CS475/CS675 - Computer Graphics

Lecture 17: Interpolation for Animation
Animation

- Keyframing
  - Selected (key) frames are specified.
  - Interpolation of intermediate frames.
  - Simple and popular approach.
  - May give incorrect (inconsistent) results.

In-between Frames
Animation

- Keyframing
Animation

- Keyframing
  - Interpolate Position
Animation

• Keyframing
  - Interpolate Orientation
Animation

- Keyframing
  - Interpolate Orientation
  - Interpolate Position
  - Interpolate Shape
  - Interpolate Colour
  - Light Intensity
  - Camera Zoom
  - Any other parameter

Animation

• Keyframing Position
  - Moving on curves.
  - Specify spatial position to fix the curve
  - In addition, we specify the speed at which we travel along the curve

\[ A, t = 0 \]
\[ B, t = 10 \]
\[ C, t = 35 \]
\[ D, t = 60 \]
Animation

- Controlling speed on curves
  - Typically parametrization is not arc length.
  - Arc length is the distance along the curve.
  - Arc length parametrization can be computed using
    - Analytical Computation
    - Table-based
      - Summed linear distances (forward differencing)
      - Gaussian quadrature (numerical integration)
Animation

- Controlling speed on curves
  - Given a parametric curve, \( P(u) = (x(u), y(u), z(u)) \)
  - We may have to solve two versions of the problem:
    - Given parameters \( u_1 \) and \( u_2 \), find arc length, \( \text{LENGTH}(u_1, u_2) \)
    - Given an arc length \( s \) and parameter \( u_1 \), find \( u_2 \) so that \( \text{LENGTH}(u_1, u_2) = s \)
Animation

- Controlling speed on curves
  - Generally, neither of the two forms of the problem admit analytical solutions.

- Arc Length

\[
\text{LENGTH}(u_1, u_2) = s = \int_{u_1}^{u_2} \left\| \frac{dP(u)}{du} \right\| \, du = \int_{u_1}^{u_2} \sqrt{\left( \frac{dP(u)}{du} \right)^2} \, du
\]

\[
\sqrt{\left( \frac{dP(u)}{du} \right)^2} = \sqrt{\left( \frac{dx(u)}{du} \right)^2 + \left( \frac{dy(u)}{du} \right)^2 + \left( \frac{dz(u)}{du} \right)^2}
\]
Animation

- Controlling speed on curves
  - The arc length integral can be approximated using a forward differencing method.
  - Create a piece wise linear approximation of the curve from many parameter evaluations and sum these to form the arc length.
  - Store these values into a table.
Animation

- Controlling speed on curves
  - The inversion can then be calculated using bisection

\[ s = G(u) \]

is a monotonically increasing function. i.e., if \( u_1 < u_2 \) then \( s_1 < s_2 \)

So we can do a bisection or a binary search for \( u \), given a value of \( s \).
Animation

- Controlling speed on curves
  - Given parameters $u_1$ and $u_2$, find arc length, $\text{LENGTH}(u_1, u_2)$
    - Can we compute $s = G(u) = \text{distance from start of curve to point at } u$?
    - With $G$, arc length parametrization can be obtained by inversion as $P(G^{-1}(s))$, where $G^{-1}(s)$ gives the parameter $u$ up to which distance travelled on the curve is $s$. 

Re-parametrize to have equal spacing in the parametric interval.

Equal arc lengths $s$ over the curve.
Animation

- Controlling speed on curves
  - Space curves we have seen till now give the path.
  - What if we want to control the speeds
    - Accelerates from stop position
    - Reaches maximum speed
    - Decelerates to a stop.
  - Given the speed as a plot of s vs. t.
    - $v = ds/dt$
Animation

- Controlling speed on curves
  - A slow in – slow out curve may look like:
Animation

• Controlling speed on curves
  – Given next time instant t
  – Distance-time curve gives total distance $s$ travelled up to time $t$.
  – $P(G^{-1}(s))$ gives the position on the path space curve.

