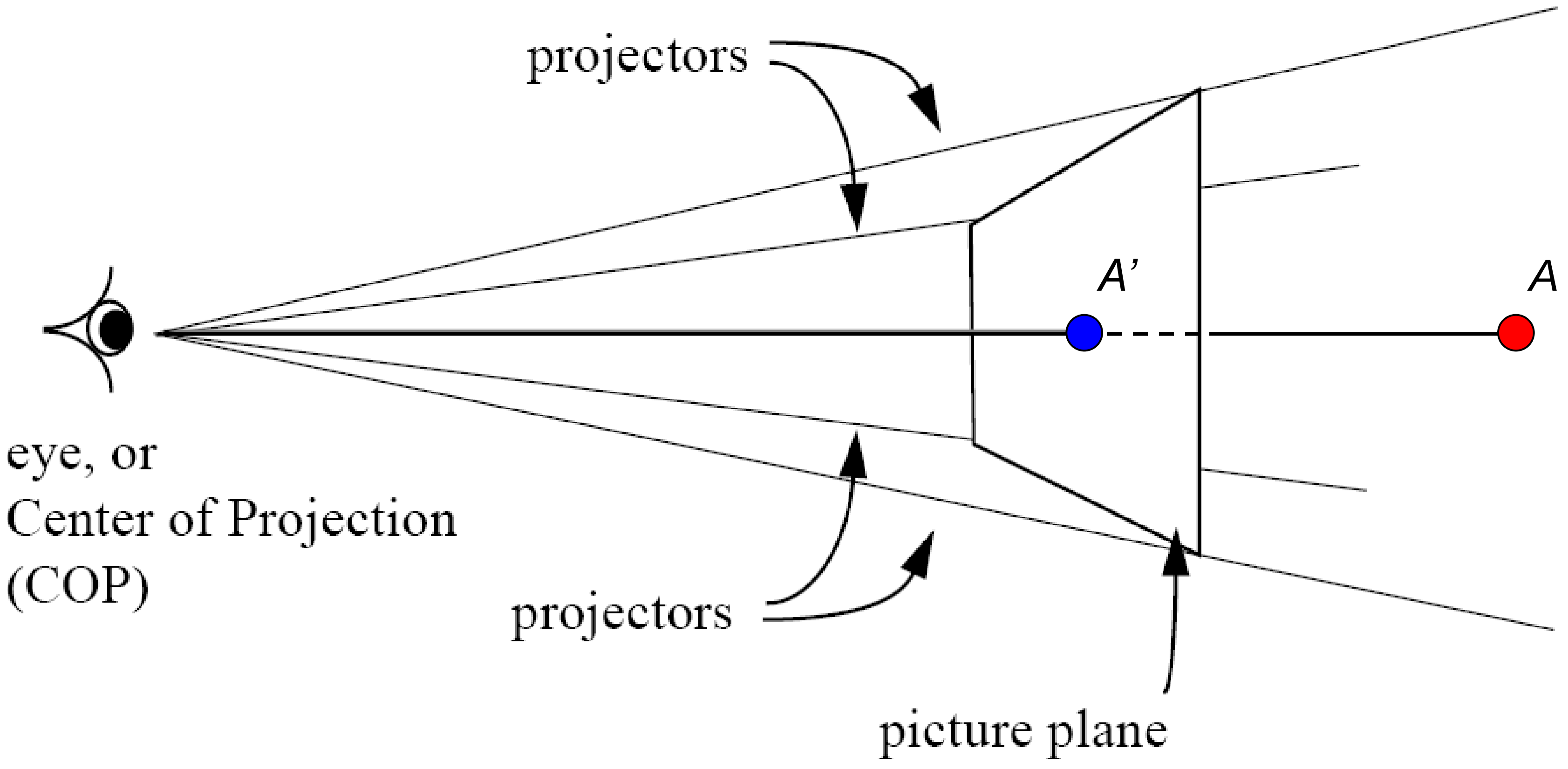
CSC 321 Computer Graphics

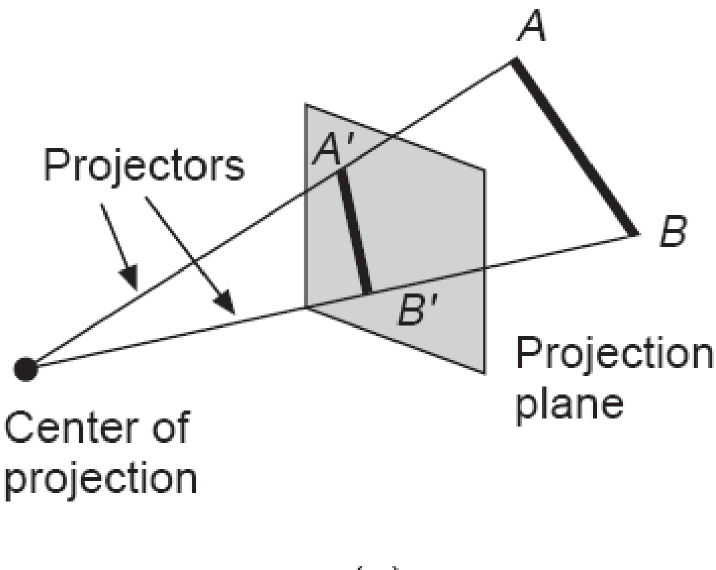
Projection

A painting based on a mythical tale as told by Pliny the Elder

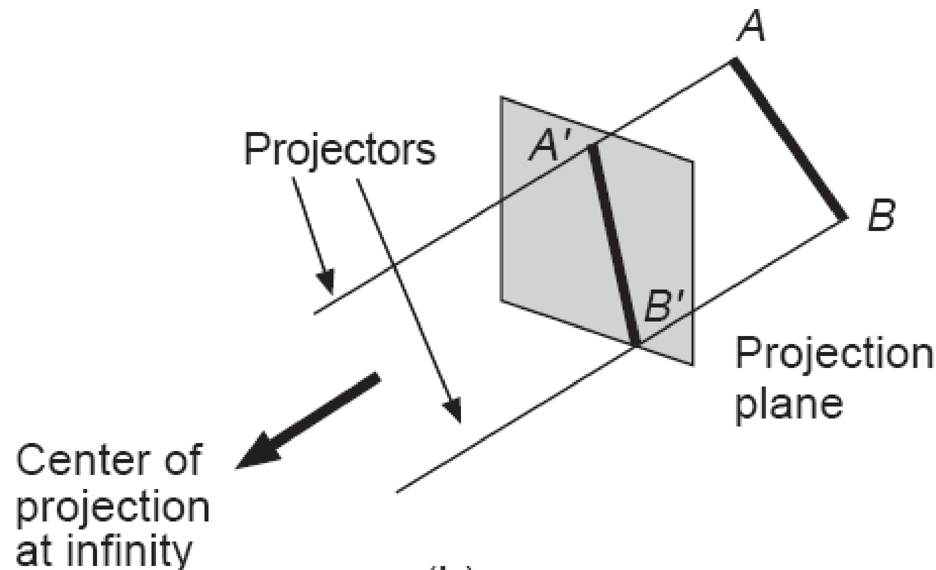
Planar Geometric Projection



Classification of Projections



(a)



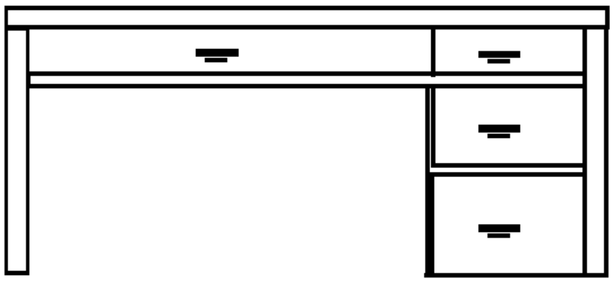
(b)

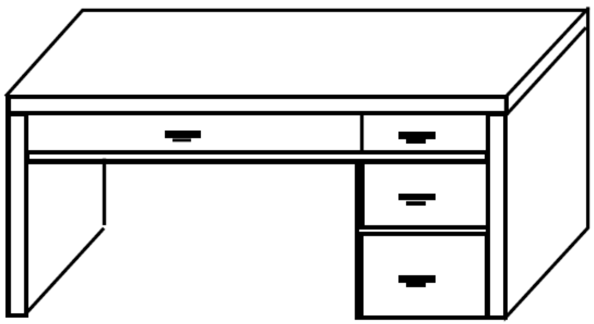
Perspective Projection

Parallel Projection

Parallel Projections

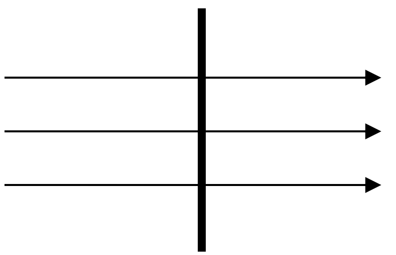
- Preserves object size
  - Edges parallel to projection plane maintain their lengths after projection
- Preserves parallelism
  - Lines that are parallel stay parallel after projection



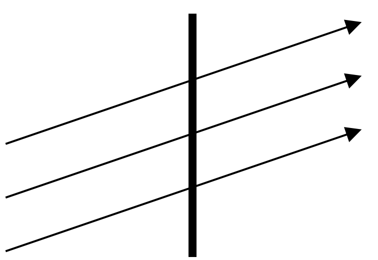


Types of Parallel Projection

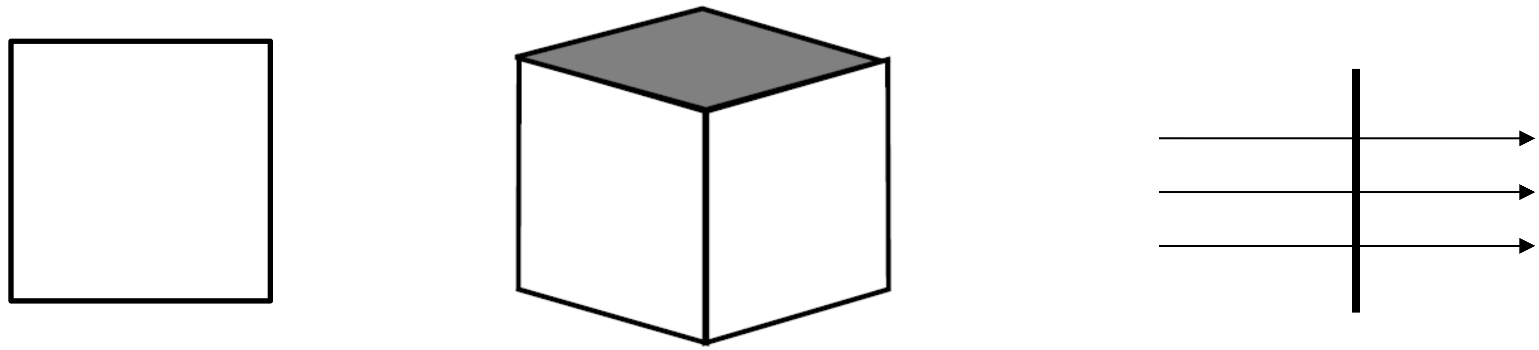
- Orthographic projection
  – Projectors orthogonal to the view plane



- Oblique projection
  – Projectors not orthogonal to the view plane



Types of Parallel Projection

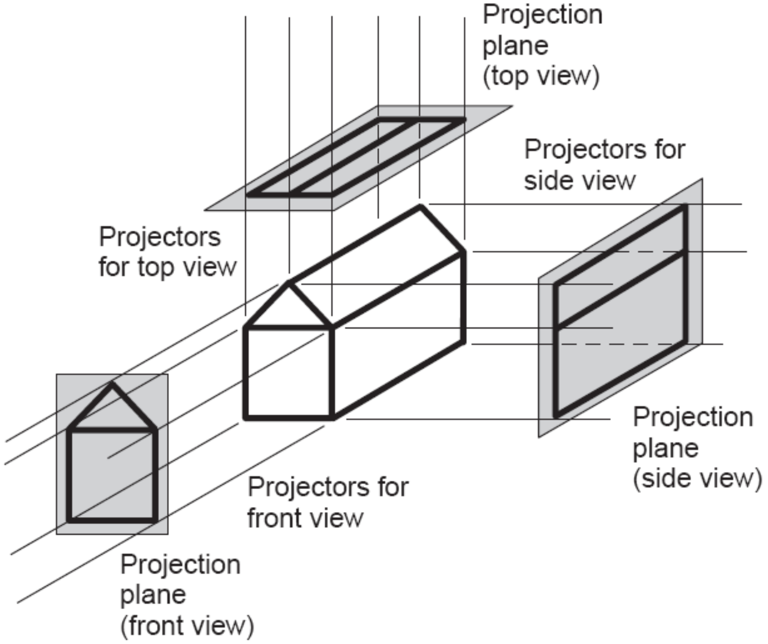


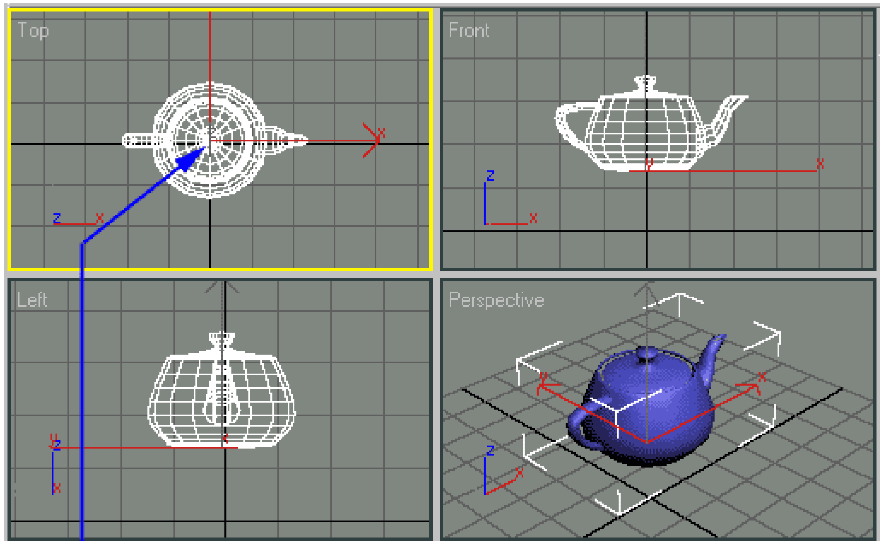
Orthographic

Oblique

Multi-view Orthographic Projection

- Projection plane is one of coordinate planes

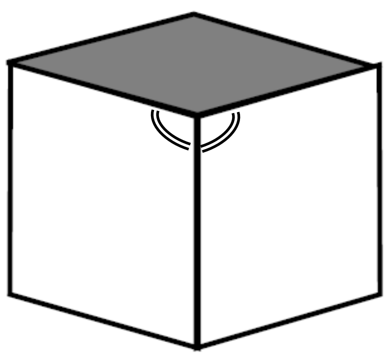




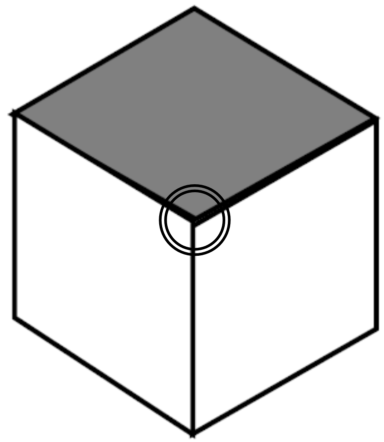
“3D Max” software interface

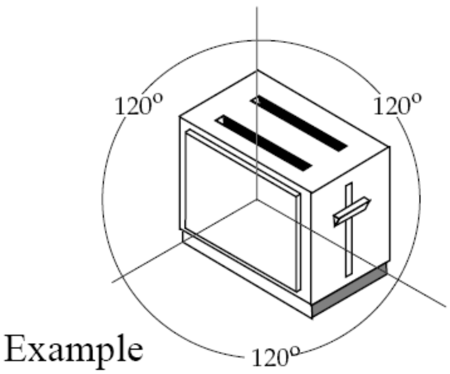
Axonometric Orthographic Projections

- Projection plane is **not** one of coordinate planes



Dimetric



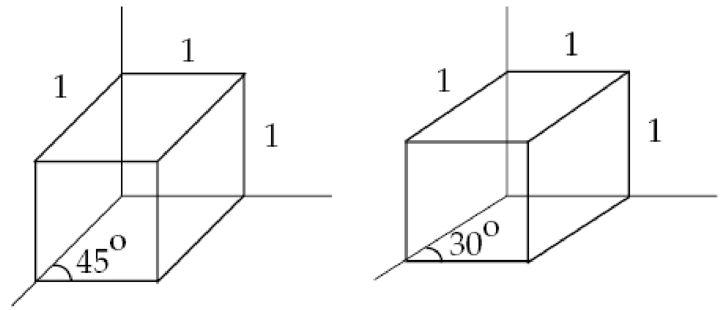


Isometric

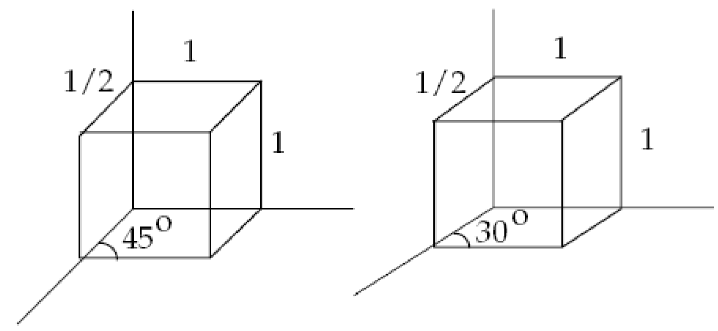
Oblique Projections

- Projectors **not** orthogonal to projection plane
  - The projection plane is typically parallel to a face of the object

- Classified by the angle between projector and plane
  - Pi/4: *Cavalier type*
    - Preserves the lengths of edges orthogonal to projection plane
  - ArcTan(2): *Cabinet type*
    - Halves the lengths of edges orthogonal to projection plane

