3D Transformations

CS 4620 Lecture 11

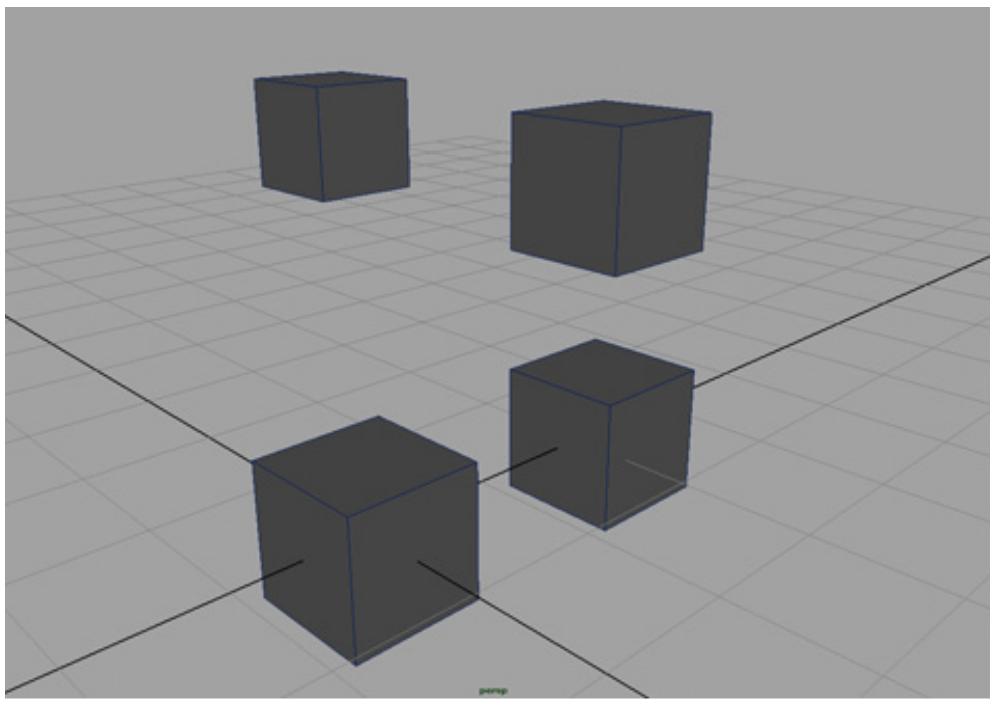
Announcements

- A2 due tomorrow

- Demos on Monday
  - Please sign up for a slot
  - Post on piazza

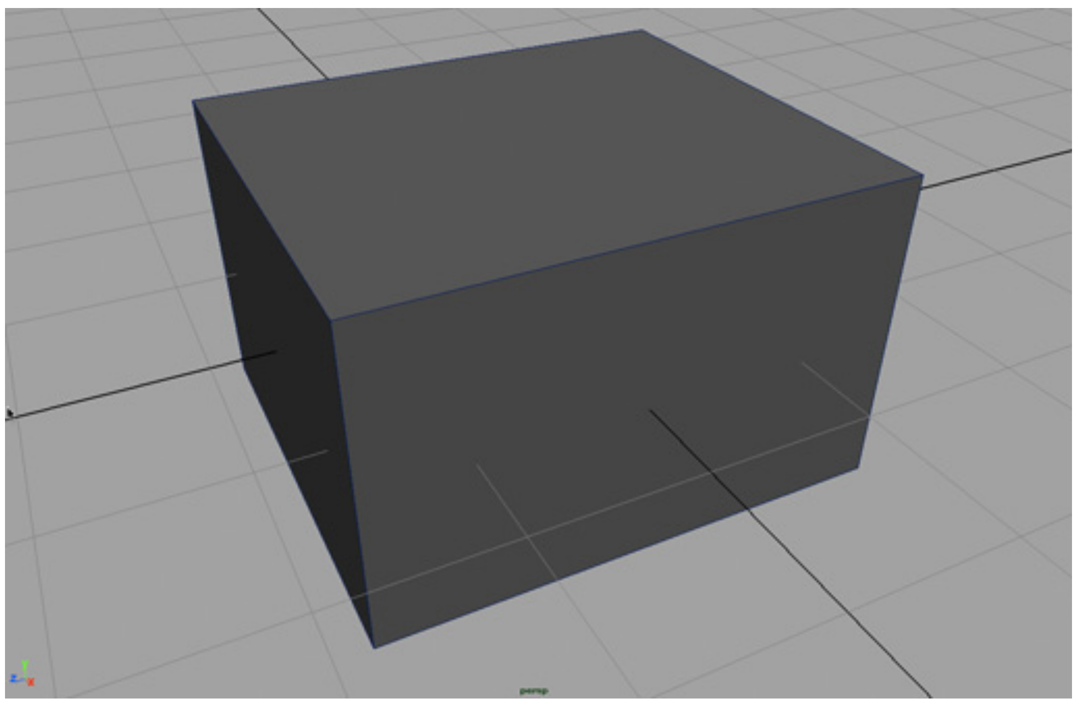
Translation

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



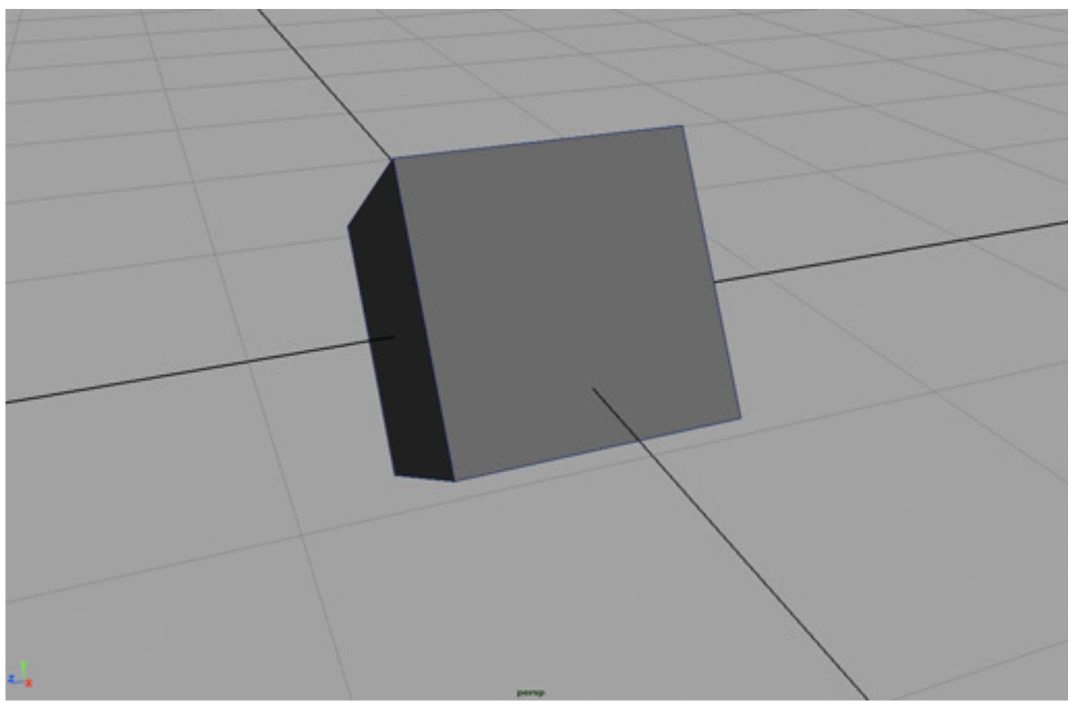
