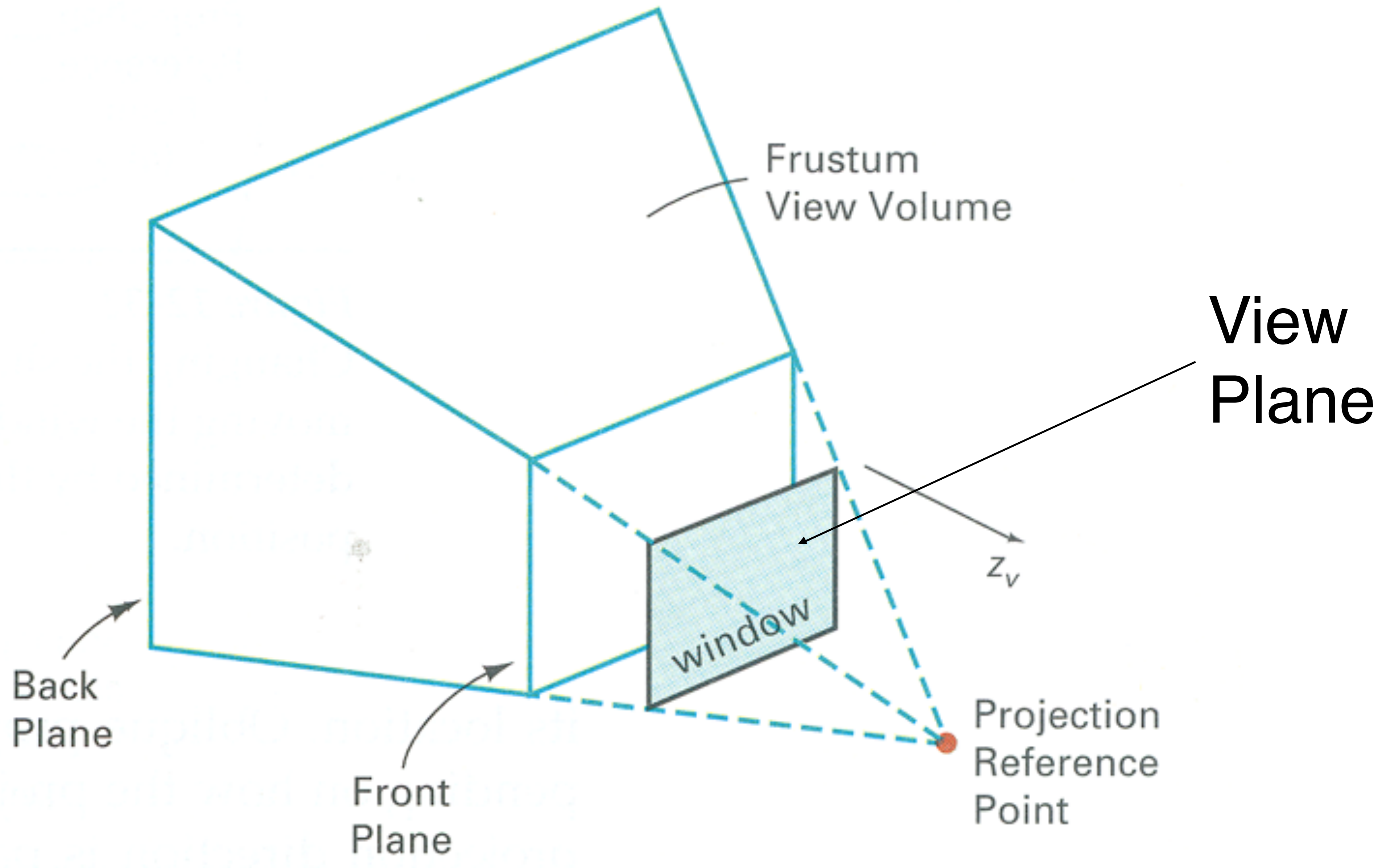
Angel Figure 5.10

Perspective Projection

- Perspective "projection" is not really a projection because it is not a linear map from 3D to 2D.
  - Parallel lines do not remain parallel!



