Modelling: Curves

Week 11, Wed Mar 23

http://www.ugrad.cs.ubc.ca/~cs314/Vjan2005
News

- reminder: my office hours today 3:45
- proposals due today 6pm
News

- midterms: min 33, max 98, median 68
Reading

- FCG Chap 13
Parametric Curves

- parametric form for a line:

\[ x = x_0 t + (1 - t) x_1 \]
\[ y = y_0 t + (1 - t) y_1 \]
\[ z = z_0 t + (1 - t) z_1 \]

- \( x, y \) and \( z \) are each given by an equation that involves:
  - parameter \( t \)
  - some user specified control points, \( x_0 \) and \( x_1 \)

- this is an example of a parametric curve
Splines

- A **spline** is a parametric curve defined by *control points*
  - Term “spline” dates from engineering drawing, where a spline was a piece of flexible wood used to draw smooth curves
  - Control points are *adjusted by the user* to control shape of curve
Splines - History

- draftsman used ‘ducks’ and strips of wood (splines) to draw curves
- wood splines have second-order continuity, pass through the control points

a duck (weight)

ducks trace out curve
Hermite Spline

- *hermite spline* is curve for which user provides:
  - endpoints of curve
  - parametric derivatives of curve at endpoints
    - parametric derivatives are $dx/dt$, $dy/dt$, $dz/dt$
  - more derivatives would be required for higher order curves
Hermite Cubic Splines

- example of knot and continuity constraints
Hermite Spline (2)

- say user provides \( x_0, x_1, x'_0, x'_1 \)
- cubic spline has degree 3, is of the form:
  \[
  x = at^3 + bt^2 + ct + d
  \]
  for some constants a, b, c and d derived from the control points, but how?
- we have constraints:
  - curve must pass through \( x_0 \) when \( t=0 \)
  - derivative must be \( x'_0 \) when \( t=0 \)
  - curve must pass through \( x_1 \) when \( t=1 \)
  - derivative must be \( x'_1 \) when \( t=1 \)
Hermite Spline (3)

- solving for the unknowns gives

\[ a = -2x_1 + 2x_0 + x'_1 + x'_0 \]
\[ b = 3x_1 - 3x_0 - x'_1 - 2x'_0 \]
\[ c = x'_0 \]
\[ d = x_0 \]

- rearranging gives

\[
x = x_1(-2t^3 + 3t^2) + x_0(2t^3 - 3t^2 + 1) + x'_1(t^3 - t^2) + x'_0(t^3 - 2t^2 + t)
\]

or

\[
x = \begin{bmatrix} x_1 & x_0 & x'_1 & x'_0 \end{bmatrix} \begin{bmatrix} -2 & 3 & 0 & 0 \\ 2 & -3 & 0 & 1 \\ 1 & -1 & 0 & 0 \\ 1 & -2 & 1 & 0 \end{bmatrix} \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix}
\]
Basis Functions

- A point on a Hermite curve is obtained by multiplying each control point by some function and summing.
- Functions are called *basis functions*.
Sample Hermite Curves
Splines in 2D and 3D

- so far, defined only 1D splines: 
  \[ x = f(t; x_0, x_1, x'_0, x'_1) \]

- for higher dimensions, define control points in higher dimensions (that is, as vectors)

\[
\begin{bmatrix}
  x \\
  y \\
  z
\end{bmatrix} = 
\begin{bmatrix}
  x_1 & x_0 & x'_1 & x'_0 \\
  y_1 & y_0 & y'_1 & y'_0 \\
  z_1 & z_0 & z'_1 & z'_0
\end{bmatrix} \begin{bmatrix}
  -2 & 3 & 0 & 0 \\
  2 & -3 & 0 & 1 \\
  1 & -1 & 0 & 0 \\
  1 & -2 & 1 & 0
\end{bmatrix} \begin{bmatrix}
  t^3 \\
  t^2 \\
  t \\
  1
\end{bmatrix}
\]
Bézier Curves

- similar to Hermite, but more intuitive definition of endpoint derivatives
- four control points, two of which are knots
Bézier Curves

