CS559: Computer Graphics

Lecture 9: 3D Transformation and Projection

Li Zhang

Spring 2010

Most slides borrowed from [Yungyu Chuang]

Last time: Image morphing

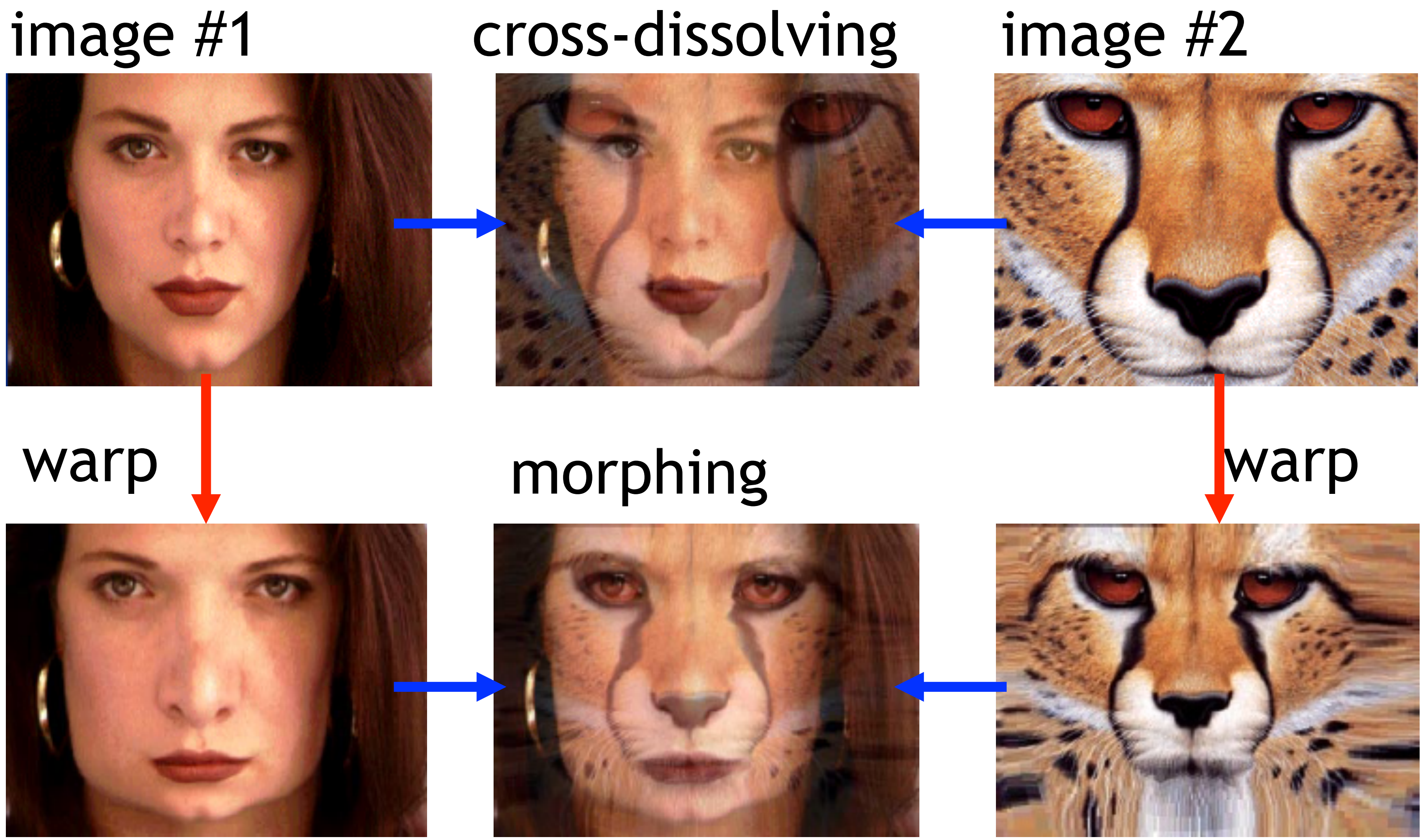
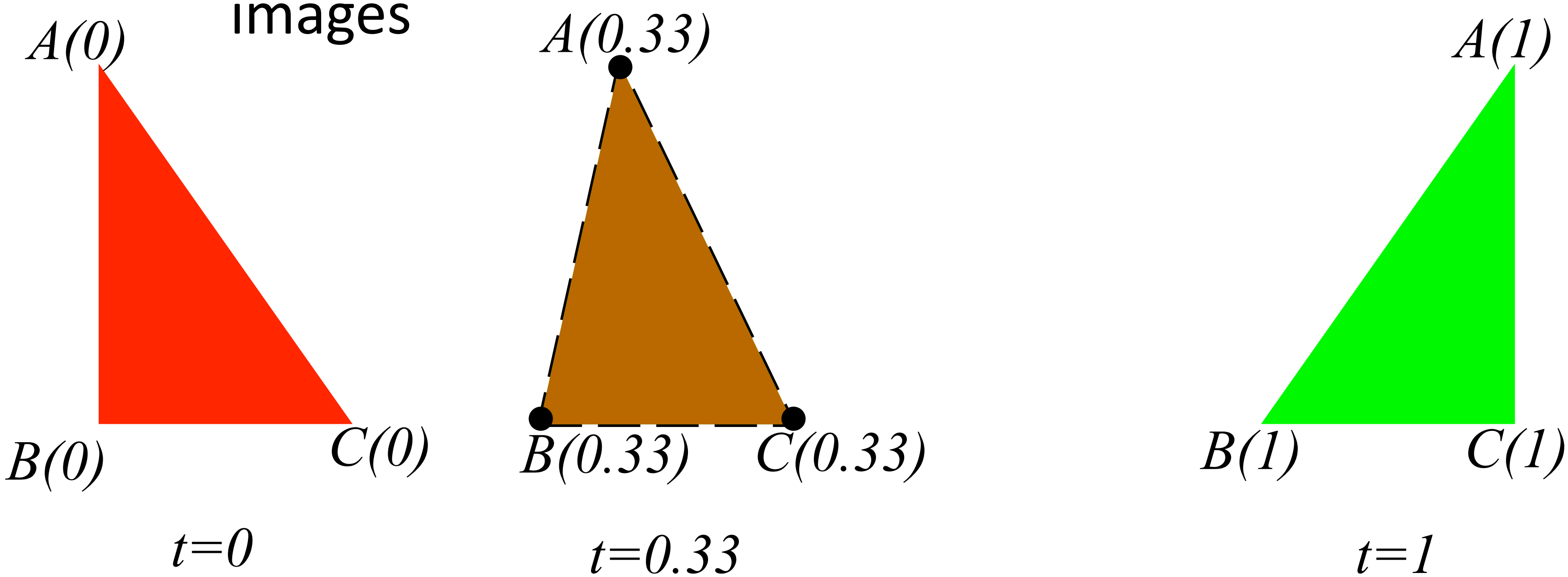


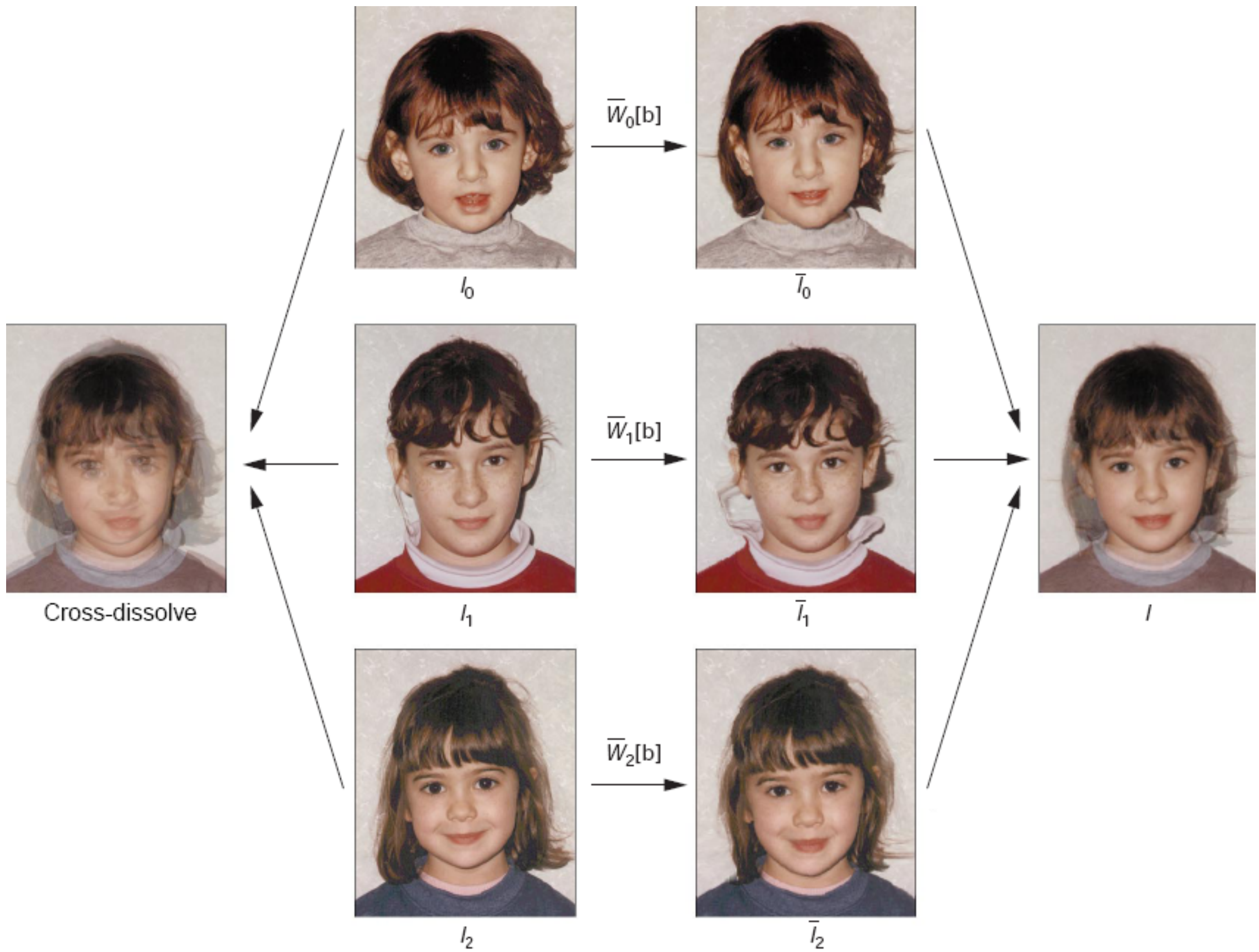
Image morphing

create a morphing sequence: for each time t

1. Create an intermediate warping field (by interpolation)
2. Warp both images towards it
3. Cross-dissolve the colors in the newly warped images



