

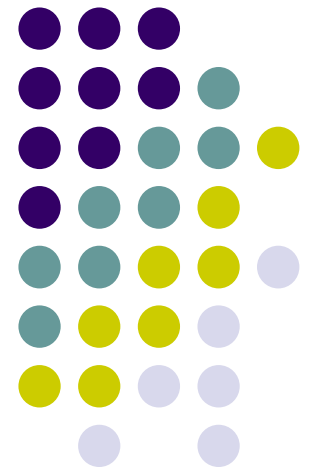
# Computer Graphics (543)

## Lecture 5 (Part 2): Transformations (Rotations and Matrix Concatenation)

---

Prof Emmanuel Agu

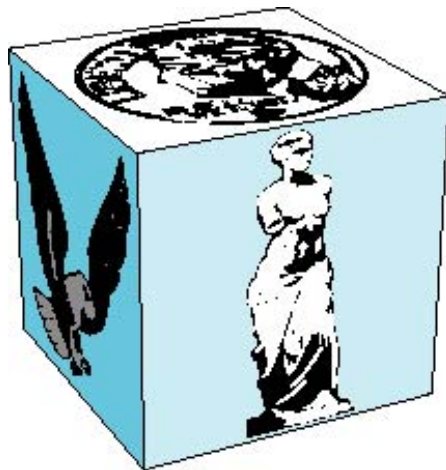
*Computer Science Dept.  
Worcester Polytechnic Institute (WPI)*



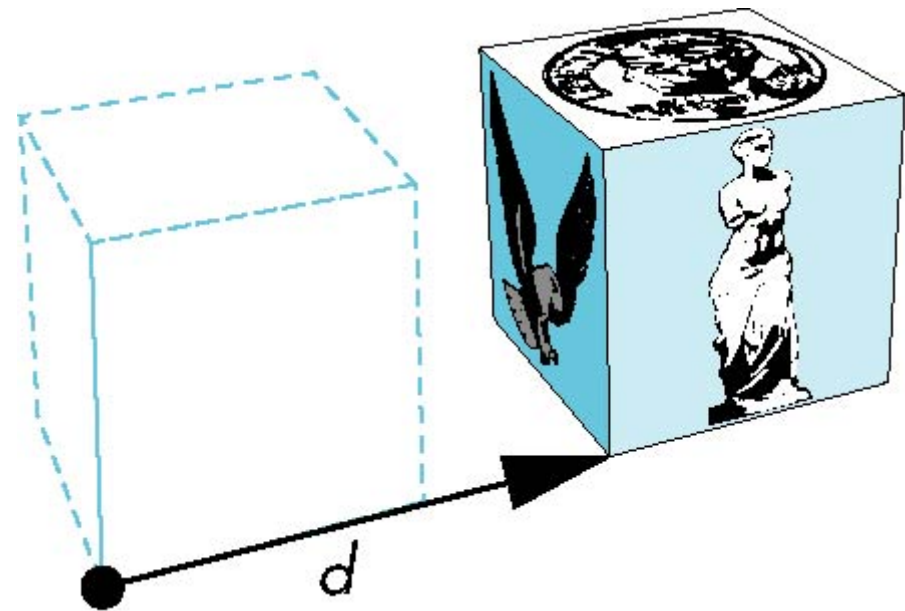


## Recall: 3D Translation

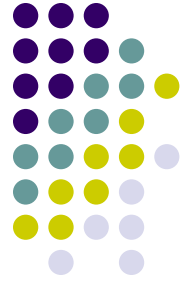
- **Translate:** Move each vertex by same distance  $\mathbf{d} = (t_x, t_y, t_z)$



object



**translation:** every vertex displaced  
by same vector

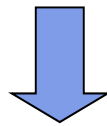


# Recall: 3D Translation Matrix

▪ In 3D :

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix}$$

Translate(tx,ty,tz)



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

▪ Where:  $x' = x \cdot 1 + y \cdot 0 + z \cdot 0 + t_x \cdot 1 = x + t_x, \dots$  etc



# Recall: Scaling

**Scale:** Expand or contract along each axis (fixed point of origin)

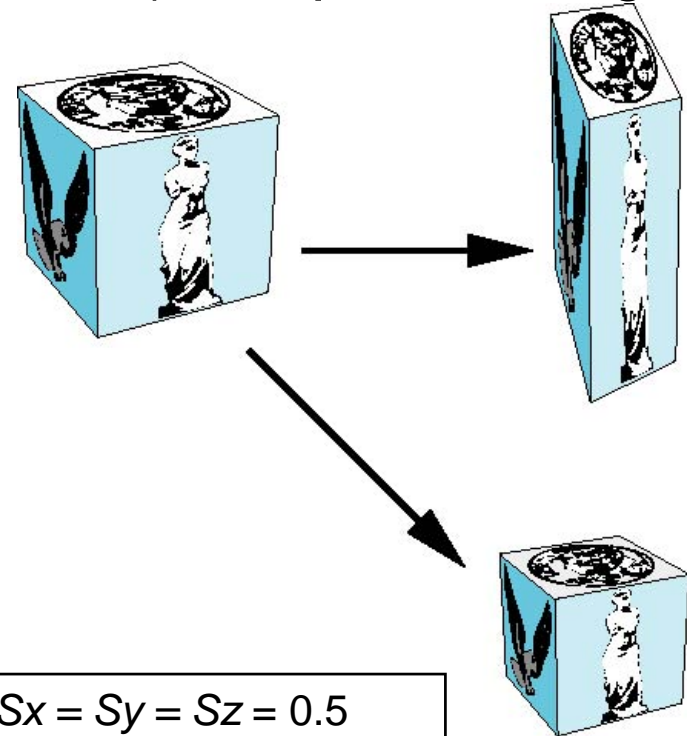
$$S = S(s_x, s_y, s_z)$$

$$p' = Sp$$

$$x' = s_x x$$

$$y' = s_y y$$

$$z' = s_z z$$



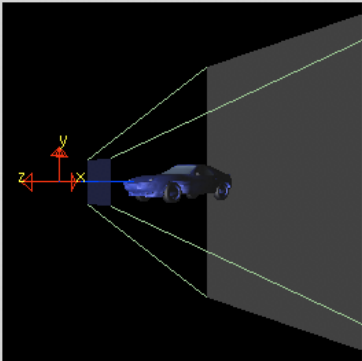
$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

