CS 536
Computer Graphics

Hermite Curves, B-Splines and NURBS

Week 2, Lecture 4
David Breen, William Regli and Maxim Peysakhov
Department of Computer Science
Drexel University

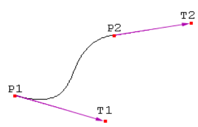
Additional slides from Don Fussell, University of Texas

Outline

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

Hermite Curve

- 3D curve of polynomial bases
- Geometrically defined by position and tangents at end points
- No convex hull guarantees
- Able to tangent-continuous (C^1) composite curve



Algebraic Representation

- All of these curves are just parametric algebraic polynomials expressed in different bases
- Parametric linear curve (in $\mathcal{R}^3$)
  $P(u) = \mathbf{a}u + \mathbf{b}$
  $x = a_x u + b_x$
  $y = a_y u + b_y$
  $z = a_z u + b_z$
- Parametric cubic curve (in $\mathcal{R}^3$)
  $P(u) = \mathbf{a}u^3 + \mathbf{b}u^2 + \mathbf{c}u + \mathbf{d}$
  $x = a_x u^3 + b_x u^2 + c_x u + d_x$
  $y = a_y u^3 + b_y u^2 + c_y u + d_y$
  $z = a_z u^3 + b_z u^2 + c_z u + d_z$
- Basis (monomial or power)
  $\begin{bmatrix} u & 1 \end{bmatrix}$
  $\begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix}$

Hermite Curves

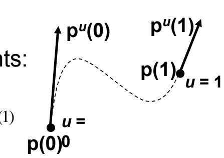
- 12 degrees of freedom (4 3-d vector constraints)
- Specify endpoints and tangent vectors at endpoints

$$P(0) = \mathbf{d}$$
$$P(1) = \mathbf{a} + \mathbf{b} + \mathbf{c} + \mathbf{d}$$
$$P^u(0) = \mathbf{c}$$
$$P^u(1) = 3\mathbf{a} + 2\mathbf{b} + \mathbf{c}$$

$$\mathbf{p}^u(u) \equiv \frac{dP}{du}(u)$$

- Solving for the coefficients:

$$\mathbf{a} = 2\mathbf{p}(0) - 2\mathbf{p}(1) + \mathbf{p}^u(0) + \mathbf{p}^u(1)$$
$$\mathbf{b} = -3\mathbf{p}(0) + 3\mathbf{p}(1) - 2\mathbf{p}^u(0) - \mathbf{p}^u(1)$$
$$\mathbf{c} = \mathbf{p}^u(0)$$
$$\mathbf{d} = \mathbf{p}(0)$$



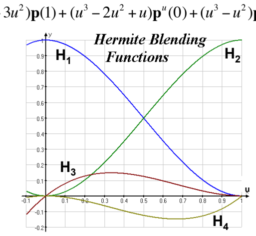
Hermite Basis

- Substituting for the coefficients and collecting terms gives

$$P(u) = (2u^3 - 3u^2 + 1)\mathbf{p}(0) + (-2u^3 + 3u^2)\mathbf{p}(1) + (u^3 - 2u^2 + u)\mathbf{p}^u(0) + (u^3 - u^2)\mathbf{p}^u(1)$$

- Call

$$\mathrm{H}_1(u) = (2u^3 - 3u^2 + 1)$$
$$\mathrm{H}_2(u) = (-2u^3 + 3u^2)$$
$$\mathrm{H}_3(u) = (u^3 - 2u^2 + u)$$
$$\mathrm{H}_4(u) = (u^3 - u^2)$$

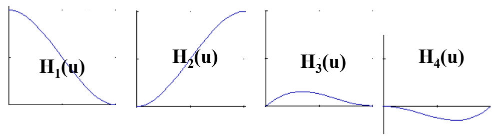


the Hermite **blending functions** or **basis functions**

- Then $P(u) = \mathrm{H}_1(u)\mathbf{p}(0) + \mathrm{H}_2(u)\mathbf{p}(1) + \mathrm{H}_3(u)\mathbf{p}^u(0) + \mathrm{H}_4(u)\mathbf{p}^u(1)$

