CptS 442/542 (Computer Graphics) Unit 10: Curves and Surfaces

Bob Lewis

School of Engineering and Applied Sciences
Washington State University

Fall, 2017

Motivation

- Pixel representations limit resolution and have no meaning in 3D.
- Line segment and polygonal representations (like OpenGL uses) limit accuracy.
- Real objects are not usually multifacted polyhedra. Manufactured objects in particular need to look *smooth*.
- Fonts need to be resolution-independent (to first order).
- Objects often move along curves.
- Camera paths *must* be smooth curves.
- Looking beyond OpenGL, some rendering techniques (mainly ray tracing) can create an image directly from a smooth shape without tesselation.
  - Think: No polygonal silhouettes!

Ground Rules

- We'll talk mainly about curves.
- Surfaces are (surprisingly) easy, once we understand curves.
- Our curves will be 2D (easier to display), but 3D curves are (we'll see) trivial once we understand 2D curves.

Geometric Observations

- Curves have one degree of freedom (i.e. a parameter), regardless of how many dimensions they're embedded in.
- Surfaces have two degrees of freedom.
- Volumes (which can also be modelled geometrically) have three degrees of freedom.

How Do We Model a Smooth Shape?

- We need to distinguish modelling from rendering:
  - Modelling produces a representation.
  - Rendering produces an image.
- Even if we end up *rendering* a shape with lines and triangles, it would useful to *model* the shape with a smooth representation beforehand.
- General procedure (cf. *coaster*):
  - Manipulate the shape model.
  - Tesselate it.
  - Render the tesselation.

Possible (2D) Representations

- explicit functions: $y = f(x)$
  problems:
  - no multiple values
  - not axis independent
  - vertical tangents difficult to represent
- implicit functions: $f(x, y) = 0$
  problems:
  - multiple values may require additional constraints
  - hard to draw in general
- parametric functions: $x = x(t), y = y(t)$
  problems:
  - may not be intuitive

Polynomials

- Simple-to-evaluate (even in hardware), but are they useful?
- Choices:
  - Linear: $\tilde{\mathbf{P}}(t) = \tilde{\mathbf{a}}t + \tilde{\mathbf{b}}$
    - What do you get with this form?
    - $\tilde{\mathbf{a}}$ and $\tilde{\mathbf{b}}$ are pretty easy to figure out. (You've done it before.)
  - Quadratic: $\tilde{\mathbf{P}}(t) = \tilde{\mathbf{a}}t^2 + \tilde{\mathbf{b}}t + \tilde{\mathbf{c}}$
    - but what are $\tilde{\mathbf{a}}$, $\tilde{\mathbf{b}}$, and $\tilde{\mathbf{c}}$? $\tilde{\mathbf{c}}$ is easy, but what are $\tilde{\mathbf{a}}$ and $\tilde{\mathbf{b}}$? They *look* like points.
  - Cubic: $\tilde{\mathbf{P}}(t) = \tilde{\mathbf{a}}t^3 + \tilde{\mathbf{b}}t^2 + \tilde{\mathbf{c}}t + \tilde{\mathbf{d}}$
    - Again, what are $\tilde{\mathbf{a}}$, $\tilde{\mathbf{b}}$, and $\tilde{\mathbf{c}}$?
    - Can we generate useful curves with this?
    - Eight values (coefficients) to get a smooth (2D) curve is not bad, but are those values intuitive?
    - How many values would this take in 3D?
- What would be the drawbacks of using a higher-degree expression?

Aside: Can We Do Conics With Parametric Polynomials?

- It would be useful to be able to represent conics (circles, ellipses, parabolas, etc.)
- We can represent them, but not with the polynomial forms we've used so far.
- We need to use *rational* curves, (Bézier or B-spline), which we'll talk about later.

Interactive Curve Design

- control points are useful
  - fewer controls
  - guaranteed smoothness
- can evaluate $\tilde{\mathbf{P}}(t)$ for any t
- independent of scale: no pixels involved
- major application? (You're looking at one.)
- interpolation vs. approximation

Interactive Spline Demo

http://geometrie.foretnik.net/files/NURBS-en.swf

Notes:

- Your browser needs to support Shockwave Flash. (My Firefox does.)
- They constrain the parameter t to go from 0 to 1 for the whole spline curve, which is non-standard.

