Announcements

- Assignment 2
  - Programming is graded (Mean: 80%)

- Assignment 3
  - Programming was due
  - Theory is due Friday (Nov 21st)

- Assignment 4
  - You can start planning
What is interpolation?

Why do we need interpolation?

- Animation
- Curved surface
Keyframe Animation

- **Idea:** specify variables that describe *keyframes* and interpolate them over the sequence

(e.g. Assignment 1 & 2)
Interpolation Basics

- **Goal**: develop vocabulary of modeling primitives, that can extend meshes or global analytic shapes

- We would like to define curves that meet the following criteria:
  - Interaction should be natural and intuitive
  - Smoothness should be controllable
  - Analytic derivatives should exist and be easy to compute
  - Adjustable resolution (easy to zoom in and out)
  - Representation should be compact
Curves Basics

- **Interpolation**
  - Curve goes through “control points”

- **Approximation**
  - Curve approximates but does not go through “control points”

- **Extrapolation**
  - Extending curve beyond domain of control points
Continuity

- $C^n$ continuous function implies that $n$-th order derivatives exist

For animation purposes, $C^2$ continuous functions typically are sufficient

What is the continuity of the $n$-th order polynomial?
Linear Interpolation

- Simplest possible interpolation technique
  - Peace wise linear curve

- **Pros:**
  - Really simple to implement
  - Local (interpolation only depends on the closest two control points)

- **Cons:**
  - Only $C^1$ continuous (typically **bad for animation**)
Cubic Interpolation

- Consider a 2D cubic interpolation (a curve in 2D)

\[ c(t) = [x(t) \ y(t)] \]

where
\[ x(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 \]
\[ y(t) = b_0 + b_1 t + b_2 t^2 + b_3 t^3 \]

Alternatively,
\[ c(t) = [1 \ t \ t^2 \ t^3] \]
Cubic Interpolation

We have 8 unknowns (coefficients) how many 2D points do we need to constrain the curve?
Cubic Interpolation

\[ c\left(\frac{1}{3}\right) = [x_2, y_2] \]

\[ c(0) = [x_1, y_1] \]

\[ c\left(\frac{2}{3}\right) = [x_3, y_3] \]

\[ c(1) = [x_4, y_4] \]

\[ c(t) = \begin{bmatrix} 1 & t & t^2 & t^3 \end{bmatrix} \begin{bmatrix} a_0 & b_0 \\ a_1 & b_1 \\ a_2 & b_2 \\ a_3 & b_3 \end{bmatrix} \]

bases

coefficients
Cubic Interpolation

\[ c\left(\frac{1}{3}\right) = [x_2, y_2] \]

\[ c(0) = [x_1, y_1] \]

\[ c\left(\frac{2}{3}\right) = [x_3, y_3] \]

\[ c(1) = [x_4, y_4] \]
Cubic Interpolation

\[
\begin{bmatrix}
  x_1 & y_1 \\
  x_2 & y_2 \\
  x_3 & y_3 \\
  x_4 & y_4 \\
\end{bmatrix}
= \begin{bmatrix}
  1 & 0 & 0 & 0 \\
  1 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\
  1 & \frac{2}{3} & \frac{2}{3} & \frac{2}{3} \\
  1 & 1 & 1 & 1 \\
\end{bmatrix}
\begin{bmatrix}
  a_0 & b_0 \\
  a_1 & b_1 \\
  a_2 & b_2 \\
  a_3 & b_3 \\
\end{bmatrix}
\]

\[
c\left(\frac{1}{3}\right) = [x_2, y_2]
\]

\[
c(0) = [x_1, y_1]
\]

\[
c\left(\frac{2}{3}\right) = [x_3, y_3]
\]

\[
c(1) = [x_4, y_4]
\]
Cubic Interpolation

\[ c\left(\frac{1}{3}\right) = [x_2, y_2] \]

\[ c(0) = [x_1, y_1] \]

\[ c\left(\frac{2}{3}\right) = [x_3, y_3] \]

\[ c(1) = [x_4, y_4] \]

\[ c(t) = \begin{bmatrix} 1 & t & t^2 & t^3 \end{bmatrix} \]

bases

coefficients
Cubic Interpolation

- Consider a 2D cubic interpolant (a curve in 2D)

\[ c(t) = [x(t) \ y(t)] \]

where
\[ x(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 \]
\[ y(t) = b_0 + b_1 t + b_2 t^2 + b_3 t^3 \]

Alternatively we can place derivative constraints

\[ \tau(t) = \frac{d\ c(t)}{dt} = \frac{d}{dt} \begin{bmatrix} 1 & t & t^2 & t^3 \end{bmatrix} \]

\[
\begin{bmatrix}
  a_0 & b_0 \\
  a_1 & b_1 \\
  a_2 & b_2 \\
  a_3 & b_3 \\
\end{bmatrix}
\]

coefficients

bases
Cubic Interpolation

- Consider a 2D cubic interplant (a curve in 2D)

\[
c(t) = [x(t) \ y(t)]
\]

where

\[
x(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3
\]

\[
y(t) = b_0 + b_1 t + b_2 t^2 + b_3 t^3
\]