Blending Functions

$$P(u) = (2u^3 - 3u^2 + 1)\mathbf{p}(0) + (-2u^3 + 3u^2)\mathbf{p}(1) + (u^3 - 2u^2 + u)\mathbf{p}^u(0) + (u^3 - u^2)\mathbf{p}^u(1)$$



- At u = 0:
  - $H_1 = 1$, $H_2 = H_3 = H_4 = 0$
  - $H_1' = H_2' = H_4' = 0$, $H_3' = 1$
- At u = 1:
  - $H_1 = H_3 = H_4 = 0$, $H_2 = 1$
  - $H_1' = H_2' = H_3' = 0$, $H_4' = 1$

P(0) = p0
P'(0) = T0

P(1) = p1
P'(1) = T1

Hermite Curves - Matrix Form

- Putting this in matrix form $\mathbf{H} = \begin{bmatrix} \mathrm{H}_1(u) & \mathrm{H}_2(u) & \mathrm{H}_3(u) & \mathrm{H}_4(u) \end{bmatrix}$

$$= \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

$$= \mathbf{UM}_{\mathrm{H}}$$

- $\mathbf{M}_{\mathrm{H}}$ is called the Hermite **characteristic matrix**
- Collecting the Hermite geometric coefficients into a geometry vector **B**, we have a matrix formulation for the Hermite curve **P**(*u*)

$$\mathbf{B} = \begin{bmatrix} \mathbf{p}(0) \\ \mathbf{p}(1) \\ \mathbf{p}^u(0) \\ \mathbf{p}^u(1) \end{bmatrix}$$

$$P(u) = \mathbf{UM}_{\mathrm{H}}\mathbf{B}$$

Hermite and Algebraic Forms

- $\mathbf{M}_{\mathrm{H}}$ transforms geometric coefficients ("coordinates") from the Hermite basis to the algebraic coefficients of the monomial basis

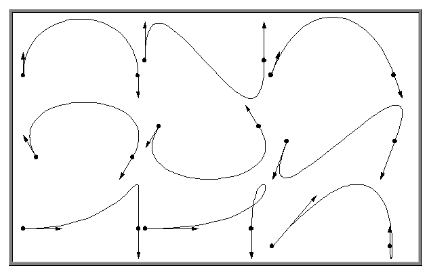
$$\mathbf{A} = \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \\ \mathbf{c} \\ \mathbf{d} \end{bmatrix}$$

$$P(u) = \mathbf{UA} = \mathbf{UM}_{\mathrm{H}}\mathbf{B}$$
$$\mathbf{A} = \mathbf{M}_{\mathrm{H}}\mathbf{B}$$
$$\mathbf{B} = \mathbf{M}_{\mathrm{H}}^{-1}\mathbf{A}$$

$$\mathbf{M}_{\mathrm{H}}^{-1} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix}$$

Hermite Curves

- Geometrically defined by position and tangents at end points



Issues with Bézier Curves

- Creating complex curves may (with lots of wiggles) requires many control points
  – potentially a very high-degree polynomial
- Bézier blending functions have *global support* over the whole curve
  – move just one point, change whole curve
- *Improved Idea: link (C^1) lots of low degree (cubic) Bézier curves end-to-end*

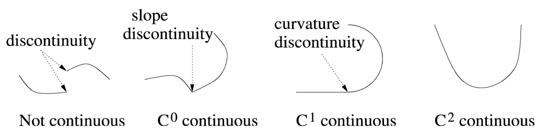
Continuity

Two types:

- Geometric Continuity, G^i:
  – endpoints meet
  – tangent vectors' directions are equal
- Parametric Continuity, C^i:
  – endpoints meet
  – tangent vectors' directions are equal
  – tangent vectors' magnitudes are equal
- In general: C implies G but not vice versa

Parametric Continuity

- ***Continuity*** (recall from the calculus):
  - Two curves are C^i continuous at a point p iff the i-th derivatives of the curves are equal at p



