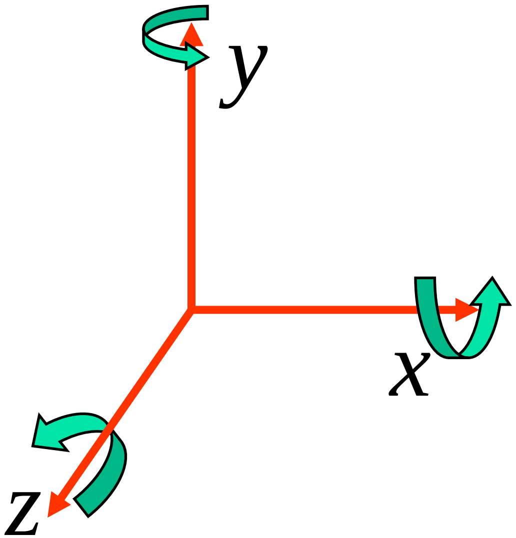
3D Geometric Transformations

Chapter 5
Intro. to Computer Graphics
Spring 2009, Y. G. Shin

3D Transformation

Right-handed coordinate system



$$\begin{bmatrix} x' \\ y' \\ z' \\ h \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & x_3 & t_x \\ y_1 & y_2 & y_3 & t_y \\ z_1 & z_2 & z_3 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

3D Transformation

- Translation

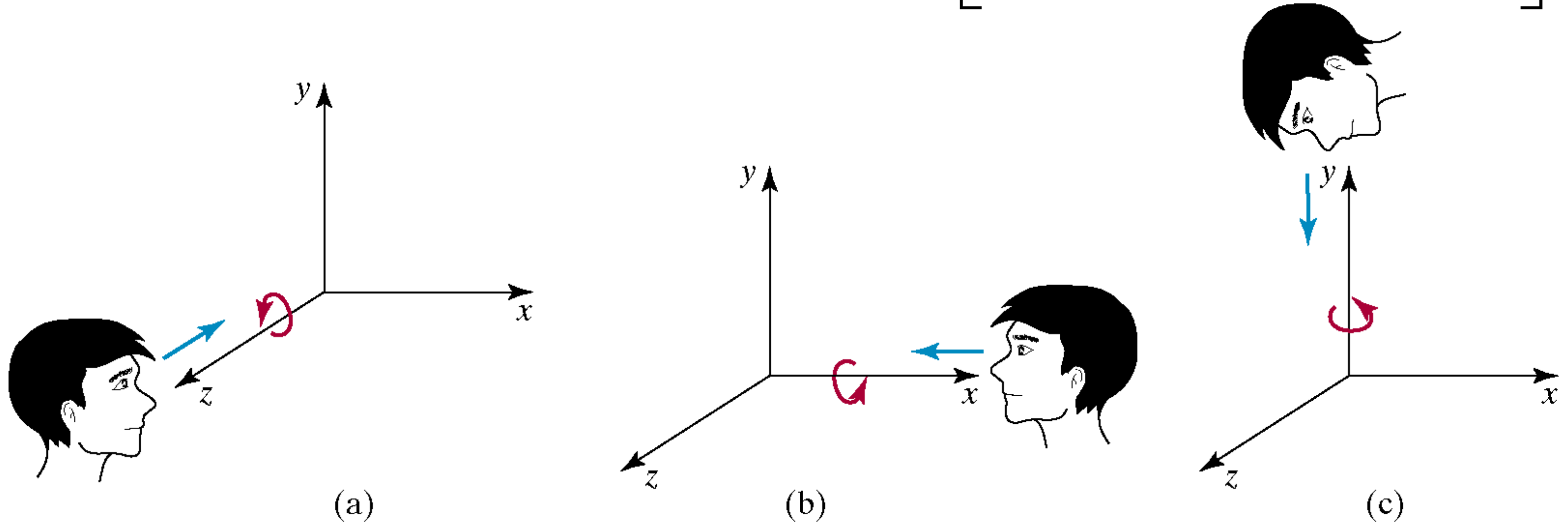
$$\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Scaling

$$\begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

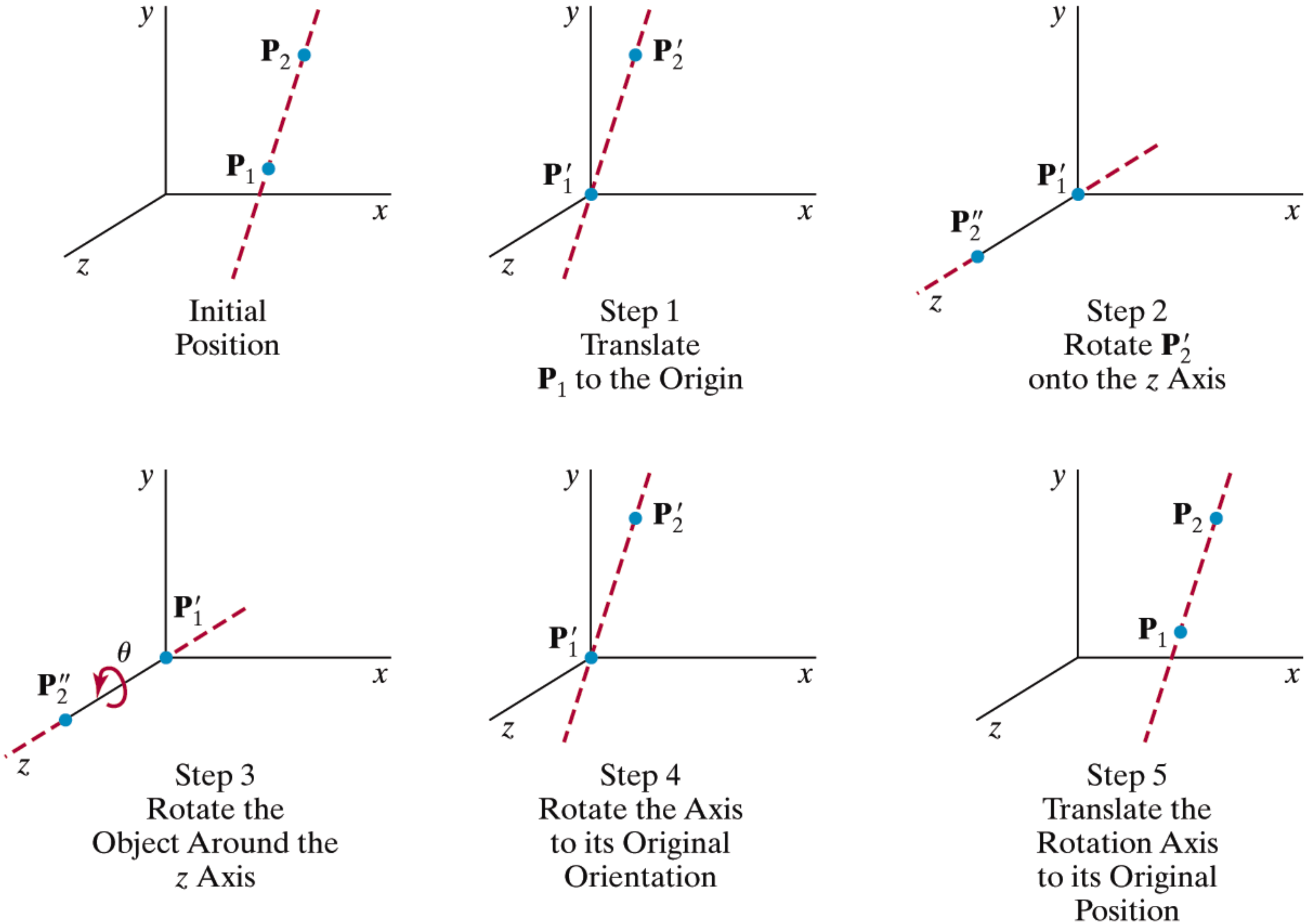
3D Rotation

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



3D rotations do NOT commute!

