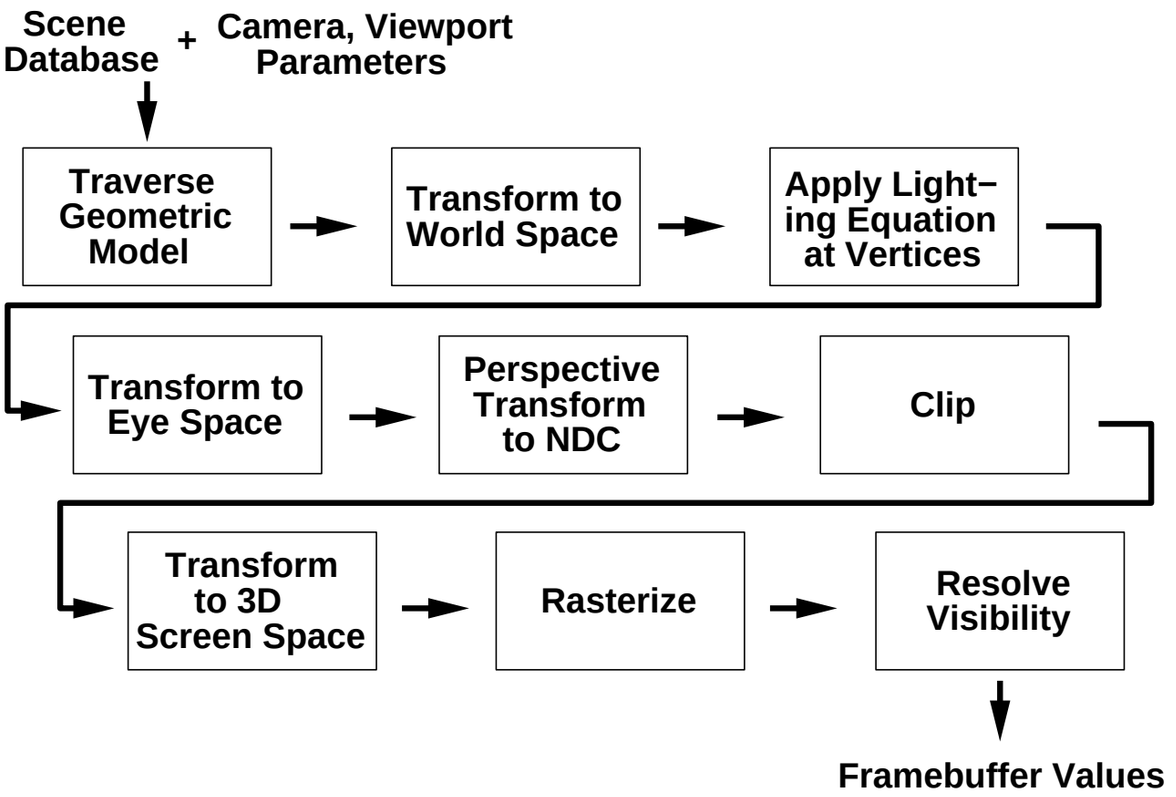
Lecture 5: Thursday, 19 September 2002

Administrative notes:

- Ass't 1 (`uid_object, *.iv`) due tomorrow 5pm
  - See list of FAQ's posted to course page
  - Object exhibition next week in class !
- Ass't 2 (Scene Modeling & Lighting) now posted
  - Due next Friday at 5pm
- Review sessions scheduled
  - See course page for times & locations
- Teaching staff office hours (corrected):
  - Addy: R4-6 in W20-575 (Linux cluster)
  - Jingyi: F1-5 in LCS NE43-246 (Graphics Lab)
  - Durand: R4-5 in 4-035 (SGI cluster)
    R5-6 in LCS NE43-254
  - Teller: W 3-4 in LCS NE43-252
    R 4-5 in 4-035 (SGI cluster)
- Note office hours immediately after class

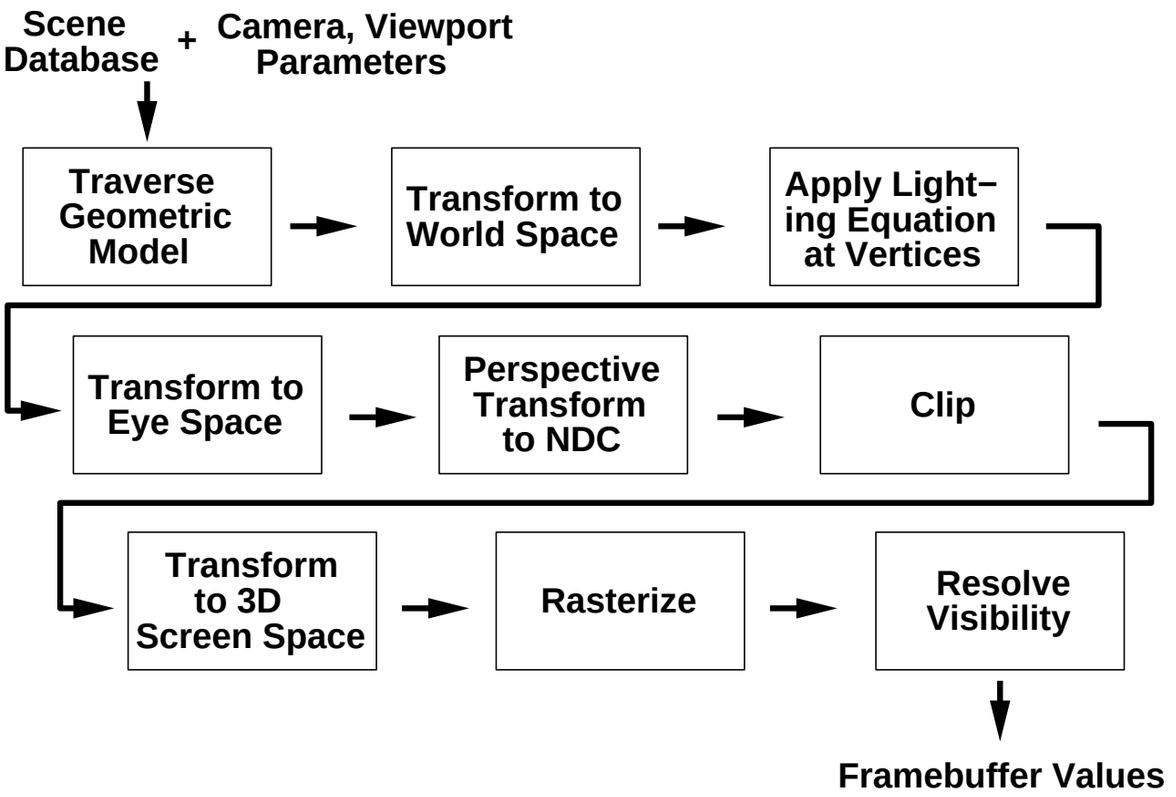
Lecture Plan



Last week and Tuesday:

- Pipeline Overview
- Object-to-world space transform
- Local Illumination model

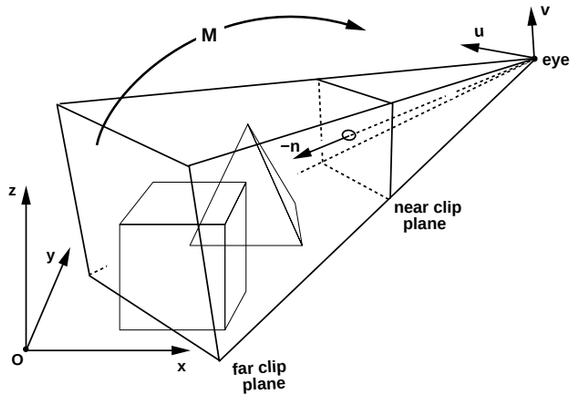
Lecture Plan



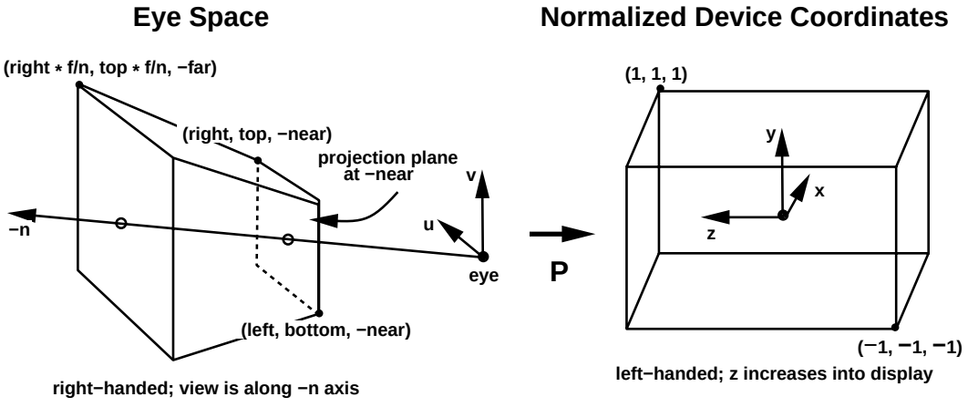
Today:

