Viewing in 3D

Taking a Real Photograph

- Arrange objects
- Position and point the camera
- Choose a lens, set the zoom
- Take a picture
- Enlarge and crop to get a print

Taking a Virtual Photograph

- Arrange objects
  - Apply modeling transformations to objects: change from *object coordinates* to *world coordinates*
- Position and point the camera
  - Position, point, and orient the virtual camera: define a transformation from world to *eye coordinates*
- Choose a lens, set the zoom
  - Specify a view volume: define a perspective transformation that transforms eye coordinates to canonical normalized viewing space (*clip coordinates*)

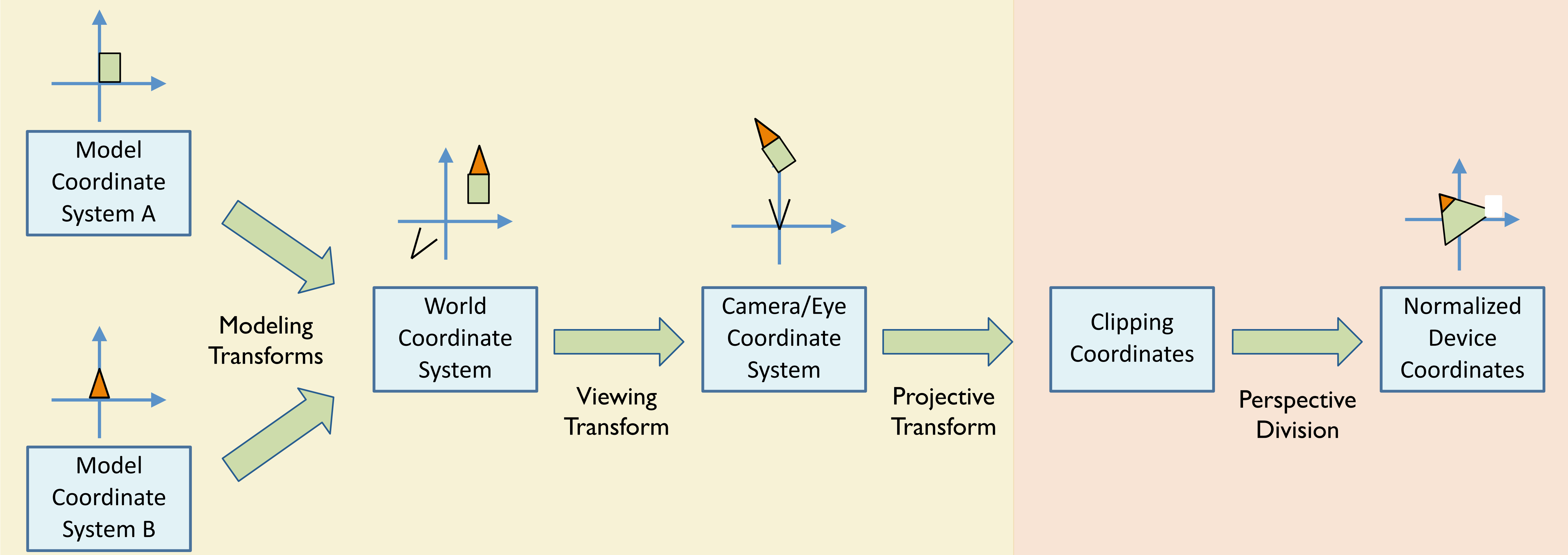
Taking a Virtual Photograph

- Take a picture
  - Project objects by applying the projective transformation followed by a *perspective divide*. The result is *normalized device coordinates*.
- Enlarge and crop to get a print
  - Apply viewport transformation to obtain actual *window coordinates*.

Viewing in 3D

- How to transform 3D world coordinates to 2D display coordinates?
  - Projections
- How to specify which part of the 3D world is to be viewed?
  - Define a 3D viewing volume
- How to avoid displaying primitives outside the viewing volume?
  - Culling and Clipping

3D Viewing Pipeline



Object Units	World Units	World Units	Clipping Units	[-1,1]x[-1,1]
Initial modeling coordinates used by the application	Final/common modeling coordinates used by the application	Lighting/Shading takes place in these coordinates	These coordinates are designed to make culling/clipping easy	These coordinates are the input needed by the rasterizer

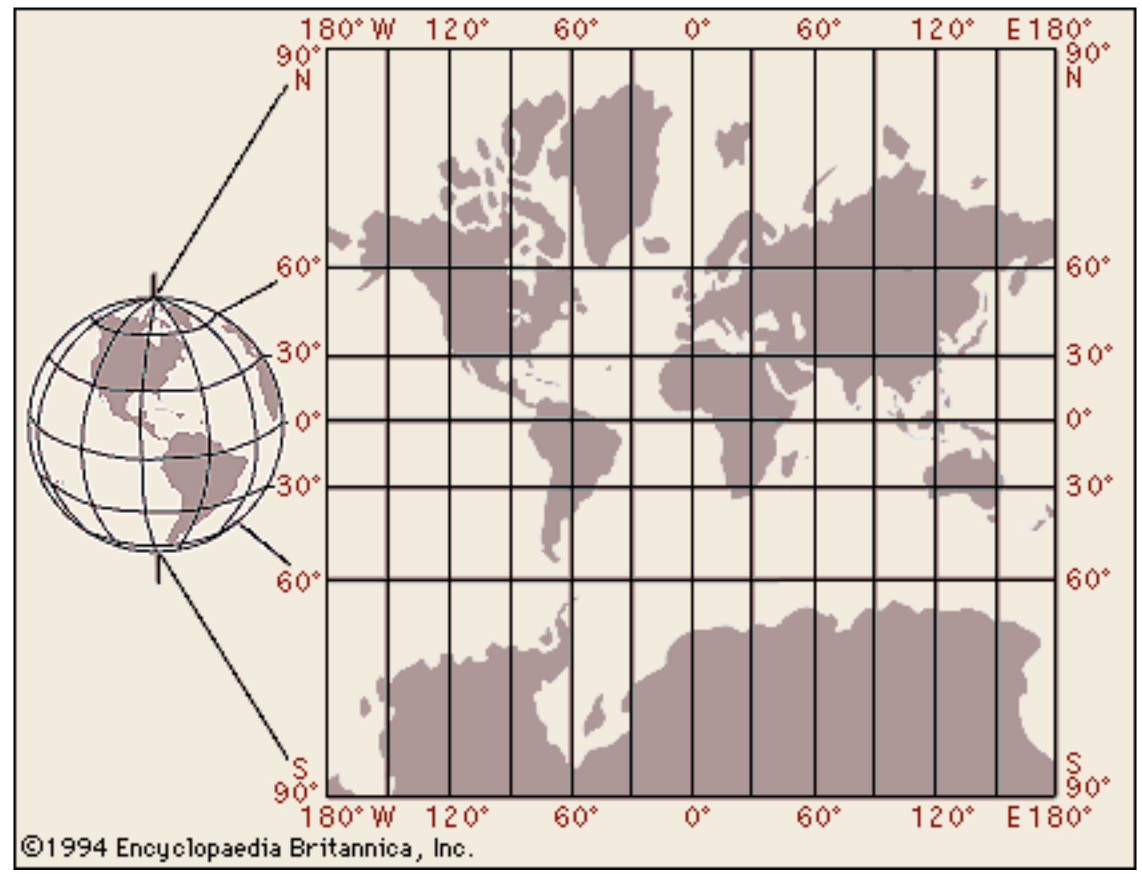
Projections

Planar Geometric Projections

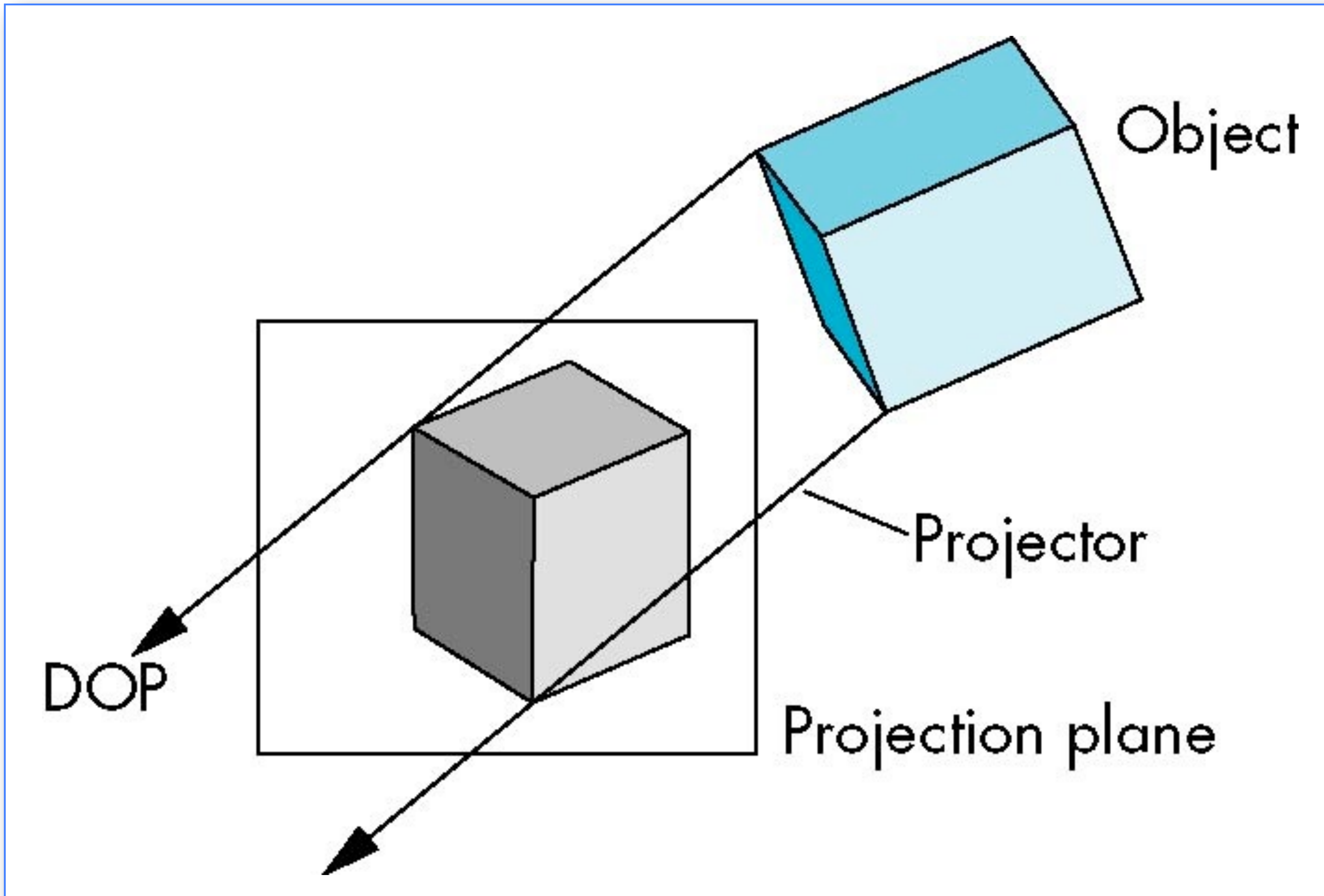
- A projection is formed by the intersection of certain ***lines*** (projectors) with a ***plane*** (the projection plane)
- Projectors are lines from the center of projection through each point on object
- Center of projection at infinity results in a ***parallel projection***
- A finite center of projection results in a ***perspective projection***

