Rotation representation and Interpolation
### Rotation Matrices

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & \cos(\alpha) & -\sin(\alpha) & 0 \\
0 & \sin(\alpha) & \cos(\alpha) & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

\[
\begin{bmatrix}
\cos(\beta) & 0 & \sin(\beta) & 0 \\
0 & 1 & 0 & 0 \\
-\sin(\beta) & 0 & \cos(\beta) & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

\[
\begin{bmatrix}
\cos(\gamma) & -\sin(\gamma) & 0 & 0 \\
\sin(\gamma) & \cos(\gamma) & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]
Rotation Matrix must be orthonormal

- Rotation matrix is also the local coordinate system!

\[
M = \begin{bmatrix}
 u_x & v_x & w_x \\
 u_y & v_y & w_y \\
 u_z & v_z & w_z
\end{bmatrix}
\]
Rotation Matrix must be orthonormal

• Since $u$, $v$, and $w$ are orthonormal vectors, we must have:

$$M^T M = \begin{bmatrix} u_x & u_y & u_z \\ v_x & v_y & v_z \\ w_x & w_y & w_z \end{bmatrix} \begin{bmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

• In other words, $M^T = M^{-1}$.

• A rotation matrix must be orthonormal.
Flawed Solution

• Interpolate each entry independently

• Example: $M_0$ is identity and $M_1$ is $90^\circ$ around x-axis:

\[
\begin{bmatrix}
0.5 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1 \\
\end{bmatrix}
+ 0.5
\begin{bmatrix}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1 \\
\end{bmatrix}
= \begin{bmatrix}
1 & 0.5 & -0.5 \\
0.5 & 0.5 & 0.5 \\
\end{bmatrix}
\]

• Is the result a rotation matrix?
Orientation Representations

Rotation matrix

Euler angles: rotate about the axes of a global coordinate system
Euler angles: rotate about the axes of a local coordinate system

Quatertion: mathematically handy axis-angle
Euler Angles: Local Coordinate System

• Any orientation can be described by composing three rotations, one around each global coordinate axis

• Roll, pitch and yaw (perfect for flight simulation)
Gimbal Lock

- Two or more axes align resulting in a loss of rotation degrees of freedom.
- Also occurs in Unity! Can you find it out?
The axis-angle idea

\[
\begin{bmatrix}
\text{Axis} & \theta
\end{bmatrix}
\]

Rotate about any given axis

Fairly easy to interpolate between orientations
Quaternions (intuitively)

\[
\text{Quaternion}_{\theta} = \begin{bmatrix}
\tilde{A}\sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
\vec{v} & S
\end{bmatrix}
\]
## Quaternion Arithmetic

### Addition

\[
\begin{bmatrix}
\vec{v}_1 + \vec{v}_2 & s_1 + s_2
\end{bmatrix}
= \begin{bmatrix}
\vec{v}_1 & s_1
\end{bmatrix}
+ \begin{bmatrix}
\vec{v}_2 & s_2
\end{bmatrix}
\]

### Multiplication

\[
q_1 q_2 = \begin{bmatrix}
   s_2 \vec{v}_1 + s_1 \vec{v}_2 + \vec{v}_1 \times \vec{v}_2 & s_1 s_2 - \vec{v}_1 \cdot \vec{v}_2
\end{bmatrix}
\]

### Inner Product

\[
q_1 \cdot q_2 = \vec{v}_1 \cdot \vec{v}_2 + s_1 s_2
\]

### Length

\[
\|q\| = \sqrt{q \cdot q} = 1
\]
Quatertion-Rotation Matrix Conversions

• $q = [x, y, z, w]$

$$R = \begin{bmatrix}
w^2 + x^2 - y^2 - z^2 & 2(xy - wz) & 2(xz + wy) \\
2(xy + wz) & w^2 - x^2 + y^2 - z^2 & 2(yz - xw) \\
2(xz - wy) & 2(yz + xw) & w^2 - x^2 - y^2 + z^2
\end{bmatrix}$$

• What if $q$ is scaled? $[sx, sy, sz, sw]$?

• So $q$ has to be normalized!
Quaternions
What an animator needs to know

Avoid gimbal lock

Easy to rotate a point

Easy to convert to a rotation matrix

Easy to concatenate – quaternion multiply
   First rotate $q_1$, then $q_2$: $q_2 q_1$

Easy to interpolate – interpolate 4-tuples and then normalize
Quaterrion interpolation: A simple way

Quaternions can be interpolated to produce in-between orientations:

$$q = (1 - k)q_1 + kq_2$$

Linearly interpolating unit quaternions: not equally spaced
Interpolating quaternions in great arc
=> equal spacing
Interpolating quaternions with equal spacing

\[
\text{slerp}(q_1, q_2, t) = \frac{\sin(1 - t)\alpha}{\sin\alpha} q_1 + \frac{\sin t\alpha}{\sin\alpha} q_2
\]

where \( q_1 \cdot q_2 = \cos\alpha \)

**Quaternion.Slerp** in Unity

- spherical linear interpolation is a function of
  - the beginning quaternion orientation, \( q_1 \)
  - the ending quaternion orientation, \( q_2 \)
  - the interpolant, \( u \)

Still a linear order interpolation
Rotations within Unity

```csharp
transform.Rotate(eulerAngles : Vector3,
                    relativeTo : Space = Space.Self)
```

Description

Applies a rotation of eulerAngles.z degrees around the z axis, eulerAngles.x degrees around the x axis, and eulerAngles.y degrees around the y axis (in that order).
Rotations within Unity

`transform.Rotate(axis : Vector3, angle : float, relativeTo : Space = Space.Self)`

**Description**
Rotates the transform around axis by angle degrees.
Rotations within Unity


Description
Rotates the transform about axis passing through point in world coordinates by angle degrees. This modifies both the position and the rotation of the transform.
Rotations within Unity

Quaternion q;
transform.rotation=q;

Description
Simply set the current origination to q. (In other words, rotating from the original orientation by q.)