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

Coordinate transformation to Eye Space



Then, Eye Space to Normalized Device Coordinates



(NDC also called “clip space”)

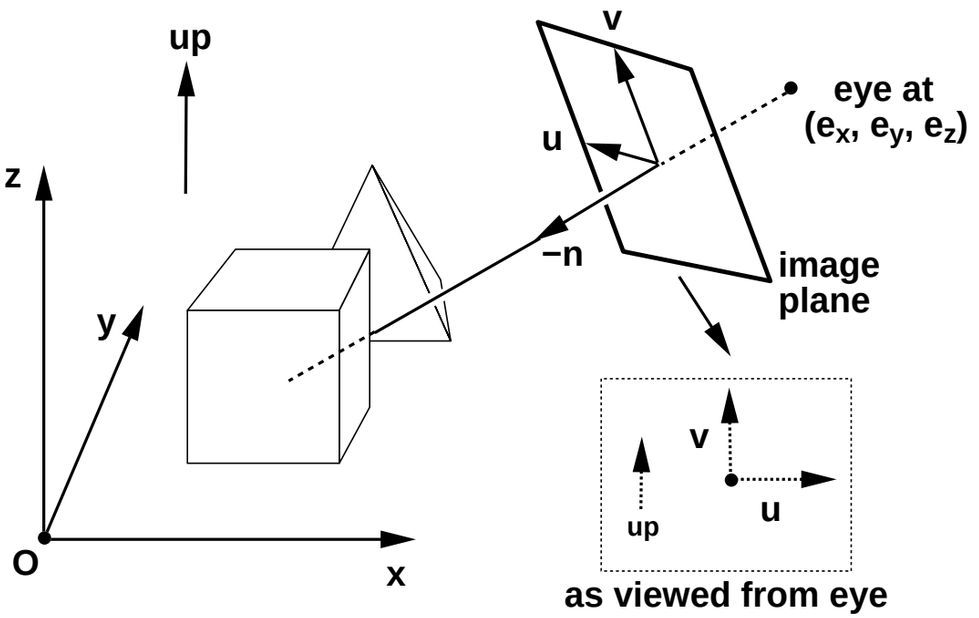
Why do this?

Makes math of clipping, projection particularly simple

Positioning Synthetic Camera

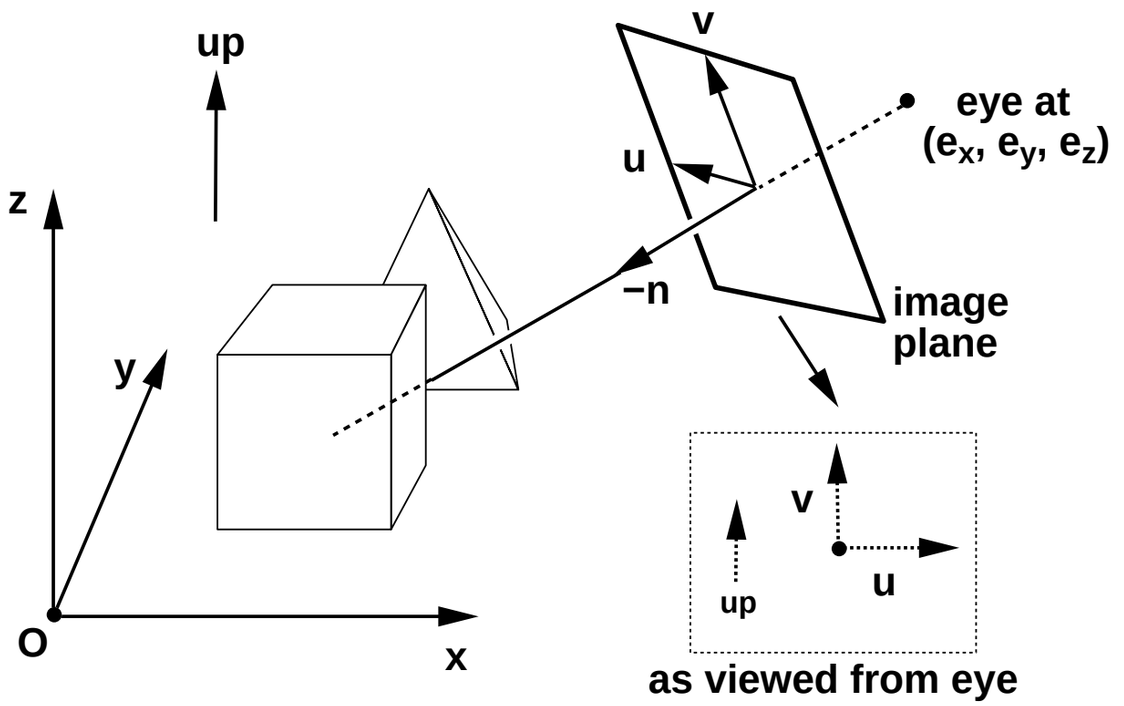
What are our "degrees of freedom" in camera positioning?
To achieve effective visual simulation, we want:

1) the eyepoint 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

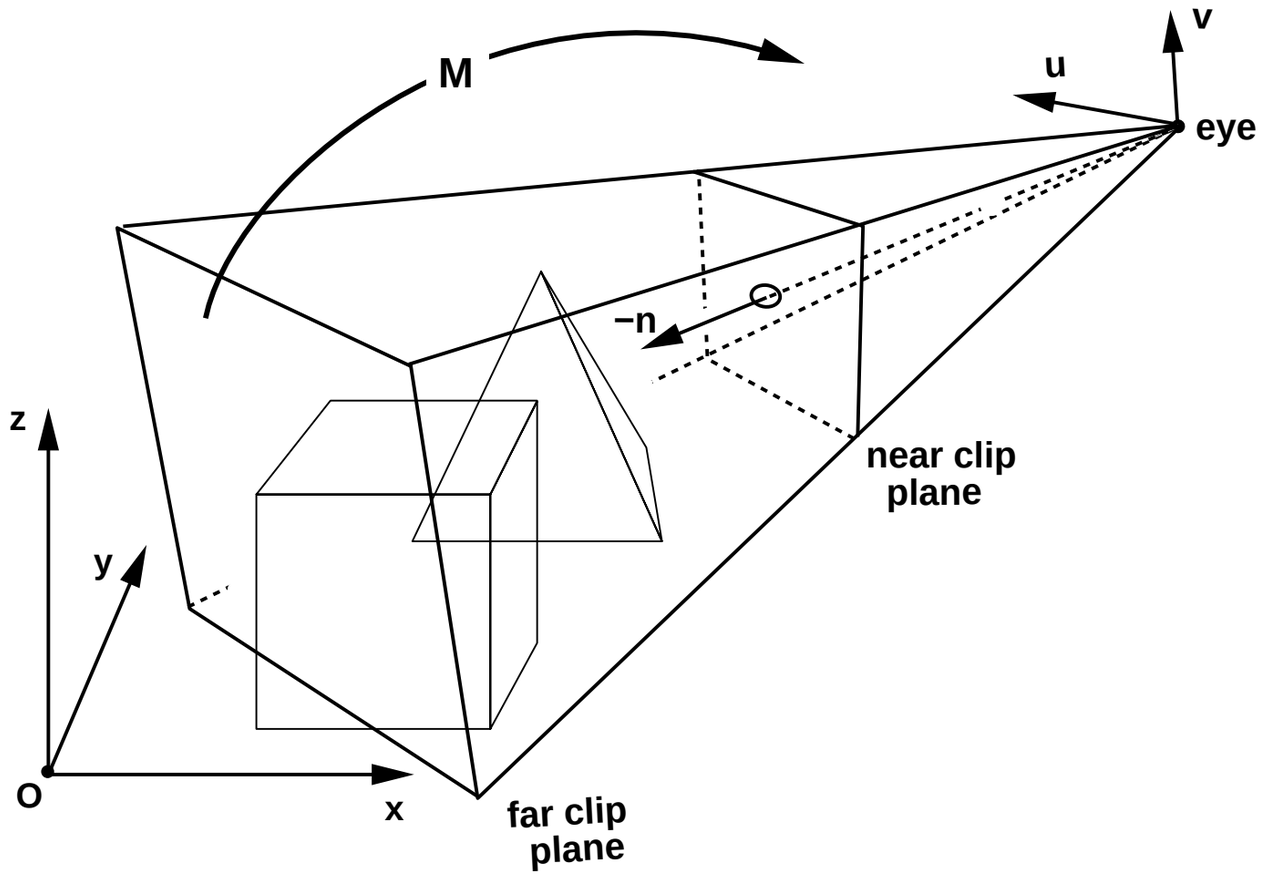
Define *eye coordinate* system **uvn** as follows:



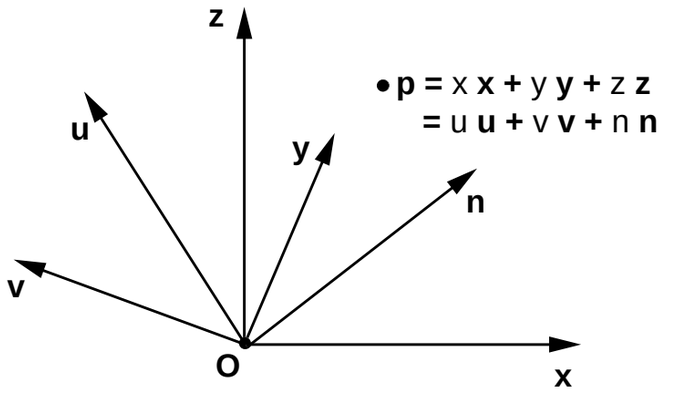
Eyepoint at origin
u axis toward "right" of image plane
v axis toward "top" of image plane
view direction along *negative* **n** axis

Transformation to Eye Coordinates

Our task: construct the transformation $\mathbf{M}$ that
re-expresses world coordinates in the viewer frame



Machinery: Changing Orthobases

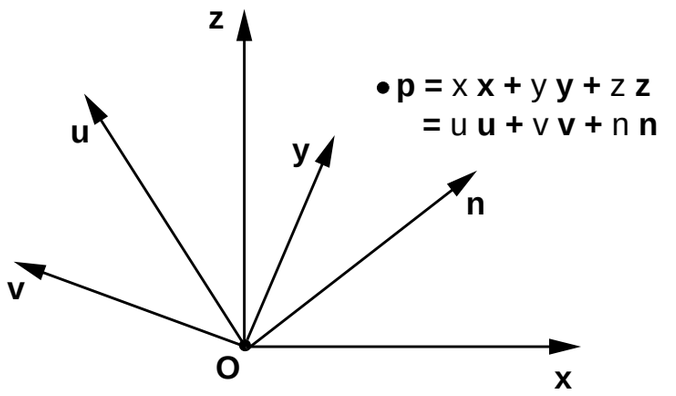


Suppose you are given an orthobasis $\mathbf{u}, \mathbf{v}, \mathbf{n}$
What is the action of the matrix $\mathbf{M}$ with
rows $\mathbf{u}, \mathbf{v}$, and $\mathbf{n}$ as below?

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

Applying M to u, v, n

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



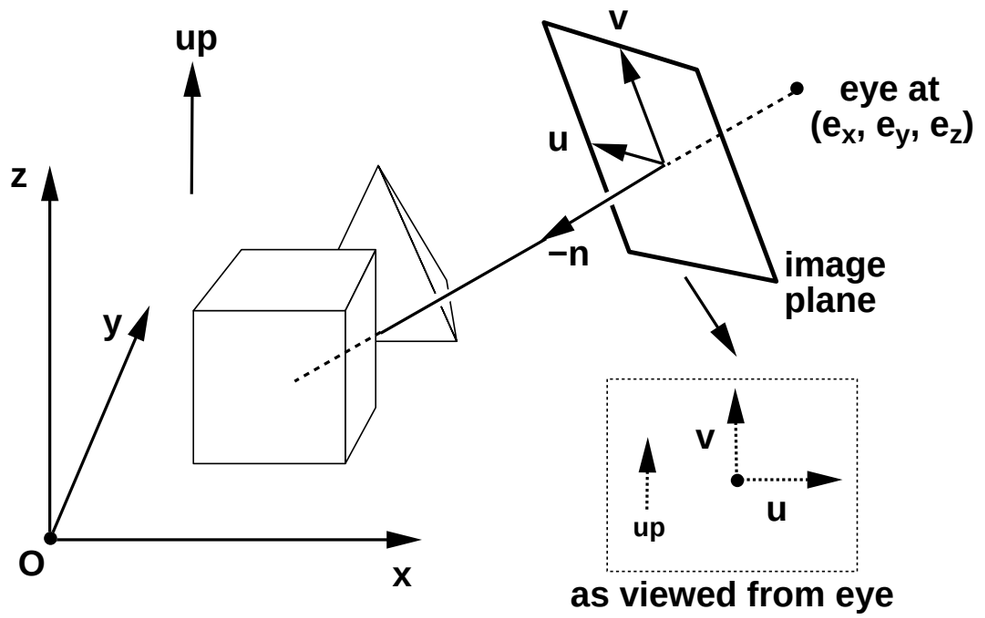
Two equally valid interpretations, depending on reference frame:

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 $\mathbf{e}$, basis $\hat{\mathbf{u}}$, $\hat{\mathbf{v}}$, $\hat{\mathbf{n}}$

Deduce $\mathbf{M}$ that expresses world in eye coordinates:

Overlay origins, then change bases:

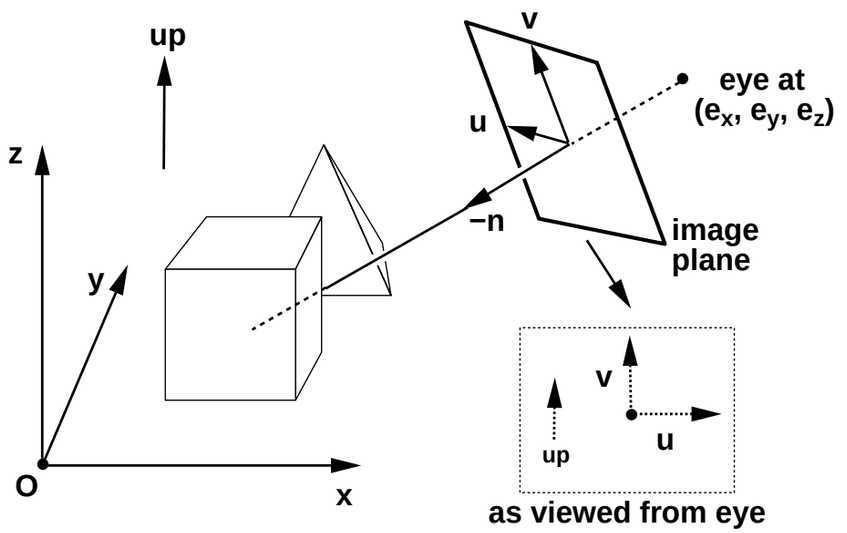
$$\mathbf{T} = \begin{pmatrix} 1 & 0 & 0 & -e_x \\ 0 & 1 & 0 & -e_y \\ 0 & 0 & 1 & -e_z \\ 0 & 0 & 0 & 1 \end{pmatrix}; \qquad \mathbf{R} = \begin{pmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

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

Positioning Synthetic Camera

Check: does $\mathbf{M}$ re-express world geometry in eye coordinates?