Bézier Curves: The deCasteljau Construction

Given a set of points $\tilde{\mathbf{P}}_0 \ldots \tilde{\mathbf{P}}_n$, let

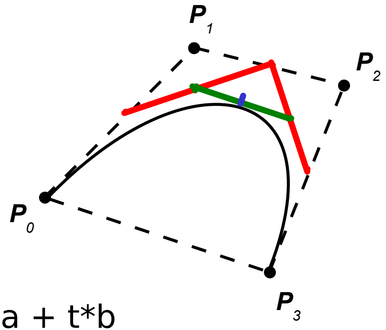
$$\tilde{\mathbf{P}}_i^0(t) \equiv \tilde{\mathbf{P}}_i$$

Then recursively let

$$\begin{aligned}\tilde{\mathbf{P}}_i^r(t) &= (1-t)\tilde{\mathbf{P}}_i^{r-1}(t) \\ &\quad + t\tilde{\mathbf{P}}_{i+1}^{r-1}(t) \\ &= \mathrm{lerp}\left(\tilde{\mathbf{P}}_i^{r-1}, t, \tilde{\mathbf{P}}_{i+1}^{r-1},\right)\end{aligned}$$

for $r \in [1 \ldots n]$ and $i \in [0 \ldots n-r]$. $\tilde{\mathbf{P}}_0^n(t)$ is the point on the Bézier curve at t.

There's an easy way to see this graphically...



lerp(a, t, b) = (1-t)*a + t*b

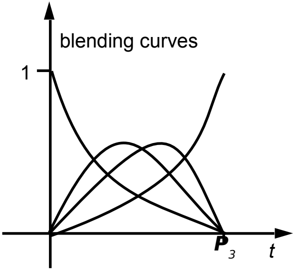
Cubic Bézier Blending Functions

Expand the construction into a single expression of the form:

$$\tilde{\mathbf{P}}(t) = \sum_{i=0}^{3} \tilde{\mathbf{P}}_i B_i(t)$$

Where the *blending curves* $B_i(t)$ are the *Bernstein polynomials*:

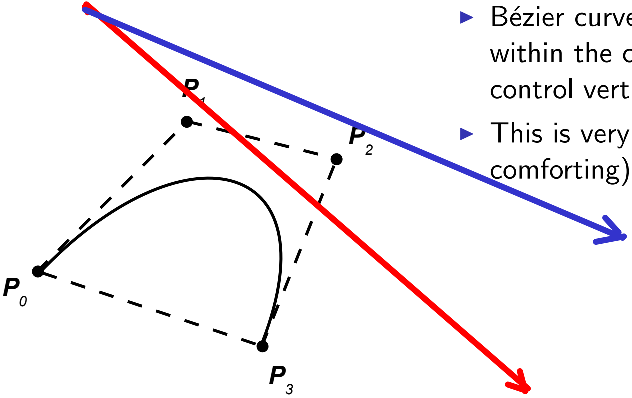
$$\begin{aligned} B_0(t) &= (1-t)^3 \\ B_1(t) &= 3t\,(1-t)^2 \\ B_2(t) &= 3t^2\,(1-t) \\ B_3(t) &= t^3 \end{aligned}$$



Definition: Convex Hulls

- A convex hull of any collection of points is the smallest (polygon, polyhedron, etc.) that contains all of them.
- convex hulls intuitively defined:
  - in 2D: wrap rubber band around “nails” at points
  - in 3D: shrink wrap around points

Bézier Curves: The Convex Hull Property



- Bézier curve is contained within the convex hull of the control vertices that define it
- This is very useful (and comforting) to designers.