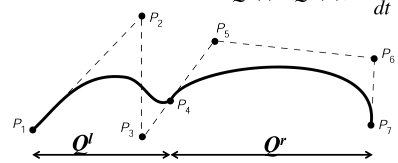
13

Pics/Math courtesy of Dave Mount @ UMD-CP

Continuity

- What are the conditions for C^0 and C^1 continuity at the joint of curves x^l and x^r?
  - tangent vectors at end points equal
  - end points equal

$$Q^l(1) = Q^r(0), \quad \frac{dQ^l}{dt}(1) = \frac{dQ^r}{dt}(0)$$



14

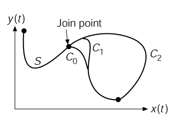
1994 Foley/VanDam/Finer/Huges/Phillips ICG

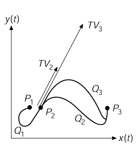
Continuity

- The derivative of $Q(t)$ is the parametric tangent vector of the curve:

$$\frac{d}{dt}Q(t) = Q'(t) = \left[\ \tfrac{d}{dt}x(t) \quad \tfrac{d}{dt}y(t) \quad \tfrac{d}{dt}z(t)\ \right]^T = \frac{d}{dt}C \cdot T = C \cdot \left[\ 3t^2 \quad 2t \quad 1 \quad 0\ \right]^T =$$

$$\left[\ 3a_xt^2 + 2b_xt + c_x \quad 3a_yt^2 + 2b_yt + c_y \quad 3a_zt^2 + 2b_zt + c_z\ \right]^T$$





15

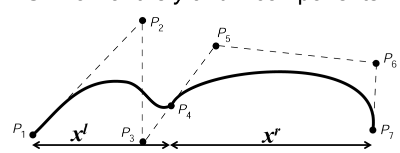
1994 Foley/VanDam/Finer/Huges/Phillips ICG

Continuity

- In 3D, compute this for each component of the parametric function
  - For the x component:

$$x^l(1) = x^r(0) = P_{4_x}, \ \frac{d}{dt}x^l(1) = 3(P_{4_x} - P_{3_x}), \ \frac{d}{dt}x^r(0) = 3(P_{5_x} - P_{4_x})$$

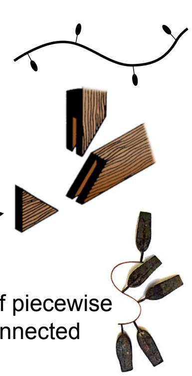
- Similar for the *y* and *z* components.



16

1994 Foley/VanDam/Finer/Huges/Phillips ICG

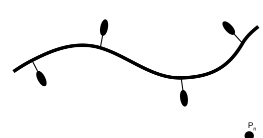
Splines

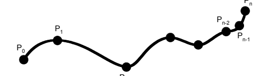


- Popularized in late 1960s in US Auto industry (GM)
  - R. Riesenfeld (1972)
  - W. Gordon
- Origin: the thin wood or metal strips used in building/ship construction
- Goal: define a curve as a set of piecewise simple polynomial functions connected together

Natural Splines

- Mathematical representation of physical splines
- C2 continuous
- Interpolate all control points
- Have Global control (no local control)





18

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!)

19

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!

20

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***)

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

  - B_i is zero for most values of t!

21

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:

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

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

22

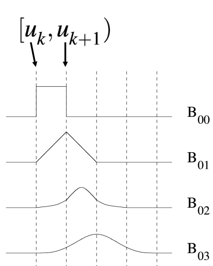
B-spline Blending Functions

$B_{k,0}(t)$ is a step function that is 1 in the interval

$B_{k,1}(t)$ spans two intervals and is a piecewise linear function that goes from 0 to 1 (and back)

$B_{k,2}(t)$ 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

$B_{k,3}(t)$ is a cubic that spans four intervals growing from 0 to 1/6 to 2/3, then back to 1/6 and to 0



