Animation and Modeling

Adam Mally
CIS 277 Spring 2015
University of Pennsylvania
Keyframes

- Fundamental aspect of many forms of computer animation
- Set specific transform values for scene elements at specific times (frames)
- When a transform is not specified at a given frame, interpolate between the value at the previous keyframe and the next keyframe
Interpolation

- Review: Linear interpolation and spherical interpolation
  - LERP: \((1-t)\mathbf{a} + t\mathbf{b}\)
  - SLERP:

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

\(t = [0,1]\) and \(\theta = \cos^{-1}(\dot{\mathbf{q}}(q_1, q_2))\)
Interpolation

- LERP and SLERP work wonderfully when we are dealing with only two points of animation.
- What if we want our interpolation’s slope to be continuous across all keyframes?
- We can compute Bézier curves between each keyframe pair.
Review: Bézier curves

- Bezier curves can be thought of as a higher order extension of linear interpolation
- We use de Casteljau’s algorithm to compute them
de Casteljau’s Algorithm

- Find point $q$ on each edge
- $q = (p1 - p0)t + p0$, or $q = \text{LERP}(t, p0, p1)$
- $t = [0,1]$
de Casteljau’s Algorithm

- Find point $r$ on the edge between two $q$s
- $r = (q_1 - q_0)*t + q_0$, or $r = \text{LERP}(t, q_0, q_1)$
- $t = [0,1]$
de Casteljau’s Algorithm

- Find point $x$ on the edge between two $r$s
- $x = (r1 - r0) \times t + r0$, or $r = \text{LERP}(t, r0, r1)$
- $t = [0,1]$
de Casteljau’s Algorithm

Repeat for all $t$s between 0 and 1 to get all points on the curve bounded by $p0$ through $p3$
Continuous Cubic Interpolation

- For N keyframes where N > 2, for all keyframe pairs not at the beginning or end of the animation, we must compute a pair of control points
  - For any keyframe, the control point to its left and to its right must be colinear along with the keyframe in order for the interpolation to appear continuous
- Simple solution: for keyframes A and D,
  \[
  \text{control point B} = \langle A.\text{value}, A.\text{time} + (D.\text{time} - A.\text{time}) / 2 \rangle
  \]
  \[
  \text{control point C} = \langle D.\text{value}, A.\text{time} + (D.\text{time} - A.\text{time}) / 2 \rangle
  \]
- This gives us a “perfect” cubic function curve
- However, the animation “slows down” around each keyframe because the slope of the animation is roughly 0.
Smooth Cubic Interpolation

- May want to get interpolation with a continuous curve that does not “slow down” around keyframes
- Solution: for each keyframe not at the start/end of the animation, compute a Bézier curve handle such that the “slope” of the handle is the slope of the line from the previous keyframe to the next keyframe
Smooth Cubic Interpolation

- How far should each handle anchor be from the keyframe?
  - Sort of up to artistic license
  - $\frac{1}{3}$ to $\frac{1}{2}$ of the distance from the current keyframe to the adjacent keyframe works pretty well
Smooth quaternion interpolation

- Interpolating keyframes that represent “linear” values such as translation, transparency, or color is fairly straightforward when using Bézier curves.
- What about quaternions, which are interpolated in and of themselves rather than by component (e.g. the x, y, and z of a translation)?
- We use a technique called SQUAD, a term composed of “spherical and quadrangle”
  ○ Proposed by Ken Shoemake in 1987
Quaternion interpolation: SQUAD

- Given a pair of quaternions $q_i$ and $q_{i+1}$ as well as two intermediate “auxiliary” quaternions $s_i$ and $s_{i+1}$, we can cubically interpolate as follows:

$$
\text{Squad}(q_i, q_{i+1}, s_i, s_{i+1}, h) = \text{Slerp}(\text{Slerp}(q_i, q_{i+1}, h), \text{Slerp}(s_i, s_{i+1}, h), 2h(1 - h))
$$