•**Example:**  $S_x = S_y = S_z = 0.5$   
scales big cube (sides = 1)  
to small cube ( sides = 0.5)


# Nate Robbins Translate, Scale Rotate Demo



World-space view



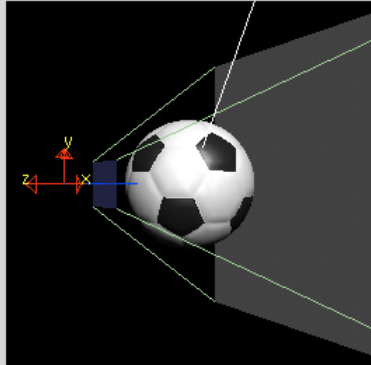
Screen-space view




Command manipulation window

```
glTranslatef( 0.00 , 0.00 , 0.00 );  
glRotatef( 0.0 , 0.00 , 1.00 , 0.00 );  
glScalef( 1.00 , 1.00 , 1.00 );  
glBegin( ... );  
...  
Click on the arguments and move the mouse to modify values.
```

World-space view



Screen-space view



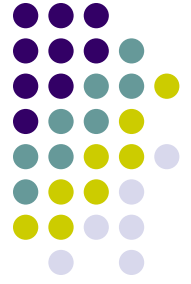
Command manipulation window

```
GLfloat pos[4] = { 1.50 , 1.00 , 1.00 , 0.00 };  
gluLookAt( 0.00 , 0.00 , 2.00 , <- eye  
          0.00 , 0.00 , 0.00 , <- center  
          0.00 , 1.00 , 0.00 ); <- up  
glLightfv(GL_LIGHT0, GL_POSITION, pos);  
Click on the arguments and move the mouse to modify values.
```



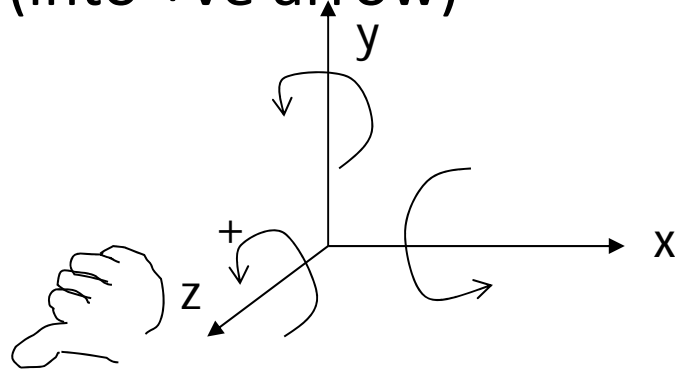
## Rotating in 3D

- Many degrees of freedom. Rotate about what axis?
- 3D rotation: about a defined axis
- Different transform matrix for:
  - Rotation about x-axis
  - Rotation about y-axis
  - Rotation about z-axis

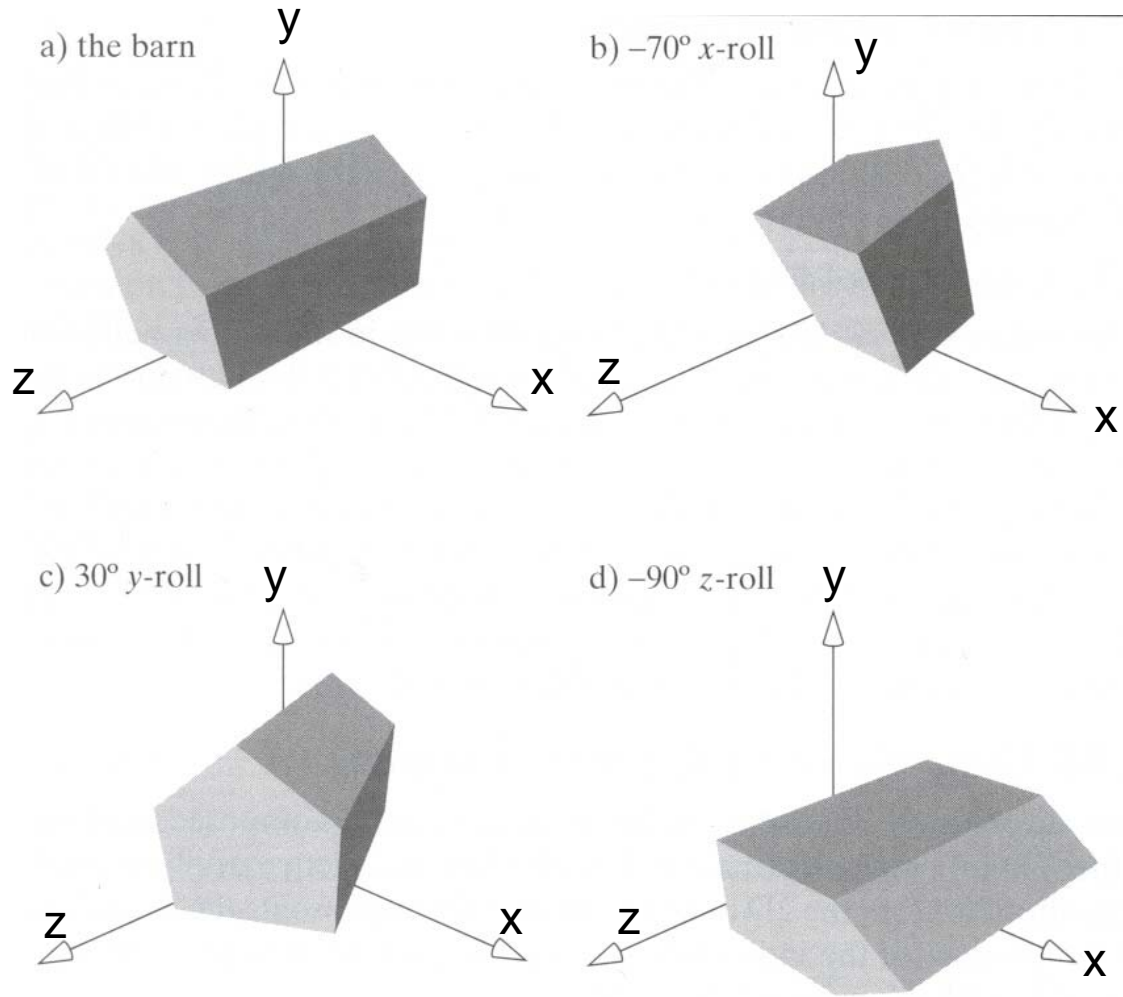
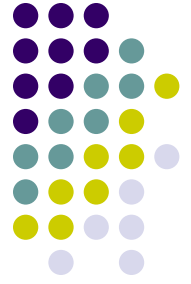