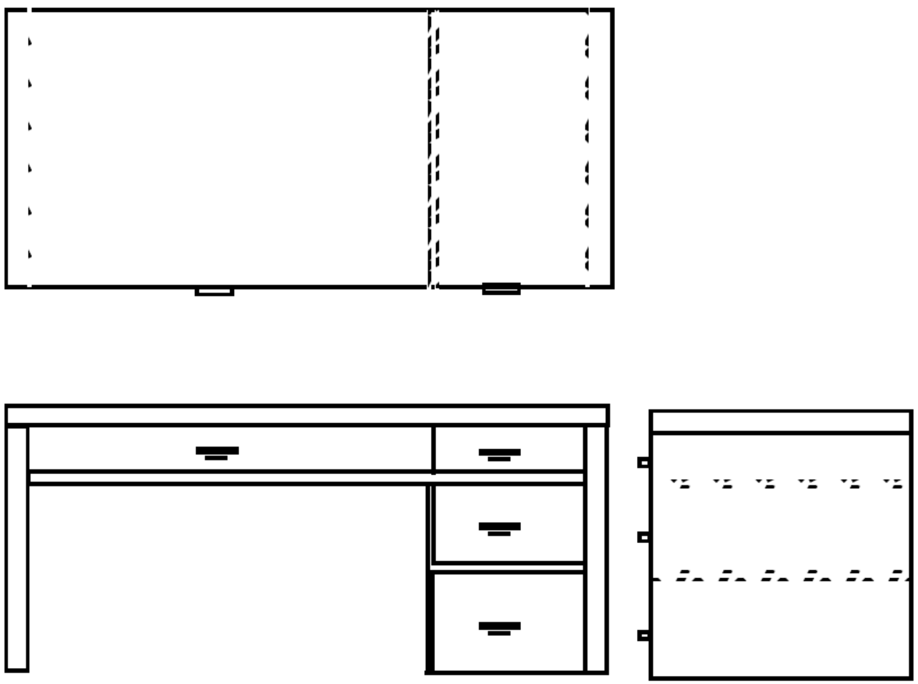


Cavalier: 45 degree

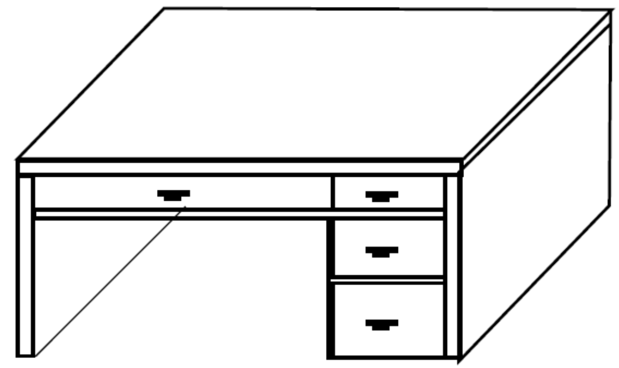


Cabinet: arctan(2) =63.4 degree

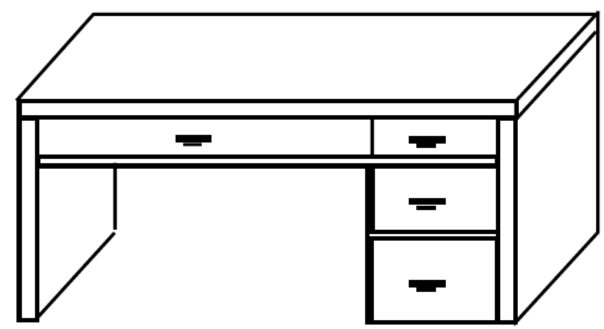
Examples of Parallel Projections



multiview orthographic

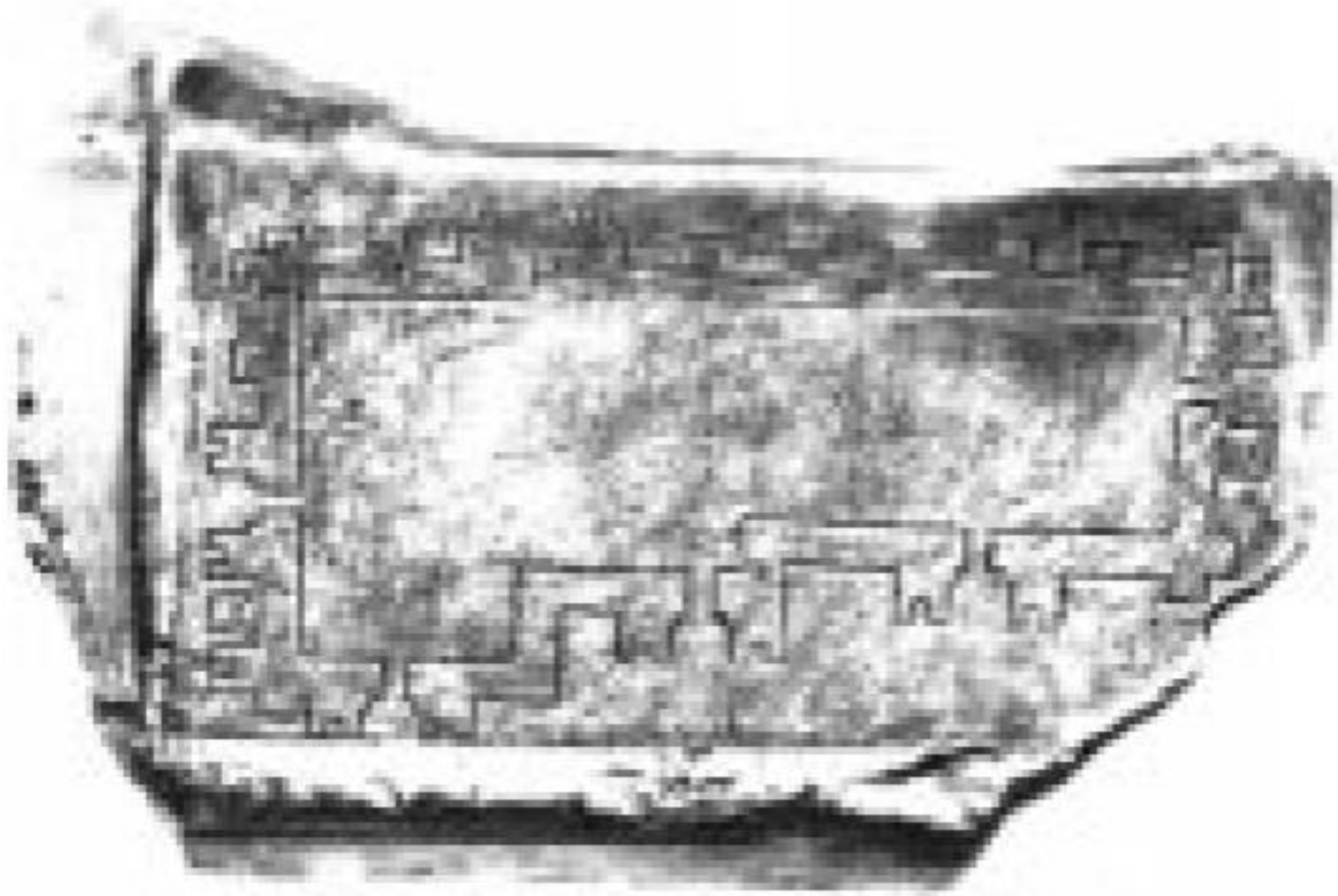


cavalier



cabinet

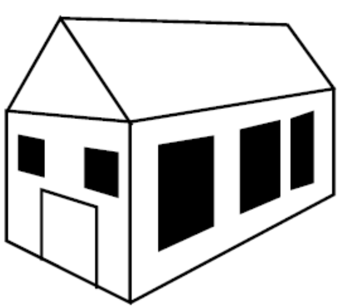
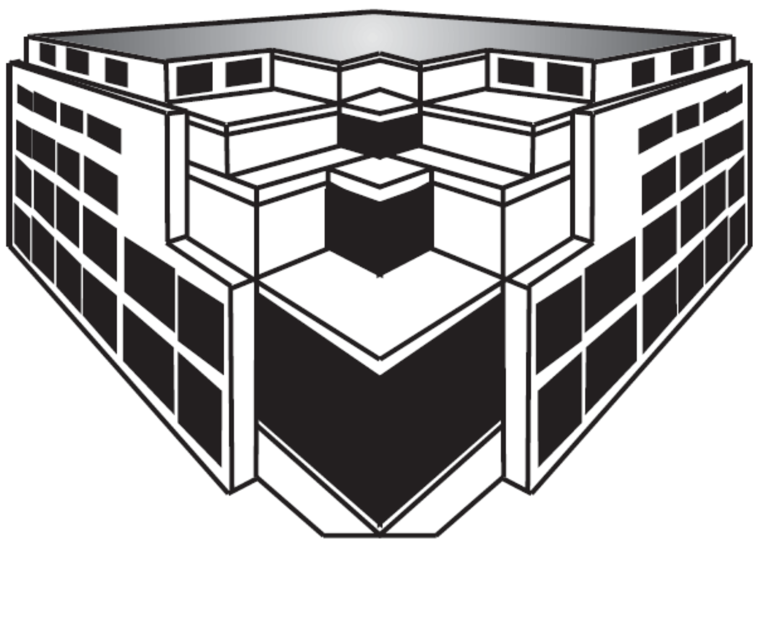
Parallel Projection in Drawing



Earliest known technical drawing: Plan view (orthographic projection) from Mesopotamia, 2150 BC

Perspective Projections

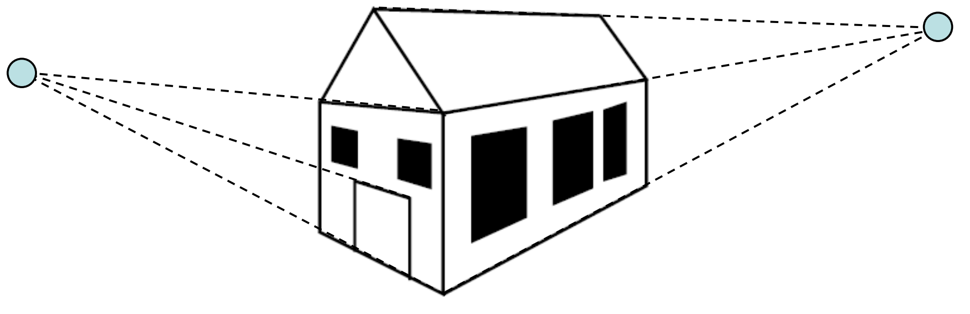
- How our eyes see the world
  - Objects further away look smaller (*foreshortening*)
  - Parallel lines may not remain parallel

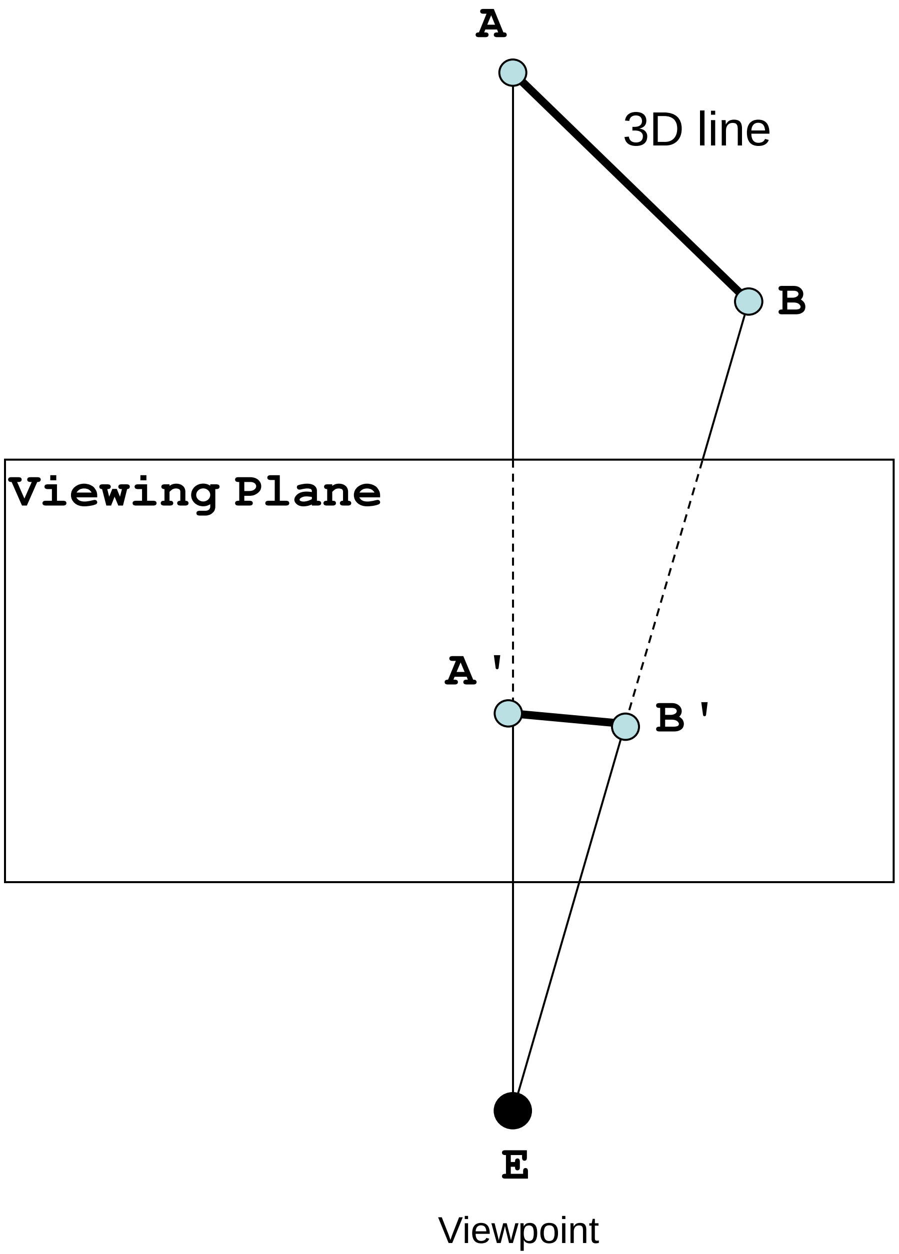



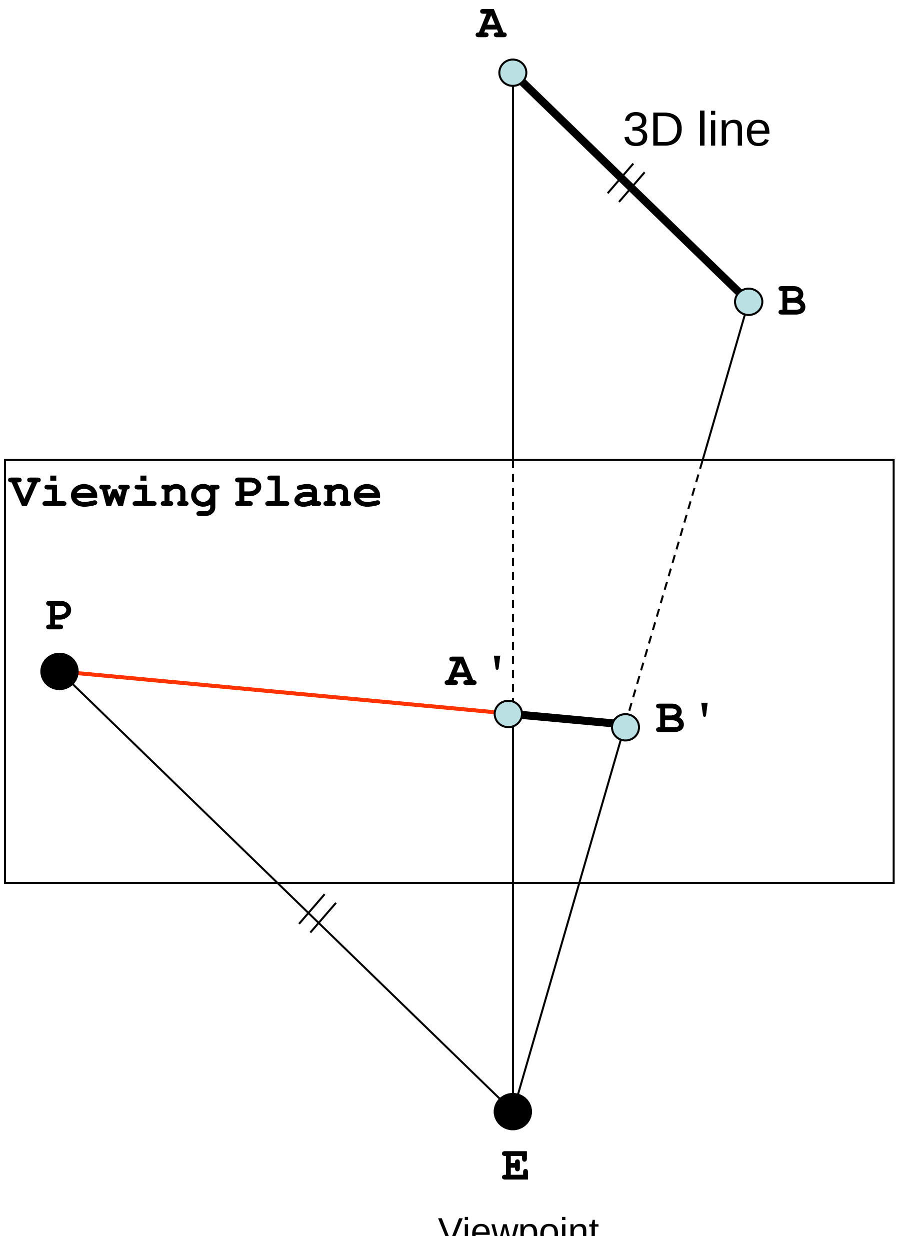
Vanishing Points

- Perspective projection of a group of parallel lines intersects at a single *vanishing point*
  - Unless the group is parallel to the projection plane

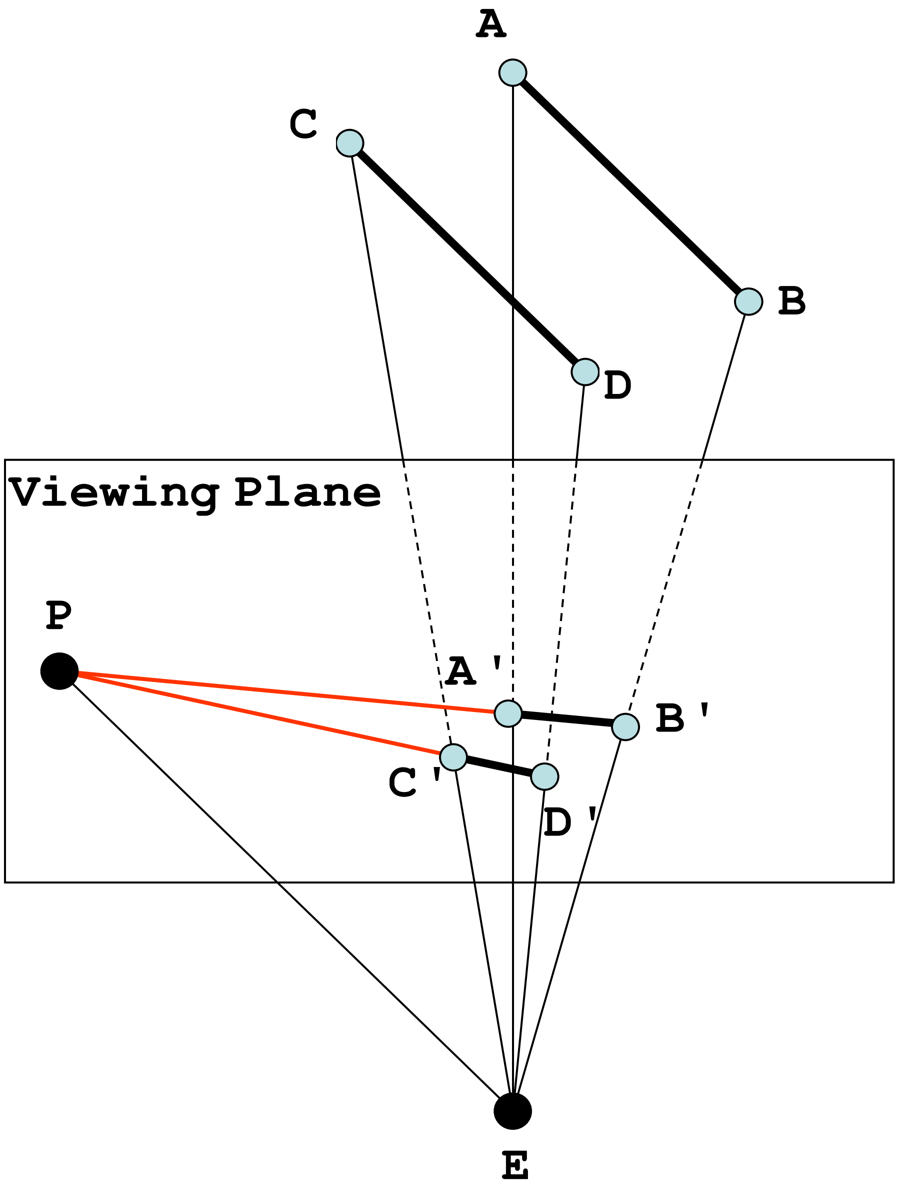
- Why?




A
3D line
B
Viewing Plane
A'
B'
E
Viewpoint



1. Find P on the viewing plane so that **PE is parallel to AB**
2. **P,A',B' lie on the same line**, because of these facts:
    1. A,B,P,E defines a plane.
    2. P,A',B' all lie on the plane of ABPE
    3. P,A',B' also lie on the viewing plane.
    4. P,A',B' all lie on the **intersecting line** between viewing plane and the plane of ABPE.



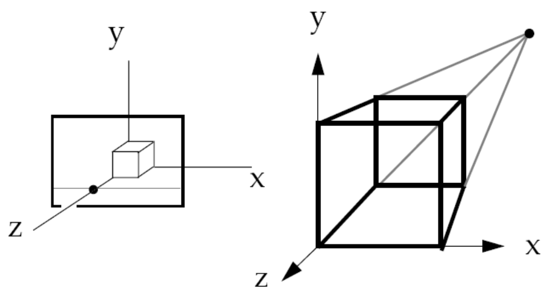
Similarly, P,C',D' are **co-linear** for any CD parallel to PE.