Rotation About Arbitrary Axis



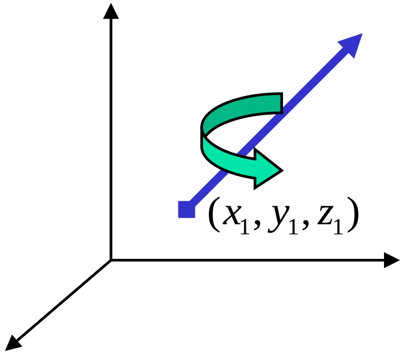
Rotation about an arbitrary axis

1. Translation : rotation axis passes through the origin

$$T = \begin{bmatrix} 1 & 0 & 0 & -x_1 \\ 0 & 1 & 0 & -y_1 \\ 0 & 0 & 1 & -z_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2. Make the rotation axis on the z-axis

$$R_x(\alpha) \cdot R_y(\beta)$$



3. Do rotation

$$R_z(\theta)$$

4. Rotation & translation

$$T^{-1} \cdot R_y^{-1}(\beta) \cdot R_x^{-1}(\alpha)$$

Rotation about an arbitrary axis

$$R(\theta) = T^{-1} \cdot R_x^{-1}(\alpha) \cdot R_y^{-1}(\beta) \cdot R_z(\theta) \cdot R_y(\beta) \cdot R_x(\alpha) \cdot T$$

Rotation About Arbitrary Axis

- Rotate **u** onto the *z*-axis

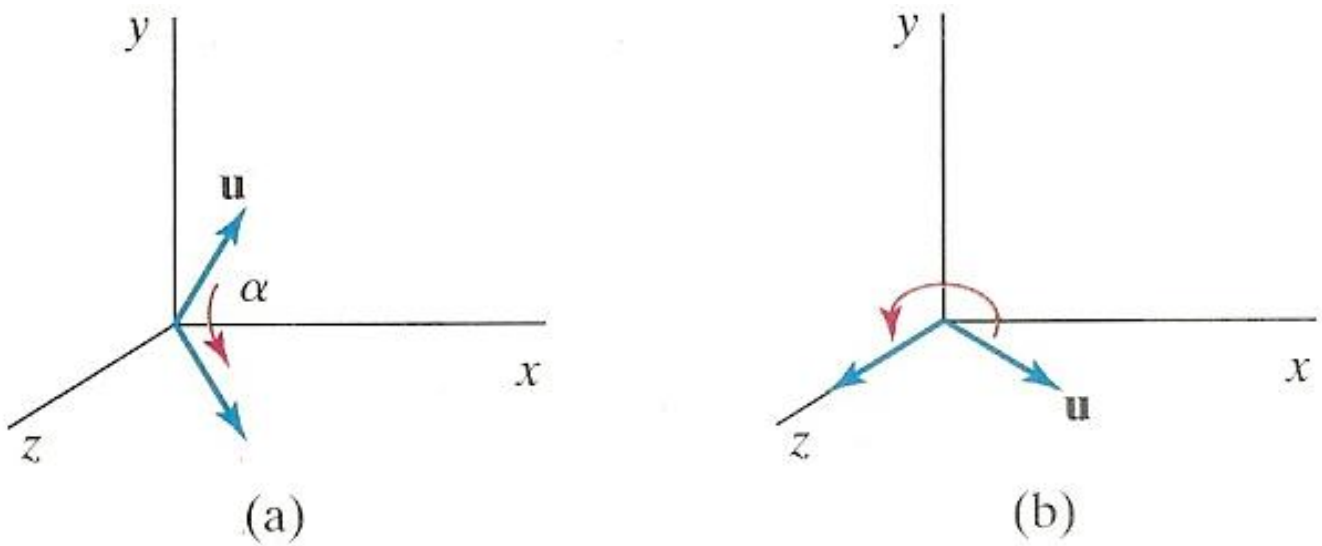


FIGURE 5-45 Unit vector **u** is rotated about the x axis to bring it into the xz plane (a), then it is rotated around the y axis to align it with the z axis (b).

Rotation About Arbitrary Axis

- Rotate a unit vector **u** onto the z-axis
  - **u′ :** Project **u** onto the yz-plane to compute angle α
  - **u″ :** Rotate **u** about the x-axis by angle α
  - Rotate **u″** onto the z-asis

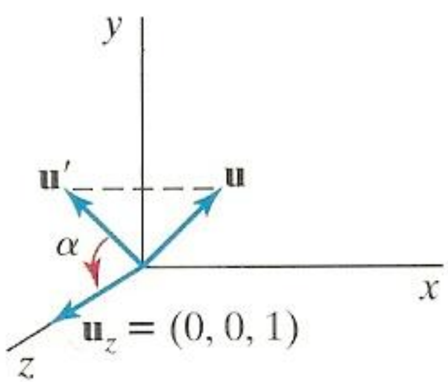


FIGURE 5-46 Rotation of **u** around the *x* axis into the *xz* plane is accomplished by rotating **u**′ (the projection of **u** in the *yz* plane) through angle α onto the *z* axis.

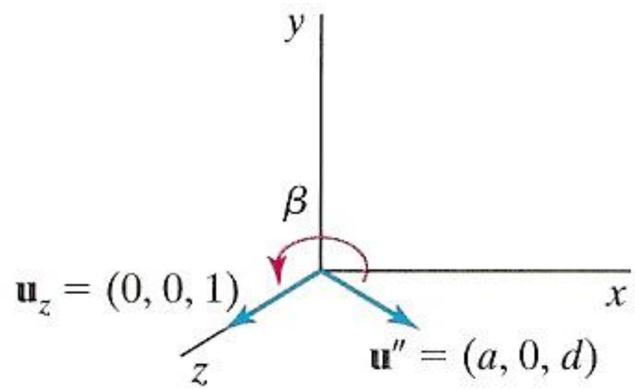
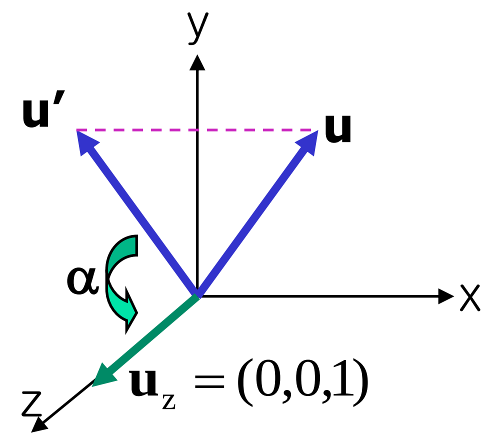


FIGURE 5-47 Rotation of unit vector **u**″ (vector **u** after rotation into the *xz* plane) about the *y* axis. Positive rotation angle β aligns **u**″ with vector $\mathbf{u}_z$.

Rotation About Arbitrary Axis

- Rotate **u'** about the x-axis onto the *z*-axis
  - Let **u**=(a,b,c) and thus **u'**=(0,b,c)
  - Let $\mathbf{u}_z$=(0,0,1)

$$\cos\alpha = \frac{\mathbf{u}' \cdot \mathbf{u}_z}{\|\mathbf{u}'\| \|\mathbf{u}_z\|} = \frac{c}{\sqrt{b^2 + c^2}}$$



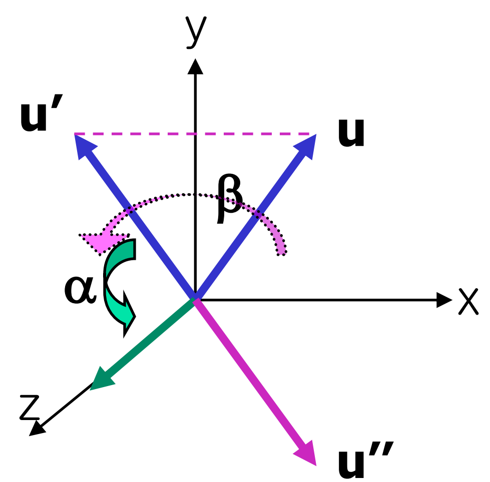
$$\mathbf{u}' \times \mathbf{u}_z = \mathbf{u}_x \|\mathbf{u}'\| \|\mathbf{u}_z\| \sin\alpha$$
$$= \mathbf{u}_x b$$

$$\Longrightarrow \quad \sin\alpha = \frac{b}{\|\mathbf{u}'\| \|\mathbf{u}_z\|} = \frac{b}{\sqrt{b^2 + c^2}}$$

Rotation About Arbitrary Axis

- Rotate **u'** about the x-axis onto the *z*-axis
  - Since we know both cos α and sin α, the rotation matrix can be obtained

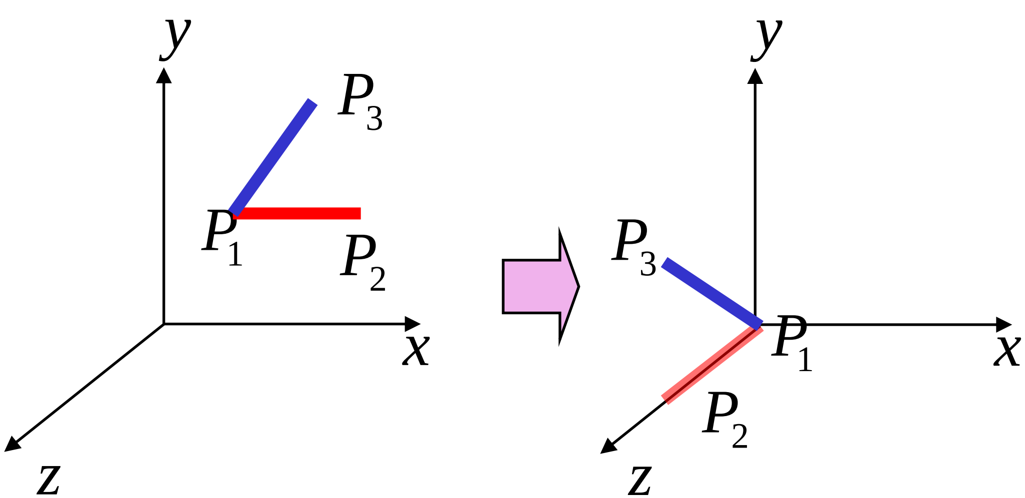
$$\mathbf{R}_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{c}{\sqrt{b^2+c^2}} & \frac{-b}{\sqrt{b^2+c^2}} & 0 \\ 0 & \frac{b}{\sqrt{b^2+c^2}} & \frac{c}{\sqrt{b^2+c^2}} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



- Rotate **u''** onto the z-asis
  - With the similar way, we can compute the angle β

Rotation about an arbitrary axis using orthogonal matrix

- Unit row vector of R rotates into the principle axes x, y, and z.

