Transforming the Normal Vector

- The normal to a polygon does **not** always transform in the same way as the points on the polygon
  - One case where is does: ?
  - One case where it does not: ?
- One way to find the tranformed normal is to first transform the polygon, and then re-compute the normal.
- We can often save some time by computing the transformed normal (*why can this save time*?)

Transforming the Normal Vector

- The normal to a polygon does **not** always transform in the same way as the points on the polygon
  - One case where is does: ?
  - One case where it does not: ?
- One way to find the tranformed normal is to first transform the polygon, and then re-compute the normal.
- We can often save some time by computing the transformed normal (*why can this save time*?)

Transforming the Normal Vector

- The normal to a polygon does **not** always transform in the same way as the points on the polygon
  - One case where is does: ?
  - One case where it does not: ?

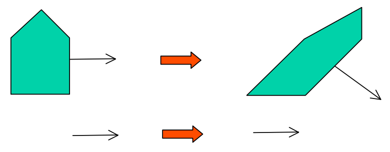
Transforming the Normal Vector

- The normal to a polygon does **not** always transform in the same way as the points on the polygon
  - One case where is does: **Orthogonal**

Somewhat intuitive

  - One case where it does not: **Shear**

$$M = \begin{vmatrix} 1 & a \\ 0 & 1 \end{vmatrix}$$



Details optional

Transforming the Normal Vector

Let **t** be a vector tangent to the polygon, and **n** the normal

M**t** is on the transformed polygon

Want a transformation N so that N**n** is perpendicular to all M**t**

$\mathbf{n} \cdot \mathbf{t} = \mathbf{n}^{\mathrm{T}}\mathbf{t} = 0$ and so (trick!) $\left(\mathbf{n}^{\mathrm{T}} M^{-1}\right)\left(\mathrm{M}\mathbf{t}\right) = 0$

So, $\mathbf{n}^{\mathrm{T}} M^{-1}$ is the row vector version of the transformed normal

So, $\mathrm{N}\mathbf{n} = (\mathbf{n}M^{-1})^{\mathrm{T}} = (M^{-1})^{\mathrm{T}}\mathbf{n}$

And $\mathrm{N} = (M^{-1})^{\mathrm{T}}$

Transforming the Normal Vector

Derived transformation for the normal: $\mathrm{N} = (M^{-1})^{\mathrm{T}}$

Note that Nn is not necessarily a unit vector

The formula proves that orthogonal transformations also transform the normal because orthogonal transformations satisfy:

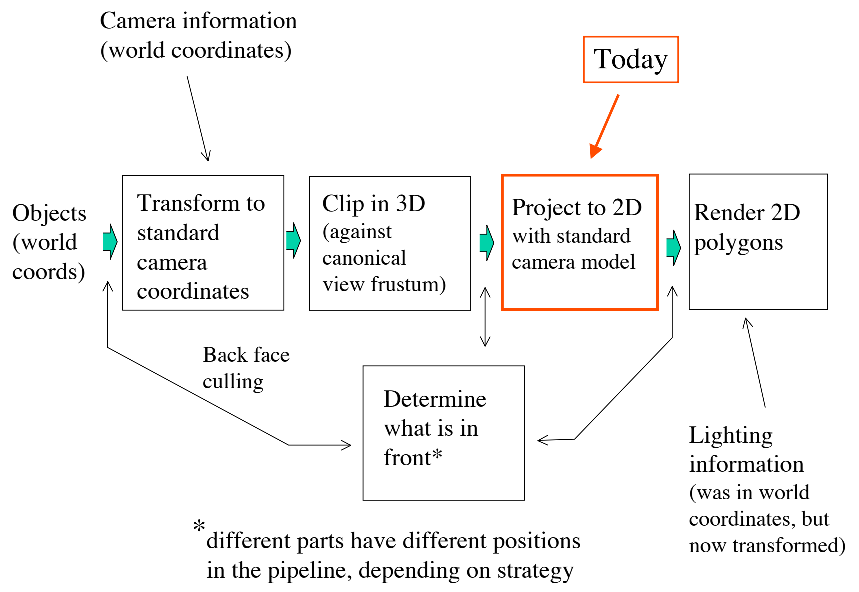
$$\mathrm{M} = (M^{-1})^{\mathrm{T}}$$

(The inverse of an orthogonal matrix is its transpose)

