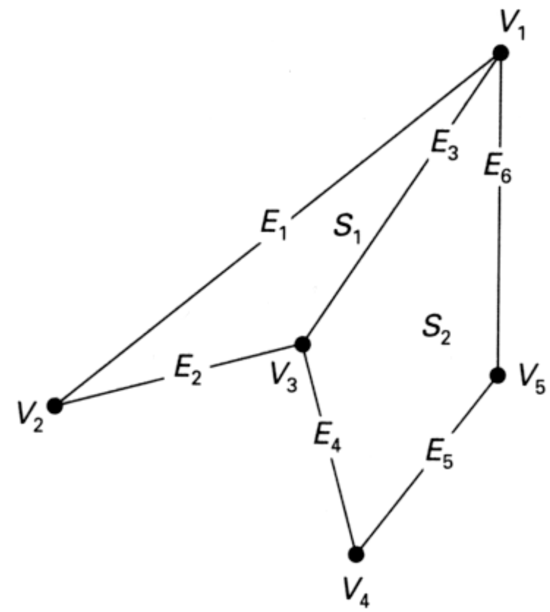
Models and The Viewing Pipeline

Jian Huang
CS456

Polygon Mesh

- Vertex coordinates list, polygon table and (maybe) edge table
- Auxiliary:
  - Per vertex normal
  - Neighborhood information, arranged with regard to vertices and edges



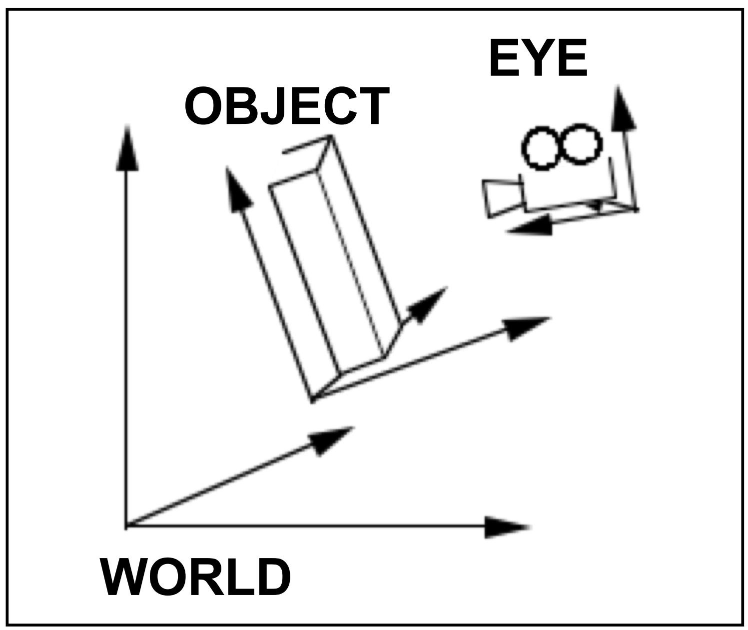
VERTEX TABLE	
V_1:	x_1, y_1, z_1
V_2:	x_2, y_2, z_2
V_3:	x_3, y_3, z_3
V_4:	x_4, y_4, z_4
V_5:	x_5, y_5, z_5

EDGE TABLE	
E_1:	V_1, V_2
E_2:	V_2, V_3
E_3:	V_3, V_1
E_4:	V_3, V_4
E_5:	V_4, V_5
E_6:	V_5, V_1

POLYGON-SURFACE TABLE	
S_1:	E_1, E_2, E_3
S_2:	E_3, E_4, E_5, E_6

Transformations – Need ?

- Modeling transformations
  - build complex models by positioning simple components
- Viewing transformations
  - placing virtual camera in the world
  - transformation from world coordinates to eye coordinates
- Animation: vary transformations over time to create motion



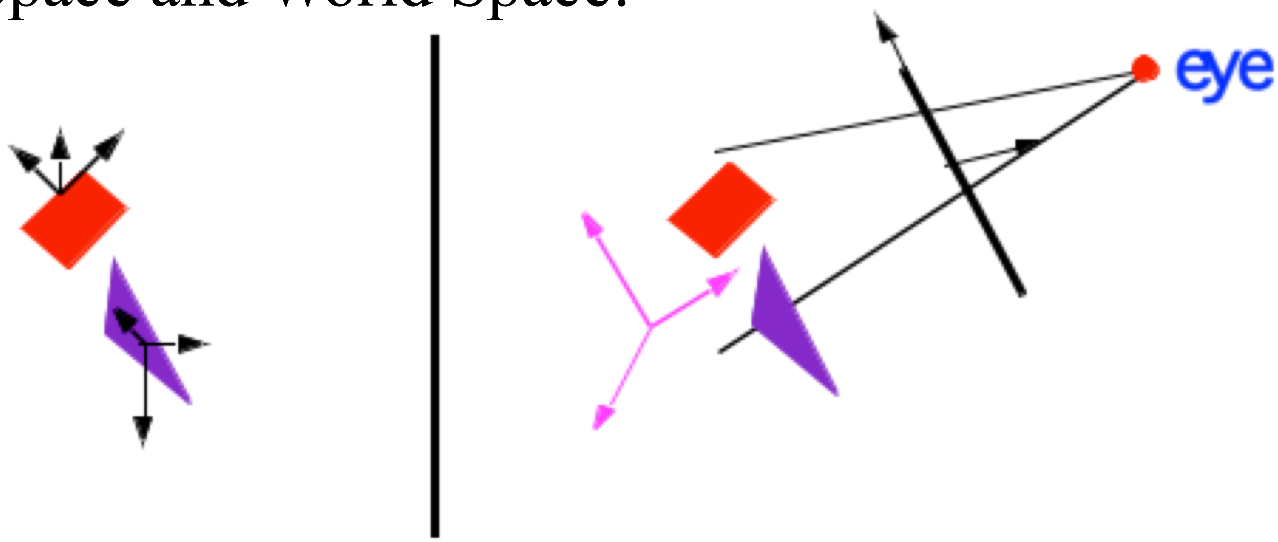
Viewing Pipeline

Object Space	World Space	Eye Space	Clipping Space	Canonical view volume	Screen Space

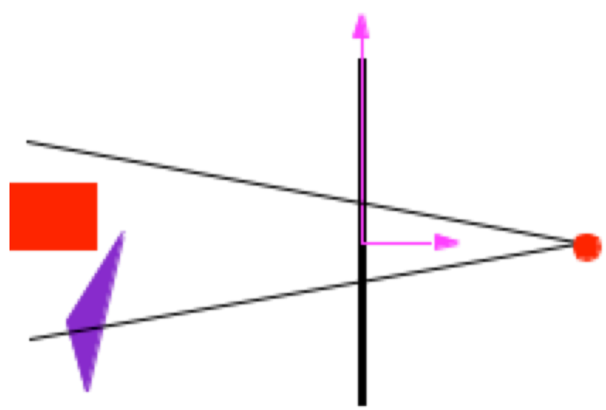
- Object space: coordinate space where each component is defined
- World space: all components put together into the same 3D scene via affine transformation. (camera, lighting defined in this space)
- Eye space: camera at the origin, view direction coincides with the z axis. Hither and Yon planes perpendicular to the z axis
- Clipping space: do clipping here. All point is in homogeneous coordinate, i.e., each point is represented by (x,y,z,w)
- 3D image space (Canonical view volume): a parallelpipied shape defined by (-1:1,-1:1,0,1). Objects in this space is distorted
- Screen space: x and y coordinates are screen pixel coordinates

Spaces

Object Space and World Space:



Eye-Space:



Spaces

Clip Space:

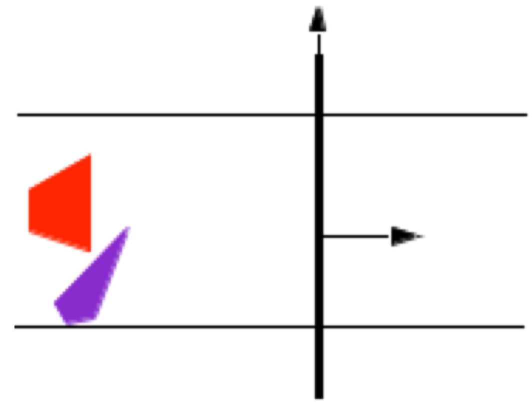
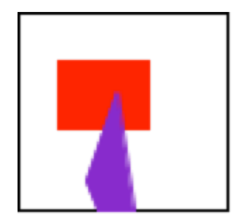
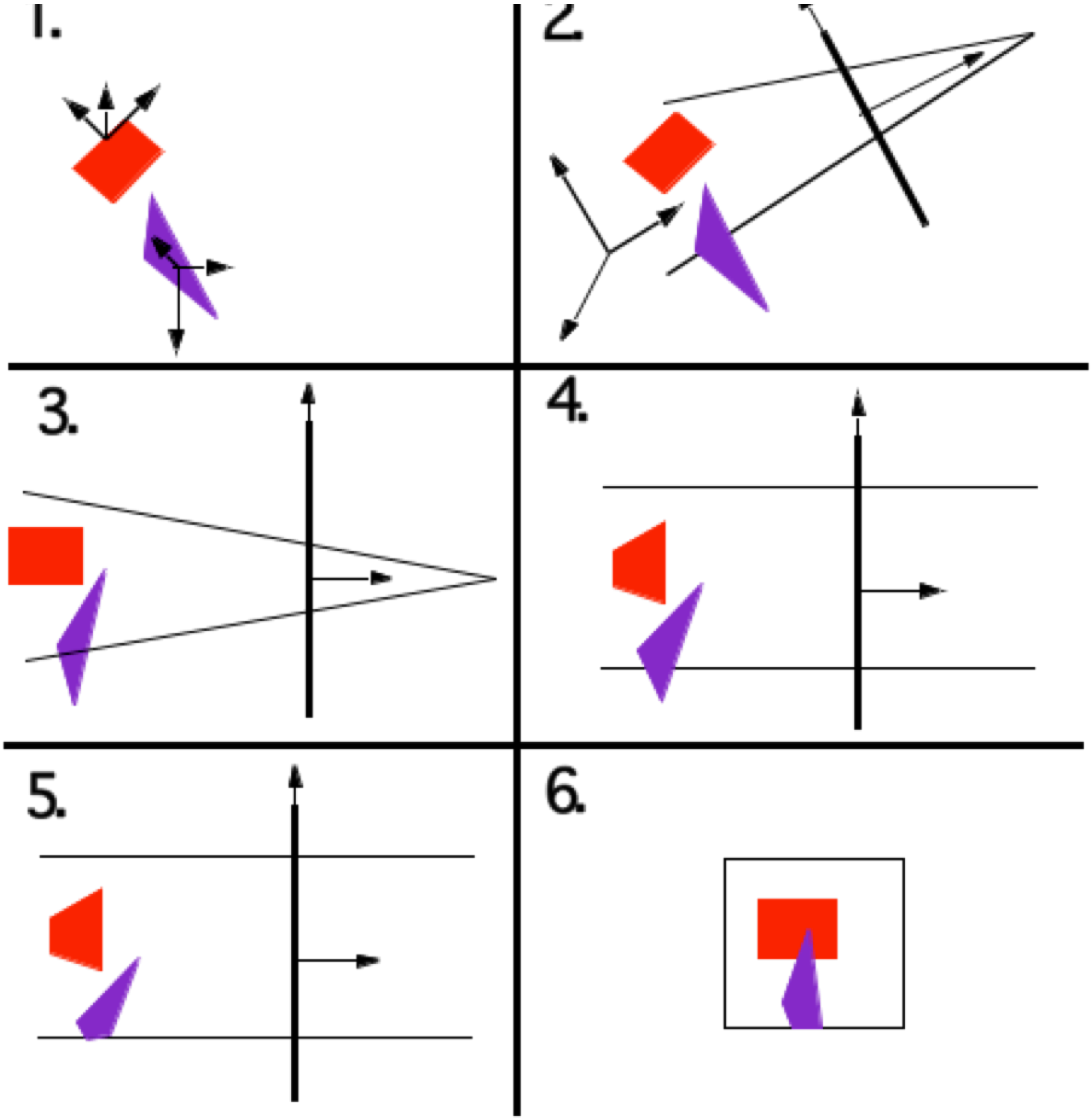


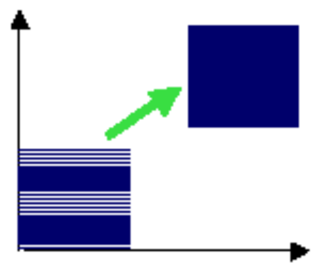
Image Space:




1.
2.
3.
4.
5.
6.

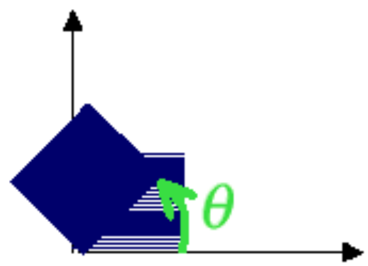
2D Transformation

- Translation



$$\begin{cases} x' = x + t_x \\ y' = y + t_y \end{cases}$$

- Rotation



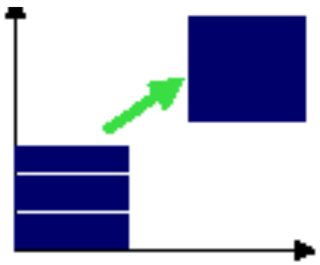
$$\begin{cases} x' = x \cdot \cos\theta \text{ - } y \cdot \sin\theta \\ y' = x \cdot \sin\theta + y \cdot \cos\theta \end{cases}$$

Matrix and Vector format:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = M \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Homogeneous Coordinates

- Matrix/Vector format for translation:



$$\begin{cases} x' = x + t_x \\ y' = y + t_y \end{cases}$$

Matrix format?

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = M \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} ?? & ?? \\ ?? & ?? \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$M = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Homogenous coordinates!

$$\begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix} = M \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} m_{00} & m_{01} & m_{02} \\ m_{10} & m_{11} & m_{12} \\ m_{20} & m_{21} & m_{22} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Translation in Homogenous Coordinates

- There exists an inverse mapping for each function
- There exists an identity mapping

$$M^{-1} = \begin{bmatrix} 1 & 0 & -t_x \\ 0 & 1 & -t_y \\ 0 & 0 & 1 \end{bmatrix}$$

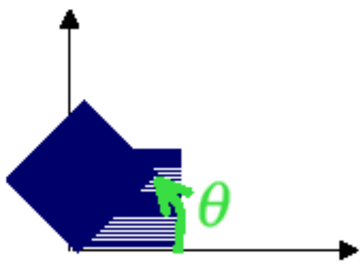
$$M\Big|_{\substack{t_x=0 \\ t_x=0}} = \begin{bmatrix} 1 & 0 & -t_x \\ 0 & 1 & -t_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = Identity(I)$$

Why these properties are important

- when these conditions are shown for any class of functions it can be proven that such a class is closed under composition
- i. e. any series of translations can be composed to a single translation.