# Rotating in 3D

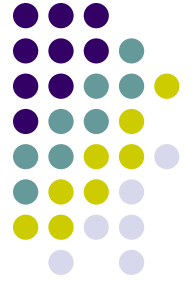
- New terminology
  - **X-roll:** rotation about x-axis
  - **Y-roll:** rotation about y-axis
  - **Z-roll:** rotation about z-axis
- Which way is +ve rotation
  - Look in –ve direction (into +ve arrow)
  - CCW is +ve rotation



# Rotating in 3D







# Rotating in 3D

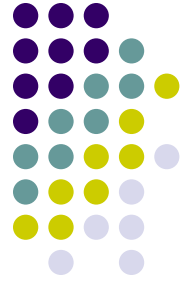
- For a rotation angle,  $\beta$  about an axis
- Define:

$$c = \cos(\beta) \qquad s = \sin(\beta)$$

x-roll or (RotateX)

$$R_x(\beta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & c & -s & 0 \\ 0 & s & c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

# Rotating in 3D



y-roll (or RotateY)

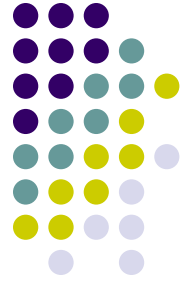
$$R_y(\beta) = \begin{pmatrix} c & 0 & s & 0 \\ 0 & 1 & 0 & 0 \\ -s & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

## Rules:

- Write 1 in rotation row, column
- Write 0 in the other rows/columns
- Write c,s in rect pattern

z-roll (or RotateZ)

$$R_z(\beta) = \begin{pmatrix} c & -s & 0 & 0 \\ s & c & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



## Example: Rotating in 3D

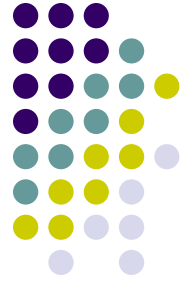
**Question:** Using **y-roll** equation, rotate  $P = (3,1,4)$  by 30 degrees:

**Answer:**  $c = \cos(30) = 0.866$ ,  $s = \sin(30) = 0.5$ , and

$$Q = \begin{pmatrix} c & 0 & s & 0 \\ 0 & 1 & 0 & 0 \\ -s & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 3 \\ 1 \\ 4 \\ 1 \end{pmatrix} = \begin{pmatrix} 4.6 \\ 1 \\ 1.964 \\ 1 \end{pmatrix}$$

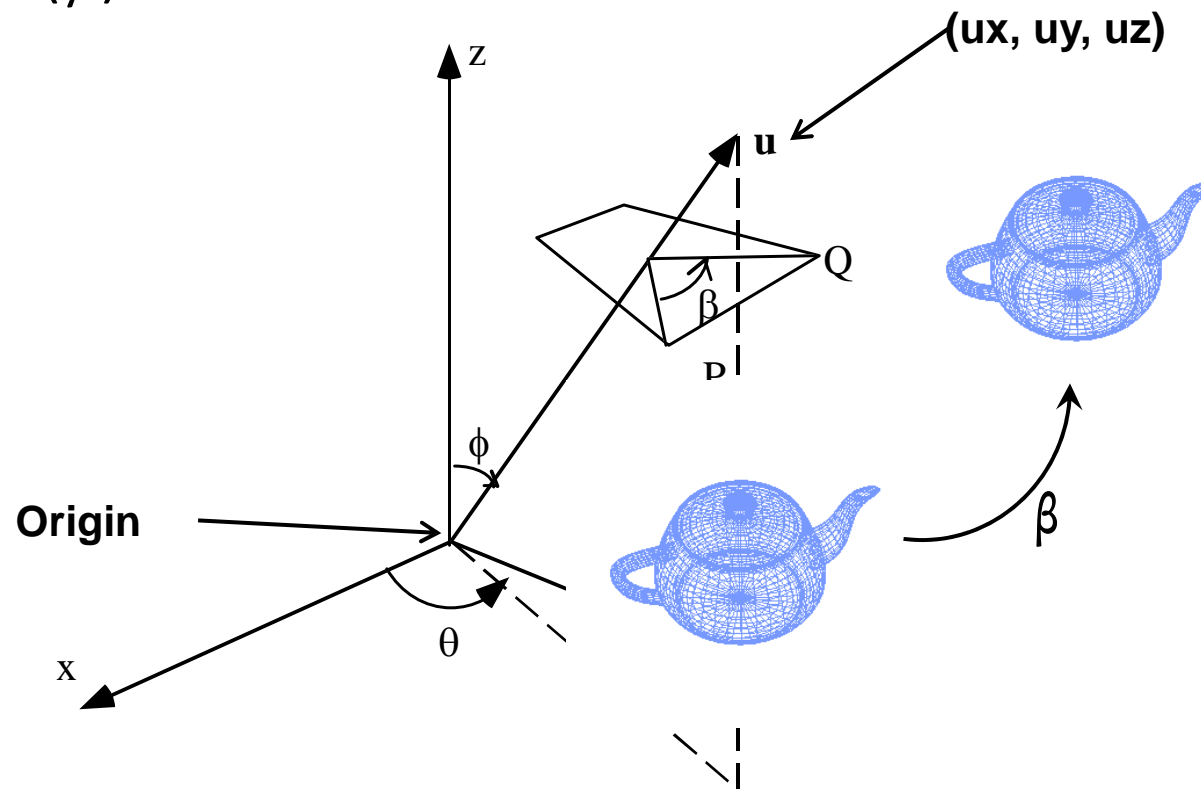
$$\text{Line 1: } 3.c + 1.0 + 4.s + 1.0$$

