3D Geometric Transformations
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

\[
R_z(\theta) = \begin{bmatrix}
\cos \theta & -\sin \theta & 0 & 0 \\
\sin \theta & \cos \theta & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix},
\quad
R_x(\theta) = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & \cos \theta & -\sin \theta & 0 \\
0 & \sin \theta & \cos \theta & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

3D rotations do NOT commute!
Rotation About Arbitrary Axis

- **Initial Position**
- **Step 1** Translate $P_1$ to the Origin
- **Step 2** Rotate $P'_2$ onto the $z$ Axis
- **Step 3** Rotate the Object Around the $z$ Axis
- **Step 4** Rotate the Axis to its Original Orientation
- **Step 5** Translate the Rotation Axis to its Original Position
Rotation about an arbitrary axis

1. Translation: rotation axis passes through the origin
   \[ T = \begin{bmatrix} 1 & 0 & 0 & -x_i \\ 0 & 1 & 0 & -y_i \\ 0 & 0 & 1 & -z_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \]

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 $\mathbf{u}$ onto the $z$-axis

**FIGURE 5-45** Unit vector $\mathbf{u}$ 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 \( \mathbf{u} \) onto the z-axis
  - \( \mathbf{u}' \): Project \( \mathbf{u} \) onto the yz-plane to compute angle \( \alpha \)
  - \( \mathbf{u}'' \): Rotate \( \mathbf{u} \) about the x-axis by angle \( \alpha \)
  - Rotate \( \mathbf{u}'' \) onto the z-axis

\[ \mathbf{u}' = \text{Project } \mathbf{u} \text{ onto the yz-plane} \]
\[ \mathbf{u}'' = \text{Rotate } \mathbf{u} \text{ about the x-axis by angle } \alpha \]

**Figure 5-46** Rotation of \( \mathbf{u} \) around the x axis into the xz plane is accomplished by rotating \( \mathbf{u}' \) (the projection of \( \mathbf{u} \) in the yz plane) through angle \( \alpha \) onto the z axis.

**Figure 5-47** Rotation of unit vector \( \mathbf{u}'' \) (vector \( \mathbf{u} \) after rotation into the xz plane) about the y axis. Positive rotation angle \( \beta \) aligns \( \mathbf{u}'' \) with vector \( \mathbf{u}_z \).
Rotation About Arbitrary Axis

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

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

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

$$= \mathbf{u}_x b$$

$$\rightarrow \quad \sin \alpha = \frac{b}{\|\mathbf{u}'\| \|\mathbf{u}_z\|} = \frac{b}{\sqrt{b^2 + c^2}}$$
Rotation About Arbitrary Axis

- **Rotate** \( \mathbf{u}' \) **about the** \( x \)-**axis onto the** \( z \)-**axis**
  - Since we know both \( \cos \alpha \) and \( \sin \alpha \), the rotation matrix can be obtained

\[
R_x(\alpha) = \begin{pmatrix}
1 & 0 & 0 \\
0 & \frac{c}{\sqrt{b^2 + c^2}} & \frac{-b}{\sqrt{b^2 + c^2}} \\
0 & \frac{b}{\sqrt{b^2 + c^2}} & \frac{c}{\sqrt{b^2 + c^2}} \\
0 & 0 & 1
\end{pmatrix}
\]

- **Rotate** \( \mathbf{u}'' \) **onto the** \( z \)-**axis**
  - With the similar way, we can compute the angle \( \beta \)
Rotation about an arbitrary axis using orthogonal matrix

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

\[
\begin{bmatrix}
P_1 \\
P_2 \\
P_3
\end{bmatrix} \rightarrow
\begin{bmatrix}
P_1' \\
P_2' \\
P_3'
\end{bmatrix}
\]

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,

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

\[ R = R_x(\beta) \cdot R_y(\alpha) = \begin{bmatrix} R_x \\ R_y \\ R_z \end{bmatrix} \]
Reflection

- Reflection about xy plane

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

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

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 $\left(\frac{a}{c}, \frac{b}{c}, 1\right)$

Direction vector $(a, b, c)$
How we represent Rotations?

- Rotation (direction cosine) matrix
- Euler angles
- Angular Displacement
- Unit quaternions
Euler Angles (Euler’s Theorem)

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: \( R_{XYZ}(\gamma, \beta, \alpha) \leftrightarrow \text{rotation matrix} \)

Not easy
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**
  - Coincidence of inner most and outermost gimble’s rotation axes
  - Loss of degree of freedom
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

\[ R(0,0,0) \]
\[ R(50,0,0) \]
\[ R(50,\frac{\pi}{2},0) \]
\[ R(50,\pi,0) \]
\[ R(\pi,\pi,0) \]

\[ R(0,0,0) \]
\[ R(0,\frac{\pi}{2},0) \]
\[ R(0,\frac{\pi}{2},-50) \]
\[ R(0,\frac{\pi}{2},\pi) \]
\[ R(\pi,\pi,0) \]
Set $\theta_y = \frac{\pi}{2}$, and set $\theta_x$ and $\theta_z$ arbitrarily.

$$R = (\theta_x, \theta_y, \theta_z) = \begin{bmatrix}
0 & 0 & -1 & 0 \\
-s_xc_z + c_zs_x & ss_z + c_zc_x & 0 & 0 \\
c_xc_z + s_zs_x & c_zs_x - s_zc_x & 0 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}$$

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

Transformation only depends on the difference

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 $\theta$

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

$$q = (s, \vec{v}) = s + v_1i + v_2j + v_3k$$

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

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

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

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

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

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

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

$$q_1 \cdot q_2 = (s_1s_2 - \mathbf{v}_1 \cdot \mathbf{v}_2, s_1\mathbf{v}_2 + s_2\mathbf{v}_1 + \mathbf{v}_1 \times \mathbf{v}_2)$$

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

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

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

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

[Example] Rotation about z-axis

\[ s = \cos \frac{\theta}{2}, \quad \mathbf{v} = (0, 0, 1) \sin \frac{\theta}{2} \]
Quatertion 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 Euler angle
- Interpolation can be expensive in practice
Modeling Transformation
Modeling Transformation
Modeling Transformation

\[ M_{x' y' \rightarrow x y} = Scale(x, y) \cdot T(x, y) \]
Modeling Transformation

Translate by \(-r\), bringing \(r\) to the origin.

\[ \text{Rotate}(A) \]

\[ \text{translate}(A) \]
translate both A and B by \(-p\), bringing \(p\) to the origin.
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!