Scaling

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



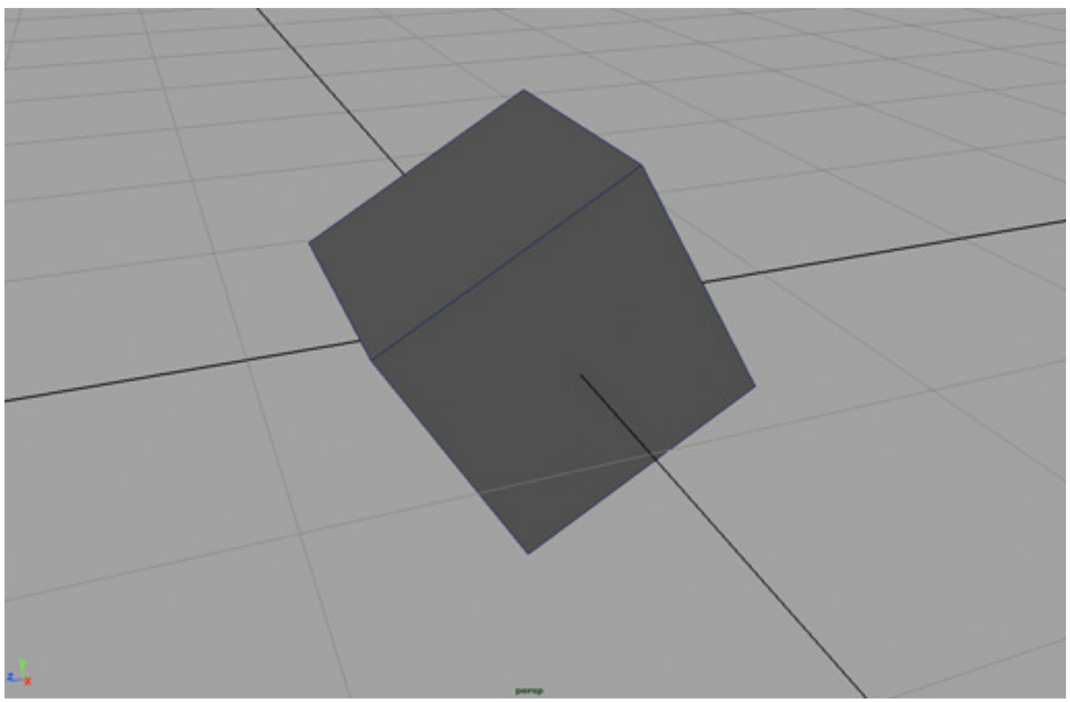
Rotation about z axis

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



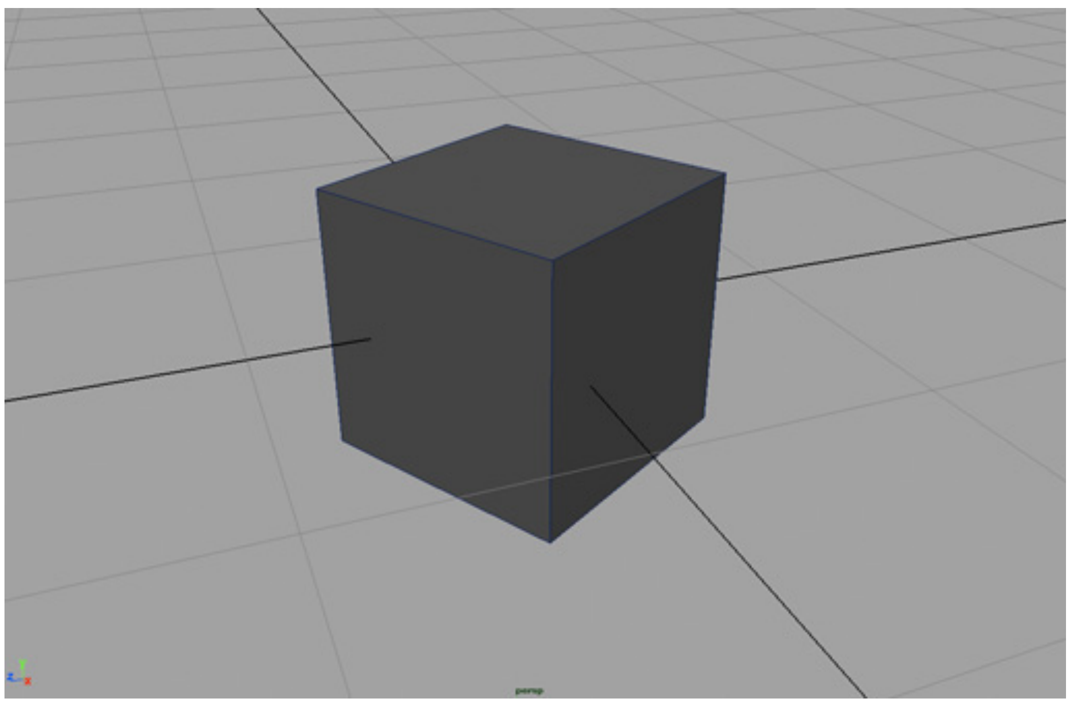
Rotation about *x* axis

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



Rotation about *y* axis

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



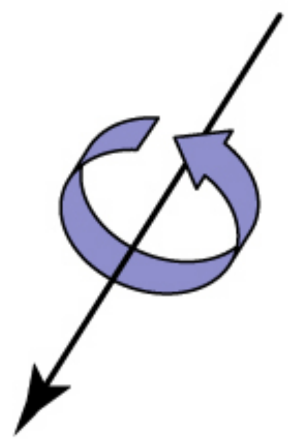
Properties of Matrices

- Translations: linear part is the identity
- Scales: linear part is diagonal
- Rotations: linear part is orthogonal
  - Columns of R are mutually orthonormal: $RR^T = R^T R = I$
  - Also, determinant of R $\det(R) = 1$