3D Graphics Concepts

(H&B ch, 7, Watt ch. 5, Foley et al ch. 6)

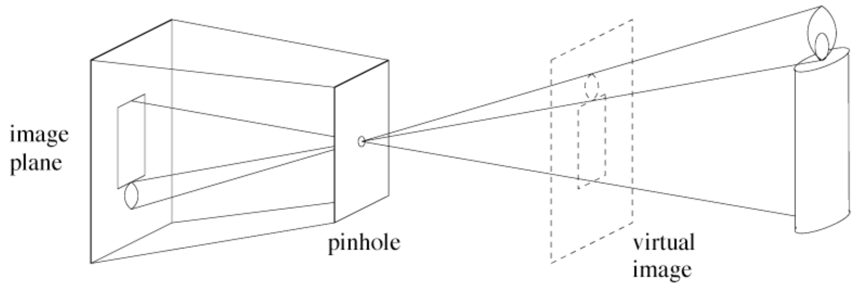
- Modeling: For now, objects will be collections of polygons in 3D. Complex shapes will be many small polygons.
- Issues:
  - Which polygons can be seen? (some polygons hide others, and some are outside the relevant volume of space and need to be clipped).
  - Where do they go in the 2D image? (key abstraction is a virtual camera)
  - How bright should they be? (for example, to make it look as if we are looking at a real surface)



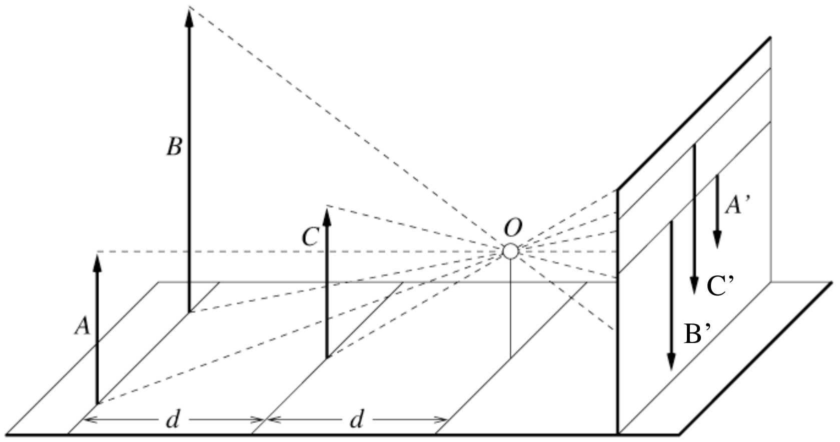
Camera information (world coordinates)
Today
Objects (world coords)
Transform to standard camera coordinates
Clip in 3D (against canonical view frustum)
Project to 2D with standard camera model
Render 2D polygons
Back face culling
Determine what is in front*
Lighting information (was in world coordinates, but now transformed)
*different parts have different positions in the pipeline, depending on strategy

Pinhole cameras

- Abstract camera model--box with a small hole in it
- Pinhole cameras work for deriving algorithms--a real camera needs a lens