Non-Planar Projections

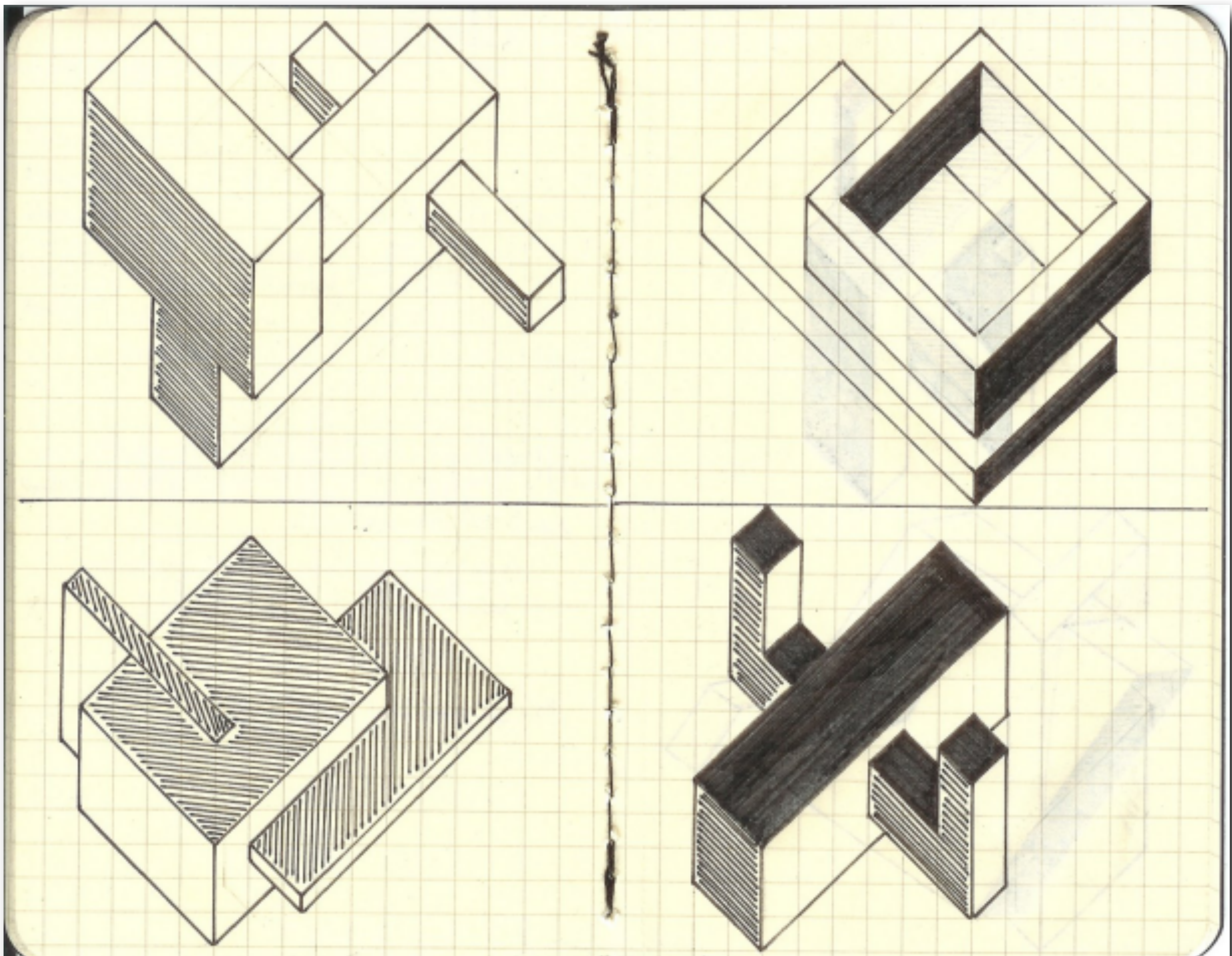
- Cartography (for example, Mercator):
- Fish-eye lenses



Parallel Projection

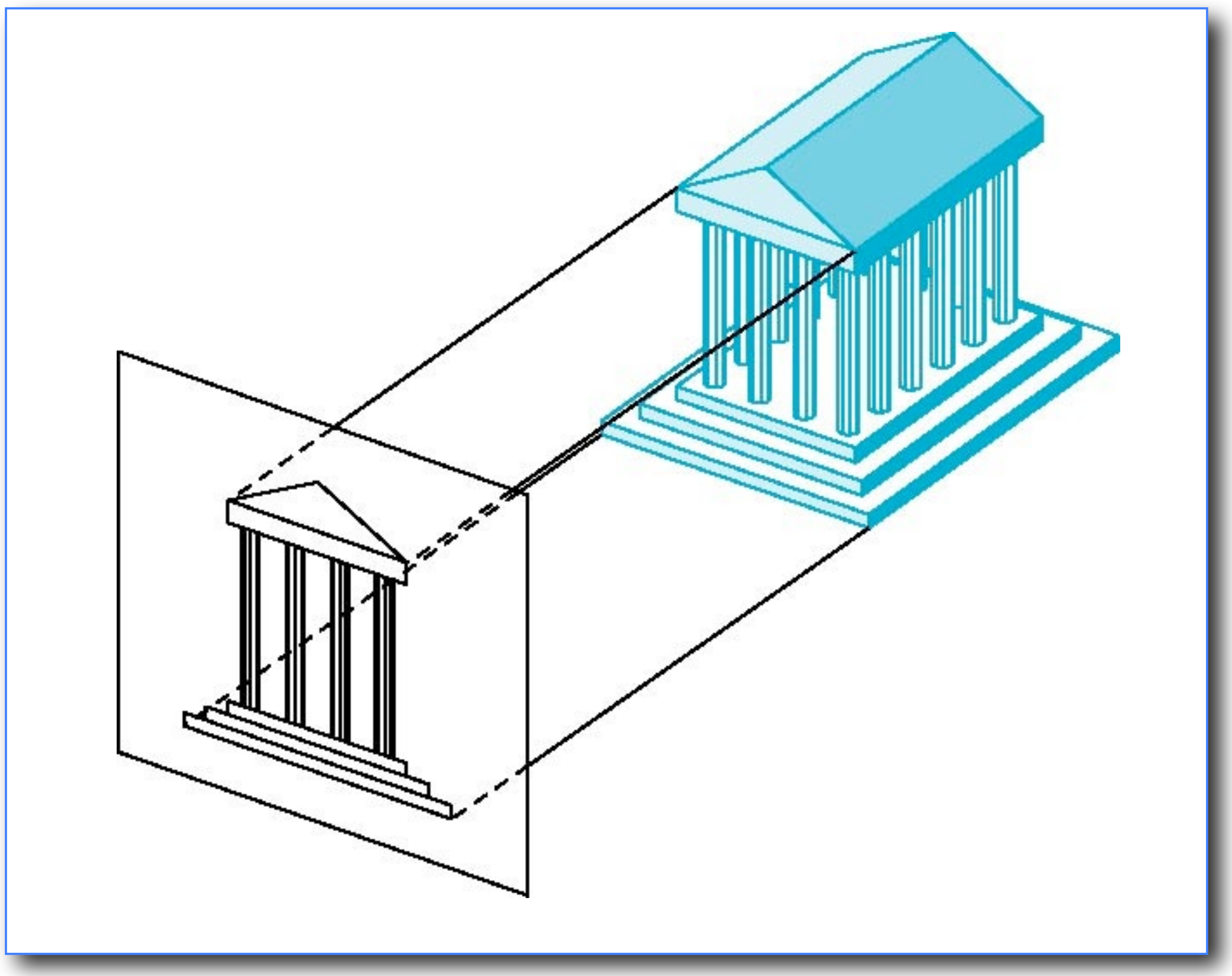


Parallel Projection



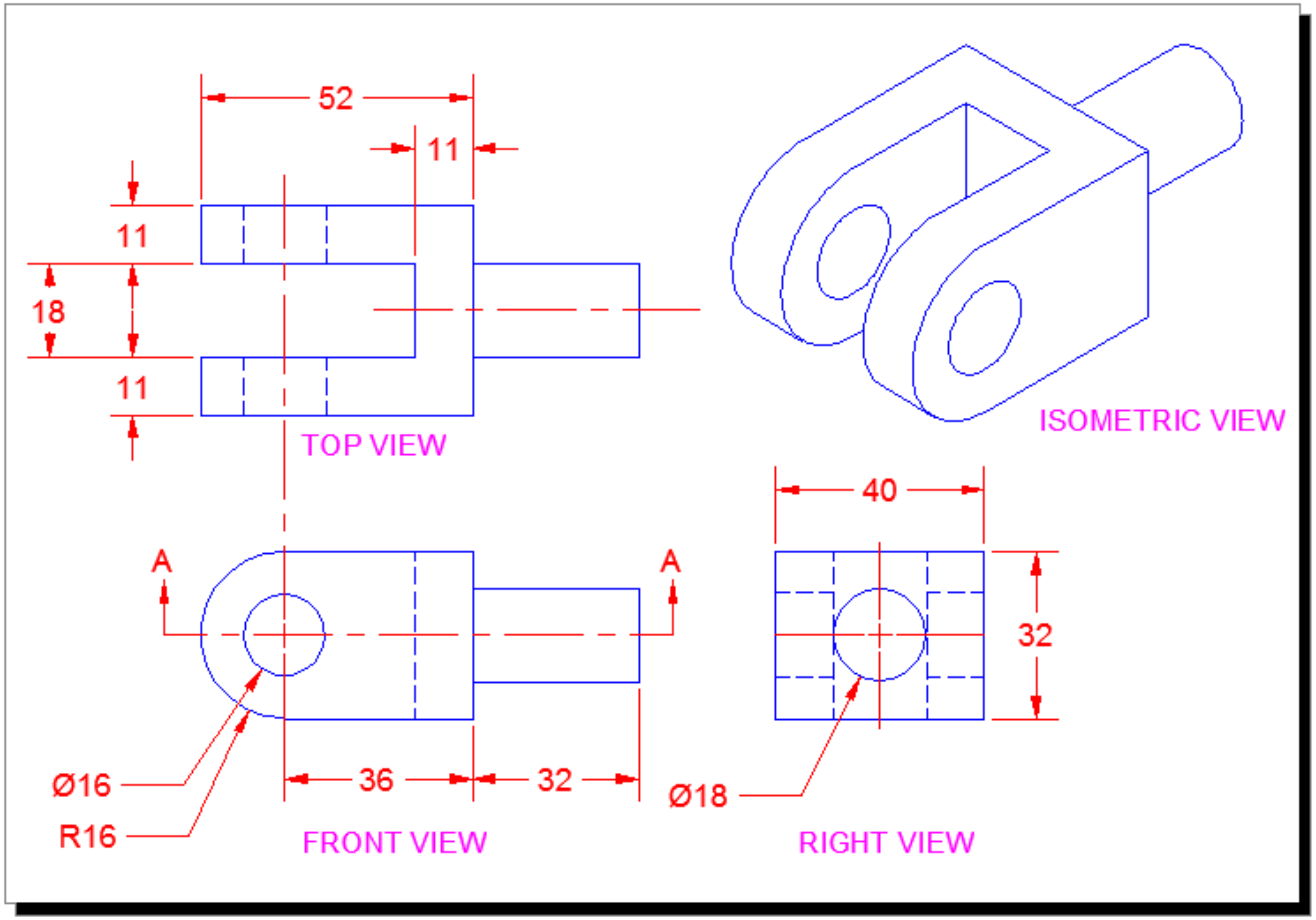
Orthographic Projection

- Projectors are orthogonal to projection surface, which is typically parallel to one of the coordinate planes:

