Curves and Surfaces

- Why Curves and Surfaces?
- Parametric Curves & Surfaces
- Subdivision Curves & Surfaces

Why Curves and Surfaces?

- Often want arbitrary shapes rather than geometric shapes, e.g.
  - “Freehand” drawings
  - Natural objects (e.g. animals)
  - CAD (e.g. mechanical engineering)
    - mechanical engineering (e.g. car bodies)
    - pottery (the Great Teapot)
- Creates problems of…
  - How to represent arbitrary curves and surfaces
  - How to interactively design them
  - How to render them
    - How to render their geometry
    - How to texture-map them

Parametric Curves

References: Hill §11; Foley & van Dam et al (F&vD) §11.2

The simplest curve representation is a sequence of straight line segments. But requires too many points to get something reasonably smooth looking.

In these notes we look at ways of piecing together higher-order curves (almost inevitably cubics) to achieve continuity of gradient with far fewer points.

- Introduction
- Hermite Curves
- Bezier Curves
- Uniform B-Spline Curves
- Catmull-Rom Spline Curve
- Non-uniform B-splines
- Non-uniform Rational B-spline [NURBS]

Introduction

- Parametric Straight Lines
- Extension to higher order curves
- Putting the bits together

Parametric Straight Lines

Have seen parametric equation of straight line:

$$\mathbf{p}(t) = \mathbf{p}_1 + t(\mathbf{p}_2 - \mathbf{p}_1) = (1-t)\mathbf{p}_1 + t\,\mathbf{p}_2$$

The factors (1-*t*) and *t* are blending functions that select the "mix" of $\mathbf{p}_1$ and $\mathbf{p}_2$ for any value of *t*.

Can also be written as:

$$\mathbf{p}(t) = \begin{pmatrix} t & 1 \end{pmatrix} \begin{pmatrix} -1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \mathbf{p}_1 \\ \mathbf{p}_2 \end{pmatrix} = \mathbf{T} \cdot \mathbf{M} \cdot \mathbf{G}$$

where **T** is called the "power basis", **M** the "basis matrix", and **G** the "geometric constraint vector".

Extension to higher order curves

The equation **p** = **T**.**M**.**G** can be extended to higher order curves:

Quadratic Curves: $\mathbf{T} = (t^2 \quad t \quad 1)$

M is a 3 x 3 matrix,
G is a 3-element vector (of vectors!)

Cubic Curves: $\mathbf{T} = (t^3 \quad t^2 \quad t \quad 1)$

M is a 4 x 4 matrix
G is a 4-element vector

etc.

Following Foley et al we concentrate on cubic curves – the most common sort.

Putting the bits together

Complex curves are built by assembling cubic curves end to end.

Generally want "continuity". Can distinguish between G^n and C^n continuity classes.

- **G^0 continuity.**
- **G^1 continuity.**
- **C^1 continuity.**
- **C^2 continuity.**

G^0 continuity.

- Zeroth order **G**eometric Continuity
- End points match.

G^1 continuity.

- First order **G**eometric Continuity
- End-points *and* gradients match.
  - This implies two constraints at each end of curve, i.e. 4 in total.

C^1 continuity.

- First order *parametric continuity*.

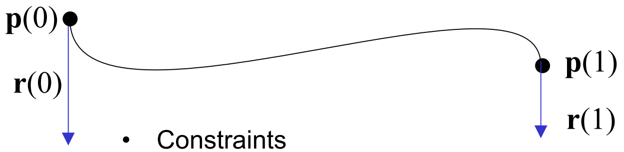
  Requires G^1 continuity AND “speed” around curve wrt *t* continuous, i.e. $d\mathbf{p}/dt$ (parametric tangent vector) matches at join.

C^2 continuity.

- Second order parametric continuity
  - Requires C^1 continuity plus matching of 2nd derivative of **p** wrt *t*.