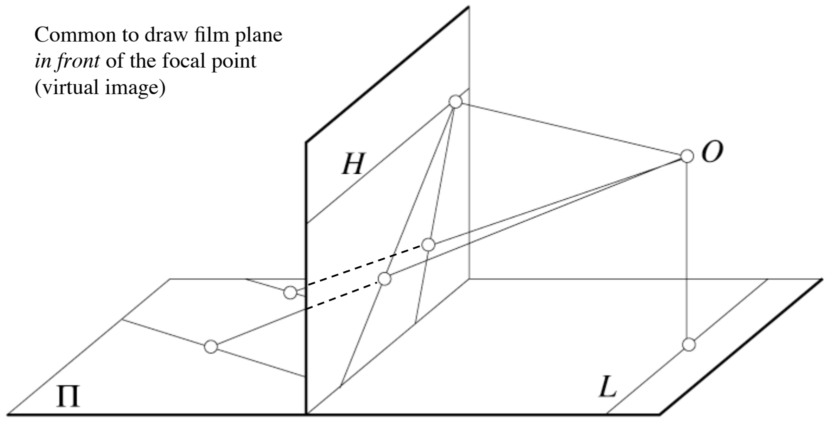


Distant objects are smaller



Parallel lines meet*

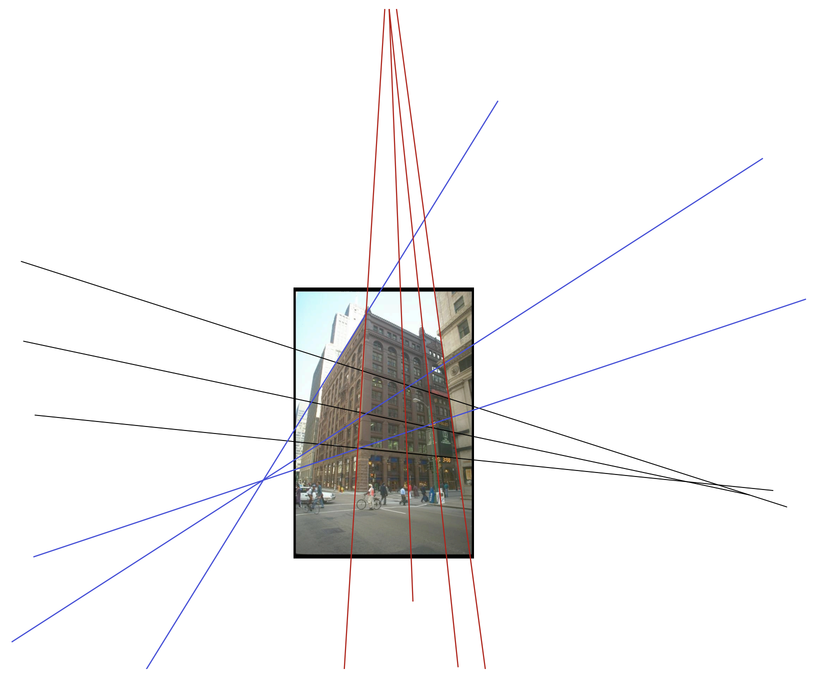
Common to draw film plane *in front* of the focal point (virtual image)

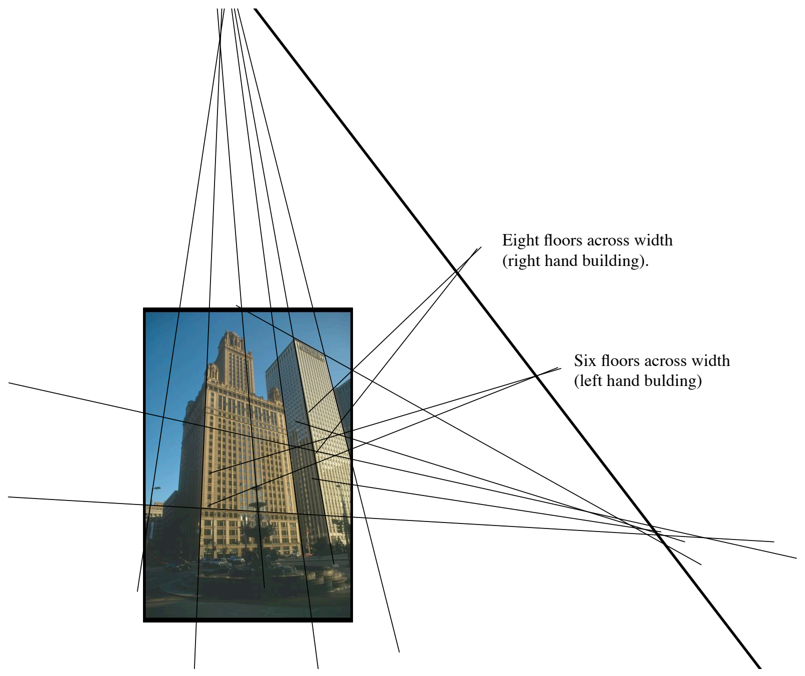


*Exceptions?