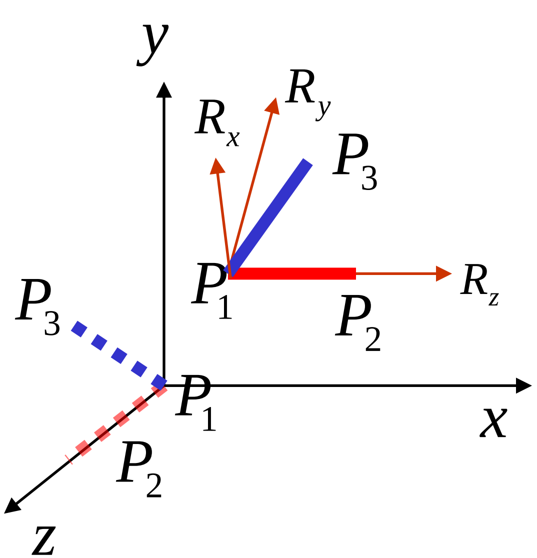


Initial position

Final position

$$R = \begin{bmatrix} r_{1x} & r_{2x} & r_{3x} \\ r_{1y} & r_{2y} & r_{3y} \\ r_{1z} & r_{2z} & r_{3z} \end{bmatrix} ?$$

Rotation about an arbitrary axis using orthogonal matrix



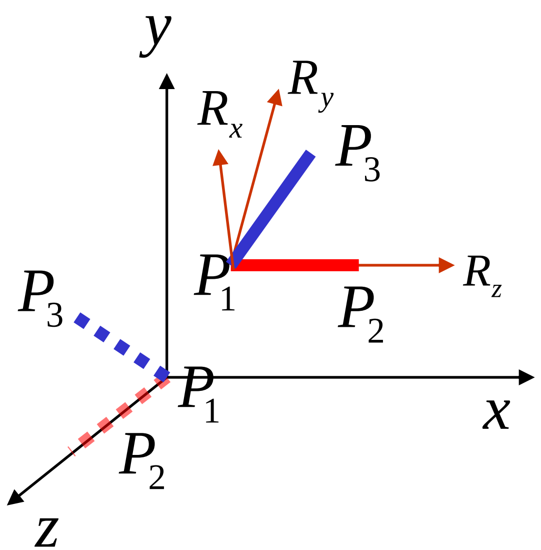
R_z is the unit vector along P_1P_2 that will rotate into the positive z axis

$$R_z = \begin{bmatrix} r_{1z} & r_{2z} & r_{3z} \end{bmatrix} = \frac{P_1P_2}{|P_1P_2|}$$

R_x is perpendicular to the plane P_1, P_2 and P_3 that will rotate into the positive x axis

$$R_x = \begin{bmatrix} r_{1x} & r_{2x} & r_{3x} \end{bmatrix} = \frac{P_1P_3 \times P_1P_2}{|P_1P_3 \times P_1P_2|}$$

Rotation about an arbitrary axis using orthogonal matrix



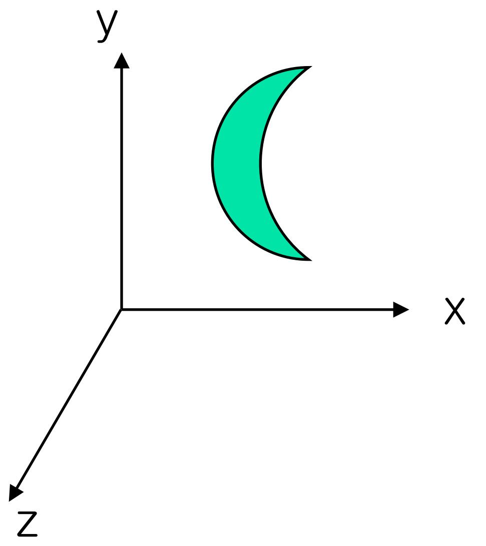
Finally,

$$R_y = \begin{bmatrix} r_{1y} & r_{2y} & r_{3y} \end{bmatrix} = R_z \times R_x$$

$$\Rightarrow \quad R = R_x(\beta) \cdot R_y(\alpha) = \begin{bmatrix} R_x \\ R_y \\ R_z \end{bmatrix}$$

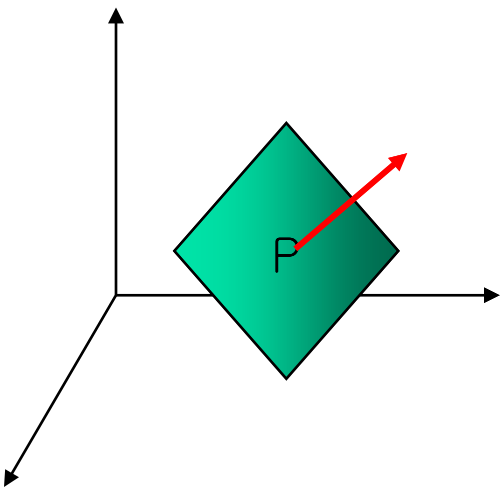
Reflection

- Reflection about xy plane



$$RF_z = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Reflection about an arbitrary plane



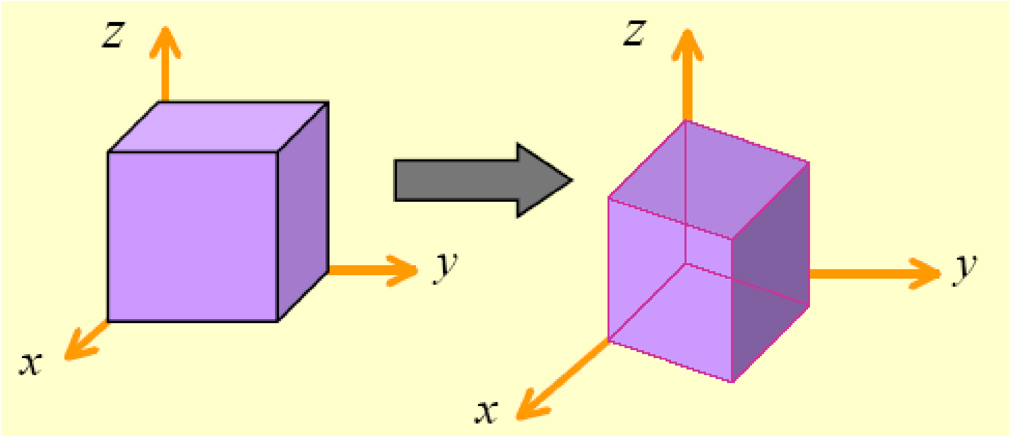
1. Translate a known point P, that lies in the reflection plane, to the origin of the coordinate system
2. Rotate the normal vector to the reflection plane at the origin until the plane lies on z=0 plane.
3. After also applying the above transformations to the object, reflect the object through z=0 coordinate plane.
4. Perform the inverse transformations.

Shear

- Shear in x-direction

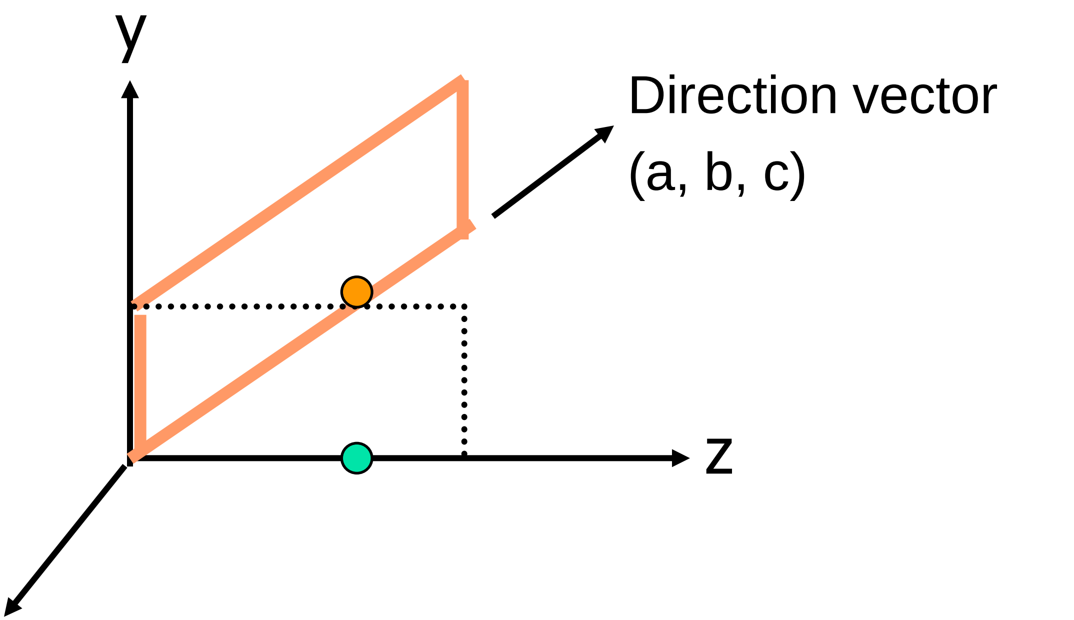
$$SH_x = \begin{bmatrix} 1 & a & b & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x + ay + bz \\ y \\ z \\ 1 \end{bmatrix}$$

When $b = 0$



Shearing along xy-plane

$$SH_{xy} = \begin{bmatrix} 1 & 0 & \frac{a}{c} & 0 \\ 0 & 1 & \frac{b}{c} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



(0, 0, 1) is moved to $\left(\frac{a}{c}, \quad \frac{b}{c}, \quad 1 \right)$

How we represent rotations?