Conclusion: P is the **vanishing point** (unless AB is parallel to the viewing plane, in which case P does not exist)

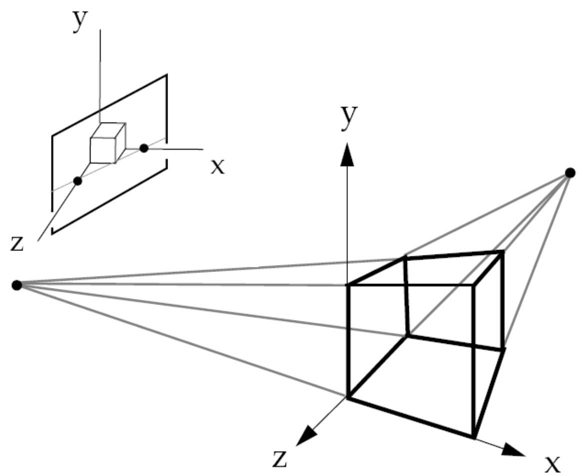
Types of Perspective Projections

- Based on number of vanishing points for lines parallel to the three coordinate axes
  - Determined by # of axes parallel to the viewing plane

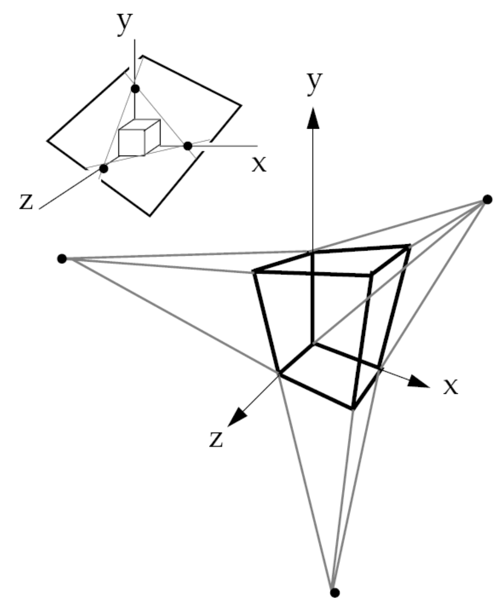
A unit cube:



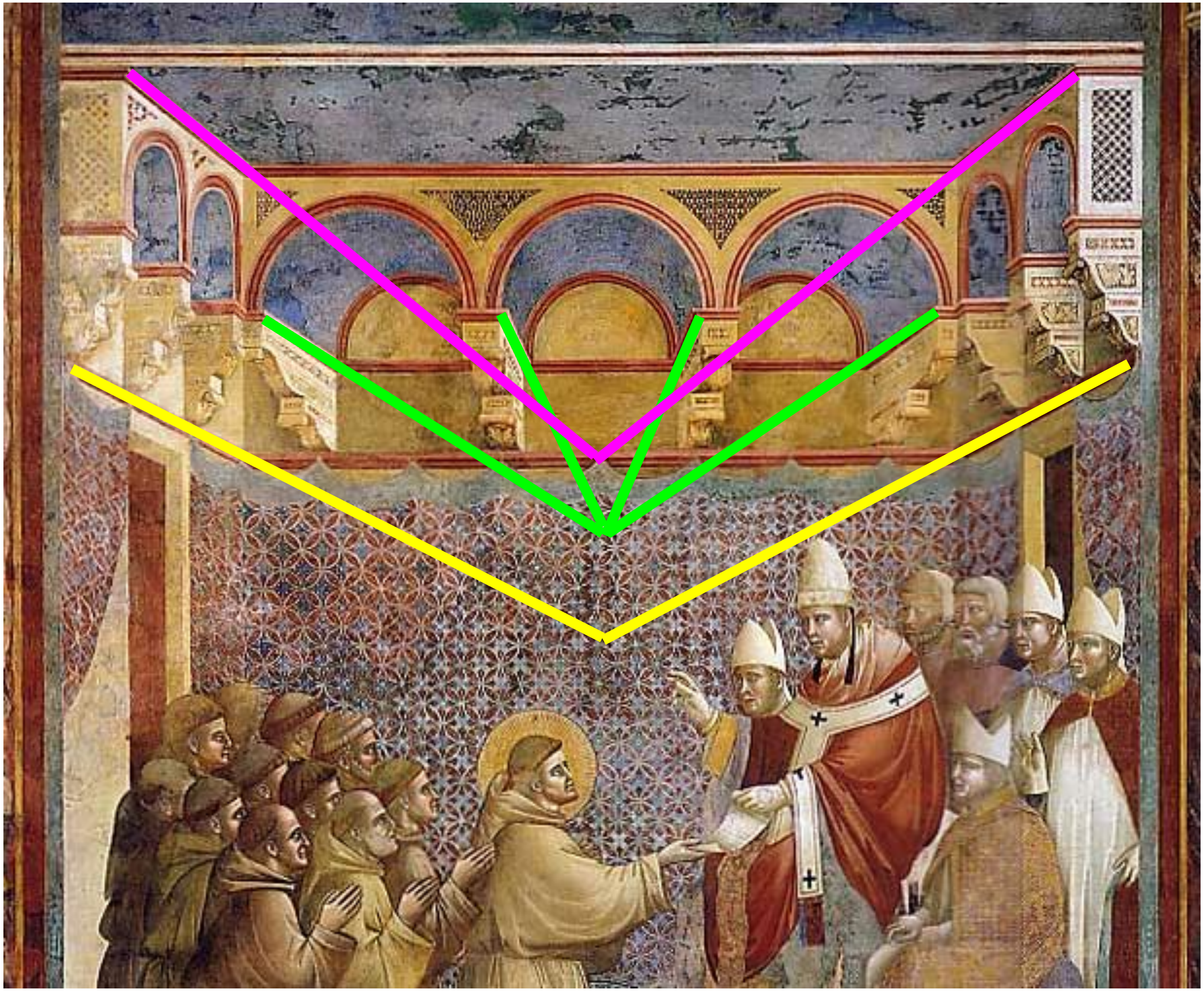
One-point Perspective
(view plane parallel to 2 axes)



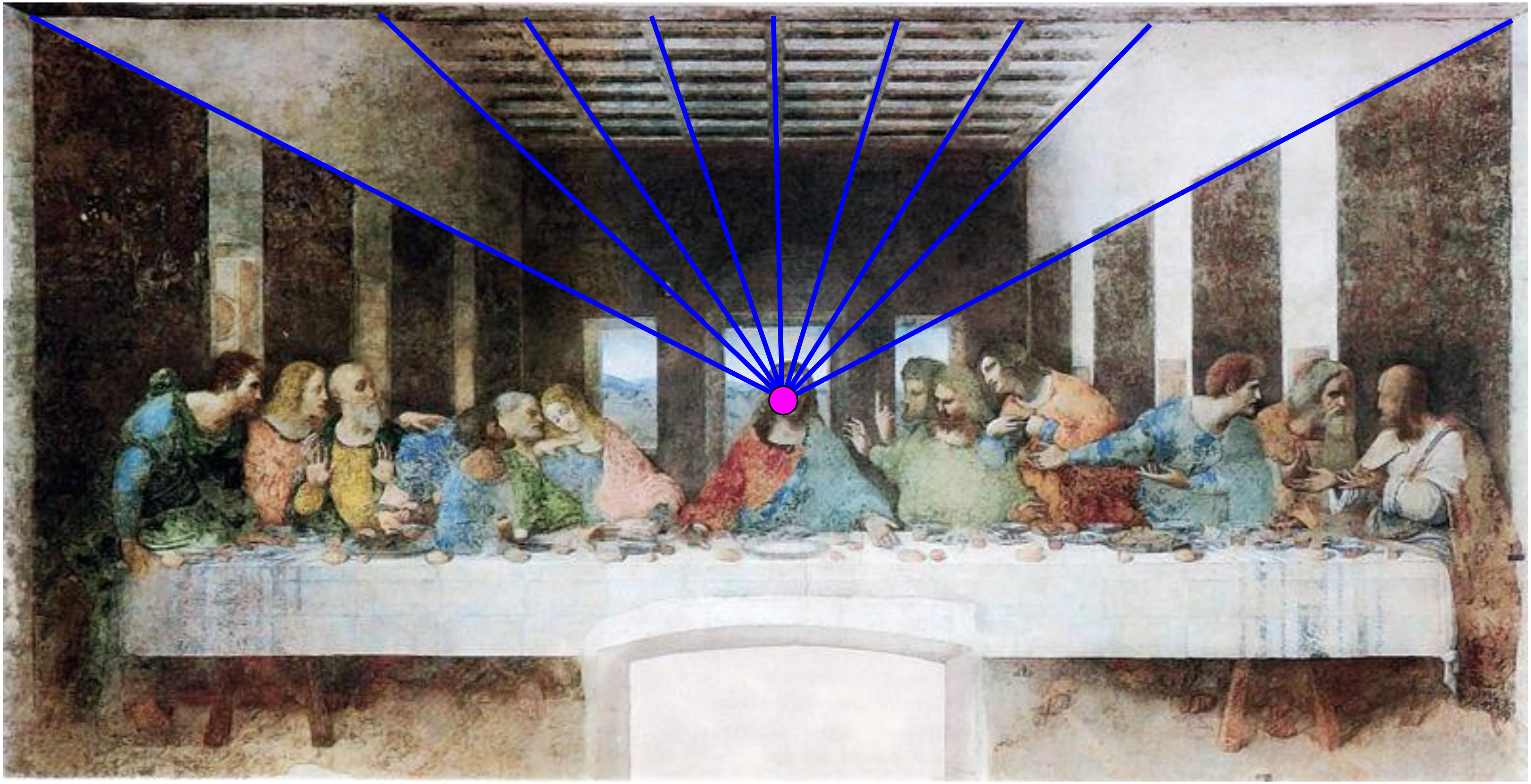
Two-point Perspective
(view plane parallel to 1 axis)



Three-point Perspective
(view plane not parallel to any axis)

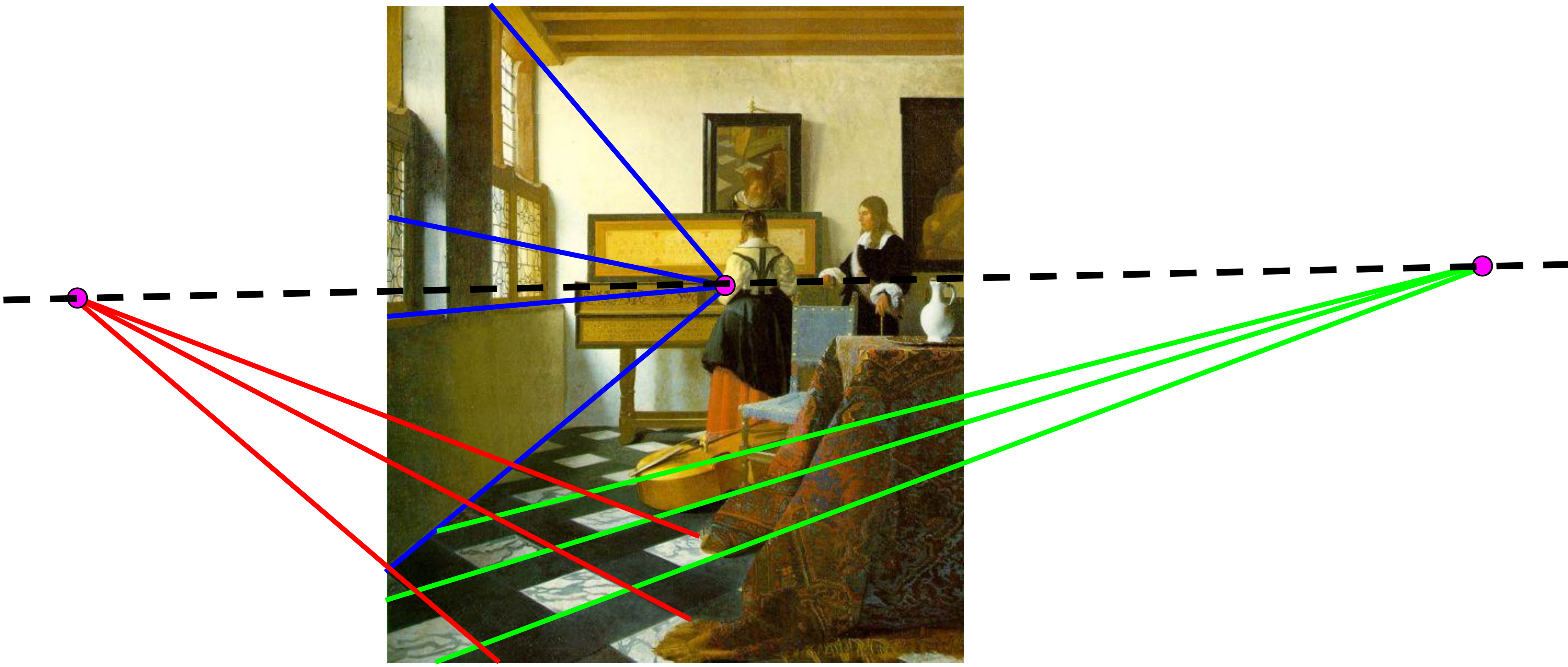


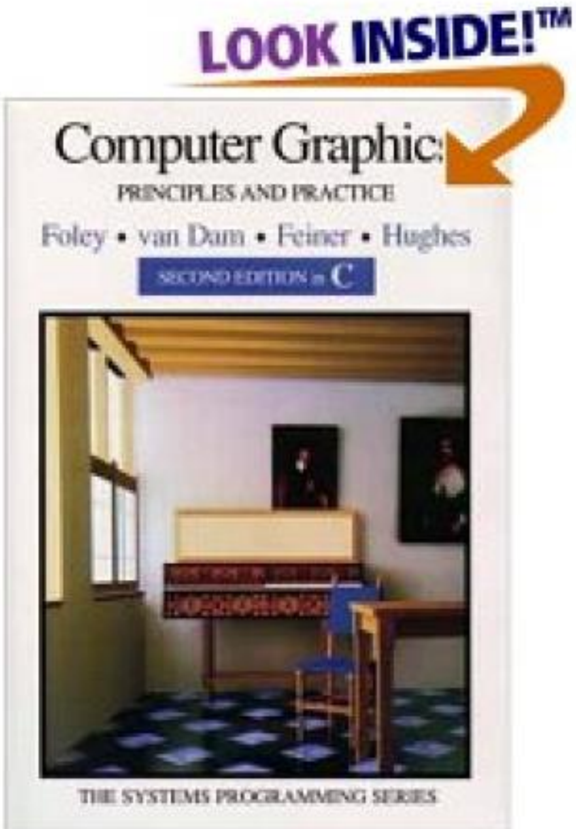
Giotto, Franciscan Rule Approved, 1295-1300