General Rotation Matrices

- 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

2D



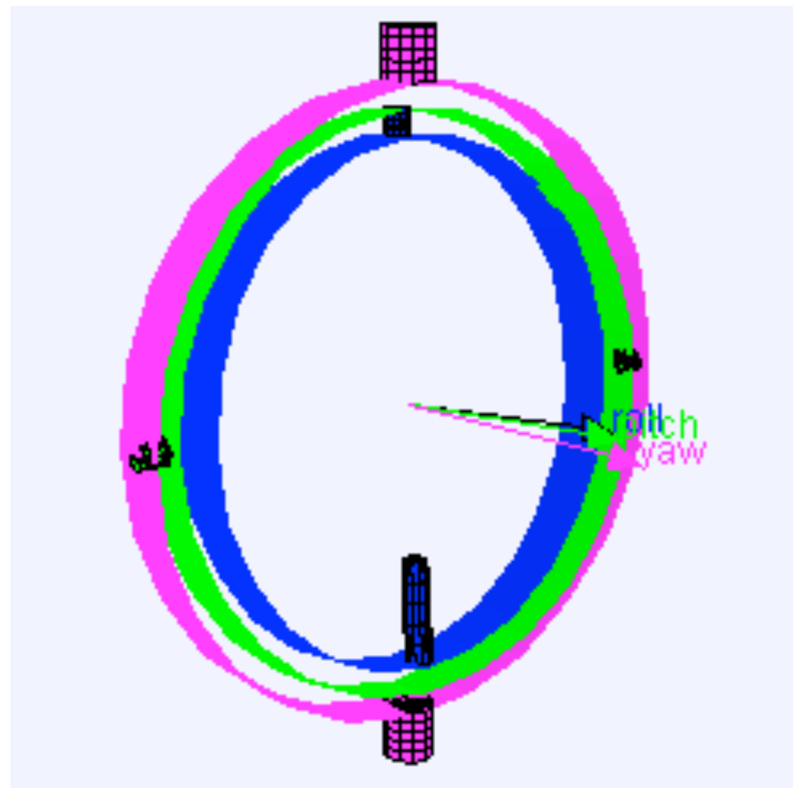
3D

Specifying rotations

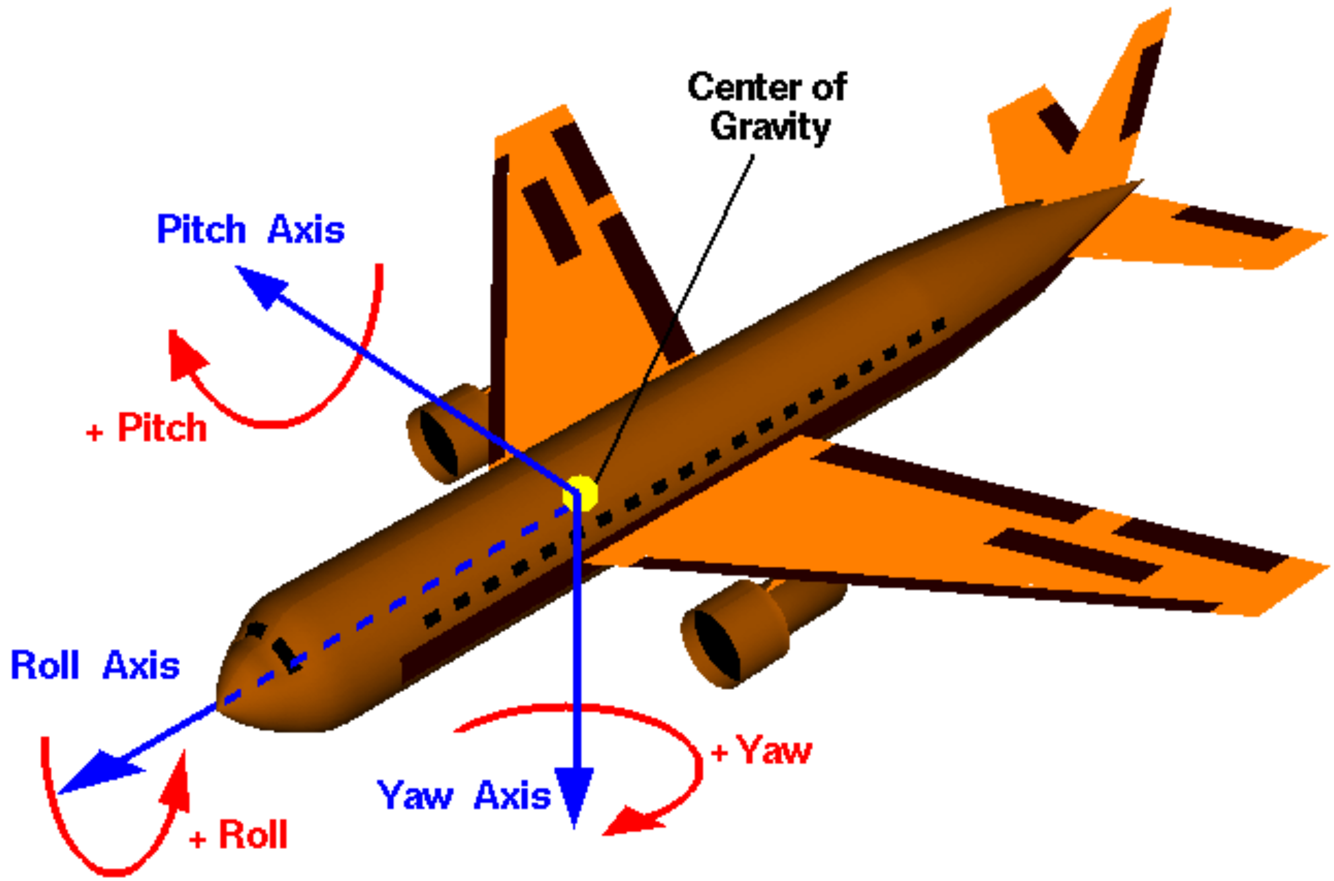
- In 2D, a rotation just has an angle
- In 3D, specifying a rotation is more complex
  - basic rotation about origin: unit vector (axis) and angle
    - convention: positive rotation is CCW when vector is pointing at you
- Many ways to specify rotation
  - Indirectly through frame transformations
  - Directly through
    - Euler angles: 3 angles about 3 axes
    - (Axis, angle) rotation
    - Quaternions

Euler angles

- An object can be oriented arbitrarily
- Euler angles: stack up three coord axis rotations
  - ZYX case: Rz(thetaz)*Ry(thetay)*Rx(thetax)
  - “heading, attitude, bank”
    (common for airplanes)
  - “pitch, yaw, roll”
    (common for ground vehicles)
  - “pan, tilt, roll”
    (common for cameras)