- Rotation (direction cosine) matrix
- Euler angles
- Angular Displacement
- Unit quaternions

Euler Angles

Arbitrary rotation can be represented by three rotation along x,y,z axis.

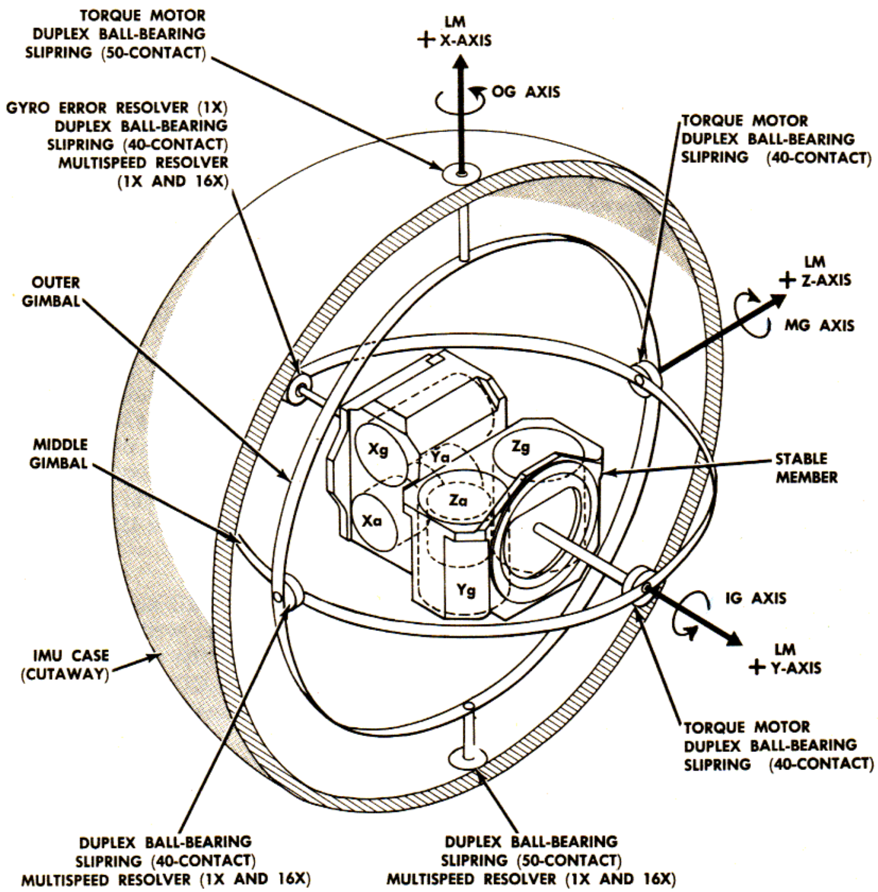
$$R_{XYZ}(\gamma,\beta,\alpha)=R_z(\alpha)R_y(\beta)R_x(\gamma)$$

$$=\begin{bmatrix} C\alpha C\beta & C\alpha S\beta S\gamma - S\alpha C\gamma & C\alpha S\beta C\gamma + S\alpha S\gamma & 0 \\ S\alpha C\beta & S\alpha S\beta S\gamma + C\alpha C\gamma & S\alpha S\beta C\gamma - C\alpha S\gamma & 0 \\ -S\beta & C\beta S\gamma & C\beta C\gamma & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Euler angle : $R_{XYZ}(\gamma,\beta,\alpha) \leftrightarrow$ rotation matrix

← Not easy

Gimble

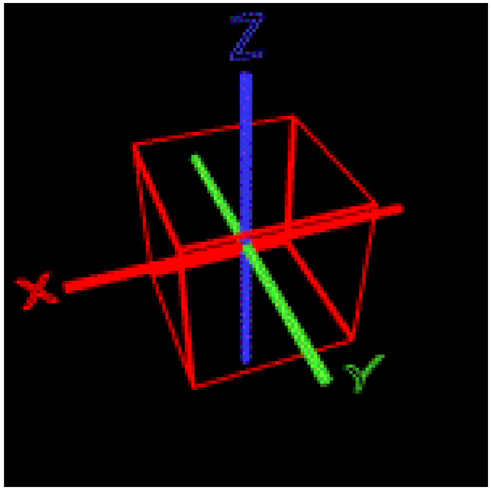


300LM4-152

Figure 2.1-24. IMU Gimbal Assembly

Gimble Lock

- Rotation about three orthogonal axes
  - 12 combinations
    - XYZ, XYX, XZY, XZX
    - YZX, YZY, YXZ, YXY
    - ZXY, ZXZ, ZYX, ZYZ
- ***Gimble lock***
  - Gimbal lock is the phenomenon of two rotational axis of an object pointing in the same direction. Simply put, it means your object won't rotate how you think it ought to rotate.
  - Loss of degree of freedom
  - http://www.anticz.com/eularqua.htm



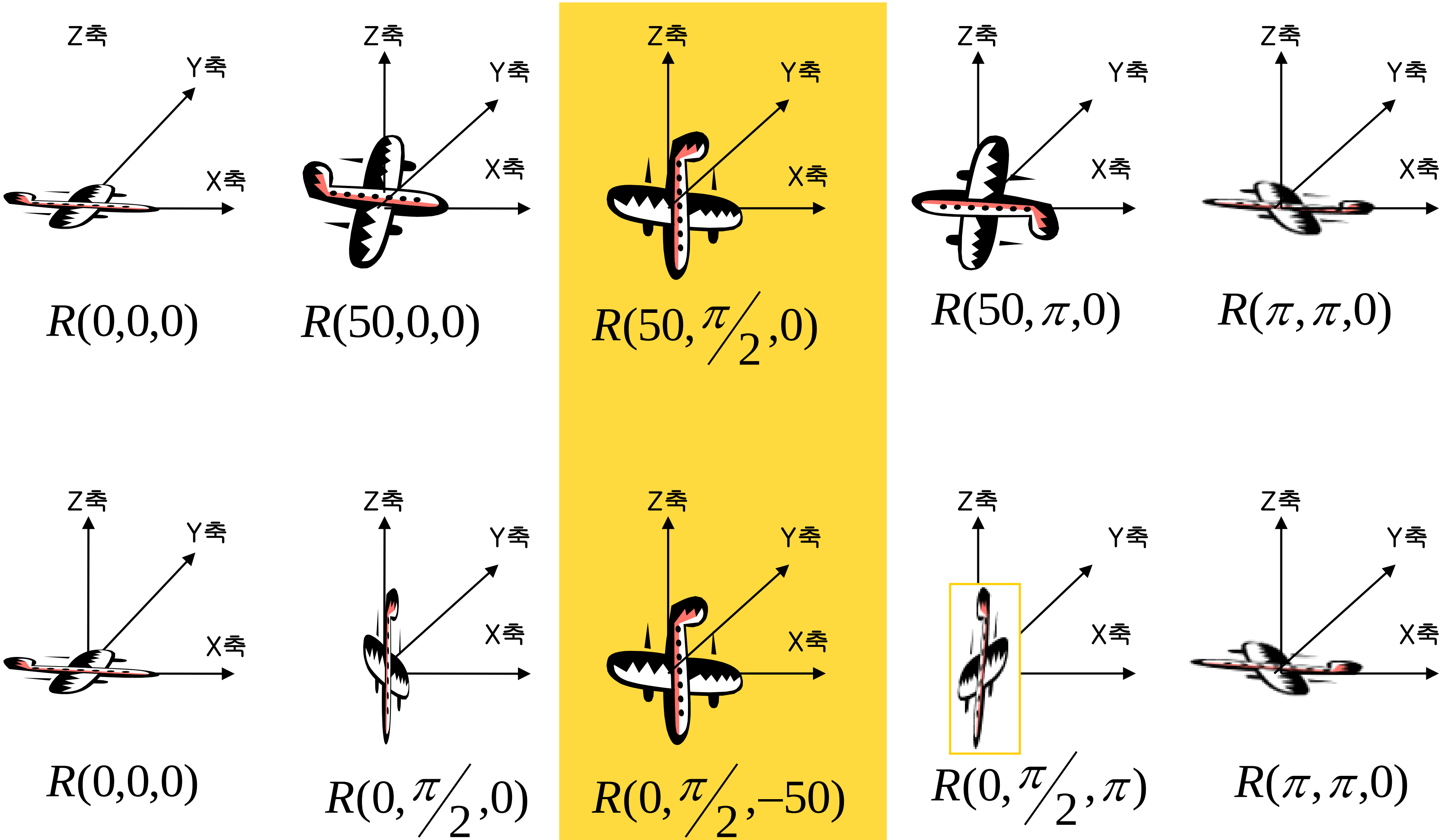
Gimble Lock

- Gimble lock gives ambiguous representation of a rotation angle.
- Two different Euler angles can represent the same orientation

$$R_1 = (r_x, r_y, r_z) = (\theta, \frac{\pi}{2}, 0) \quad \text{and} \quad R_2 = (0, \frac{\pi}{2}, -\theta)$$

- This ambiguity brings unexpected results of animation where frames are generated by interpolation.