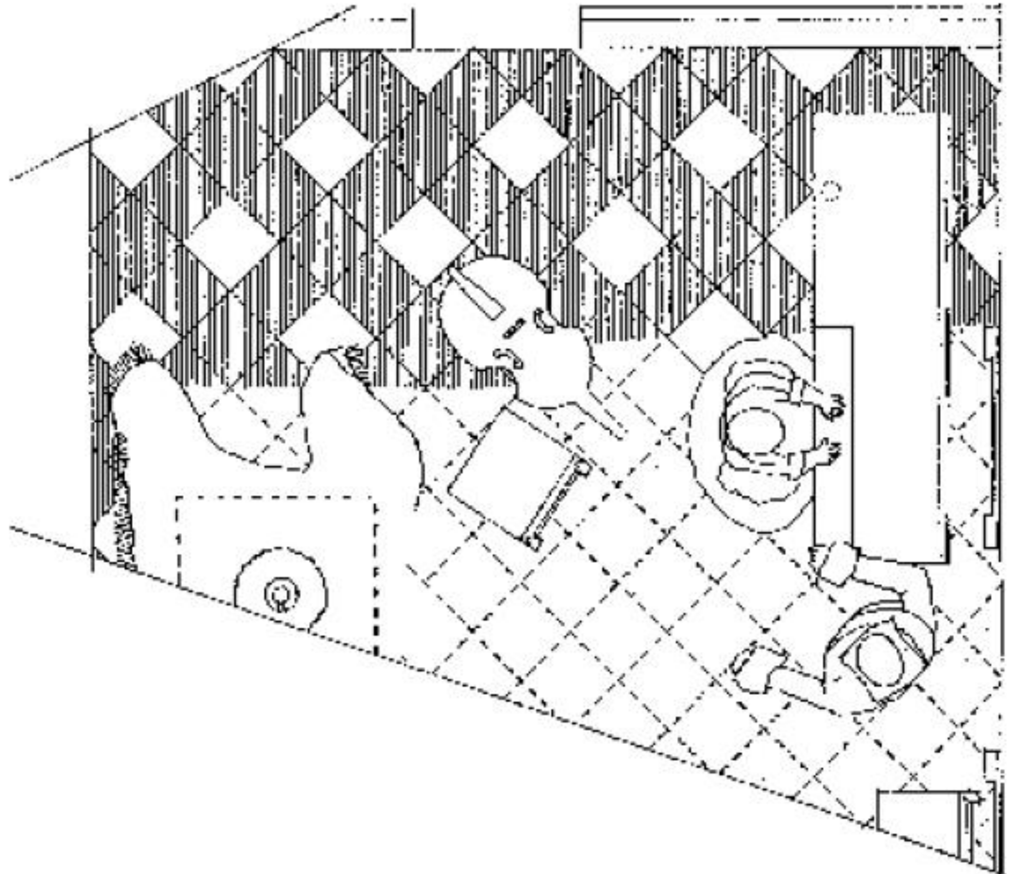


Leonardo da Vinci, **The Last Supper**, 1495–1498

Jan Vermeer, **The Music Lesson**, 1662

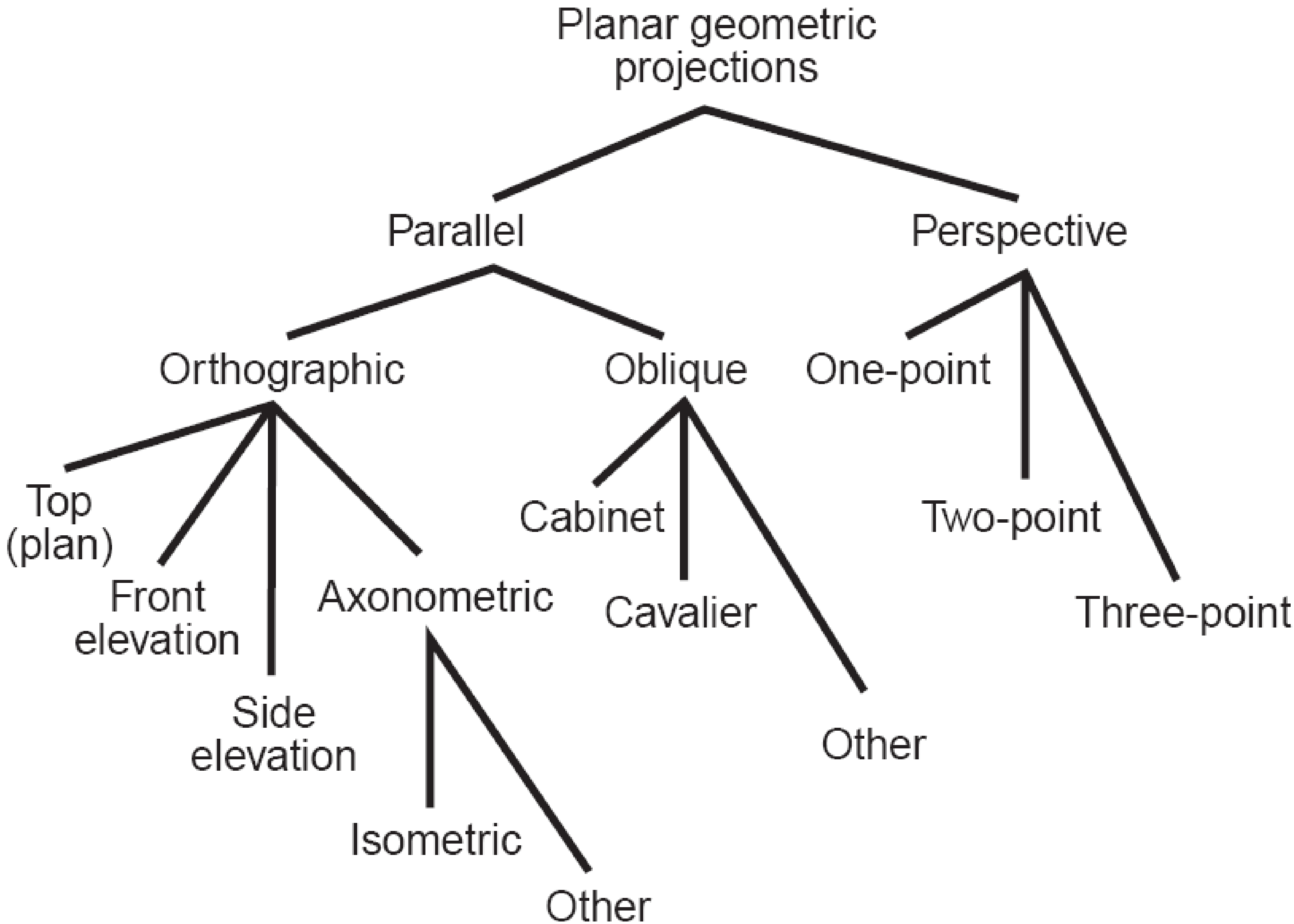




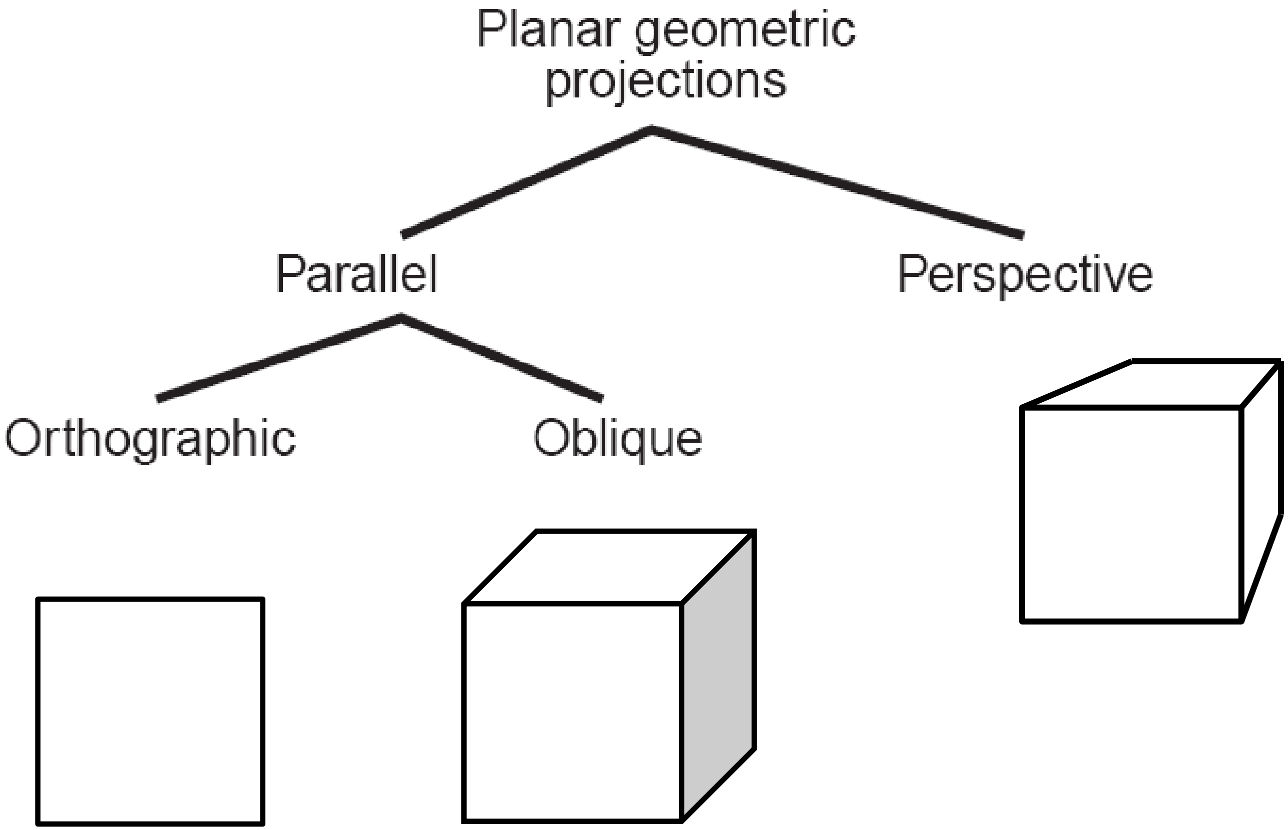


3D Reconstruction of The Music Lesson

Classification of Projections



Classification of Projections

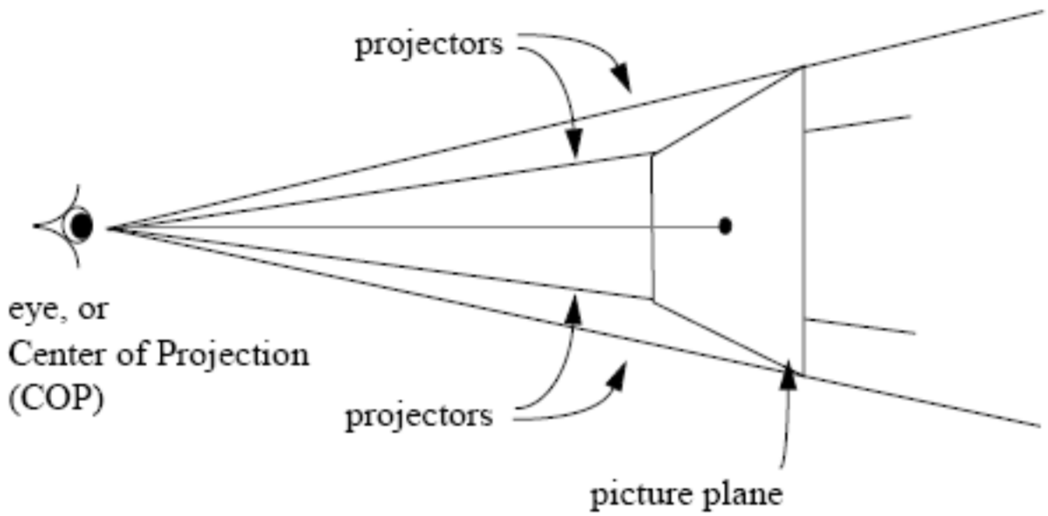


CSC 321 Computer Graphics

Computer Projection 1

Review

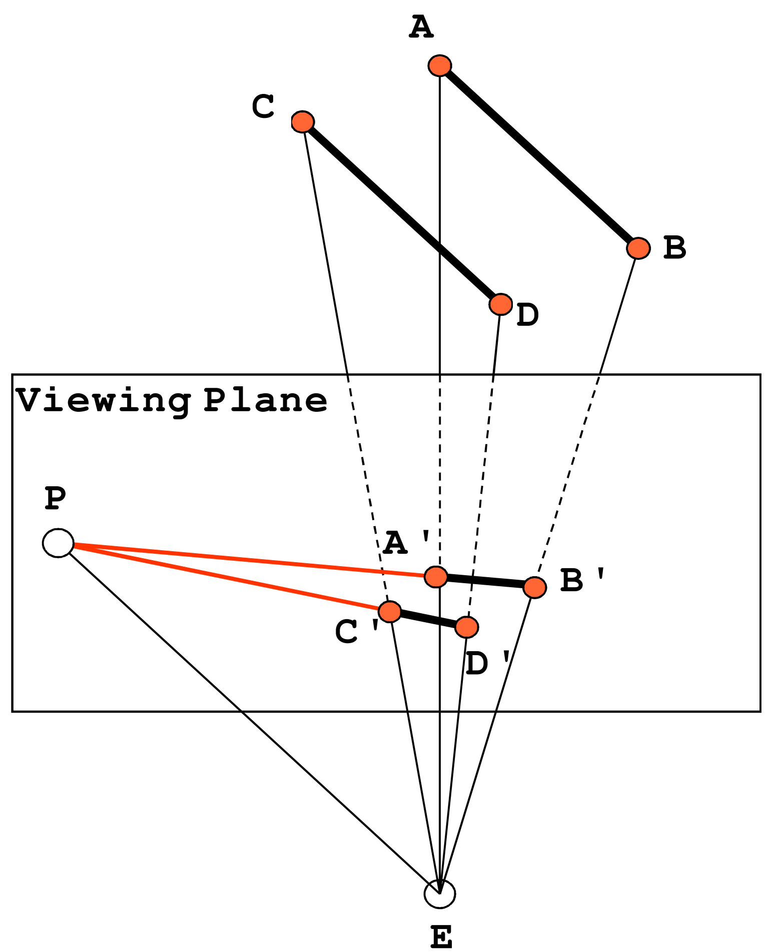
- In the last lecture
  - Definition: view point, view plane, projectors



  - Types of projection
    - Parallel (orthographic, oblique): parallel projectors (COP at infinity)
    - Perspective: projectors as rays from COP

Review

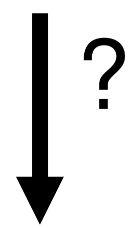
- In the last lecture
  - Geometric construction of Vanishing Points in perspective projection
    - For parallel lines in the direction v
    - Vanishing point after projection is the intersection of viewing plane with the ray from eye in the direction v



Preview

- In this lecture (and next)
  - How to perform projection in the computer? Or, given a point in 3D, where do I draw it on the 2D computer screen?

World Coordinate: $\{x_w, y_w, z_w\}$



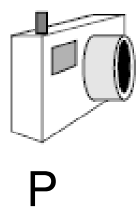
Screen Coordinate: $\{x_s, y_s\}$

Virtual Camera

- Programmer's reference model
- General parameters
  - Position of camera
  - Orientation
  - Field of view (wide angle, telephoto)
  - Clipping plane (near distance, far distance)
  - Perspective or parallel projection?
  - Focal distance
  - Tilt of view/film plane (for oblique views)

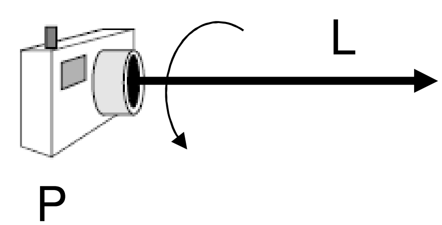
Position

- From where the camera is
  - Like a photographer choosing the vantage point to shoot a photo
- Any 3D point $\mathbf{P} = \{p_x, p_y, p_z\}$
  - Use right-hand rule for coordinate axes
    - Align right hand fingers with +X axis
    - Curl fingers towards +Y axis
    - Your thumb points towards +Z axis



Orientation – Look Vector

- Where the camera is looking
- Any 3D vector $\mathbf{L} = \{l_x, l_y, l_z\}$
  - Not necessarily a unit vector
- Look vector alone is not sufficient to describe orientation…

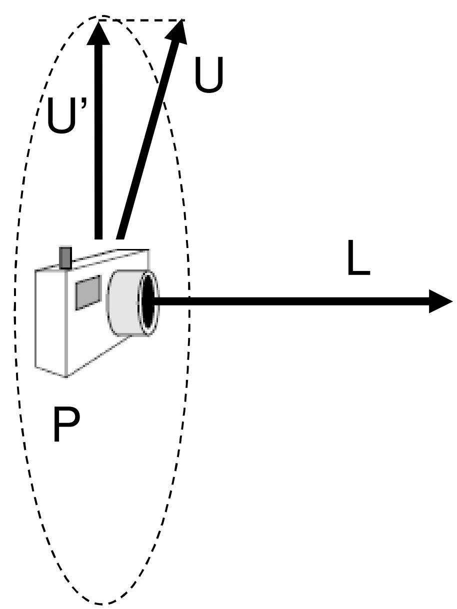


Orientation – Up Vector

- How the camera is rotated around the look vector
  - If you are holding the camera horizontally or vertically, or in between.
- Any 3D vector $\mathbf{U} = \{u_x, u_y, u_z\}$
  - Not necessarily orthogonal to look vector L
  - Actual "Up-right" direction, U', is:

$$\mathbf{U'} = \mathbf{U} - \frac{\mathbf{U} \cdot \mathbf{L}}{\mathbf{L} \cdot \mathbf{L}} * \mathbf{L}$$

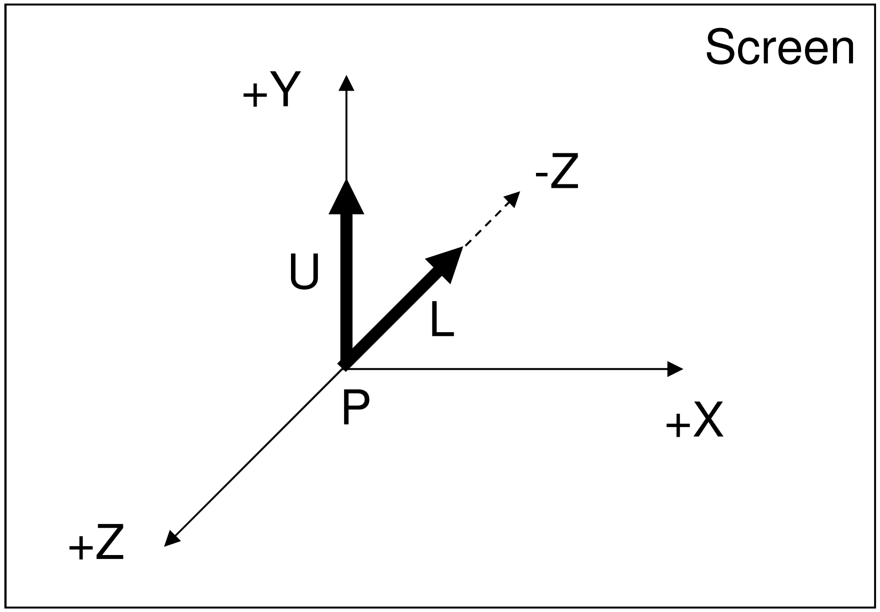
(projecting U onto the plane orthogonal to L)



Default Position, Orientation

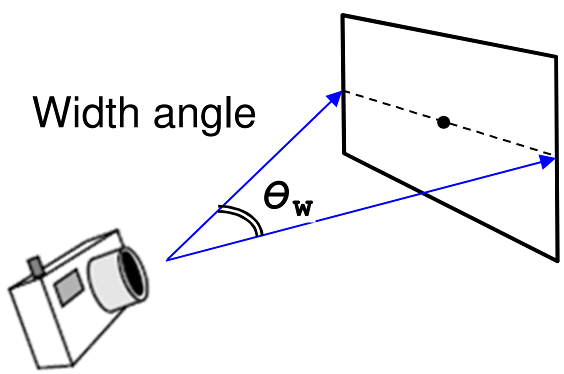
- Camera at origin, looking down –Z axis, and in upright pose
  - E.g., in OpenGL

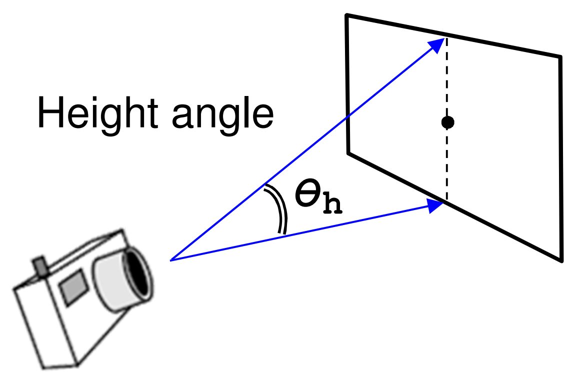
```
P = {0, 0, 0}
L = {0, 0, -1}
U = {0, 1, 0}
```



Viewing Angle

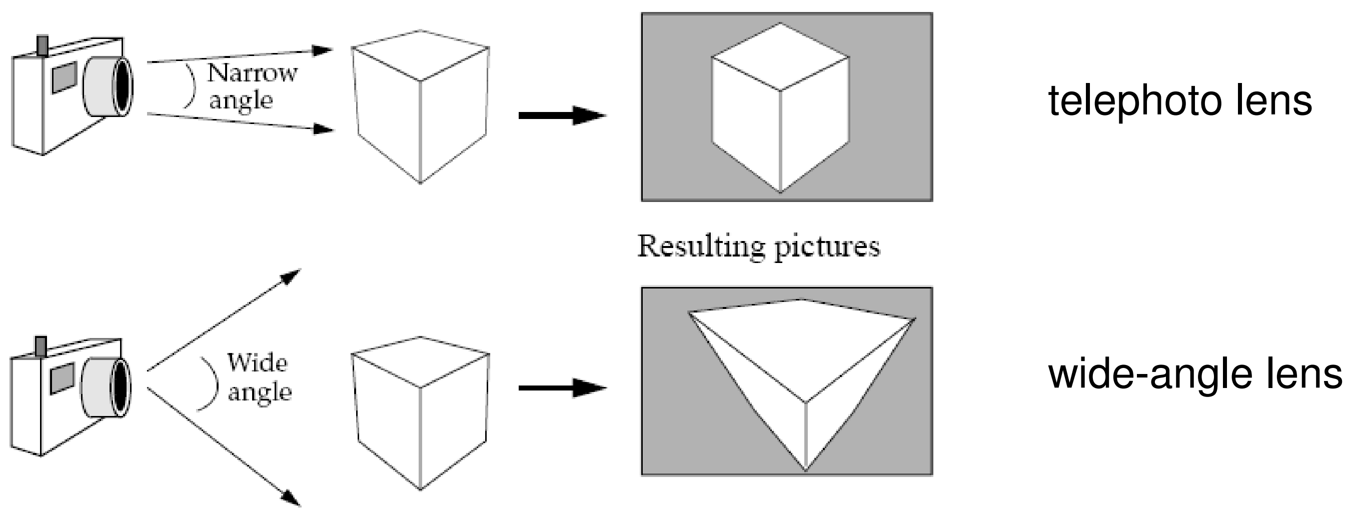
- Describes the field of view
  - Like choosing a specific type of lens, e.g., a wide-angle lens or telephoto lens
- Width and height angles θ_w, θ_h
  - Assuming the view region is a rectangle





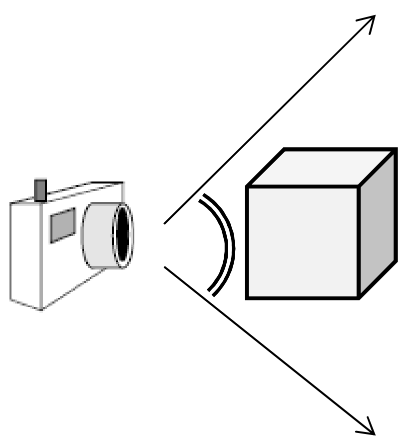
Viewing Angle

- Determines amount of perspective distortion
  - Small angles result in near-parallel projectors, hence little distortion
  - Large angles result in widely varying projectors with large distortion

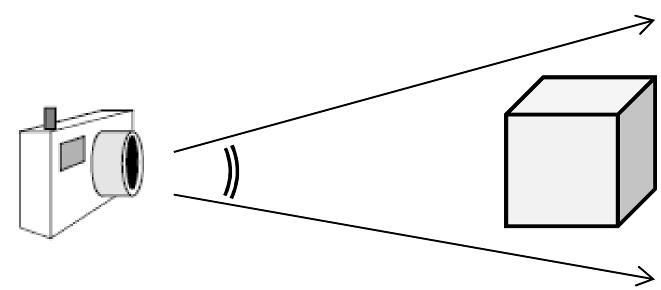


Viewing Angle

- When keeping the size of the main object in view, longer distances gives narrower view angle