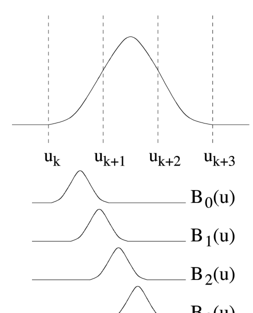
B-spline blending functions

23

Pics/Math courtesy of Dave Mount @ UMD-CP

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

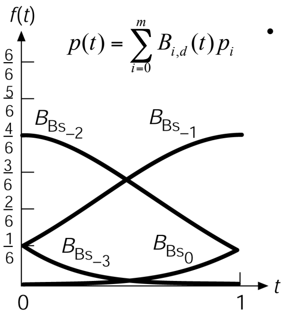
- Note: can't define a polynomial with these properties (both 0 and non-zero for ranges)
- Idea: subdivide the parameter space into *intervals* and build a *piecewise polynomial*
  - Each interval gets different polynomial function



Pics/Math courtesy of Dave Mount @ UMD-CP

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

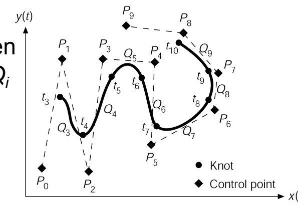
$$p(t) = \sum_{i=0}^{m} B_{i,d}(t)\, p_i$$



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

B-splines: Knot Selection

- Instead of working with the parameter space $0 \le t \le 1$, use $t_{\min} \le t_0 \le t_1 \le t_2 \ldots \le t_{m-1} \le t_{\max}$
- The ***knot points***
  - joint points between curve segments, Q_i
  - Each has a *knot value*
  - *m-1* knots for *m+1* points



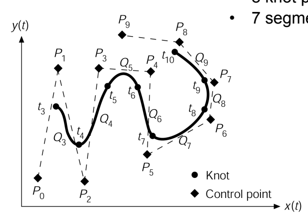
Uniform B-splines: Setting the Options

- Specified by
  - $m \ge 3$
  - *m+1* ***control points***, $P_0 \ldots P_m$
  - *m-2* ***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 are
    - defined over a knot interval $[t_i, t_{i+1}]$
    - 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

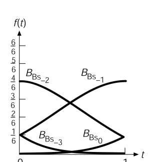
$$p(t) = \sum_{i=0}^{m} B_{i,d}(t)\, p_i$$

- m = 9
- 10 control points
- 8 knot points
- 7 segments



B-spline: 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

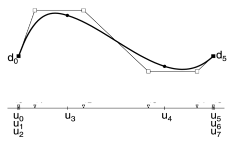
- Cox-deBoor Algorithm: defines the blending functions for spline curves (not limited to deg 3)
  - curves are weighted avgs of lower degree curves
- Let $B_{i,d}(t)$ denote the *i*-th blending function for a B-spline of degree *d,* then:

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

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

Creating a Non-Uniform B-spline: Knot Selection

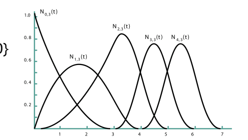
- Given curve of degree $d=3$, with $m+1$ control points $\mathbf{p}_0, \cdots, \mathbf{p}_m$
  - first, create $m+d$ knot values
  - use knot values *(0,0,0,1,2,…, m-2, m-1,m-1,m-1)* (adding two extra *0*' s and *m-1*' s)
  - Note
    - Causes Cox-deBoor to give added weight in blending to the first and last points when t is near t_{min} and t_{max}

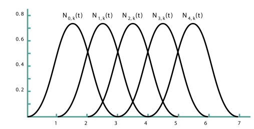


Pics/Math courtesy of G. Farin @ ASU

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!



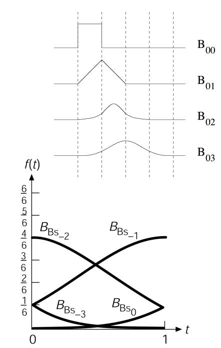


33

From http://devworld.apple.com/dev/techsupport/develop/issue25/schneider.html

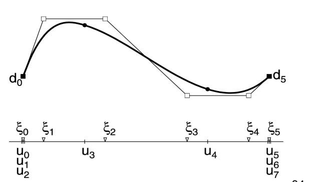
$$p(t) = \sum_{i=0}^{m} B_{i,d}(t) p_i$$