Gimble Lock



Z축
Y축
X축
R(0,0,0)
R(50,0,0)
R(50,π/2,0)
R(50,π,0)
R(π,π,0)
R(0,0,0)
R(0,π/2,0)
R(0,π/2,−50)
R(0,π/2,π)
R(π,π,0)

Gimble Lock

Set $\theta_y = \frac{\pi}{2}$, and set θ_x and θ_z arbitrarily.

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

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

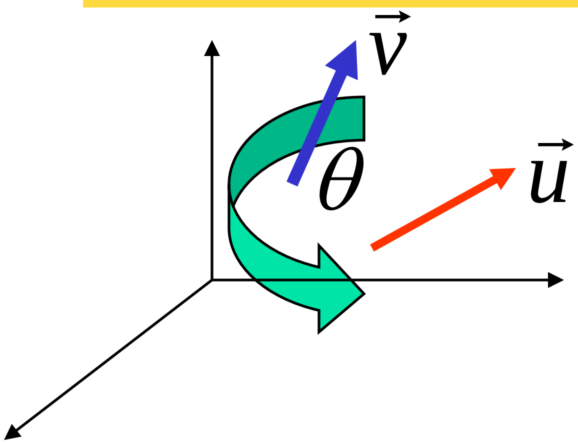
Transformation only depends on the difference

$\longrightarrow$ We lost one DOF.

Angular Displacement

- Arbitrary rotation can be represented by one rotation (by a scalar angle) around an axis(unit vector)
- No Gimble lock but *NOT* smooth interpolation for animation
- Supported by OpenGL

Rotating $\vec{v}$ about a unit vector $\vec{u}$ by an angle θ

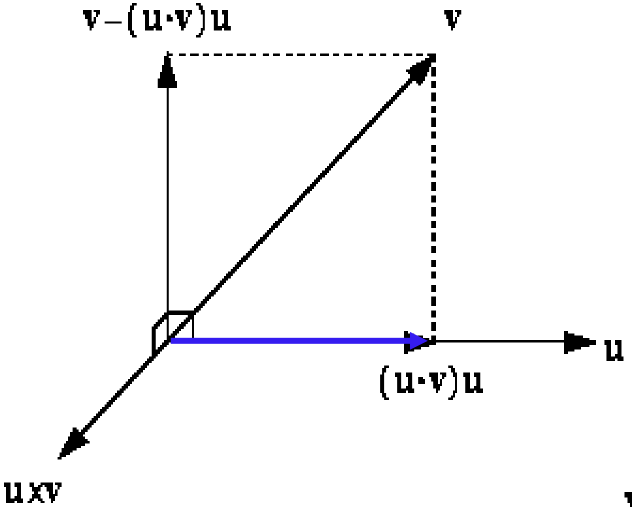


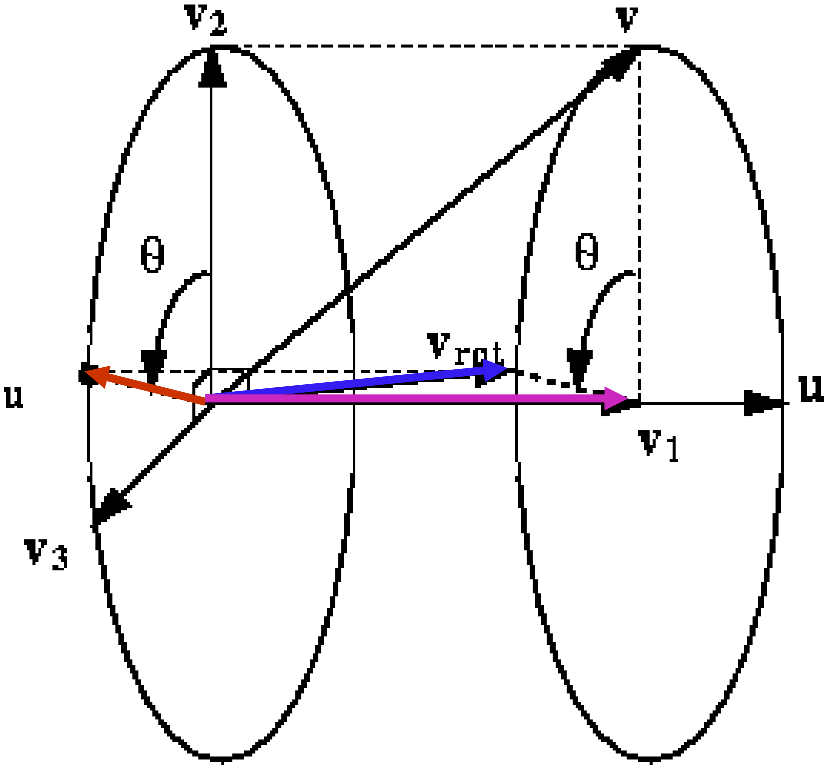
$$\vec{v}_{rot} = (\vec{u}\cdot\vec{v})\vec{u} + \cos\theta(\vec{v} - (\vec{u}\cdot\vec{v})\vec{u})$$
$$+ \sin\theta(\vec{u}\times\vec{v})$$

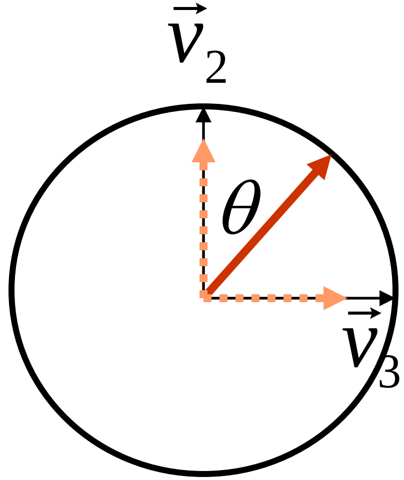
Angular Displacement

$$\vec{v}_{rot} = (\vec{u}\cdot\vec{v})\vec{u} + \cos\theta(\vec{v} - (\vec{u}\cdot\vec{v})\vec{u}) + \sin\theta(\vec{u}\times\vec{v})$$

$$= \vec{v}_1 + \cos\theta\,\vec{v}_2 + \sin\theta\,\vec{v}_3$$

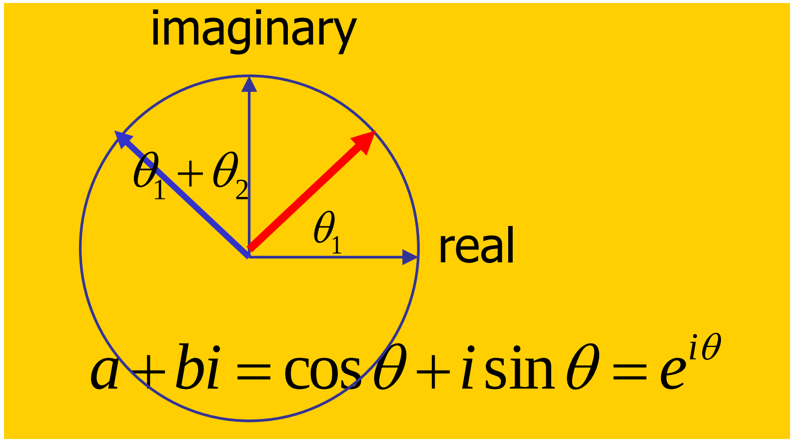






Quaternions

- Multiplication of two complex numbers ⇔



Rotation in 2D space

$$p_1 p_2 = (a_1 + b_1 i)(a_2 + b_2 i)$$
$$= e^{i\theta_1} e^{i\theta_2} = e^{i(\theta_1 + \theta_2)}$$

- Quaternions are 4D analogs of complex number
- Multiplication of two quaternions ⇔

Rotation in 3D space

Quaternions

- Quaternions are defined using one real part and three imaginary quantities, ***i***, ***j*** and ***k***

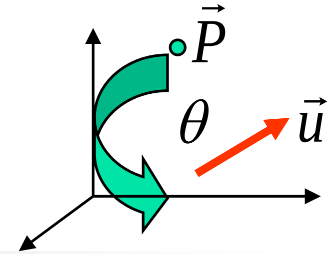
$$q = (s, \vec{v}) = s + v_1 i + v_2 j + v_3 k$$

$$i^2 = j^2 = k^2 = -1,$$

$$ij = -ji = k, \quad jk = -kj = i, \quad ki = -ik = j$$

$$|q| = \sqrt{s^2 + {v_1}^2 + {v_2}^2 + {v_3}^2}$$

Quaternions



- A rotation of a vector point $\mathrm{P} = (x, y, z)$ about the unit vector $\mathbf{u}$ by an angle θ can be computed using the quaternion

$$\mathbf{P} = (0, \mathbf{p}), \quad q = (s, \mathbf{v}) = (\cos\frac{\theta}{2}, \mathbf{u}\sin\frac{\theta}{2})$$

$$\boxed{\mathbf{P}_{rotated} = q \cdot \mathbf{P} \cdot q^{-1}} \quad \text{where} \quad q^{-1} = (s, -v)$$