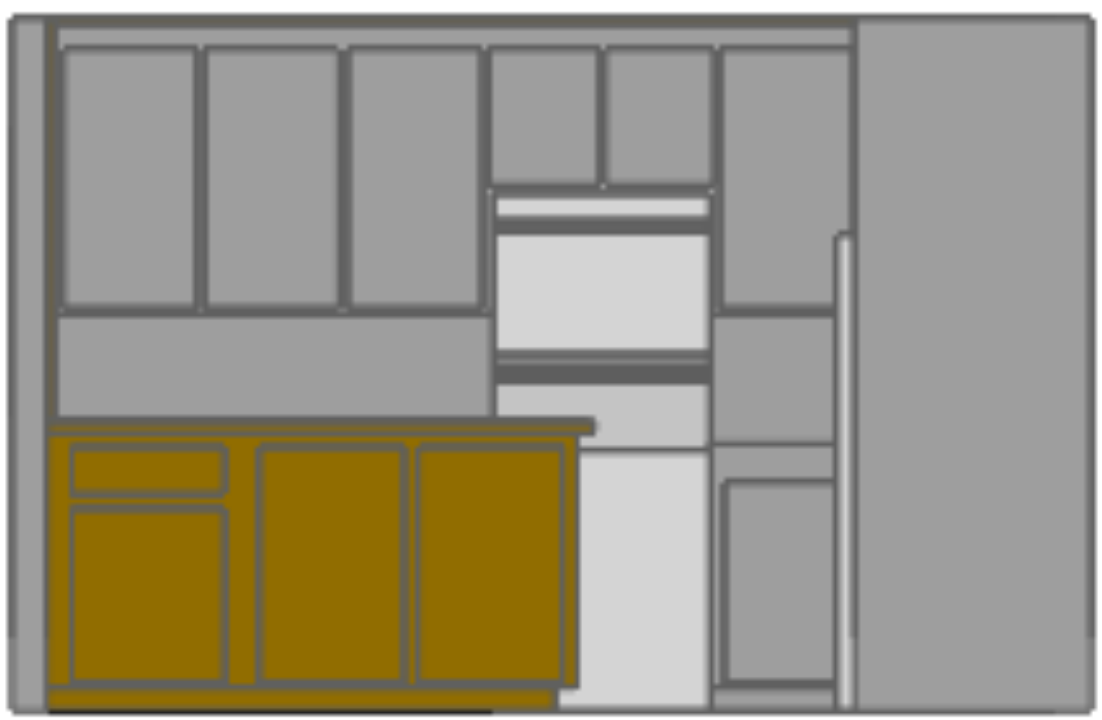


Orthographic Projection

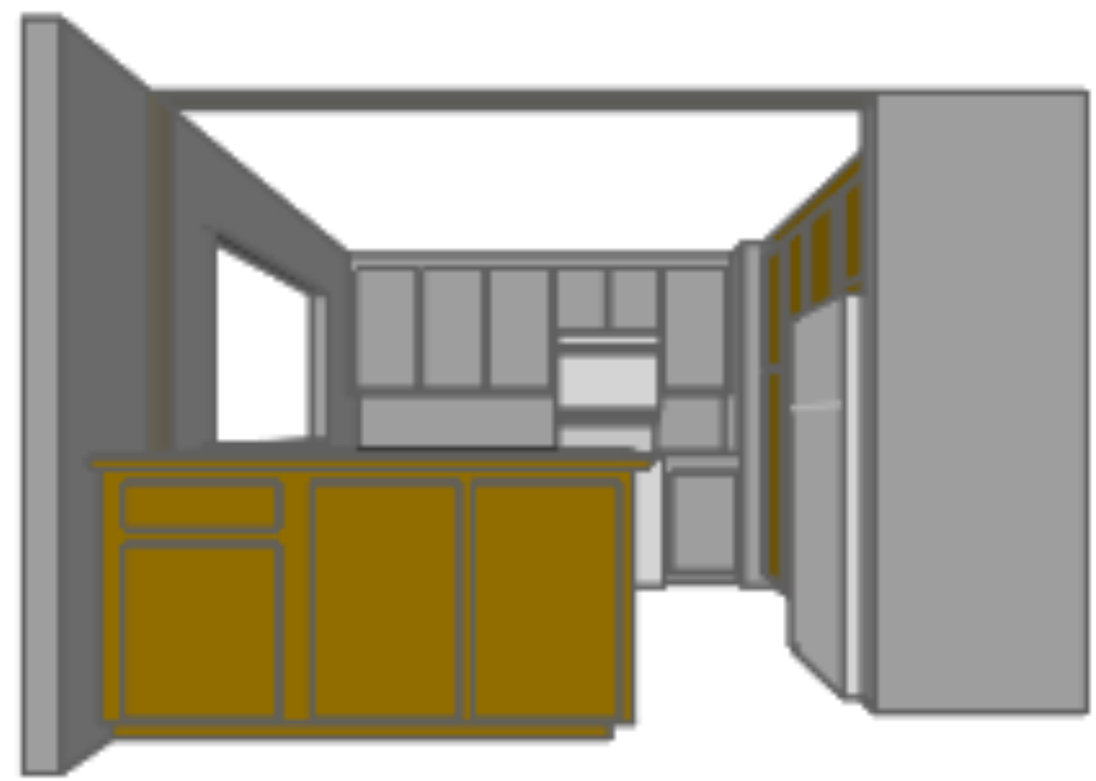
- Convenient for measuring distances and angles.
- Typically several simultaneous projections are shown.