Hermite Curves

A Hermite curve is a cubic polynomial curve segment constrained to a given position **p** and tangent vector **r** at each endpoint



- Constraints
- The Basis Matrix
- The Blending Functions
- Properties
- Interactive Design
- Piecing Hermites Together
- Drawing Hermites

Constraints

Have constraint vector

$$\mathbf{G} = (\mathbf{p}_1, \mathbf{p}_4, \mathbf{r}_1, \mathbf{r}_4)$$

where subscripts 1 and 4 denote the two endpoints
(reserving 2 and 3 for mid-curve control points later!).

$$\left.\begin{array}{ll} \text{At } t = 0\text{, want} & \mathbf{p}(t) = \mathbf{p}_1,\ \mathbf{p}'(t) = \mathbf{r}_1 \\ \\ \text{At } t = 1\text{, want} & \mathbf{p}(t) = \mathbf{p}_4,\ \mathbf{p}'(t) = \mathbf{r}_4. \end{array}\right\} \text{4 constraints}$$

where $\mathbf{p}'(t)$ = parametric tangent vector:

$$\mathbf{p}'(t) = \frac{d(\mathbf{TMG})}{dt} = \underbrace{\begin{pmatrix} 3t^2 & 2t & 1 & 0 \end{pmatrix}}_{\mathbf{T'}} \cdot \mathbf{M} \cdot \mathbf{G}$$

Constraints (cont'd)

Substituting into **p=TMG** and **p'=T'MG** the constraints are:

$$\begin{aligned} \mathbf{p}(0) &= \begin{pmatrix} 0 & 0 & 0 & 1 \end{pmatrix}.\mathbf{M}.\mathbf{G} = \mathbf{p}_1 \\ \mathbf{p}'(0) &= \begin{pmatrix} 0 & 0 & 1 & 0 \end{pmatrix}.\mathbf{M}.\mathbf{G} = \mathbf{r}_1 \\ \mathbf{p}(1) &= \begin{pmatrix} 1 & 1 & 1 & 1 \end{pmatrix}.\mathbf{M}.\mathbf{G} = \mathbf{p}_4 \\ \mathbf{p}'(1) &= \begin{pmatrix} 3 & 2 & 1 & 0 \end{pmatrix}.\mathbf{M}.\mathbf{G} = \mathbf{r}_4 \end{aligned}$$

The Basis Matrix

These four equations can be written:

$$\begin{pmatrix} p_1 \\ p_4 \\ r_1 \\ r_4 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix} \mathrm{M} \begin{pmatrix} p_1 \\ p_4 \\ r_1 \\ r_4 \end{pmatrix}$$

From which we get

$$\mathrm{M} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix}^{-1} = \begin{pmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

The Blending Functions

Have

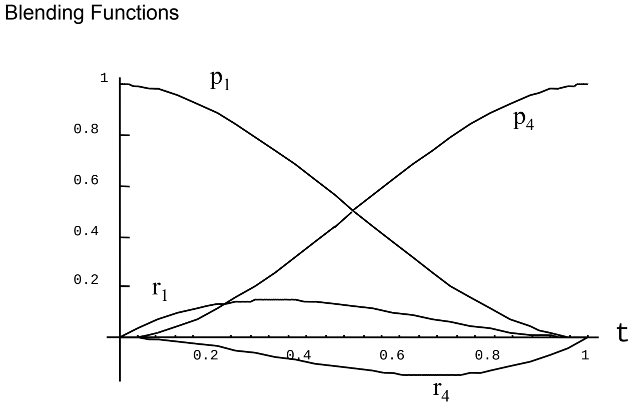
$$\mathbf{p} = \mathbf{TMG} = \begin{pmatrix} t^3 & t^2 & t & 1 \end{pmatrix} \begin{pmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} \mathbf{p}_1 \\ \mathbf{p}_4 \\ \mathbf{r}_1 \\ \mathbf{r}_4 \end{pmatrix}$$