Roll, yaw, Pitch



Specifying rotations: Euler rotations

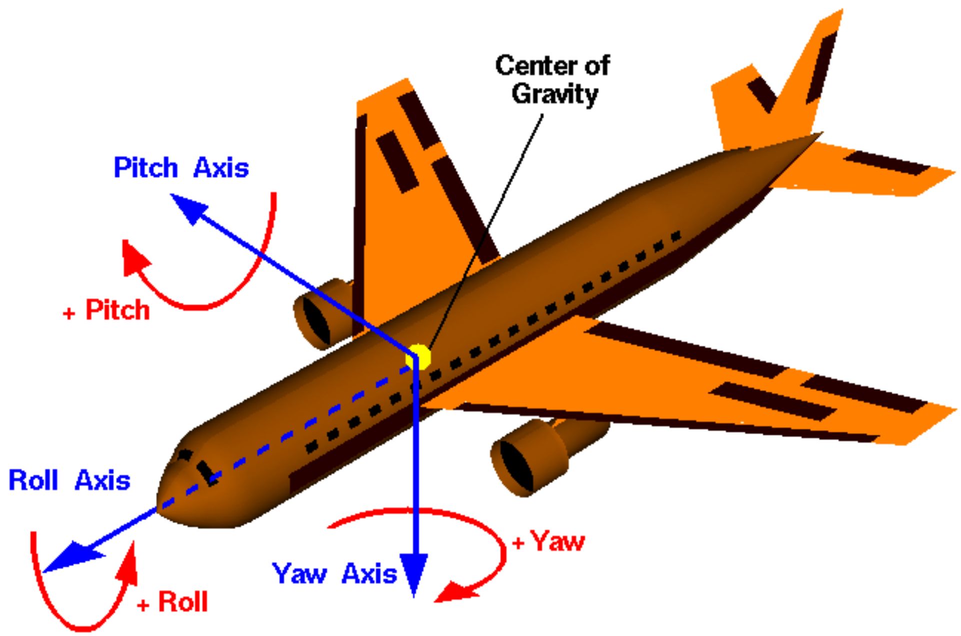
- Euler angles

$$R(\theta_x, \theta_y, \theta_z) = R_z(\theta_z) R_y(\theta_y) R_x(\theta_x)$$

$$R(\theta_x, \theta_y, \theta_z) = \begin{bmatrix} c_y c_z & s_x s_y c_z - c_x s_z & c_x s_y s_z - s_x c_z & 0 \\ c_y s_z & s_x s_y s_z + c_x c_z & c_x s_y s_z - s_x c_z & 0 \\ -s_y & s_x c_y & c_x c_y & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$c_i = \cos(\theta_i)$$
$$s_i = \sin(\theta_i)$$

Gimbal Lock


Center of
Gravity
Pitch Axis
+ Pitch
Roll Axis
+ Roll
Yaw Axis
+ Yaw

Euler angles

- Gimbal lock removes one degree of freedom

$$R(\theta_x, \theta_y, \theta_z) = \begin{bmatrix} 0 & \sin(\theta_x - \theta_z) & \cos(\theta_x - \theta_z) & 0 \\ 0 & \cos(\theta_x - \theta_z) & \sin(\theta_x - \theta_z) & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

worth a look:
http://www.youtube.com/watch?v=zc8b2Jo7mno
(also http://www.youtube.com/watch?v=rrUCBOIJdt4)

Matrices for axis-angle rotations

- Showed matrices for coordinate axis rotations
  - but what if we want rotation about some other axis?
- Compute by composing elementary transforms
  - transform rotation axis to align with *x* axis
  - apply rotation
  - inverse transform back into position
- Just as in 2D this can be interpreted as a similarity transform

Building general rotations

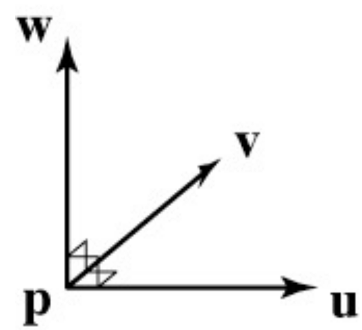
- Using elementary transforms you need three
  - translate axis to pass through origin
  - rotate about *y* to get into *x*-*y* plane
  - rotate about *z* to align with *x* axis

- Alternative: construct frame and change coordinates
  - choose *p*, *u*, *v*, *w* to be orthonormal frame with *p* and *u* matching the rotation axis
  - apply transform $T = F\, R_x(\theta)\, F^{-1}$

Orthonormal frames in 3D

- Useful tools for constructing transformations
- Recall rigid motions
  - affine transforms with pure rotation
  - columns (and rows) form right handed ONB
    - that is, an **o**rtho**n**ormal **b**asis

$$F = \begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{w} & \mathbf{p} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Building 3D frames