• Or solve a space-time optimization for the whole path.
Animation

- Interpolating orientation
  - **Fixed Angle Representation** - Ordered triple of rotations about *global axes.*
  - Any triple is valid that doesn't immediately repeat an axis, e.g., x-y-z or z-x-y. But not x-x-y.
  - Let us assume a z-y-x order for now.
    \[ P' = R_z(\gamma) \cdot R_y(\beta) \cdot R_x(\alpha) \cdot P \quad (\alpha, \beta, \gamma) \]
Animation

- Interpolating orientation – Fixed Angle Representation
  - Make a rotation matrix from the angles and interpolate

\[ R_z(90) = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \]

\[ R_z(-90) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \]

\[ R_z(?) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \]
Animation

- Interpolating orientation – Fixed Angle Representation
  - Interpolate the angles and then form the matrix.
  - Suffers from the Gimbal lock!

\[(0, 0, 0), (0, 90, 0), (0, 90, \pm \epsilon)\]

\[(\pm \epsilon, 90, 0)\]
Animation

- Interpolating orientation
  - **Euler Angle Representation** - Ordered triple of rotations about *local axes*.

- Any triple is valid that doesn't immediately repeat an axis, e.g., x-y-z or z-x-y. But not x-x-y.

- Let us assume a x-y-z order for now. 
  \[
  P' = R_x(\alpha) \cdot R_y(\beta) \cdot R_z(\gamma) \cdot P
  \]

Rotation given by a triad of Euler angles is the same as given by a triad of Fixed angles considered in **opposite** order. So it has the same Gimbal Lock problem.
Animation

- Interpolating orientation
  - When to form and apply the matrix if rotation $\Theta$ has to be incremented by $\delta \Theta$ in each frame?
    - Form a rotation matrix for $\delta \Theta$ and apply repeatedly to rotated object in each frame.
    - Update the rotation matrix $R_{axis}(\Theta)$ by multiplying with $R_{axis}(\delta \Theta)$ in each frame. Apply updated matrix to the object.
    - Update the rotation angle, $\Theta$ by the increment $\delta \Theta$ and form the new matrix $R_{axis}(\Theta + \delta \Theta)$ in each frame. Apply this matrix to the object.
Animation

- Interpolating orientation
  - **Axis Angle Representation** - Specified as an axis of rotation $A(x, y, z)$ and an angle of rotation, $\theta$ around it.
  - Euler’s Theorem – Any orientation can be derived from another by a single rotation about and axis.
Animation

- Interpolating orientation - Axis Angle Representation
  
  To interpolate between two orientations \((A_1, \theta_1)\) and \((A_2, \theta_2)\)

\[
B = A_1 \times A_2 \quad \phi = \cos^{-1}\left( \frac{A_1 \cdot A_2}{||A_1|| ||A_2||} \right)
\]

\[
A_k = R_B(k \cdot \phi) A_1
\]

\[
\theta_k = (1 - k) \theta_1 + k \theta_2
\]

with \(0 \leq k \leq 1\)
Animation

- Interpolating orientation
- **Unit Quaternions**
  - Have the same information as the axis-angle representation but in a more convenient form.

\[ q = [s, x, y, z] = [s, v] = [\cos \theta / 2, \sin \theta / 2 \cdot \hat{a}], \text{ where } \hat{a} = \frac{A}{\|A\|} \]
Interpolating orientation – Quaternions

- A non commutative number system that extends complex numbers
- Defined as: $q = s + x \hat{i} + y \hat{j} + z \hat{k} = [s, v]
- where $1, \hat{i}, \hat{j}, \hat{k}$ are called the Hamilton basis
- The product of the basis elements is defined as:
  
  $\hat{i}^2 = \hat{j}^2 = \hat{k}^2 = \hat{i}\hat{j}\hat{k} = -1$  
  
  $\Rightarrow \hat{i}\hat{j} = \hat{k}, \hat{j}\hat{k} = \hat{i}, \hat{k}\hat{i} = \hat{j}$  
  
  $\Rightarrow \hat{j}\hat{i} = -\hat{k}, \hat{k}\hat{j} = -\hat{i}, \hat{i}\hat{k} = -\hat{j}$

