CS 543:
Computer Graphics

3D Transformations

Robert W. Lindeman
Associate Professor
Interactive Media & Game Development
Department of Computer Science
Worcester Polytechnic Institute
gogo@wpi.edu

(with lots of help from Prof. Emmanuel Agu :-)

Introduction to Transformations

- A *transformation* changes an object's
  - Size (scaling)
  - Position (translation)
  - Orientation (rotation)
  - Shape (shear)
- Previously developed 2D or *(x, y)*
- Now we extend to 3D *(x, y, z)* case
- Transform object by applying sequence of matrix multiplications to 3D object vertices

Point Representation

- Previously, point in 2D as column matrix

$$\begin{pmatrix} x \\ y \end{pmatrix} \rightarrow \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

- Now, extending to 3D, add z-component

$$\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \quad \text{or} \quad P = \begin{pmatrix} P_x \\ P_y \\ P_z \\ 1 \end{pmatrix}$$

Transforms in 3D

- 2D: 3x3 matrix multiplication
- 3D: 4x4 matrix multiplication in homogenous coordinates
- Recall
  - Transform object = transform each vertex
- General form:

$$M = \begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix} \xrightarrow{\text{Transform of } P} \begin{pmatrix} Q_x \\ Q_y \\ Q_z \\ 1 \end{pmatrix} = M \begin{pmatrix} P_x \\ P_y \\ P_z \\ 1 \end{pmatrix}$$

Recall: 3x3 2D Translation Matrix

- Previously, in 2D

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

$$\Downarrow$$

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

4x4 3D Translation Matrix

- Now, in 3D

OpenGL:

`glTranslated(` t_x`,` t_y`,` t_z `);`

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

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

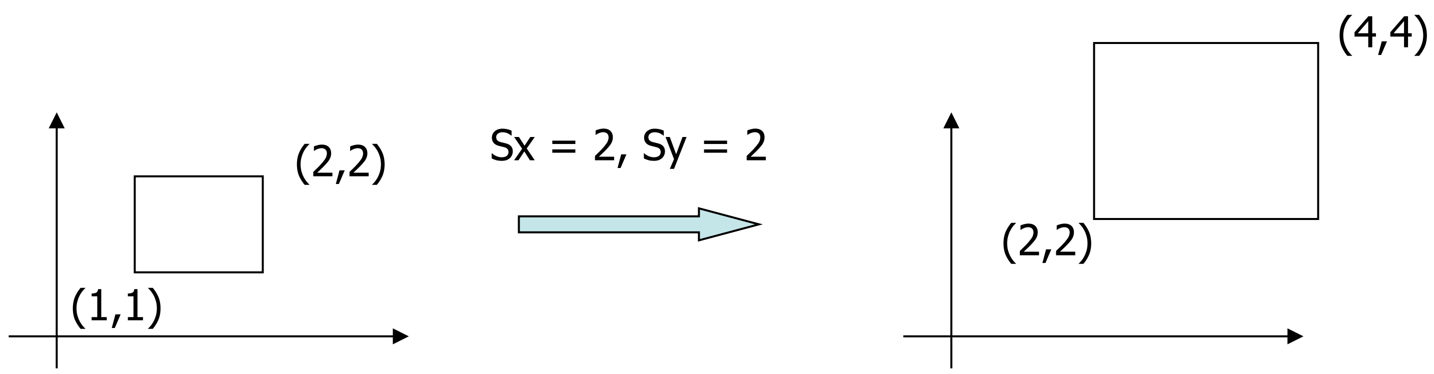
- Where: $x' = x*1 + y*0 + z*0 + t_x*1 = x + t_x$, … *etc.*

2D Scaling

- Scale: Alter object size by scaling factor (sx, sy). *i.e.*,

$$x' = x * Sx$$
$$y' = y * Sy$$

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} Sx & 0 \\ 0 & Sy \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$



Recall: 3x3 2D Scaling Matrix

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} Sx & 0 \\ 0 & Sy \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

↓

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

4x4 3D Scaling Matrix

- Example:
  - If $Sx = Sy = Sz = 0.5$
  - Can scale:
  - big cube (sides = 1) to small cube (sides = 0.5)
  - 2D: square, 3D cube