$$= 3 \times 0.866 + 4 \times 0.5 = 4.6$$

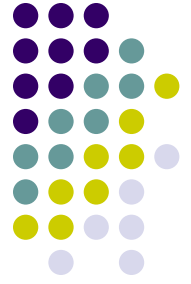


# 3D Rotation

- **Rotate(angle, ux, uy, uz):** rotate by angle  $\beta$  about an **arbitrary** axis (a vector) passing through **origin** and **(ux, uy, uz)**
- **Note:** Angular position of **u** specified as azimuth ( $\theta$ ) and latitude ( $\phi$ )



# Approach 1: 3D Rotation About Arbitrary Axis



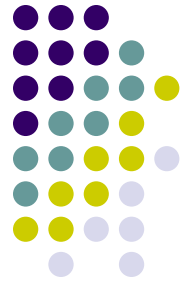
- Can compose arbitrary rotation as combination of:
  - X-roll (by an angle  $\beta_1$ )
  - Y-roll (by an angle  $\beta_2$ )
  - Z-roll (by an angle  $\beta_3$ )

$$M = R_z(\beta_3)R_y(\beta_2)R_x(\beta_1)$$



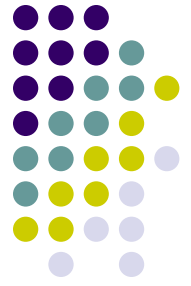
Read in reverse order

# Approach 1: 3D Rotation About Arbitrary Axis

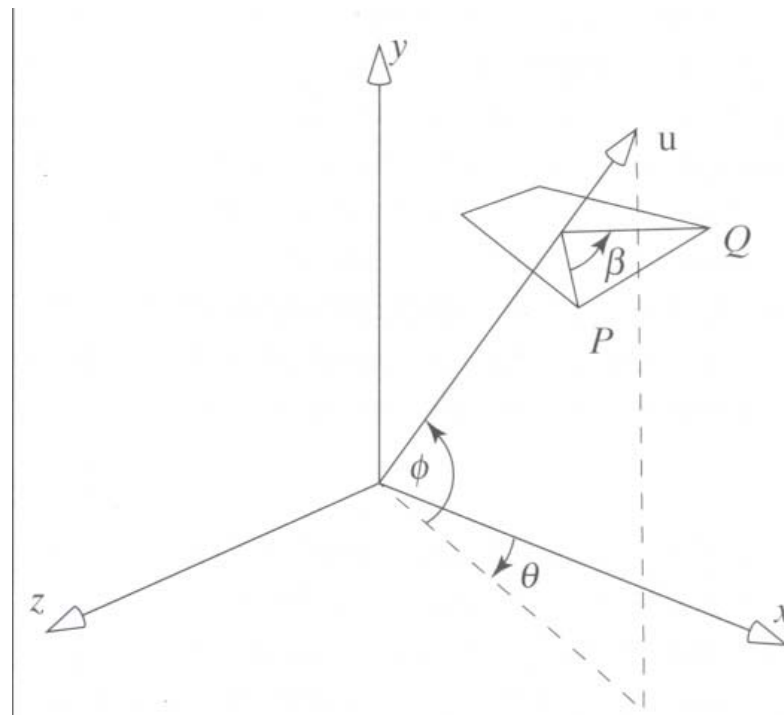


- **Classic:** use Euler's theorem
- **Euler's theorem:** any sequence of rotations = one rotation about some axis
- Want to rotate  $\beta$  about arbitrary axis  $\mathbf{u}$  through origin
- Our approach:
  1. Use two rotations to align  $\mathbf{u}$  and  $\mathbf{x}$ -axis
  2. Do  $\mathbf{x}$ -roll through angle  $\beta$
  3. Negate two previous rotations to de-align  $\mathbf{u}$  and  $\mathbf{x}$ -axis

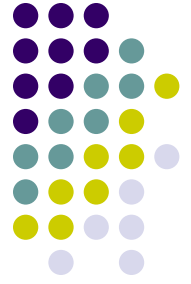
# Approach 1: 3D Rotation using Euler Theorem