Other Bézier Curve Properties

- interpolates (passes thru) endpoints
- affine invariance:
  affine transform of curve is curve defined by affine transform of control points
- linear precision:
  if control points line up, so does curve
- derivative (i.e., tangent) easy to compute
  $\frac{d\tilde{\mathbf{P}}(t)}{dt}$ is polynomial one degree less than that of curve
- Cubics are used most often, but lower and higher degree curves are possible (again, using deCasteljau).
  - You pay a price for higher degrees.

Better Than Bézier Blending Functions

Bézier curves are okay, but we can do better. We want to compute a weighted sum of control points

$$\tilde{\mathbf{P}}(t) = \sum_{i=0}^{3} \tilde{\mathbf{P}}_i g_i(t)$$

with blending functions $g_i()$ that

- provide local control i.e., have local support
- allow for more than four control points, but are still numerically stable and easy to compute
- sum to unity for all t
- allow control of interpolation vs. approximation
- allow control of smoothness

Editing a Curve

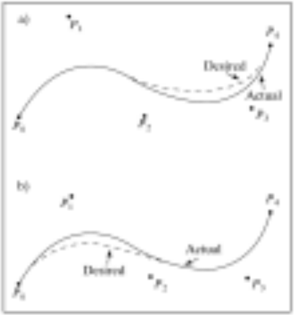
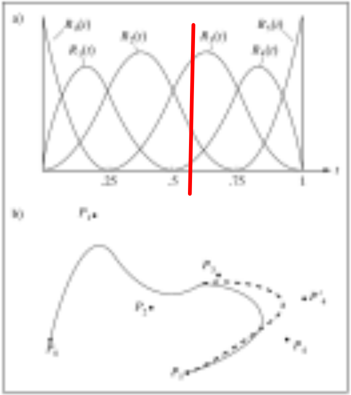


FIGURE 11.19 Editing portions of a curve.

Blending Functions

FIGURE 11.28 Blending functions having concentrated support.



Piecewise Polynomials

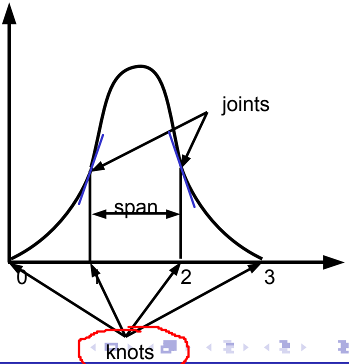
A single polynomial can't do all of these, but let's try a *piecewise polynomial*:

Procedure:

- subdivide [0,M] into M regions
- create one polynomial of degree M-1 for each region
- require zero values and derivatives at ends (small support)

(other constraints are possible)

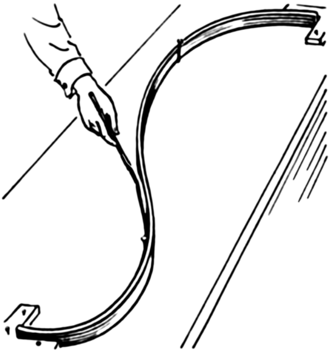
Terminology ($M = 3$):



The Origin of Splines

Originally used in shipbuilding and drafting in general (see image), these were long strips of metal held in place by "ducks" – lead weights.

They were smooth, and the ducks were like (interpolated) control points, but these "natural splines" have two undesirable properties for our purposes: non-locality and a required (possibly big) matrix inversion every time a position changes.



(from Wikipedia)

(Mathematical) Splines

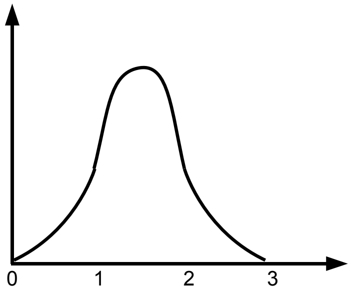
Definition:

- An $\cancel{M}$ D th degree spline is a piecewise polynomial of degree $\cancel{M}$ that is $\cancel{M}-1$ smooth at D each knot. D

Splines are used as blending functions. A *lot*.