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

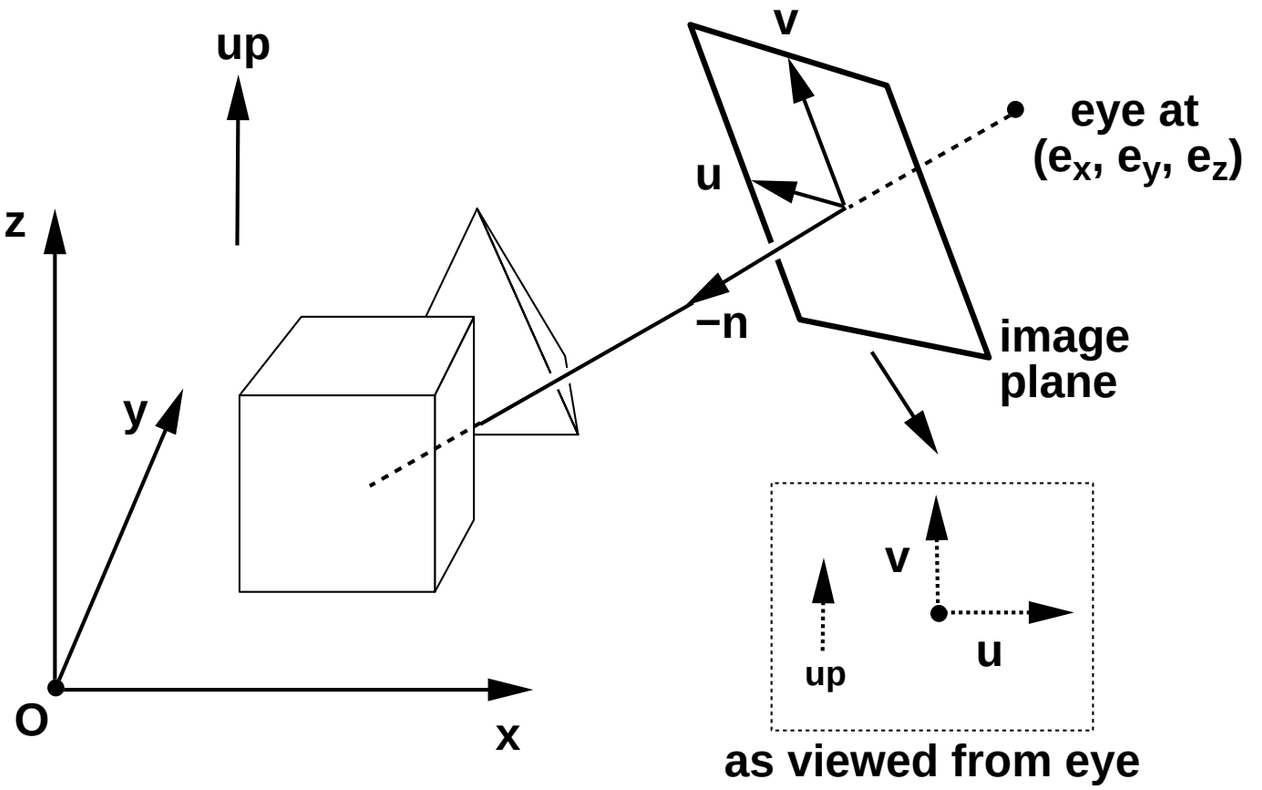


Positioning Synthetic Camera

Camera specification must include:

World-space eye position **e**

World-space "lookat direction" **-n**



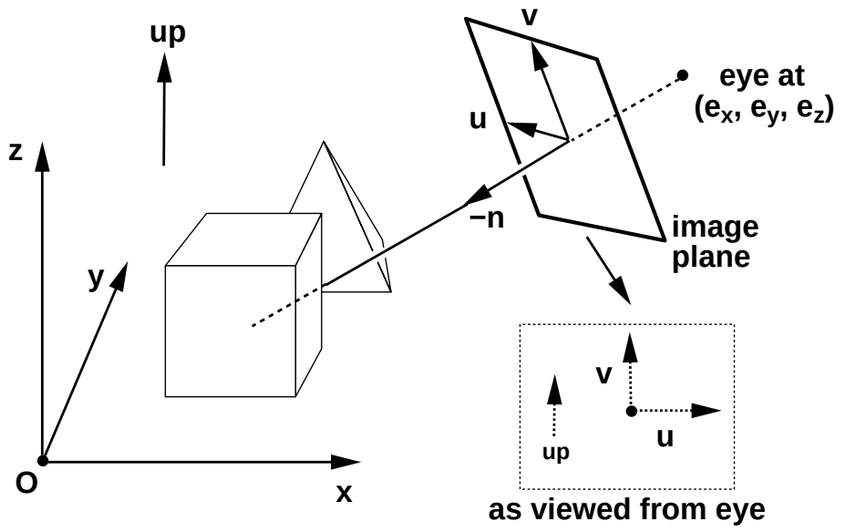
Are **e** and **-n** enough to determine the camera DOFs (degrees of freedom)?

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**.

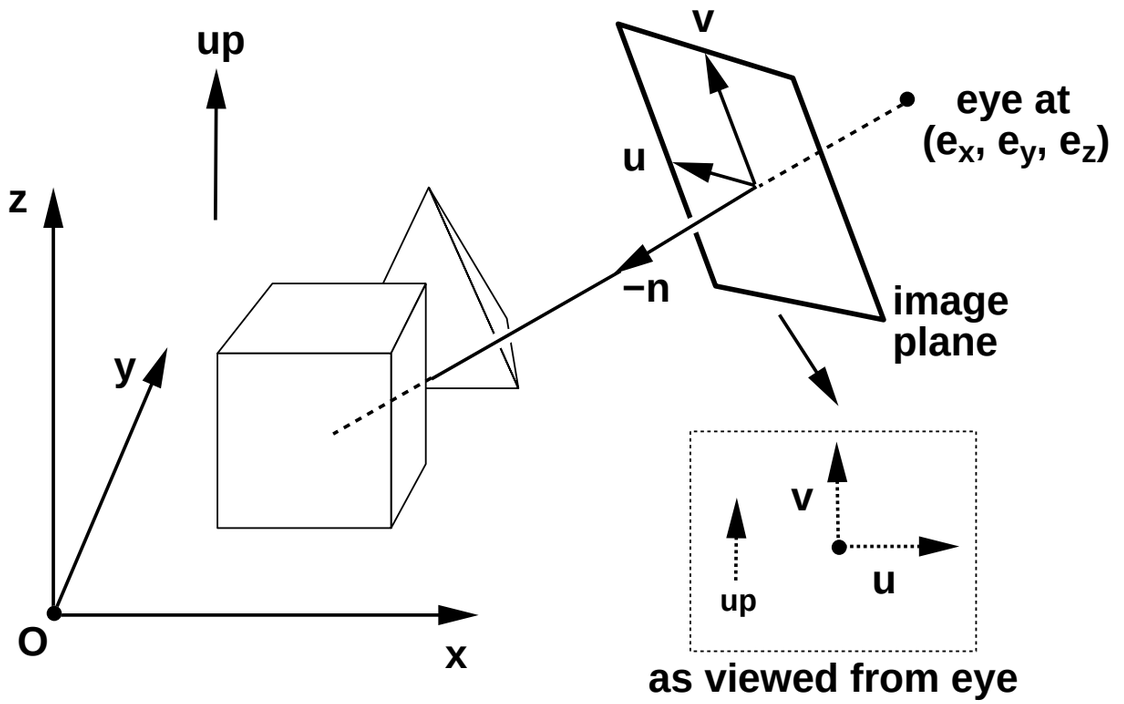
Typically done by specification of a world-space "**up** vector"

provided by user interface, e.g., using `gluLookat(e, c, up)`

"Twist" constraint: Align **v** with world **up** vector (How?)

Applying the Twist Constraint

Trick: *construct* $\mathbf{u}$ and $\mathbf{v}$ from available information!
"Twist" constraint: Align $\mathbf{v}$ with world $\mathbf{up}$ vector



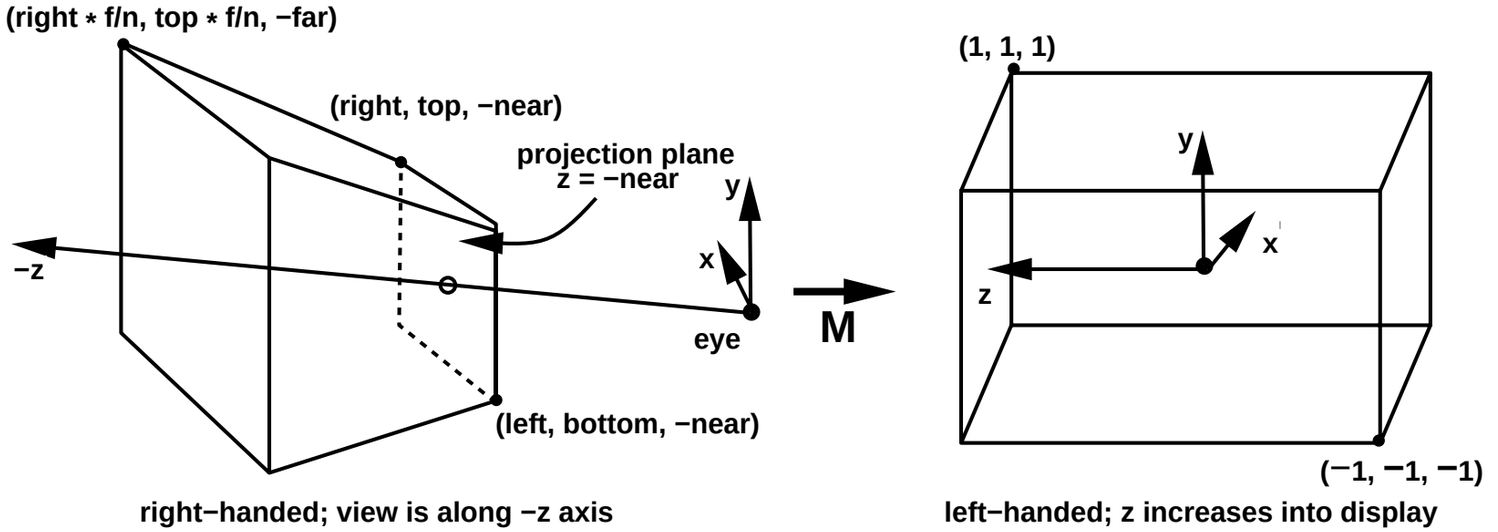
Given: eyepoint $\mathbf{e}$, view direction $\mathbf{n}$, and world $\mathbf{up}$ vector:

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

... Will this always work ?