Vanishing points

- Each set of parallel lines (=direction) meets at a different point
  - The *vanishing point* for this direction
- Sets of parallel lines on the same plane lead to *collinear* vanishing points.
  - The line is called the *horizon* for that plane
  - Standard horizon is the horizon of the ground plane.
- One way to spot fake images

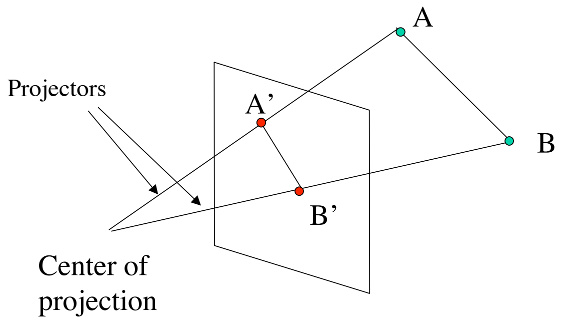




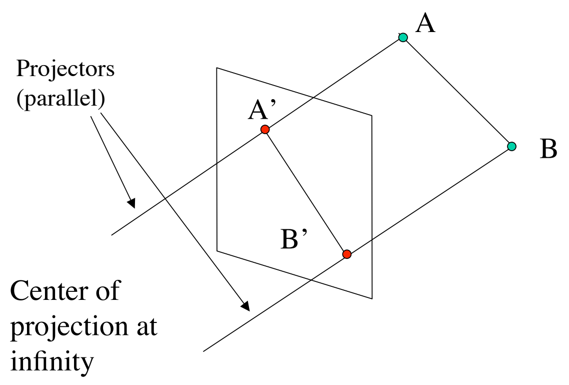
Projections

- Mathematical definition of a projection: PP=P
- (Doing it a second time has no effect).
- Generally rank deficient (non-invertable)--exception is P=I
- Transformation looses information (e.g., depth)
- Given a 2D image, there are many 3D worlds that could have lead to it.

Projections



Parallel Projection



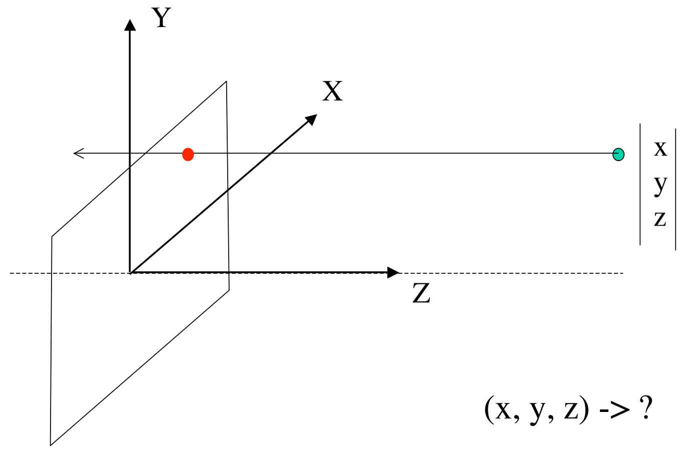
Parallel Projection

Parallel lines remain parallel, some 3D measurements can be made using 2D picture

Does not give realistic 3D view because eye is more like perspective projection.

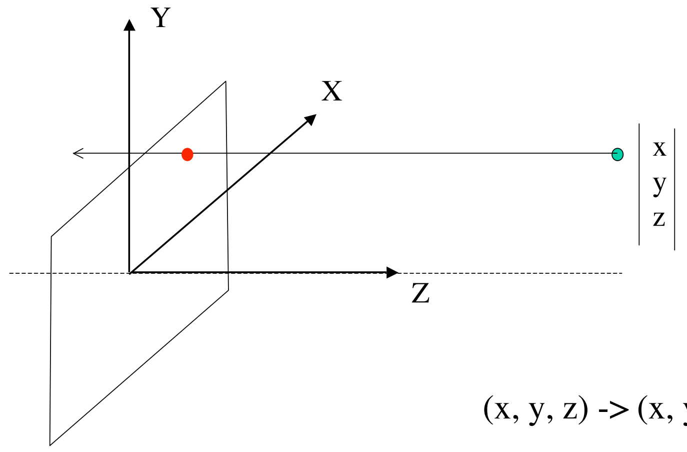
If projection plane is perpendicular to projectors the projection is *orthographic*

Orthographic example (onto z=0)



(x, y, z) -> ?

Orthographic example (onto z=0)



(x, y, z) -> (x, y, 0)

Can we do this with a linear transformation?