A quadratic (degree $\cancel{M}$ D $= 2$) spline ($\cancel{M}=1$ is trivial):

D

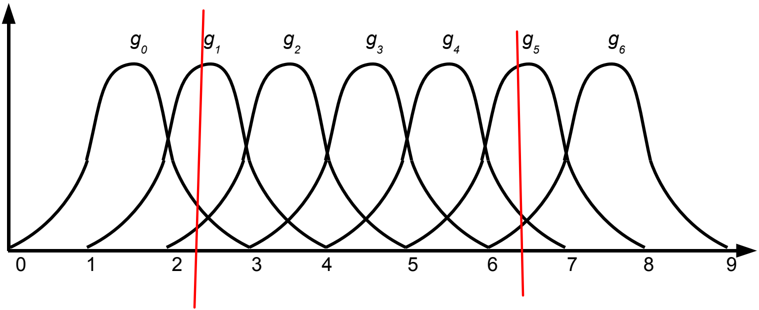


Using Splines as Blending Functions

Construct shifted versions of $g(t)$, $g_k(t)$:

$$g_k(t) \equiv g(t-k)$$

These look like:



Splines as Blending Functions

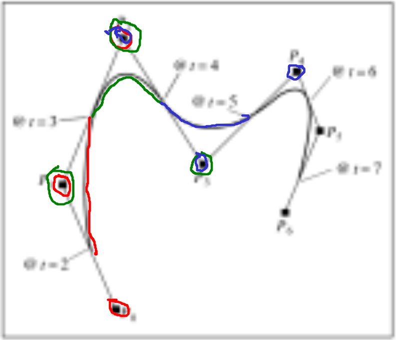
For $2 \le t \le 7$, the curve is

$$\tilde{\mathbf{P}}(t) = \sum_{k=0}^{6} \tilde{\mathbf{P}}_k g_k(t)$$

- What if the P_k's are all identical?
- Is the curve smooth enough?
- Is $\tilde{\mathbf{P}}(t)$ continuous at knots $t = k$? $\tilde{\mathbf{P}}'(t = k)$? $\tilde{\mathbf{P}}''(t = k)$?

Example of Curve Design

FIGURE 11.25 Curve design using translates of $g(.)$.



More Generalized Spline Curves I

We can increase flexibility by allowing for a knot vector $\tilde{\mathbf{T}} = t_0, t_1, t_2, \ldots$ where the t_k's are a non-decreasing sequence. We take

$$\tilde{\mathbf{P}}(t) = \sum_{k=0}^{L} \tilde{\mathbf{P}}_k R_k(t)$$

- $\tilde{\mathbf{P}}()$ is a weighted sum of $L+1$ control points.
- $R_k()$'s are blending functions.
- $t_i's$ can be any non-decreasing sequence, but the only thing that really matters (this can be proven) is whether $t_i = t_{i+1}$ or not, so we usually choose integer $t_i's$, starting at 0 and t_{i+1} is either t_i or $t_i + 1$. (The `geometrie.foretnik.net` spline demo doesn't do this.)

Blending Functions for Generalized Knot Vector

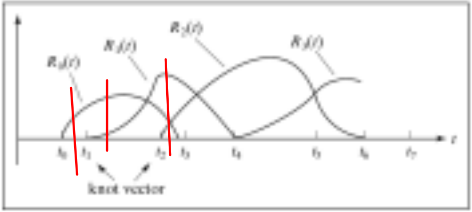


FIGURE 11.27 Generalizing on the knot vector and blending functions.

More Generalized Spline Curves II

Reinterpreting what we've done:

- Bézier: one span, $L+1$ knots at each end.
- translates $g(t)$: knots at 0, 1, 2, ... , $L+1$, so each translate has 3 quadratic polynomials, each with a span of 3.

The Big Question

Given a knot vector $\tilde{\mathbf{T}}$, can we construct blending functions to represent every possible spline on that knot vector?

- Such functions form the *basis* for the spline.
- If we further look to minimize the support of each blending function, we arrive at "B-splines".
  - (The "B" stands for "basis", *not* Bézier.)