Where are we?

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



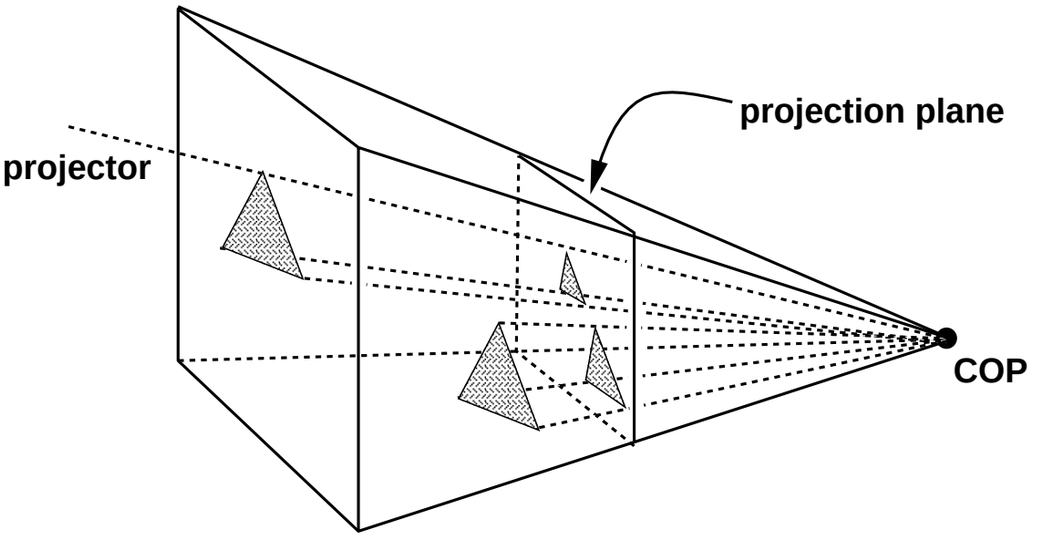
Next we must transform to NDC (Normalized Device Coordinates)
to prepare for (simple) clipping and projection

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)



Ray family ("projectors") emanates from COP ("center of projection"), through object onto "projection plane"

Parallel (orthographic) projection – COP at infinity

Perspective projection – COP local

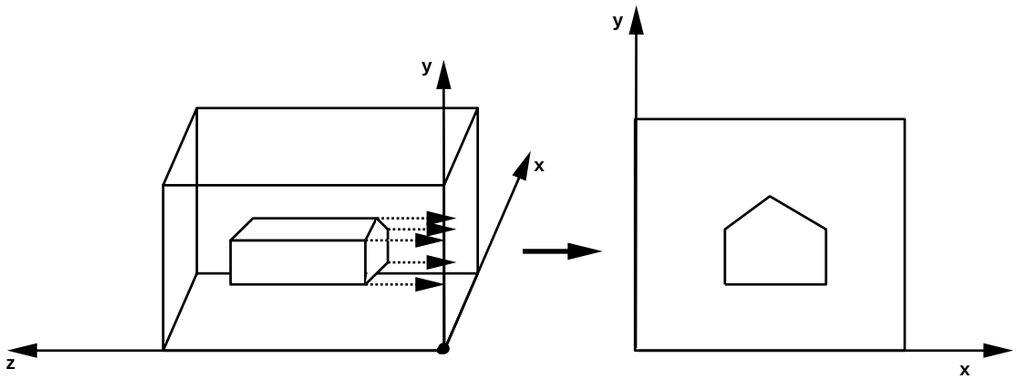
We'll use Planar Geometric Projections

"Planar" means projection is onto a plane

"Geometric" means projectors (rays) are straight

Distinction between perspective transformation and perspective projection

Orthographic Projection



Models eye "at infinity" along projection direction

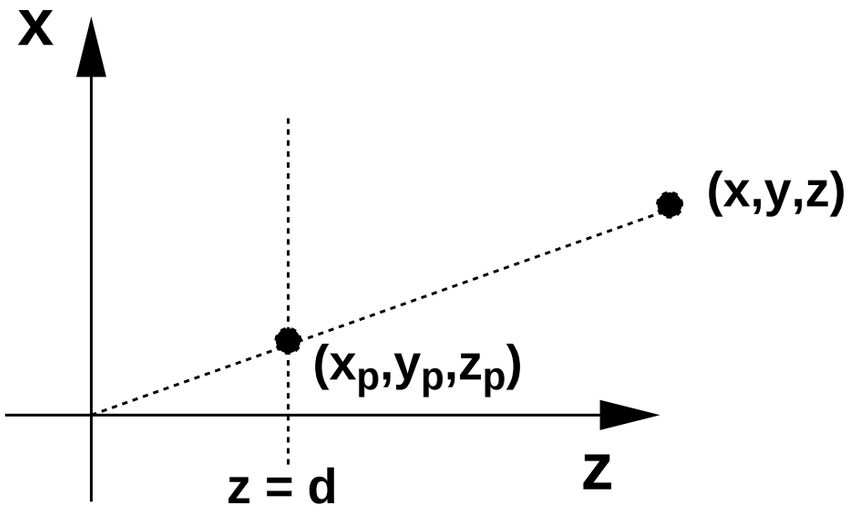
No decrease in apparent size with distance from eye

To implement projection in z, simply ignore z coordinate

Projected entity described by only 2 coordinates x, y

Perspective Projection

Example: project through origin onto plane $z = d$



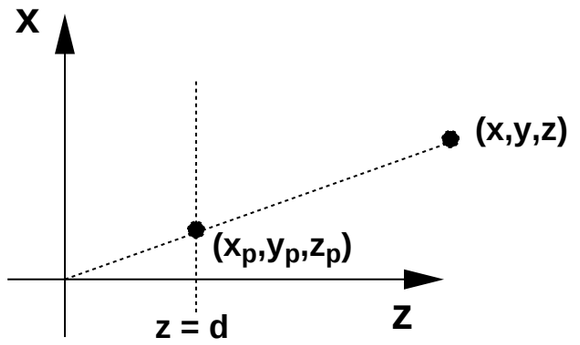
What are coordinates of projected point x_p, y_p, z_p?
By similar triangles,

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

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



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

Some observations:

Vary d: x_p and y_p scale linearly with d

Vary z: division by z shrinks distant objects

$z = 0$ 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":

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

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

Finally, recover projected point using *homogenous convention*:

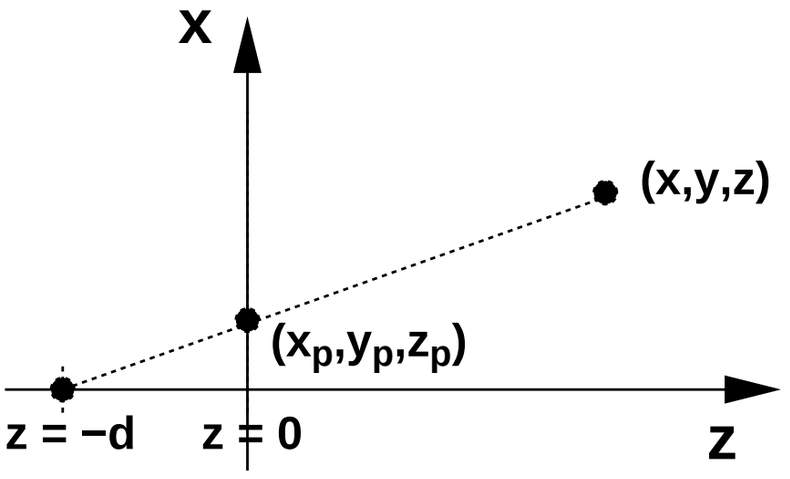
Divide by 4^{th} element to convert 4-vector to 3-vector:

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

(Believe it for now; we'll come back to this later in the term)

Another Perspective Projection

Projection *plane* at $z = 0$
Center of projection at $z = -d$



$$x_p = \frac{d \cdot x}{z + d} = \frac{x}{(z/d) + 1} \qquad y_p = \frac{d \cdot y}{z + d} = \frac{y}{(z/d) + 1}$$

In matrix form,

$$\mathbf{M} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \mathbf{0} & 0 \\ 0 & 0 & 1/d & 1 \end{pmatrix}; \qquad \mathbf{M} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ 0 \\ z/d + 1 \end{pmatrix}$$