OpenGL:

```
glScaled( Sx, Sy, Sz );
```

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

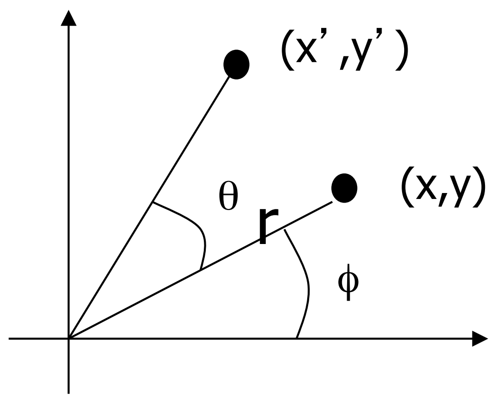
$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Example: OpenGL Table Leg

```
// define table leg
//------------------
void tableLeg( double thick, double len )  {
  glPushMatrix( );
    glTranslated( 0, ( len * 0.5 ), 0);
    glScaled( thick, len, thick );
    glutSolidCube( 1.0 );
  glPopMatrix( );
}
```

Recall: 3x3 2D Rotation Matrix

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$



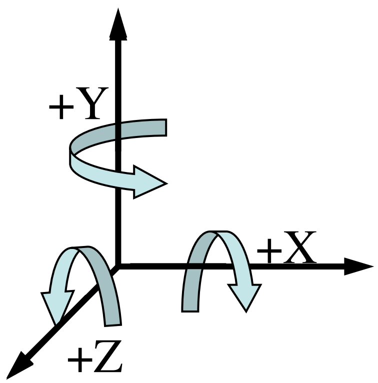
$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

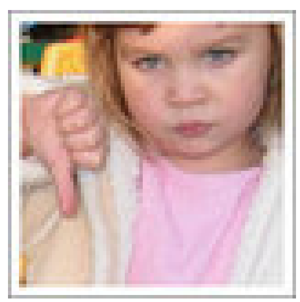
Rotating in 3D

- Cannot do mindless conversion like before
- Why?
  - 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
- New terminology
  - Pitch: rotation about x-axis
  - Yaw: rotation about y-axis
  - Roll: rotation about z-axis