$$x' = \underbrace{T_1 T_2 \bullet \cdots T_n}_{T'} x$$

Rotation in Homogeneous Space



$$\begin{bmatrix} x' \\ y' \end{bmatrix} = M \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$M_R = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

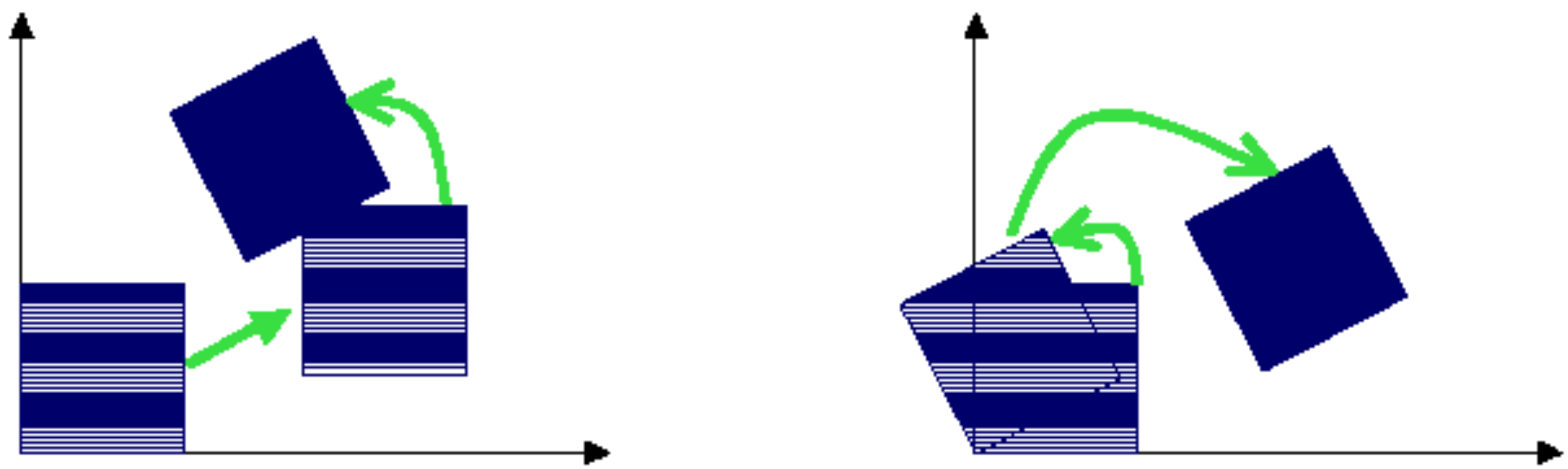
The two properties still apply.

$$M_R^{-1} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$M_R\big|_{\theta=0} = Identity$$

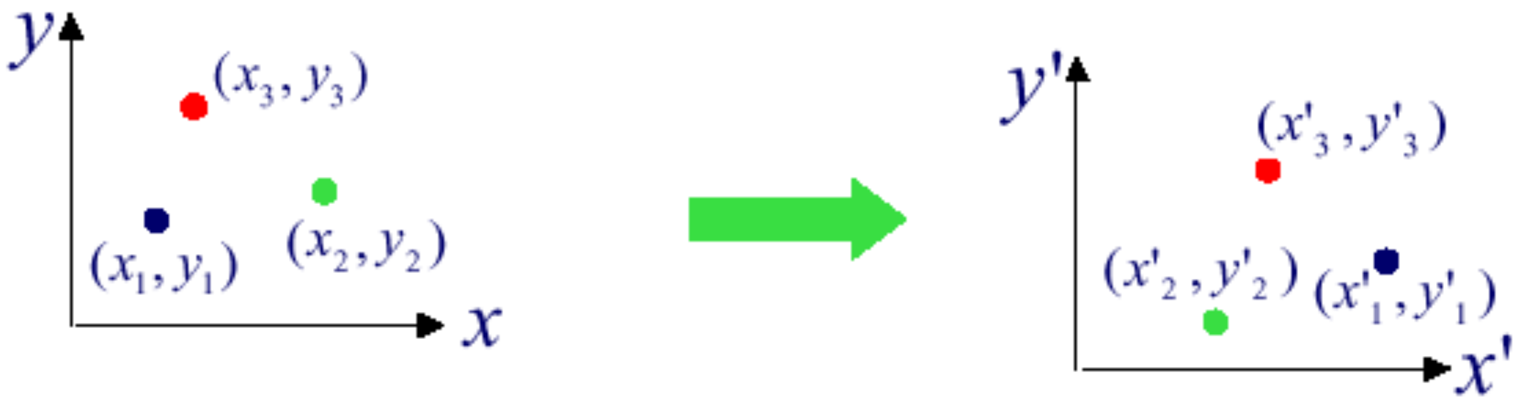
Putting Translation and Rotation Together

- Order matters !!



Affine Transformation

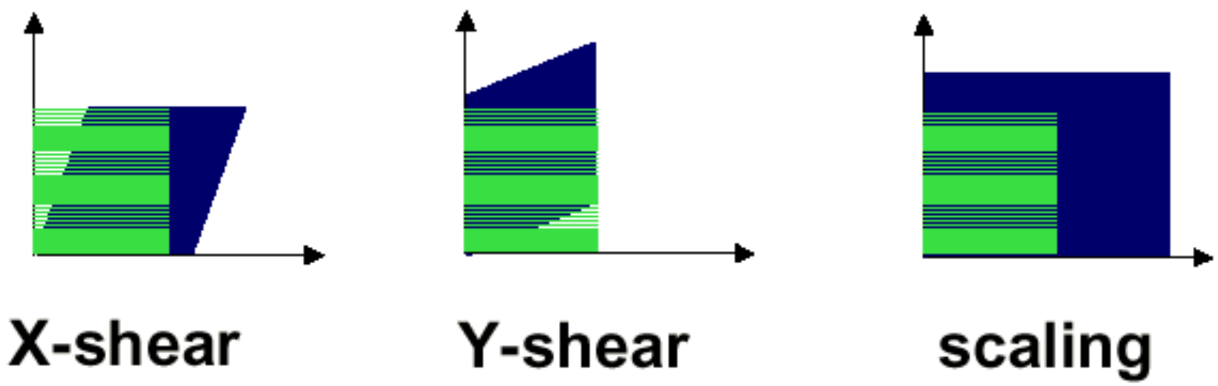
- Property: preserving parallel lines
- The coordinates of three corresponding points uniquely determine any Affine Transform!!



Affine Transformations

- Translation
- Rotation
- Scaling
- Shearing

$$M = \begin{bmatrix} m_{00} & m_{01} & 0 \\ m_{10} & m_{11} & 0 \\ m_{20} & m_{21} & 1 \end{bmatrix}^{\mathbf{T}}$$



How to determine an Affine 2D Transformation?

- We set up 6 linear equations in terms of our 6 unknowns. In this case, we know the 2D coordinates before and after the mapping, and we wish to solve for the 6 entries in the affine transform matrix

$$\underbrace{\begin{bmatrix} x_1' \\ y_1' \\ x_2' \\ y_2' \\ x_3' \\ y_3' \end{bmatrix}}_{x'} = \underbrace{\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_2 & y_2 & 1 \\ x_3 & y_3 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_3 & y_3 & 1 \end{bmatrix}}_{X} \underbrace{\begin{bmatrix} m_{00} \\ m_{01} \\ m_{10} \\ m_{11} \\ m_{20} \\ m_{21} \end{bmatrix}}_{m}$$

Affine Transformation in 3D

- Translation

$$\begin{pmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Rotate

$$\begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Scale

$$\begin{pmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Shear

$$\begin{pmatrix} 1 & 0 & SH_x & 0 \\ 0 & 1 & SH_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

More Rotation

- Which axis of rotation is this?