Multi-source morphing



More complex morph

- Triangular Mesh

Homogeneous Directions

- Translation does not affect directions!
- Homogeneous coordinates give us a very clean way of handling this
- The direction *(x,y)* becomes the homogeneous direction *(x,y,0)*

$$\begin{bmatrix} M & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{p} \\ 1 \end{bmatrix} = \begin{bmatrix} M\mathbf{p} + \mathbf{t} \\ 1 \end{bmatrix} \quad \begin{bmatrix} M & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{v} \\ 0 \end{bmatrix} = \begin{bmatrix} M\mathbf{v} \\ 0 \end{bmatrix}$$

- M can be any linear transformation: rotation, scaling, etc
  - Uniform scaling changes the length of the vector, but not the direction
  - How to represent this equivalence

Homogeneous Coordinates

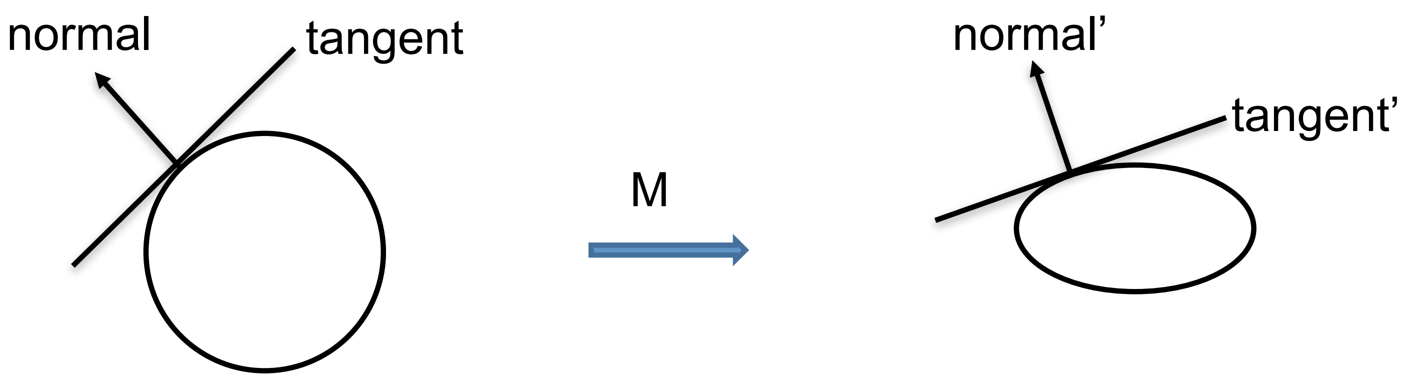
- In general, homogeneous coordinates (x, y, w)

$$\begin{bmatrix} x/w \\ y/w \end{bmatrix} \Leftarrow \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \end{bmatrix} \Rightarrow \begin{bmatrix} wx \\ wy \\ w \end{bmatrix} \text{for any w} \neq 0\text{, usually w} = 1$$

- How to interpret the case for w = 0?
  - Point at infinity: directional vector
- (1,2,1) ⇔ (2,4,2)
- (2,3,0) ⇔ (6,9,0)

Transforming normal vectors



$$\mathbf{n}^{T}\mathbf{t} = 0$$

$$\mathbf{t}' = \boxed{\mathbf{M}\mathbf{t}}$$

$$\mathbf{n}'^{T}\,\mathbf{t}' = 0$$

$$(\mathbf{n}^{T}\mathbf{M}^{-1})(\mathbf{M}\mathbf{t}) = 0$$

$$\mathbf{n}' = (\mathbf{n}^{T}\mathbf{M}^{-1})^{T} = \boxed{(\mathbf{M}^{-1})^{T}\mathbf{n}}$$

If M is a rotation,

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

3D Transformations

- Homogeneous coordinates: *(x,y,z)=(wx,wy,wz,w)*

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \Rightarrow \begin{bmatrix} wx \\ wy \\ wz \\ w \end{bmatrix} \text{for any w} \neq 0, \text{usually w} = 1$$

$$\begin{bmatrix} x/w \\ y/w \\ z/w \end{bmatrix} \Leftarrow \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

- Transformations are now represented as 4x4 matrices

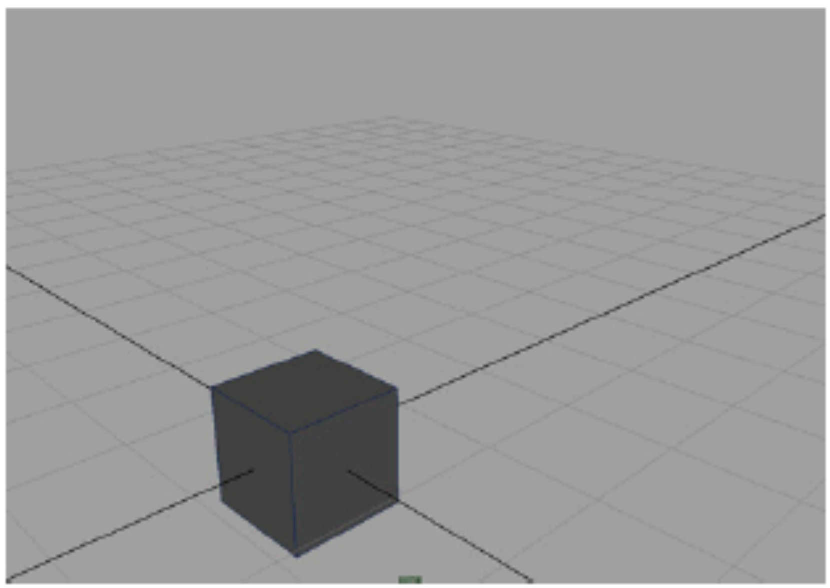
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

3D Affine Transform

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c & t_x \\ d & e & f & t_y \\ g & h & i & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