- Given a vector **a** and a secondary vector **b**
  - The **u** axis should be parallel to **a**; the **u**–**v** plane should contain **b**
    - **u** = **a** / ||**a**||
    - **w** = **u** x **b**; **w** = **w** / ||**w**||
    - **v** = **w** x **u**
- Given just a vector **a**
  - The **u** axis should be parallel to **a**; don't care about orientation about that axis
    - Same process but choose arbitrary **b** first
    - Good choice for **b** is not near **a**: e.g. set smallest entry to 1

Building general rotations

- Construct frame and change coordinates
  - choose *p*, *u*, *v*, *w* to be orthonormal frame with *p* and *u* matching the rotation axis
  - apply similarity transform $T = F\, R_x(\theta)\, F^{-1}$
  - interpretation: move to *x* axis, rotate, move back
  - interpretation: rewrite *u*-axis rotation in new coordinates
  - (each is equally valid)

$$\begin{bmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\theta) & -sin(\theta) \\ 0 & sin(\theta) & cos(\theta) \end{bmatrix} \begin{bmatrix} u_x & u_y & u_z \\ v_x & v_y & v_z \\ w_x & w_y & w_z \end{bmatrix}$$

  - (note above is linear transform; add affine coordinate)

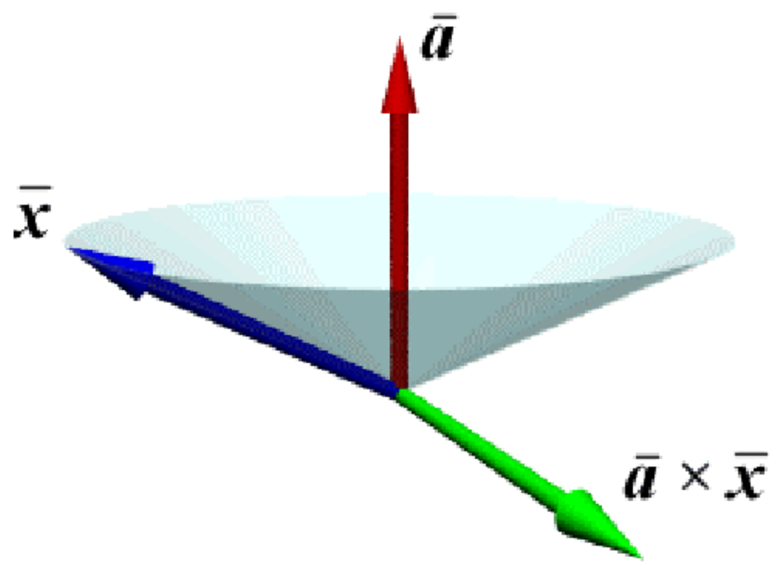
Building general rotations

- Construct frame and change coordinates
  - choose *p*, *u*, *v*, *w* to be orthonormal frame with *p* and *u* matching the rotation axis
  - apply similarity transform $T = F\, R_x(\theta)\, F^{-1}$
  - interpretation: move to *x* axis, rotate, move back
  - interpretation: rewrite *u*-axis rotation in new coordinates
  - (each is equally valid)

- Sleeker alternative: Rodrigues' formula

Derivation of General Rotation Matrix

- Axis angle rotation



Axis-angle ONB

$$\vec{x}_{||} = (\vec{a}.\vec{x})\vec{a}$$

$$\vec{x}_{\perp} = (\vec{x} - \vec{x}_{||}) = (\vec{x} - (\vec{a}.\vec{x})\vec{a})$$

$$\vec{a} \times \vec{x}_{\perp} = \vec{a} \times (\vec{x} - \vec{x}_{||}) = \vec{a} \times (\vec{x} - (\vec{a}.\vec{x})\vec{a}) = \vec{a} \times \vec{x}$$

Axis-angle rotation

$$x_{rotated} = \vec{x}_{\|} + \vec{v}$$

$$x_{rotated} = \alpha\ \vec{a} + \beta\ \vec{x}_{\perp} + \gamma\ \vec{a} \times \vec{x}$$

$$\vec{v} = \cos\theta\ \vec{x}_{\perp} + \sin\theta\ \vec{a} \times \vec{x}$$

$$x_{rotated} = \vec{x}_{\|} + \cos\theta\ \vec{x}_{\perp} + \sin\theta\ \vec{a} \times \vec{x}$$

$$x_{rotated} = (\vec{a}.\vec{x})\vec{a} + \cos\theta\ (x - (\vec{a}.\vec{x})\vec{a}) + \sin\theta\ \vec{a} \times \vec{x}$$

$$x_{rotated} = (\vec{a}.\vec{x})(1 - \cos\theta)\vec{a} + \cos\theta\ \vec{x} + \sin\theta\ \vec{a} \times \vec{x}$$