Parallel vs. Perspective Projection

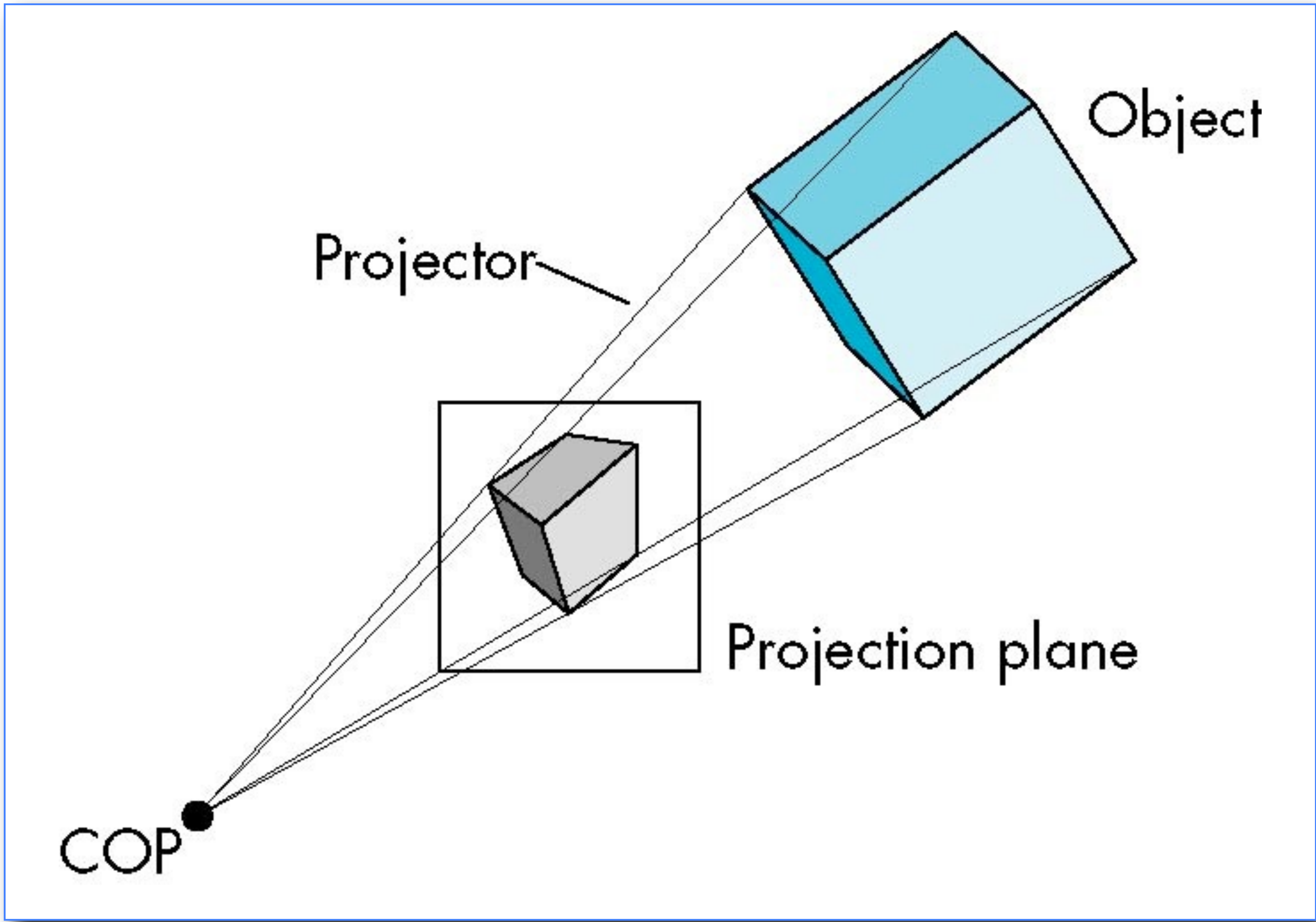


Parallel projection



Perspective projection

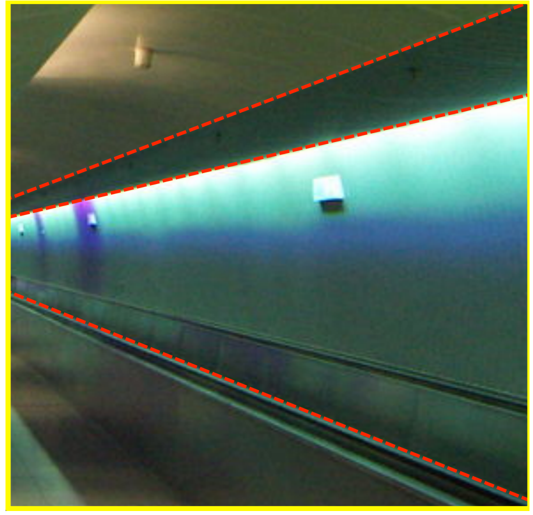
Perspective Projection



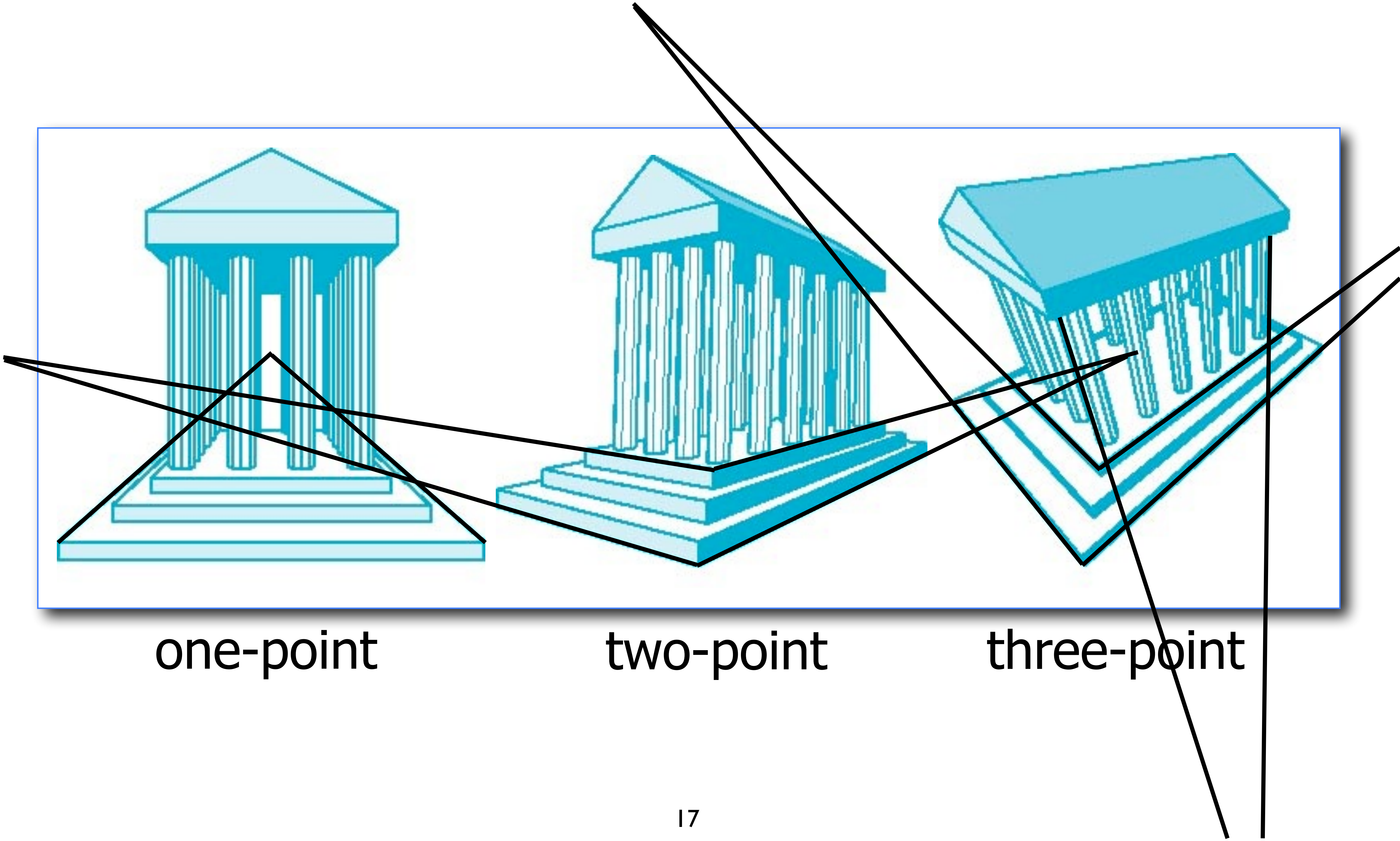
Vanishing Points

- Parallel lines (not parallel to the projection plane) in the scene converge at a single point on the projection plane (the vanishing point):

vanishing points may lie outside the view

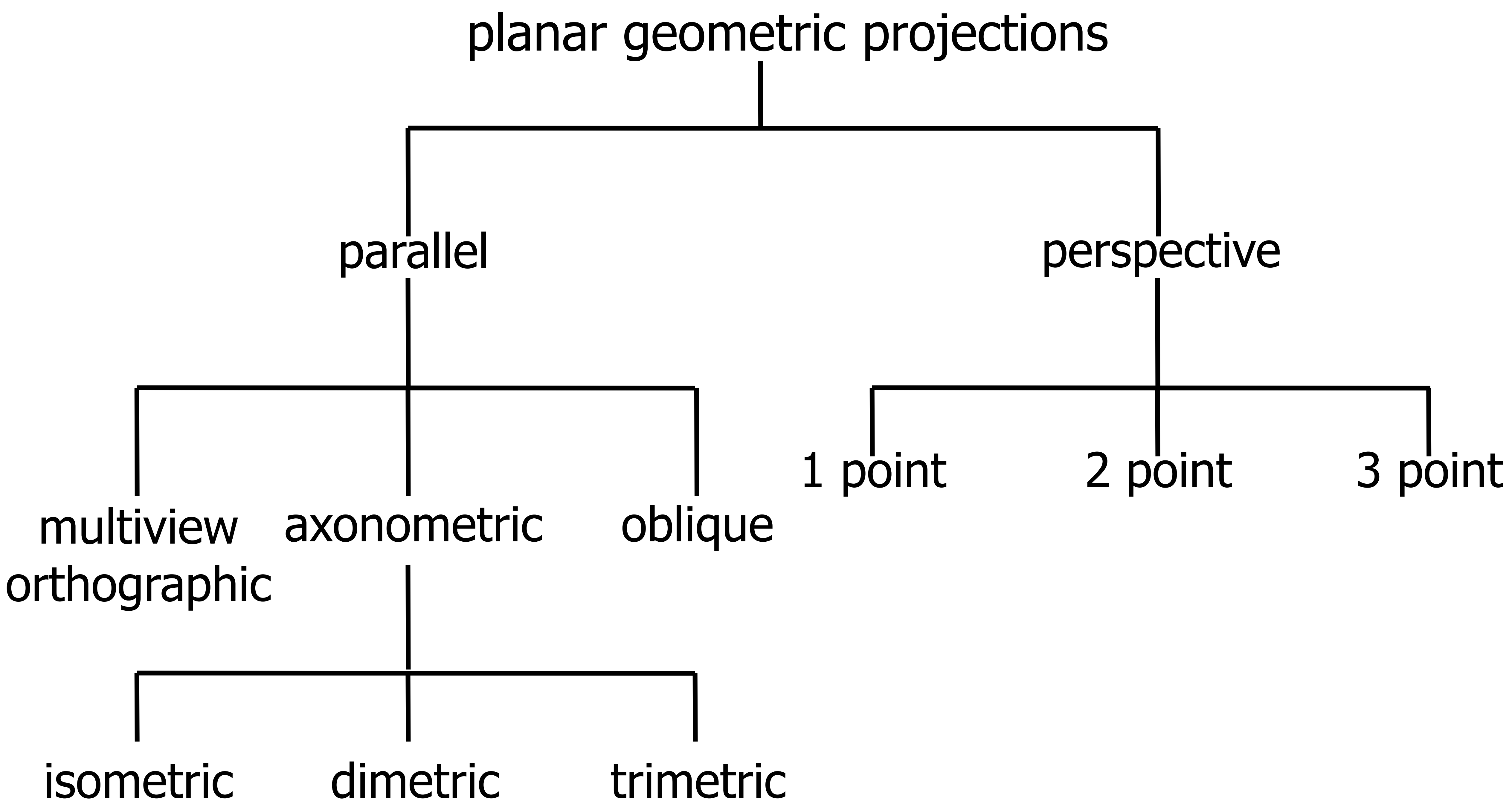


N-point Perspective



one-point two-point three-point

Taxonomy of Projections



Orthographic Projection

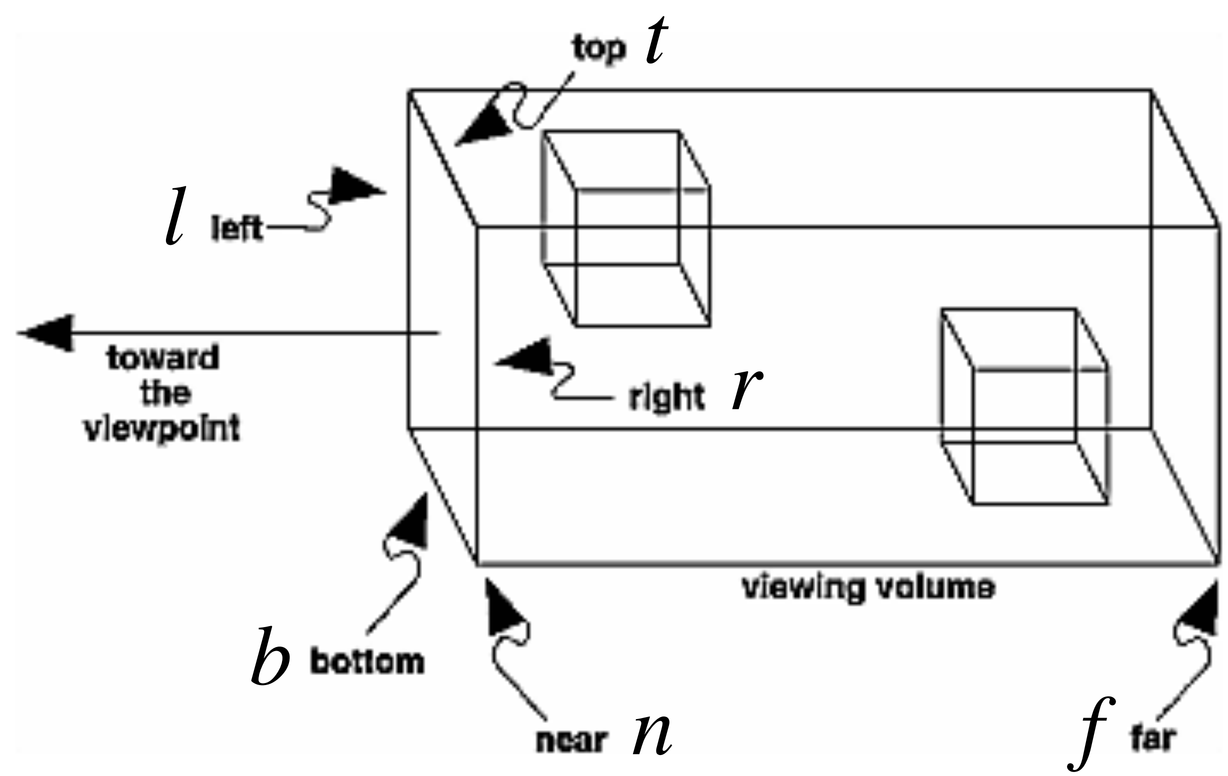
- Direction of projection is normal to the projection plane.
- Typically, project onto one of the coordinate planes. For example onto the z = 0 plane:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 0 \\ 1 \end{bmatrix}$$

- This matrix discards all depth information, so it cannot be used in the graphics pipeline.