B-spline Summary



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

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

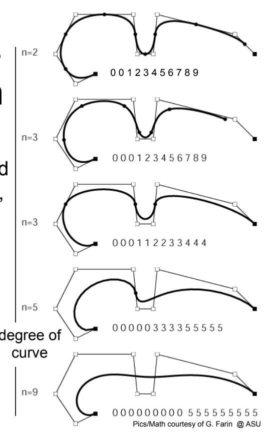


34

Watching Effects of Knot Selection

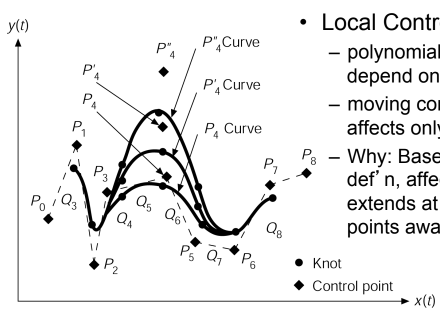
- 9 knot points (initially)
  - Note: knots are distributed parametrically based on t, hence why they "move"
- 10 control points
- Curves have as many segments as they have non-zero intervals in u

degree of curve



Pics/Math courtesy of G. Farin @ ASU

B-splines: Local Control Property

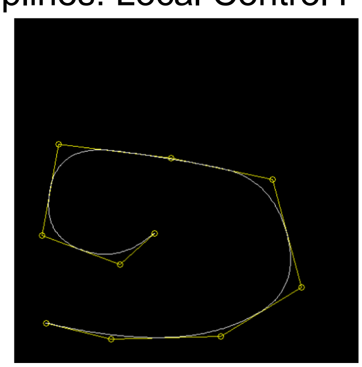


- Local Control
  - polynomial coefficients depend on a few points
  - moving control point (P_4) affects only local curve
  - Why: Based on curve def' n, affected region extends at most 2 knot points away

36

1994 Foley/VanDam/Finer/Huges/Phillips ICG

B-splines: Local Control Property

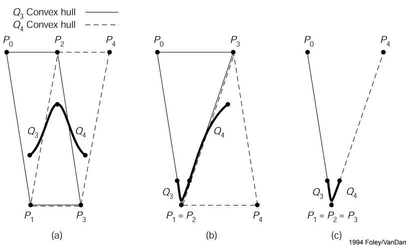


37

Recorded from: http://heim.ifi.uio.no/~trondbre/OsloAlgApp.html

B-splines: Convex Hull Property

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



39

1994 Foley/VanDam/Finer/Huges/Phillips ICG

B-splines: Continuity

- Derivatives are easy for cubics

$$p(u) = \sum_{k=0}^{3} u^k c_k$$

- Derivative:

$$p'(u) = c_1 + 2c_2u + 3c_3u^2$$

Easy to show C^0, C^1, C^2

40

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**

41

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

U of Utah, Alpha_1

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

43

NURBS

Non-**u**niform **R**ational **B-spline**s: ***NURBS***

- Basic idea: four dimensional non-uniform B-splines, followed by normalization via homogeneous coordinates
  - If P_i is *[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 Bézier curve

44

Rational Functions

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

45

Rational Functions: Example

- **Example:**
  – rational function: a *ratio* of polynomials
  – a rational parameterization in *u* of a unit circle in xy-plane:
  $$\begin{aligned} x(u) &= \frac{1-u^2}{1+u^2} \\ y(u) &= \frac{2u}{1+u^2} \\ z(u) &= 0 \end{aligned}$$
  – a unit circle in 3D homogeneous coordinates:
  $$\begin{aligned} x(u) &= 1-u^2 \\ y(u) &= 2u \\ z(u) &= 0 \\ w(u) &= 1+u^2 \end{aligned}$$

46

NURBS: Notation Alert

- Depending on the source/reference
  – Blending functions are either $B_{i,d}(u)$ or $N_{i,d}(u)$
  – 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, ….*

47

NURBS: Notation Alert

1. If defined using *homogenous coordinates*, the 4th (3rd for 2D) dimension of each P_i is the weight
2. If defined as *weighted euclidian*, a separate constant w_i, is defined for each control point