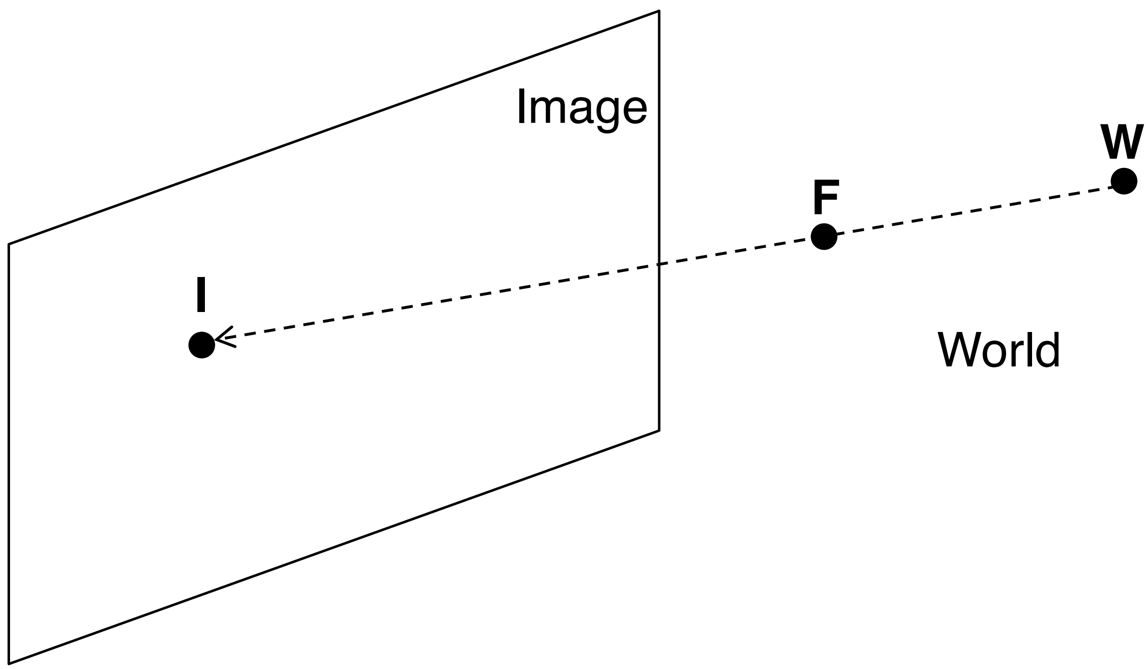
$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_y = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R_z = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Viewing

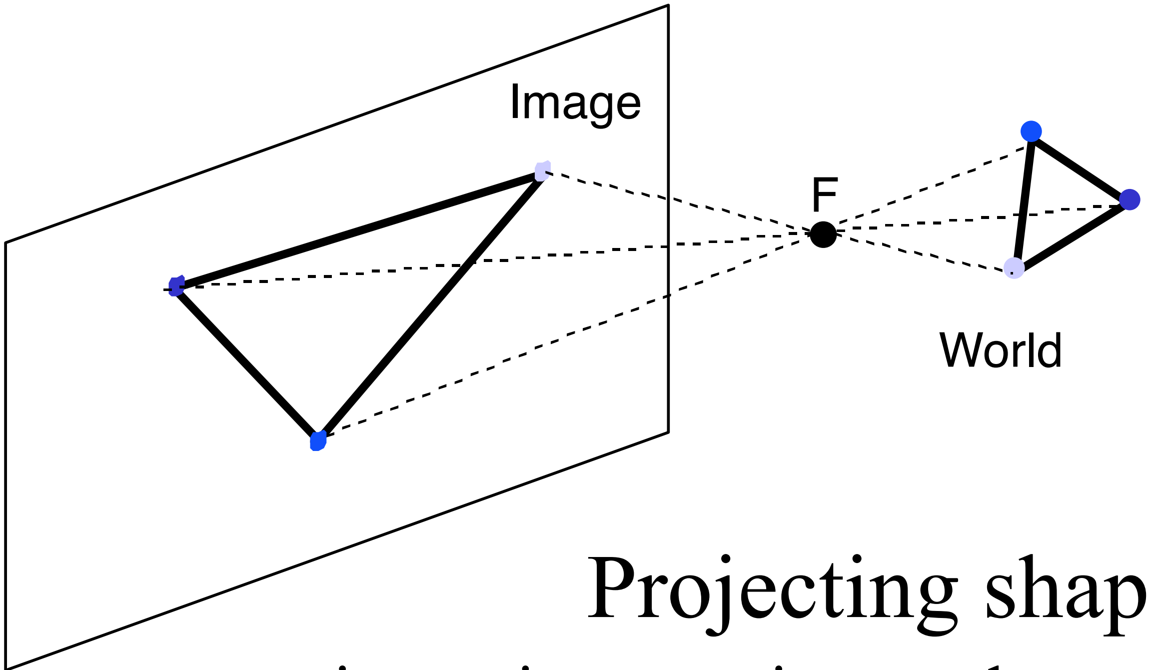
- Object space to World space: affine transformation
- World space to Eye space: how?
- Eye space to Clipping space involves projection and viewing frustum

Perspective Projection



- Projection point sees anything on ray through pinhole *F*
- Point *W* projects along the ray through *F* to appear at *I* (intersection of WF with image plane)