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

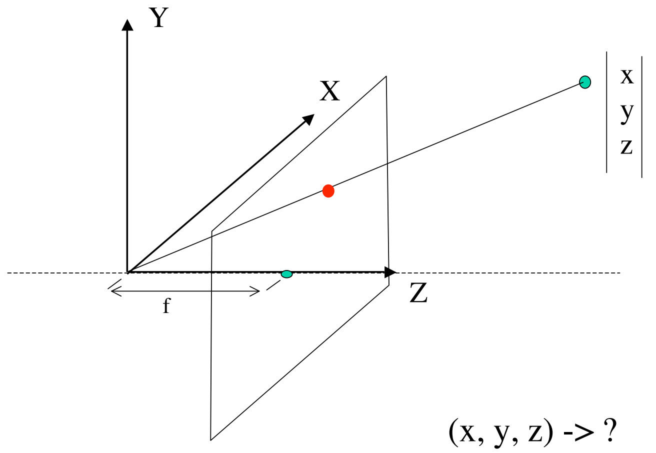
Can we do this with a linear transformation?

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

Camera matrix in homogeneous coordinates

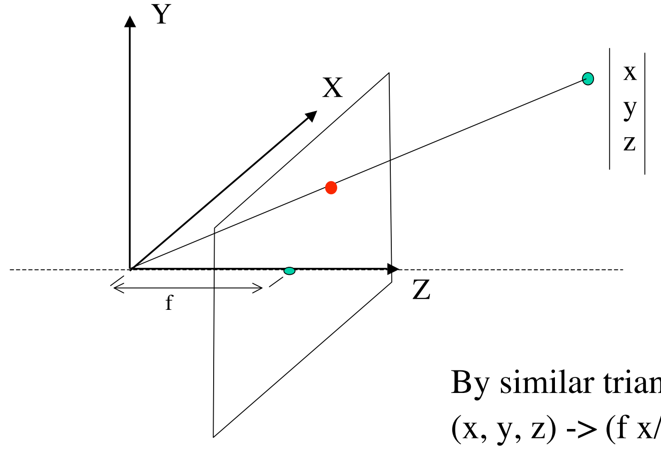
$$\begin{pmatrix} x \\ y \\ 0 \\ w \end{pmatrix} = \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 0 & \\ & & & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

Perspective example (onto z=f)



(x, y, z) -> ?

Perspective example (onto z=f)



By similar triangles,
(x, y, z) -> (f x/z, f y/z, f)

Can we do this with a linear transformation?

$$f\begin{pmatrix} x/z \\ y/z \\ 1 \end{pmatrix} = \begin{bmatrix} & ? & \end{bmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Now try homogeneous coordinates

- In homogeneous coordinates

$$(x,\ y,\ z,\ 1)\ \Rightarrow\ (f\frac{x}{z},\ f\frac{y}{z},\ f,\ 1)$$

- Equivalently

$$(x,\ y,\ z,\ 1)\ \Rightarrow\ (x,\ y,\ z,\ \frac{z}{f})$$

- (Now H.C. are being used to store foreshortening)

Is there a linear transformation?

$$\begin{pmatrix} x \\ y \\ z \\ \frac{z}{f} \end{pmatrix} = \begin{bmatrix} & ? & \end{bmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

The camera matrix

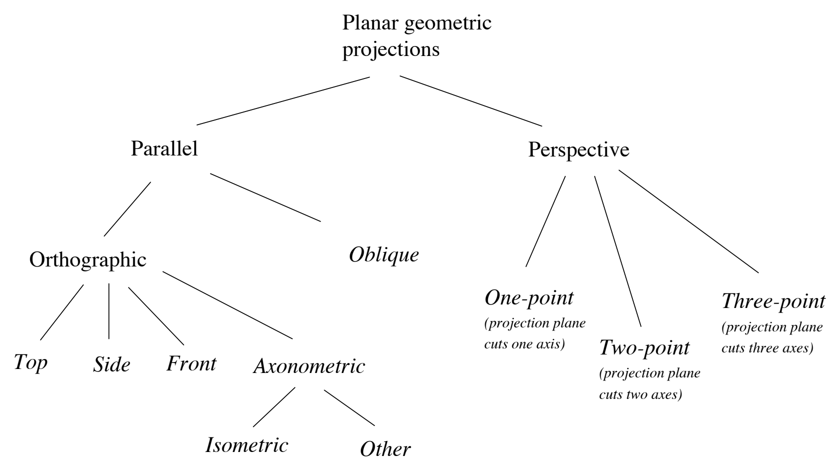
$$\begin{pmatrix} x \\ y \\ z \\ \frac{z}{f} \end{pmatrix} = \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & \frac{1}{f} & 0 \end{bmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Summary so far