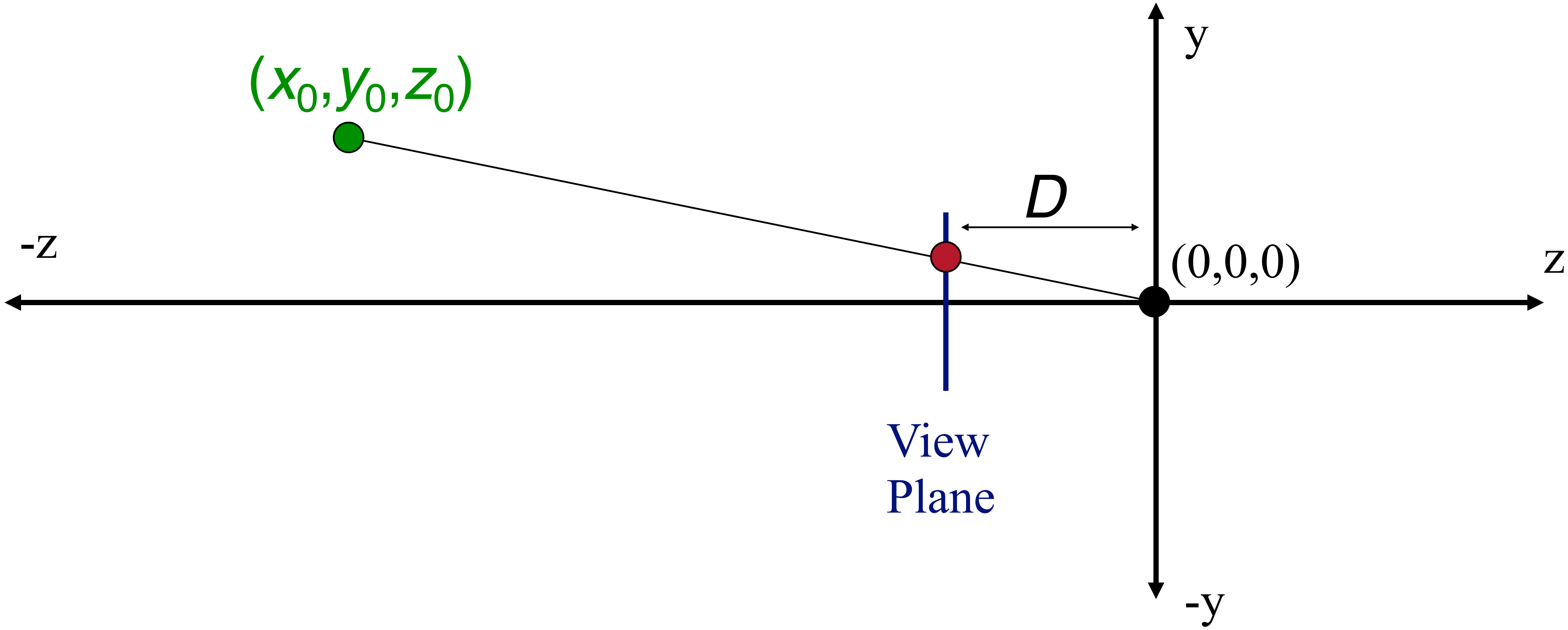
Perspective Projection View Volume



H&B Figure 12.30

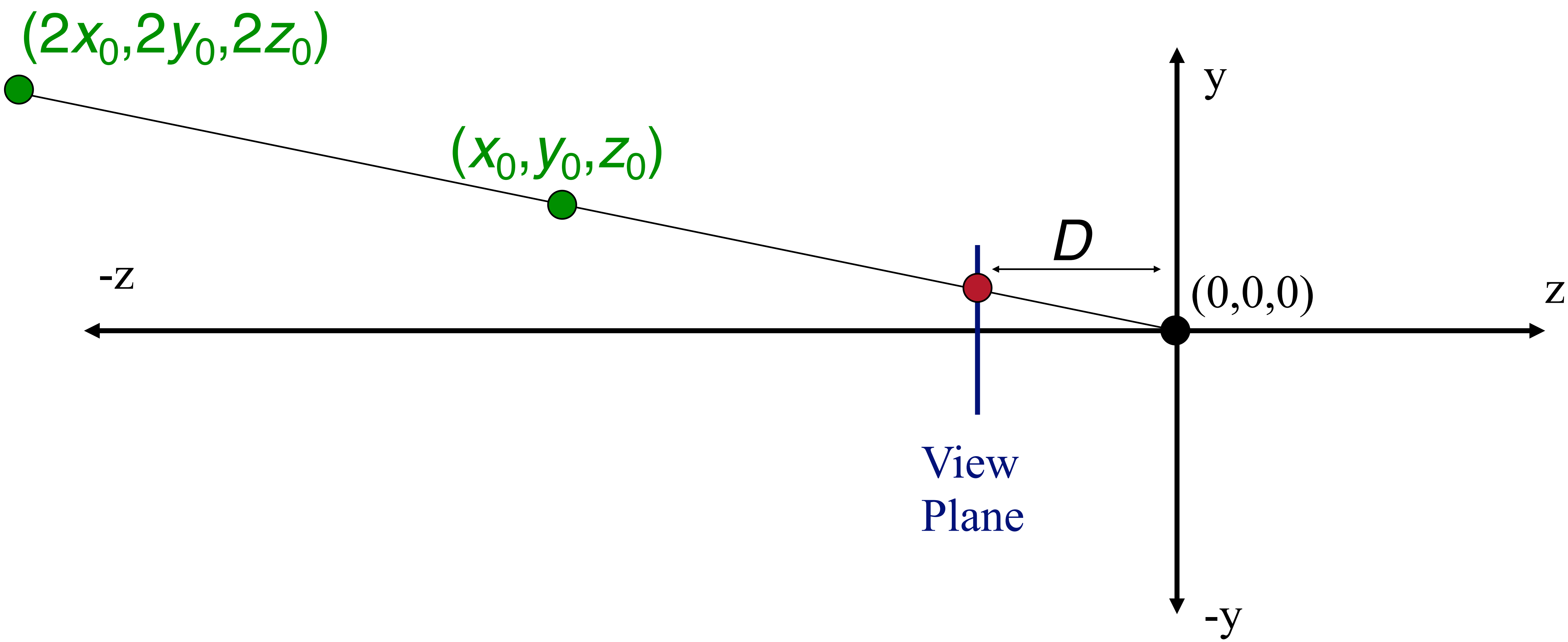
Perspective Projection

- What are the coordinates of the point resulting from projection of (x_0,y_0,z_0) onto the view plane at a distance of D along the z-axis?