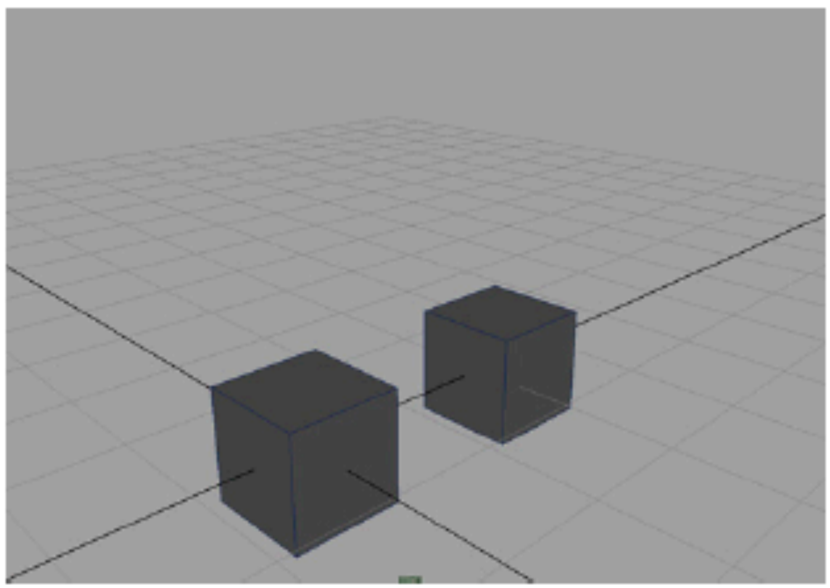
Translation

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



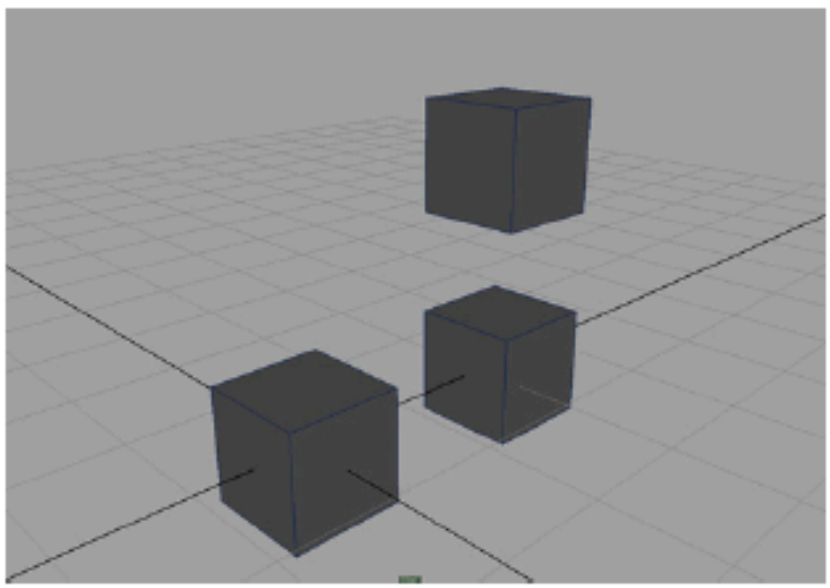
Translation

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



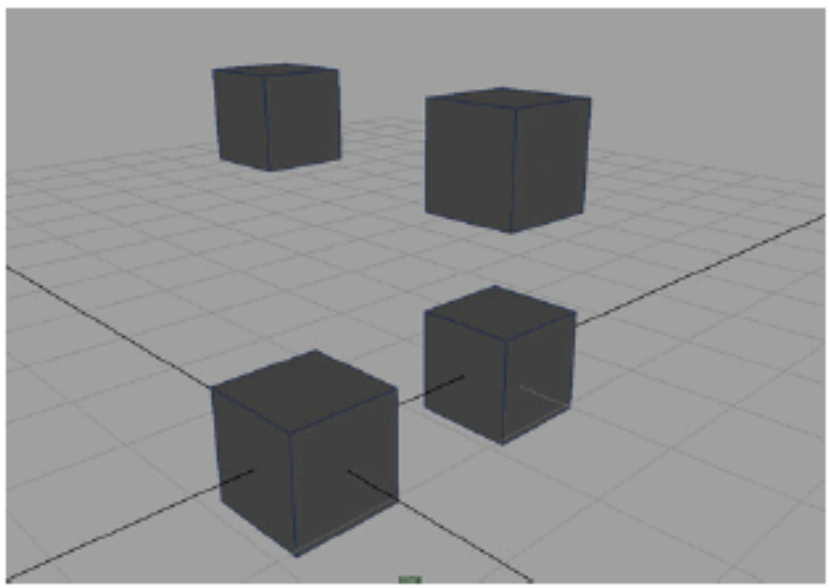
Translation

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



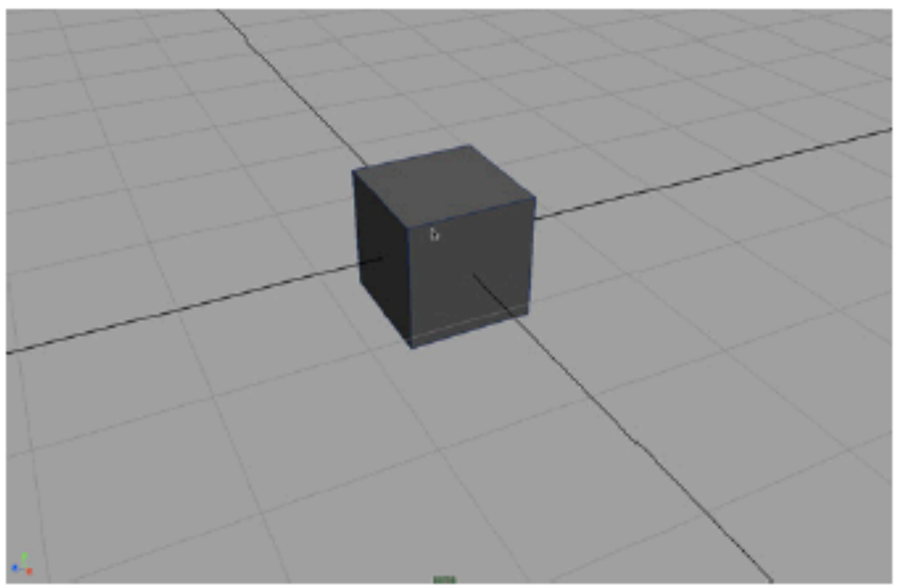
Translation

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



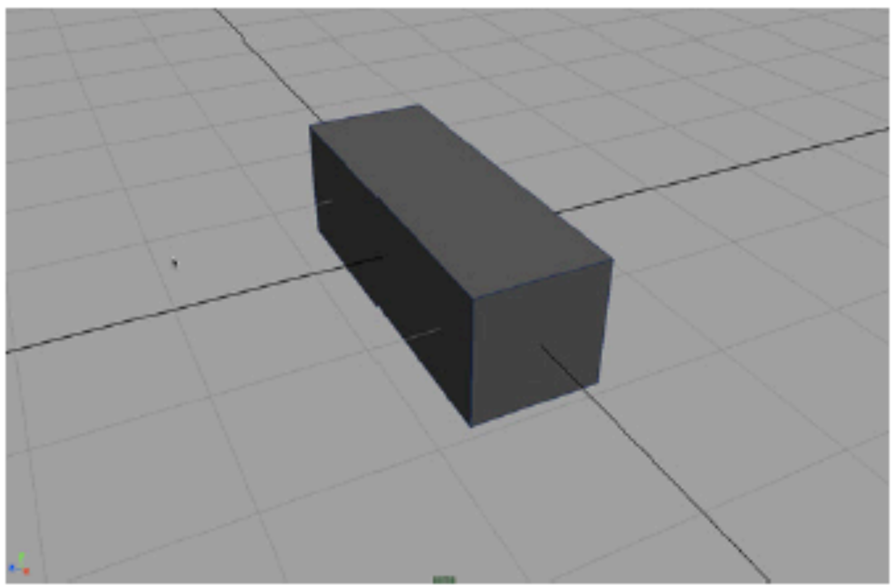
Scaling

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



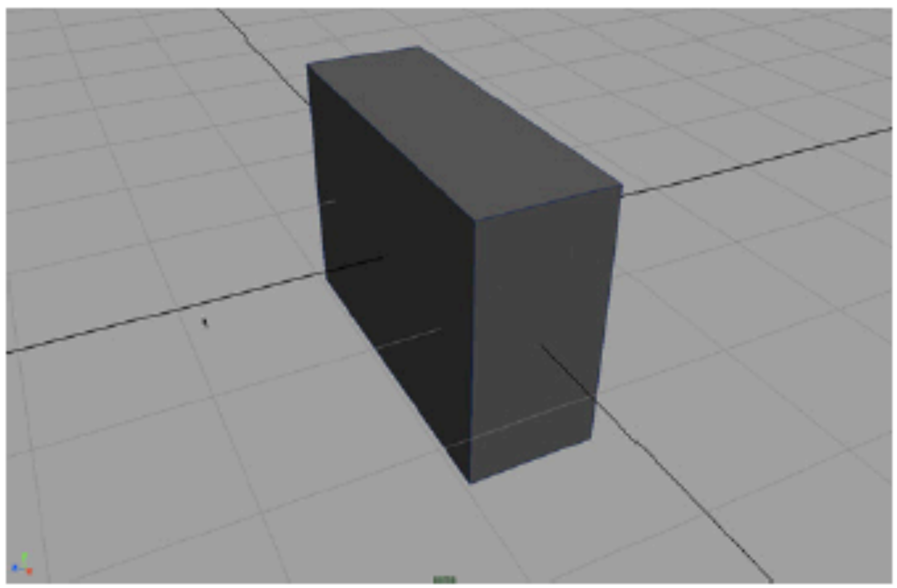
Scaling

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



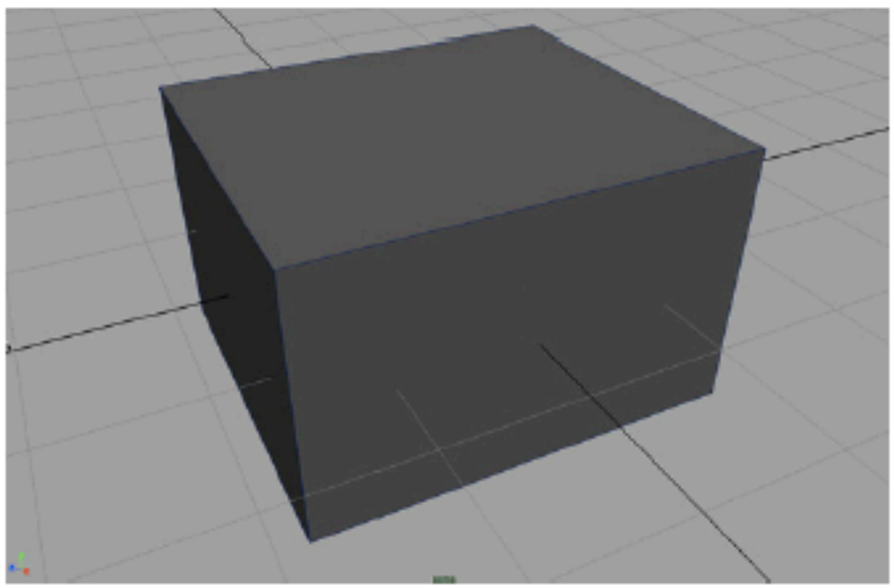
Scaling

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



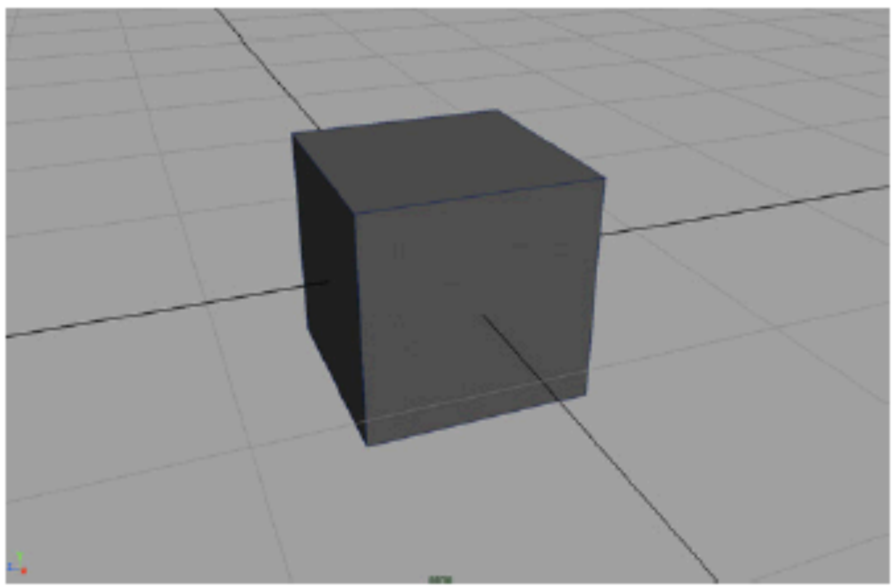
Scaling

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



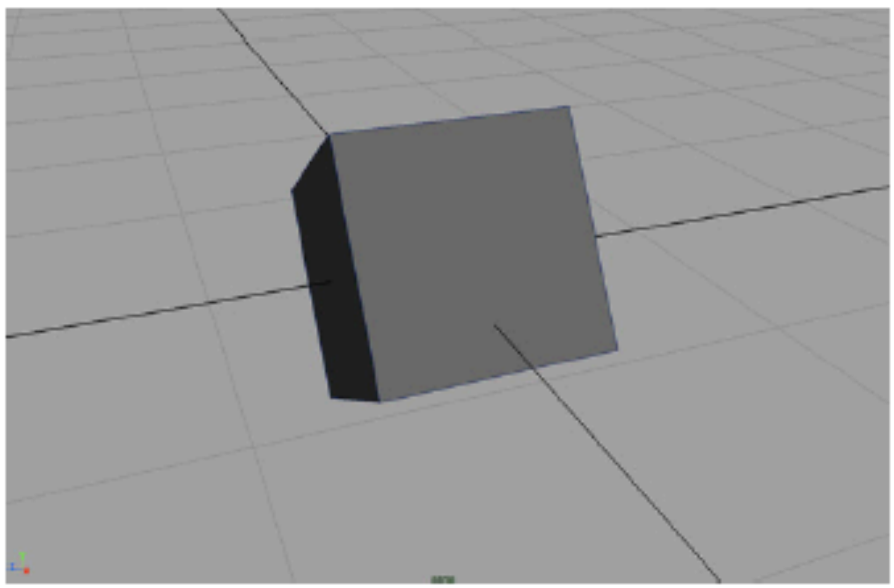
Rotation about z axis

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



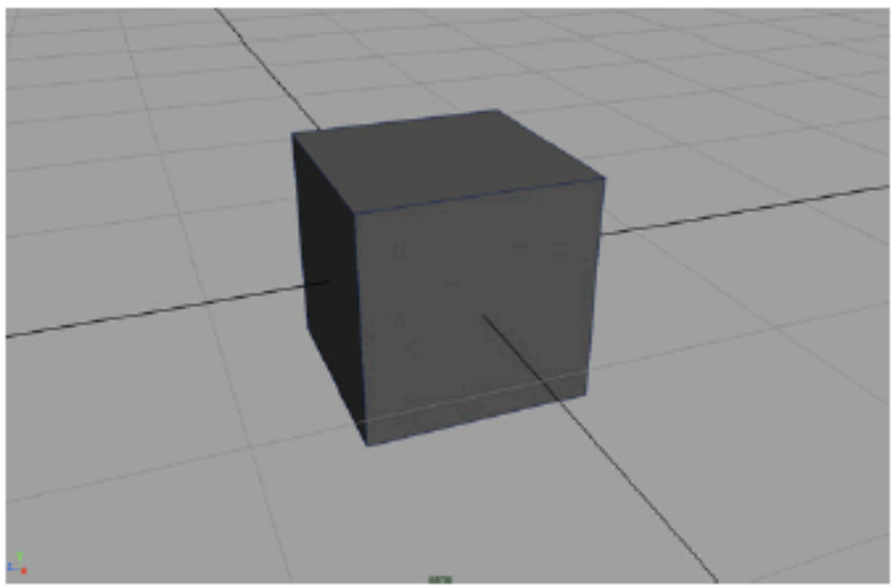
Rotation about z axis

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



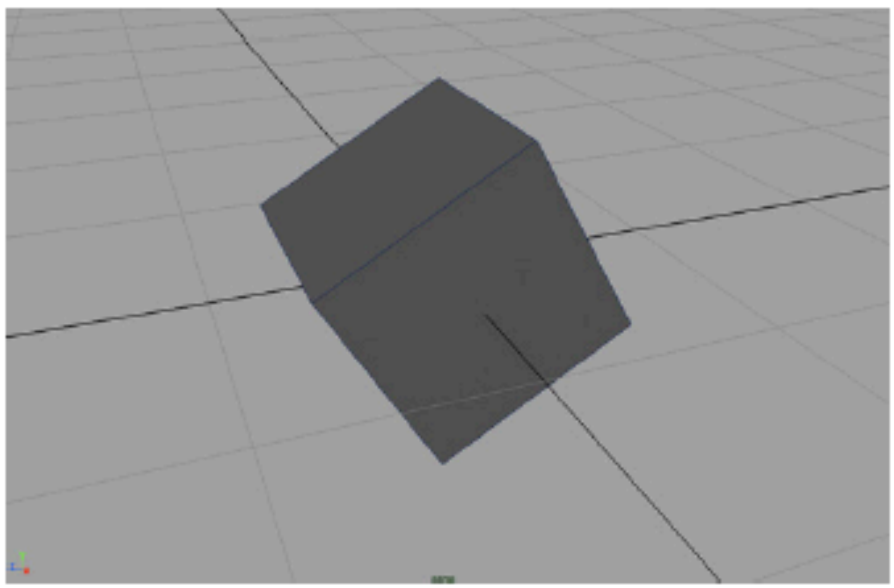
Rotation about x axis

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



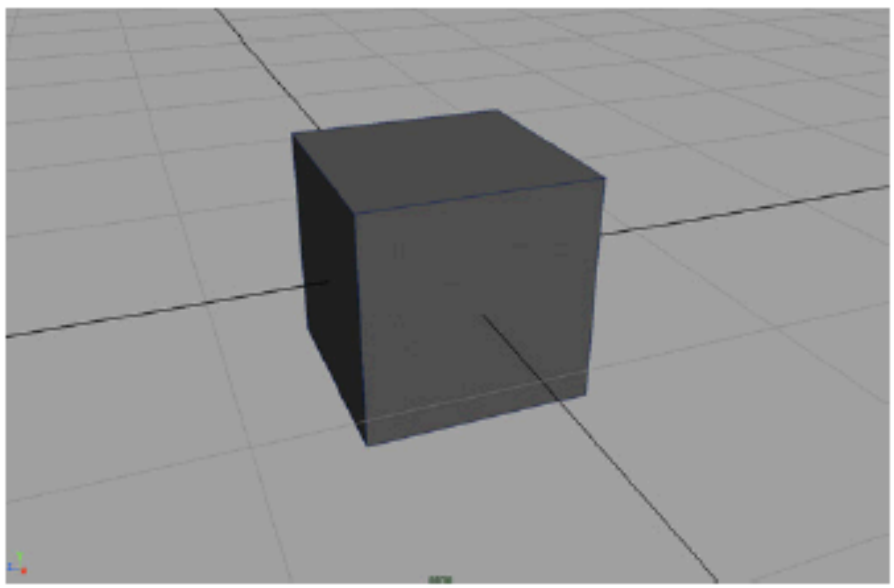
Rotation about x axis

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



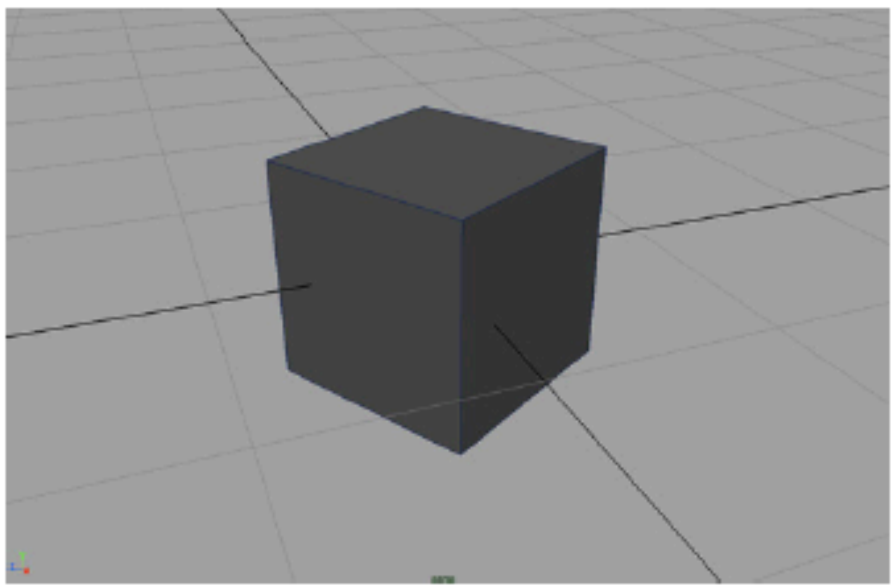
Rotation about y axis

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



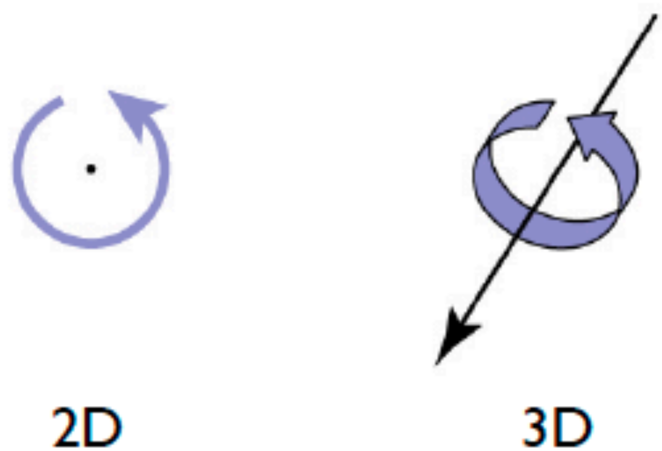
Rotation about y axis

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



General rotations

- A rotation in 2D is around a point