Image Formation

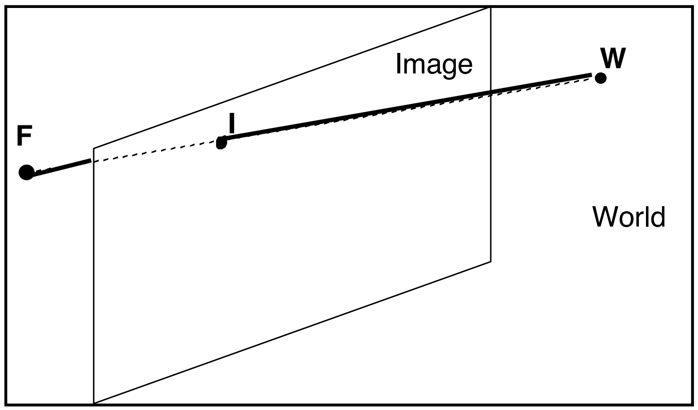


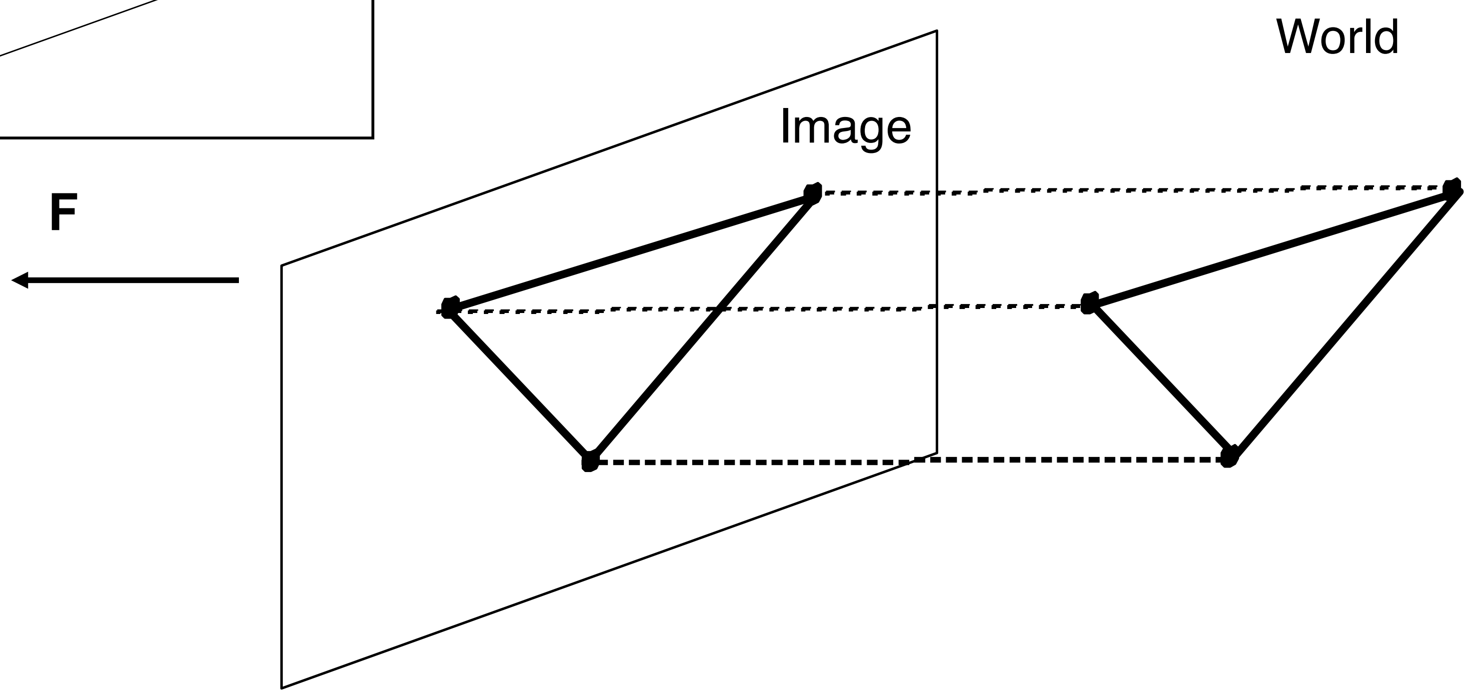
Projecting shapes

- project points onto image plane
- lines are projected by projecting its end points only