When $q_1 = (s_1, \vec{v}_1)$ and $q_2 = (s_2, \vec{v}_2)$

$$q_1 \cdot q_2 = (s_1 s_2 - \vec{v}_1 \cdot \vec{v}_2,\ s_1\vec{v}_2 + s_2\vec{v}_1 + \vec{v}_1 \times \vec{v}_2)$$

$$\mathbf{P}_{\text{rotated}} = (0, \mathbf{p}_{rotated})$$

$$\mathbf{p}_{rotated} = s^2\mathbf{p} + \mathbf{v}(\mathbf{p}\cdot\mathbf{v}) + 2s(\mathbf{v}\times\mathbf{p}) + \mathbf{v}\times(\mathbf{v}\times\mathbf{p})$$

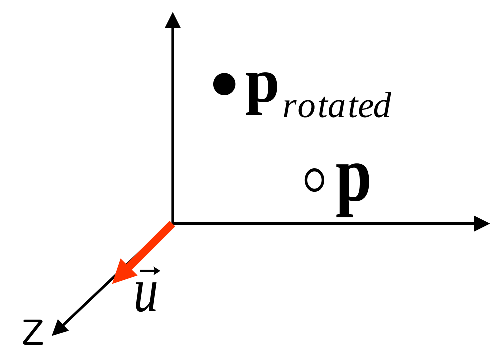
Quaternions

Quaternions for Rotation: $\mathbf{P} = (0, \mathbf{p}), \quad q = (s, \mathbf{v}) = (\cos\frac{\theta}{2}, \mathbf{u}\sin\frac{\theta}{2})$

$$\mathbf{p}_{rotated} = s^2\mathbf{p} + \mathbf{v}(\mathbf{p}\cdot\mathbf{v}) + 2s(\mathbf{v}\times\mathbf{p}) + \mathbf{v}\times(\mathbf{v}\times\mathbf{p})$$

[Example] Rotation about z-axis

$$s = \cos\frac{\theta}{2}, \qquad \mathbf{v} = (0, \quad 0, \quad 1)\sin\frac{\theta}{2}$$



Quaternion Rotation in a Matrix Form

Assuming that a unit quaternion has been created in the form: (s, a, b, c)

Then the quaternion can then be converted into a 4x4 rotation matrix using the following expression

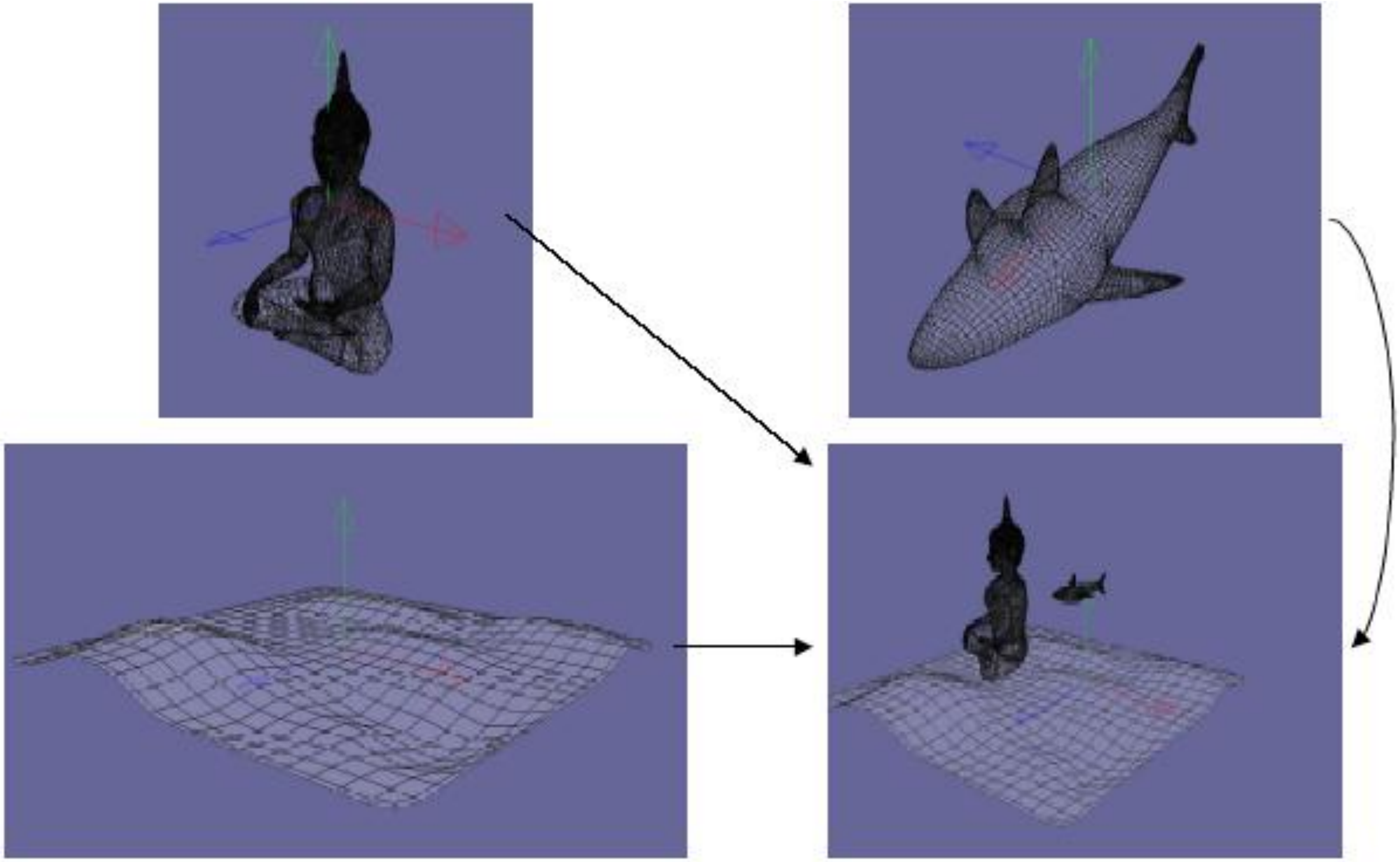
$$M_R(\theta) = \begin{bmatrix} 1-2b^2-2c^2 & 2ab-2sc & 2ac+2sb \\ 2ab+2sc & 1-2a^2-2c^2 & 2bc-2sa \\ 2ac-2sb & 2bc+2sa & 1-2a^2-2b^2 \end{bmatrix}$$

It is often necessary to have a quaternion rotation in a matrix form, e.g., to load onto graphics hardware for hardware vertex transformations.

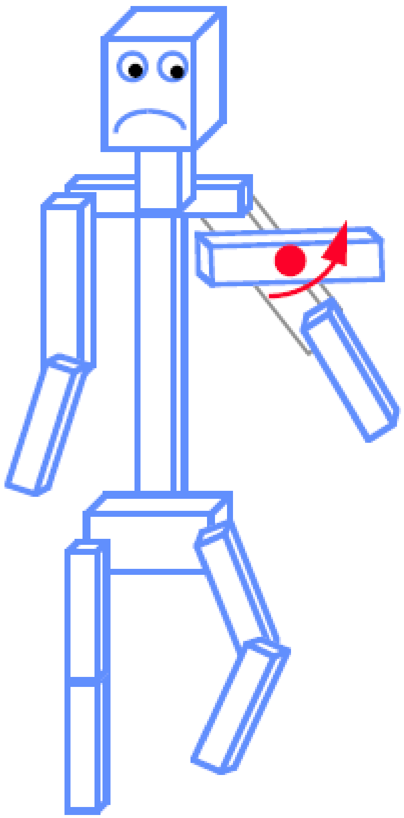
Quaternions

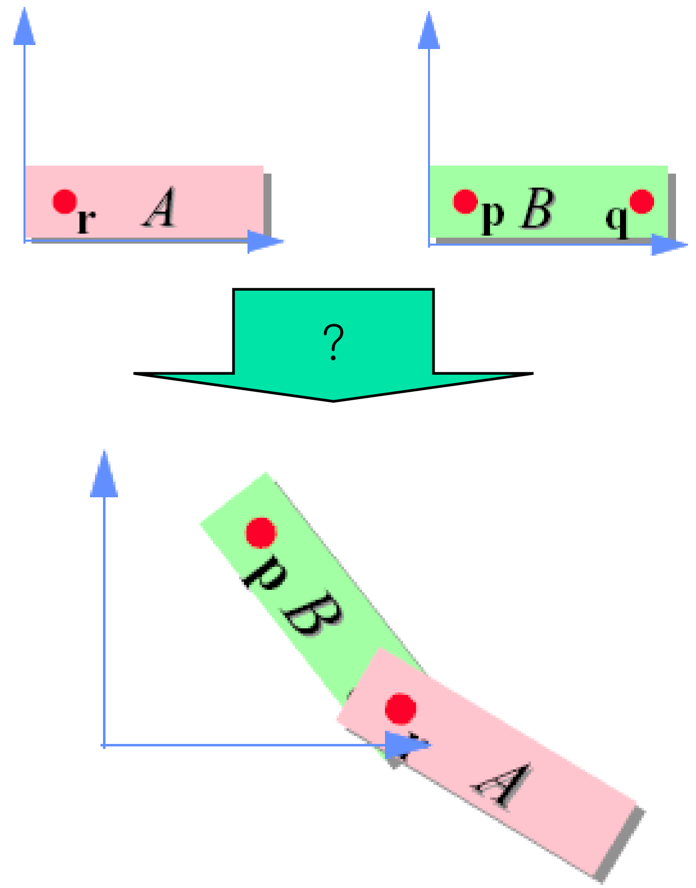
- Useful for animations
  - Independent definition of an axis of rotation and an angle
  - Smooth interpolation
  - No Gimble lock
- Far more complicated to read and conceptualize than Eular angle
- Interpolation can be expensive in practice

