CS4620/5620: Lecture 29

Splines

Announcements

• 4621
  – Today
  – Next Friday

• PPA 2 will be released this weekend

• HWV 3 will be released on Monday
**Hermite to Bézier**

\[ p_0 = q_0 \]
\[ p_1 = q_3 \]
\[ v_0 = 3(q_1 - q_0) \]
\[ v_1 = 3(q_3 - q_2) \]

\[
\begin{bmatrix}
a \\
b \\
c \\
d
\end{bmatrix} =
\begin{bmatrix}
-1 & 3 & -3 & 1 \\
3 & -6 & 3 & 0 \\
-3 & 3 & 0 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix}
\begin{bmatrix}
q_0 \\
q_1 \\
q_2 \\
q_3
\end{bmatrix}
\]

**Bézier matrix**

\[
p(t) = [t^3 \ t^2 \ t \ 1]
\begin{bmatrix}
-1 & 3 & -3 & 1 \\
3 & -6 & 3 & 0 \\
-3 & 3 & 0 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix}
\begin{bmatrix}
p_0 \\
p_1 \\
p_2 \\
p_3
\end{bmatrix}
\]

– note that these are the Bernstein polynomials

\[ C(n,k) \ t^k \ (1 - t)^{n-k} \]

and that defines Bézier curves for any degree
Bézier basis

Control

• Local control
  – changing control point only affects a limited part of spline
  – without this, splines are very difficult to use
  – many likely formulations lack this
Control

• Convex hull property
  – convex hull = smallest convex region containing points
  • think of a rubber band around some pins
  – some splines stay inside convex hull of control points
  • simplifies clipping, culling, picking, etc.

Convex hull

• If basis functions are all positive and sum to 1, the spline has the convex hull property

  – if any basis function is ever negative, no convex hull prop.
  • proof: take the other three points at the same place
Affine invariance

- Transforming the control points is the same as transforming the curve
  – extremely convenient in practice…

Continuity

- Smoothness can be described by degree of continuity
  – zero-order ($G^0$): position matches from both sides
  – first-order ($G^1$): tangent also matches from both sides
  – second-order ($G^2$): curvature also matches from both sides
  – $G^n$ vs. $C^n$
Continuity

• Parametric continuity ($C$)
  – is continuity of coordinate functions, e.g., $x(t)$, $y(t)$, $z(t)$

• Geometric continuity ($G$)
  – is continuity of the geometric curve itself

• Neither form of continuity is guaranteed by the other
  – Typically $C^1$ implies $G^1$
    • Can be $C^1$ but not $G^1$ when $p(t)$ comes to a halt (next slide)
    • Can be $G^1$ but not $C^1$ when the tangent vector changes length abruptly

Geometric vs. parametric continuity
Continuity

- A curve is said to be $C^n$ continuous if $p(t)$ is continuous, and all derivatives of $p(t)$ up to and including degree $n$ have the same direction and magnitude:

$$\lim_{x \to t_-} p^{(m)}(x) = \lim_{x \to t_+} p^{(m)}(x), \quad m = 0 \ldots n$$

- $G^n$ continuity is like $C^n$ but only requires the derivatives to have the same direction:

$$\lim_{x \to t_-} p^{(n)}(x) = k \lim_{x \to t_+} p^{(n)}(x), \quad \text{for some } k > 0$$

Chaining spline segments

- Hermite curves are convenient because they can be made long easily

- Bézier curves are convenient because their controls are all points and they have nice properties
  - but they interpolate every 4th point, which is a little odd
Chaining Bézier splines

- No continuity built in
- Achieve $C^1$ using collinear control points

Cubic Bézier splines

- Very widely used type, especially in 2D
  - e.g. it is a primitive in PostScript/PDF
- Can represent $C^1$ and/or $G^1$ curves with corners
- Can easily add points at any position

- Disadvantage
  - Special points
  - Only $C^1$
Evaluating splines for display

• Need to generate a list of line segments to draw
  – generate efficiently
  – use as few as possible
  – guarantee approximation accuracy

• Approaches
  – recursive subdivision (easy to do adaptively)
  – uniform sampling (easy to do efficiently)

Rendering the curve: Option 1

• Uniformly sample in t

• Problem
  • may oversample smooth regions: slow
  • may undersample highly curved regions: faceted rendering
Interpolation property

- $C(t^*)$ can be evaluated using interpolation
- $C(t) = (1-t)^3 P_0 + 3 (1-t)^2 P_1 + 3 t^2 (1-t) P_2 + t^3 P_3$

Interpolation property

Red$Pt = (1-t) P_i + t P_{i+1}$
Blue$Pt = (1-t) \text{Red}Pt_i + t \text{Red}Pt_{i+1}$
Yellow$Pt = (1-t) \text{Blue}Pt_i + t \text{Blue}Pt_{i+1}$
De Casteljau algorithm

- Adaptive subdivision!

Recursive algorithm: Option 2

```c
void DrawRecBezier (float eps) {
    if Linear (curve, eps)
        DrawLine (curve);
    else
        SubdivideCurve (curve, leftC, rightC);
        DrawRecBezier (leftC, eps);
        DrawRecBezier (rightC, eps);
}
```
Termination Criteria

• Test for linearity
  – distance between control points
  – distance of control points from line

\[ \begin{align*}
  d_0 &< \varepsilon \\
  d_1 &< \varepsilon
\end{align*} \]

B-splines

• We may want more continuity than \( C^1 \)

• B-splines are a clean, flexible way of making long splines with arbitrary order of continuity

• Various ways to think of construction
  – a simple one is convolution

• An approximating spline
Deriving the B-Spline

• Approached from a different tack than Hermite-style constraints
  – Want a cubic spline; therefore 4 active control points
  – Want $C^2$ continuity
  – Turns out that is enough to determine everything

Cubic B-spline matrix

$$p(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \cdot \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \begin{bmatrix} p_{k-1} \\ p_k \\ p_{k+1} \\ p_{k+2} \end{bmatrix}$$
Cubic B-spline curves

- Treat points uniformly
- $C^2$ continuity
- $C(t) = \frac{[(1-t)^3 P_{i-3} + (3t^3 - 6t^2 + 4)P_{i-2} + (-3t^3 + 3t^2 + 3t + 1) P_{i-1} + t^3 P_i]}{6}$
- Notice blending functions still add to 1

Cubic B-spline basis

- B-spline from each 4-point sequence matches previous, next sequence with $C^2$ continuity!
- Treats all points uniformly
Cubic B-spline basis

• B-spline from each 4-point sequence matches previous, next sequence with $C^2$ continuity!
• Treats all points uniformly

Rendering the spline-curve

• Given B-spline points $d_{-1}, \ldots, d_{L+1}$
• Compute Bézier points $b_0, \ldots, b_{3L}$
• Use De Casteljau algorithm to render
Equations and boundary conditions

- Equations
  - \( b_{3i} = \frac{b_{3i-1} + b_{3i+1}}{2} \)
  - \( b_{3i-1} = \frac{d_{i-1}}{3} + 2d_i/3 \)
  - \( b_{3i-2} = 2d_{i-1}/3 + d_i/3 \)

- Boundary conditions
  - \( b_0 = d_{-1}, b_1 = d_0, b_2 = \frac{d_0 + d_1}{2} \)
  - \( b_{3L} = d_{L+1}, b_{3L-1} = d_L, b_{3L-2} = \frac{d_{L-1} + d_L}{2} \)