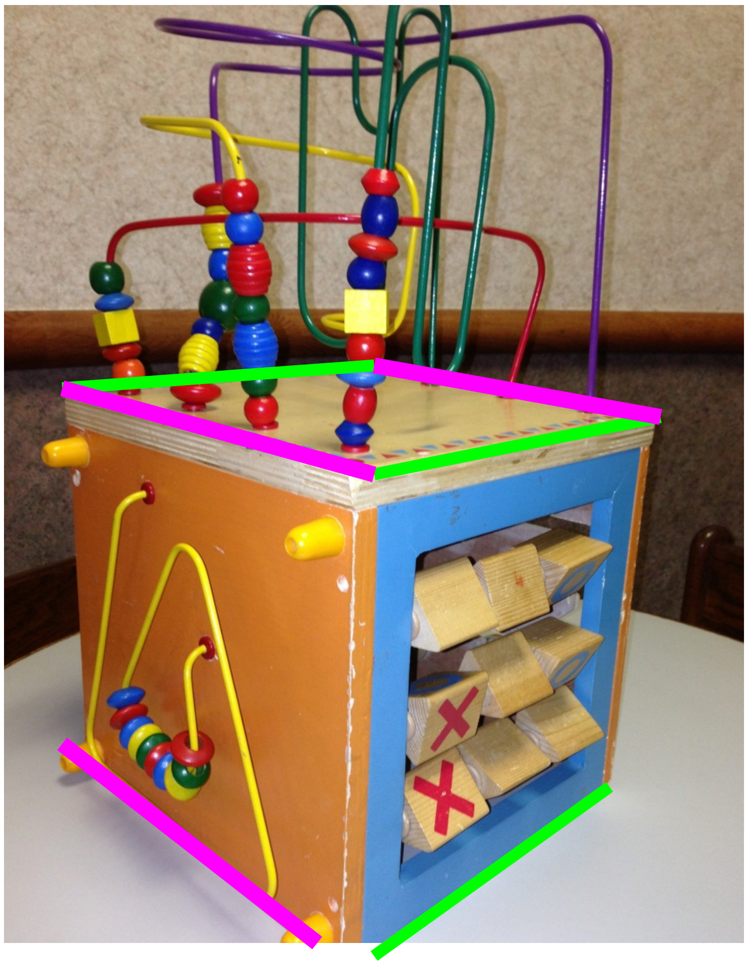


Close-up (wide angle)

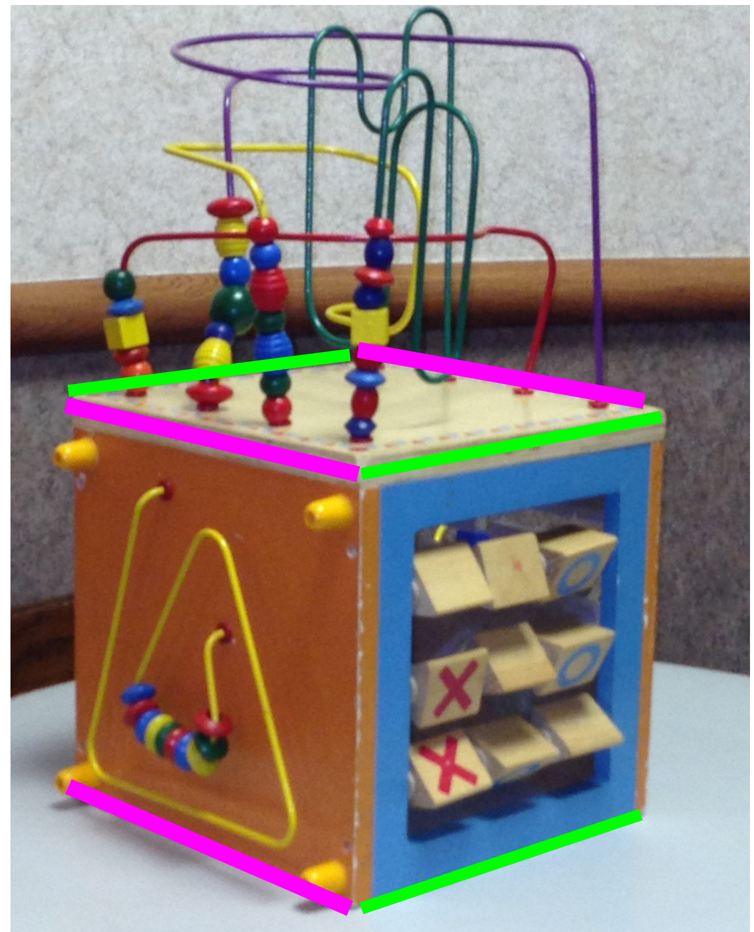


Far away (narrow angle)

Viewing Angle



Close-up (wide angle)



Far away (narrow angle)

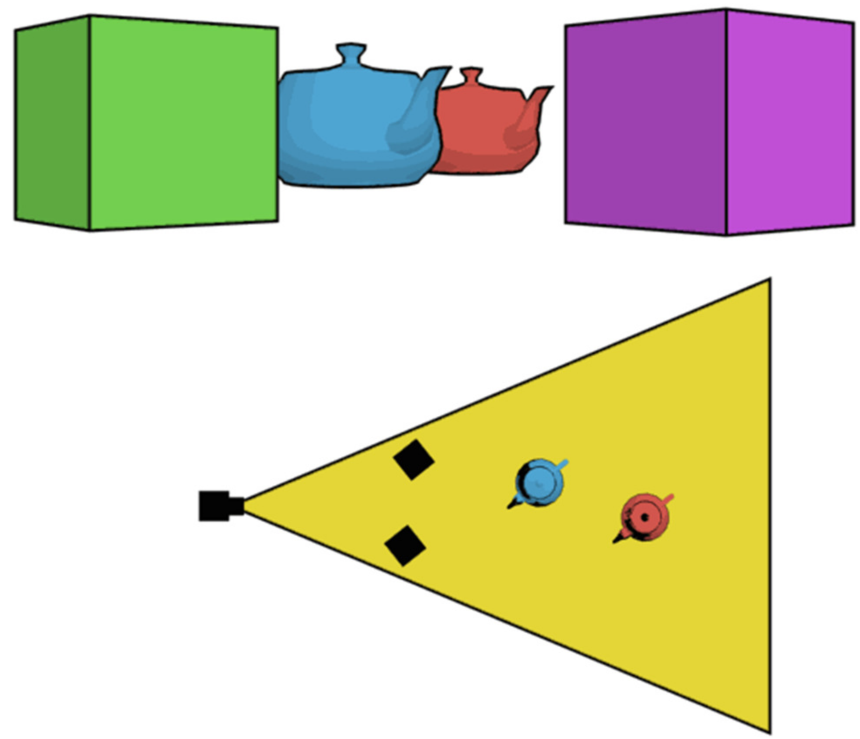
Viewing Angle

Close-up (wide angle)

Far away (narrow angle)

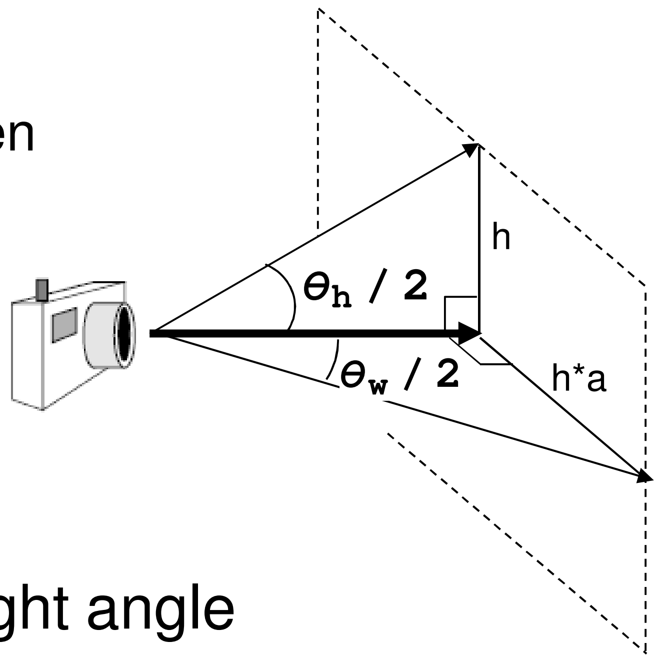
Viewing Angle

- Fun example: dolly-zoom effect (or “Hitchcock zoom”)
  – Moves away the camera and shrinks the viewing angle at the same time, so that the main subjects stays the same size on screen
  – The background gets “closer”, and perspective distortion lessens



Viewing Angle

- Aspect Ratio α
  - Ratio of width over height of the screen
    - 1:1 (square)
    - 4:3 (NTSC)
    - 16:9 (HDTV)
    - 2.35:1 (Widescreen Films)

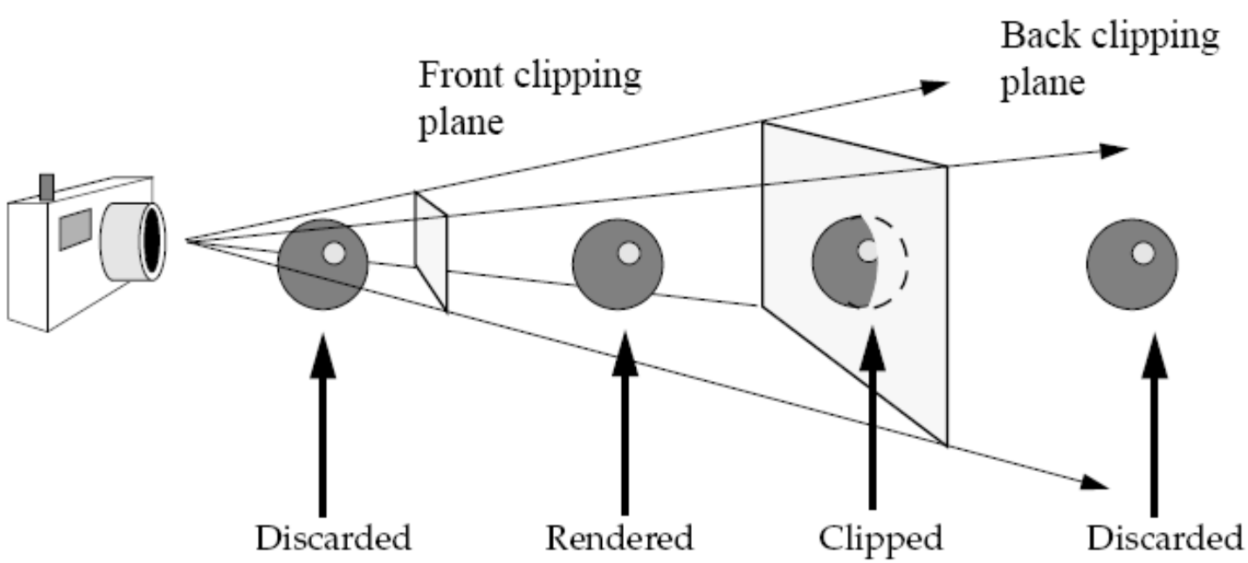


- Width angle as aspect ratio and height angle
  - Compute width angle:

$$\theta_w = 2\,\mathrm{ArcTan}\left[\mathrm{Tan}\left[\frac{\theta_h}{2}\right] * \alpha\right]$$

Clipping Planes

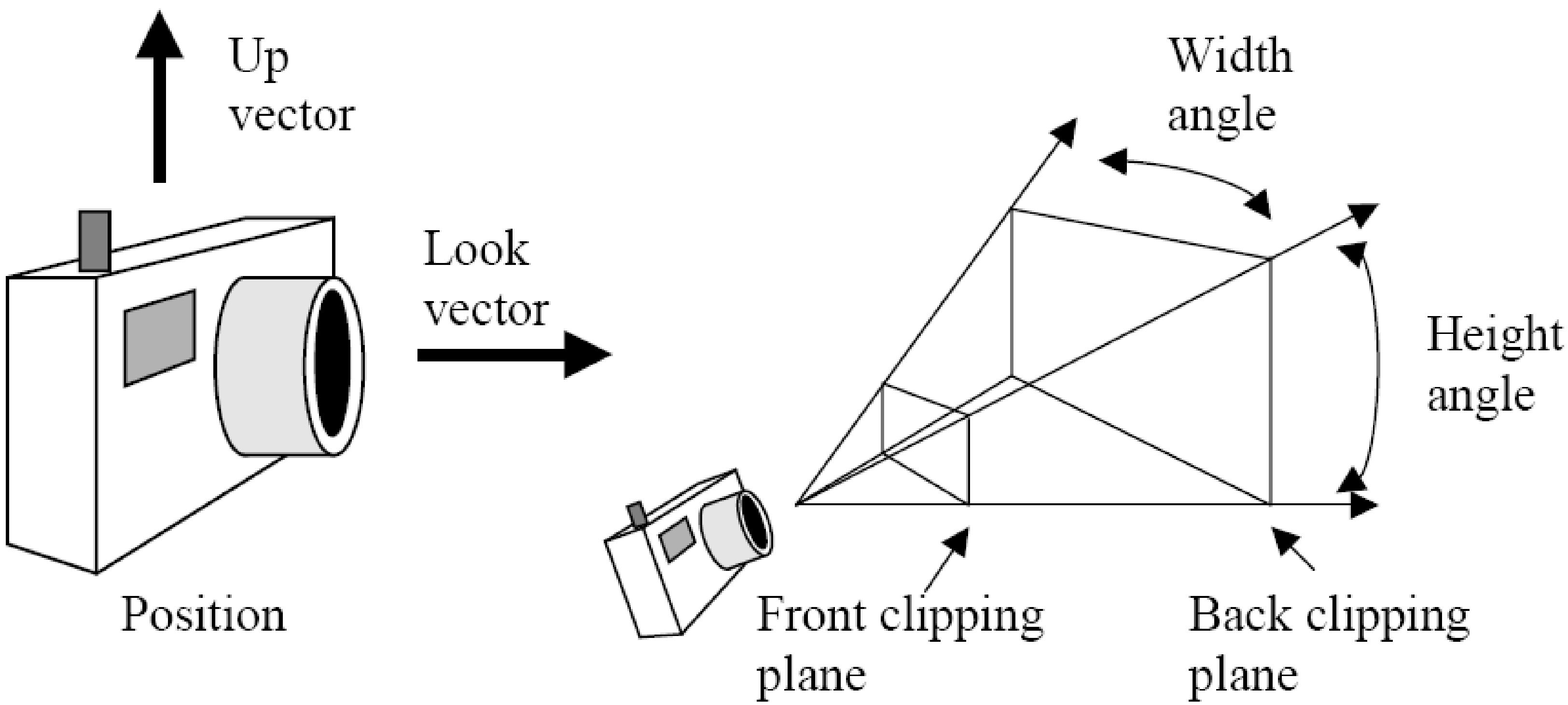
- Restricts visible volume between *near* and *far* clipping planes
  - Objects closer than the near plane or further than the far plane are not drawn
  - Objects intersecting the two planes are clipped
- Defined as distances $\mathbf{d_n}, \mathbf{d_f}$ from camera along look vector



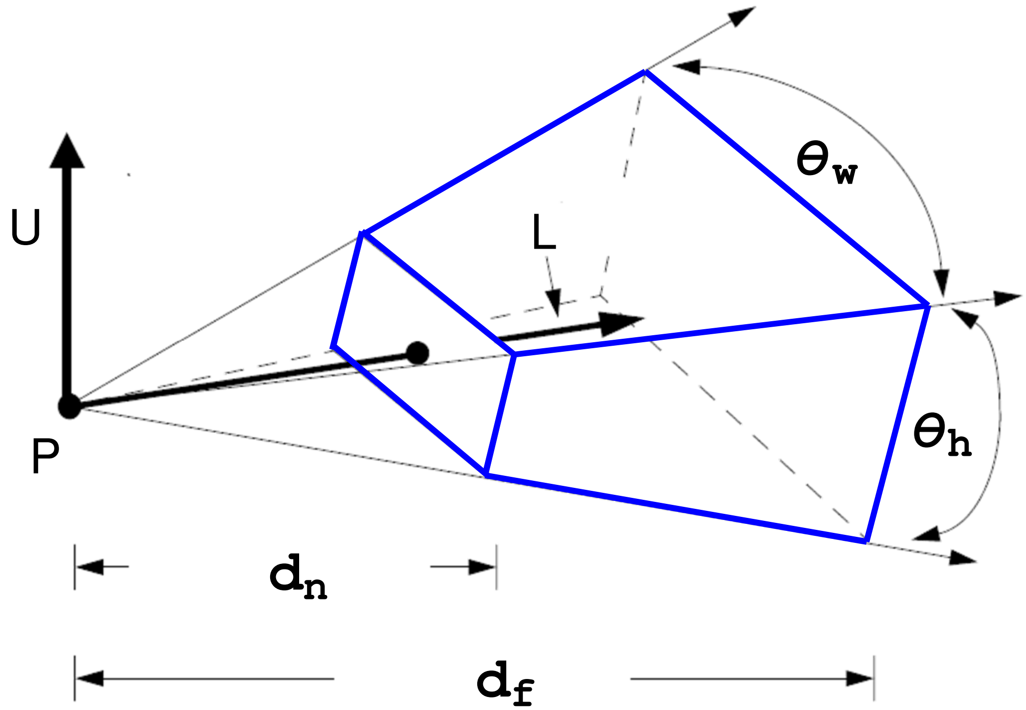
Clipping Planes

- Why do we need near plane
  - Avoid drawing things too close to camera
    - They will appear with large distortion, and may block view
  - Avoid drawing things behind the camera
    - They will appear upside-down and inside-out
- Why do we need far plane
  - Avoid drawing things too far away
    - They will complicate the scene
    - They appear small on the screen anyway
    - Saving rendering time

Perspective Camera Model



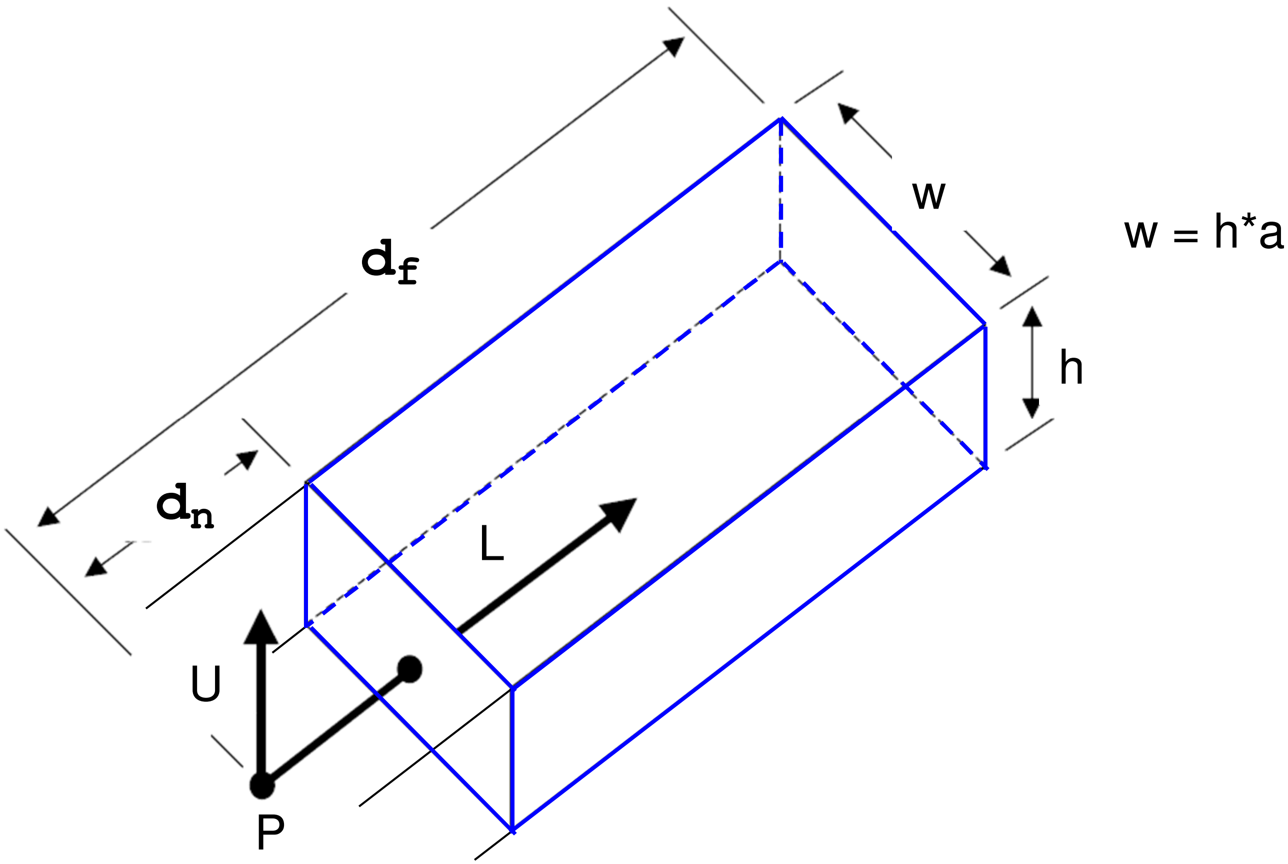
Perspective Camera Model



- View frustum: a truncated pyramid region that the camera can "see"