Orthographic Projection (in OpenGL)

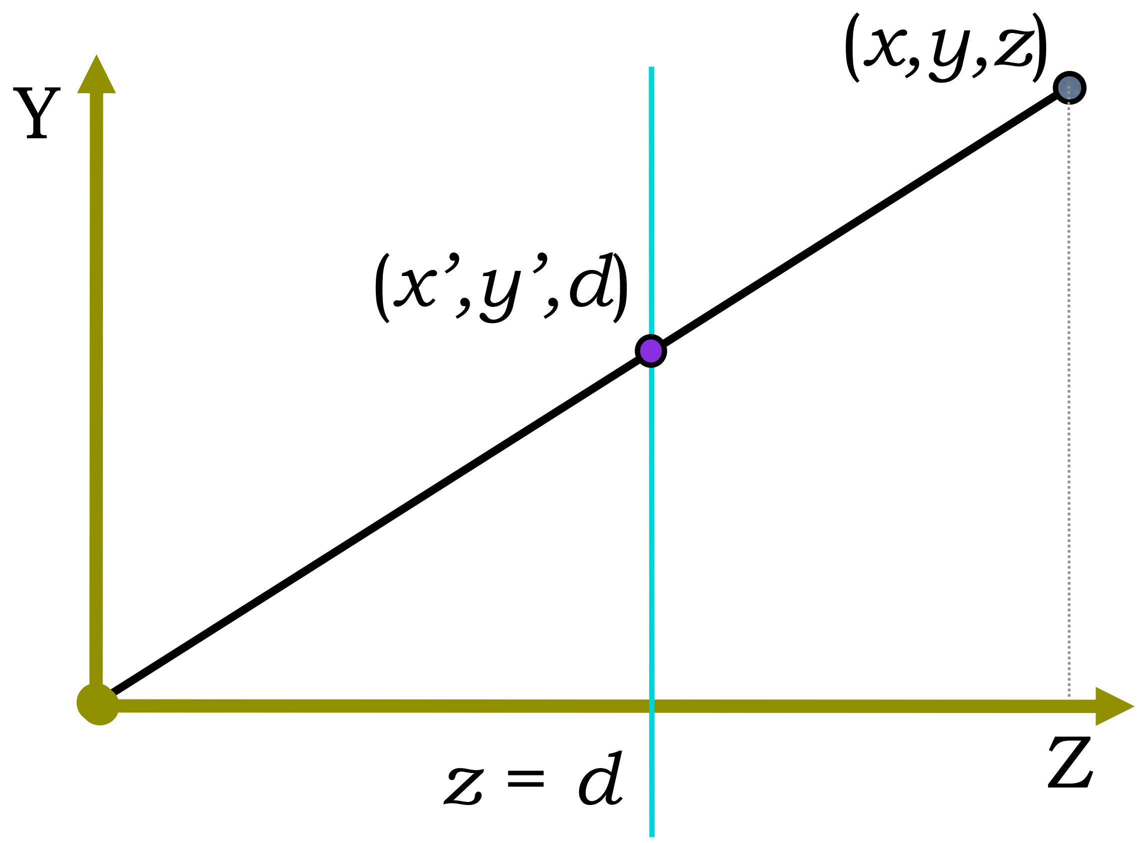
- Specify the boundaries of an axis aligned view volume in eye coordinates:
- (*l*eft, *r*ight, *b*ottom, *t*op, *n*ear, *f*ar)



- Map the above view volume to normalized device coordinates (everything in $[-1,1]^3$):
- All three dimensions are preserved!

$$\begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{-2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Perspective Projection



$$\frac{x'}{d} = \frac{x}{z} \quad \Rightarrow \quad x' = \frac{xd}{z}$$

$$\frac{y'}{d} = \frac{y}{z} \quad \Rightarrow \quad y' = \frac{yd}{z}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix} \quad \Rightarrow \quad \begin{bmatrix} xd/z = x' \\ yd/z = y' \\ zd/z = d \\ 1 \end{bmatrix}$$

Observations

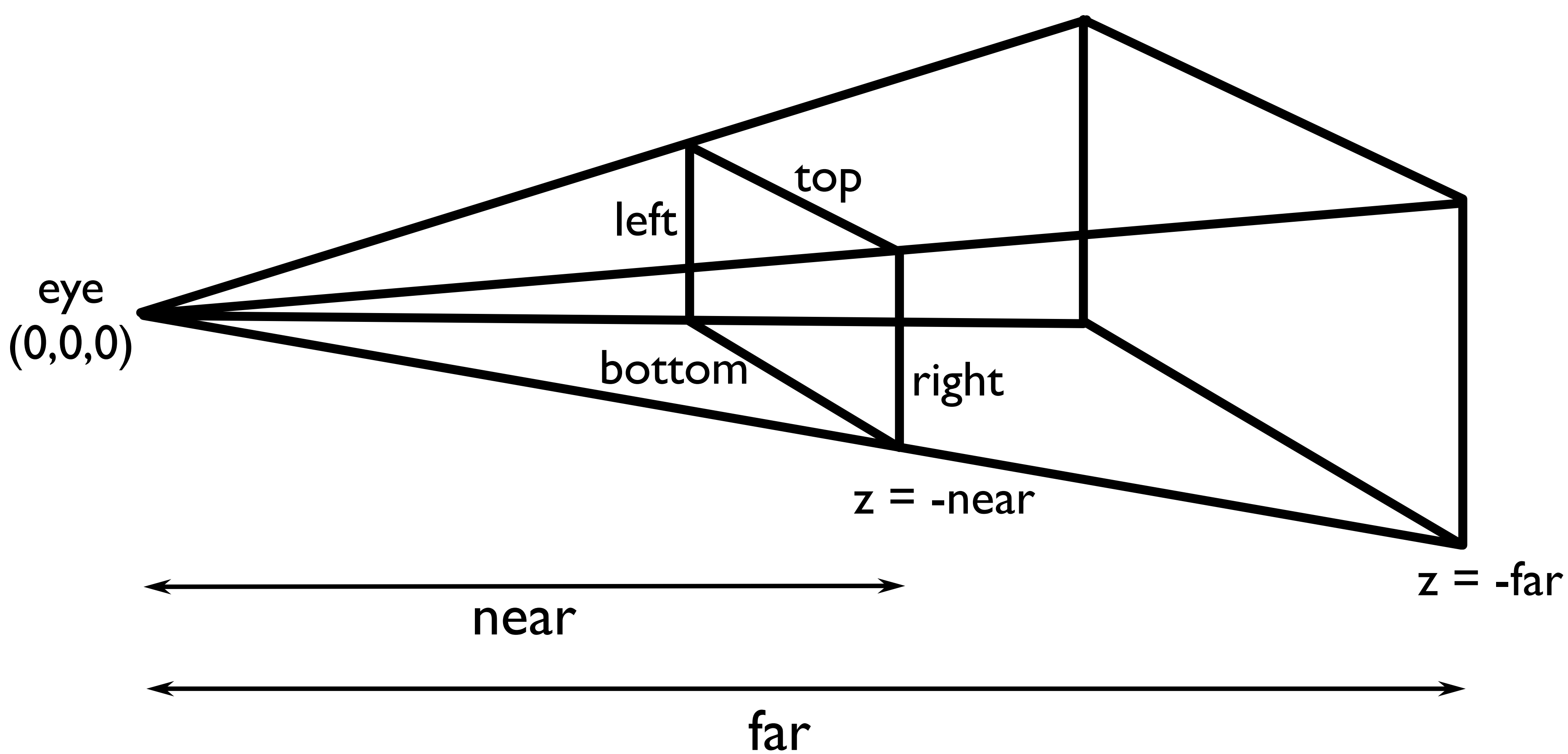
- The rank of the matrix is 3 (= projection)
- Points on the projection plane are not changed by the perspective projection
- Let's see what happens to a point at infinity along the Z axis:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1/d \end{bmatrix} \quad \Rightarrow \quad \begin{bmatrix} 0 \\ 0 \\ d \\ 1 \end{bmatrix}$$

- This is a vanishing point!

Perspective Projection (in OpenGL)

- Specify a pyramidal view volume (view frustum) in eye coordinates:



Perspective Projection Matrix

- Maps the view volume to normalized device coordinates (everything in $[-1,1]^3$):

$$\begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

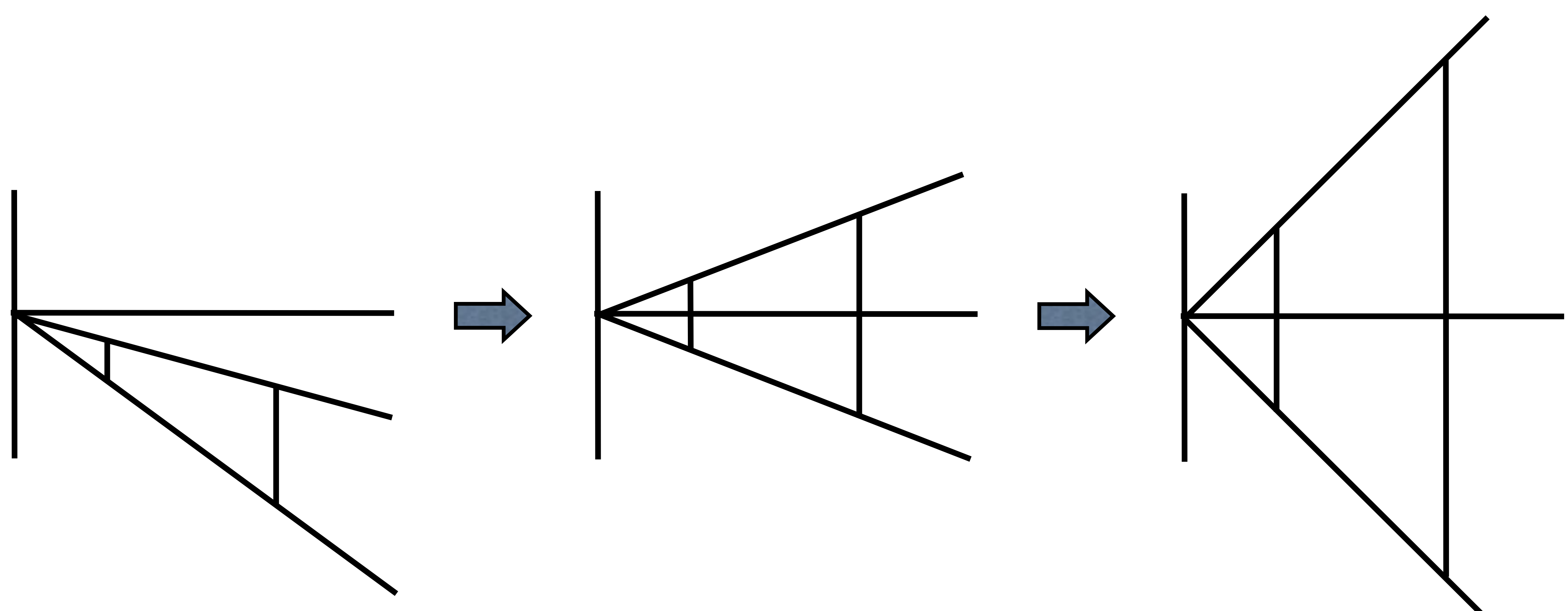
Perspective Projection Matrix

- Example:

$$\begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} l \\ b \\ -n \\ 1 \end{bmatrix} = \begin{bmatrix} -n \\ -n \\ -n \\ n \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \\ -1 \\ 1 \end{bmatrix}$$