Alternatively we can place derivative constrains

\[
\tau(t) = \frac{d}{dt} c(t) = [0 \ 1 \ 2t \ 3t^2]
\]

same coefficients

different bases
Cubic Interpolation

\[
c\left(\frac{1}{3}\right) = [x_2, y_2] \quad \frac{dc}{dt}\left(\frac{1}{3}\right) = [x'_2, y'_2] = c(1) = [x_4, y_4]
\]
\[
c(0) = [x_1, y_1]
\]

\[
\begin{bmatrix}
  x_1 & y_1 \\
  x_2 & y_2 \\
  x_3 & y_3 \\
  x'_2 & y'_2
\end{bmatrix} = \begin{bmatrix}
  1 & 0 & 0 & 0 \\
  1 & \left(\frac{1}{3}\right) & \left(\frac{1}{3}\right)^2 & \left(\frac{1}{3}\right)^3 \\
  1 & 1 & 1 & 1 \\
  0 & 1 & 2\left(\frac{1}{3}\right) & 3\left(\frac{1}{3}\right)^2
\end{bmatrix} \begin{bmatrix}
  a_0 & b_0 \\
  a_1 & b_1 \\
  a_2 & b_2 \\
  a_3 & b_3
\end{bmatrix}
\]

coefficients
Cubic Interpolation

- What happens if there are more than 4 points?
  - There may **not be a solution** that goes through all the control points (or any of the control points)
  - Interpolation may not result in intuitive results

- Cubic interpolation is **global**
  - Changing one control point changes the interpolation for all points

- In general (at least for animation) local control is better
Beziers Curves

- **Idea:** cascade of linear interpolations

\[
\overline{\alpha}_0(t) = \overline{p}_0 + t(\overline{p}_1 - \overline{p}_0)
\]

\[
\overline{\alpha}_1(t) = \overline{p}_1 + t(\overline{p}_2 - \overline{p}_1)
\]

\[
c(t) = \overline{\alpha}_0(t) + t(\overline{\alpha}_1(t) - \overline{\alpha}_0(t))
\]

If we plug in all the expressions into \(c(t)\) we get a polynomial in terms of control points
Beziers Curves

- **Idea**: cascade of linear interpolations

\[
\alpha_0(t) = \overline{p}_0 + t(\overline{p}_1 - \overline{p}_0)
\]

\[
\alpha_1(t) = \overline{p}_1 + t(\overline{p}_2 - \overline{p}_1)
\]

\[
c(t) = \overline{p}_0 (1-t)^2 + 2 \overline{p}_1 (1-t)t + \overline{p}_2 t^2
\]

\[
= \sum_{i=0}^{2} \overline{p}_i B_i^2(t)
\]

i-th **Bernstein polynomial** of degree 2
Beziers Curves Generalization

Generalization to $N+1$ points $c(t) = \sum_{i=0}^{N} \overline{p}_i B_i^N(t)$

$$B_i^N(t) = \binom{N}{i}(1-t)^{N-i} t^i = \frac{N!}{(N-i)!i!} (1-t)^{N-i} t^i$$
Bernstein Polynomials of Degree 3

- **Note:** Bezier curve with 4 points will be a combination of these curves.
Bezier Curves Properties

- Bezier curve interpolates between the first and the last point, but not the intermediate points.

- Bezier curves have **nice properties** that make them useful in graphics:
  - **Affine invariance**: affine transformation of the curve implies transformation of control points (nothing else).
  - **Convex hall property**: any point on a curve is by definition a convex combination of the control points, hence the curve must be inside the (convex) polygon defined by those points.
  - **Linear precision**: as convex polygon approximates the line, so will the curve.
  - **Variation Diminishing**: No line has more intersections with the curve than with control points (no accessive fluctuations).
Derivatives of Bezier Curves

\[ c(t) = \sum_{i=0}^{N} \overline{p}_i B_i^N(t) \quad B_i^N(t) = \binom{N}{i} (1-t)^{N-i} t^i = \frac{N!}{(N-i)!i!} (1-t)^{N-i} t^i \]

convex sum of points, hence is a point

- We want to differentiate with respect to \( t \)

\[ \tau(t) = \frac{d}{dt} c(t) = \frac{d}{dt} \sum_{i=0}^{N} \overline{p}_i B_i^N(t) \]

with some work

\[ = \sum_{i=0}^{N-1} (\overline{p}_{i+1} - \overline{p}_i) B_i^{N-1}(t) \]

convex sum of vectors, hence is a vector
Derivatives of Bezier Curves

**Property:** tangents at the end points of a Bezier curve are always parallel to vector from the end point to the adjacent point.
Final word on Bezier curves

- **Pros:**
  - Has nice properties (e.g. affine invariance)
  - Derivatives are easy to compute

- **Cons:**
  - Tough to control a high-order polynomial
  - Global (curve is a function of all control points)
Catmull-Rom Splines

- **Idea:** piecewise cubic curves of degree-3 with $C^1$ continuity

- A user specifies points and the tangent at each point is set to be parallel to the vector between adjacent points

- $k$ is the set by the user parameter, that determines the “tension” of the curve
Catmull-Rom Splines

- To interpolate a value for the point between \( p_j \) and \( p_{j+1} \) one needs to consider 4 bits of information:

\[
\begin{align*}
\bar{p}_j \\
\bar{p}_{j+1} \\
k(\bar{p}_{j+1} - \bar{p}_{j-1}) \\
k(\bar{p}_{j+2} - \bar{p}_j)
\end{align*}
\]
Catmull-Rom Splines

To interpolate a value for the point between \( p_j \) and \( p_{j+1} \) one needs to consider 4 bits of information.

\[
\overline{p}_j, \overline{p}_{j+1}, k(\overline{p}_{j+1} - \overline{p}_{j-1}), k(\overline{p}_{j+2} - \overline{p}_j)
\]

4 points lead to cubic interplant (see lecture notes for details)