Perspective Projection

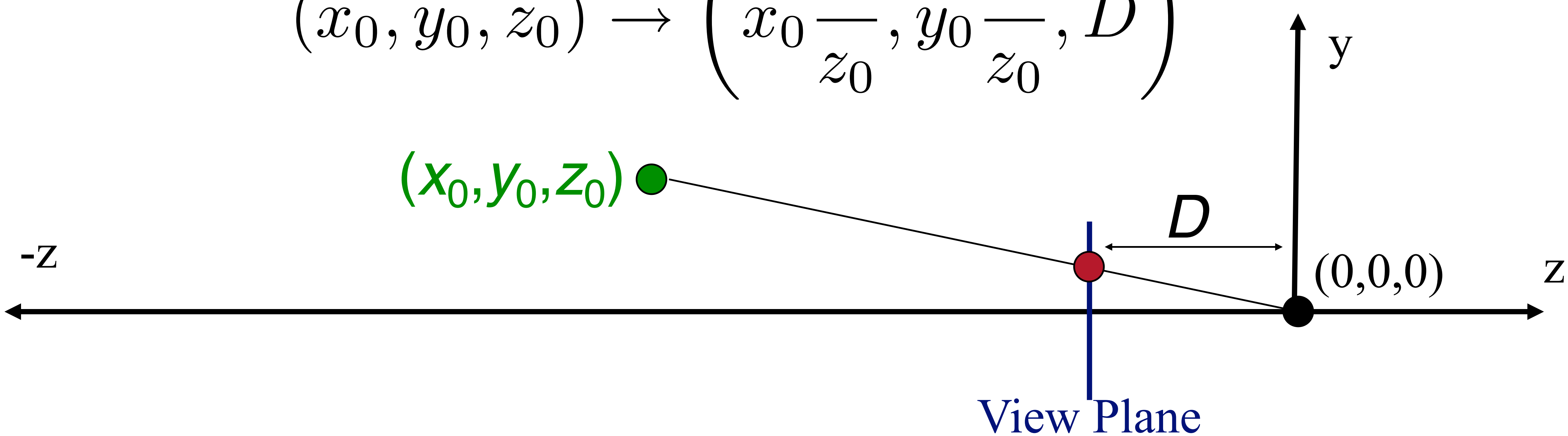
- Use the fact that for any point (x_0,y_0,z_0) and any scalar α, the points (x_0,y_0,z_0) and $(\alpha x_0, \alpha y_0, \alpha z_0)$ map to the same location:



Perspective Projection

- Use the fact that for any point (x_0,y_0,z_0) and any scalar α, the points (x_0,y_0,z_0) and $(\alpha x_0, \alpha y_0, \alpha z_0)$ map to the same location.
- Since we want the position of the point on the line that intersect the image plane at a distance of D along the z-axis:

$$(x_0, y_0, z_0) \to \left(x_0 \frac{D}{z_0}, y_0 \frac{D}{z_0}, D \right)$$



Perspective Projection Matrix

- 4x4 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

- 4x4 matrix representation?

$$x_s = x_c D / z_c$$
$$y_s = y_c D / z_c$$
$$z_s = D$$
$$w_s = 1$$

We want to divide by the *z* coordinate. How do we do that with a 4x4 matrix?

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

Perspective Projection Matrix

- 4x4 matrix representation?

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

We want to divide by the *z* coordinate. How do we do that with a 4x4 matrix?

Recall that in homogenous coordinates:
(x, y, z, w) = (x/w, y/w, z/w, 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

- 4x4 matrix representation?

$$x_s = x_c D / z_c$$
$$y_s = y_c D / z_c$$
$$z_s = D$$
$$w_s = 1$$

$$\left(\frac{x_c D}{z_c}, \frac{y_c D}{z_c}, D, 1 \right)$$

$$\updownarrow$$

$$\left(x_c, y_c, z_c, \frac{z_c}{D} \right)$$

We want to divide by the *z* coordinate. How do we do that with a 4x4 matrix?

Recall that in homogenous coordinates:
(x, y, z, w) = (x/w, y/w, z/w, 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

- 4x4 matrix representation?

