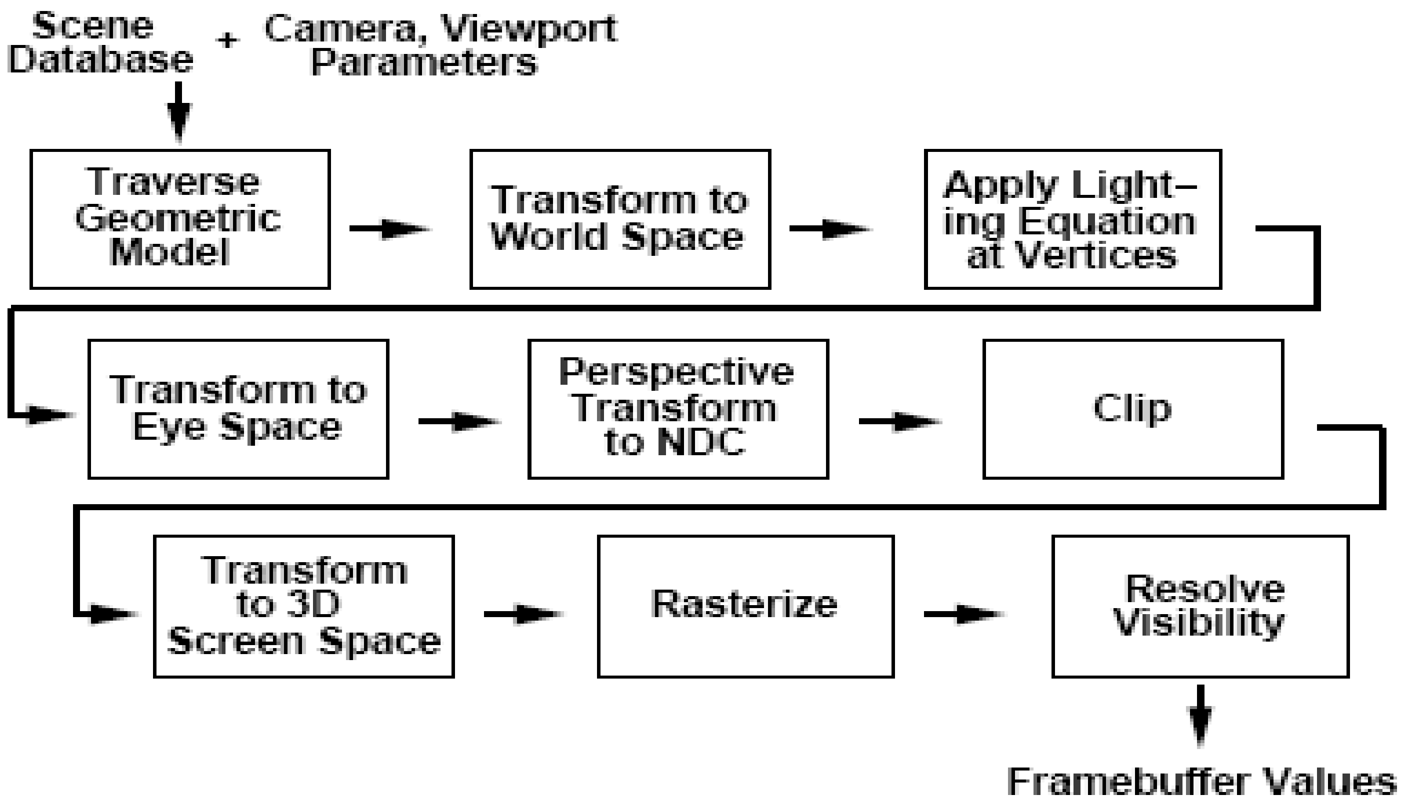
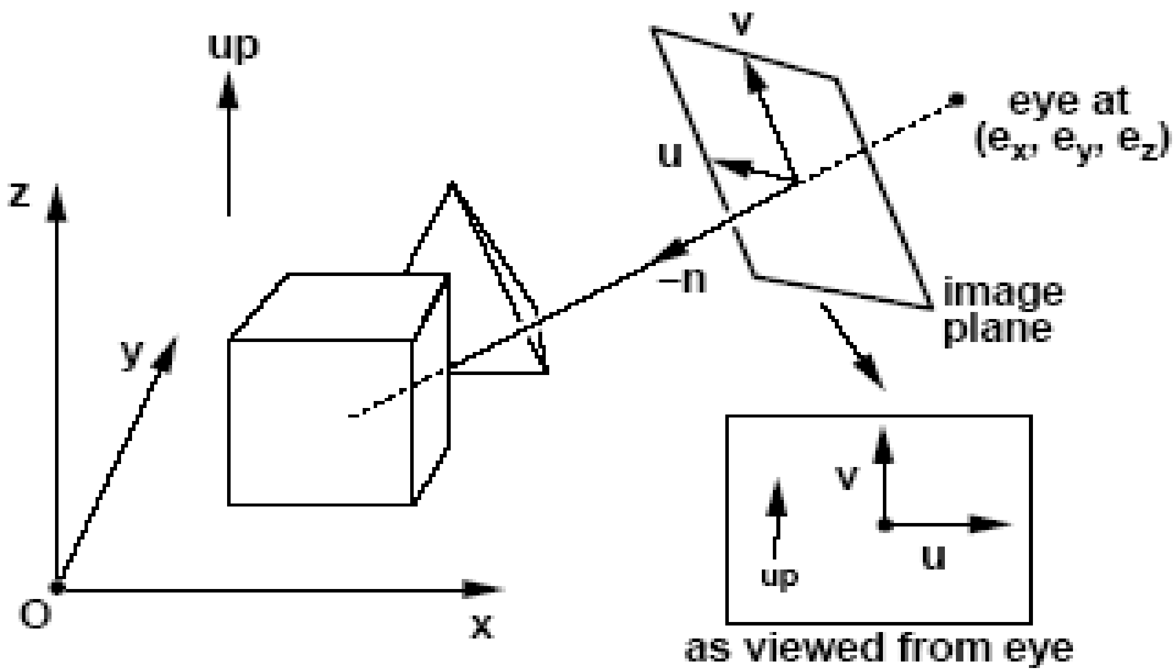
View Transformation

View Transformation



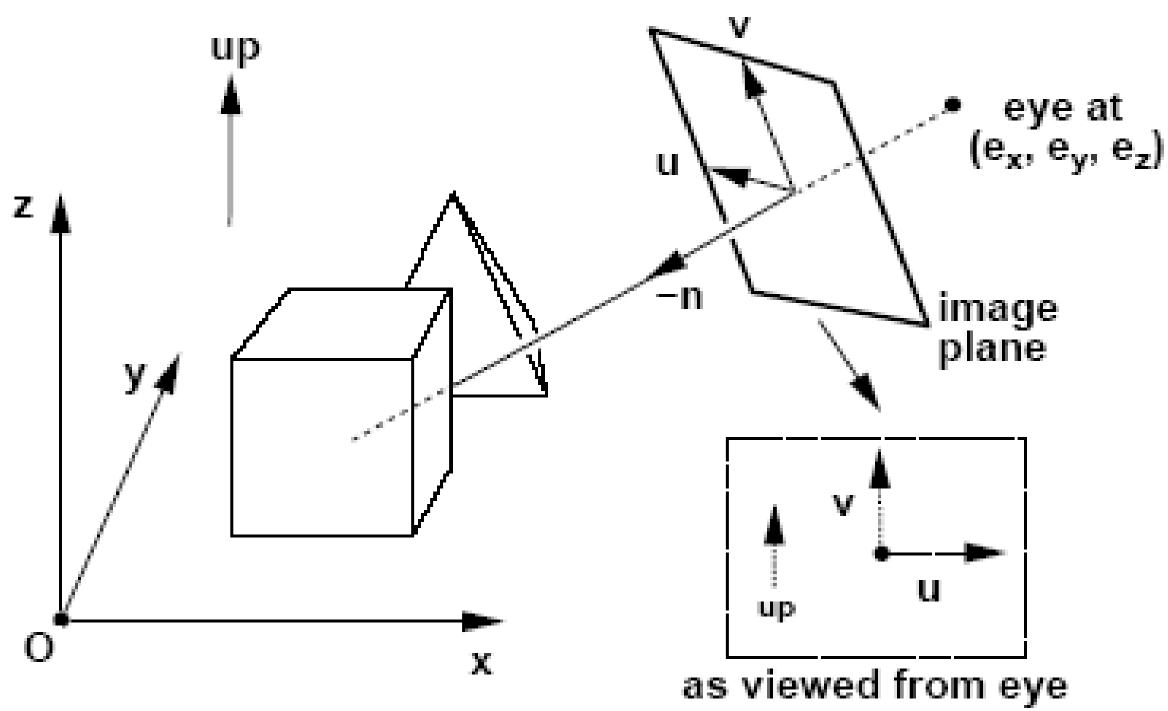
Transform (i.e., express) geometry into coordinates that are well-suited to (simple) clipping and projection hardware