Can expand to

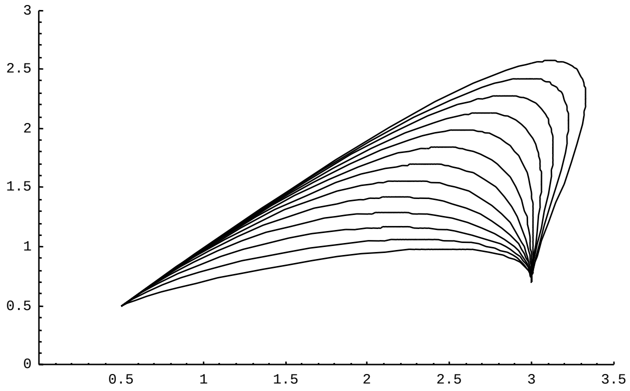
$$\mathbf{p}(t) = \left(2t^3 - 3t^2 + 1\right)\mathbf{p}_1 + \left(-2t^3 + 3t^2\right)\mathbf{p}_4 + \left(t^3 - 2t^2 + t\right)\mathbf{r}_1 + \left(t^3 - t^2\right)\mathbf{r}_4$$

giving us the blending functions that apply to the four geometry constraint vector components.

Plots of the Blending Functions

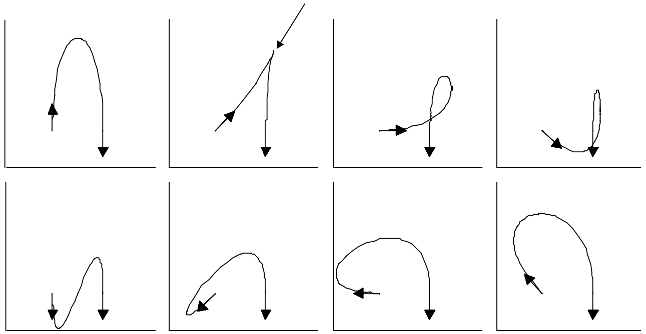


Varying magnitude of r_1, everything else fixed.



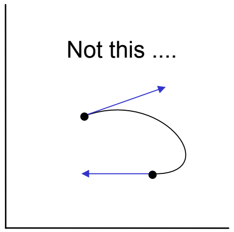
Varying direction of r_1, everything else fixed.

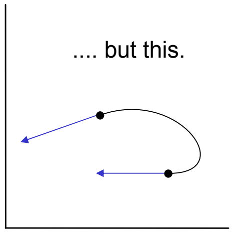
NB: C^1 continuous but not G^{1}continuous!

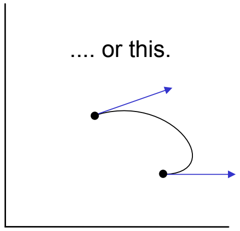


Interactive Design

- Display “handles” for control of tangents
- Normally reverse direction of $\mathbf{r}_1$ or $\mathbf{r}_4$ for symmetry







UDOO: Check out the “draw” package in MS Office.

Piecing Hermites Together

For G^1 continuity, want to match endpoints AND gradients, i.e. the successive **G** vectors must be of form:

$[\mathbf{p}_1\ \ \mathbf{p}_4\ \ \mathbf{r}_1\ \ \mathbf{r}_4]$ and $[\mathbf{p}_4\ \ \mathbf{p}_7\ \ k\mathbf{r}_4\ \ \mathbf{r}_7]$ with $k > 0$.

For C^1 continuity require $k = 1$.

Drawing Hermites

- Code to draw Hermite curve:

```
Precalculate M.G
MoveToPoint2d[ (0 0 0 1).MG ]
for t = $\delta t$ to 1 in suitably small steps of $\delta t$
   LineToPoint2d[ ($t^3$   $t^2$  t   1).MG ]
```

Bezier Curves

- Idea (text book approach)
- Bezier Basis Matrix
- Bezier Blending Functions
- Properties

Idea (F&vD approach)

Cubic Bezier curves (after Pierre Bezier, a Renault engineer) can be regarded as a variation on a Hermite curve, in which the tangent vectors are specified by two intermediate control points $\mathbf{p}_2$ and $\mathbf{p}_3$ such that

$\mathbf{r}_1 = 3(\mathbf{p}_2 - \mathbf{p}_1)$ and $\mathbf{r}_4 = 3(\mathbf{p}_4 - \mathbf{p}_3)$.

Factor of 3 is the value such that a sequence of equally spaced points $\mathbf{p}_1$ to $\mathbf{p}_4$ on a straight line gives constant parametric “speed”.

Proof: UDOO!