B-Spline Basis Functions

Given:

- L control points,
- an order m ($=$ degree $+$ 1), and
- a knot vector $\tilde{\mathbf{T}} = \{t_0, t_1, t_2, ..., t_{L+m-1}\}$,

We define the order 1 basis as

$$N_{k,1}(t) \equiv \begin{cases} 1 & \text{if } t_k < t \leq t_{k+1} \\ 0 & \text{otherwise} \end{cases}$$

and recursively define the rest (the Cox-DeBoor relation)

$$N_{k,m}(t) = \left(\frac{t - t_k}{t_{k+m-1} - t_k}\right) N_{k,m-1}(t) + \left(\frac{t_{k+m} - t}{t_{k+m} - t_{k+1}}\right) N_{k+1,m-1}(t)$$

for $k = 0, 1, ..., L$, with the caveat that if a denominator is 0, we ignore the term.

Linear B-Spline Bases

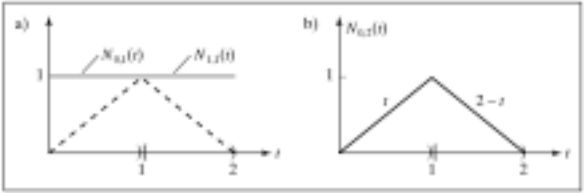


FIGURE 11.28 Construction of linear B-splines.

Quadratic B-Spline Bases

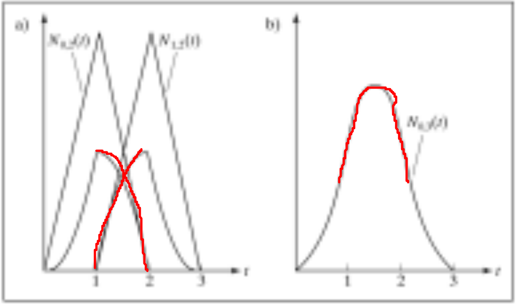


FIGURE 11.29 The first quadratic B-spline shape.

Cubic B-Spline Bases

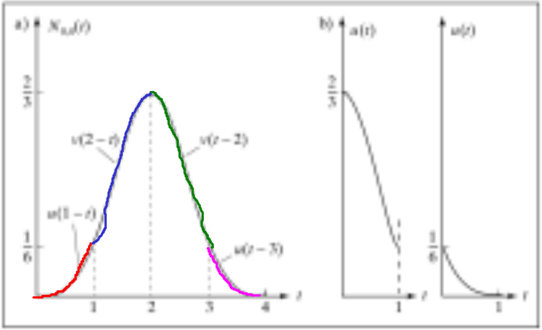


FIGURE 11.30 The cubic B-spline on equispaced knots.

Any Degree B-Spline Construction

Note the use of recursion.

```
float bSpline(int k, int m, float t, float knot[])
{
      float denom1, denom2, sum = 0.0;
      if(m == 1)
         return (t >= knot[k] && t < knot[k+1]); // 1 or 0
      // m exceeds 1.. use recursion
      denom1 = knot[k + m -1] - knot[k];
      if(denom1 != 0.0)
         sum = (t - knot[k]) * bSpline(k,m-1,t, knot) / denom1;
      denom2 = knot[k + m] - knot[k+1];
      if(denom2 != 0.0)
         sum += (knot[k+m] - t) * bSpline(k+1,m-1,t,knot) / denom2;
      return sum;
}
```

FIGURE 11.31 Computing B-spline blending functions.

Weighted B-Splines

To give some control vertices more influence than others:

$$\tilde{\mathbf{P}}(t) = \sum_{k=0}^{L} h_k \tilde{\mathbf{P}}_k R_k(t)$$

Reprise: Can We Do Conics With Parametric Polynomials?

We discussed rendering conics (circles, ellipses, parabolas, etc.) and mentioned that we need to use *rational quadratic polynomial*s. They look like this:

$$\tilde{\mathbf{P}}(t) = \frac{(1-t)^2\tilde{\mathbf{P}}_0 + 2wt(1-t)\tilde{\mathbf{P}}_1 + t^2\tilde{\mathbf{P}}_2}{(1-t)^2 + 2wt(1-t) + t^2}$$

- This can generate all conics. If the "weight" w is...
  - < 1, the curve is an ellipse,
  - $= 1$, the curve is a parabola, and
  - > 1, the curve is a hyperbola.
- This is actually a *rational B-spline*.