- derivative values of Bezier curve at knots dependent on adjacent points

\[ \nabla p_1 = 3(p_2 - p_1) \]
\[ \nabla p_4 = 3(p_4 - p_3) \]
Bézier vs. Hermite

- can write Bezier in terms of Hermite
  - note: just matrix form of previous

\[
\begin{bmatrix}
  x_1 & y_1 \\
  x_2 & y_2 \\
  \frac{dx_1}{dt} & \frac{dy_1}{dt} \\
  \frac{dx_2}{dt} & \frac{dy_2}{dt}
\end{bmatrix}
= \begin{bmatrix}
  1 & 0 & 0 & 0 \\
  0 & 0 & 0 & 1 \\
  -3 & 3 & 0 & 0 \\
  0 & 0 & -3 & 3
\end{bmatrix}
\begin{bmatrix}
  x_1 & y_1 \\
  x_2 & y_2 \\
  x_3 & y_3 \\
  x_4 & y_4
\end{bmatrix}
\]

\(G_{\text{Hermite}}\) \(G_{\text{Bezier}}\)
Bézier vs. Hermite

- Now substitute this in for previous Hermite
Bézier Basis, Geometry Matrices

\[
\begin{bmatrix}
  a_x & a_y \\
b_x & b_y \\
c_x & c_y \\
d_x & d_y \\
\end{bmatrix}
= \begin{bmatrix}
  -1 & 3 & -3 & 1 \\
  3 & -6 & 3 & 0 \\
 -3 & 3 & 0 & 0 \\
  1 & 0 & 0 & 0 \\
\end{bmatrix}
\begin{bmatrix}
x_1 & y_1 \\
x_2 & y_2 \\
x_3 & y_3 \\
x_4 & y_4 \\
\end{bmatrix}
\]

\[M_{\text{Bez}} \quad G_{\text{Bez}}\]

- but why is \(M_{\text{Bez}}\) a good basis matrix?
look at blending functions

family of polynomials called order-3 Bernstein polynomials
- \( C(3, k) t^k (1-t)^{3-k}; 0 \leq k \leq 3 \)
- all positive in interval \([0,1]\)
- sum is equal to 1

\[
p(t) = \begin{bmatrix}
(1-t)^3 \\
3t(1-t)^2 \\
3t^2(1-t) \\
t^3
\end{bmatrix}^T \begin{bmatrix}
p_1 \\
p_2 \\
p_3 \\
p_4
\end{bmatrix}
\]
Bézier Blending Functions

- every point on curve is linear combination of control points
- weights of combination are all positive
- sum of weights is 1
- therefore, curve is a convex combination of the control points
Bézier Curves

- curve will always remain within convex hull (bounding region) defined by control points
Bézier Curves

- Interpolate between first, last control points
- 1\textsuperscript{st} point’s tangent along line joining 1\textsuperscript{st}, 2\textsuperscript{nd} pts
- 4\textsuperscript{th} point’s tangent along line joining 3\textsuperscript{rd}, 4\textsuperscript{th} pts
Comparing Hermite and Bézier

Hermite

Bézier
Comparing Hermite and Bezier

demo: [www.siggraph.org/education/materials/HyperGraph/modeling/splines/demoprog/curve.html](http://www.siggraph.org/education/materials/HyperGraph/modeling/splines/demoprog/curve.html)
Rendering Bezier Curves: Simple

- evaluate curve at fixed set of parameter values, join points with straight lines
- advantage: very simple
- disadvantages:
  - expensive to evaluate the curve at many points
  - no easy way of knowing how fine to sample points, and maybe sampling rate must be different along curve
  - no easy way to adapt: hard to measure deviation of line segment from exact curve
Rendering Beziers: Subdivision

- A cubic Bezier curve can be broken into two shorter cubic Bezier curves that exactly cover the original curve.
- Suggests a rendering algorithm:
  - Keep breaking the curve into sub-curves.
  - Stop when control points of each sub-curve are nearly collinear.
  - Draw the control polygon: polygon formed by control points.
Sub-Dividing Bezier Curves