- A rotation in 3D is around an axis
  - so 3D rotation is w.r.t a line, not just a point
  - there are many more 3D rotations than 2D
    - a 3D space around a given point, not just 1D



3D Rotation

- Rotation in 3D is about an *axis* in 3D space passing through the origin

3D Rotation

- Rotation in 3D is about an *axis* in 3D space passing through the origin
- Using a matrix representation, any matrix with an *orthonormal* top-left 3x3 sub-matrix is a rotation
  - Rows/columns are mutually orthogonal (0 dot product)

$$R = \begin{bmatrix} | & | & | & 0 \\ \mathbf{r}_1 & \mathbf{r}_2 & \mathbf{r}_3 & 0 \\ | & | & | & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ then } \mathbf{r}_1 \bullet \mathbf{r}_2 = 0, \mathbf{r}_1 \bullet \mathbf{r}_3 = 0, \mathbf{r}_2 \bullet \mathbf{r}_3 = 0, \mathbf{r}_1 \bullet \mathbf{r}_1 = 1, \mathbf{r}_2 \bullet \mathbf{r}_2 = 1, \mathbf{r}_3 \bullet \mathbf{r}_3 = 1.$$

3D Rotation

- Rotation in 3D is about an *axis* in 3D space passing through the origin
- Using a matrix representation, any matrix with an *orthonormal* top-left 3x3 sub-matrix is a rotation
  - Rows/columns are mutually orthogonal (0 dot product)
  - Determinant is 1
  - Implies columns are also orthogonal, and that the transpose is equal to the inverse

$$R = \begin{bmatrix} | & | & | & 0 \\ \mathbf{r}_1 & \mathbf{r}_2 & \mathbf{r}_3 & 0 \\ | & | & | & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ then } \mathbf{r}_1 \bullet \mathbf{r}_2 = 0, \mathbf{r}_1 \bullet \mathbf{r}_3 = 0, \mathbf{r}_2 \bullet \mathbf{r}_3 = 0, \mathbf{r}_1 \bullet \mathbf{r}_1 = 1, \mathbf{r}_2 \bullet \mathbf{r}_2 = 1, \mathbf{r}_3 \bullet \mathbf{r}_3 = 1.$$