Modeling Transformation



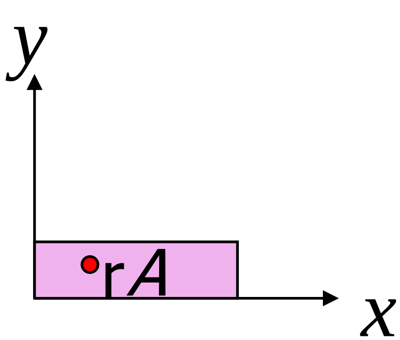
Modeling Transformation

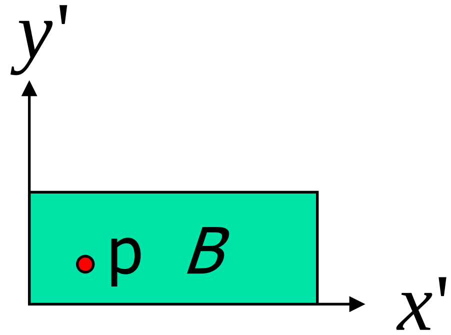




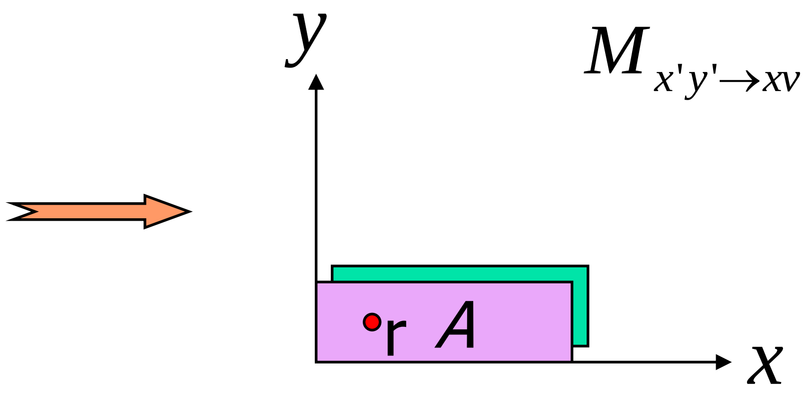
Modeling Transformation

Modeling Transformation

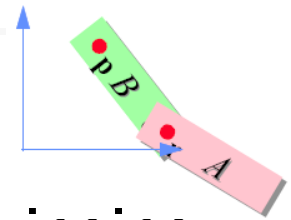


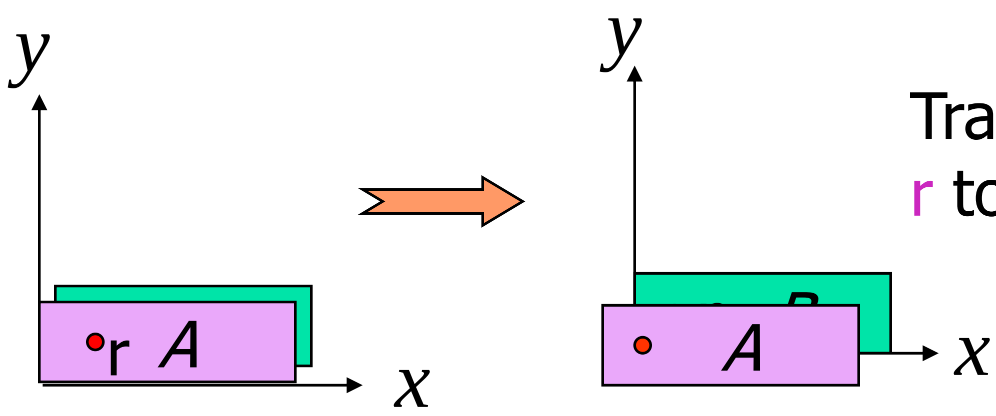


$$M_{x'y' \rightarrow xv} = Scale(x, y) \cdot T(x, y)$$

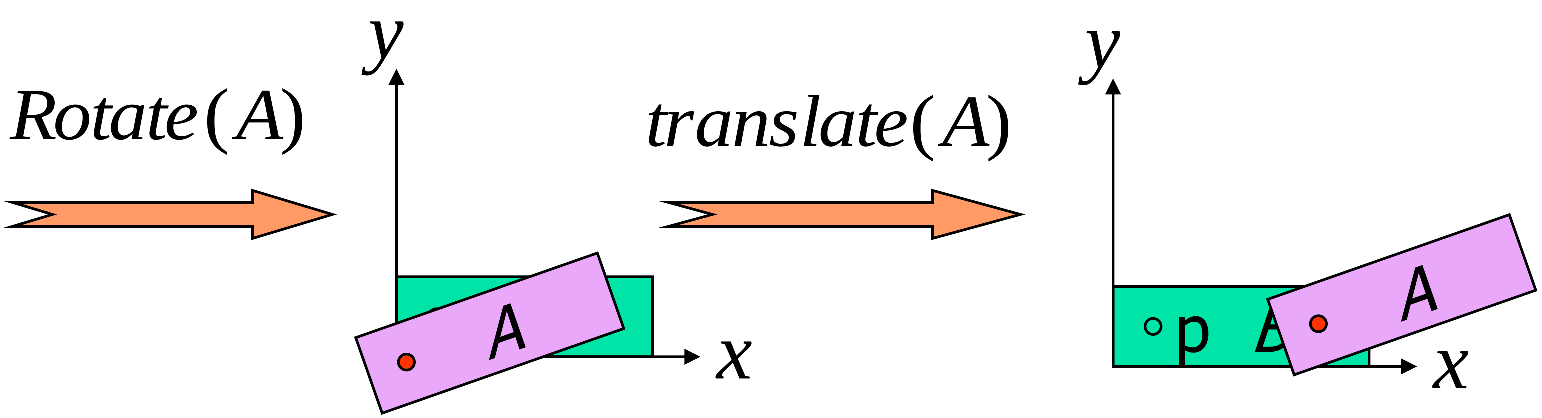


Modeling Transformation

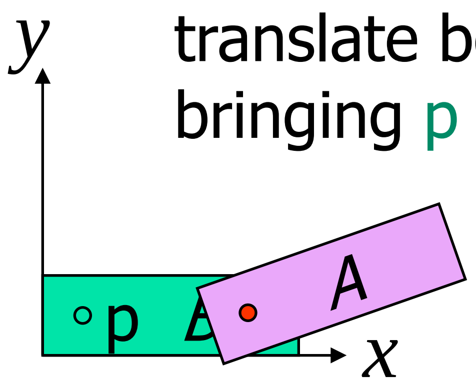




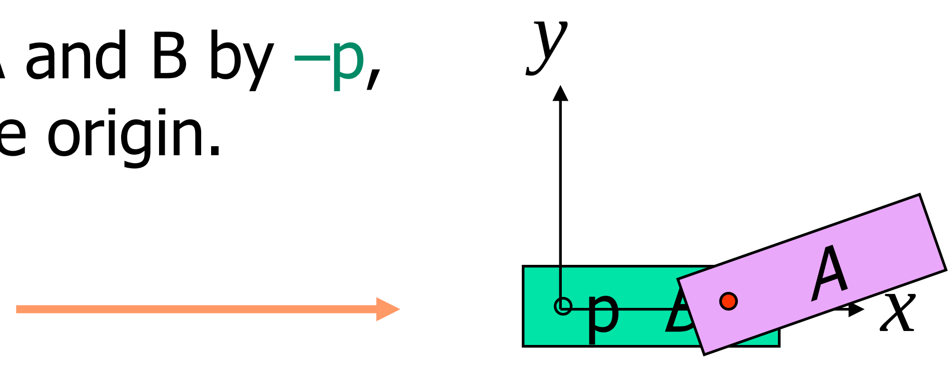
Translate by -r, bringing r to the origin.