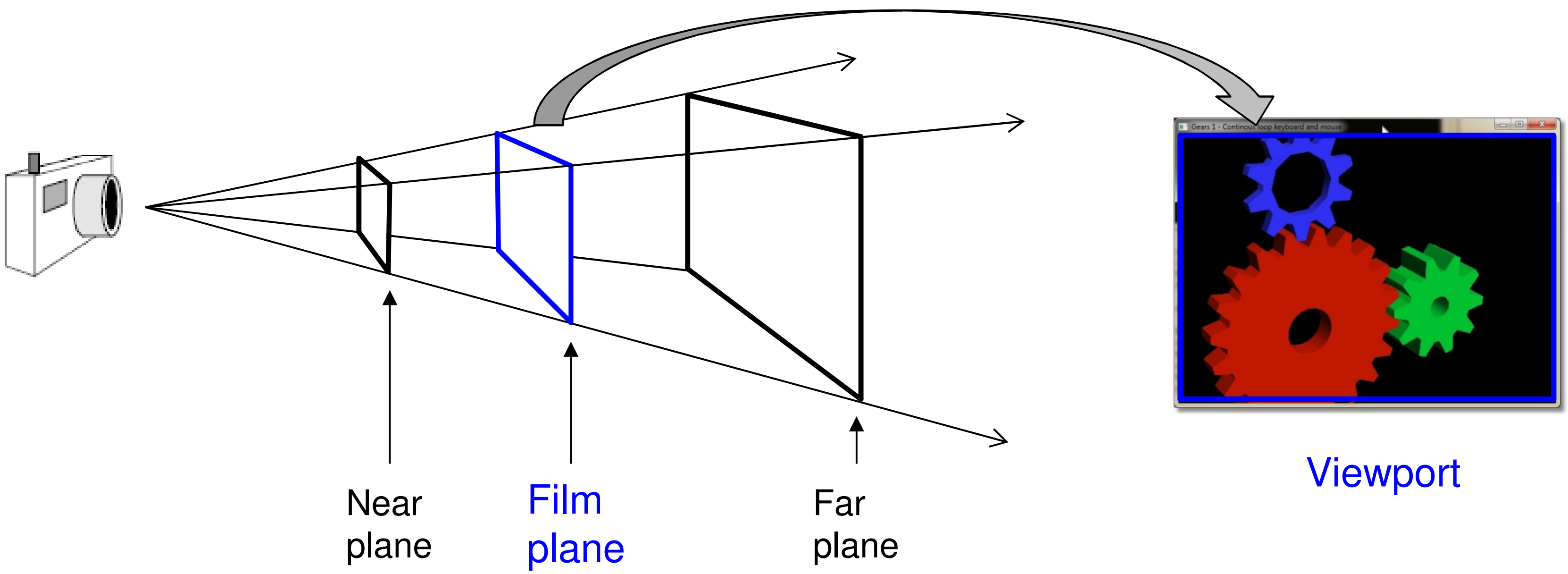
Orthographic Camera Model

- Width **w** and height **h** replace viewing angles
  - Both width angle and height angle are effectively zero



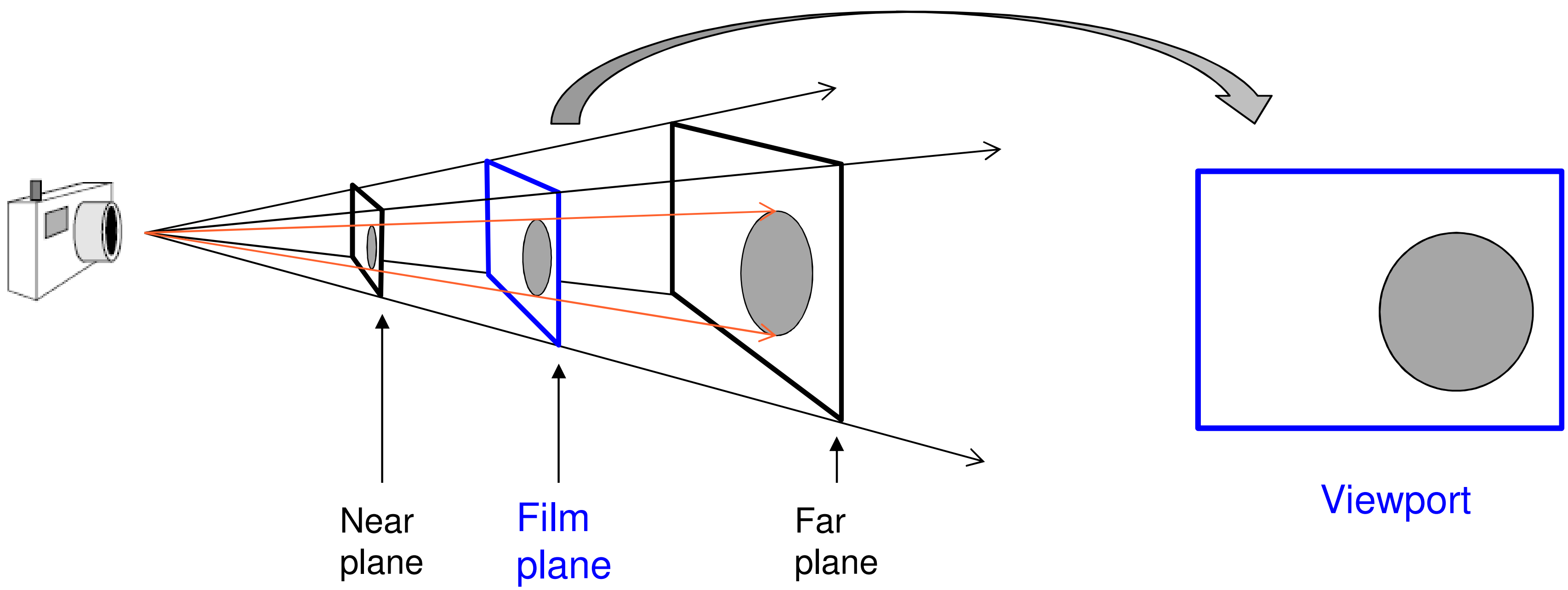
Film Plane and Viewport

- Film Plane
  - *Any* plane parallel to the near/far clipping planes.
- Viewport
  - A rectangular region on the screen displaying what's projected on the film plane (may have different aspect ratio as the film plane)



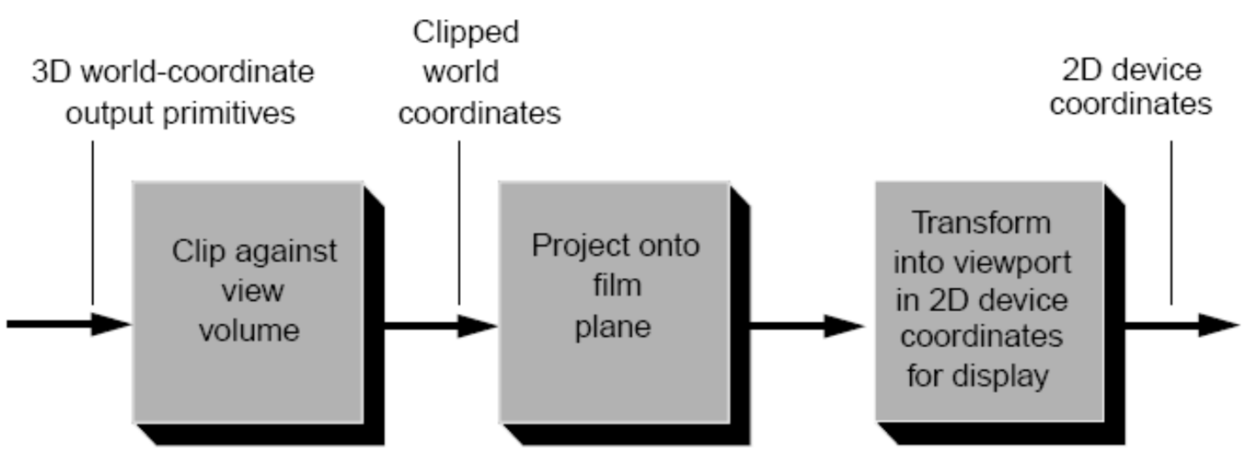
Film Plane and Viewport

- No matter where the film plane is, the final image shown in the viewport is the same!



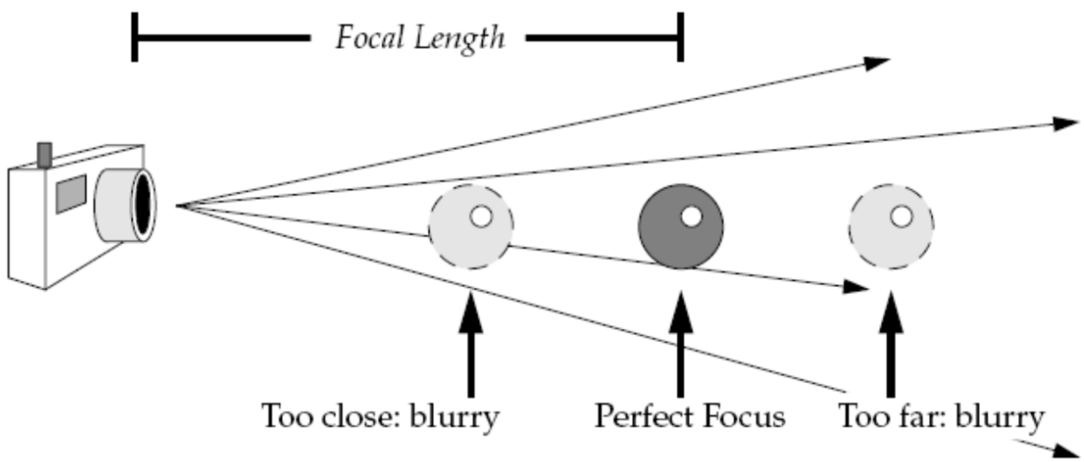
What next?

- Three steps
  - **Clipping**: removes geometry outside the frustum
  - **Projecting**: transforms 3D coords. to 2D coords. on the film plane
  - **Viewport transformation**: gets pixel coordinate in the viewport



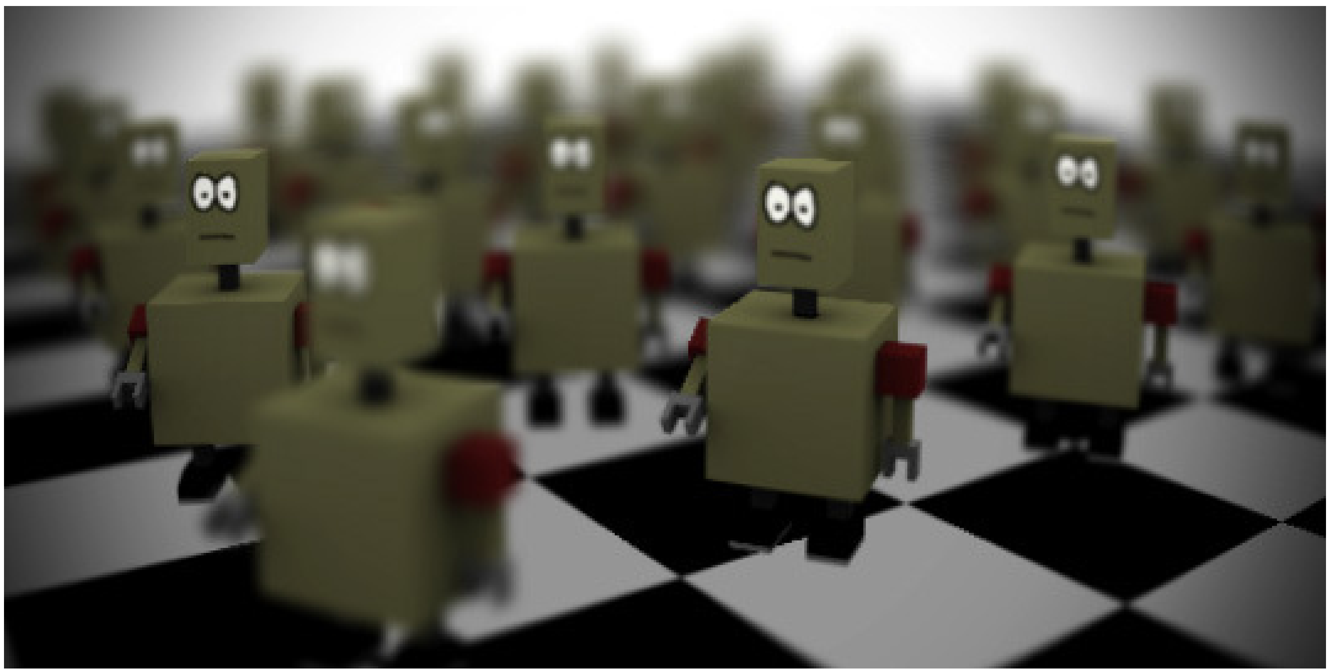
Other Camera Models

- **Focal length**
  - Approximates behavior of real camera lens
    - Objects at distance of focal length from camera are rendered in focus; other objects get blurred
  - Focal length used in conjunction with clipping planes
    - Only objects within view volume are rendered, whether blurred or not.



Other Camera Models

- **Focal length**



Rendering with focal blur

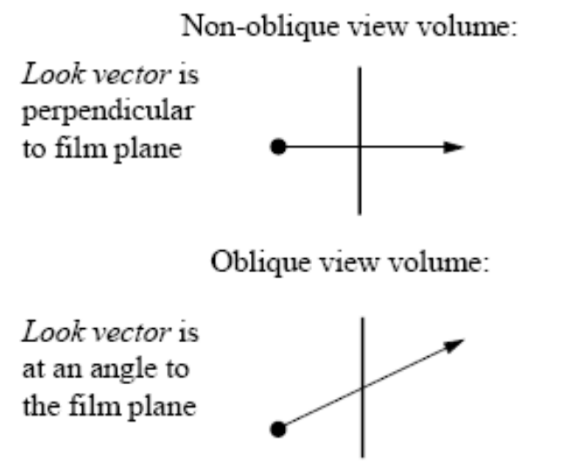
Other Camera Models

- **Focal length**
  - Focal blur can serve as a cue for depth and (even) size

Held et al., "Making Big Things Look Small: Blur combined with other depth cues affects perceived size and distance“, 2008

Other Camera Models

- **Oblique projection**
  - Look vector *not* perpendicular to film plane



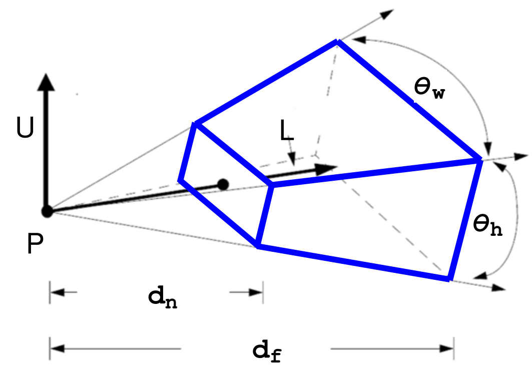
Nikon PC-E Nikkor 24mm Tilt/Shift lens

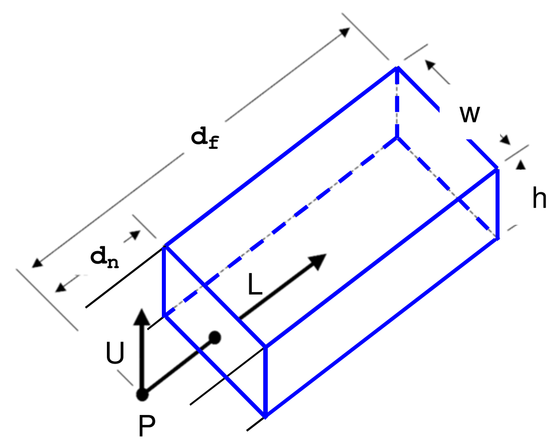
CSC 321 Computer Graphics

Computer Projection 2

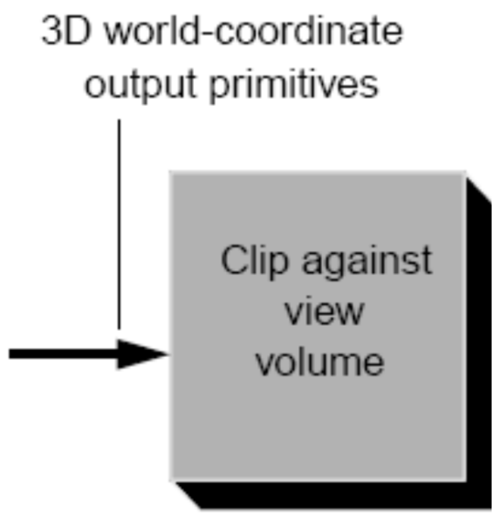
Review

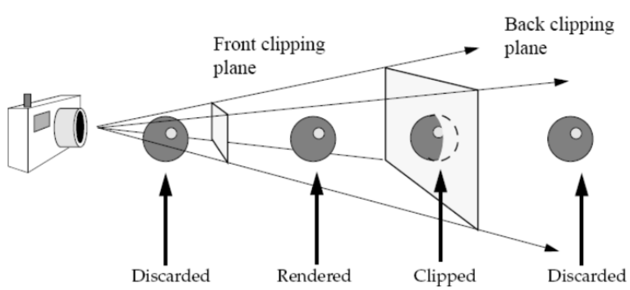
- In the last lecture
  - We set up a Virtual Camera
    - Position
    - Orientation
    - Clipping planes
    - Viewing angles
    - Orthographic/Perspective
- We are ready to project!