Rotation Matrix for Axis-Angle

$$x_{rotated} = (\vec{a}.\vec{x})(1-\cos\theta)\vec{a} + \cos\theta\ \vec{x} + \sin\theta\ \vec{a}\times\vec{x}$$

$$x_{rotated} = (Sym(\vec{a})(1-\cos\theta) + I\cos\theta\ + Skew(\vec{a})\sin\theta\ )\vec{x}$$

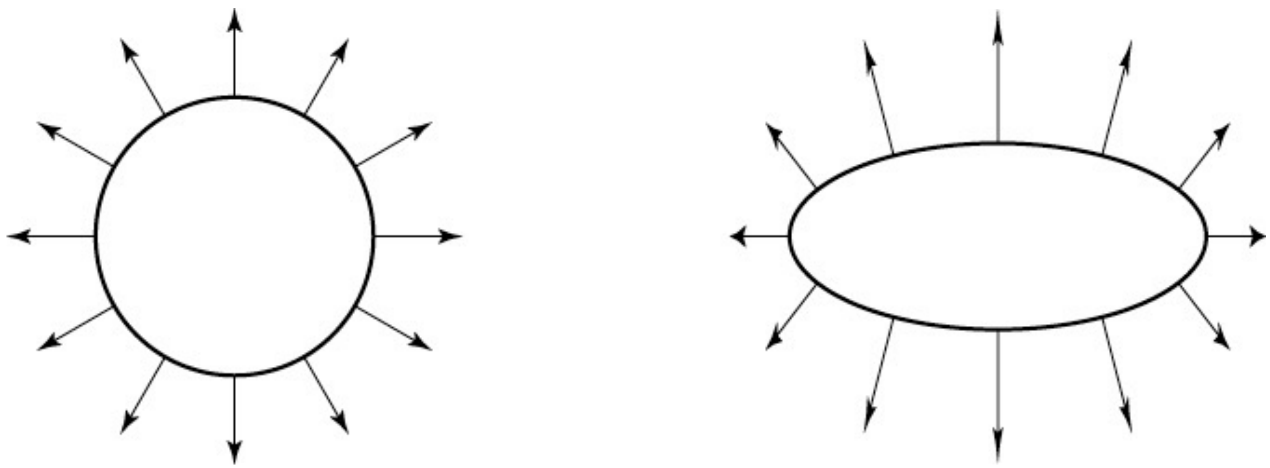
$$Sym(\vec{a}) = \begin{bmatrix} a_x \\ a_y \\ a_z \\ 0 \end{bmatrix} \begin{bmatrix} a_x & a_y & a_z & 0 \end{bmatrix} = \begin{bmatrix} a_x^2 & a_x a_y & a_x a_z & 0 \\ a_x a_y & a_y^2 & a_y a_z & 0 \\ a_x a_z & a_y a_z & a_z^2 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$Skew(\vec{a}) = \begin{bmatrix} 0 & -a_z & a_y & 0 \\ a_z & 0 & -a_x & 0 \\ -a_y & a_x & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$Skew(\vec{a})\vec{x} = \vec{a}\times\vec{x}$$

Transforming normal vectors

- Transforming surface normals
  - differences of points (and therefore tangents) transform OK
  - normals do not. Instead, use inverse transpose matrix



have: $\mathbf{t} \cdot \mathbf{n} = \mathbf{t}^T \mathbf{n} = 0$
want: $M\mathbf{t} \cdot X\mathbf{n} = \mathbf{t}^T M^T X \mathbf{n} = 0$
so set $X = (M^T)^{-1}$
then: $M\mathbf{t} \cdot X\mathbf{n} = \mathbf{t}^T M^T (M^T)^{-1} \mathbf{n} = \mathbf{t}^T \mathbf{n} = 0$