CS 536
Computer Graphics

B-Splines and NURBS
Week 4, Lecture 5

David Breen, William Regli and Maxim Peysakhov
Department of Computer Science
Drexel University

Outline

• More Types of Curves
  – Splines
  – B-splines
  – NURBS
• Knot sequences
• Effects of the weights

Splines

• Popularized in late 1960s in US Auto industry (GM)
  – R. Riesenfeld (1972)
  – W. Gordon
• Origin: the thin wood or metal strips used in building/ship construction
• Goal: define a curve as a set of piecewise simple polynomial functions connected together

Natural Splines

• Mathematical representation of physical splines
• C^2 continuous
• Interpolate all control points
• Have Global control (no local control)

B-splines: Basic Ideas

• Similar to Bézier curves
  – Smooth blending function times control points
• But:
  – Blending functions are non-zero over only a small part of the parameter range (giving us local support)
  – When nonzero, they are the “concatenation” of smooth polynomials. (They are piecewise!)

B-spline: Benefits

• User defines degree
  – Independent of the number of control points
• Produces a single piecewise curve of a particular degree
  – No need to stitch together separate curves at junction points
• Continuity comes for free!
B-splines

- Defined similarly to Bézier curves
  - $p_i$ are the control points
  - Computed with basis functions (Basis-splines)
- B-spline basis functions are blending functions
- Each point on the curve is defined by the blending of the control points
  (B_i is the i-th B-spline blending function)

$$p(t) = \sum_{i=0}^{m} B_{i,d}(t) p_i$$
- No limits on the value of $t$
- B_i is zero for most values of $t$!

B-spline Blending Functions

- $B_{i,0}(t)$ is a step function that is 1 in the interval $[u_k, u_{k+1})$
- $B_{i,1}(t)$ spans two intervals and is a piecewise linear function that goes from 0 to 1 (and back)
- $B_{i,2}(t)$ spans three intervals and is a piecewise quadratic that grows from 0 to 1/4, then up to 3/4 in the middle of the second interval, back to 1/4, and back to 0
- $B_{i,3}(t)$ is a cubic that spans four intervals growing from 0 to 1/6 to 2/3, then back to 1/6 and to 0

Transitions at Knots

- As one blending function goes to zero, another smoothly becomes non-zero

B-spline: Cox-deBoor Recursion

- Cox-deBoor Algorithm: defines the blending functions for spline curves (not limited to deg 3)
  - curves are weighted avgs of lower degree curves
- Let $B_{i,d}(t)$ denote the i-th blending function for a B-spline of degree $d$, then:

$$B_{i,d}(t) = \begin{cases} 1, & \text{if } t_k \leq t < t_{k+1} \\ 0, & \text{otherwise} \end{cases}$$

$$B_{i,d}(t) = \frac{t-t_k}{t_{k+1}-t_k} B_{i,d-1}(t) + \frac{t_{k+1}-t}{t_{k+1}-t_k} B_{i+1,d-1}(t)$$

B-spline Blending Functions: Example for 2nd Degree Splines

- Note: can’t define a polynomial with these properties (both 0 and non-zero for ranges)
- Idea: subdivide the parameter space into intervals and build a piecewise polynomial
  - Each interval gets different polynomial function

B-spline Blending Functions: Example for 3rd Degree Splines

- Observe:
  - in $t=0$ to $t=1$ range just four of the functions are non-zero
  - all are $\geq 0$ and sum to 1, hence the convex hull property holds for each curve segment of a B-spline

B-spline: Cox-deBoor Recursion

- Cox-deBoor Algorithm: defines the blending functions for spline curves (not limited to deg 3)
  - curves are weighted avgs of lower degree curves
- Let $B_{i,d}(t)$ denote the i-th blending function for a B-spline of degree $d$, then:

$$B_{i,d}(t) = \begin{cases} 1, & \text{if } t_k \leq t < t_{k+1} \\ 0, & \text{otherwise} \end{cases}$$

$$B_{i,d}(t) = \frac{t-t_k}{t_{k+1}-t_k} B_{i,d-1}(t) + \frac{t_{k+1}-t}{t_{k+1}-t_k} B_{i+1,d-1}(t)$$

Transitions at Knots

- As one blending function goes to zero, another smoothly becomes non-zero
Example: Creating a B-spline Curve Segment

B-splines: Knot Selection

• Instead of working with the parameter space \(0 \leq t \leq 1\), use \(t_{\min} \leq t_0 \leq t_1 \leq t_2 \ldots \leq t_{m-1} \leq t_{\max}\)

• The knot points
  – joint points between curve segments, \(Q\)
  – Each has a knot value
  – \(m-1\) knots for \(m+1\) points