Specifying a rotation matrix

- *Euler angles:* Specify how much to rotate about X, then how much about Y, then how much about Z
  - Hard to think about, and hard to compose

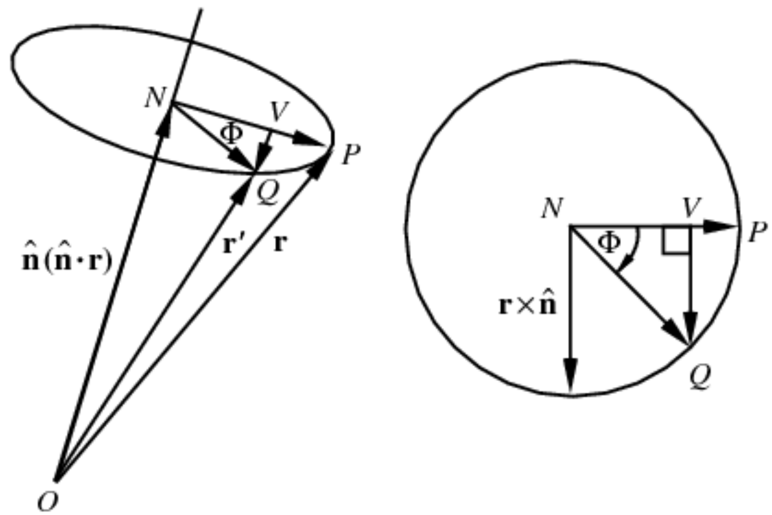
$$[\mathbf{R}] = \begin{bmatrix} \cos\gamma & \sin\gamma & 0 \\ -\sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\beta & \sin\beta \\ 0 & -\sin\beta & \cos\beta \end{bmatrix} \begin{bmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Alternative Representations

- Specify the axis and the angle (OpenGL method)
  - Hard to compose multiple rotations
  - Axis ω, angle θ

Alternative Representations

- Specify the axis and the angle (OpenGL method)
  - Hard to compose multiple rotations



A rotation by an angle $\theta \in \mathbb{R}$ around axis specified by the unit vector $\hat{\omega} = (\omega_x, \omega_y, \omega_z) \in \mathbb{R}^3$ is given by $I + \tilde{\omega}\sin\theta + \tilde{\omega}^2 (1-\cos\theta)$

$$\tilde{\omega} = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix}.$$

http://mathworld.wolfram.com/RodriguesRotationFormula.html

Non-Commutativity

- Not Commutative (unlike in 2D)!!
- Rotate by x, then y is not same as y then x
- Order of applying rotations does matter
- Follows from matrix multiplication not commutative
  - R1 * R2 is not the same as R2 * R1

Other Rotation Issues

- Rotation is about an axis at the origin
  - For rotation about an arbitrary axis, use the same trick as in 2D: Translate the axis to the origin, rotate, and translate back again

Transformation Leftovers

- Shear etc extend naturally from 2D to 3D
- Rotation and Translation are the *rigid-body transformations:*
  - Do not change lengths or angles, so a body does not deform when transformed

Building transforms from points

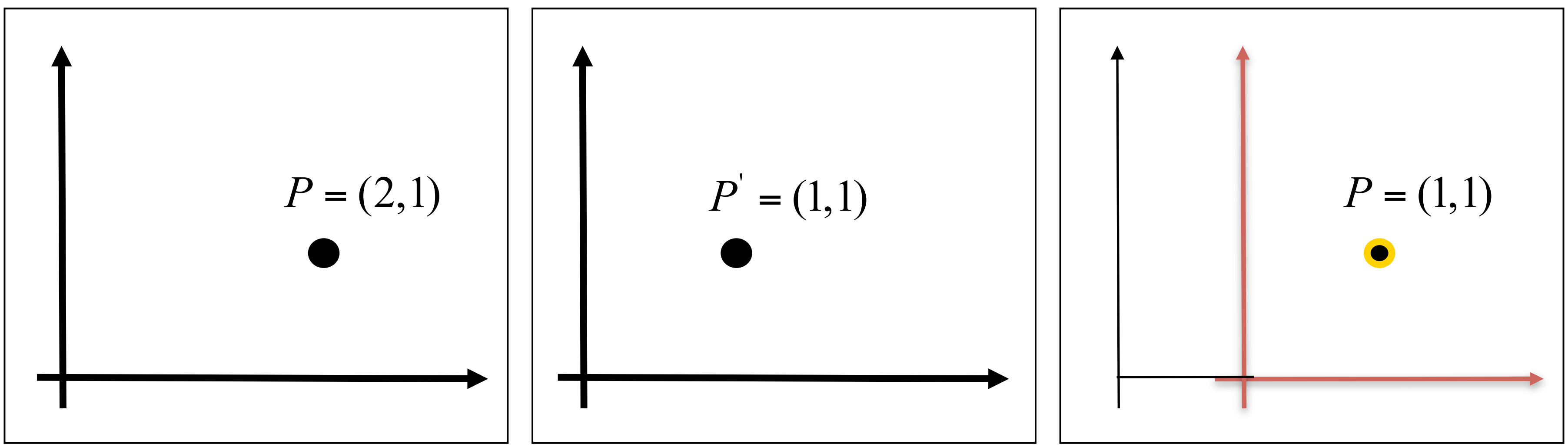
- Recall2D affine transformation has 6 degrees of freedom (DOFs)
  - this is the number of "knobs" we have to set to define one
- Therefore 6 constraints suffice to define the transformation
  - handy kind of constraint: point **p** maps to point **q** (2 constraints at once)
  - three point constraints add up to constrain all 6 DOFs (i.e. can map any triangle to any other triangle)

Building transforms from points

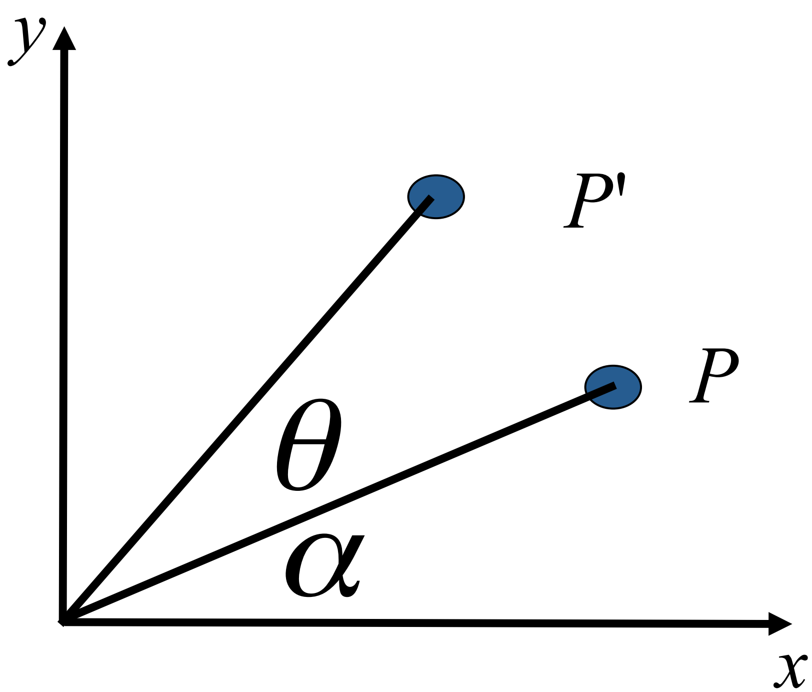
- Recall 2D affine transformation has 6 degrees of freedom (DOFs)
  - this is the number of "knobs" we have to set to define one
- Therefore 6 constraints suffice to define the transformation
  - handy kind of constraint: point **p** maps to point **q** (2 constraints at once)
  - three point constraints add up to constrain all 6 DOFs
    (i.e. can map any triangle to any other triangle)
- 3D affine transformation has 12 degrees of freedom
  - count them by looking at the matrix entries we're allowed to change
- Therefore 12 constraints suffice to define the transformation
  - in 3D, this is 4 point constraints
    (i.e. can map any tetrahedron to any other tetrahedron)

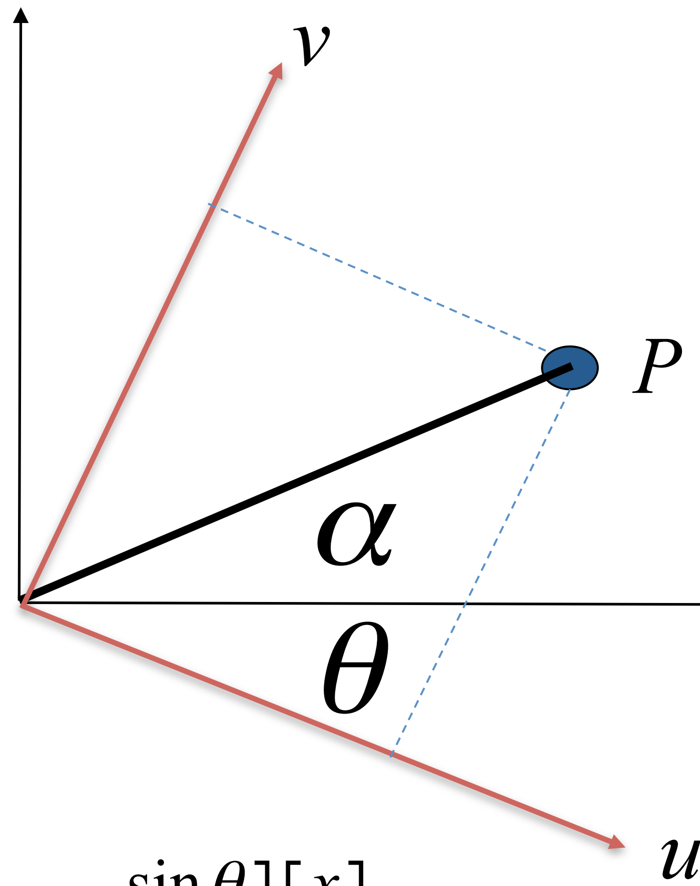
Coordinate Frames

- All of discussion in terms of operating on points
- But can also change coordinate system
- Example, motion means either point moves backward, or coordinate system moves forward