- This gives us smooth interpolation across the interval $[q_i, q_{i+1}]$, but what about continuous interpolation across a set of $N$ quaternions?
- We need to compute useful values for $s_i$ and $s_{i+1}$
Quaternion interpolation: SQUAD

- We need to find values for $s_i$ and $s_{i+1}$ such that we have continuous, smooth interpolation across our set of quaternion keyframes

$$s_i = q_i \exp \left( -\frac{\log(q_i^{-1} q_{i+1}) + \log(q_i^{-1} q_{i-1})}{4} \right)$$

- Reminder: $\exp$ is short for “$e$ to the power of”
- The inverse of a quaternion $q = a + bi + cj + dk$ is equivalent to its conjugate, $q^{-1} = a - bi - cj - dk$
  - This can also be written as $q^{-1} = [a, -b, -c, -d]$
- What about the handles for the start and end quaternions of our animation?
  - $s_0 = q_0$ and $s_N = q_N$ work well enough
Implementing keyframe interpolation

- Given a set of keyframes $k_i$ each described as $[\text{frame}_i, \text{anim}_\text{value}_i]$ and our current frame $f$, compute the interpolated $\text{anim}_\text{value}$ at $f$.
- If $f$ is in the set of frames that are keyframes, then our animation value is read from the keyframe.
- Otherwise, compute a value $t = (f - k_0)/(k_1 - k_0)$ where $t = [0,1]$.
  - Given $f$, $k_0$ is the keyframe with the highest frame value that is less than $f$.
  - Likewise, $k_1$ is the keyframe with the lowest frame value that is higher than $f$.
  - Compute an interpolated animation value using one of the functions previously described, given at least $t$, $k_0$, and $k_1$. 
NURBS curves

- Stands for Non-Uniform Rational B-Spline
- Generalization of a Bézier curve
- Parameters: degree, control points, weights, knot vector
  - **degree**: the power of the polynomial basis function used to compute the curve. Higher degree = smoother curve
  - **control points**: coordinates that determine shape of curve
  - **weight**: positive number assigned to each control point given a $t$ parameter along our control curve
  - **knot vector**: sequence of parameter values that determines how the control points affect the curve. Mathematically necessary to compute the curve, but unintuitive for the purposes of interactively modeling a curve
NURBS curves - basis function

Basis function is determined by degree. For instance, use linear or cubic interpolation as seen before.

knots: {... 0, 1, 2, 3, 4, 4.1, 5.2, 6.1, 7.1 }

note that 4 to 4.1 is a smaller knot span, meaning that the curve will more closely approach that control point.

linear basis

quadratic basis
NURBS curves

\[ C(u) = \frac{\sum_{i=0}^{n-1} w_i N_{i,p}(u) P_i}{\sum_{i=0}^{n-1} w_i N_{i,p}(u)} \]

- \( n \): degree
- \( w_i \): weight for control point \( i \)
- \( N_{i,p} \): b-spline basis function of degree \( p \) for \( i^{th} \) control point
- \( P_i \): control point coordinate
Similarly, NURBS surfaces are defined as follows:

\[
S(u, v) = \frac{\sum_{i=0}^{n-1} \sum_{j=0}^{m-1} w_{i,j} N_{i,p}(u) N_{j,q}(v) P_{i,j}}{\sum_{i=0}^{n-1} \sum_{j=0}^{m-1} w_{i,j} N_{i,p}(u) N_{j,q}(v)}
\]

- \( n \) and \( m \): degrees
- \( w_{i,j} \): weight for control point \( i \)
- \( N_{i,p} \): b-spline basis function of degree \( p \) for \( i^{\text{th}} \) control point
- \( P_{i,j} \): control point coordinate
Surfaces of revolution

- Technique for modeling surfaces that are symmetrical across a particular axis
- Given some curve $C$ and an axis $V$, rotate $C$ about $V$ by some interval of degrees a certain number of times
  - The rotation amount and the number of rotations are defined by the user
  - Example: To create an open pentagonal prism, rotate a vertical line about the Y axis 5 times, each time by 72 degrees ($360/5$).
Modeling: Surfaces of revolution