Positioning Synthetic Camera



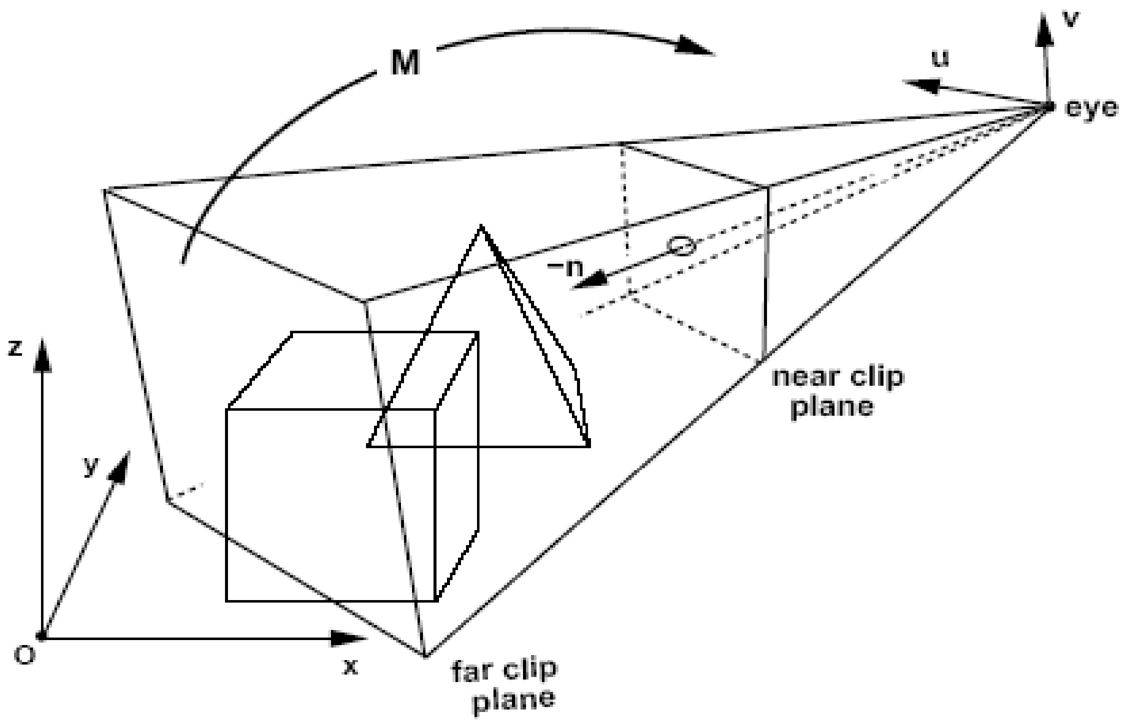
What are our "degrees of freedom" in camera positioning?
To achieve effective visual simulation, we want:
1) the eye point to be in proximity of modeled scene
2) the view to be directed toward region of interest, and
3) the image plane to have a reasonable "twist"

Eye Coordinates



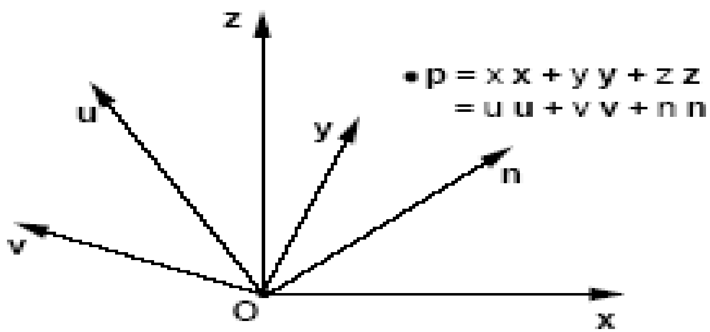
Eyepoint at origin
u axis toward “right” of image plane
v axis toward “top” of image plane
view direction along *negative* **n** axis

Transformation to Eye Coordinates



Our task: construct the transformation **M** that re-expresses world coordinates in the viewer frame

Machinery: Changing Orthobases

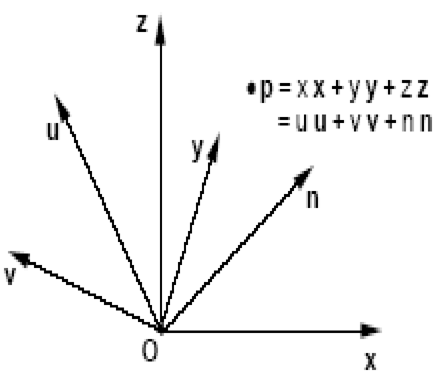


Suppose you are given an orthobasis **u**, **v**, **n**
What is the action of the matrix **M** with rows **u**, **v**, and **n** as below?

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

Applying M to u, v, n

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



Two equally valid interpretations, depending on reference frame:

1: Think of **uvn** basis as a rigid object in a *fixed* world space

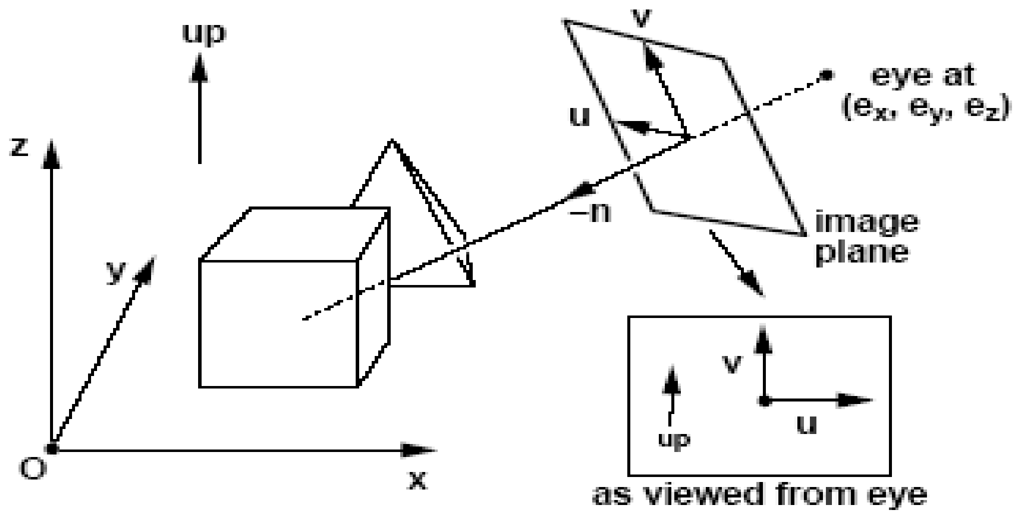
Then **M** "rotates" **uvn** basis into **xyz** basis

2: Think of a *fixed* axis triad, with "labels" from **xyz** space

Then **M** "reexpresses" an **xyz** point **p** in **uvn** coords!

