Keyframe animation

- Keyframing is the technique used for pose-to-pose animation
  - User creates key poses—just enough to indicate what the motion is supposed to be
  - Interpolate between the poses

Controlling shape for animation

- Start with modeling DOFs (control points)
- Deformations control those DOFs at a higher level
  - Example: move first joint of second finger on left hand
- Animation controls control those DOFs at a higher level
  - Example: open/close left hand
- Both cases can be handled by the same kinds of deformers

Rigid motion: the simplest deformation

- Move a set of points by applying an affine transformation
- How to animate the transformation over time?
  - Interpolate the matrix entries from keyframe to keyframe?
    - Translation: ok
      - start location, end location, interpolate
    - Rotation: not so good

Parameterizing rotations

- Euler angles
  - Rotate around x, then y, then z
  - Problem: gimbal lock
    - If two axes coincide, you lose one DOF
- Unit quaternions
  - A 4D representation (like 3D unit vectors for 2D sphere)
  - Good choice for interpolating rotations
- These are first examples of motion control
  - Matrix = deformation
  - Angles/quaternion = animation controls
Quaternions

- Remember that
  - Orientations can be expressed as rotation
- Why?
  - Start in a default position (say aligned with z axis)
  - New orientation is rotation from default position
- Rotations can be expressed as (axis, angle)

- Quaternions let you express (axis, angle)

Quaternions for Rotation

- A quaternion is an extension of complex numbers

Review complex numbers

- Each of i, j and k are three square roots of -1
- \[ i^2 = j^2 = k^2 = ijk = -1 \]
- Cross-multiplication is like cross product
  \[ ij = -ji = k \]
  \[ jk = -kj = i \]
  \[ ki = -ik = -j \]

ONB in quaternions

- Quaternion is extension of complex number in 4D space

\[
q = w + xi + yj + zk
\]
\[
q' = w - xi - yj - zk
\]
\[
||q|| = \sqrt{w^2 + x^2 + y^2 + z^2}
\]
Quaternion Properties

• Linear combination of 1, i, j, k

\[ q = w + xi + yj + zk = (s, v) \]
\[ s = w, v = [x, y, z] \]

• Multiplication

\[ q_1 = (s_1, v_1), q_2 = (s_2, v_2) \]
\[ q_1 \cdot q_2 = (s_1 s_2 - v_1 \cdot v_2, s_1 v_2 + s_2 v_1 + v_1 \times v_2) \]

Quaternion Properties

• Associative

\[ q_1 \cdot (q_2 \cdot q_3) = (q_1 \cdot q_2) \cdot q_3 \]

• Not commutative

\[ q_1 \cdot q_2 \neq q_2 \cdot q_1 \]

• Unit quaternion

\[ \|q\| = 1 \]
\[ q^{-1} = q^t \]

Quaternion for Rotation

• Rotate about axis a by angle \( \theta \)

\[ q = (s, v) = (s, v_1, v_2, v_3) \]
\[ s = \cos \left( \frac{\theta}{2} \right) \]
\[ v = \sin \left( \frac{\theta}{2} \right) \hat{a} \]

• Note: unit quaternion

Rotation Using Quaternion

• A point in space is a quaternion with 0 scalar

\[ X = (0, \vec{x}) \]

Rotation Using Quaternion

• A point in space is a quaternion with 0 scalar

\[ X = (0, \vec{x}) \]

• Rotation is computed as follows

\[ x_{\text{rotated}} = q X q^{-1} = q X q^t \]

• See Buss 3D CG: A mathematical introduction with OpenGL, Chapter 7

Matrix for quaternion

\[
\begin{bmatrix}
w^2 + x^2 - y^2 - z^2 & 2xy - 2wz & 2x + 2wy & 0 \\
2xy + 2wz & w^2 - x^2 + y^2 - z^2 & 2yz - 2wx & 0 \\
2xz - 2wy & 2yz + 2wx & w^2 - x^2 - y^2 + z^2 & 0 \\
0 & 0 & 0 & w^2 + x^2 + y^2 + z^2
\end{bmatrix}
\]
Why Quaternions?
- Fast, few operations, not redundant
- Numerically stable for incremental changes
- Composes rotations nicely
- Convert to matrices at the end
- Biggest reason: spherical interpolation

Interpolating between quaternions
- Why not linear interpolation?
  - Need to be normalized
  - Does not have constant rate of rotation

Spherical Linear Interpolation
- Intuitive interpolation between different orientations
  - Nicely represented through quaternions
  - Useful for animation
  - Given two quaternions, interpolate between them
- Shortest path between two points on sphere
  - Geodesic, on Great Circle

Quaternion Interpolation
- Shortest arc on the 4D unit sphere between q1 and q2
  - Path is spherical geodesic
  - Uniform angular rotation velocity about a fixed axis
  \[
  \text{lerp}(q_1, q_2, t) = \frac{\sin((1 - t)\psi)}{\sin\psi} q_1 + \frac{\sin(t\psi)}{\sin\psi} q_2
  \]
  \[
  \cos(\psi) = q_1 q_2 = s_1 v_2 + v_1 s_2
  \]
Practical issues

- When angle gets close to zero, use small angle approximation
  - degenerate to linear interpolation between q1 and q2
- When angle close to 180, there is no shortest geodesic. Be careful
- q is same rotation as -q
  - if q1 and q2 angle < 90, slerp between them
  - else, slerp between q1 and -q2

Rotation Using Quaternion

- Composing rotations
  - q1 and q2 are two rotations
  - First, q1 then q2

\[ \text{rotated} = q_2(q_1 X q_1^{-1} q_2^{-1}) \]
\[ \text{rotated} = (q_2 q_1 X (q_1^{-1} q_2^{-1}) \]
\[ \text{rotated} = (q_2 q_1 X (q_2 q_1)^{-1} \]