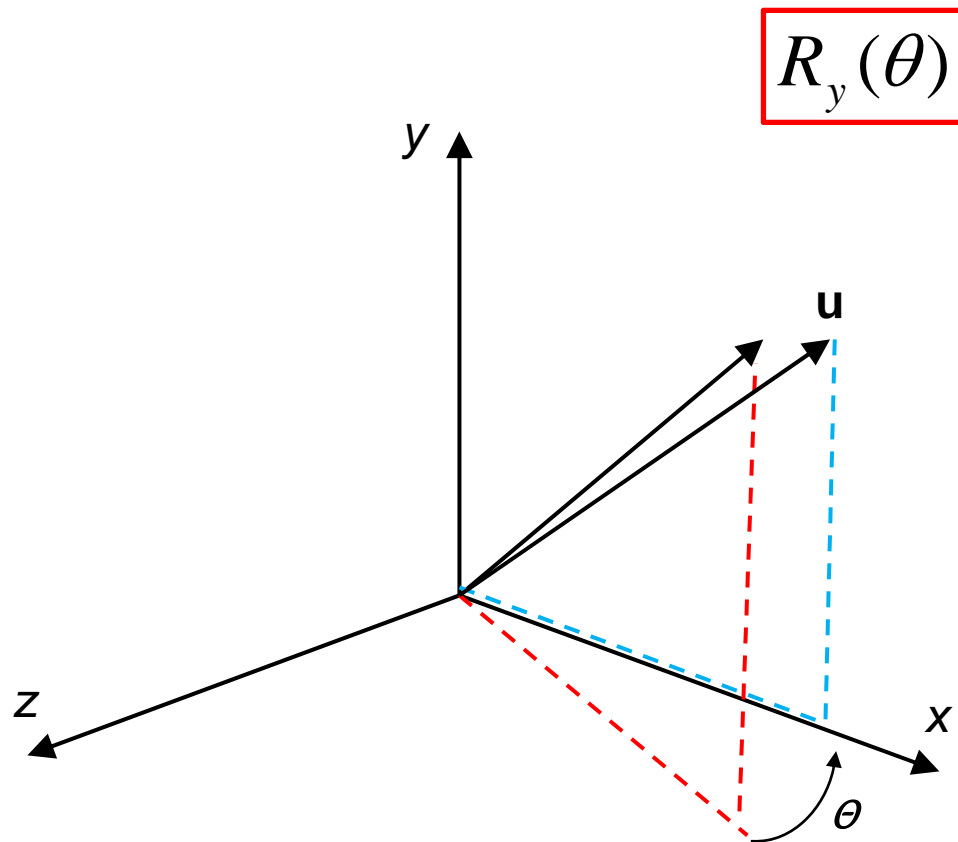
- **Note:** Angular position of  $\mathbf{u}$  specified as azimuth ( $\theta$ ) and latitude ( $\phi$ )
- First try to align  $\mathbf{u}$  with x axis