It is this second interpretation that we use today to "relabel" world-space geometry with eye space coordinates

Positioning Synthetic Camera



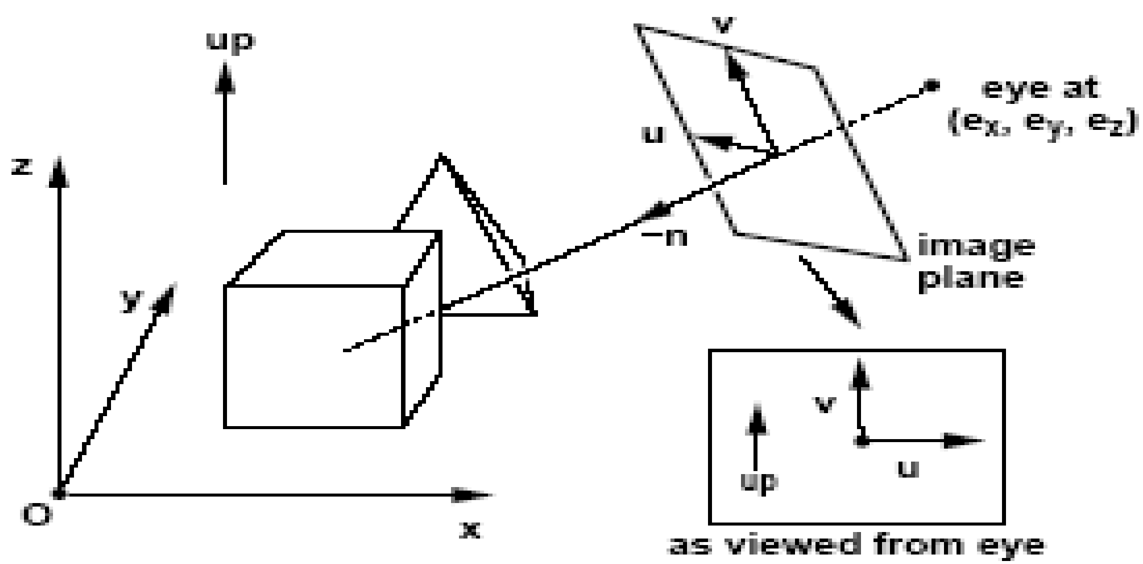
Given eyepoint **e**, basis **^u**, **^v**, **^n**
Deduce **M** that expresses world in eye coordinates:
Overlay origins, then change bases:

$$\mathbf{T} = \begin{pmatrix} 1 & 0 & 0 & -e_x \\ 0 & 1 & 0 & -e_y \\ 0 & 0 & 1 & -e_z \\ 0 & 0 & 0 & 1 \end{pmatrix}; \qquad \mathbf{R} = \begin{pmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\mathbf{M} = \mathbf{RT}$$

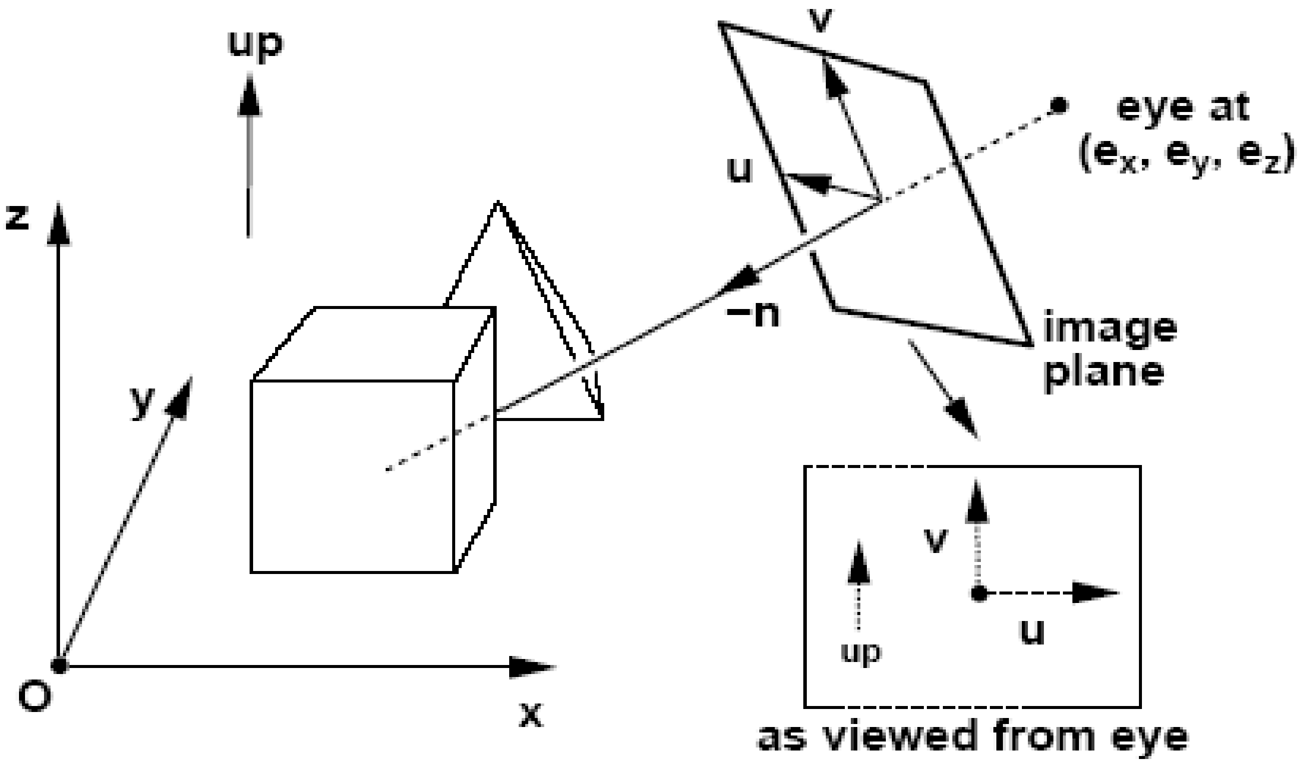
Positioning Synthetic Camera

$$\mathbf{M}\begin{pmatrix} e_x \\ e_y \\ e_z \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}; \qquad \mathbf{M}\begin{pmatrix} e_x - \hat{n}_x \\ e_y - \hat{n}_y \\ e_z - \hat{n}_z \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ -1 \\ 1 \end{pmatrix};$$



Check: does **M** re-express world geometry in eye coordinates?

Positioning Synthetic Camera

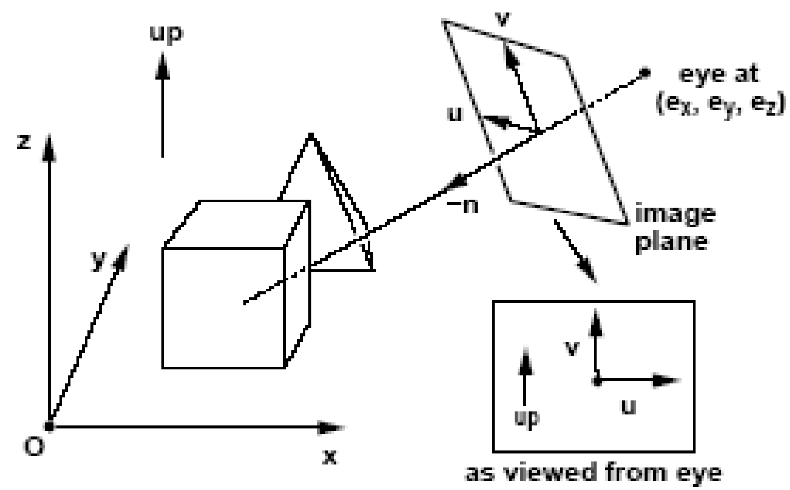


Camera specification must include:
World-space eye position **e**
World-space "lookat direction" **-n**

Are **e** and **-n** enough to determine the camera DOFs (degrees of freedom)?