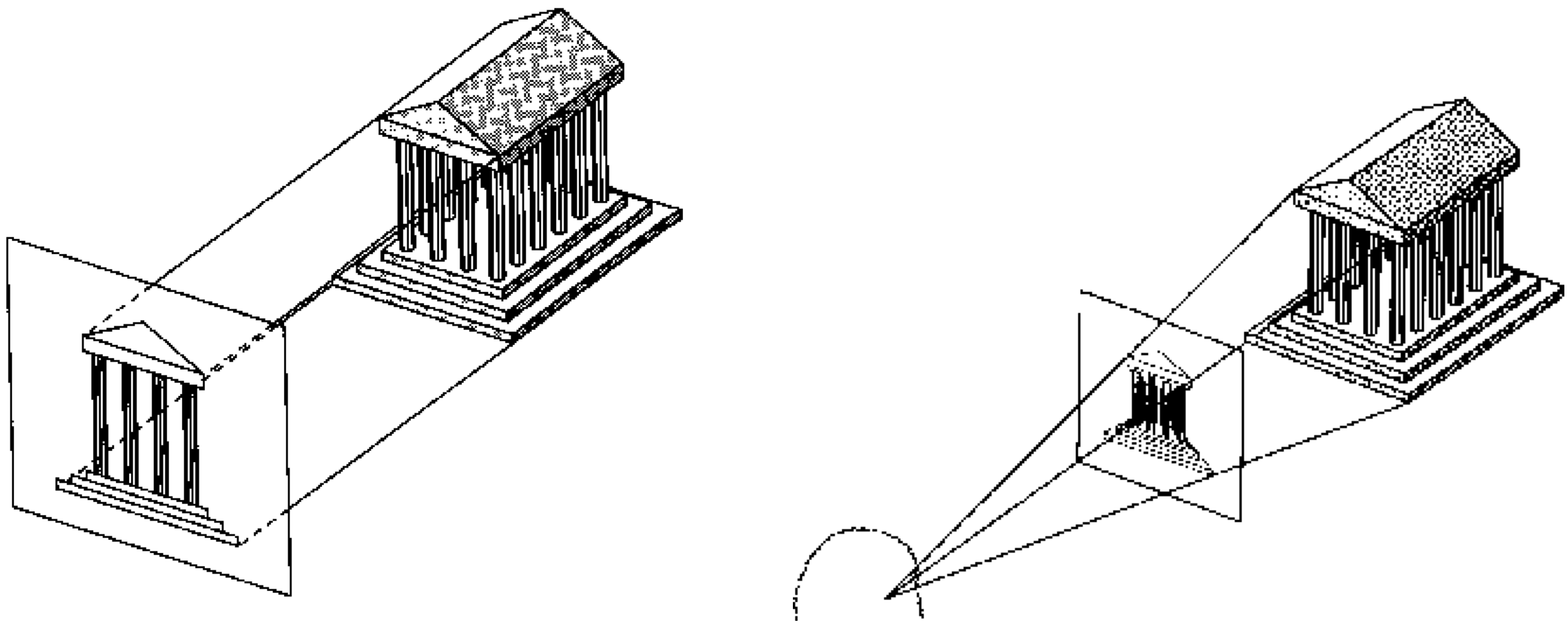
Orthographic Projection

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



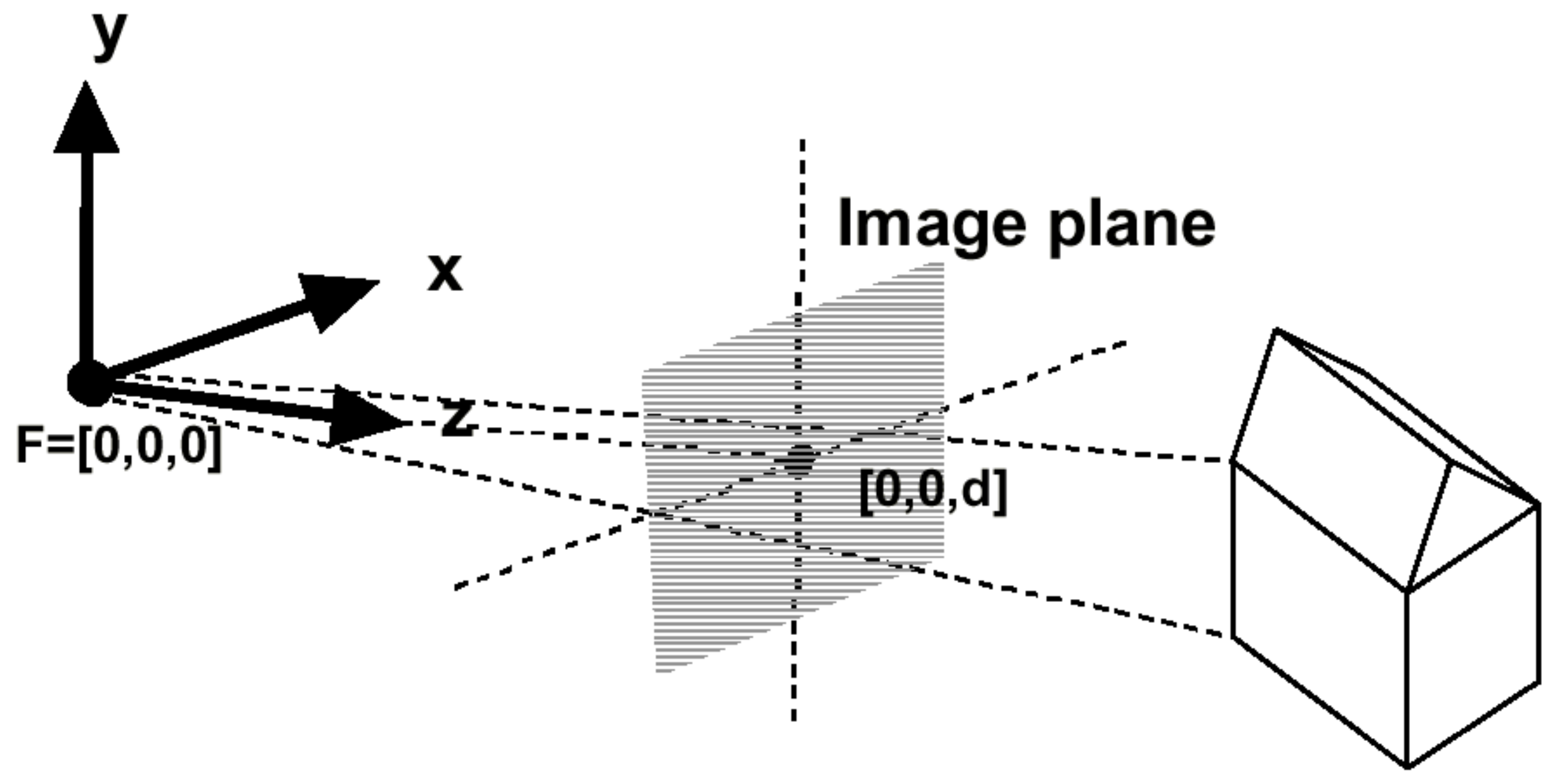


Comparison

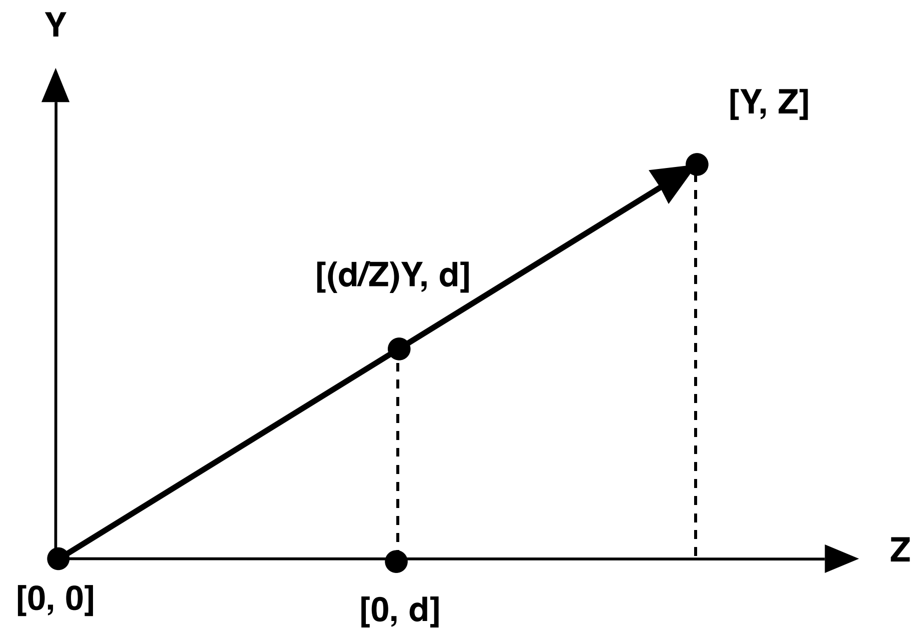


Simple Perspective Camera

- 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 for historical reason



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]

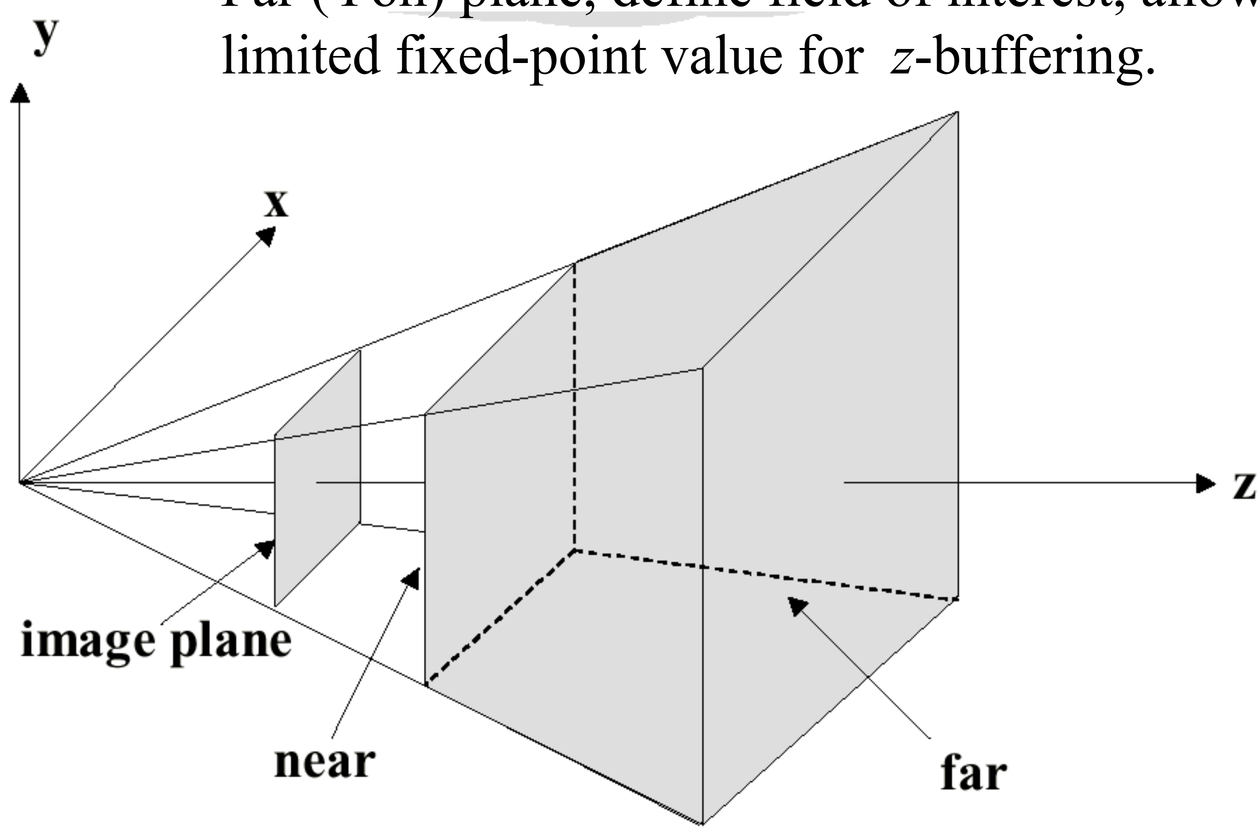
$$\begin{bmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & d & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} dx & dy & dz & z \end{bmatrix} \Rightarrow \begin{bmatrix} \frac{d}{z}x & \frac{d}{z}y & \mathrm{d} \end{bmatrix}$$

