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

a) the barn

b) $-70^\circ$ x-roll

c) $30^\circ$ y-roll

d) $-90^\circ$ z-roll
Rotating in 3D

- For a rotation angle, $\beta$ about an axis
- Define:
  \[ c = \cos(\beta) \quad s = \sin(\beta) \]

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 \textbf{\textit{y-roll}} equation, rotate $P = (3,1,4)$ by 30 degrees:

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

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

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/longitude ($\theta$) 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 using Euler Theorem

- **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 \( u \) specified as azimuth (\( \theta \)) and latitude (\( \phi \))
- First try to align \( u \) with x axis
Approach 1: 3D Rotation using Euler Theorem

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

\[
R_y(\theta)
\]
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 $u$ 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 $i, j, k$

\[ q = q_0 + q_1 i + q_2 j + q_3 k \]

- Quaternions can express rotations on sphere smoothly and efficiently
Approach 2: Rotation using Quaternions

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

\[ R(\beta) = \begin{pmatrix} c + (1-c)u_x^2 & (1-c)u_yu_x + su_z & (1-c)u_zu_x + su_y & 0 \\ (1-c)u_xu_y + su_z & c + (1-c)u_y^2 & (1-c)u_zu_y - su_x & 0 \\ (1-c)u_xu_z - su_y & (1-c)u_yu_z - su_x & c + (1-c)u_z^2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \]

\[ c = \cos(\beta) \quad s = \sin(\beta) \quad \text{Arbitrary axis } u \]
Inverse Matrices

- Can compute inverse matrices by general formulas
- But some easy **inverse transform** observations
  - Translation: $T^{-1}(d_x, d_y, d_z) = T(-d_x, -d_y, -d_z)$
  - Scaling: $S^{-1}(s_x, s_y, s_z) = S\left(\frac{1}{s_x}, \frac{1}{s_y}, \frac{1}{s_z}\right)$
  - Rotation: $R^{-1}(q) = R(-q)$
    - 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
- E.g. declare 1 mesh for soldier, 500 instances to create army
- Then apply *instance transformation* to its vertices to Scale, Orient, Locate
Rotation About Arbitrary Point other than the Origin

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

So, $M = T(p_f) \ R(\theta) \ T(-p_f)$
Scale about Arbitrary Center

- Similarly, 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** object by $(S_x, S_y, S_z)$
  3. **Translate** object back: $T(P_x, P_y, P_z)$

- In matrix form: $T(P_x, P_y, P_z) \cdot (S_x, S_y, S_z) \cdot T(-P_x, -P_y, -P_z) \cdot P$

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

- Rotation about z axis by 30 degrees about a fixed point (1.0, 2.0, 3.0)

```cpp
mat 4 m = Identity();
m = Translate(1.0, 2.0, 3.0) * 
   Rotate(30.0, 0.0, 0.0, 1.0) * 
   Translate(-1.0, -2.0, -3.0);
```

- Remember last matrix specified in program (i.e. translate matrix in example) is first applied
References

- Angel and Shreiner, Chapter 3