Quaternions form a four dimensional normed division algebra, $\mathbb{H}$ over the real numbers.
• Interpolating orientation – Quaternions
  - A non commutative number system that extends complex numbers
  - Defined as: \( q = s + x \hat{i} + y \hat{j} + z \hat{k} = [s, v] \)
  - where \( \hat{1}, \hat{i}, \hat{j}, \hat{k} \) are called the Hamilton basis
  - The product of the basis elements is defined as:
    \[
    \hat{i}^2 = \hat{j}^2 = \hat{k}^2 = \hat{i} \hat{j} \hat{k} = -1
    \]
    \[
    \Rightarrow \hat{i} \hat{j} = \hat{k}, \hat{j} \hat{k} = \hat{i}, \hat{k} \hat{i} = \hat{j}
    \]
    \[
    \Rightarrow \hat{j} \hat{i} = -\hat{k}, \hat{k} \hat{j} = -\hat{i}, \hat{i} \hat{k} = -\hat{j}
    \]

Note that the multiplication being defined here is a quaternion multiplication and not the inner or out product of vectors. It is **not commutative.**
Animation

- Interpolating orientation – Quaternions
  - Quaternion Arithmetic
  - Addition: \( q_1 + q_2 = [s_1 + s_2, v_1 + v_2] \)
  - Scalar Multiplication: \( k q = [k s, k v] = ks + kx \hat{i} + ky \hat{j} + kz \hat{k} \)
  - Quaternion Multiplication:
    \[
    q_1 q_2 = (a_1 + b_1 \hat{i} + c_1 \hat{j} + d_1 \hat{k})(a_2 + b_2 \hat{i} + c_2 \hat{j} + d_2 \hat{k})
    = (a_1 a_2) - (b_1 b_2 + c_1 c_2 + d_1 d_2) + (a_1 b_2 + b_1 a_2 + c_1 d_2 - d_1 c_2) \hat{i} + \\
    (a_1 c_2 - b_1 d_2 + c_1 a_2 + d_1 b_2) \hat{j} + (a_1 d_2 + b_1 c_2 - c_1 b_2 + d_1 a_2) \hat{k}
    \]
Animation

- Interpolating orientation – Quaternions
  - Conjugate Quaternion: \( q^* = s - x \hat{i} - y \hat{j} - z \hat{k} \)
  - Quaternion Norm: \( ||q|| = \sqrt{q q^*} = \sqrt{q^* q} = \sqrt{s^2 + x^2 + y^2 + z^2} \)
  - If \( \alpha \) is real then, \( ||\alpha q|| = |\alpha||q|| \)
  - The norm is multiplicative: \( ||pq|| = ||p||||q|| \)
  - Unit Quaternion: \( \hat{q} = \frac{q}{||q||} \)
  - Quaternion Inverse: \( q^{-1} = \frac{q^*}{||q||^2} \)
Animation

- Interpolating orientation – Quaternions
  - Quaternion and the Geometry of $\mathbb{R}^3$
    $\hat{i}, \hat{j}, \hat{k}$ denote both the basis vectors of $\mathbb{H}$ and a basis for $\mathbb{R}^3$
  - Vectors in $\mathbb{R}^3$ can be written as pure imaginary quaternions
    \[ v = 0 + x \hat{i} + y \hat{j} + z \hat{k} = [0, u] \]
  - Inner product of vectors in $\mathbb{R}^3$
    \[ v_1 \cdot v_2 = x_1 x_2 + y_1 y_2 + z_1 z_2 = \frac{1}{2} (v_1^* v_2 + v_2^* v_1) = \frac{1}{2} (v_1 v_2^* + v_2 v_1^*) \]
  - Cross product of vectors in
    \[ v_1 \times v_2 = \frac{1}{2} (v_1 v_2^* - v_2 v_1^*) \]
  - Quaternion multiplication can be written as:
    \[ q_1 q_2 = [s_1 s_2 - v_1 \cdot v_2, s_2 v_1 + s_1 v_2 + v_1 \times v_2] \]
Animation