Turning Curves Into Surfaces I

Intuition (from Gerald Farin's book *Curves and Surfaces for CAGD*):

> *A surface is the locus [set of points] of a curve that is moving through space and thereby changing its shape.*

Consider a parametric 3D cubic curve:

$$\tilde{\mathbf{P}}(u) = \sum_{k=0}^{L} \tilde{\mathbf{P}}_k R_k(u)$$

Turning Curves Into Surfaces II

Surfaces need two degrees of freedom, so let $\tilde{\mathbf{P}}(u, v)$ map the unit square $[0, 1] \times [0, 1]$ to a parametric 3D surface. Get this by (!) replacing the control points $\tilde{\mathbf{P}}_j$ with parametric functions $\tilde{\mathbf{\Gamma}}_j(v)$:

$$\tilde{\mathbf{P}}(u, v) = \sum_{k=0}^{L} \tilde{\mathbf{\Gamma}}_k(u) R_k(v)$$

For a fixed u, this is just a curve in space w.r.t. v. Now, however, the $\tilde{\mathbf{\Gamma}}_j$'s are space curves w.r.t. u. How to define *them*?

Turning Curves Into Surfaces III

Well, $\tilde{\mathbf{\Gamma}}_j(u)$ is just another space curve, which we already know how to do. Letting each $\tilde{\mathbf{\Gamma}}_j(u)$ have its own set of control points $\tilde{\mathbf{P}}_{ij}$:

$$\tilde{\mathbf{\Gamma}}_j(u) = \sum_{i=0}^{L} \tilde{\mathbf{P}}_{ij} R_i(u)$$

So combining this with what we've done previously...

$$\tilde{\mathbf{P}}(u, v) = \sum_{i=0}^{L} \sum_{j=0}^{L} \tilde{\mathbf{P}}_{ij} R_i(u) R_j(v)$$

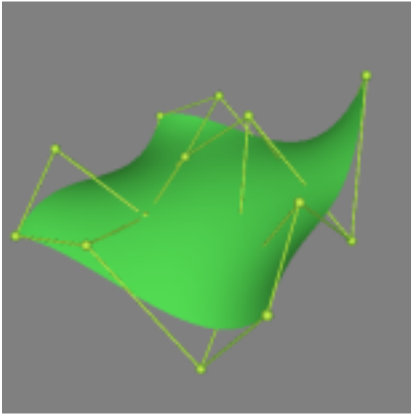
This is a *parametric bicubic* (if $L = 3$) *surface*. Note the generality (any basis) and extensibility (higher degrees and dimensions).

Cubic Bézier Surfaces

We specify a mesh of 16 control points:

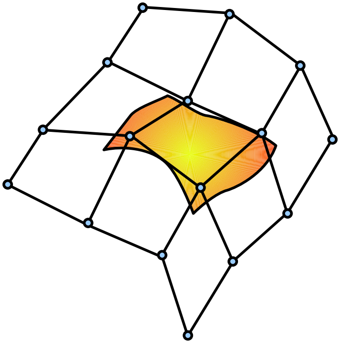
$$\begin{bmatrix} \tilde{\mathbf{P}}_{i-1,j-1} & \tilde{\mathbf{P}}_{i-1,j} & \tilde{\mathbf{P}}_{i-1,j+1} & \tilde{\mathbf{P}}_{i-1,j+2} \\ \tilde{\mathbf{P}}_{i,j-1} & \tilde{\mathbf{P}}_{i,j} & \tilde{\mathbf{P}}_{i,j+1} & \tilde{\mathbf{P}}_{i,j+2} \\ \tilde{\mathbf{P}}_{i+1,j-1} & \tilde{\mathbf{P}}_{i+1,j} & \tilde{\mathbf{P}}_{i+1,j+1} & \tilde{\mathbf{P}}_{i+1,j+2} \\ \tilde{\mathbf{P}}_{i+2,j-1} & \tilde{\mathbf{P}}_{i+2,j} & \tilde{\mathbf{P}}_{i+2,j+1} & \tilde{\mathbf{P}}_{i+2,j+2} \end{bmatrix}$$

The convex hull of these points is the convex hull of the surface.