Positioning Synthetic Camera

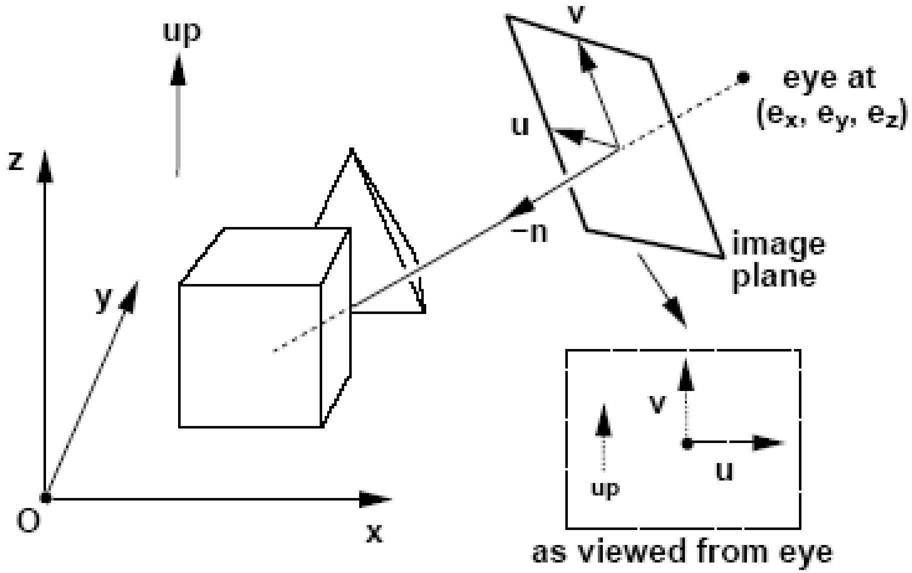
Are **e** and **-n** enough to determine the camera DOFs?
No. Note that we were *not* given **u** and **v**!
(Why not simply require the user to specify them?)



We must also determine **u** and **v**, i.e., camera "twist" about **n**.
Typically done by specification of a world-space "**up** vector" provided by user interface, e.g., using gluLookat(e, c, up)
"Twist" constraint: Align **v** with world **up** vector (How?)

Positioning Synthetic Camera

Trick: *construct* **u** and **v** from available information!
"Twist" constraint: Align **v** with world **up** vector

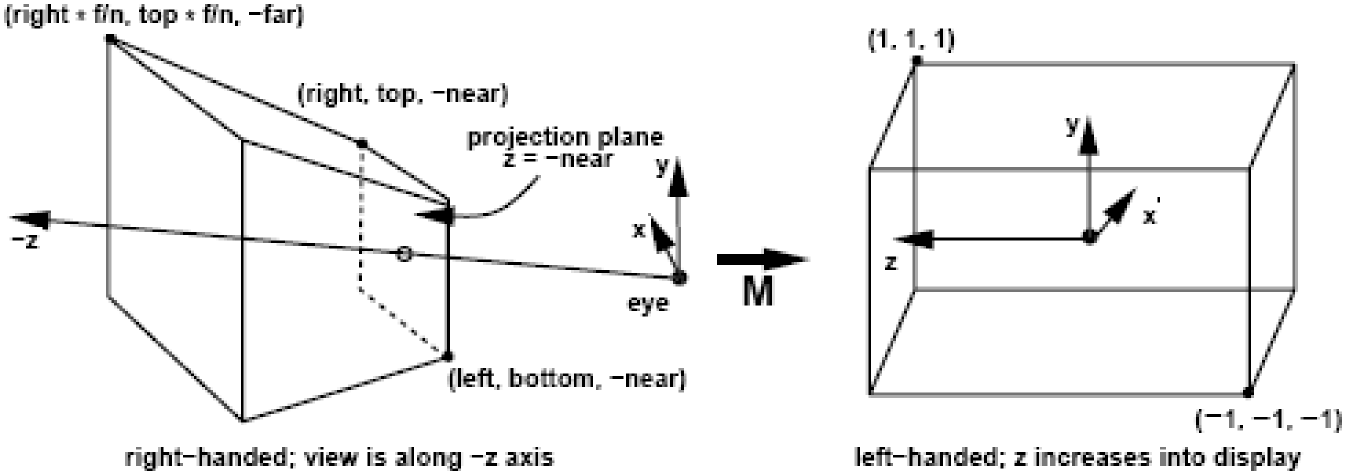


Given: eyepoint **e**, view direction **n**, and world **up** vector:

1. Compute $\mathbf{u} = -\mathbf{n} \times \mathbf{up}$
2. Compute $\mathbf{v} = \mathbf{u} \times -\mathbf{n}$
3. Construct M as above from **u**, **v**, **n**, and **e**

Where are we?

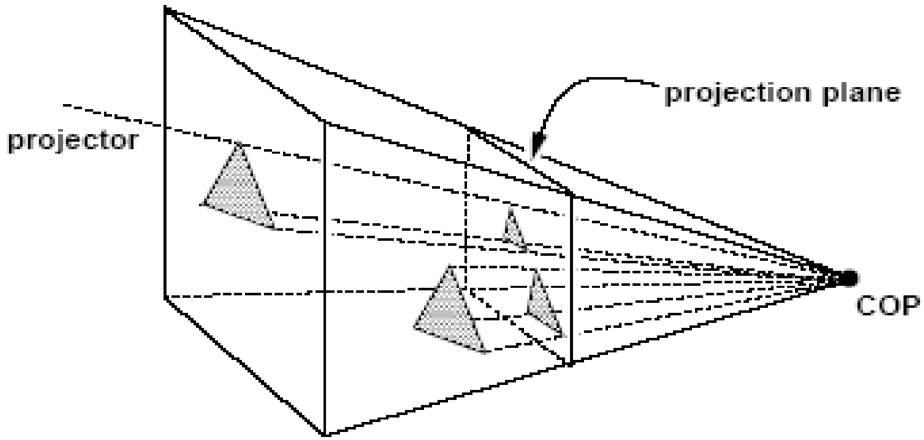
We've re-expressed world geometry in eye's frame of reference:



Next we must transform to NDC (Normalized Device Coordinates)
to prepare for (simple) clipping and projection

For that, we need the *Perspective Transformation*
We'll study *Perspective Projection* first, then generalize

What is Projection?

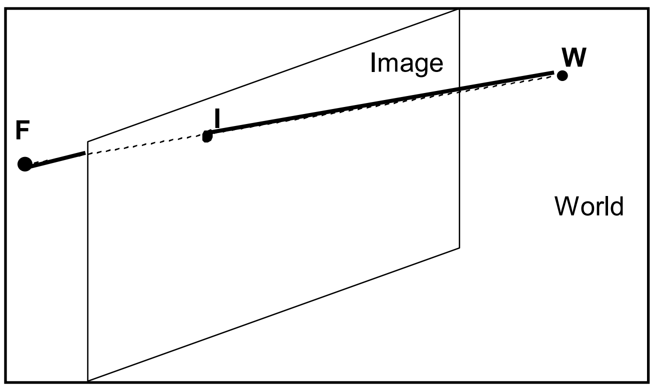


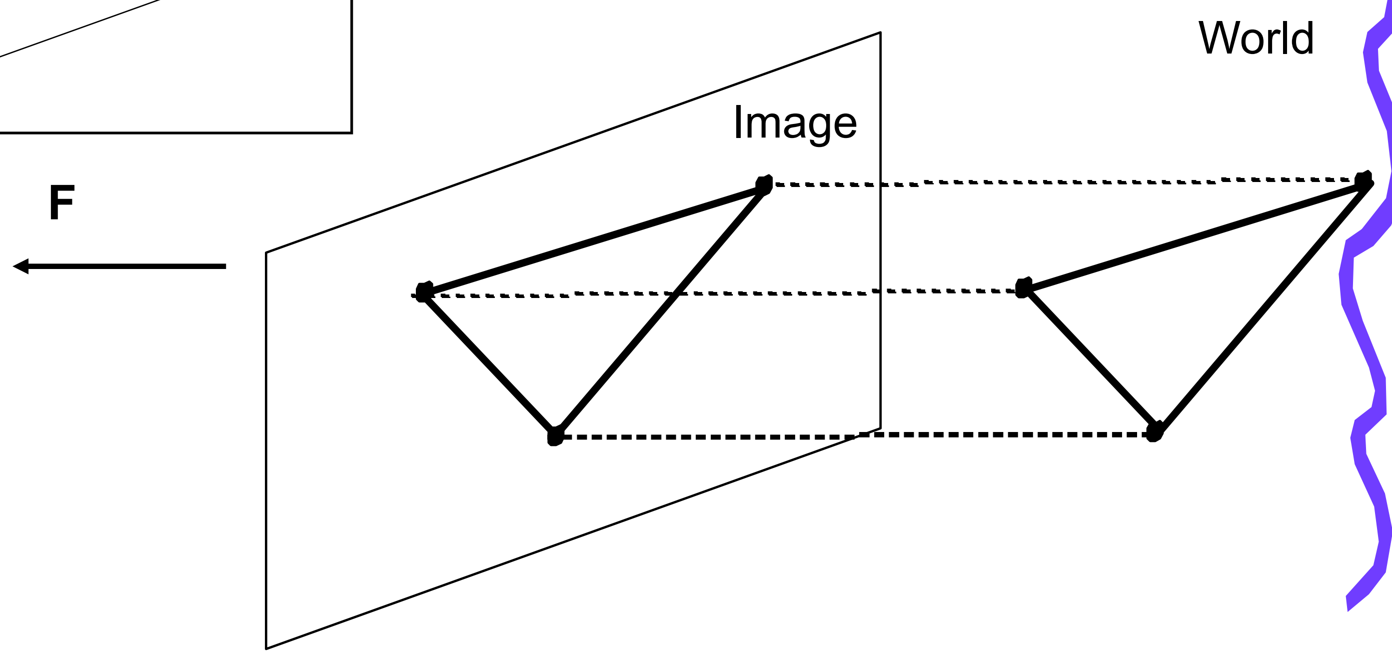
Any operation that reduces dimension (e.g., 3D to 2D)