- Parallel and perspective are projections (non-invertable)
- Perspective viewing is not a linear transform in (x,y,z)
- Perspective viewing can be encoded as a linear tranformation in homogenous coordinates
- Perspective becomes parallel projection as *f* becomes infinite

Projection Taxonomy

Terminology in italics is optional

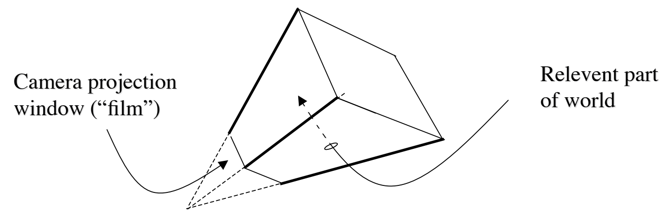

Planar geometric projections
Parallel
Perspective
Orthographic
Oblique
Top
Side
Front
Axonometric
Isometric
Other
One-point
(projection plane cuts one axis)
Two-point
(projection plane cuts two axes)
Three-point
(projection plane cuts three axes)

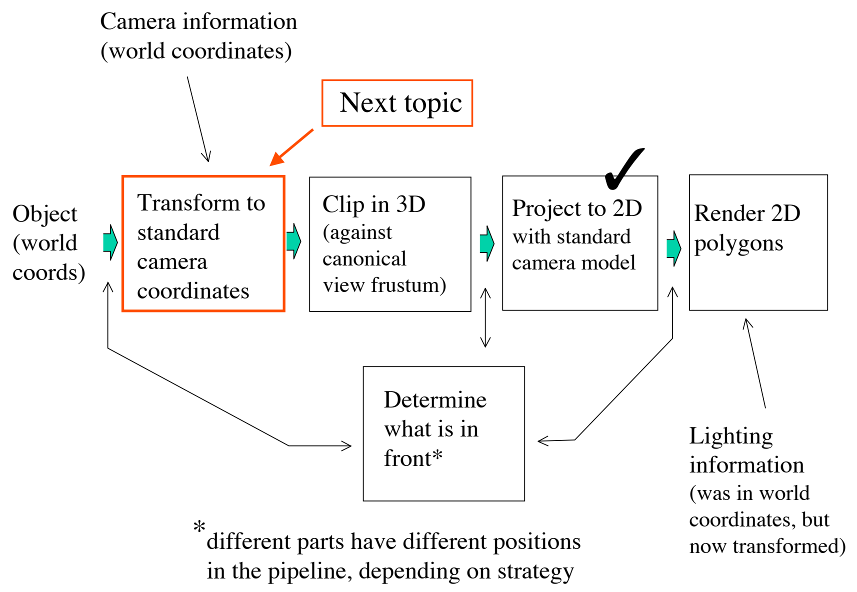
Specifying a camera

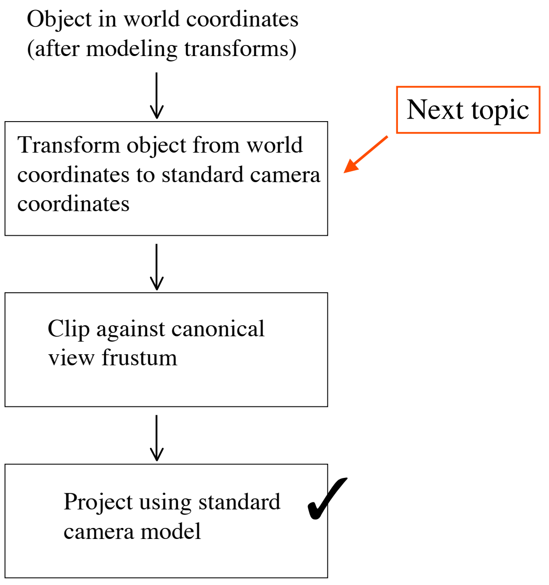
- Makes sense to tell rendering system where camera is in world coordinates
- We want to transform the world into camera coordinates so that projection is easy
  - In particular, we want to use the projection matrix from a few slides back.
- Need to specify focal point and film plane.
- Convenient to construct a coordinate system for the camera with origin on film plane