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

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

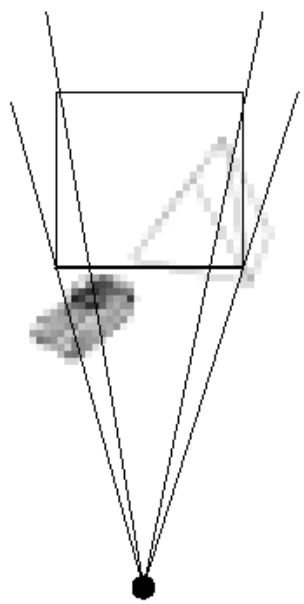
View Volume

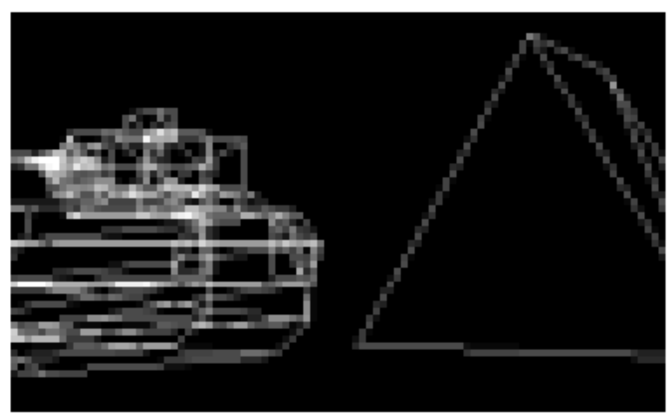
- Defines visible region of space, pyramid edges are clipping planes
- *Frustum* :truncated pyramid with near and far clipping planes
  - Near (Hither) plane ? Don't care about behind the camera
  - Far (Yon) plane, define field of interest, allows *z* to be scaled to a limited fixed-point value for *z*-buffering.



Difficulty

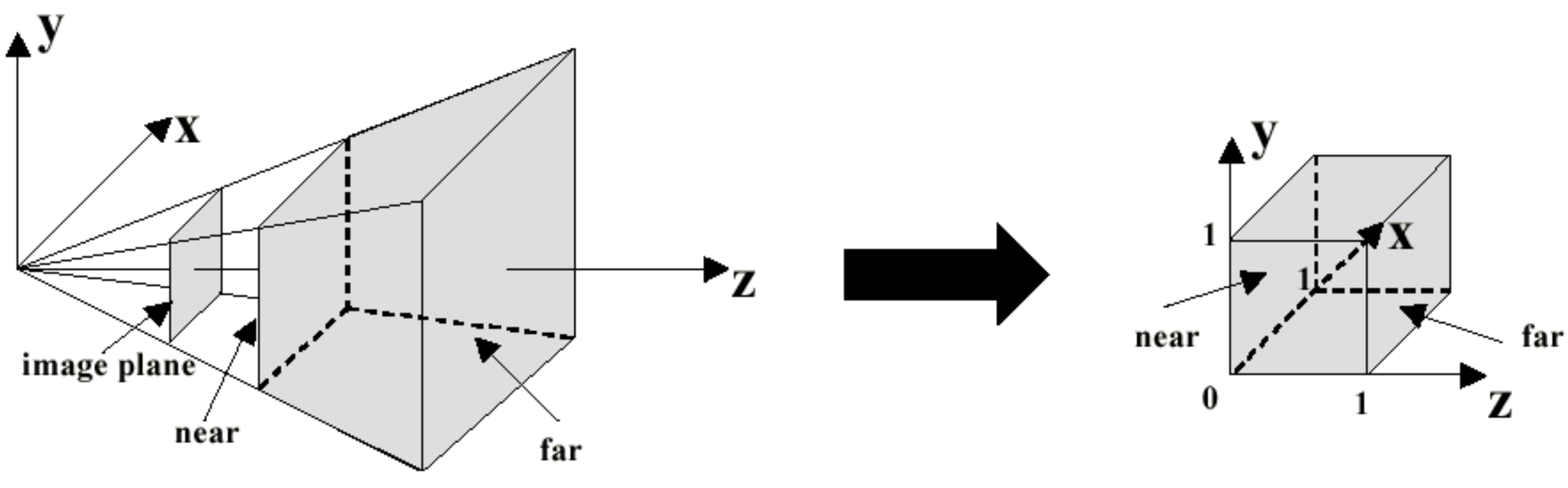
- It is difficult to do clipping directly in the viewing frustum





Canonical View Volume

- Normalize the viewing frustum to a cube, canonical view volume
- **`Converts perspective frustum to orthographic frustum – perspective transformation`**



Perspective Transform

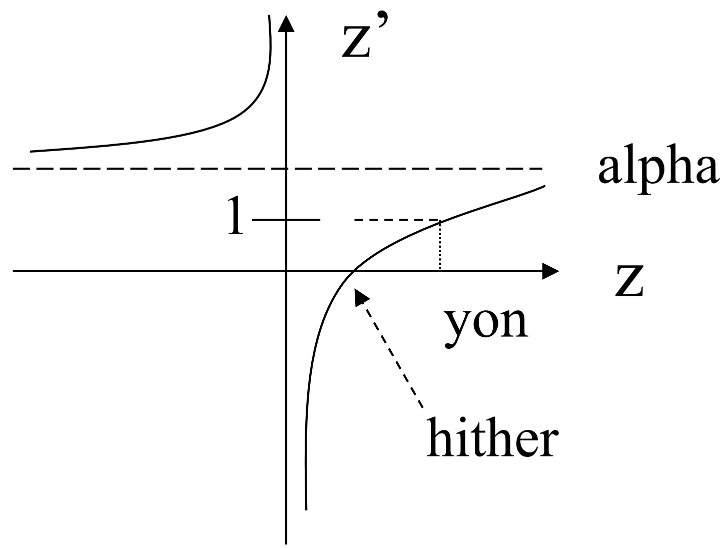
- The equations

$$
\begin{cases}
x \leftarrow \dfrac{x}{z}\dfrac{d}{s} \\[2ex]
y \leftarrow \dfrac{y}{z}\dfrac{d}{s} \\[2ex]
z \leftarrow \alpha + \dfrac{\beta}{z}
\end{cases}
$$

alpha = yon/(yon-hither)

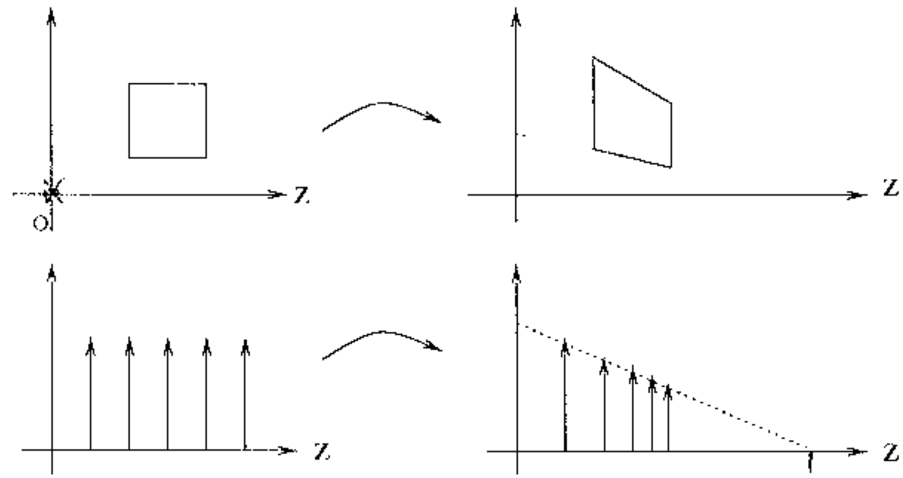
beta = yon*hither/(hither - yon)