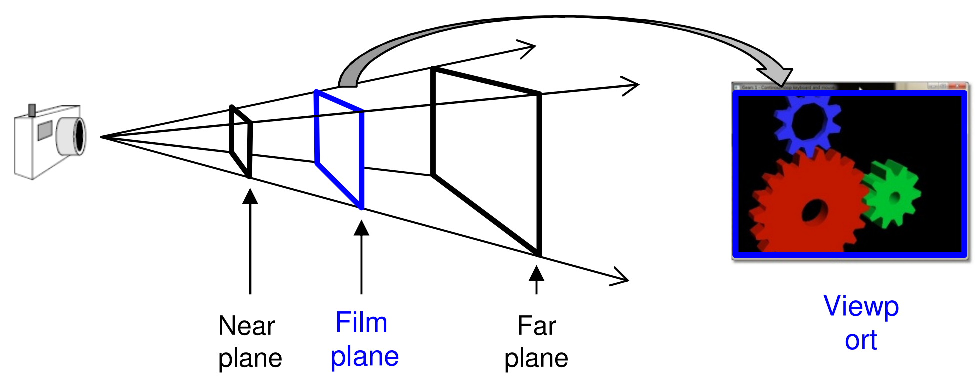




Preview

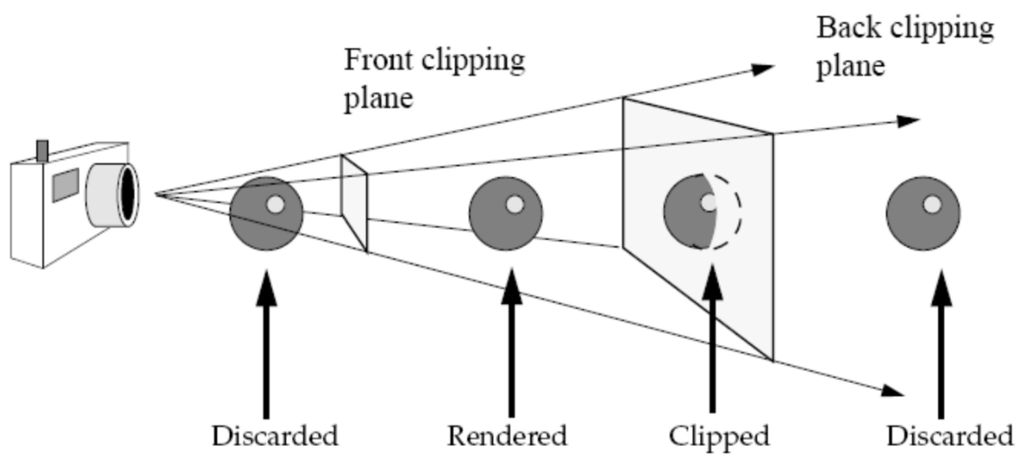

3D world-coordinate
output primitives
Clip against
view
volume


Front clipping
plane
Back clipping
plane
Discarded
Rendered
Clipped
Discarded

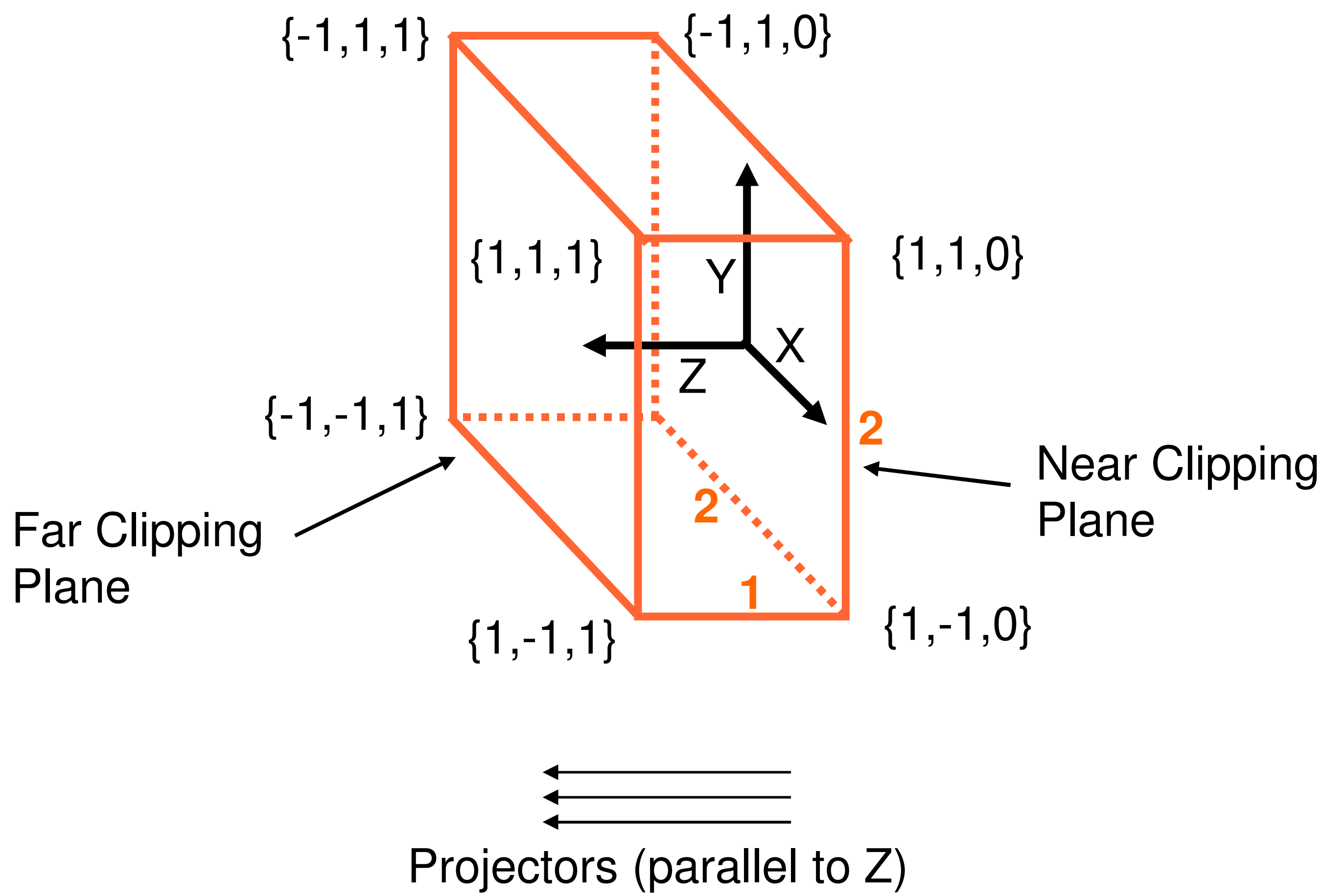

Near
plane
Film
plane
Far
plane
Viewp
ort

Preview

- The perspective view frustum (i.e., a truncated pyramid) is non-trivial to clip against
  - We first transform the frustum to a canonical volume



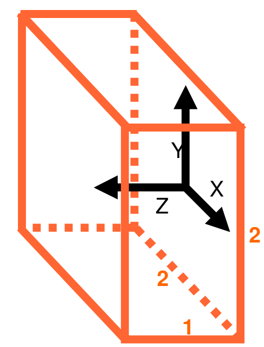
Canonical View Volume



Canonical View Volume

- Canonical view volume makes things easy:
  - Easy clipping: Clip against the coordinates range

$$-1 \le x \le 1,\ -1 \le y \le 1$$
$$0 \le z \le 1$$



  - Easy projecting: drop the Z coordinate! (because viewing plane is the XY plane, and projectors are parallel to Z axis)

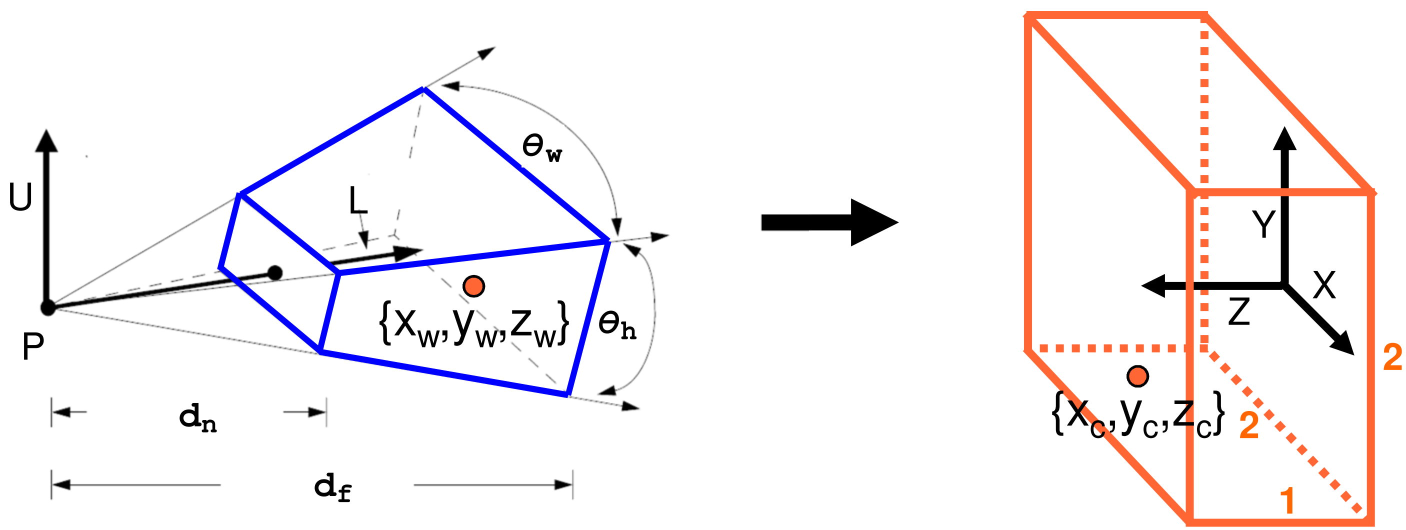
$\{x_c, y_c, z_c\}$ Coordinates in the canonical volume

$\downarrow$

$\{x_c, y_c\}$ Projected 2D coordinates

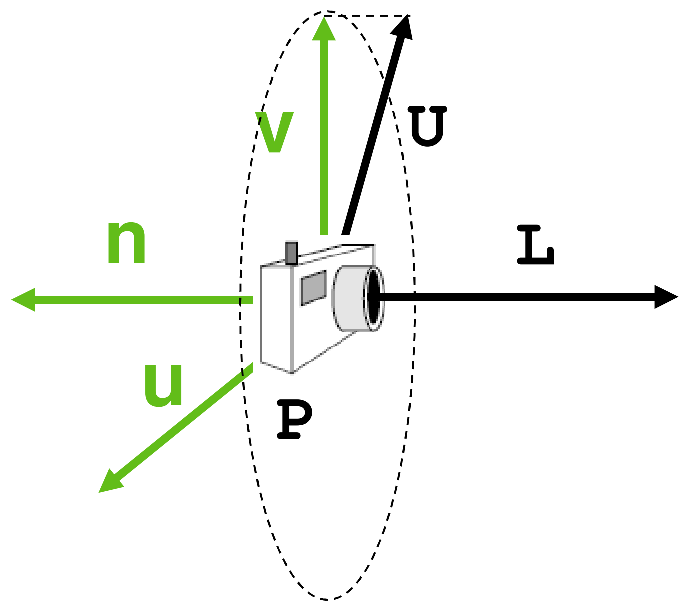
Viewing Transformation

- The transformation that warps the perspective frustum to the canonical view volume
  - Transforms world coordinates $\{x_w,y_w,z_w\}$ into canonical coordinates $\{x_c,y_c,z_c\}$



Camera Coordinate System

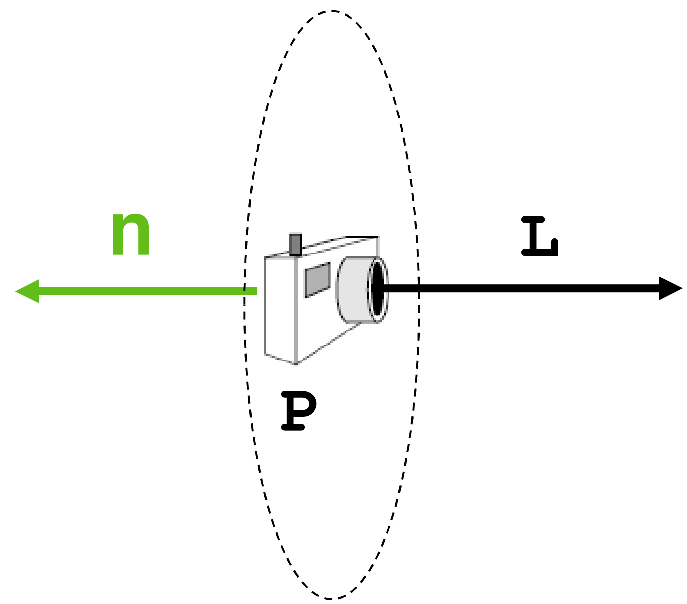
- First, let's setup a coordinate system for the camera
  - Origin at the camera
  - Three axes: right (u), straight-up (v), negative look (n)
    - Unit vectors forming an orthonormal, right-hand basis



Camera Coordinate System

- Computing n
  - Opposite to look vector L, normalized

$$\mathbf{n} = \frac{-\mathbf{L}}{|\mathbf{L}|}$$

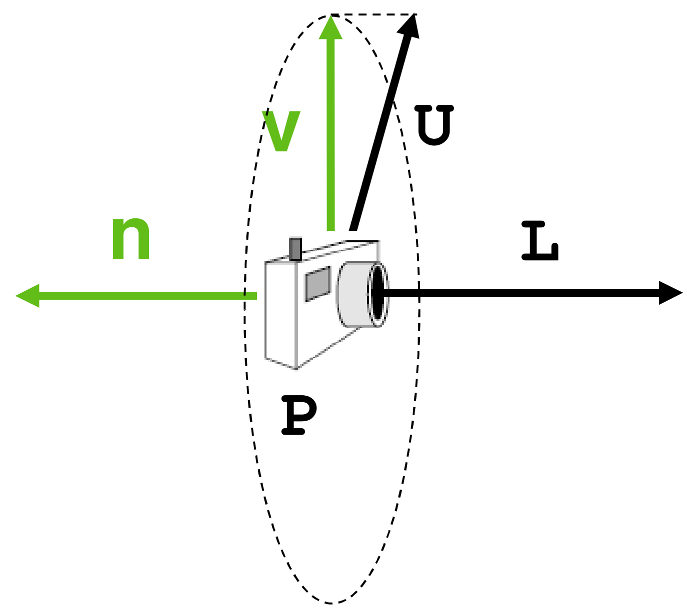


Camera Coordinate System

- Computing v
  - Projection of up vector U onto the camera plane, normalized

$$v' = U - (U \cdot n) * n$$

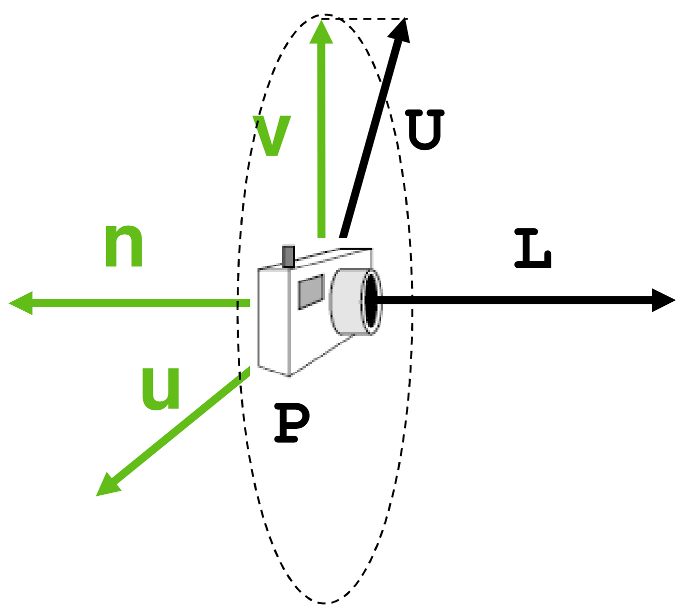
$$v = \frac{v'}{|v'|}$$



Camera Coordinate System

- Computing u
  - Cross product of v and n

$$\mathbf{u} = \mathbf{v} \times \mathbf{n}$$



Camera Coordinate System

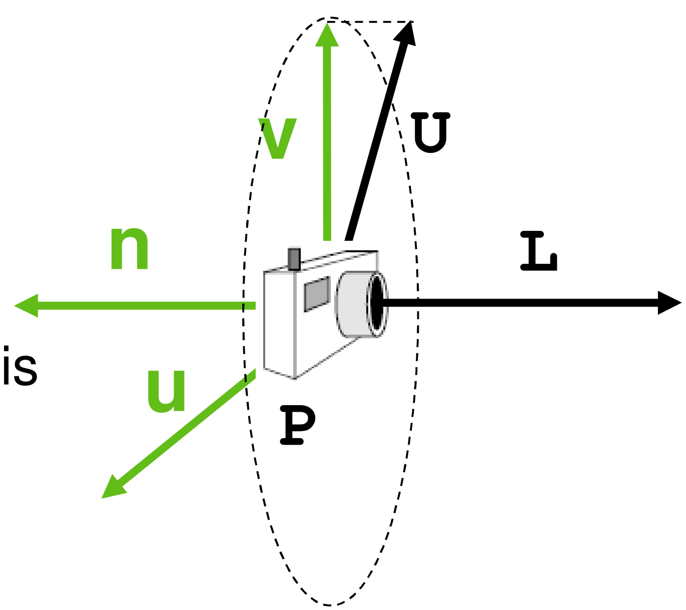
- Summary
  - Three axes, computed from look vector L and up vector U:

$$\mathbf{n} = \frac{-\mathbf{L}}{|\mathbf{L}|}$$

$$\mathbf{v} = \frac{\mathbf{U} - (\mathbf{U} \cdot \mathbf{n}) * \mathbf{n}}{|\mathbf{U} - (\mathbf{U} \cdot \mathbf{n}) * \mathbf{n}|}$$

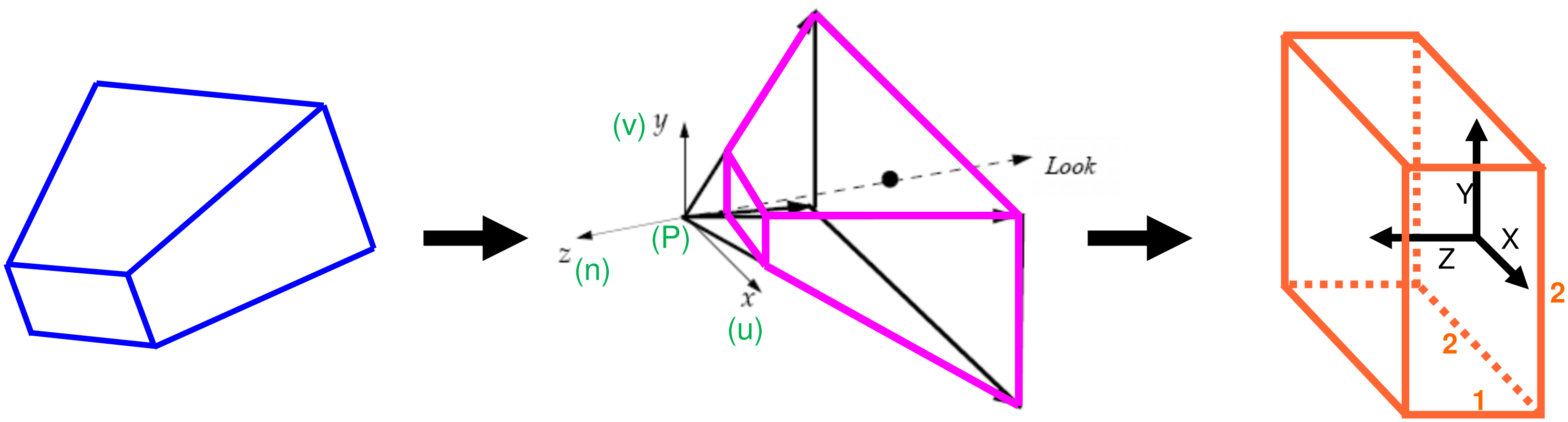
$$\mathbf{u} = \mathbf{v} \times \mathbf{n}$$

  - u,v,n form a right-hand coordinate basis
    - "Camera coordinate system"



Computing Viewing Transformation

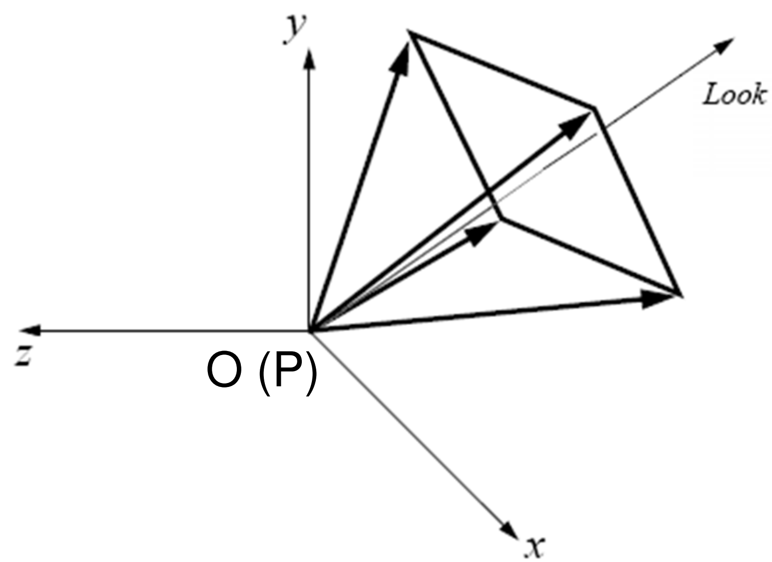
- Two steps
  - **Step 1**: align camera coordinate system P,u,v,n with world coordinate system O,X,Y,Z
  - **Step 2**: scale and stretch the frustum to the cuboid
- As a product of transformation matrices
  - Using homogenous coordinates



Step 1

- First, translate the eye point P to the origin
  - Let P have coordinates (p_x,p_y,p_z)

$$\mathbf{T} = \begin{pmatrix} 1 & 0 & 0 & -p_x \\ 0 & 1 & 0 & -p_y \\ 0 & 0 & 1 & -p_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



Step 1 (cont)

- Then, rotate the three axes u,v,n to X,Y,Z
  - Let's set up the equation to solve for the rotation matrix (R):
    - Note the homogenous coordinates for a vector ends with **0**!