Clipping volume

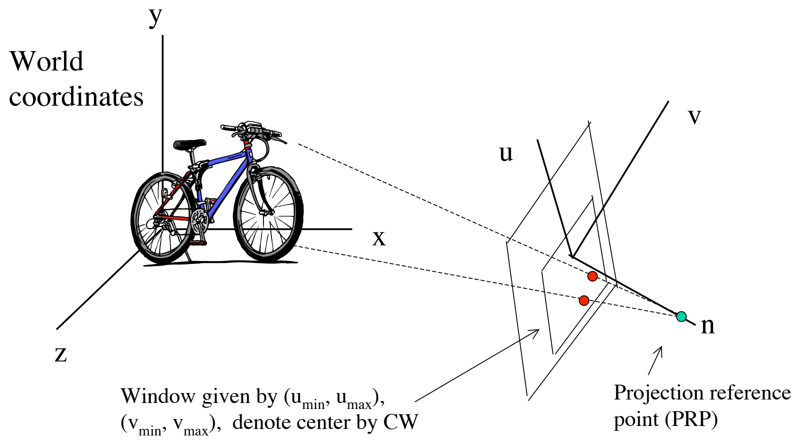
- We render only a window in the film plane
- Things beyond any of four sides don’t get rendered
- Things that are too far away don’t get rendered
- Things that are too near don’t get rendered
- Taken together, this means that everying outside a truncated pyramid (frustum) is irrelavent.




Camera information (world coordinates)
Next topic
Object (world coords)
Transform to standard camera coordinates
Clip in 3D (against canonical view frustum)
Project to 2D with standard camera model
✓
Render 2D polygons
Determine what is in front*
Lighting information (was in world coordinates, but now transformed)
*different parts have different positions in the pipeline, depending on strategy


Object in world coordinates (after modeling transforms)
Transform object from world coordinates to standard camera coordinates
Next topic
Clip against canonical view frustum
Project using standard camera model
✓

Camera coordinate system


World coordinates
y
x
z
u
v
n
Window given by (u_{min}, u_{max}), (v_{min}, v_{max}), denote center by CW
Projection reference point (PRP)

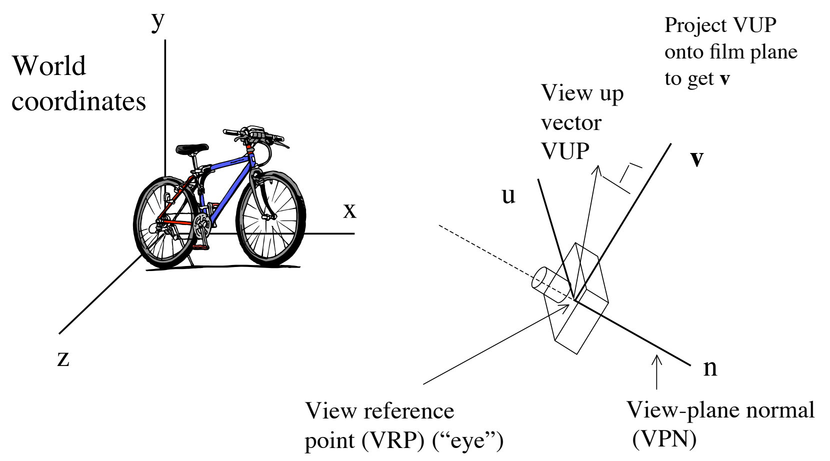
Specifying a camera

- We link camera specifications that relate to the user or the application to the (**u, n, v**) coordinate system
- The picture on the previous page is the most general case. Often one assumes that **n** goes through the center of the window (CW).

Specifying a camera

- There are many ways for the user to specify the camera. OpenGL has several. We will study one common and flexible one based on world coordinate entities "VUP" and "VPN"
- Our treatment is very similar to H&S (§7.3), but note that they call VUP, **V**

Specifying a camera



y
World coordinates
x
z
Project VUP onto film plane to get **v**
View up vector VUP
v
u
n
View reference point (VRP) ("eye")
View-plane normal (VPN)

Specifying a camera

- Why use VUP?
  - Convenient for the user but there are other ways (OpenGL has several ways to negotiate camera parameters, including one which is very much how we are doing it).
  - A world centric coordinate system is natural for the user. In particular, the user may think in terms of the camera rotation around the axis (**n**) relative to a natural horizon and/or "up" direction.
  - This will mean that VUP cannot be parallel to **n**. Often one will fix VUP (e.g. to the Y-axis) but this is too restrictive for some applications.
- Why use a "backwards" pointing **n**?
  - It is more natural to make the camera direction point the other way, but this makes the camera coordinates left handed. (You will see it done both ways).