Orthographic Projection
Perspective Projection

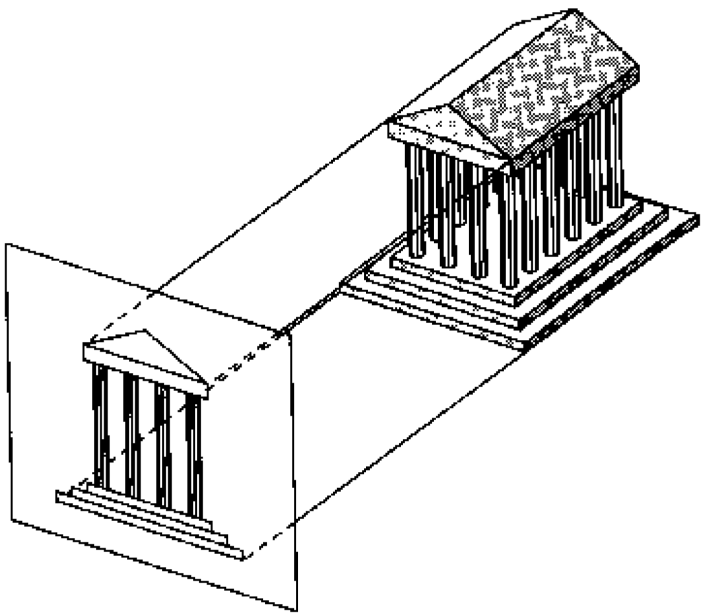
Orthographic Projection

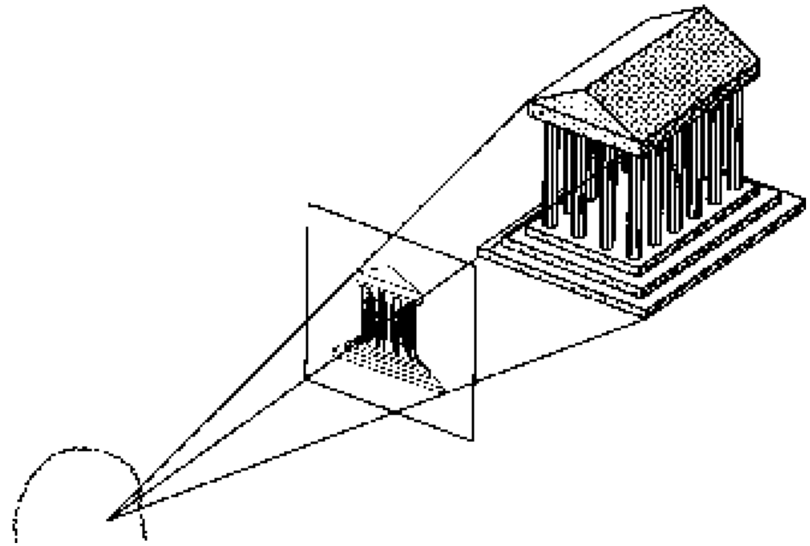
- focal point at infinity
- rays are parallel and orthogonal to the image plane





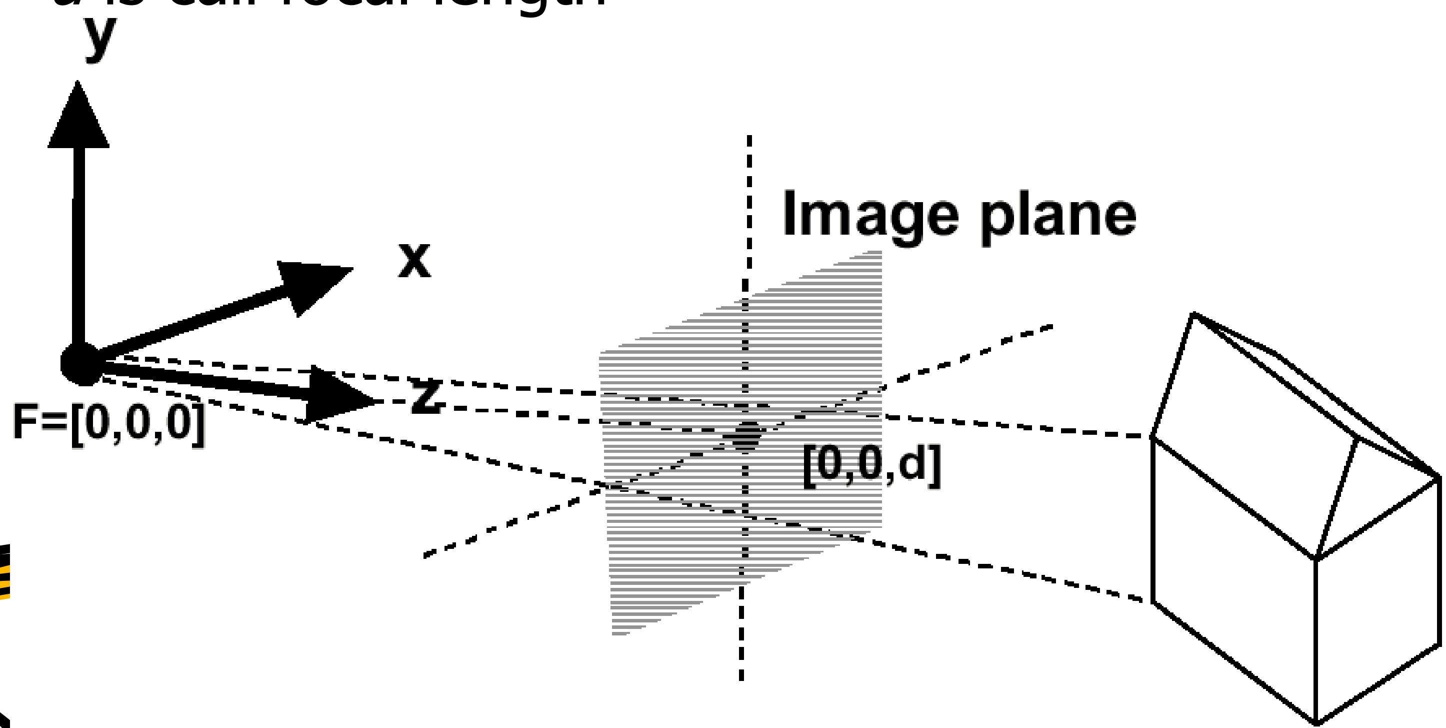
Comparison



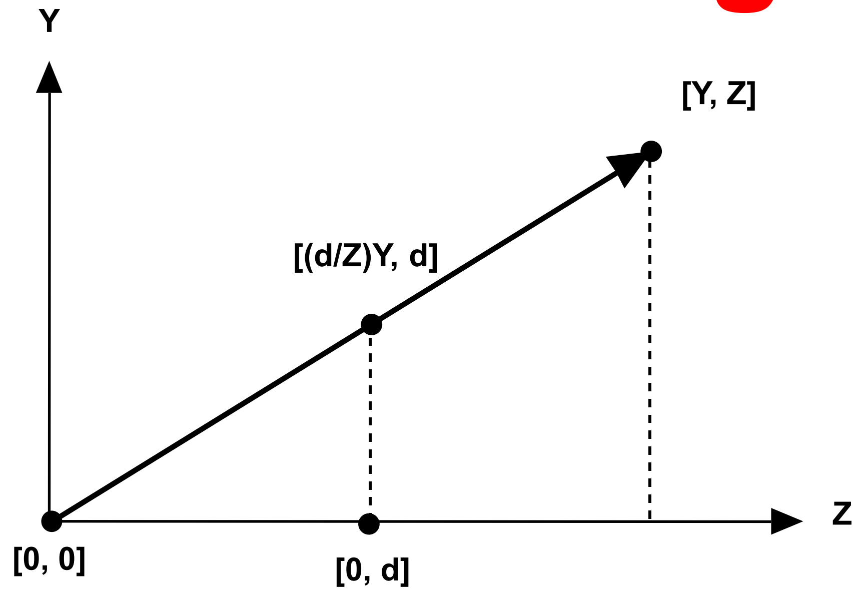


Simple Perspective Camera

- camera looks along *z*-axis
- focal point is the origin
- image plane is parallel to *xy*-plane at distance *d*
- *d* is call focal length