# Approach 1: 3D Rotation using Euler Theorem



- **Step 1:** Do y-roll to line up rotation axis with x-y plane



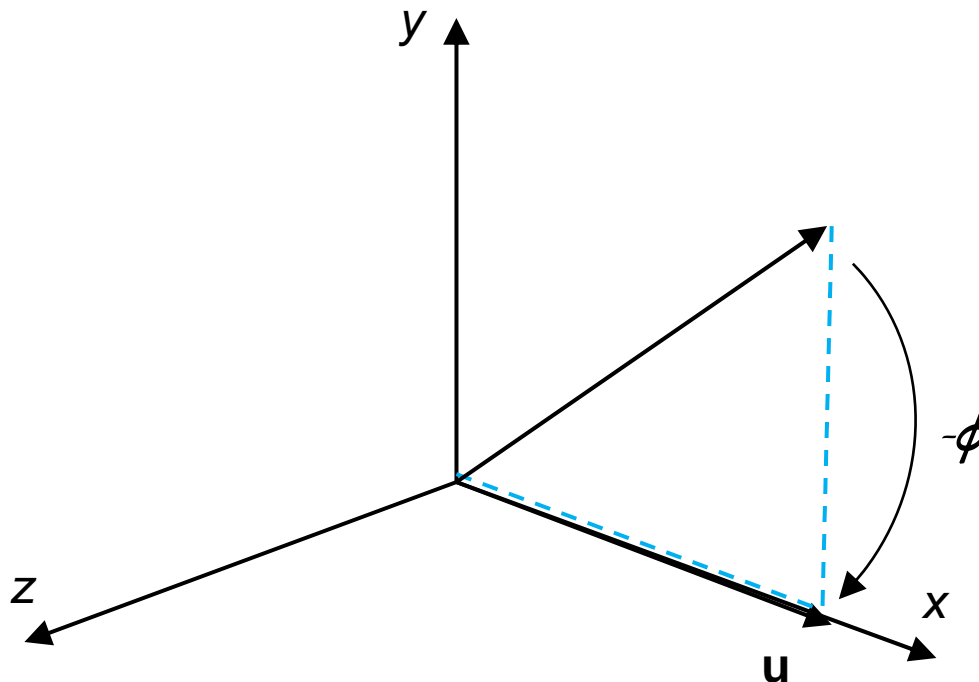


# Approach 1: 3D Rotation using Euler Theorem

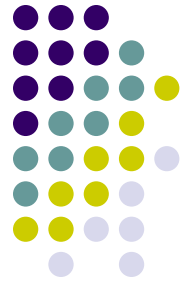


- **Step 2:** Do z-roll to line up rotation axis with x axis

$$R_z(-\phi)R_y(\theta)$$

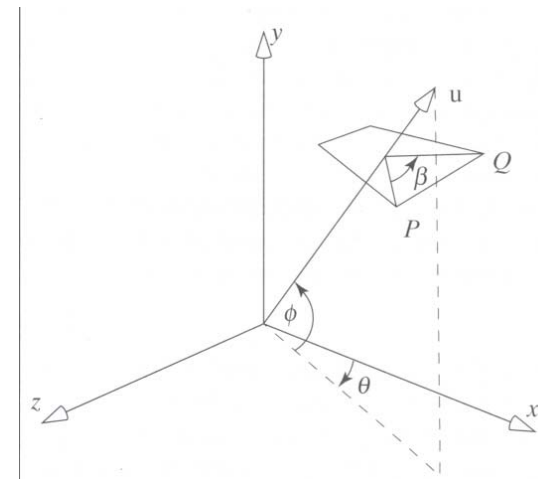
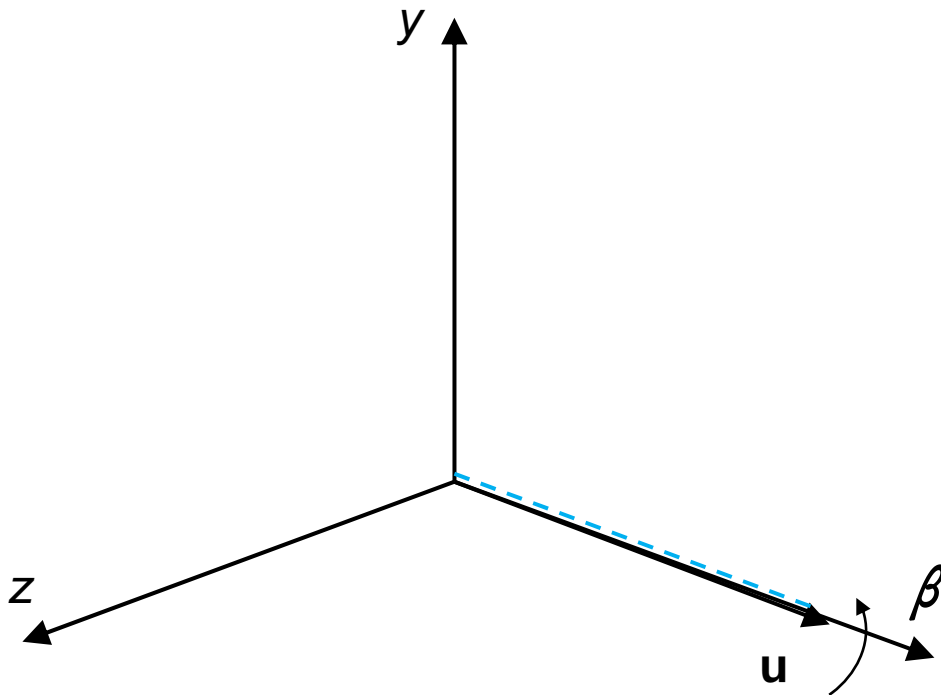


# Approach 1: 3D Rotation using Euler Theorem



- **Remember:** Our goal is to do rotation by  $\beta$  around  $\mathbf{u}$
- But axis  $\mathbf{u}$  is now lined up with x axis. So,
- **Step 3:** Do x-roll by  $\beta$  around axis  $\mathbf{u}$

$$R_x(\beta)R_z(-\phi)R_y(\theta)$$

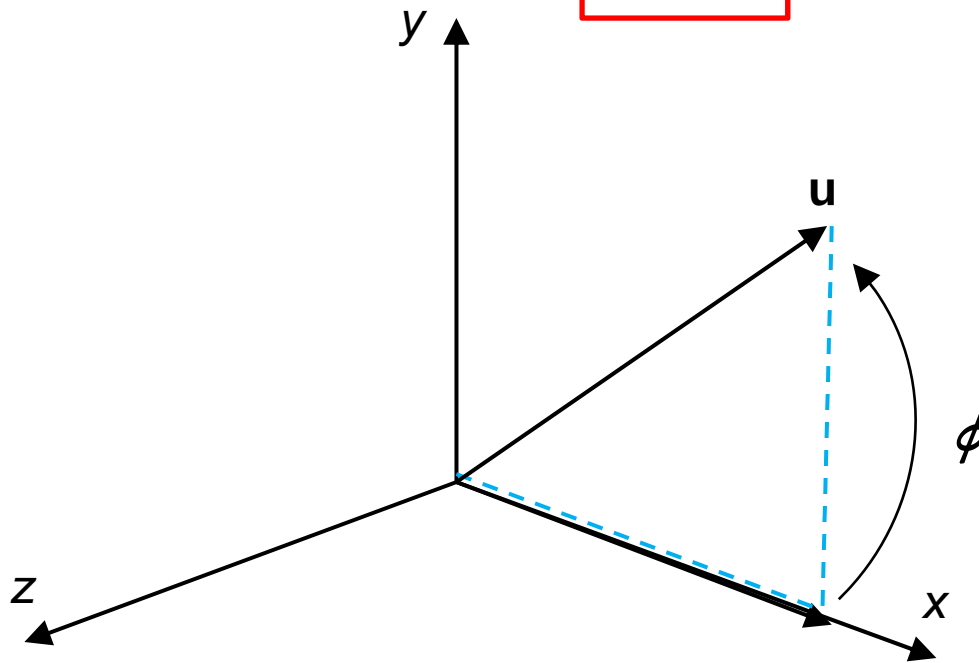


# Approach 1: 3D Rotation using Euler Theorem

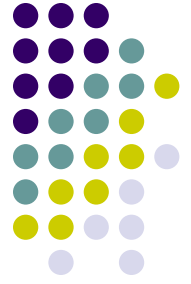


- Next 2 steps are to return vector  $\mathbf{u}$  to original position
- **Step 4:** Do z-roll in x-y plane