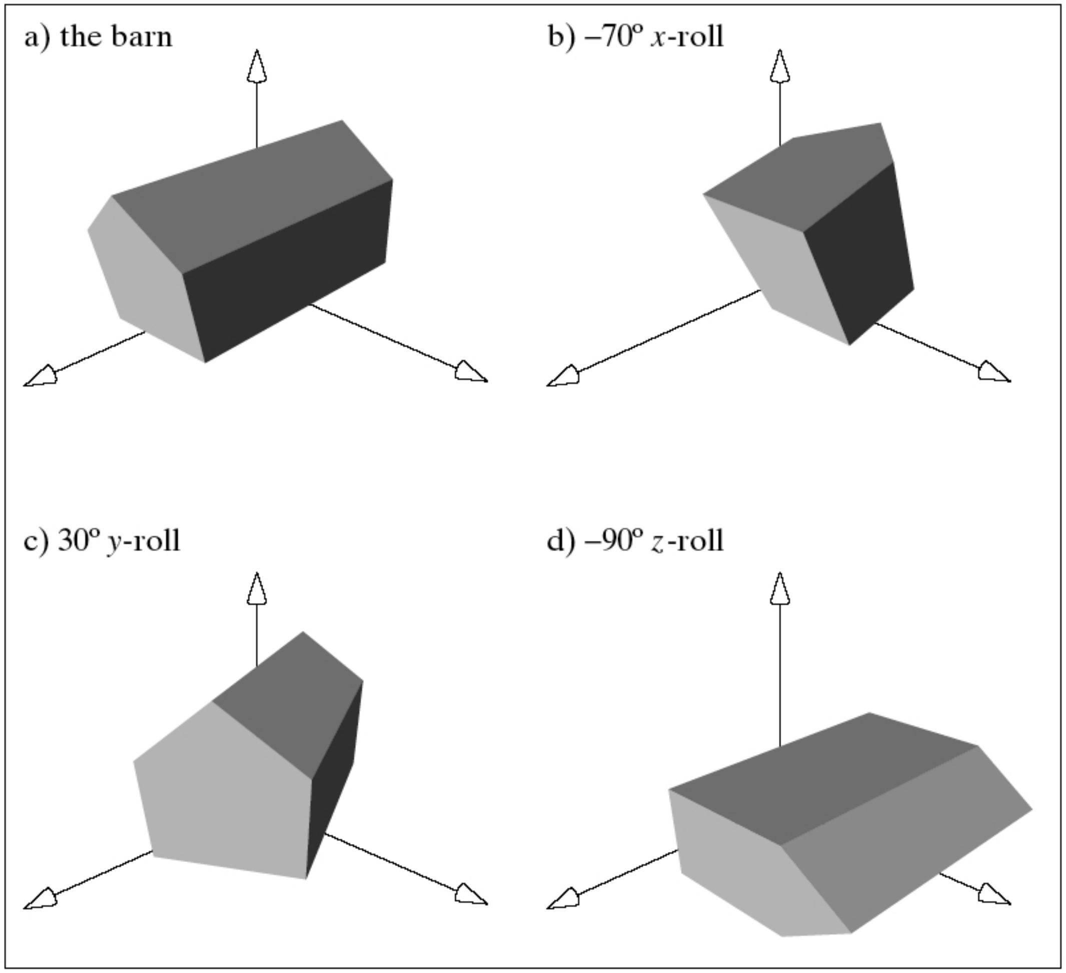
Recall: Right-Handed Coordinates

- To determine positive rotations
  - Make a fist with your right hand, and stick thumb up in the air (CCW)





Rotating in 3D (cont.)



Rotating in 3D (cont.)

- For a rotation angle, β about an axis
- Define

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

- An x-rot:

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

OpenGL:

```
glRotated( ß, 1, 0, 0 );
```

Rotating in 3D (cont.)

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

- A y-rot:

OpenGL:

```
glRotated( ß, 0, 1, 0 );
```

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

- A z-rot:

OpenGL:

```
glRotated( ß, 0, 0, 1 );
```

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

- Rules:
  - Rotation (row, col) is 1
  - c, s in rectangular pattern
  - Rest of rows & cols. are 0

Example: Rotating in 3D

- Q: Using y-rot. equation, rotate P = (3, 1, 4) by 30 degrees
- A: c = cos(30) = 0.866, s = sin(30) = 0.5, and

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

- *e.g.*, first line: 3*c + 1*0 + 4*s + 1*0 = 4.6

Matrix Multiplication Code

- Q: Write C code to Multiply point P = (Px, Py, Pz, 1) by the 4x4 matrix shown below to give new point Q = (Qx,Qy,Qz, 1)

$$\begin{pmatrix} Q_x \\ Q_y \\ Q_z \\ 1 \end{pmatrix} = M \begin{pmatrix} P_x \\ P_y \\ P_z \\ 1 \end{pmatrix} \qquad M = \begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Matrix Multiplication Code (cont.)

- Outline of solution:
  - Declare *P, Q* as arrays:
    - **`double P[4], Q[4];`**
  - Declare transform matrix as two-dimensional array
    - **`double M[4][4];`**
  - Remember: C/C++ indexes from **`0`**, not **`1`**
  - Long way
    - Write out line by line expressions for **`Q[i]`**
    - ```
      Q[0] =    P[0]*M[0][0] + P[1]*M[0][1] +
                P[2]*M[0][2] + P[3]*M[0][3]
      ```
  - Cute way:
    - Use indexing, say **`i`** for outer loop, **`j`** for inner loop
      ```

Matrix Multiplication Code

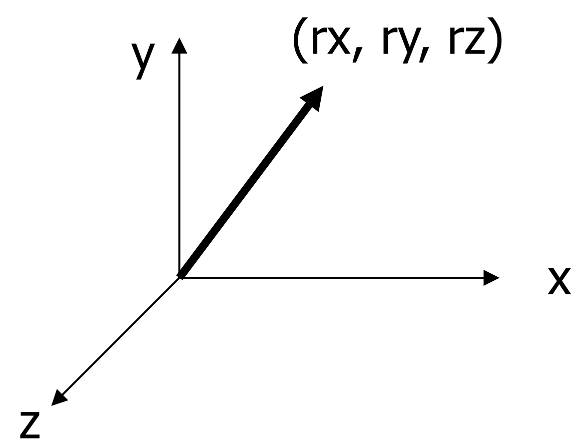
- Using loops looks like:

```
for( i = 0; i < 4; i++ )  {
  temp = 0;
  for( j = 0; j < 4; j++ )  {
    temp += P[j]*M[i][j];
  }
  Q[i] = temp;
}
```