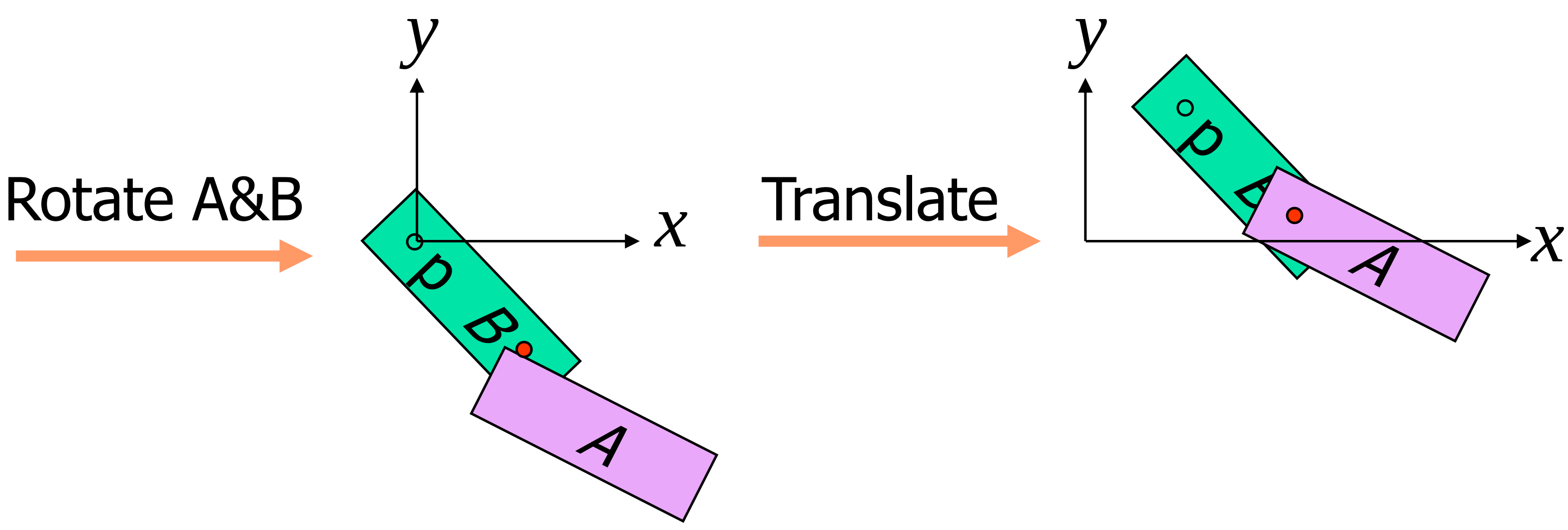


Modeling Transformation

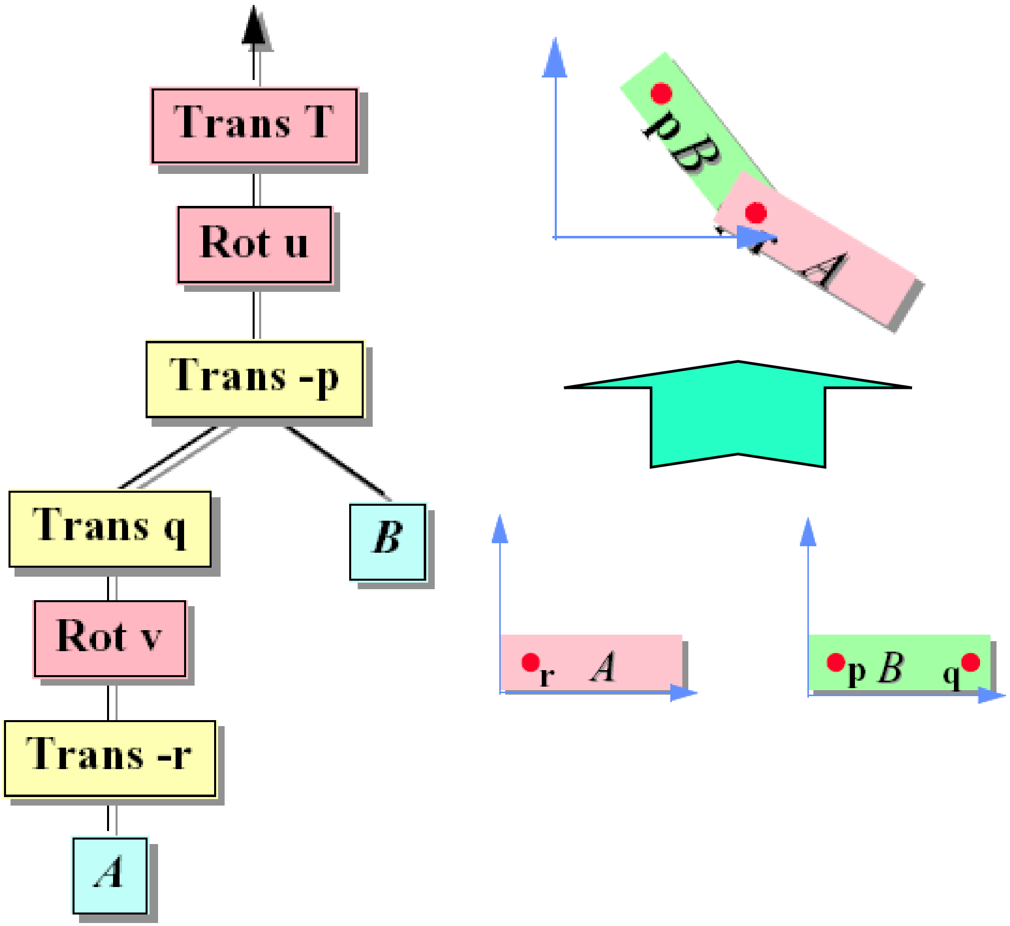


translate both A and B by −p, bringing p to the origin.





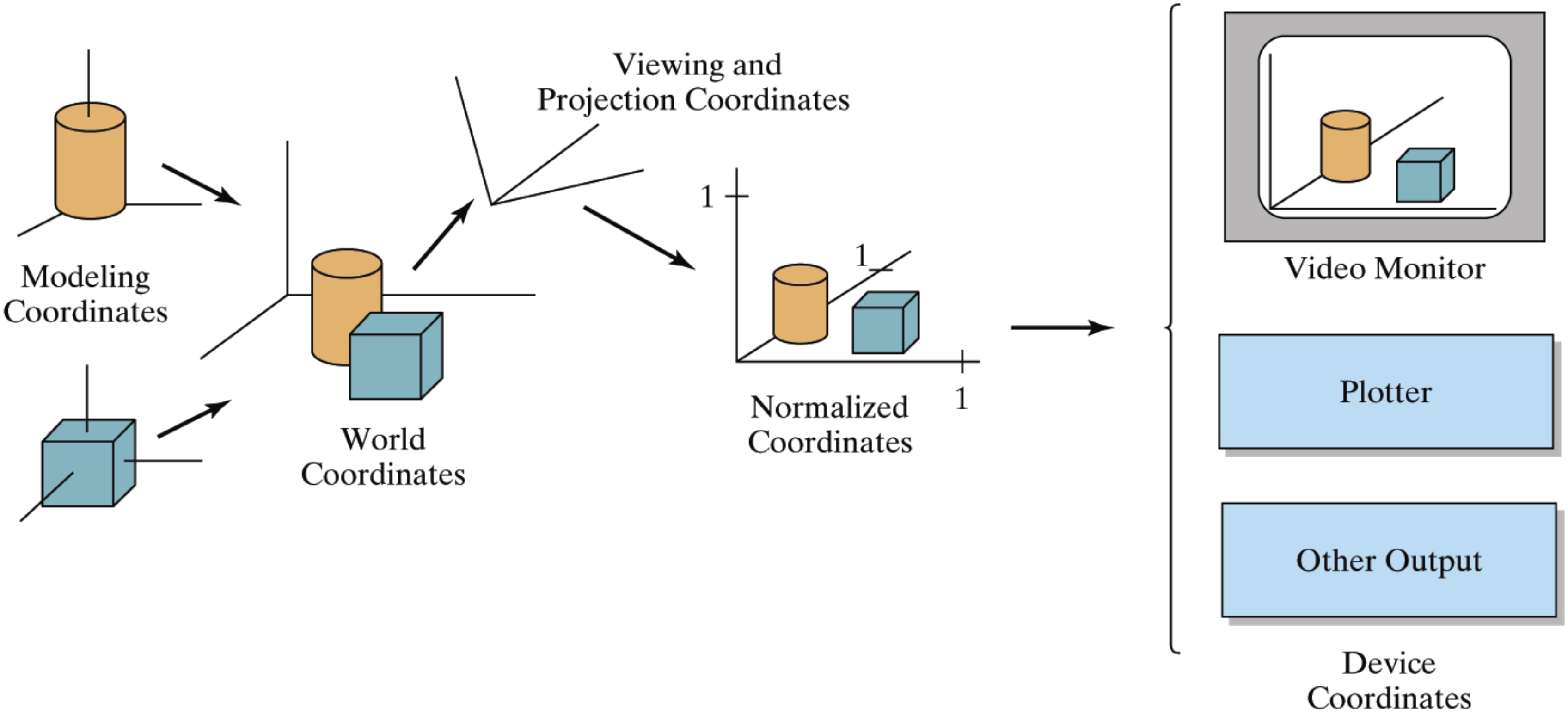
Modeling Transformation



Trace of Opengl calls

```
glLoadIdentity();
glOrtho(…);
glPushMatrix();
glTranslatef(Tx,Ty,0);
glRotatef(u,0,0,1);
glTranslatef(-px,-py,0);
glPushMatrix();
glTranslatef(qx,qy,0);
glRotatef(v,0,0,1);
glTranslatef(-rx,-ry,0);
Draw(A);
glPopMatrix();
Draw(B);
glPopMatrix();
```

What next!



Modeling
Coordinates
World
Coordinates
Viewing and
Projection Coordinates
1
1
1
Normalized
Coordinates
Video Monitor
Plotter
Other Output
Device
Coordinates