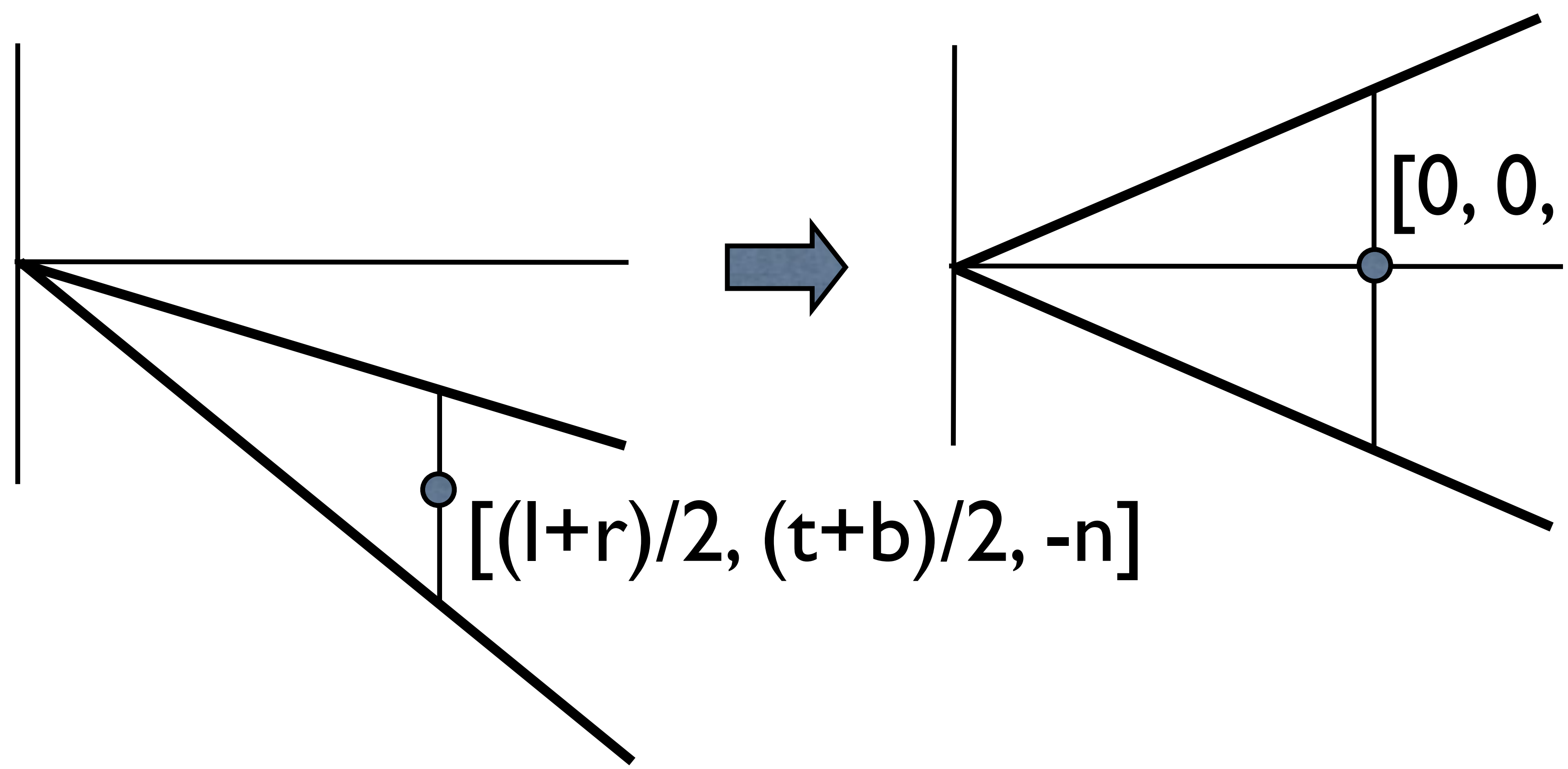
Derivation, part I (affine)

- The view volume is defined as a general (possibly skewed) viewing pyramid. We first make this pyramid into a canonical one:
- We first shear the skewed pyramid, then scale.

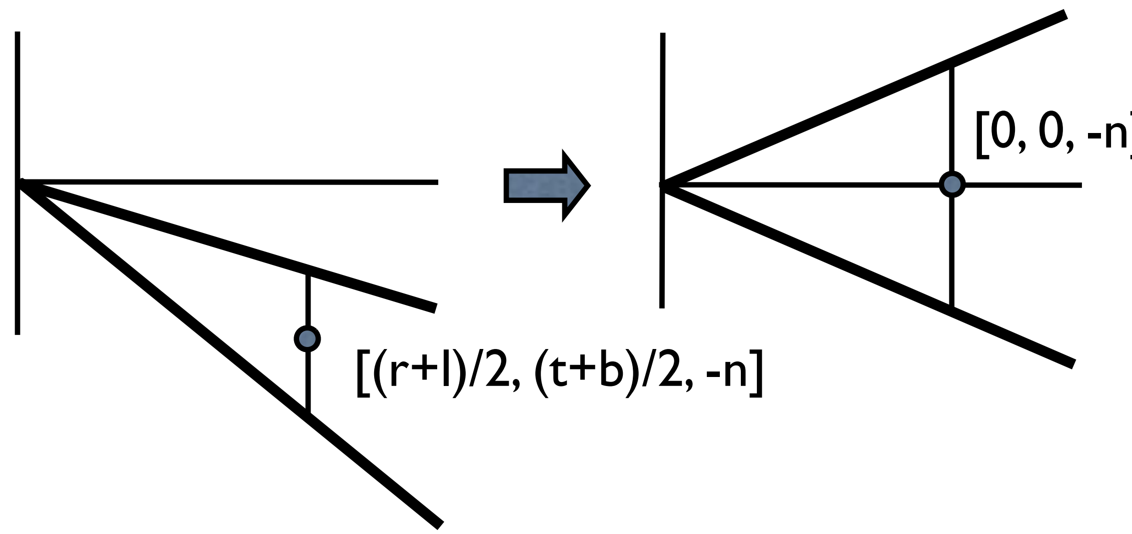


Shearing Matrix

- Transforms the center of the viewing window on the near plane [(r+l)/2, (t+b)/2, -n] to [0, 0, -n], making the view pyramid symmetric about the Z-axis:



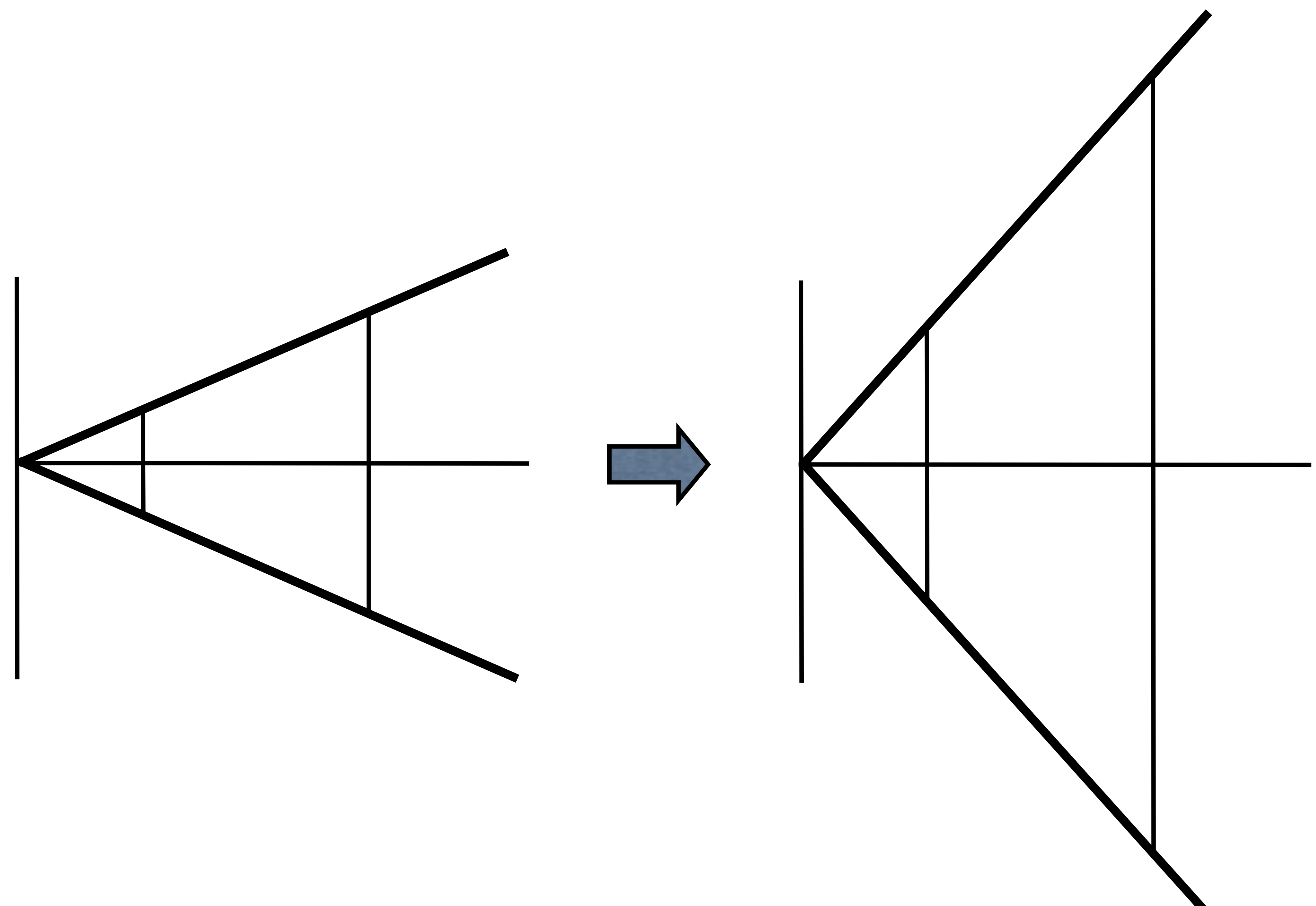
Shearing Matrix



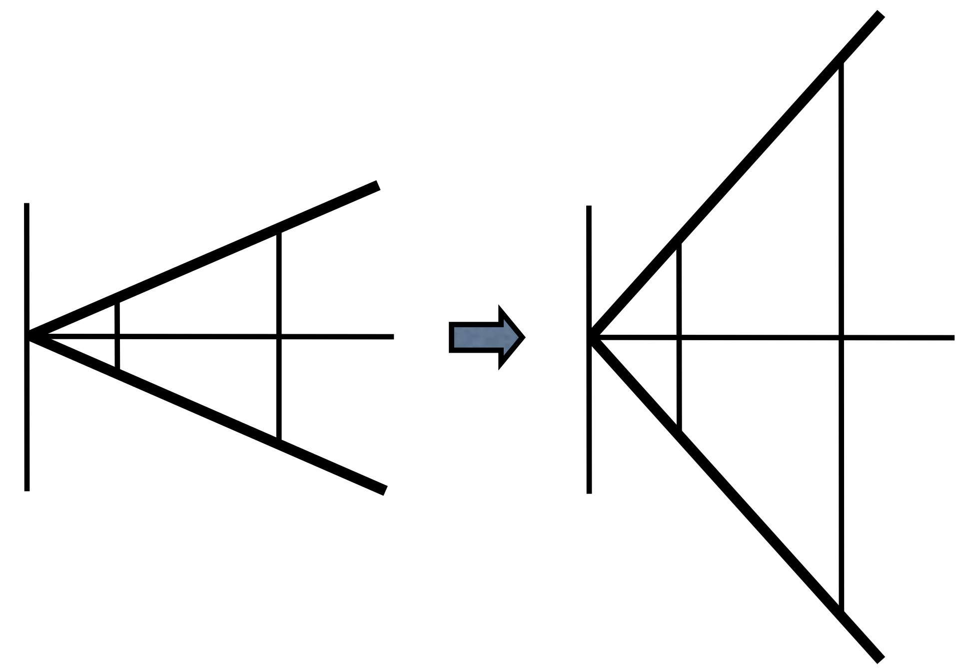
$$\begin{bmatrix} 1 & 0 & \frac{r+l}{2n} & 0 \\ 0 & 1 & \frac{t+b}{2n} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{r+l}{2} \\ \frac{t+b}{2} \\ -n \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -n \\ 1 \end{bmatrix}$$

Scaling Matrix

- Scale the symmetric pyramid to create a 45 degree angle between each plane and the Z-axis:



Scaling Matrix



$$\begin{bmatrix} \frac{2n}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2n}{t-b} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{r-l}{2} \\ \frac{t-b}{2} \\ -n \\ 1 \end{bmatrix} = \begin{bmatrix} n \\ n \\ -n \\ 1 \end{bmatrix}$$

Derivation, part II (perspective)

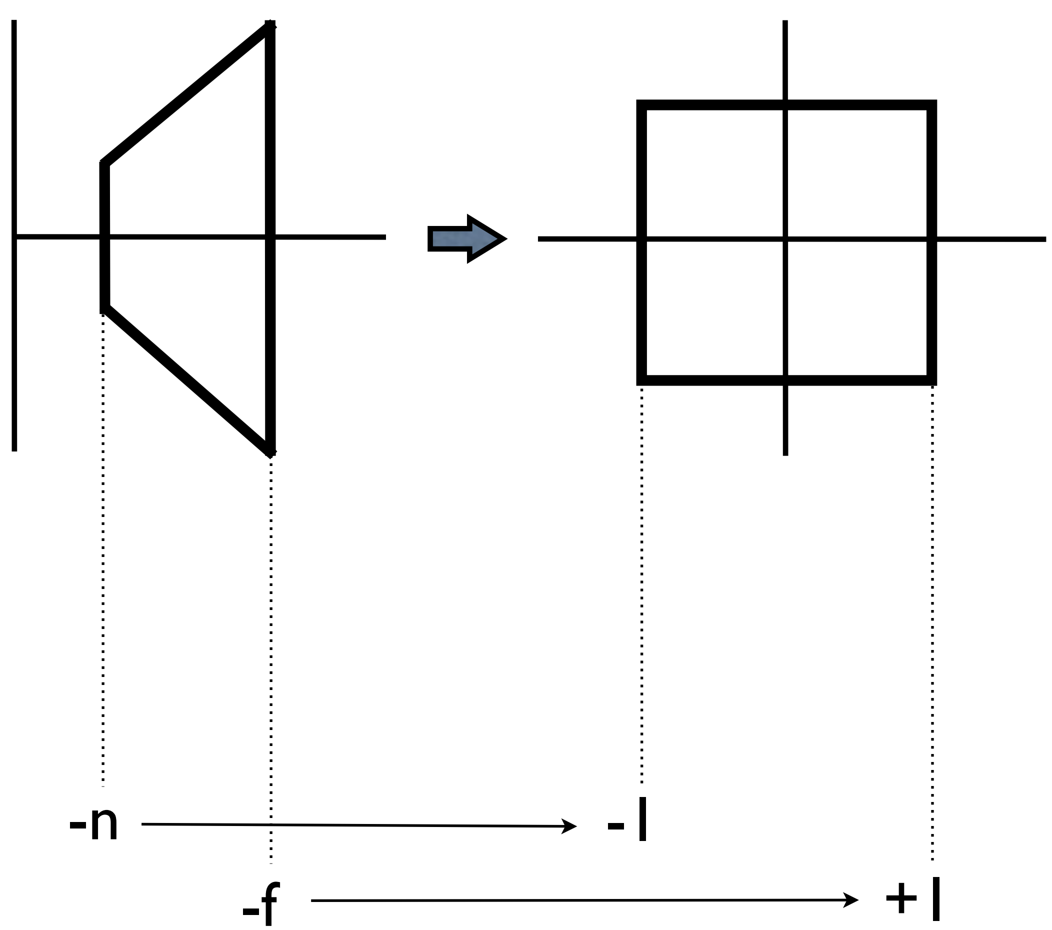
- Assuming the center of projection at (0,0,0) and the projection plane is at z = -1, we have (from similar triangles):

$$\begin{aligned} x_{out} &= x_{eye} / -z_{eye} \\ y_{out} &= y_{eye} / -z_{eye} \end{aligned}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & A & B \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ Az+B \\ -z \end{bmatrix}$$

Derivation, part II (perspective)

- The canonical pyramid is then transformed into a cube, using a perspective transformation:



$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & A & B \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

$$A = -\frac{f+n}{f-n}$$

$$B = \frac{-2fn}{f-n}$$

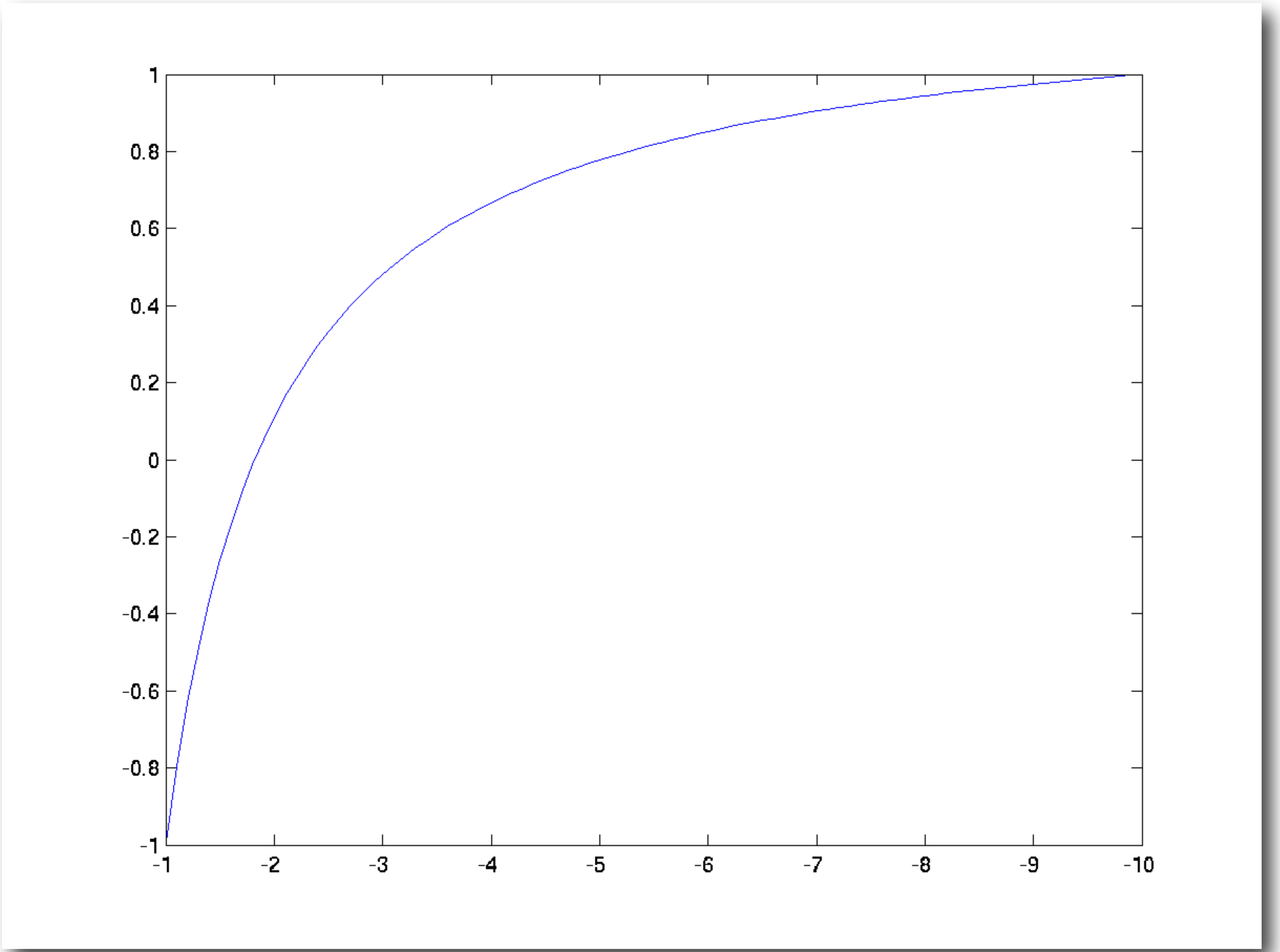
Finally...

- Multiplying these three transformations gives us the desired matrix:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} \frac{2n}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2n}{t-b} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & \frac{r+l}{2n} & 0 \\ 0 & 1 & \frac{t+b}{2n} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} =$$

$$= \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Non-linear (but monotonic) mapping of Z values:



View Frustum Clipping

- After applying the projection and normalizing, all of the coordinates lie in the canonical cube $[-1,1]\times[-1,1]\times[-1,1]$
- View frustum clipping may be done by checking each coordinate against the interval $[-1,1]$
- **Problem**: points *behind* the camera can also be mapped to the canonical view cube.
- **Solution**: perform clipping *before* the perspective division.