- step 1: find the midpoints of the lines joining the original control vertices. Call them $M_{01}$, $M_{12}$, $M_{23}$
Sub-Dividing Bezier Curves

- step 2: find the midpoints of the lines joining $M_{01}$, $M_{12}$ and $M_{12}$, $M_{23}$. call them $M_{012}$, $M_{123}$
Sub-Dividing Bezier Curves

- step 3: find the midpoint of the line joining $M_{012}$, $M_{123}$. call it $M_{0123}$
Sub-Dividing Bezier Curves

- curve \( P_0, M_{01}, M_{012}, M_{0123} \) exactly follows original from \( t=0 \) to \( t=0.5 \)
- curve \( M_{0123}, M_{123}, M_{23}, P_3 \) exactly follows original from \( t=0.5 \) to \( t=1 \)
Sub-Dividing Bezier Curves

- continue process to create smooth curve
de Casteljau’s Algorithm

- can find the point on a Bezier curve for any parameter value $t$ with similar algorithm
  - for $t=0.25$, instead of taking midpoints take points 0.25 of the way

demo: [www.saltire.com/applets/advanced_geometry/spline/spline.htm](http://www.saltire.com/applets/advanced_geometry/spline/spline.htm)
Longer Curves

- A single cubic Bezier or Hermite curve can only capture a small class of curves
  - At most 2 inflection points

- One solution is to raise the degree
  - Allows more control, at the expense of more control points and higher degree polynomials
  - Control is not local; one control point influences entire curve

- Better solution is to join pieces of cubic curve together into piecewise cubic curves
  - Total curve can be broken into pieces, each of which is cubic
  - Local control: each control point only influences a limited part of the curve
  - Interaction and design is much easier
Piecewise Bezier: Continuity Problems

demo: [www.cs.princeton.edu/~min/cs426/jar/bezier.html](http://www.cs.princeton.edu/~min/cs426/jar/bezier.html)
Continuity

- when two curves joined, typically want some degree of continuity across knot boundary
  - C0, “C-zero”, point-wise continuous, curves share same point where they join
  - C1, “C-one”, continuous derivatives
  - C2, “C-two”, continuous second derivatives
Geometric Continuity

- derivative continuity is important for animation
  - if object moves along curve with constant parametric speed, should be no sudden jump at knots
- for other applications, *tangent continuity* suffices
  - requires that the tangents point in the same direction
  - referred to as $G^1$ geometric continuity
  - curves could be made $C^1$ with a re-parameterization
  - geometric version of $C^2$ is $G^2$, based on curves having the same radius of curvature across the knot
Achieving Continuity

- **Hermite curves**
  - user specifies derivatives, so $C^1$ by sharing points and derivatives across knot

- **Bezier curves**
  - they interpolate endpoints, so $C^0$ by sharing control pts
  - introduce additional constraints to get $C^1$
    - parametric derivative is a constant multiple of vector joining first/last 2 control points
    - so $C^1$ achieved by setting $P_{0,3}=P_{1,0}=J$, and making $P_{0,2}$ and $J$ and $P_{1,1}$ collinear, with $J-P_{0,2}=P_{1,1}-J$
    - $C^2$ comes from further constraints on $P_{0,1}$ and $P_{1,2}$

- leads to...
B-Spline Curve

- start with a sequence of control points
- select four from middle of sequence
  \((p_{i-2}, p_{i-1}, p_i, p_{i+1})\)
  - Bezier and Hermite goes between \(p_{i-2}\) and \(p_{i+1}\)
  - B-Spline doesn’t interpolate (touch) any of them but approximates the going through \(p_{i-1}\) and \(p_i\)
B-Spline

- by far the most popular spline used
- \( C_0, C_1, \) and \( C_2 \) continuous

demo: [www.siggraph.org/education/materials/HyperGraph/modeling/splines/demoprog/curve.html](http://www.siggraph.org/education/materials/HyperGraph/modeling/splines/demoprog/curve.html)
B-Spline

- locality of points

*Figure 10-41*
Local modification of a B-spline curve. Changing one of the control points in (a) produces curve (b), which is modified only in the neighborhood of the altered control point.