$$R_z(\phi)R_x(\beta)R_z(-\phi)R_y(\theta)$$

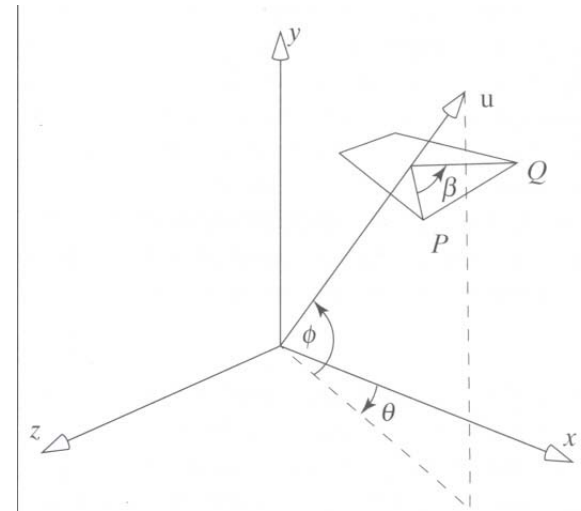
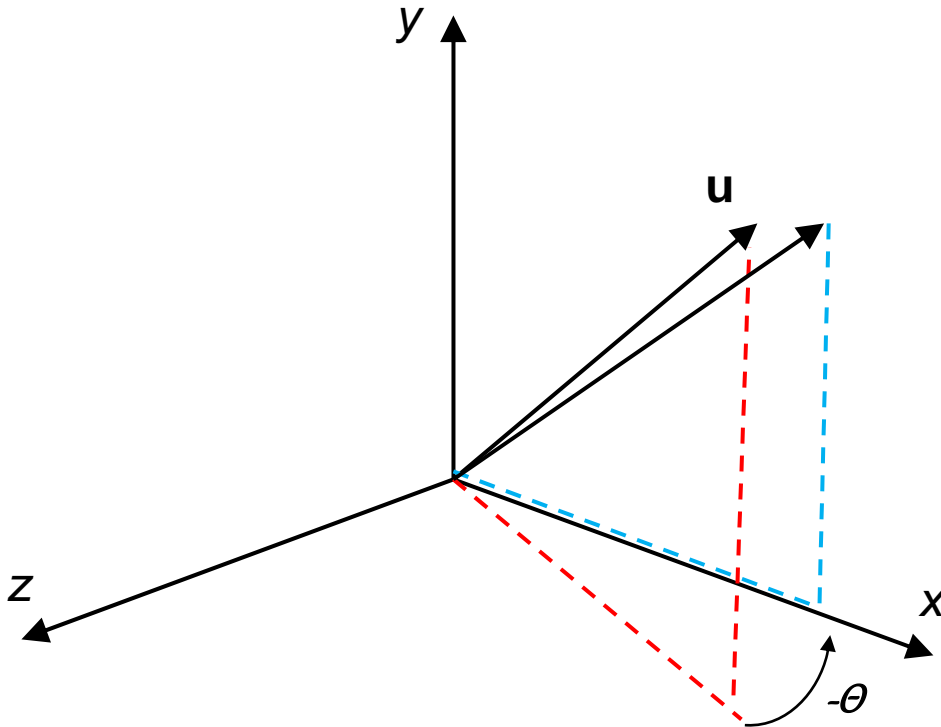


# Approach 1: 3D Rotation using Euler Theorem

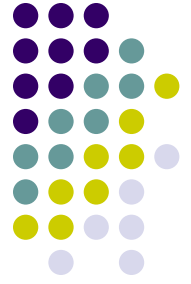


- **Step 5:** Do y-roll to return  $\mathbf{u}$  to original position

$$R_u(\beta) = R_y(-\theta)R_z(\phi)R_x(\beta)R_z(-\phi)R_y(\theta)$$



# Approach 2: Rotation using Quaternions



- Extension of imaginary numbers from 2 to 3 dimensions
- Requires 1 real and 3 imaginary components **i**, **j**, **k**

$$q = q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k}$$

- Quaternions can express rotations on sphere smoothly and efficiently

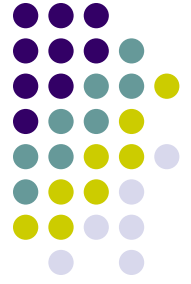
# Approach 2: Rotation using Quaternions



- Derivation skipped! Check answer
- Solution has lots of symmetry

$$R(\beta) = \begin{pmatrix} c + (1-c)\mathbf{u}_x^2 & (1-c)\mathbf{u}_y\mathbf{u}_x + s\mathbf{u}_z & (1-c)\mathbf{u}_z\mathbf{u}_x + s\mathbf{u}_y & 0 \\ (1-c)\mathbf{u}_x\mathbf{u}_y + s\mathbf{u}_z & c + (1-c)\mathbf{u}_y^2 & (1-c)\mathbf{u}_z\mathbf{u}_y - s\mathbf{u}_x & 0 \\ (1-c)\mathbf{u}_x\mathbf{u}_z - s\mathbf{u}_y & (1-c)\mathbf{u}_y\mathbf{u}_z - s\mathbf{u}_x & c + (1-c)\mathbf{u}_z^2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$c = \cos(\beta) \quad s = \sin(\beta)$$



# Inverse Matrices

- Can compute inverse matrices by general formulas
- But easier to use simple geometric observations
  - Translation:  $\mathbf{T}^{-1}(d_x, d_y, d_z) = \mathbf{T}(-d_x, -d_y, -d_z)$
  - Scaling:  $\mathbf{S}^{-1}(s_x, s_y, s_z) = \mathbf{S} ( 1/s_x, 1/s_y, 1/s_z )$
  - Rotation:  $\mathbf{R}^{-1}(q) = \mathbf{R}(-q)$ 
    - Holds for any rotation matrix



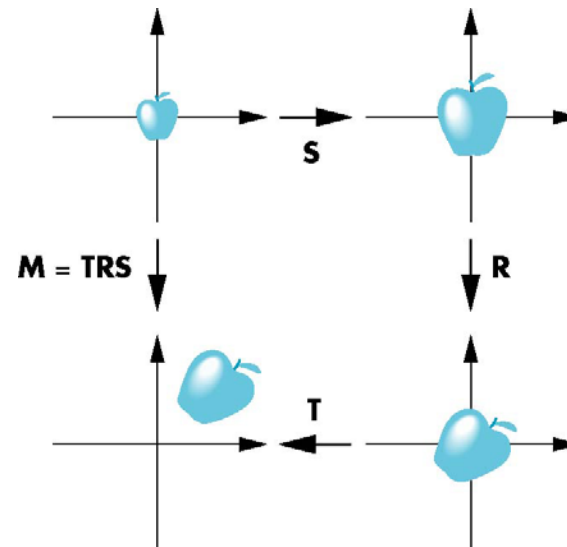
# Instancing

- During modeling, often start with simple object centered at origin, aligned with axis, and unit size
- Can declare one copy of each shape in scene
- Then apply *instance transformation* to its vertices to