Similar Triangles



- Similar situation with *x*-coordinate
- Similar Triangles:

  point [x,y,z] projects to [(d/z)x, (d/z)y, d]

Projection Matrix

Projection using homogeneous coordinates:

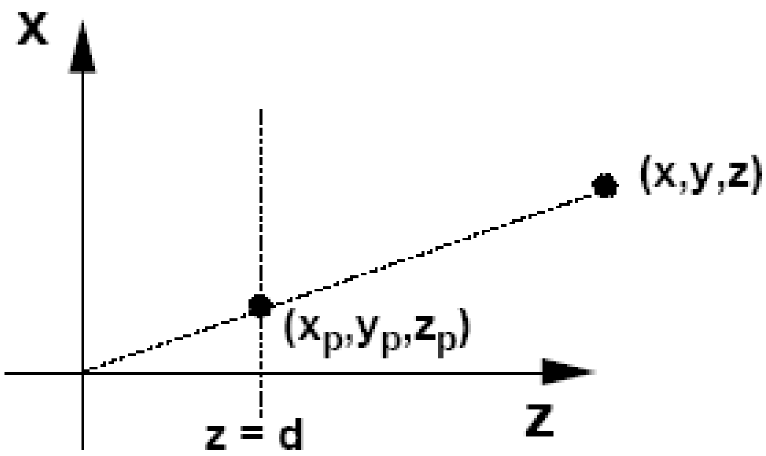
– transform [x, y, z] to [(d/z)x, (d/z)y, d]

$$\begin{bmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & d & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} dx & dy & dz & z \end{bmatrix} \Rightarrow \begin{bmatrix} \frac{d}{z}x & \frac{d}{z}y & \mathrm{d} \end{bmatrix}$$

Divide by 4th coordinate (the "w" coordinate)

- 2-D image point:
  - discard third coordinate
  - apply viewport transformation to obtain physical pixel coordinates

Perspective Projection



What are coordinates of projected point x_p, y_p, z_p?

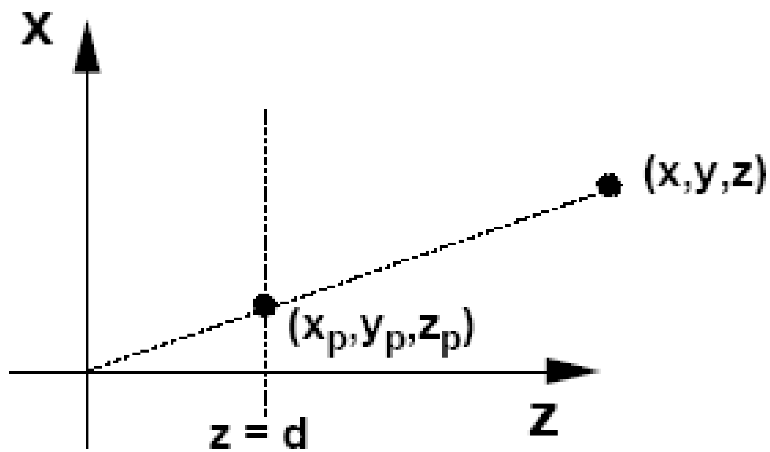
By similar triangles,

$$\frac{x_p}{d} = \frac{x}{z} \qquad \frac{y_p}{d} = \frac{y}{z}$$

Multiplying through by d yields

$$x_p = \frac{d \cdot x}{z} = \frac{x}{z/d} \qquad y_p = \frac{d \cdot y}{z} = \frac{y}{z/d} \qquad z_p = d$$

Perspective Projection



What are coordinates of projected point x_p, y_p, z_p?

By similar triangles,

$$\frac{x_p}{d} = \frac{x}{z} \qquad \frac{y_p}{d} = \frac{y}{z}$$

Multiplying through by d yields

$$x_p = \frac{d \cdot x}{z} = \frac{x}{z/d} \qquad y_p = \frac{d \cdot y}{z} = \frac{y}{z/d} \qquad z_p = d$$

$z = 0$ not allowed (what happens to points on plane $z = 0$?)

Operation well-defined for all other points

Perspective Projection

Matrix formulation using "homogeneous 4-vectors":

$$\mathrm{M} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{pmatrix}$$

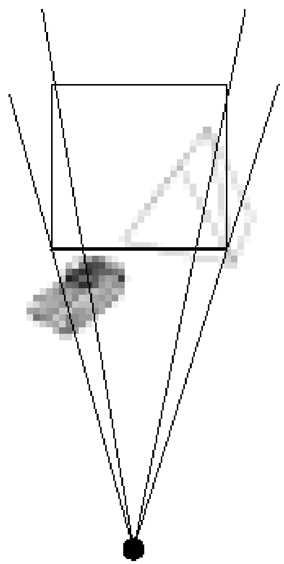
$$\begin{pmatrix} X \\ Y \\ Z \\ W \end{pmatrix} = \mathrm{M} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ z/d \end{pmatrix}$$

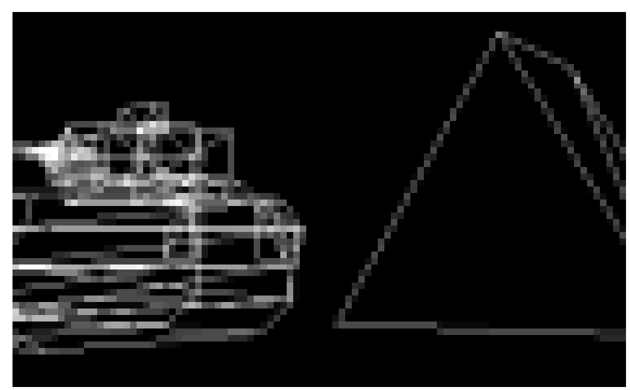
Finally, recover projected point using *homogenous convention*:
Divide by 4*th* element to convert 4-vector to 3-vector:

$$\begin{pmatrix} X/W \\ Y/W \\ Z/W \end{pmatrix} = \begin{pmatrix} x_p \\ y_p \\ z_p \end{pmatrix} = \begin{pmatrix} \frac{x}{z/d} \\ \frac{y}{z/d} \\ d \end{pmatrix}$$

Are we ready to rasterize? Not yet.