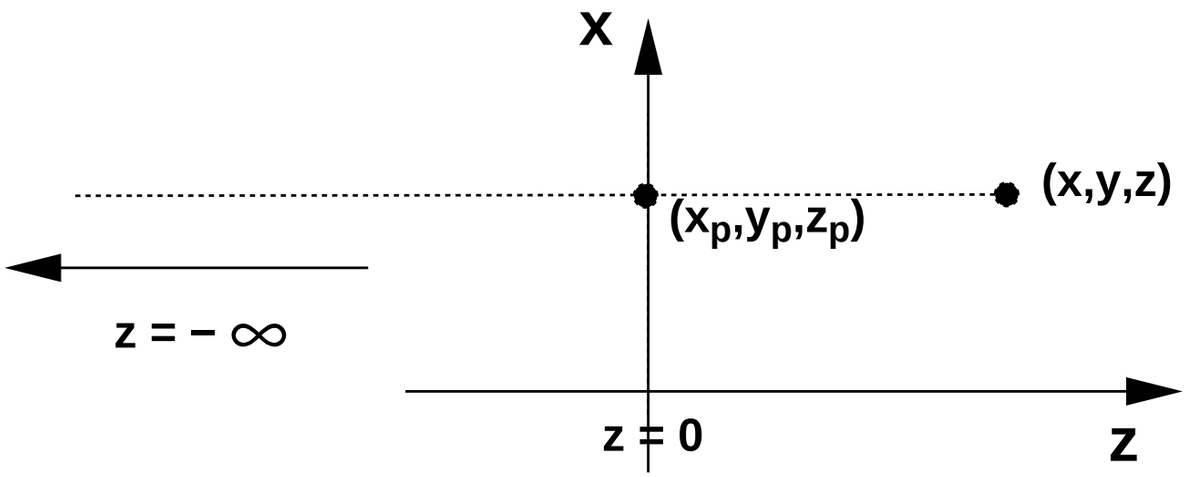
What is the limit of $\mathbf{M}$ as $d \to \infty$?

Matrix Formulation: Limit

Note that as $d \rightarrow \infty$

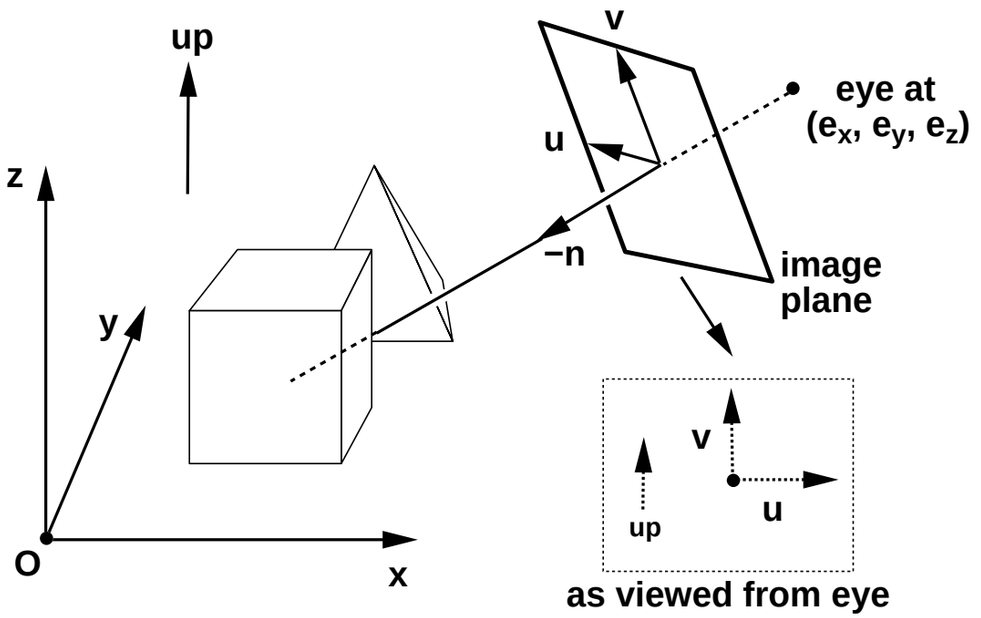
$$\mathbf{M} \rightarrow \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{0} & 1 \end{pmatrix}; \qquad \mathbf{M} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ 0 \\ 1 \end{pmatrix}$$

which is simply orthographic projection!



Ready to Rasterize?

We now know how to:

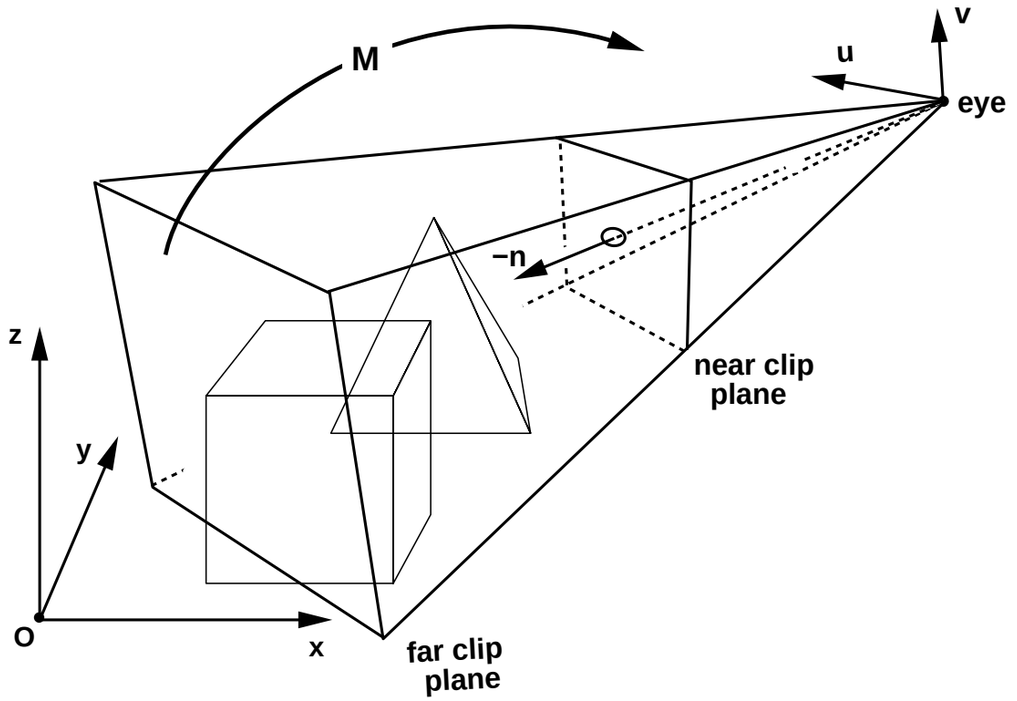


1. Transform to eye-centered coordinate frame
2. Perform perspective projection onto any plane $z \neq 0$

Are we now ready to rasterize onto the image plane?

Ready to Rasterize?

No! Two issues remain.



First, projection operator isn't defined everywhere

We wish to render only a *finite volume* of world space

So, we must define a *view frustum* and *clipping* method

Second, projection *destroys depth ordering*!

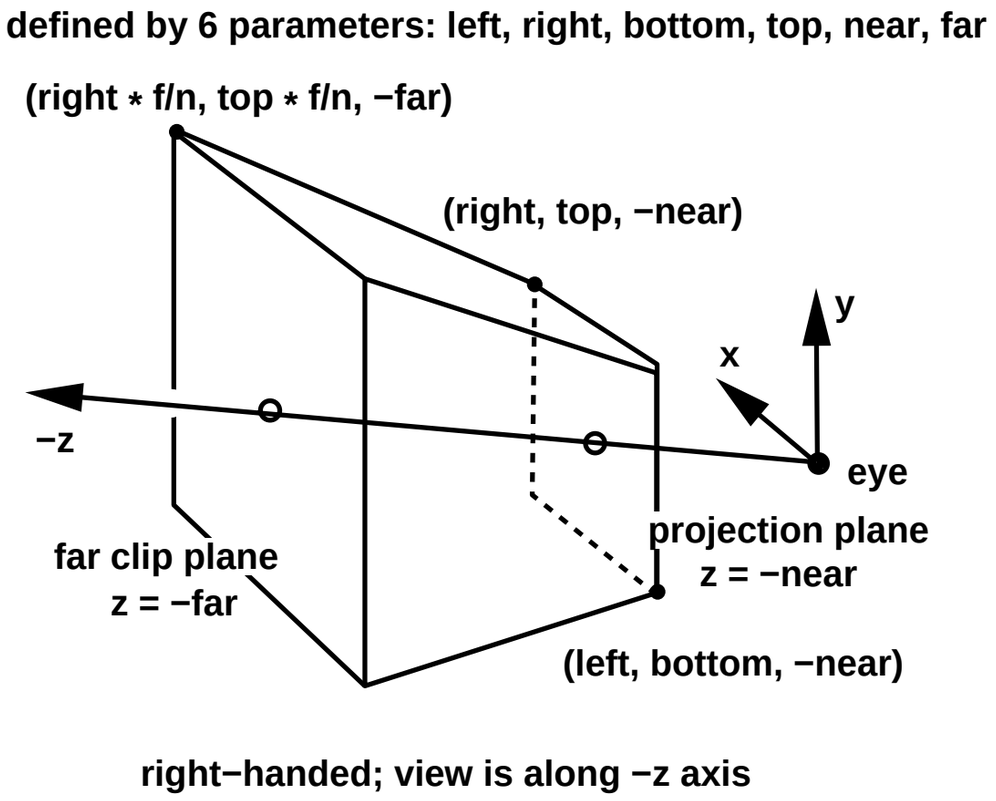
Leaves us no way to determine nearest surface at each pixel