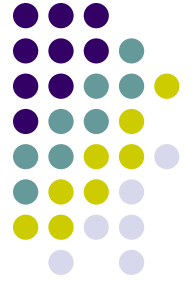
Scale

Orient

Locate







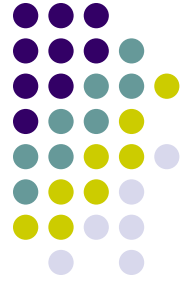
# Concatenating Transformations

- Can form arbitrary affine transformation matrices by multiplying rotation, translation, and scaling matrices
- General form:

$$\mathbf{M1} \times \mathbf{M2} \times \mathbf{M3} \times \mathbf{P}$$

where M1, M2, M3 are transform matrices applied to P

- Be careful with the order!!
- For example:
  - Translate by (5,0) then rotate 60 degrees **NOT** same as
  - Rotate by 60 degrees then translate by (5,0)



# Concatenation Order

- Note that matrix on right is first applied
- Mathematically, the following are equivalent

$$\mathbf{p}' = \mathbf{ABCp} = \mathbf{A}(\mathbf{B}(\mathbf{Cp}))$$

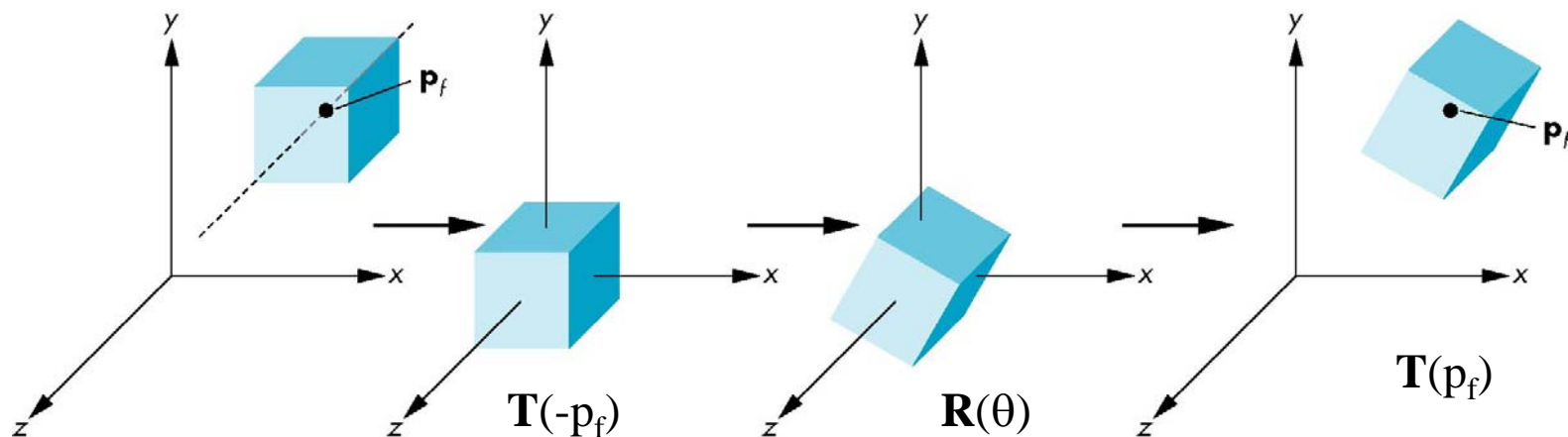
- **Efficient!!**
  - Matrix  $\mathbf{M}=\mathbf{ABC}$  is composed, then multiplied by many vertices
  - Cost of forming matrix  $\mathbf{M}=\mathbf{ABC}$  not significant compared to cost of multiplying  $(\mathbf{ABC})\mathbf{p}$  for many vertices  $\mathbf{p}$  one by one

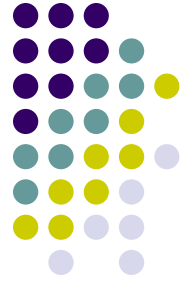
# Rotation About Arbitrary Point other than the Origin



- Default rotation matrix is about origin
- How to rotate about any arbitrary point (Not origin)?
  - Move fixed point to origin  $\mathbf{T}(-p_f)$
  - Rotate  $\mathbf{R}(\theta)$
  - Move fixed point back  $\mathbf{T}(p_f)$

**So,  $\mathbf{M} = \mathbf{T}(p_f) \mathbf{R}(\theta) \mathbf{T}(-p_f)$**

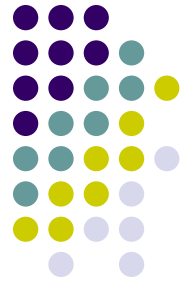




# Scale about Arbitrary Center

- Similar, default scaling is about origin
- To scale about arbitrary point  $P = (P_x, P_y, P_z)$  by  $(S_x, S_y, S_z)$ 
  1. Translate object by  $T(-P_x, -P_y, -P_z)$  so  $P$  coincides with origin
  2. Scale the object by  $(S_x, S_y, S_z)$
  3. Translate object back:  $T(P_x, P_y, P_z)$
- In matrix form:  $T(P_x, P_y, P_z) (S_x, S_y, S_z) T(-P_x, -P_y, -P_z) * P$

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & P_x \\ 0 & 1 & 0 & P_y \\ 0 & 0 & 1 & P_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & -P_x \\ 0 & 1 & 0 & -P_y \\ 0 & 0 & 1 & -P_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$



## References

- Angel and Shreiner, Chapter 3
- Hill and Kelley, Computer Graphics Using OpenGL, 3<sup>rd</sup> edition