$$\mathbf{R} \cdot \mathbf{u} = \{1, 0, 0, 0\}$$

$$\mathbf{R} \cdot \mathbf{v} = \{0, 1, 0, 0\}$$

$$\mathbf{R} \cdot \mathbf{n} = \{0, 0, 1, 0\}$$

$$\mathbf{R} \cdot \{0, 0, 0, 1\} = \{0, 0, 0, 1\}$$

  - In matrix form:

$$\mathbf{R} \cdot \begin{pmatrix} u_x & v_x & n_x & 0 \\ u_y & v_y & n_y & 0 \\ u_z & v_z & n_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Step 1 (cont)

- This is a matrix inversion problem:

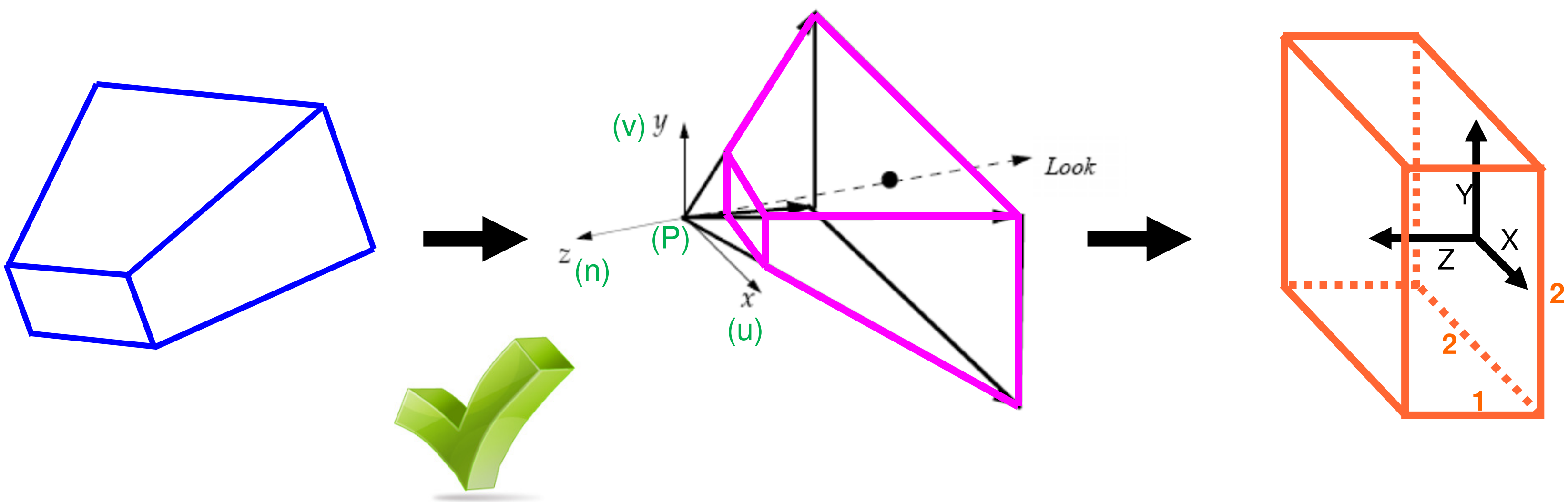
$$\mathbf{R} = \mathbf{M}^{-1} \quad \text{where} \quad \mathbf{M} = \begin{pmatrix} u_x & v_x & n_x & 0 \\ u_y & v_y & n_y & 0 \\ u_z & v_z & n_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Since M is an orthonormal matrix:

$$\mathbf{R} = \mathbf{M}^T$$

Step 1 - Done

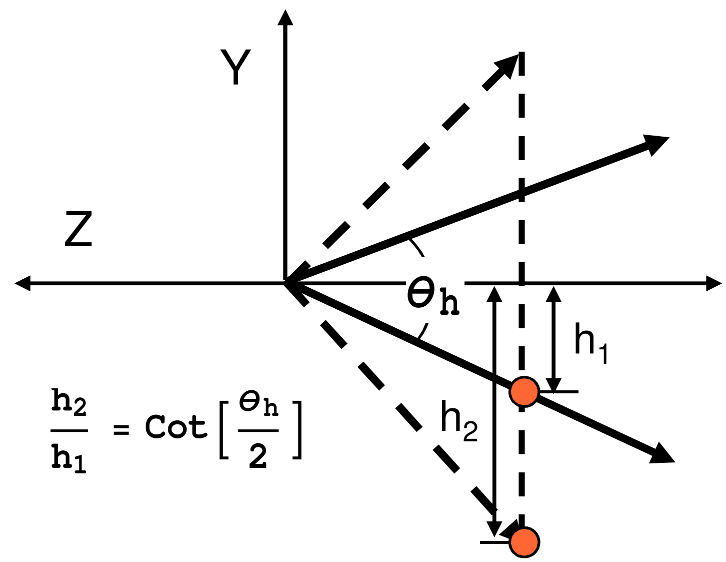
- Eye point at origin, looking down negative z axis



Step 2

- Some preparations
  - First, make width/height angles to be π/2
    - Non-uniform scaling in X,Y coordinates

$$S_{xy} = \begin{pmatrix} \text{Cot}\left[\frac{\theta_w}{2}\right] & 0 & 0 & 0 \\ 0 & \text{Cot}\left[\frac{\theta_h}{2}\right] & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

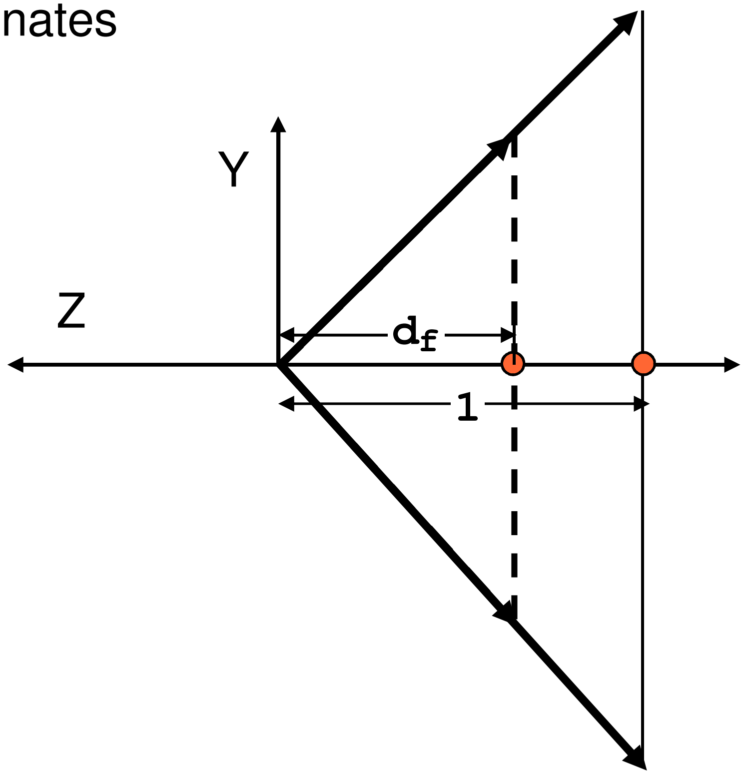


A look down the X axis

Step 2 (cont)

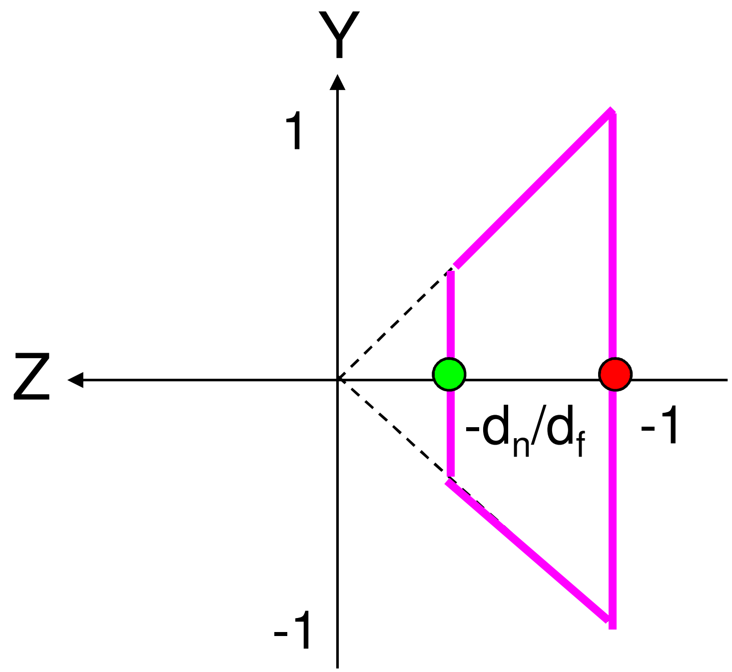
- Some preparations
  - Next, push the far plane from Z=-d_f to Z=-1
    - Uniform scaling in all three coordinates

$$S_{xyz} = \begin{pmatrix} \frac{1}{d_f} & 0 & 0 & 0 \\ 0 & \frac{1}{d_f} & 0 & 0 \\ 0 & 0 & \frac{1}{d_f} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



Step 2 (cont)

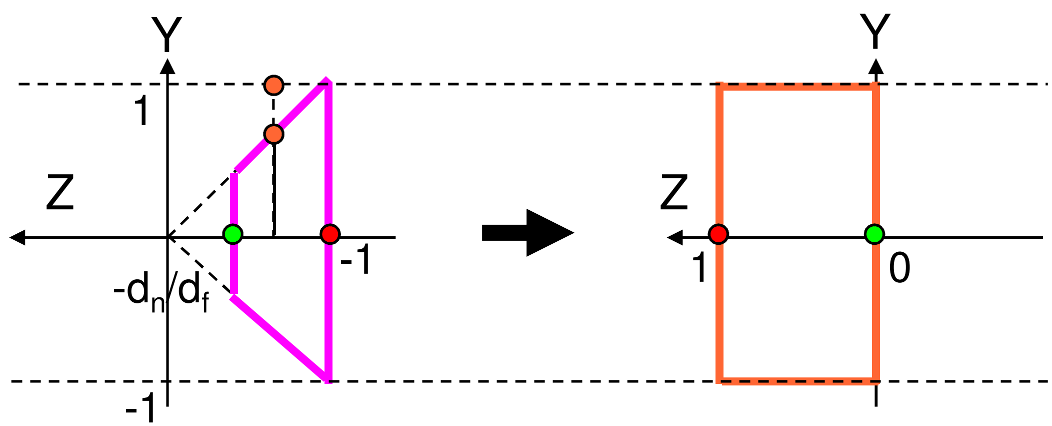
- Where we are now:



A look down the X axis (same picture when looking down Y)

Step 2 (cont)

- Perspective transformation
  - Stretching and flipping the truncated pyramid to the cuboid
    - Change Z range: ($[-d_n/d_f,-1]$ -> $[0,1]$)
    - Stretch in XY plane: Non-uniform stretching based on z



A look down the X axis

Step 2 (cont)

- Perspective transformation

$$D = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{k-1} & \frac{k}{k-1} \\ 0 & 0 & -1 & 0 \end{pmatrix}, \qquad \text{where } k = \frac{d_n}{d_f}$$

  - Applying D to homogeneous coordinates:

$$D \cdot \{x, y, z, 1\} = \left\{x, y, \frac{k}{-1+k} + \frac{z}{-1+k}, -z\right\}$$

  - Converting from homogeneous coordinates {x,y,z,w} to Cartesian coordinates (divide x,y,z by w)

$$\left\{\frac{-x}{z}, \frac{-y}{z}, \frac{k+z}{z\,(1-k)}\right\}$$

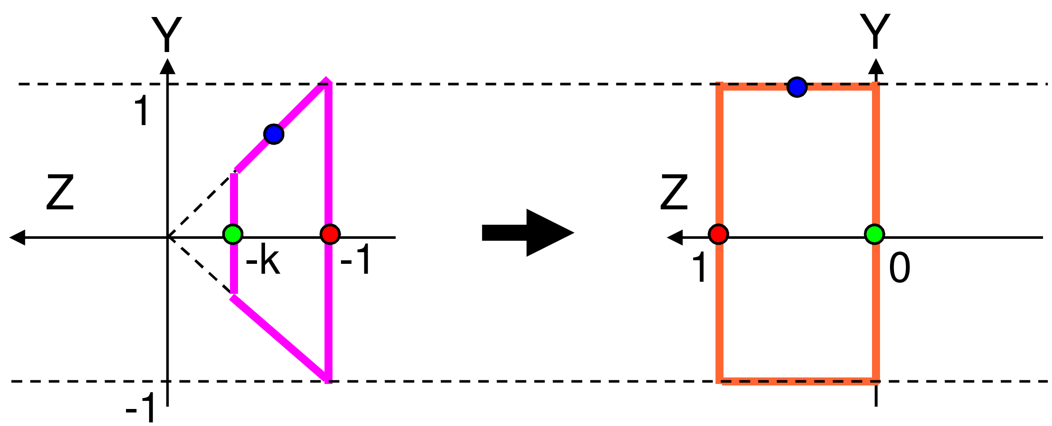
Step 2 (cont)

- Perspective transformation

$$\left\{ \frac{-x}{z}, \frac{-y}{z}, \frac{k+z}{z(1-k)} \right\}$$

Stretching in XY plane

Change Z range



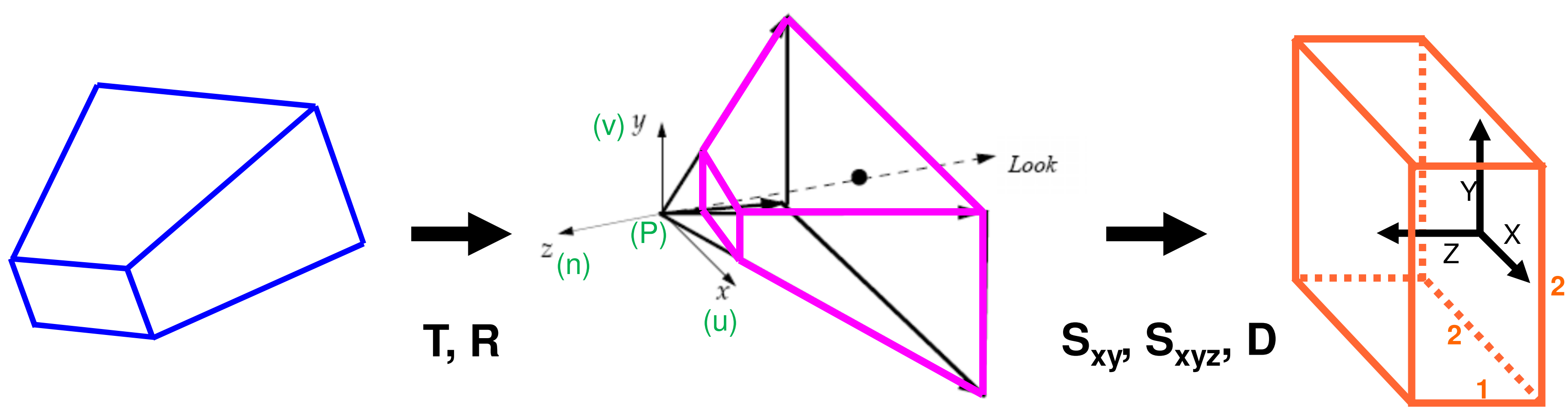
A look down the X axis

Putting Together

- Translation: **T**
- Rotation: **R**
- Scaling: $\mathbf{S_{xy}}$, $\mathbf{S_{xyz}}$

World-to-camera transformation

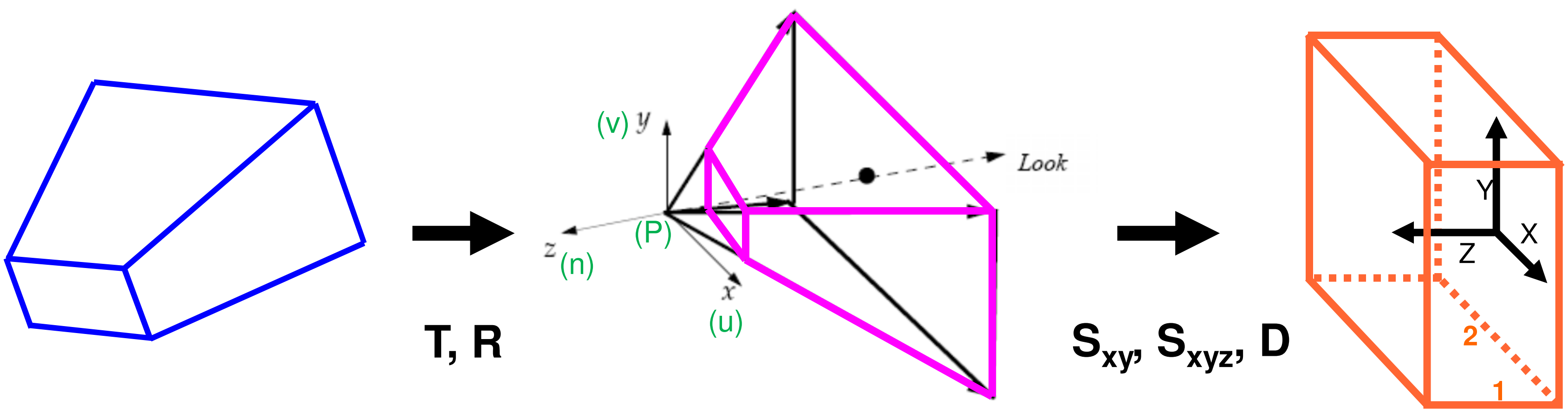
- Perspective transformation: **D**



Putting Together

- Complete viewing transformation to bring a point **q** to the canonical volume:

$$q' = D\, S_{xyz}\, S_{xy}\, R\, T\, q$$

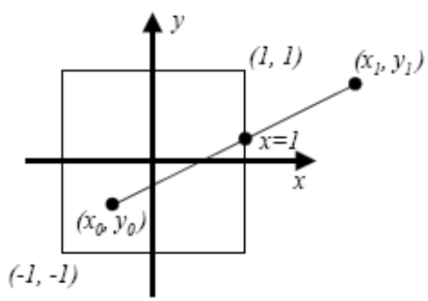


Clipping

- After transformation into the canonical volume, each object will be clipped against 6 cuboid faces.
  - Point clipping: checking coordinates range

$$-1 \le x \le 1,\ -1 \le y \le 1,\ 0 \le z \le 1$$

  - Edge clipping: computing line/plane intersections
    - We will discuss a 2D version in **next lecture**.



$$1 = (1-t)x_0 + tx_1$$
$$1 - x_0 = -tx_0 + tx_1$$
$$1 - x_0 = t(x_1 - x_0)$$
$$t = \frac{1 - x_0}{x_1 - x_0}$$

  - Triangle clipping: can be done by line/plane intersections

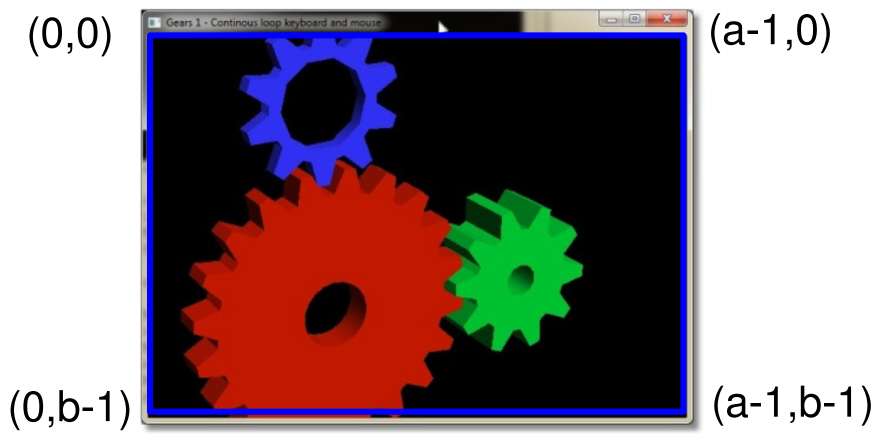
Projecting

- Dropping z coordinate
  - Resulting points have range:

    $-1 \leq x \leq 1, -1 \leq y \leq 1$

Viewport Transform

- Get viewport (pixel) coordinates
  - Viewport coordinate {0,0} is at top-left corner



  - If the viewport is **a** pixels wide and **b** pixels high, what is the pixel coordinates for a projected point {x,y}?

$$\left\{\frac{(a-1)\ (x+1)}{2},\ \frac{(b-1)\ (1-y)}{2}\right\}$$