We must modify the projection operator to *preserve depth order*

We'll look at projection first, then clipping (next time)

The View Frustum

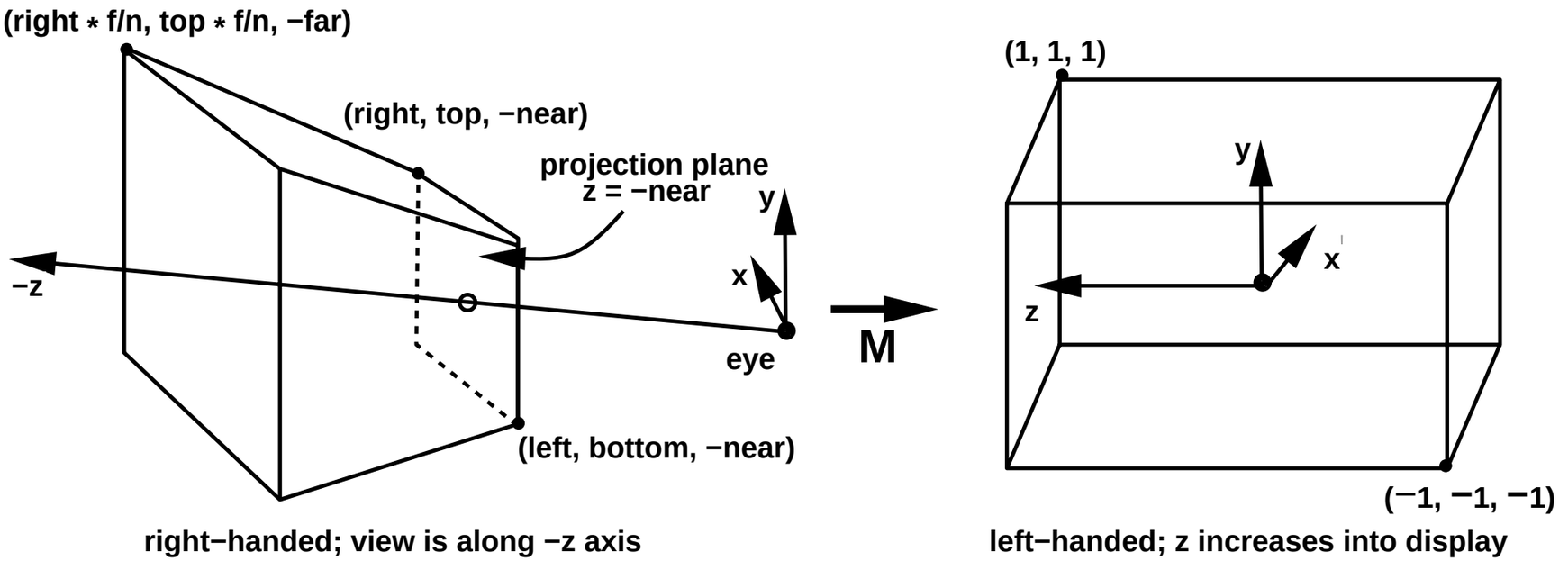
"View Frustum" defines world space *volume* to be rendered
It is a *truncated pyramid* bounded by six planes,
together defined by six positive parameters (l, r, b, t, n, f):



n and f define the frustum's near and far *clip planes*
l, b and r, t define the image boundary on the *near* plane

Canonical View Volume

Right parallelepiped bounded by $x = \pm 1$, $y = \pm 1$, $z = \pm 1$
Called NDC, or sometimes Clip Coordinates



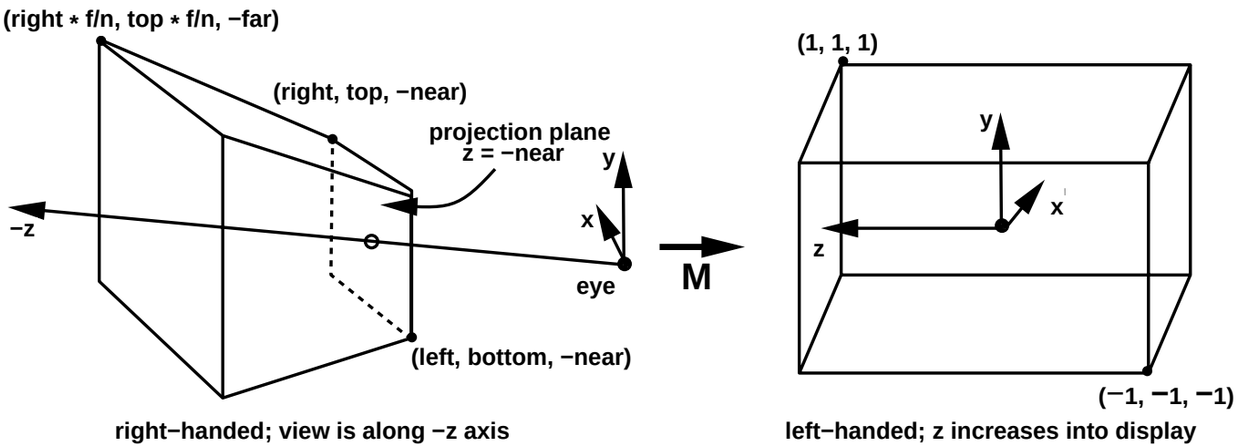
Where is the image plane in NDC?

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

Matrix Formulation

Matrix $\mathbf{M}$ that takes eye coordinates to NDC:

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



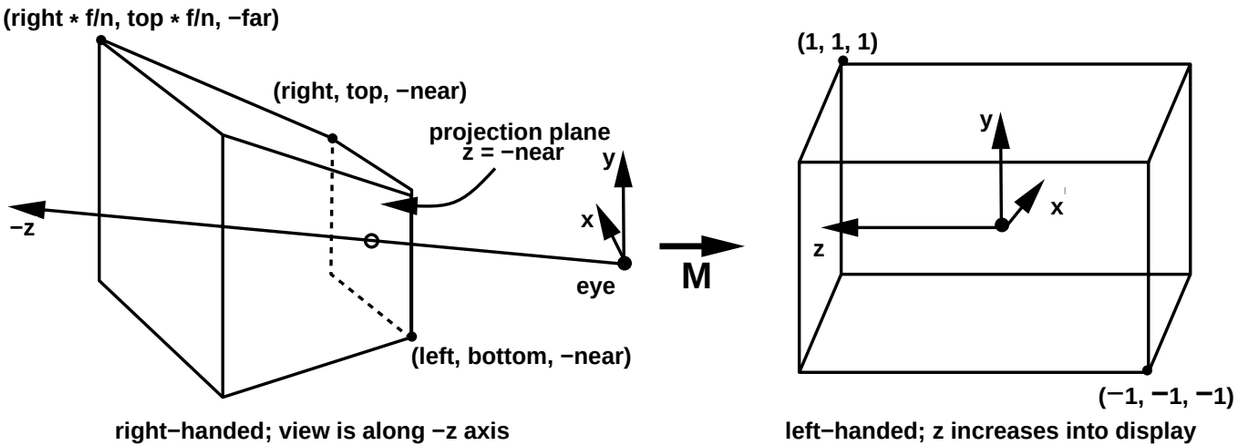
(This is the OpenGL form; several variations exist)
Check action of $\mathbf{M}$:

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

Symmetric Matrix Formulation

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

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



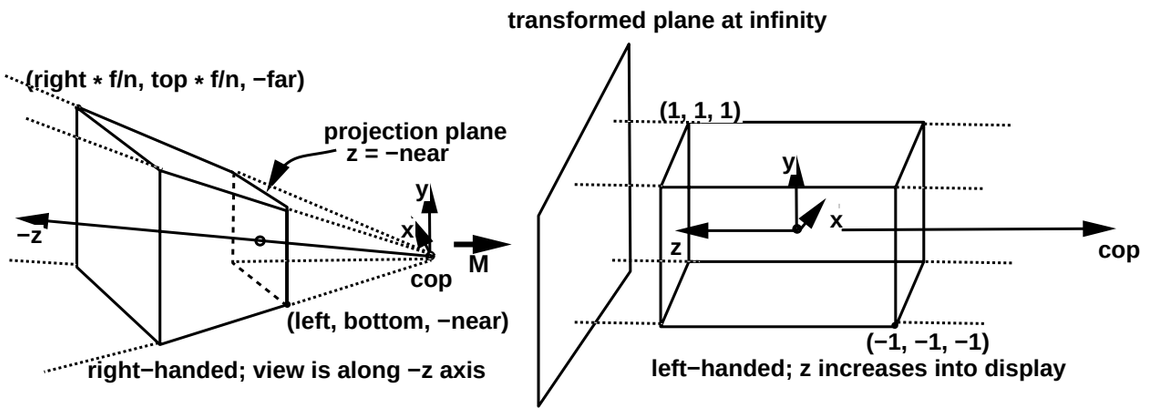
What happens to the Eye Point?