- Interpolating orientation – **Unit** Quaternions
  - A unit quaternion denotes a rotation by an angle $\theta$ about an axis $A$
  - $q = [s, x, y, z] = [s, v] = [\cos \theta/2, \sin \theta/2 \ast \hat{a}]$, where $\hat{a} = \frac{A}{\|A\|}$
  - Multiplication with a unit quaternion $q$ can be used to rotate a vector $v$
    \[
    R_q(v) = qvq^{-1} = q[0, v]q^{-1}
    \]
  - Composition of rotations is equivalent to quaternion multiplication
    \[
    R_{q_1}(R_{q_2}(\vec{v})) = R_{q_1}(q_2 \vec{v} q_2^{-1}) = q_1 q_2 [0, v] q_2^{-1} q_1^{-1}
    \]
    \[
    = (q_1 q_2)[0, v] (q_1 q_2)^{-1} = R_{q_1 q_2}(\vec{v})
    \]
  - Rotating by a scalar multiple of a unit quaternion is the same as rotating by the unit quaternion $R_q(\vec{v}) = R_{kq}(\vec{v})$

Only Unit Quaternions represent rotations!
Animation

- Interpolating orientation – **Unit** Quaternions
  - Antipodal Unit Quaternions

  - \( q = [\cos(\theta/2), \hat{a}\sin(\theta/2)] \)
  - If we rotate by \( \theta-2\pi \) instead of \( \theta \)
    \[
    \begin{bmatrix}
    \cos((\theta-2\pi)/2), \hat{a}\sin((\theta-2\pi)/2)
    \end{bmatrix}
    \]
    \[
    = \begin{bmatrix}
    \cos(\theta/2-\pi), \hat{a}\sin(\theta/2-\pi)
    \end{bmatrix}
    = [\cos(\theta/2), -\hat{a}\sin(\theta/2)] = -q
    \]
  
  - So both \( q \) and \(-q\) represent the same rotation and are called antipodal points.

  - If \( 0 < \theta < \pi \) then the positive rotation is the shorter one else the negative rotation is the shorter one, i.e., the quaternion with the positive value of the s coordinate will give the shorter path

\[
R_q(v) = R_{-q}(v)
\]

I am abusing notation here for convenience.

Please remember we are talking about unit quaternions.
Animation

- Interpolating orientation – Unit Quaternions
  - Linear Interpolation \( q = (1 - k)q_1 + kq_2 \)
  - How to take equi-distant steps along orientation path?
  - How to pass through orientations smoothly?
  - With dual unit quaternion representation

Dual representation: For Interpolation between \( q_1 \) and \( q_2 \), compute cosine between \( q_1 \) and \( q_2 \) and between \( q_1 \) and \(-q_2\); choose smallest angle.
Animation

- Interpolating orientation – Unit Quaternions
  - Linear Interpolation \( q = (1 - k)q_1 + kq_2 \)
  - This is not equally spaced.
Animation

• Interpolating orientation – Unit Quaternions
  - Spherical Linear Interpolation or SLERP
  - We write, \( q^\alpha = [\cos (\alpha \theta / 2), \hat{a} \sin (\alpha \theta / 2)] \)
  - We want to interpolate between two rotations \( q_1 \) and \( q_2 \)
  - Rotation that takes us from 1 to 2 is given by \( q_2 \hat{q}_1^{-1} \)
  - Now we start at 1, and go to 2 in \( \alpha \) steps as \( (q_2 \hat{q}_1^{-1})^\alpha q_1 \)

\[
Slerp(q_1, q_2, \alpha) = (q_2 \hat{q}_1^{-1})^\alpha q_1
\]

\[
Slerp(q_1, q_2, \alpha) = \frac{\sin (1 - \alpha) \theta}{\sin \theta} q_1 + \frac{\sin (\alpha) \theta}{\sin \theta} q_2
\]

\[
\cos \theta = q_1 \cdot q_2 = s_1 s_2 + x_1 x_2 + y_1 y_2 + z_1 z_2
\]