Specifying a camera

- Together, the view reference point, VRP, and view plane normal, VPN=**n**, specify image plane.
- The up vector, VUP, gives an “up” direction in the image plane, providing for a user twist of camera about **n**.
- The camera coordinate system axis, **v,** is the projection of the up vector, VUP, into image plane.

Specifying a camera

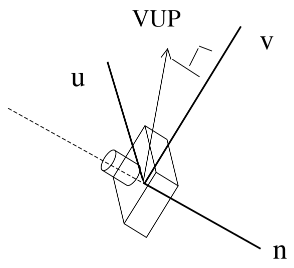
- We construct (**u**, **v**, **n**) so that it is a right handed coordinate system.
- This means that it is possible to map the world coordinates (**x, y, z**) to (**u**, **v**, **n**) so that (**x**->**u**, **y**->**v**, **z**->**n**) using only translations and rotations.

Computing (u,v,n) in world coordinates

v is the projection of VUP into the view plane which is perpendicular to **n**

u is perpendicular to the plane formed by **n** and VUP

So, we can easily compute a vector parallel to **u**.

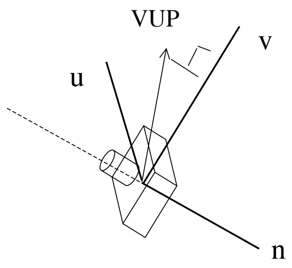


Computing (u,v,n) in world coordinates

$$\mathbf{u} \parallel \text{VUP} \times \mathbf{n}$$

$$\mathbf{u} = \frac{\text{VUP} \times \mathbf{n}}{\left|\text{VUP} \times \mathbf{n}\right|} = \frac{\text{VUP} \times \mathbf{n}}{\left|\text{VUP}\right|}$$

What about **v**?

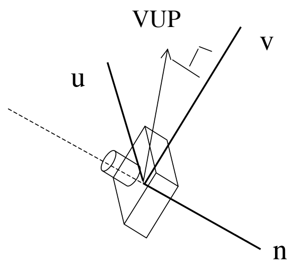


Computing (u,v,n) in world coordinates

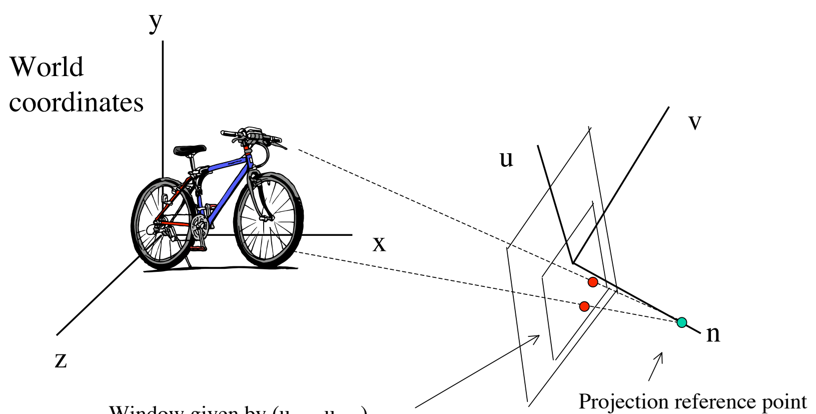
$$\mathbf{u} \parallel \mathrm{VUP} \times \mathbf{n}$$

$$\mathbf{u} = \frac{\mathrm{VUP} \times \mathbf{n}}{\left|\mathrm{VUP} \times \mathbf{n}\right|} = \frac{\mathrm{VUP} \times \mathbf{n}}{\left|\mathrm{VUP}\right|}$$

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



Specifying a camera



Window given by (u_{min}, u_{max}), (v_{min}, v_{max}), denote center by CW

Projection reference point (PRP)--assume it is on **n**

- VRP, VPN, VUP must be in world coords;
- PRP (focal point) could be in world coords, but more commonly, camera coords (which are the same scale as world coords)
- We will use camera coords, and further assume that PRP = (0,0,f).