View Frustum Clipping

- In homogeneous coordinates all points inside the view frustum satisfy all of the following inequalities:

$$w > 0 \quad \text{and} \quad \left\{ \begin{matrix} x < w & x > -w \\ y < w & y > -w \\ z < w & z > -w \end{matrix} \right\}$$

- Lines must be clipped against the planes:

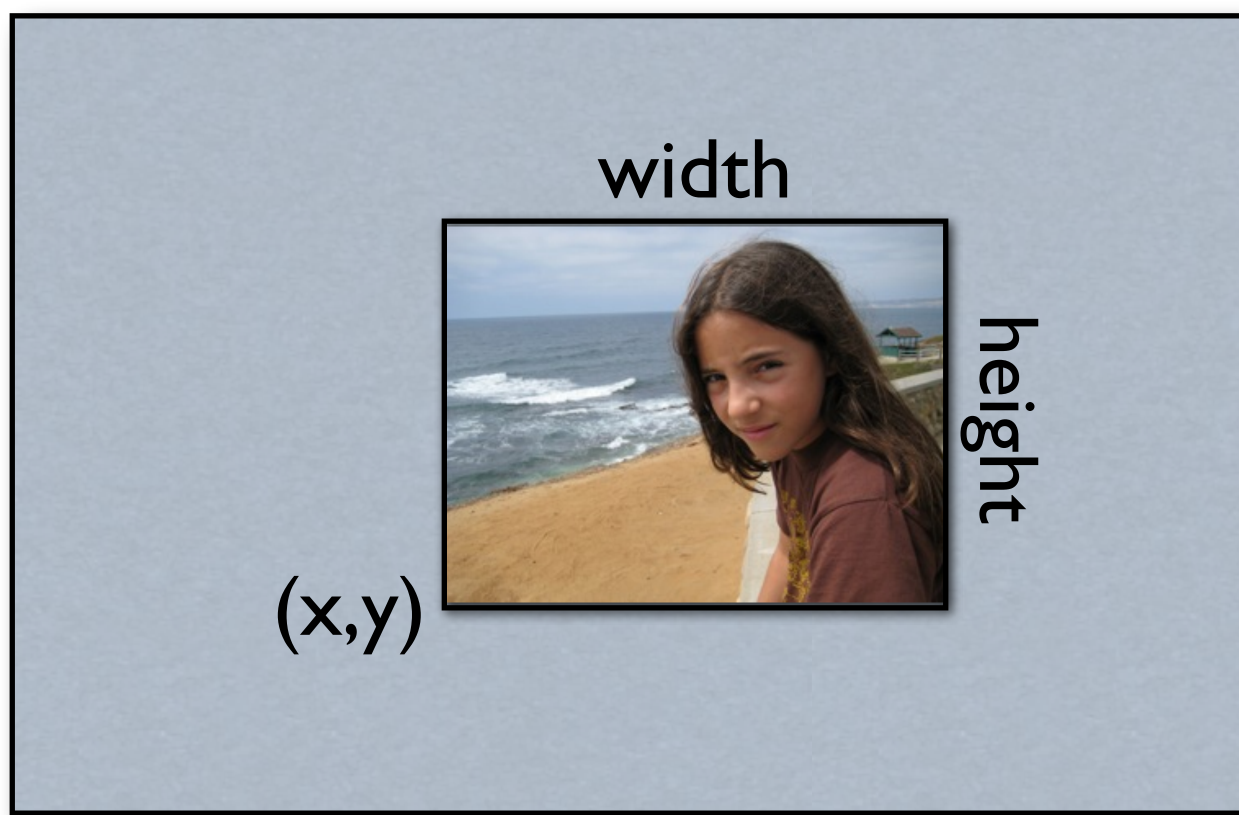
$$\left\{ \begin{matrix} x = w & x = -w \\ y = w & y = -w \\ z = w & z = -w \end{matrix} \right\}$$

Viewport Transformation

- Defines a pixel rectangle in the window into which the final image is mapped:

```
glViewport(x, y, width, height)
```

- (x, y) specify the lower left corner of the viewport:



Viewport Transformation

- Transforms normalized device (nd) coordinates to window (w) coordinates.
- nd coordinates range in [-1,1]
- window coordinates range in [x, x+width], [y,y+height]
- The resulting transformation is:

$$x_w = (x_{nd} + 1)\left(\frac{width}{2}\right) + x$$

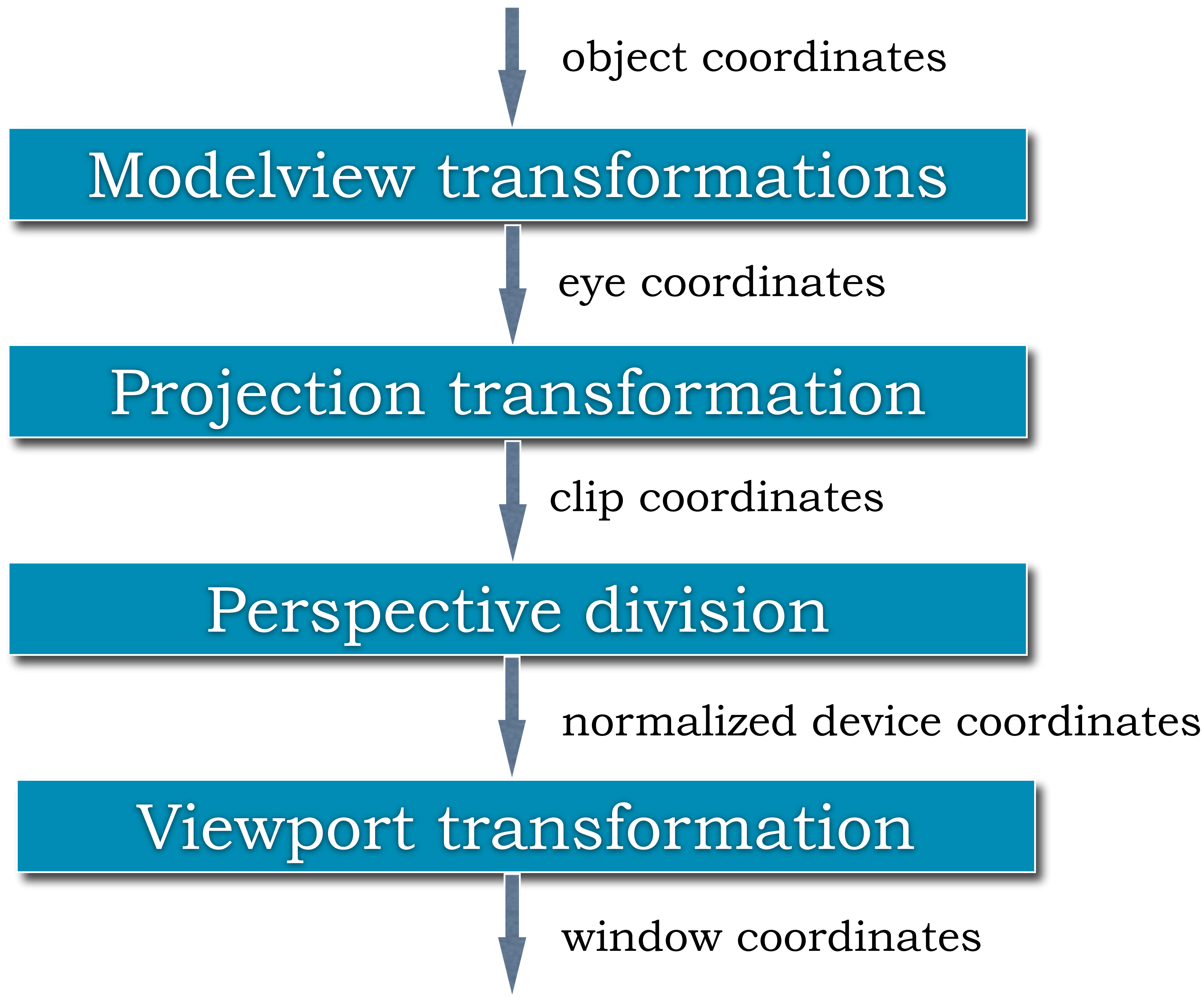
$$y_w = (y_{nd} + 1)\left(\frac{height}{2}\right) + y$$

- Done by OpenGL (and not in your program)

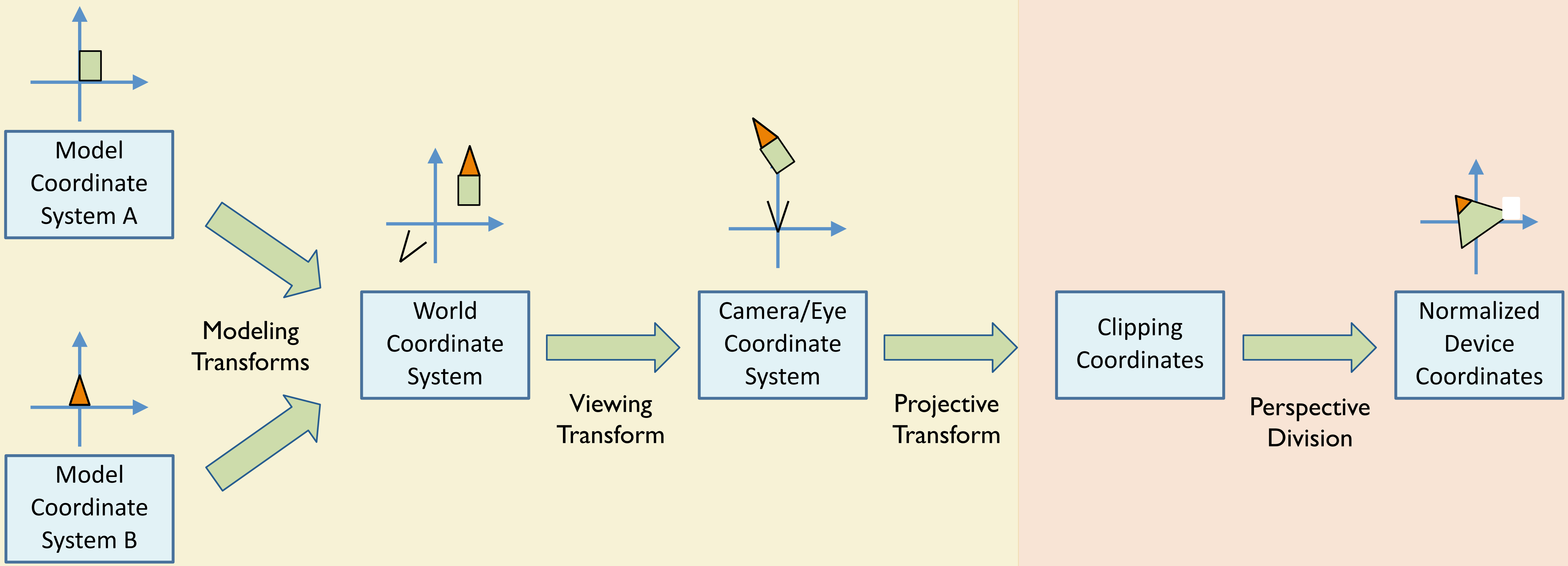
Viewport Transformation

- Transforms normalized device (nd) **depth** coordinates from [-1,1] into [0,1].
- Done by OpenGL (and not in your program)

Summary: Coordinate Systems



3D Viewing Pipeline



Object Units	World Units	World Units	Clipping Units	[-1,1]x[-1,1]
Initial modeling coordinates used by the application	Final/common modeling coordinates used by the application	Lighting/Shading takes place in these coordinates	These coordinates are designed to make culling/clipping easy	These coordinates are the input needed by the rasterizer

All transformations and lighting are done by the application: transformations are typically done in the vertex shader.

Clipping, perspective division, and rasterization are done before the fragment shader is applied.

The Complete Rendering Pipeline