Coordinate Frames: Rotations





$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

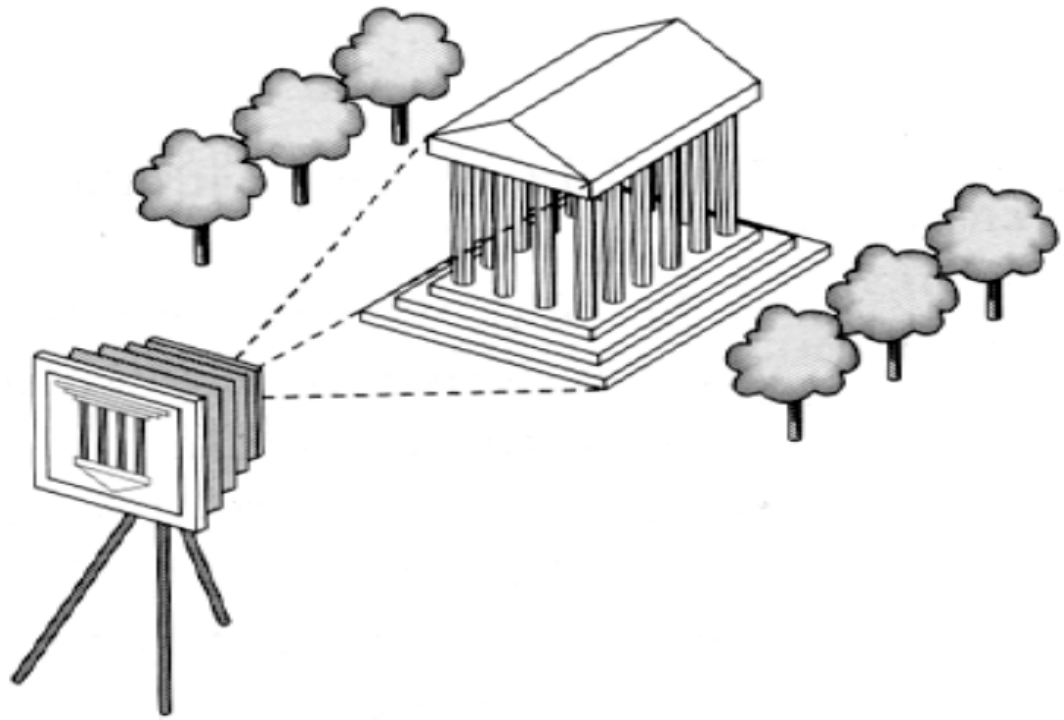
Geometric Interpretation 3D Rotations

- Rows of matrix are 3 unit vectors of new coord frame
- Can construct rotation matrix from 3 orthonormal vectors
- Effectively, projections of point into new coord frame

$$R_{uvw} = \begin{pmatrix} x_u & y_u & z_u \\ x_v & y_v & z_v \\ x_w & y_w & z_w \end{pmatrix}$$

$$Rp = \begin{pmatrix} x_u & y_u & z_u \\ x_v & y_v & z_v \\ x_w & y_w & z_w \end{pmatrix} \begin{pmatrix} x_p \\ y_p \\ z_p \end{pmatrix} = ? \quad \begin{pmatrix} u \bullet p \\ v \bullet p \\ w \bullet p \end{pmatrix}$$

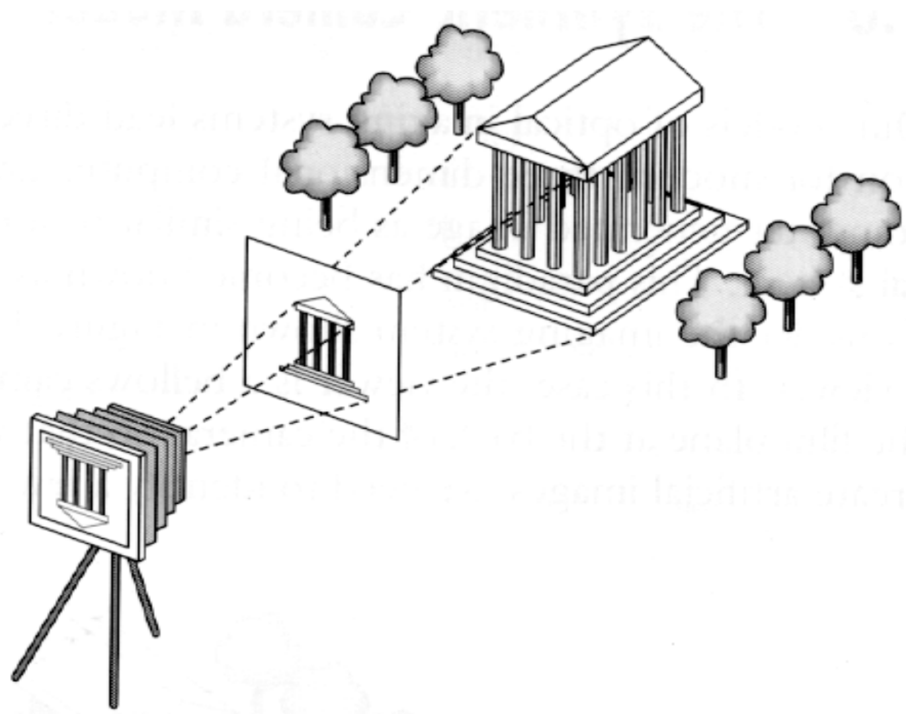
The 3D synthetic camera model



The **synthetic camera model involves two components,** specified *independently:*

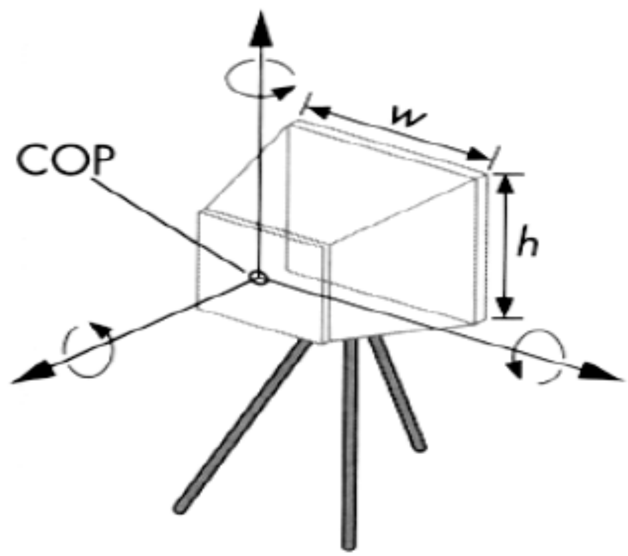
- objects (a.k.a. **geometry)**
- viewer (a.k.a. **camera)**

Imaging with the synthetic camera



The image is rendered onto an **image plane or projection plane (usually** in front of the camera).
Rays emanate from the center of projection (COP) at the center of the lens (or pinhole).
The image of an object point *P is at the intersection of the ray* through *P and the image plane.*

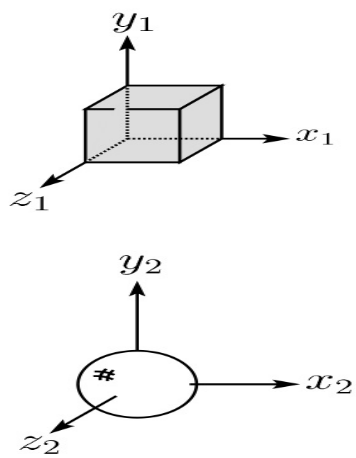
Specifying a viewer



Camera specification requires four kinds of parameters:

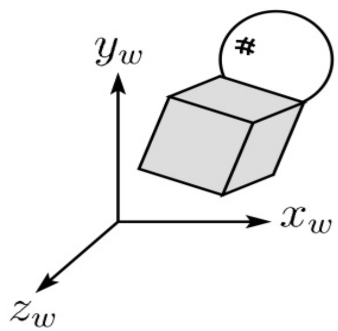
- *Position: the COP.*
- *Orientation: rotations about axes with origin at the COP.*
- *Focal length: determines the size of the image on the film plane, or* the **field of view.**
- *Film plane: its width and height.*

3D Geometry Pipeline



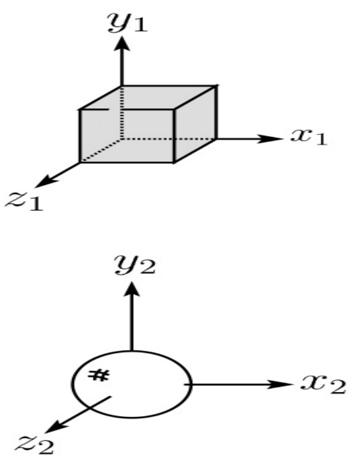
Model Space
(Object Space)

Rotation
Translation
Resizing



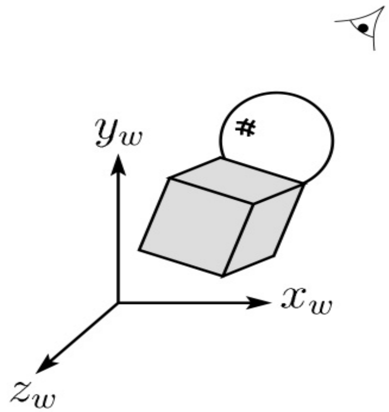
World Space

3D Geometry Pipeline



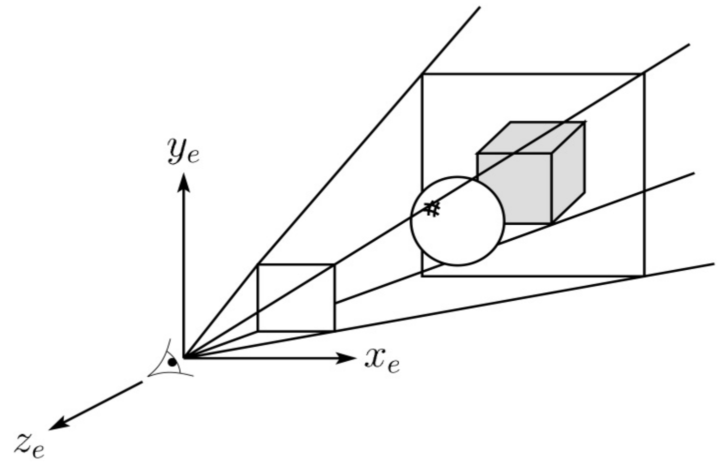
Model Space
(Object Space)