$$x_s = x_c D / z_c$$
$$y_s = y_c D / z_c$$
$$z_s = D$$
$$w_s = 1$$

$$\left(\frac{x_c D}{z_c}, \frac{y_c D}{z_c}, D, 1 \right)$$

$$\Updownarrow$$

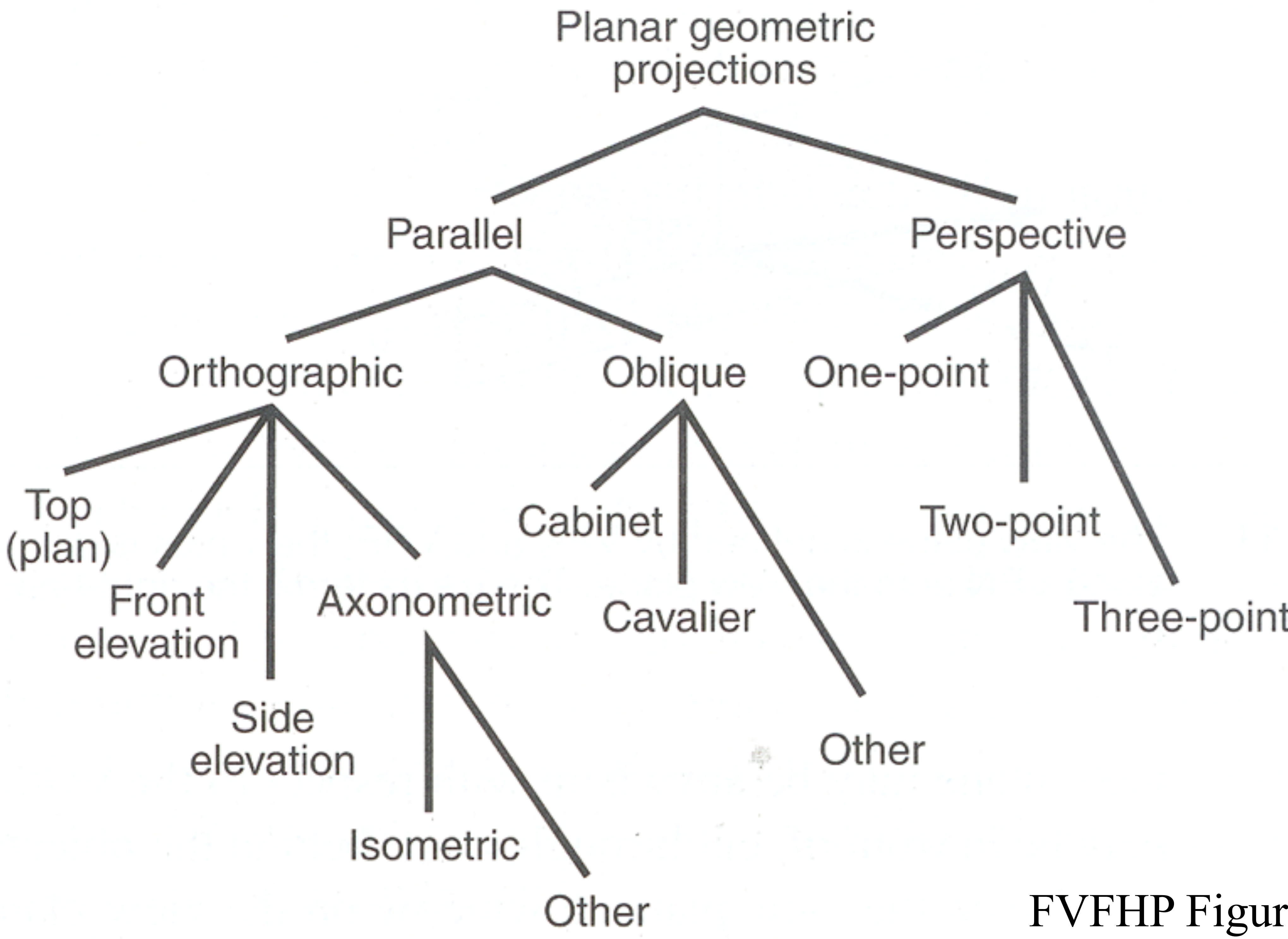
$$\left(x_c, y_c, z_c, \frac{z_c}{D} \right)$$

We want to divide by the *z* coordinate. How do we do that with a 4x4 matrix?

Recall that in homogenous coordinates:
(x, y, z, w) = (x/w, y/w, z/w, 1)