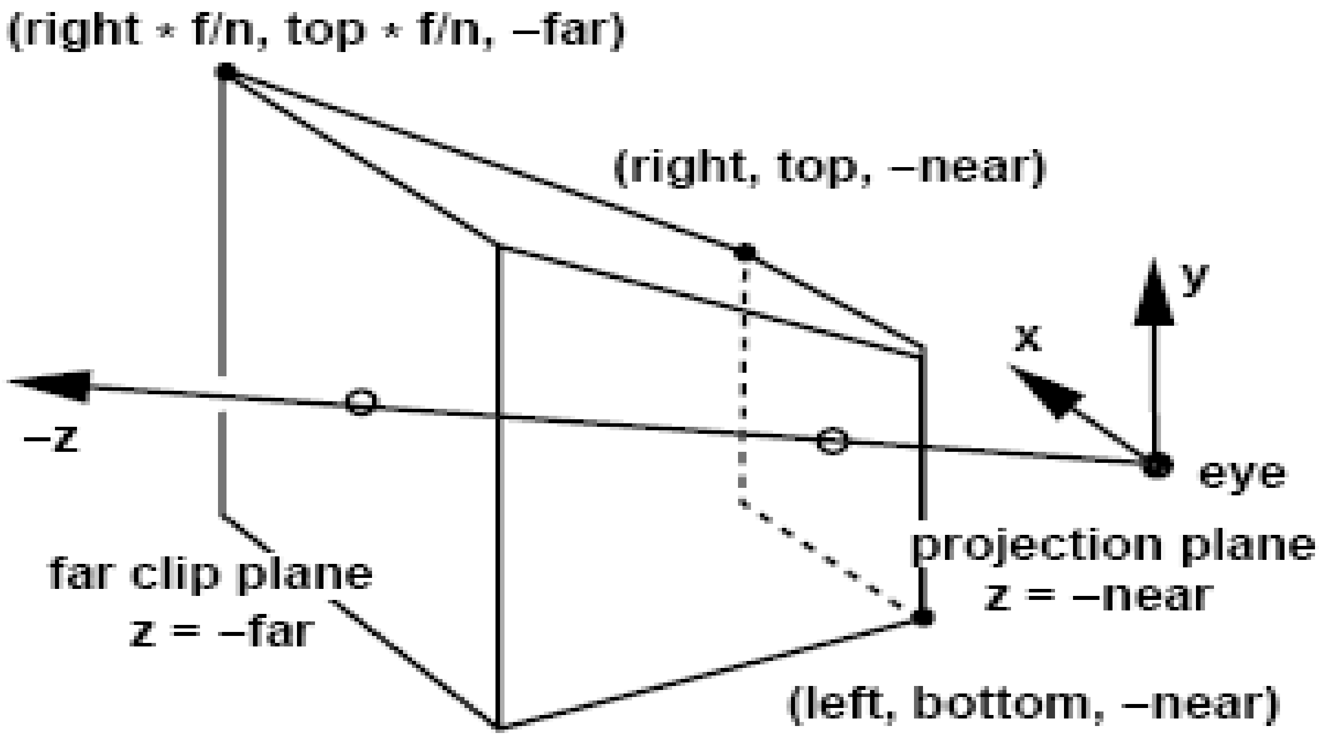
- It is difficult to do clipping directly in the viewing frustum





The View Frustum

defined by 6 parameters: left, right, bottom, top, near, far

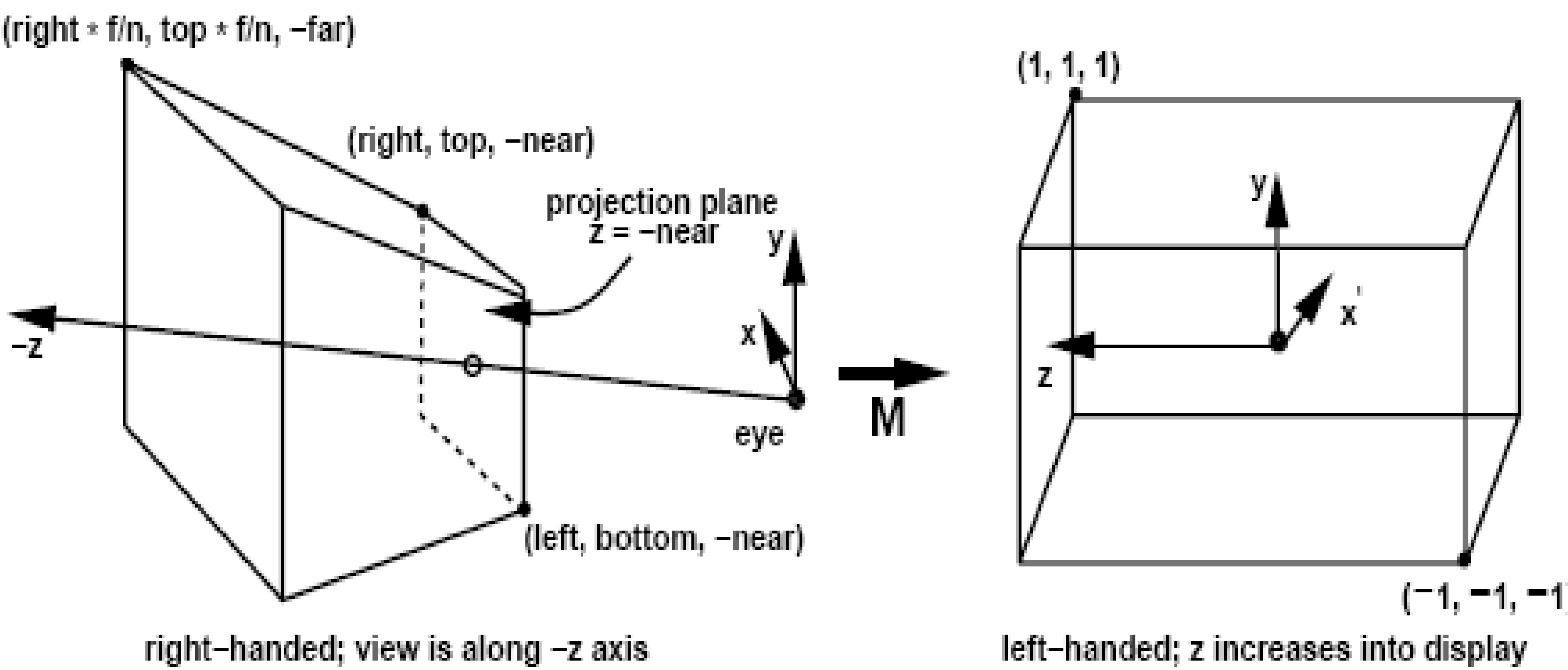


right−handed; view is along −z axis

Canonical View Volume

Right parallelepiped bounded by $x = \pm 1$, $y = \pm 1$, $z = \pm 1$

Called NDC, or sometimes Clip Coordinates



Where is the image plane in NDC?

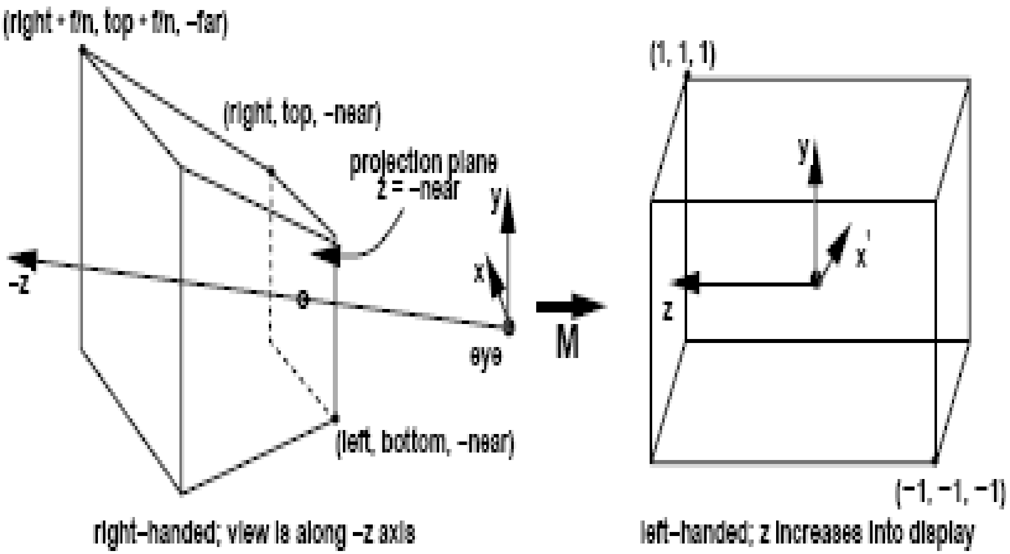
Our goal: construct a *perspective transformation* M that transforms view frustum into the canonical view volume, while preserving depth order

