Recall: 3D Translation

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

*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)

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

Where: \( x' = x.1 + y.0 + z.0 + tx.1 = x + tx, \) … etc
Recall: Scaling

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

\[
S = S(s_x, s_y, s_z)
\]

\[
x' = s_x x
\]

\[
y' = s_y y
\]

\[
z' = s_z z
\]

\[
p' = S p
\]

\[
\begin{bmatrix}
x' \\
y' \\
z' \\
1
\end{bmatrix} =
\begin{bmatrix}
S_x & 0 & 0 & 0 \\
0 & S_y & 0 & 0 \\
0 & 0 & S_z & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix}
\]

**Example:** \( S_x = S_y = S_z = 0.5 \)

tscales big cube (sides = 1)

to small cube (sides = 0.5)
Nate Robbins Translate, Scale Rotate Demo

glTranslatef(0.00, 0.00, 0.00);
glRotatef(0.0, 0.00, 1.00, 0.00);
glScalef(1.00, 1.00, 1.00);
glBegin(
......

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

gLightfv(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

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 \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 ($\Theta$) 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 \( \mathbf{u} \) 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

\[ R_y(\theta) \]
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 $u$
- But axis $u$ is now lined up with x axis. So,
- **Step 3**: Do x-roll by $\beta$ around axis $u$

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

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

$$q=q_0+q_1i+q_2j+q_3k$$

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

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

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

\[
c = \cos(\beta) \quad s = \sin(\beta)
\]
Inverse Matrices

- Can compute inverse matrices by general formulas
- But easier to use simple geometric observations
  - Translation: $T^{-1}(d_x, d_y, d_z) = T(-d_x, -d_y, -d_z)$
  - Scaling: $S^{-1}(s_x, s_y, s_z) = S(1/s_x, 1/s_y, 1/s_z)$
  - 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
Concatenating Transformations

- Can form arbitrary affine transformation matrices by multiplying rotation, translation, and scaling matrices
- General form:
  \[
  \text{M1} \times \text{M2} \times \text{M3} \times \text{P}
  \]
  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

\[ p' = ABCp = A(B(Cp)) \]

- Efficient!!
  - Matrix \( M=ABC \) is composed, then multiplied by many vertices
  - Cost of forming matrix \( M=ABC \) not significant compared to cost of multiplying \( (ABC)p \) for many vertices \( p \) 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 $T(-p_f)$
  - Rotate $R(\theta)$
  - Move fixed point back $T(p_f)$

So, $M = T(p_f) \cdot R(\theta) \cdot T(-p_f)$
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 the object by $(S_x, S_y, S_z)$
  3. Translate object back: $T(P_x, P_y, P_z)$
- In matrix form: $T(P_x, P_y, P_z) \cdot (S_x, S_y, S_z) \cdot T(-P_x, -P_y, -P_z) \cdot P$

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

- Angel and Shreiner, Chapter 3