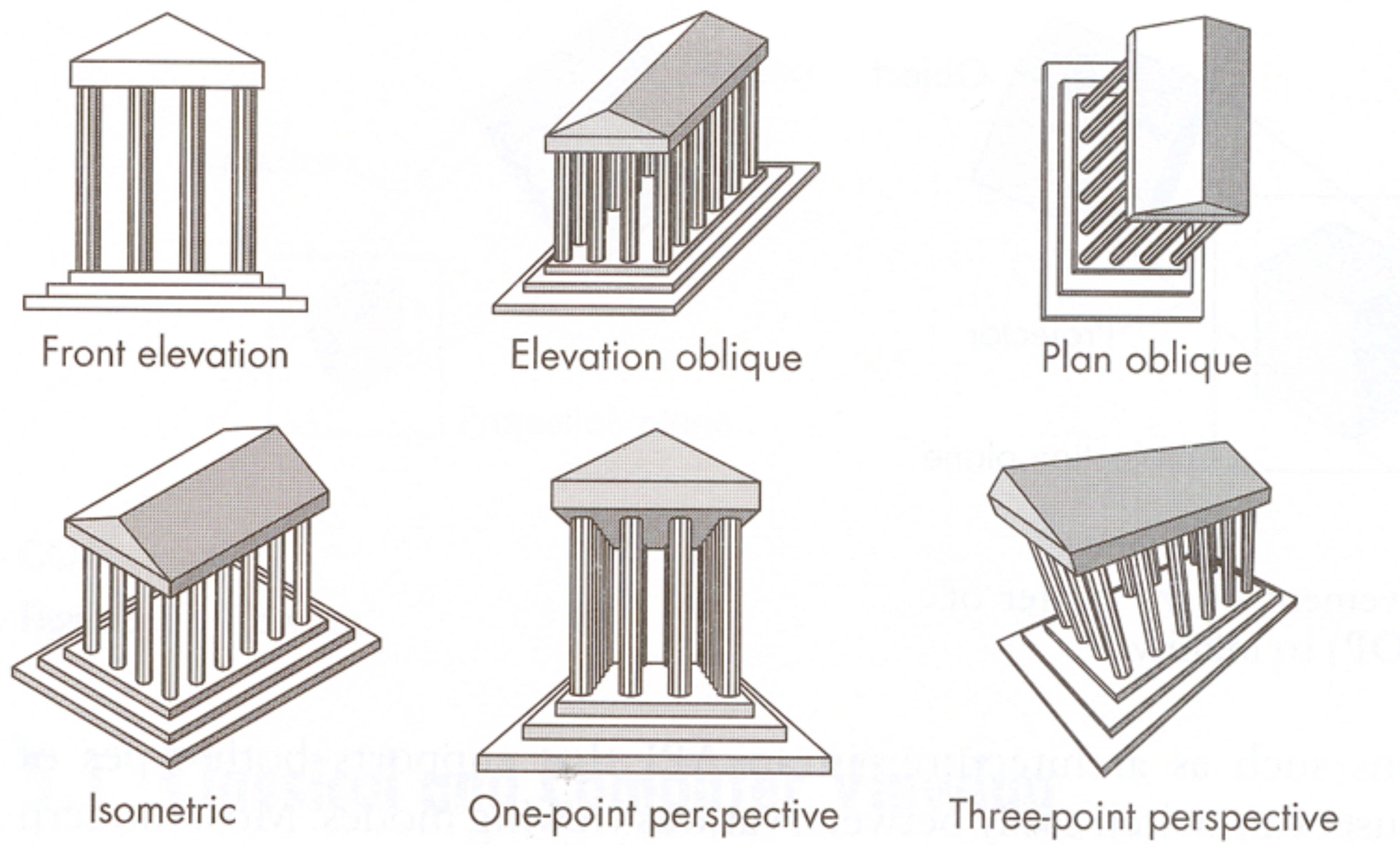
$$\begin{bmatrix} x_s \\ y_s \\ z_s \\ w_s \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/D & 0 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix}$$

Taxonomy of Projections



FVFHP Figure 6.10

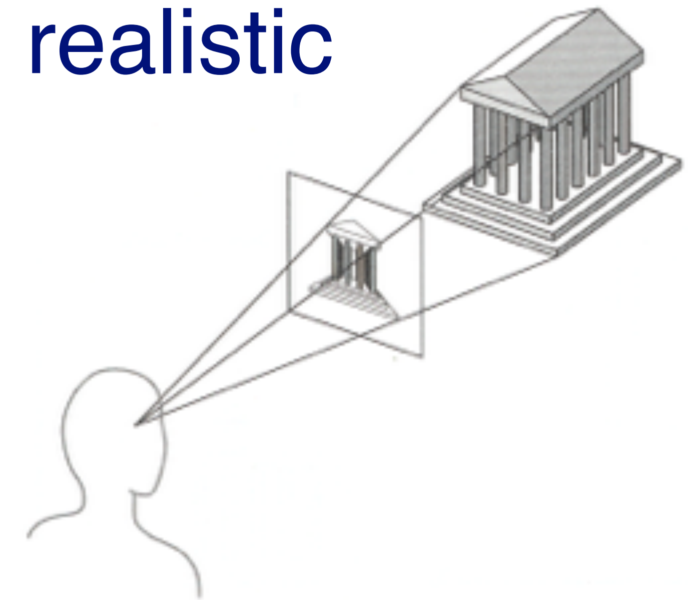
Classical Projections



Angel Figure 5.3

Perspective vs. Parallel

- Perspective projection
  - +Size varies inversely with distance - looks realistic
  - –Distance and angles are not preserved
  - –Only parallel lines that are parallel to the view plane remain parallel



- Parallel projection
  - +Good for exact measurements
  - +Parallel lines remain parallel
  - +Angles are preserved on faces parallel to the view plane
  - –Less realistic looking