Rotation
Translation
Resizing



World Space

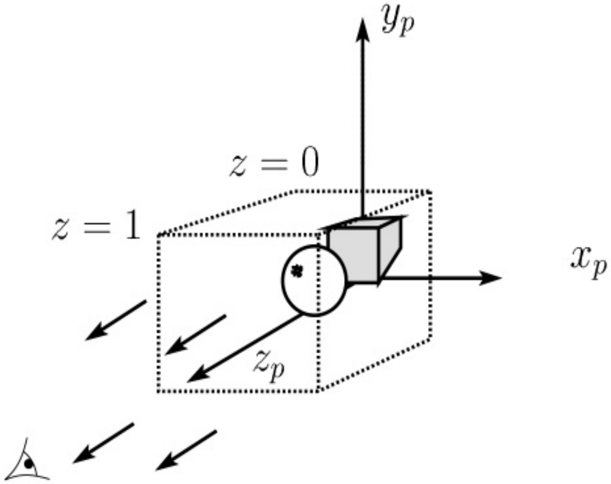
Rotation
Translation

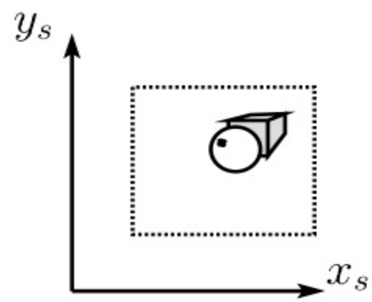


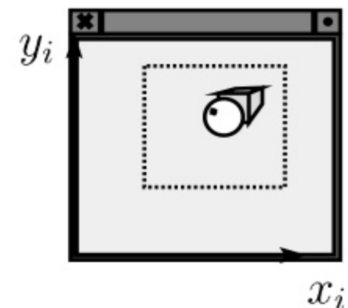
Eye Space
(View Space)

Projective,
Scale
Translation

3D Geometry Pipeline (cont'd)







Normalized Projection Space

Project

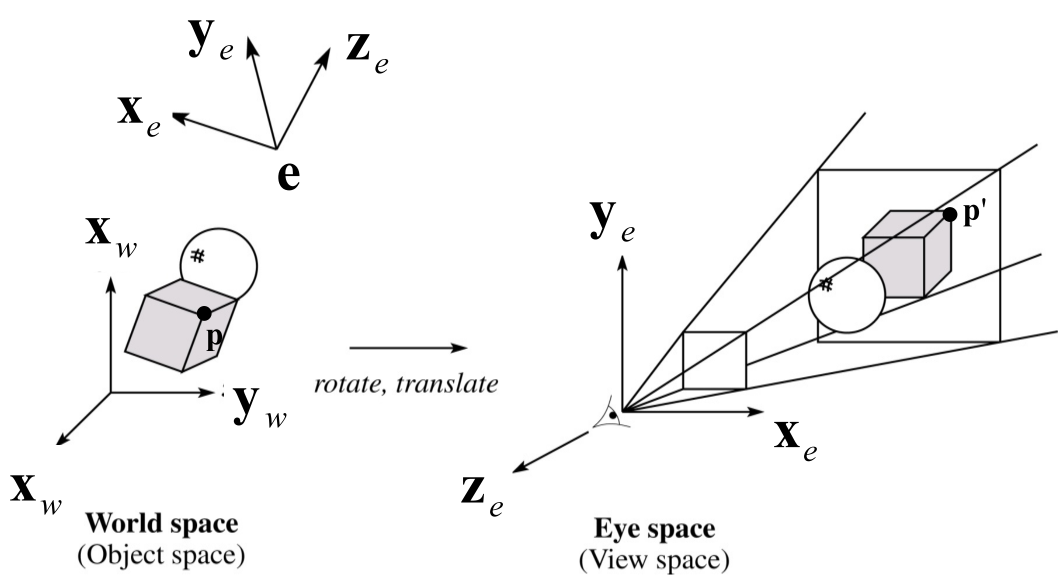
Screen Space (2D)

Scale
Translation

Image Space (pixels)
Raster Space

World -> eye transformation

- Let's look at how we would compute the world->eye transformation.

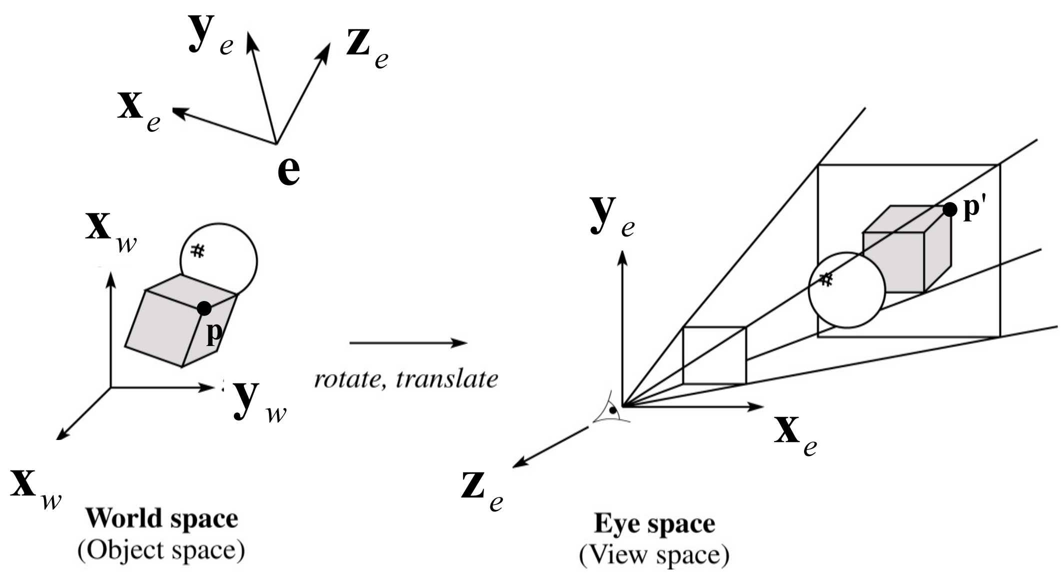


$$\mathbf{p}' = ?$$

$$x_e = \mathbf{x}_e \bullet \mathbf{p}\,?$$

World -> eye transformation

- Let's look at how we would compute the world->eye transformation.



$$\mathbf{p}' = ?$$

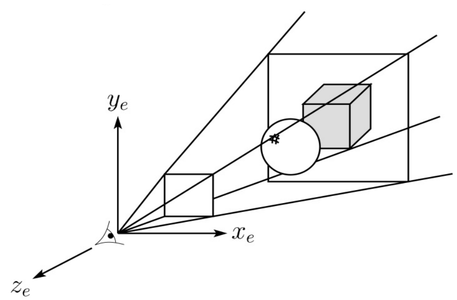
$$x_e = \mathbf{x}_e \bullet (\mathbf{p} - \mathbf{e})$$

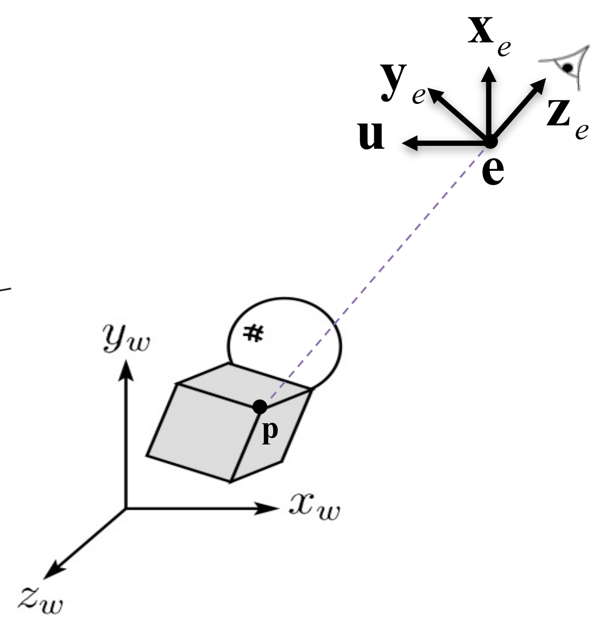
$$y_e = \mathbf{y}_e \bullet (\mathbf{p} - \mathbf{e})$$

$$z_e = \mathbf{z}_e \bullet (\mathbf{p} - \mathbf{e})$$

$$\mathbf{M}_v = \begin{bmatrix} - & \mathbf{x}_e^{\mathrm{T}} & - & 0 \\ - & \mathbf{y}_e^{\mathrm{T}} & - & 0 \\ - & \mathbf{z}_e^{\mathrm{T}} & - & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & | \\ 0 & 1 & 0 & -\mathbf{e} \\ 0 & 0 & 1 & | \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

How to specify eye coordinate?