- Test matrix code rigorously
- Use known results (or by hand) and plug into your code

3D Rotation About Arbitrary Axis

- Arbitrary rotation axis (rx, ry, rz)
- OpenGL: rotate(θ, rx, ry, rz)
  - Without OpenGL: a little hairy!!
- Important: read Hill pp. 239-241



3D Rotation About Arbitrary Axis

- Can compose arbitrary rotation as combination of
  - X-rot
  - Y-rot
  - Z-rot

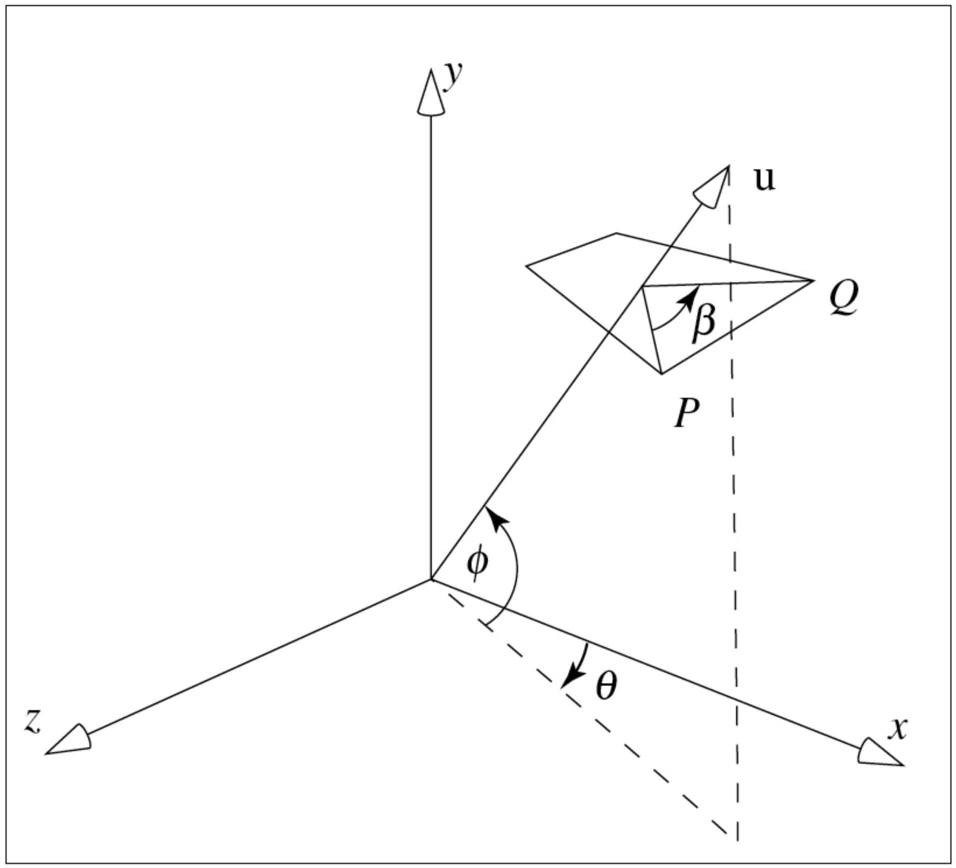
$$M = R_z(\beta_3)R_y(\beta_2)R_x(\beta_1)$$

3D Rotation About Arbitrary Axis

- Want to rotate β degrees about an axis **u** that passes through origin and an arbitrary point
- Classic: Euler's theorem
  - Any sequence of rotations = one rotation about some axis
- Our approach:
  - Use two rotations to align **u** and x-axis
  - Do x-rot through angle β
  - Negate two previous rotations to de-align **u** and x-axis

3D Rotation About Arbitrary Axis

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



Composing Transformations

- Composing transformation
  - Applying several transforms in succession to form one overall transformation
- Example:

M1 X M2 X M3 X P

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

- Be careful with the order
- Matrix multiplication is not commutative