Cubic B-Spline Surfaces

- As for a Bézier surface, we specify a mesh of 16 control points whose convex hull is a convex hull of the surface.
- This works with all flavors of B-splines.
- This could be a sub-array of a much larger array (e.g. a quad mesh).
- Continuity is built-in (as with curves).



Surface Normals of Parametric Surfaces

If we have

$$\tilde{\mathbf{P}}(s,t) = \sum_{i=0}^{L}\sum_{j=0}^{L}\tilde{\mathbf{P}}_{ij}R_i(s)R_j(t)$$

we'd like to compute vertex normals so we can light them properly. How do we compute surface normals? Hint: Think “tangents”.

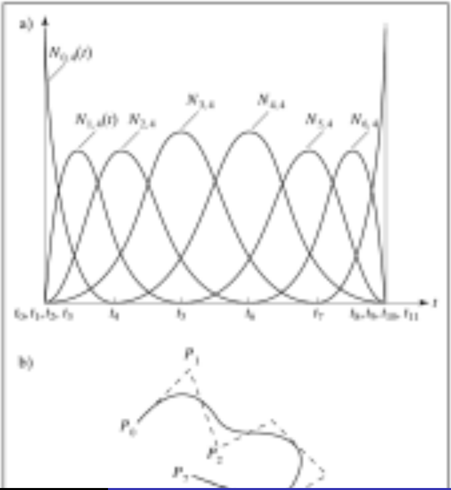
The Standard Knot Vector

Let $\tilde{\mathbf{T}} = \{m * \{0\}, 1, 2, ..., m, m * \{m+1\}\}$ knots for an order m (degree $m-1$) spline.

- What is this for cubic (m $= 4$) B-splines? (Note: $m \leq L + 1$)
- How many control points are there?

Cubic Spline Blending Functions

FIGURE 11.34 Eight cubic B-spline blending functions.



Rational Splines and NURBSes

Rational B-Splines are defined as before:

$$\tilde{\mathbf{P}}(t) = \sum_{k=0}^{L} \tilde{\mathbf{P}}_k R_k(t)$$

but the blending functions $R_k(t)$ are:

$$R_k(t) \equiv \frac{w_k N_{k,m}(t)}{\sum_{j=0}^{L} w_j N_{j,m}(t)}$$

where the w_j's are "weights" or "shape parameters" chosen so that the denominator never vanishes. (The denominator factors out.) "NURBS" means "**N**on-**U**niform **R**ational **B**-**S**pline." The "non-uniform" means that we allow their knot multiplicity to vary. The acronym is singular.

NURBS Properties I

- B-splines are affinely, but not projectively (as in perspective) invariant. NURBSes are both.
- The transformation of a NURBS is a NURBS generated by the (same) transformation of its control points.
- NURBSes can represent conics. (Rational quadratic polynomials being NURBSes.)

NURBS Properties II

- NURBSes are the most common curve and surface modelling tool in use today.
  - used in marine, industrial, automotive, jewelry, and just about every other kind of computer-aided design as well as the film and game industries
  - "Rhinoceros 3D" is a NURBS design system developed at McNeel & Associates (in Seattle)
  - We are only scratching the surface of what can be done with NURBSes. For more details, see the standard reference: Piegl & Tiller's *The NURBS Book* or Farin's book (see above).

Other Splines

Catmull[1]-Rom Like B-splines, but they *interpolate* points. They do not have the convex hull property. Nevertheless, they are useful for animation.

β-splines biased splines ($0 \leq \beta \leq 1$) are a variation of B-splines. Varying β controls the "tension": how closely the curve comes to the control points.

Cardinal

Kochanek-Bartels

etc...etc...etc

[1] Ed Catmull is the President of Pixar and Walt Disney Animation. He is also winner of 5 Academy Awards.