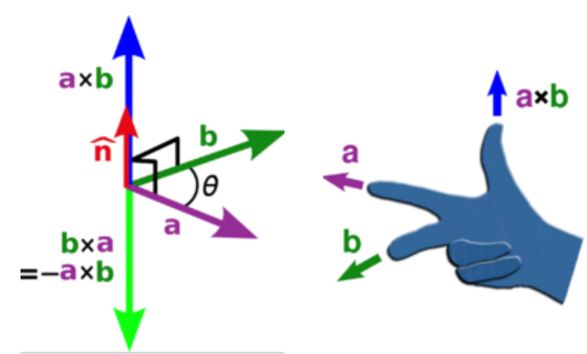


$$\mathbf{z}_e = -\frac{\mathbf{p}-\mathbf{e}}{|\mathbf{p}-\mathbf{e}|}$$

$$\mathbf{x}_e = \frac{\mathbf{u}\times\mathbf{z}_e}{|\mathbf{u}\times\mathbf{z}_e|}$$

$$\mathbf{y}_e = \frac{\mathbf{z}_e\times\mathbf{x}_e}{|\mathbf{z}_e\times\mathbf{x}_e|}$$

1. Give eye location e
2. Give target position p
3. Give upward direction u

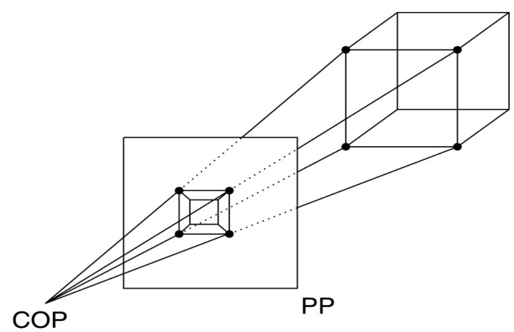


OpenGL has a helper command:

gluLookAt (*eyex*, *eyey*, *eyez*, *px*, *py*, *pz*, *upx*, *upy*, *upz*)

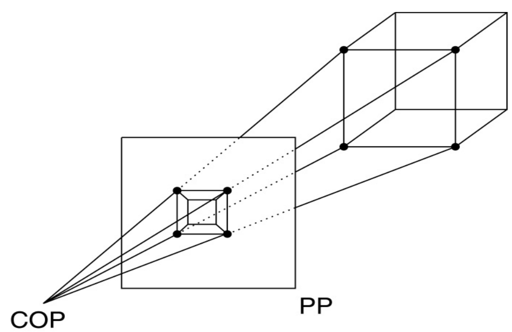
Projections

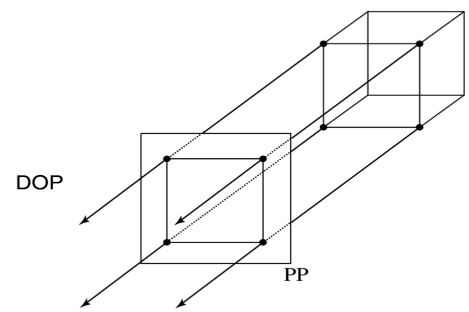
- **Projections** transform points in *n*-space to *m*-space, where *m*<*n*.
- In 3-D, we map points from 3-space to the **projection plane** (PP) (a.k.a., image plane) along **rays** emanating from the center of projection (COP):



Projections

- **Projections** transform points in *n*-space to *m*-space, where $m<n$.
- In 3-D, we map points from 3-space to the **projection plane** (PP) (a.k.a., image plane) along **rays** emanating from the center of projection (COP):





- There are two basic types of projections:
  - Perspective – distance from COP to PP finite
  - Parallel – distance from COP to PP infinite

Parallel projections

- For parallel projections, we specify a **direction of projection** (DOP) instead of a COP.
- We can write orthographic projection onto the *z=0* plane with a simple matrix, such that x'=x, y'=y.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \qquad\qquad \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Parallel projections

- For parallel projections, we specify a **direction of projection** (DOP) instead of a COP.
- We can write orthographic projection onto the *z=0* plane with a simple matrix, such that x'=x, y'=y.

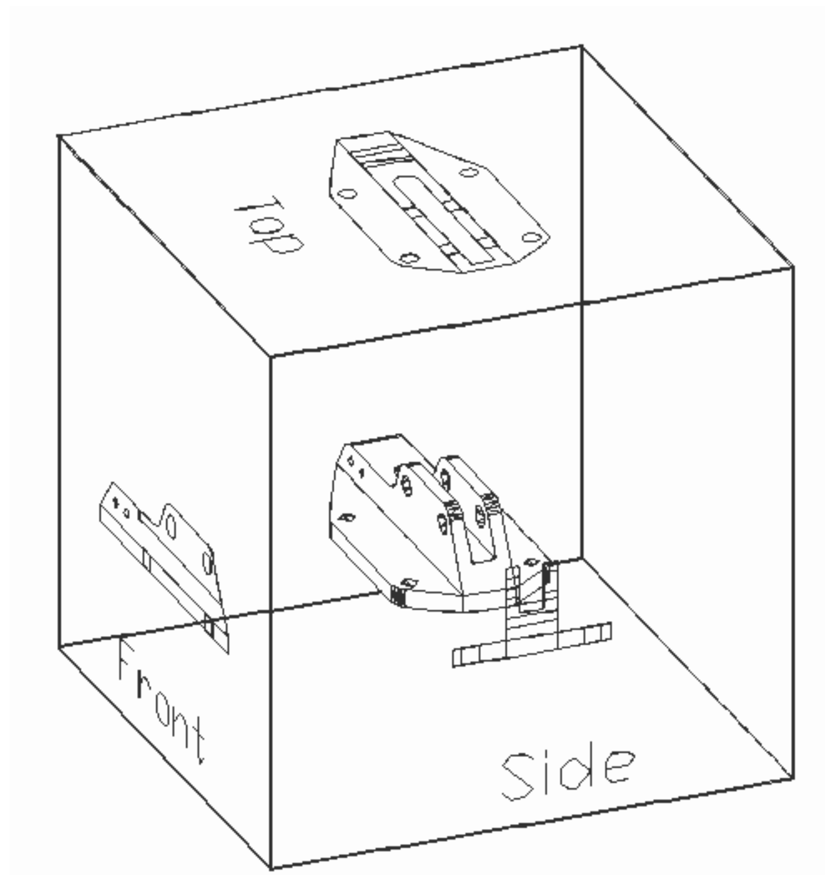
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- Normally, we do not drop the z value right away. Why not?

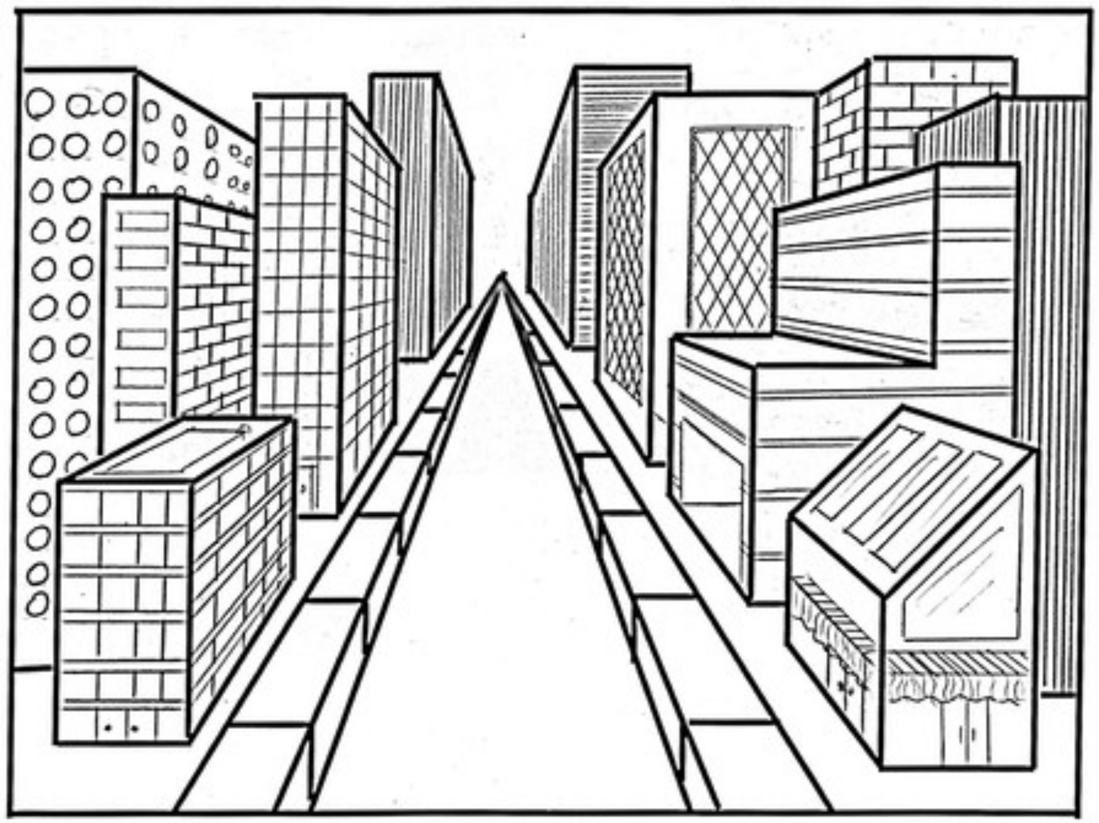
Properties of parallel projection

- Properties of parallel projection:
  - Are actually a kind of affine transformation
    - Parallel lines remain parallel
    - Ratios are preserved
    - Angles not (in general) preserved
  - Not realistic looking
  - Good for exact measurements,
    Most often used in
    - CAD,
    - architectural drawings,
    - etc.,

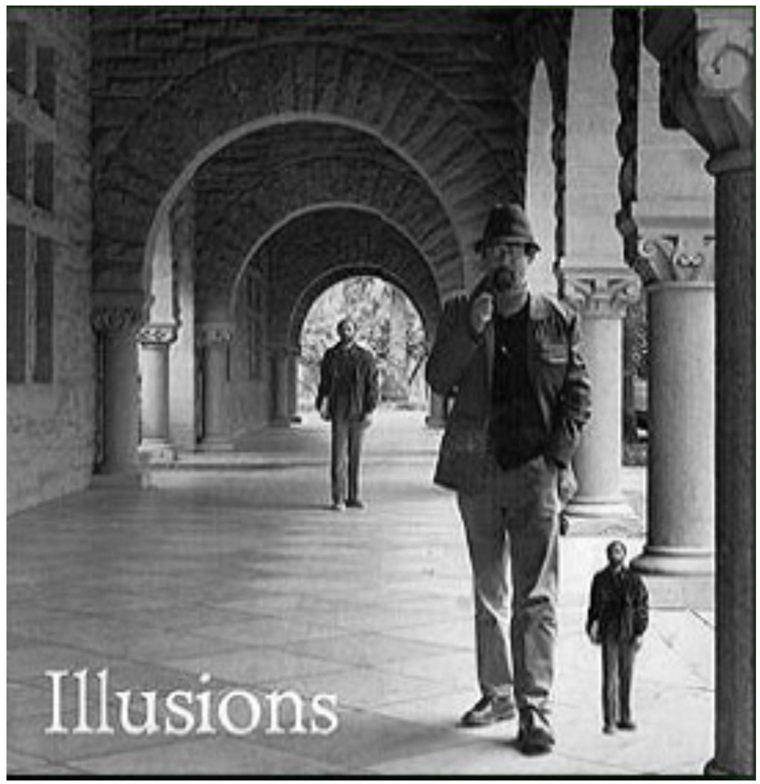
    where taking exact measurement
    is important



Perspective effect



Perspective Illusion



Perspective Illusion