Perspective transformation matrix:

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

Transform Eye Point (in eye coordinates) by $\mathbf{M}$:

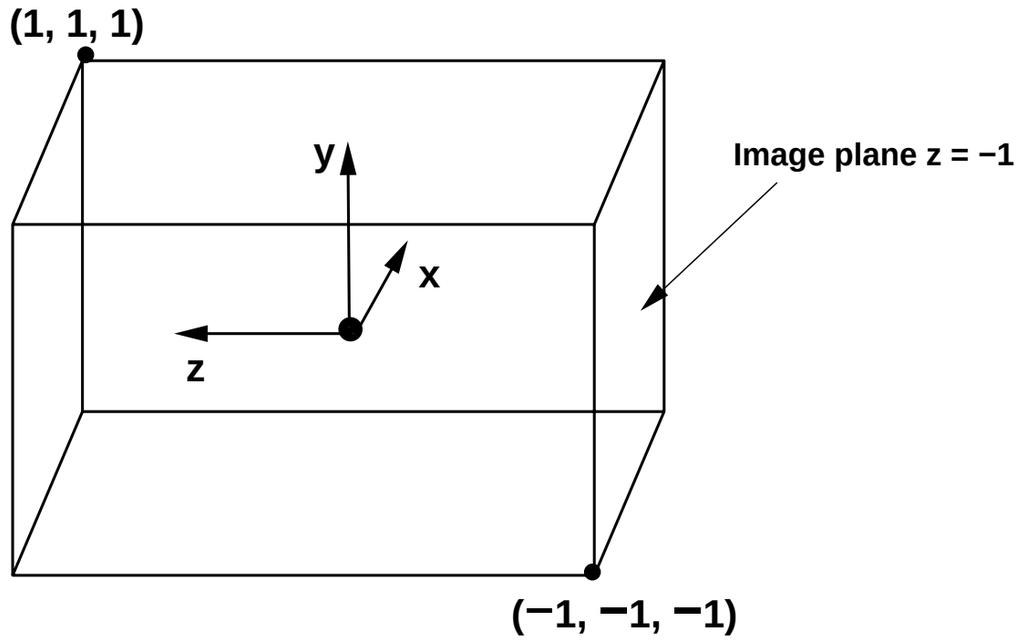
$$\mathbf{E}' = \mathbf{M}\,\mathbf{E} = \mathbf{M}\begin{pmatrix} 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & -\frac{2fn}{f-n} & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & -1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 & 0 \end{pmatrix}$$



What is the interpretation of 0 in the 4^{th} coordinate?

Perspective Projection

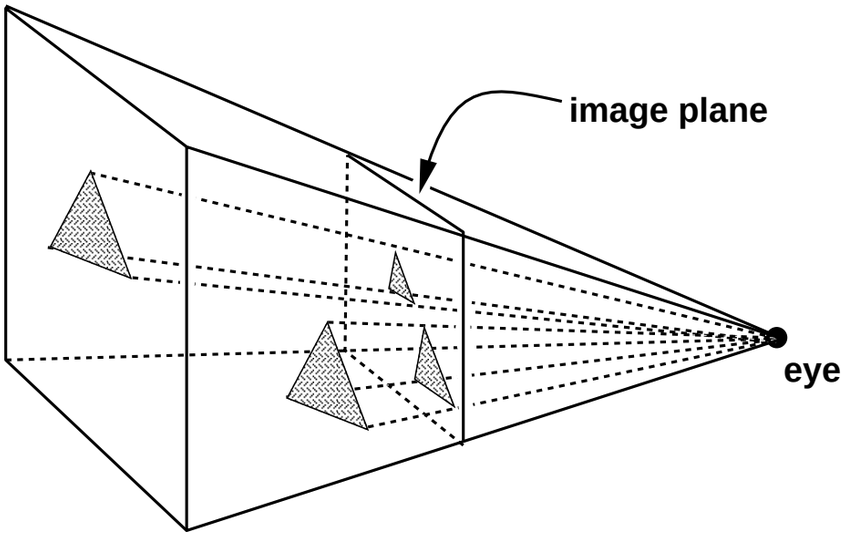
Suppose we have transformed from World to Eye to Canonical coordinates
How do we project onto "image plane"?



Simply ignore z coordinate!
(For hidden-surface elimination, we will use the fact that depth order is preserved. You should verify this on your own.)

Qualitative Features of Perspective Projection

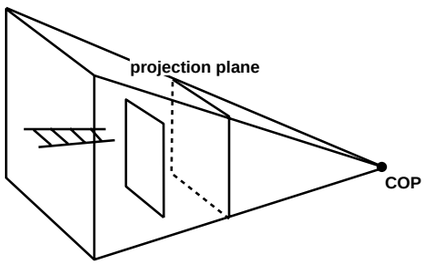
Study projection in eye-space reference frame (easier to visualize)
Equal-sized objects at different depths project to different sizes!

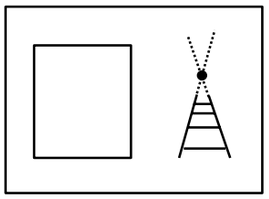


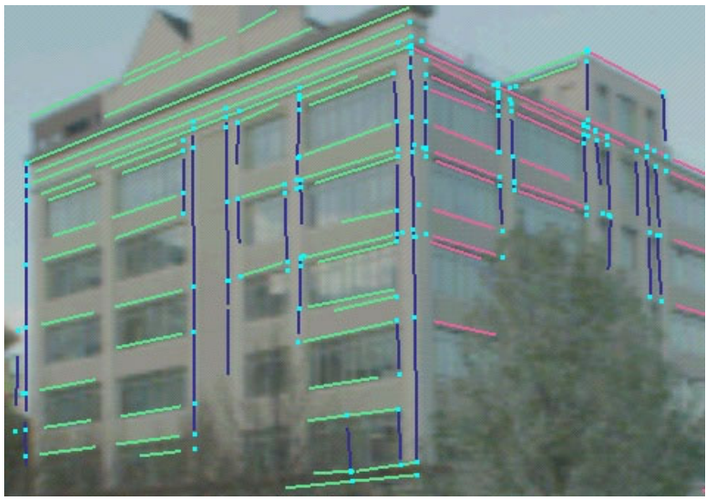
Perspective projection does *not* preserve shape of planar figures!
except those on planes parallel to projection plane

Qualitative Features of Perspective Projection

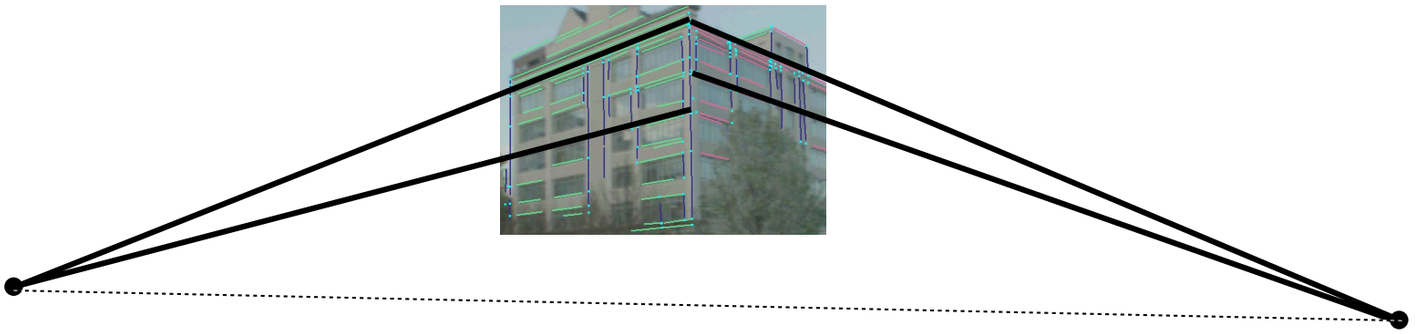
Families of parallel lines have "vanishing points"
projection of point at infinity in direction of lines







Vanishing Points



Except: line families *parallel* to projection plane have no (local) vanishing points

Putting it All Together:

We saw how to:

- Situate synthetic camera
- Define a viewing volume or *view frustum*
- Transform view volume into canonical volume
- Project contents of volume (trivially) onto image plane

Next Week:

- Continue with the rendering pipeline