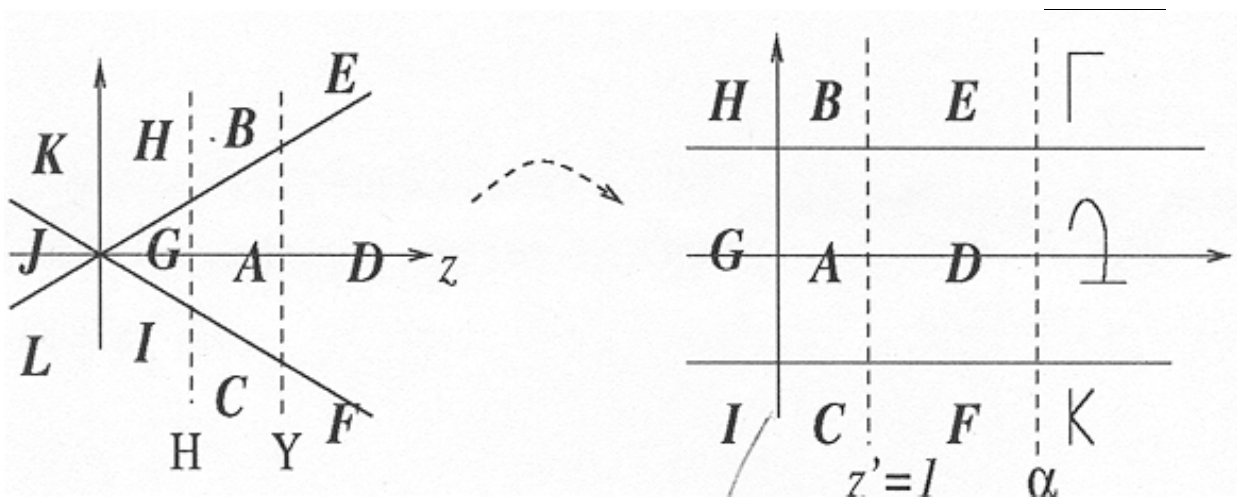
s: size of window on the image plane



About Perspective Transform

- Some properties





About Perspective Transform

- Clipping can be performed against the rectilinear box
- Planarity and linearity are preserved
- Angles and distances are not preserved
- Side effects: objects behind the observer are mapped to the front. Do we care?

Perspective + Projection Matrix

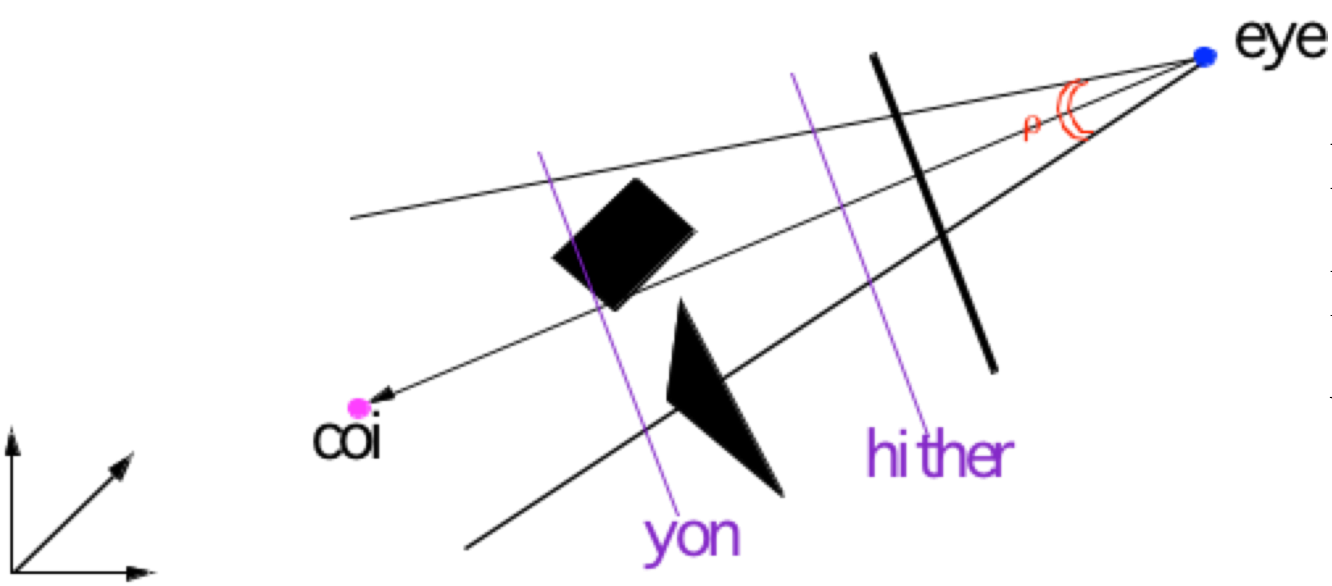
- AR: aspect ratio correction, ResX/ResY
- s= ResX,
- Theta: half view angle, tan(theta) = s/d

$$P = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & AR & 0 & 0 \\ 0 & 0 & \alpha \tan\theta & \tan\theta \\ 0 & 0 & \beta \tan\theta & 0 \end{pmatrix}.$$

Camera Control and Viewing

Focal length (d), image size/shape and clipping planes included in perspective transformation

▪ ρ	Angle or Field of view (FOV)
▪ ***AR***	Aspect Ratio of view-port
▪ ***Hither, Yon***	Nearest and farthest vision limits (WS).



Lookat - coi

Lookfrom - eye

View angle - FOV

Complete Perspective

- Specify near and far clipping planes - transform *z* between *znear* and *zfar* on to a fixed range
- Specify field-of-view (fov) angle
- OpenGL's **glFrustum** and **gluPerspective** do these

More Viewing Parameters

Camera, Eye or Observer:

lookfrom: location of focal point or camera

lookat: point to be centered in image

Camera orientation about the ***lookat-lookfrom*** axis

vup*:* a vector that is pointing straight up in the image. This is like an orientation.

Implementation … Full Blown

- Translate by *-lookfrom*, bring focal point to origin
- Rotate *lookat-lookfrom* to the *z*-axis with matrix R:
  - **v** = (*lookat-lookfrom*) (normalized) and **z** = [0,0,1]
  - rotation axis: **a** = (**v**x**z**)/|**v**x**z**|
  - rotation angle: cosθ = **a**•**z** and sinθ = |**r**x**z**|

- OpenGL: glRotate(θ, a_x, a_y, a_z)
- Rotate about *z*-axis to get *vup* parallel to the y-axis

Viewport mapping

- Change from the image coordinate system (x,y,z) to the screen coordinate system (X,Y).
- Screen coordinates are always non-negative integers.
- Let (v_r, v_t) be the upper-right corner and (v_l, v_b) be the lower-left corner.
- $X = x * (v_r - v_l)/2 + (v_r + v_l)/2$
- $Y = y * (v_t - v_b)/2 + (v_t + v_b)/2$

True Or False

- In perspective transformation parallelism is not preserved.
  - Parallel lines converge
  - Object size is reduced by increasing distance from center of projection
  - Non-uniform foreshortening of lines in the object as a function of orientation and distance from center of projection
  - Aid the depth perception of human vision, but shape is not preserved

True Or False

- Affine transformation is a combination of linear transformations
- The last column/row in the general 4x4 affine transformation matrix is $[0\ 0\ 0\ 1]^T$.
- After affine transform, the homogeneous coordinate ***w*** maintains unity.