- For a linear surface, create a quadrangle between each pair of control points. Combined, they form our surface.
- For an implicit surface, use something like a NURBS or Bézier curve as our base curve, then create rotated copies of its control points, connecting each copied control point to its source.
Modeling: Extrusions

- Given a “shell” curve $S$ and a “path” curve $P$, create copies of $S$ along $P$ so that the $S$ copies can be connected via quadrangles.
- Need to rotate $S$ as we traverse $P$ so that the extrusion loses as little volume as possible.
Modeling: Extrusions

- How do we determine the amount by which we should rotate S?
- Given a segment $AB$ of $P$ and another segment $BC$, find the axis and angle of rotation that takes us from an orientation along $AB$ to an orientation along $BC$
  - Axis = $\text{normalize}(AB \times BC)$
  - Angle = $\cos^{-1}(\text{dot}(AB, BC) / (||AB||*||BC||))$
- We can also apply this function to mesh faces
Half-Edge Face Extrusion

- Input: Half-edge face $F$ and an optional path curve $P$
- Result: $D*N + 1$ faces where $D$ is the number of edges on $F$ and $N$ is the number of control points of $P$
- In essence, extrude each edge of $F$ along $P$, forming a quadrangle at each step.
Extruding an edge

- Input: HE1
- Goal: Create a new face between HE1’s face and HE2’s face
Extruding an edge

- Step 1: Create two new vertices $V_3$ and $V_4$
Extruding an edge

- Step 2: Adjust HE1 so that it points to V3, and adjust HE1’s prev so that it points to V4.
Step 3: Create two new half-edges HE1B and HE2B
- HE1.sym = HE1B
- HE2.sym = HE2B
- HE1B.sym = HE1
- HE2B.sym = HE2
- HE1B.vert = V4
- HE2B.vert = V3
Extruding an edge

- Step 4: Create a new face $F$ and another two half-edges $HE_3$ and $HE_4$
  - Set the face pointers of $HE_{1B}$, $HE_{2B}$, $HE_3$, and $HE_4$ to $F$
  - $HE_3.\text{vert} = V3$
  - $HE_4.\text{vert} = V2$
  - $HE_{1B}.\text{next} = HE_4$
  - $HE_4.\text{next} = HE_{2B}$
  - $HE_{2B}.\text{next} = HE_3$
  - $HE_3.\text{next} = HE_{1B}$
Extruding an edge

- Step 5: Set syms of HE3 and HE4
  - If we are extruding multiple edges as part of a face extrusion, then we set the sym pointers of HE3 and HE4 to be half-edges created by other edge extrusions
  - If this is a single edge extrusion, then we create sym half-edges that point to a NULL face
An edge loop is a set of edges where the loop follows a middle edge in every four way junction. The loop ends as soon as it encounters another type of junction.
Selecting an Edge Loop

- Can only select an edge loop if vertices on the loop have exactly 4 incoming half edges (or else the loop is ambiguous - for example a cube).
- Edge loops only defined for quads (or sections of a mesh that are quad).
- How do you select an edge loop?
  - Given a starting half edge, HE0, the next half edge in the loop is HE0->next->sym->next.
- Inserting an edge loop
  - Find two edge loops, where the one you want to insert is in between. Split each face between those loops into two.
Beveling

Bevel Face:
Same as face extrusion, but with some additional parameters to determine its positioning.

Bevel Edge:
- Same as half-edge extrusion, with additional parameters to determine its positioning.
Beveling

- Given an edge and a $t$ between 0 and 1, find the faces on either side of the edge.
  - Find both edges on each face that are incident to the selected edge and compute a point on each edge based on $t$
  - Form a quad that spans the four created vertices, deleting (or shifting) our original edge
  - If the number of edges incident to our original vertices was more than 2, then make edges from the new vertices to the remaining extra edges.
- Similar process for beveling a face