Uniform B-splines: Setting the Options

• Specified by
  – \(m \geq 3\) control points, \(P_0 \ldots P_m\)
  – \(m+1\) cubic polynomial curve segments, \(Q_3 \ldots Q_m\)
  – \(m-1\) knot points, \(t_3 \ldots t_{m+1}\)
  – segments \(Q_i\) of the B-spline curve
    • defined over a knot interval \([t_{i-1}, t_i]\)
    • defined by 4 of the control points, \(P_{i-3} \ldots P_i\)
  – segments \(Q_i\) of the B-spline curve are blended together into smooth transitions via (the new & improved) blending functions

Example: Creating a B-spline

B-splines: Knot Sequences

• Even distribution of knots
  – uniform B-splines
  – Curve does not interpolate end points
    • first blending function not equal to 1 at \(t=0\)
  – Uneven distribution of knots
    – non-uniform B-splines
      – Allows us to tie down the endpoints by repeating knot values
        (in Cox-deBoor, \(0/0=0\))
      – If a knot value is repeated, it increases the effect (weight) of the blending function at that point
      – If knot is repeated \(d\) times, blending function converges to 1 and the curve interpolates the control point

B-splines: Cox-deBoor Recursion

• Cox-deBoor Algorithm: defines the blending functions for spline curves (not limited to deg 3)
  – curves are weighted avgs of lower degree curves
• Let \(B_{i,d}(t)\) denote the \(i\)-th blending function for a B-spline of degree \(d\), then:

\[
B_{i,d}(t) = \begin{cases} 1, & \text{if } t_{i-1} \leq t < t_{i+1} \\ 0, & \text{otherwise} \end{cases}
\]

\[
B_{i,d}(t) = \begin{cases} \frac{t-t_{i-1}}{t_{i+1}-t_{i-1}} B_{i-1,d}(t) + \frac{t_{i+1}-t}{t_{i+1}-t_{i+1}} B_{i+1,d}(t) \\ 0, & \text{otherwise} \end{cases}
\]
Creating a Non-Uniform B-spline: Knot Selection

- Given curve of degree $d=3$, with $m+1$ control points $p_0, \ldots, p_m$
  - first, create $m+d$ knot values
  - use knot values $(0,0,1,2, \ldots, m-2, m-1,m-1,m-1)$
    (adding two extra 0’s and 1’s)
  - Note:
    - Causes Cox-deBoor to give added weight in blending to the first and last points when $t$ is near $t_{min}$ and $t_{max}$

B-splines: Multiple Knots

- Knot Vector $(0.0, 0.0, 0.0, 3.0, 4.0, 5.0, 6.0, 7.0)$
- Several consecutive knots get the same value
- Changes the basis functions!

B-spline Summary

$$ p(t) = \sum_{i=0}^{d} B_{d,i}(t) p_i $$

Watching Effects of Knot Selection

- 9 knot points (initially)
  - Note: knots are distributed parametrically based on $t$, hence why they "move"
- 10 control points
- Curves have as many segments as they have non-zero intervals in $u$

B-splines: Local Control Property

- Local Control
  - polynomial coefficients depend on a few points
  - moving control point $(P_d)$ affects only local curve
  - Why: Based on curve def'n, affected region extends at most 2 knot points away

B-splines: Local Control Property

- Knot
- Control point

Pic s / Math courtesy of G. Farin @ ASU


B-splines: Convex Hull Property

- The effect of multiple control points on a uniform B-spline curve

\[
\Sigma_{k=0}^{3} a_k u^k
\]

B-splines: Continuity

- Derivatives are easy for cubics
  \[
p(u) = \sum_{k=0}^{3} a_k u^k
\]
  \[
p'(u) = c_1 + 2c_2 u + 3c_3 u^2
\]

Easy to show \( C^0, C', C^2 \)

B-splines: Setting the Options

- How to space the knot points?
  - Uniform
    - equal spacing of knots along the curve
  - Non-Uniform
- Which type of parametric function?
  - Rational
    - \( x(t), y(t), z(t) \) defined as ratio of cubic polynomials
  - Non-Rational

NURBS

- At the core of several modern CAD systems
  - i-DEAS, Pro/E, Alpha_1
- Describes analytic and freeform shapes
- Accurate and efficient evaluation algorithms
- Invariant under affine and perspective transformations

Benefits of Rational Spline Curves

- Invariant under rotation, scale, translation, perspective transformations
  - transform just the control points, then regenerate the curve
  - (non-rationals only invariant under rotation, scale and translation)
- Can precisely define the conic sections and other analytic functions
  - conics require quadratic polynomials
  - conics only approximate with non-rationals

NURBS

- Non-uniform Rational B-splines: NURBS
  - Basic idea: four dimensional non-uniform B-splines, followed by normalization via homogeneous coordinates
    - If \( P_i = [x, y, z, 1] \), results are invariant wrt perspective projection
  - Also, recall in Cox-deBoor, knot spacing is arbitrary
    - knots are close together, influence of some control points increases
    - Duplicate knots can cause points to interpolate
    - e.g. Knots = \( \{0, 0, 0, 0, 1, 1, 1, 1\} \) create a Bezier curve
Rational Functions