48

NURBS

- A *d*-th degree NURBS curve *C* is def' d as:
  $$C(u)=\frac{\sum_{i=0}^{n-1}\boxed{w_i B_{i,d}(u)}\boxed{P_i}}{\boxed{\sum_{i=0}^{n-1} w_i B_{i,d}(u)}}$$
  Where
  – control points, P_i
  – *d*-th degree B-spline blending functions, $B_{i,d}(u)$
  – the *weight, w_i,* for control point P_i (when all w_i=*1*, we have a B-spline curve)

49

Observe: Weights Induce New Rational Basis Functions, *R*

- Setting: $R_i(u)=\frac{w_i B_{i,d}(u)}{\sum_{i=0}^{n-1} w_i B_{i,d}(u)}$

  Allows us to write: $C(u)=\sum_{i=0}^{n-1} R_{i,d}(u) P_i$

  Where $R_{i,d}(u)$ are *rational basis functions*
  – piecewise rational basis functions on $u\in[0,1]$
  – weights are incorporated into the basis fctns

50

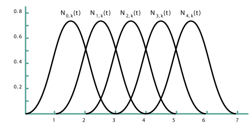
Geometric Interpretation of NURBS

- 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*)

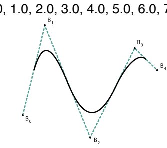
51

NURBS: Examples

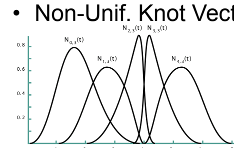
- Unif. Knot Vector



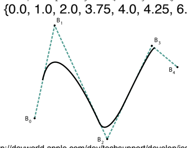
{0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0}



- Non-Unif. Knot Vector



{0.0, 1.0, 2.0, 3.75, 4.0, 4.25, 6.0, 7.0}

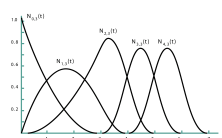


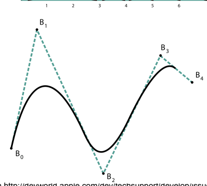
52

From http://devworld.apple.com/dev/techsupport/develop/issue25/schneider.html

NURBS: Examples

- 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
- Bunches up the curve and forces it to interpolate

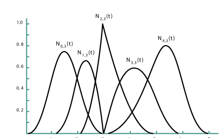


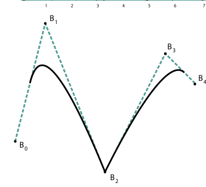


53

From http://devworld.apple.com/dev/techsupport/develop/issue25/schneider.html

NURBS: Examples





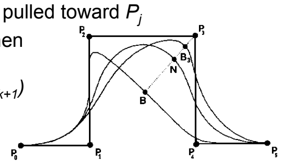
- Knot Vector
  {0.0, 1.0, 2.0, **3.0, 3.0**, 5.0, 6.0, 7.0}
- Several consecutive knots get the same value
- Bunches up the curve and forces it to interpolate
- Can be done midcurve

54

From http://devworld.apple.com/dev/techsupport/develop/issue25/schneider.html

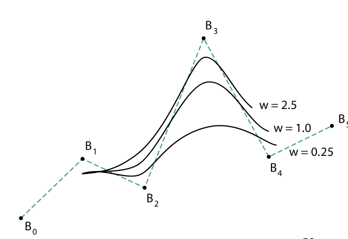
The Effects of the Weights

- w_i of P_i effects only the range $[u_i, u_{i+k+1})$
- 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 $B_i \rightarrow P_i$, if u in $[u_i, u_{i+k+1})$



The Effects of the Weights

- Increased weight pulls the curve toward B_3



56

From http://devworld.apple.com/dev/techsupport/develop/issue25/schneider.html

Programming Assignment 1

- Process command-line arguments
- Read in 3D control points
- Iterate through parameter space by du
- At each u value evaluate Bezier curve formula to produce a sequence of 3D points
- Output points by printing them to the console as a polyline and control points as spheres in Open Inventor format

57