Matrix Formulation

(This is the OpenGL form; several variations exist)
Check action of $\mathbf{M}$:

$$\mathbf{M}\begin{pmatrix} l \\ b \\ -n \\ 1 \end{pmatrix} = \begin{pmatrix} -1 \\ -1 \\ -1 \\ 1 \end{pmatrix}; \qquad \mathbf{M}\begin{pmatrix} r\frac{f}{n} \\ t\frac{f}{n} \\ -f \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}; \qquad \mathbf{M}\begin{pmatrix} (l+r)/2 \\ (b+t)/2 \\ -n \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ -1 \\ 1 \end{pmatrix}$$

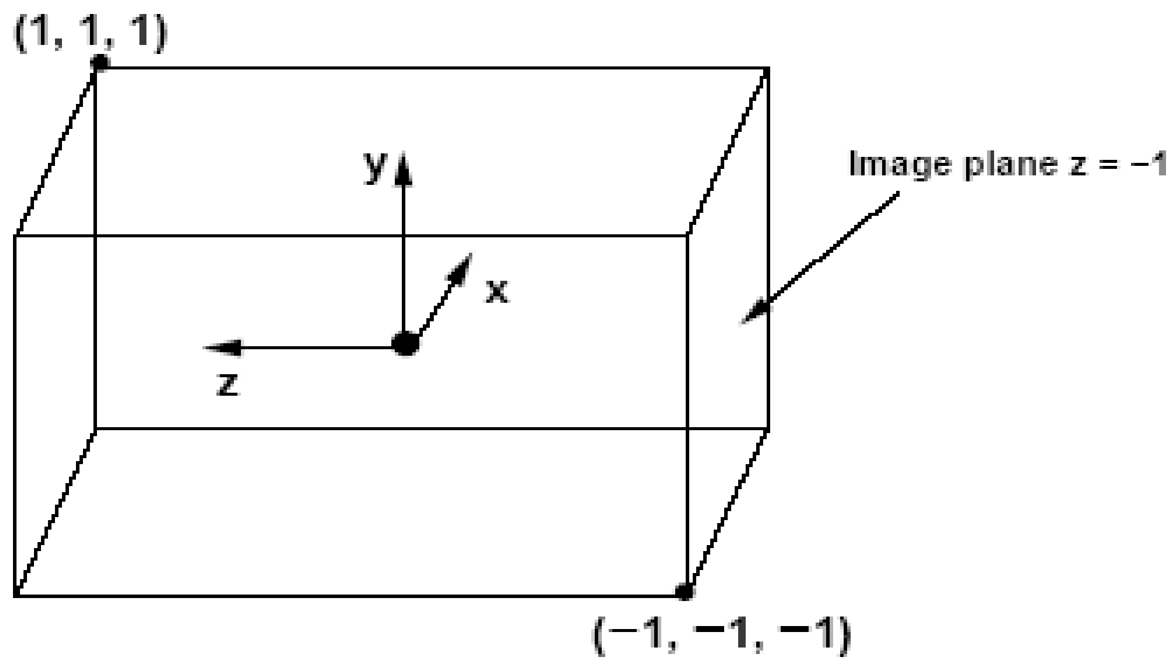
Often we set $l = -r$ and $b = -t$ (why?), so that:



$$\mathbf{M} = \begin{pmatrix} \frac{n}{r} & 0 & 0 & 0 \\ 0 & \frac{n}{t} & 0 & 0 \\ 0 & 0 & -(\frac{f+n}{f-n}) & -(\frac{2fn}{f-n}) \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Perspective Projection

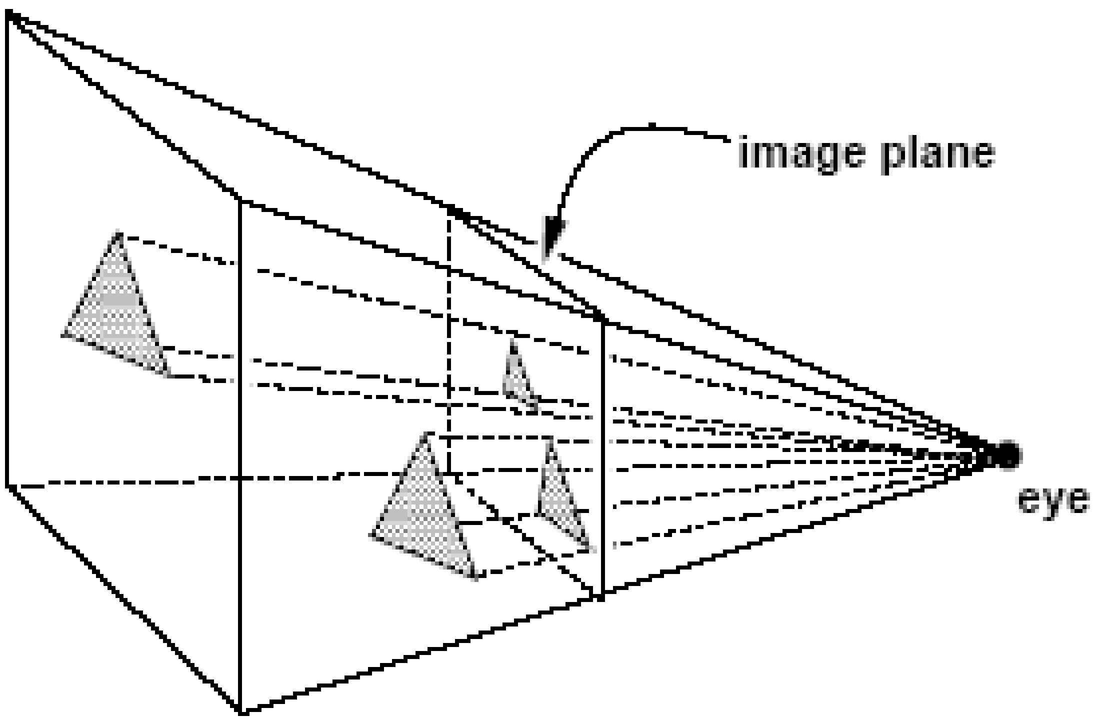
Suppose we have transformed from World to Eye to Canonical coordinates
How do we project onto “image plane”?



Normalized Device Coordinates

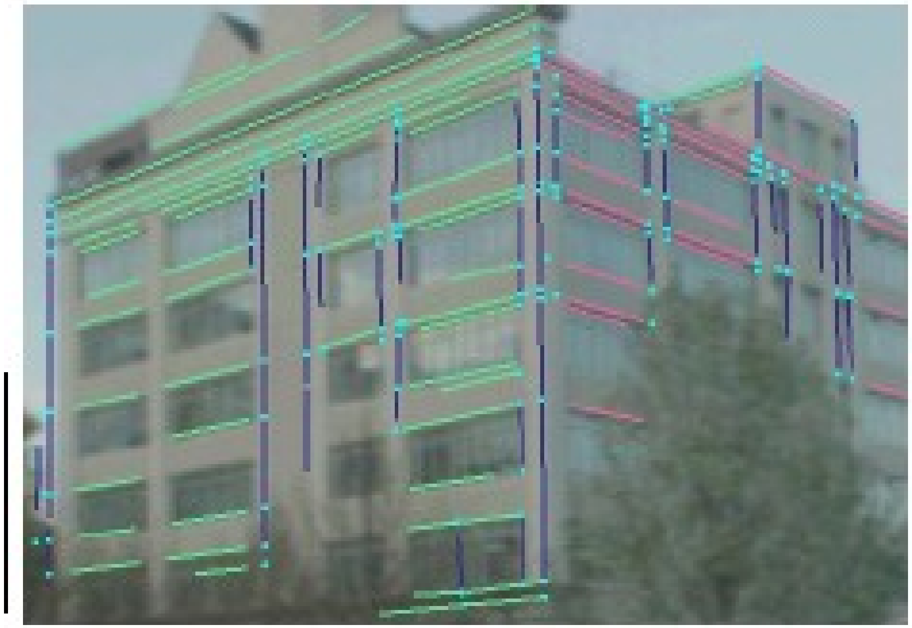
Simply ignore *z* coordinate!

Qualitative Features of Perspective Projection



Equal-sized objects at different depths project to different sizes!

Perspective projection does *not* preserve shape of planar figures!



Families of parallel lines have “vanishing points” projection of point at infinity in direction of lines