- Cubic curve segments
  \[ x(t) = \frac{X(t)}{W(t)}, \quad y(t) = \frac{Y(t)}{W(t)}, \quad z(t) = \frac{Z(t)}{W(t)} \]
  where \( X(t), Y(t), Z(t), W(t) \) are all cubic polynomials with control points specified in homogeneous coordinates, \([x,y,z,w]\)
- Note: for 2D case, \( z(t) = 0 \)

Rational Functions: Example

- Example:
  - rational function: a ratio of polynomials
  - a rational parameterization in \( u \) of a unit circle in xy-plane:
    \[ x(u) = \frac{1-u^2}{1+u^2}, \quad y(u) = \frac{2u}{1+u^2}, \quad z(u) = 0 \]
  - a unit circle in 3D homogeneous coordinates:
    \[ x[u] = 1-u^2, \quad y[u] = 2u, \quad z[u] = 0, \quad w[u] = 1 + u^2 \]

NURBS: Notation Alert

- Depending on the source/reference
  - Blending functions are either \( B_{i,j}(u) \) or \( N_{i,j}(u) \)
  - Parameter variable is either \( u \) or \( t \)
  - Curve is either \( C \) or \( P \) or \( Q \)
  - Control Points are either \( P_i \) or \( B_i \)
  - Variables for order, degree, number of control points etc are frustratingly inconsistent
    - \( k, i, j, m, n, p, L, d, \ldots \)

NURBS

- A \( d \)-th degree NURBS curve \( C \) is defined as:
  \[ C(u) = \frac{\sum_{i=0}^{n} w_i B_{i,d}(u) P_i}{\sum_{i=0}^{n} w_i B_{i,d}(u)} \]
  Where
  - control points \( P_i \)
  - \( d \)-th degree B-spline blending function \( B_{i,d}(u) \)
  - the weight, \( w_i \) for control point \( P_i \)
    (when all \( w = 1 \), we have a B-spline curve)

Observe: Weights Induce New Rational Basis Functions, \( R \)

- Setting:
  \[ R_i(u) = \frac{w_i B_{i,d}(u)}{\sum w_i B_{i,d}(u)} \]

  Allows us to write: \( C(u) = \sum R_{i,j}(u) P_i \)
  Where \( R_{i,j}(u) \) are rational basis functions
  - piecewise rational basis functions on \( \Xi \in \{0,1\} \)
  - weights are incorporated into the basis fcts
Geometric Interpretation of NURBS

- With Homogeneous coordinates, a rational \( n \)-D curve is represented by polynomial curve in \((n+1)\)-D.
- Homogeneous 3D control points are written as:
  \[ P_i^w = w_i x_i, w_i y_i, w_i z_i, w_i \]
in 4D where \( w \neq 0 \)
- To get \( P_i \), divide by \( w_i \)
  - a perspective transform with center at the origin
- Note: weights can allow final curve shape to go outside the convex hull (i.e. negative \( w \)).

\[ \begin{align*}
\text{NURBS: Examples} & \quad \text{Uniform Knot Vector} \\
\{0.0, 0.0, 0.0, 3.0, 4.0, 5.0, 6.0, 7.0\} \\
\text{Several consecutive knots get the same value} \\
\text{Bunches up the curve and forces it to interpolate}
\end{align*} \]

\[ \begin{align*}
\text{NURBS: Examples} & \quad \text{Non-Uniform Knot Vector} \\
\{0.0, 1.0, 2.0, 3.75, 4.0, 4.25, 6.0, 7.0\} \\
\text{Several consecutive knots get the same value} \\
\text{Bunches up the curve and forces it to interpolate} \\
\text{Can be done midcurve}
\end{align*} \]

The Effects of the Weights

- \( w_i \) of \( P_i \) effects only the range \([u, u_{i+k+1}]\)
- If \( w_i = 0 \) then \( P_i \) does not contribute to \( C \)
- If \( w_i \) increases, point \( B \) and curve \( C \) are pulled toward \( P_i \) and pushed away from \( P_j \)
- If \( w_i \) decreases, point \( B \) and curve \( C \) are pushed away from \( P_i \) and pulled toward \( P_j \)
- If \( w_i \) approaches infinity then \( B \) approaches 1 and \( B_i \to P_i \) if \( u \) in \([u, u_{i+k+1}]\)

\[ \begin{align*}
\text{The Effects of the Weights} & \quad \text{Increased weight pulls the curve toward} \quad B_3 \\
\text{Increased weight pulls the curve toward} \quad B_3
\end{align*} \]
Programming Assignment 2

- Process command-line arguments
- Read in 3D input points and tangents
- Compute Bezier control points for curves defined by each two input points
- Modify tangents with tension parameter
- Use HW1 code to compute points on each Bezier curve
- Each Bezier curve should be a polyline
- Output points by printing them to the console as an IndexedLineSet with multiple polylines, and control points as spheres in Open Inventor format