Bezier Basis Matrix

If subscripts *H* and *B* denote Hermite and Bezier respectively, can see that

$$\mathbf{G}_H = \begin{pmatrix} \mathbf{p}_1 \\ \mathbf{p}_4 \\ \mathbf{r}_1 \\ \mathbf{r}_4 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 \end{pmatrix} \begin{pmatrix} \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \\ \mathbf{p}_4 \end{pmatrix} = \mathbf{M}_{HB} \cdot \mathbf{G}_B$$

Now $\mathbf{p} = \mathbf{T}\,\mathbf{M}_H\,\mathbf{G}_H = \mathbf{T}\,\mathbf{M}_H\,\mathbf{M}_{HB}\,\mathbf{G}_B = \mathbf{T}\,\mathbf{M}_B\,\mathbf{G}_B$
where

$$\mathbf{M}_B = \mathbf{M}_H \mathbf{M}_{HB} = \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

Bezier Blending Functions

Can then expand $\mathbf{T}.\mathbf{M}_B$ to get

$$\mathrm{p}(t) = (1-t)^3\,\mathrm{p}_1 + 3t(1-t)^2\,\mathrm{p}_2 + 3t^2(1-t)\,\mathrm{p}_3 + t^3\,\mathrm{p}_4$$

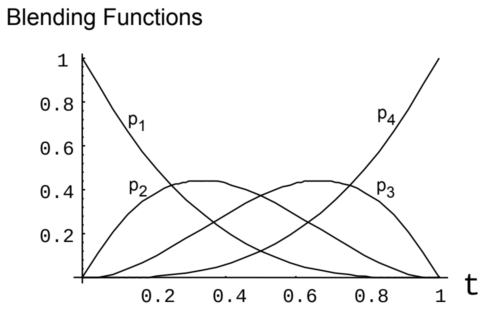
The blending functions are the Bernstein polynomials; successive terms in the binomial expansion of $[\,(1\text{-}t) + t\,]^3$.

Generalization: an n^{th} degree Bezier curve has (n-1) control points, with blending functions being terms in the expansion of $[\,(1\text{-}t) + t\,]^n$

Properties

- Blending Functions
- Convex Hull Property
- Continuity Conditions
- De Casteljau’s Construction

Blending Functions



Convex Hull Property

- The blending functions are the terms in the expansion of $[(1-t) + t]^3$
- Hence they sum to 1 for any value of t
- Hence any point $\mathbf{p}(t)$ is a "convex sum" of all the $\mathbf{p}_i$
- Hence, $\mathbf{p}$ lies within the convex hull of the set of points $\mathbf{p}_i$.

Continuity Conditions

- If two successive Bezier curves a and b are to be G^1 continuous, require
  - $\mathbf{p}_{4a} = \mathbf{p}_{1b}$, and
  - $(\mathbf{p}_{3a} - \mathbf{p}_{4a}) = k\,(\mathbf{p}_{1b} - \mathbf{p}_{2b}) \quad k > 0.$
- For C^1 continuity, require $k = 1$.

De Casteljau's Construction

(The usual way of defining Bezier curves)

Given n control points, n > 1, define a curve as follows:

```
Point PointOnCurve (PointList points, float t) {
   // A point at a given parametric distance t on a curve
   // defined by a sequence of control points.
   if (points.length() == 1) return points[0];
   else return CurvePoint(reducedPointSet(points), t);
}

PointList reducedPointSet(PointList inList, float t) {
   PointList outList = new PointList();
   for each successive pair (pa, pb) of points in inList
        outList.add( (1-t)*pa + t*pb );
   return outList;
}
```

UDOO: Prove this is a Bezier curve of degree n-1.

Uniform B-Spline Curves

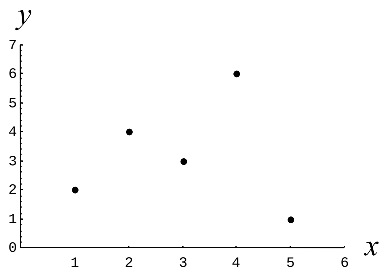
- The Problem with Hermite/Bezier Curves
- Interlude – Interpolation and Smoothing
- Back to the Main Thread (Uniform B-spline Curves)

The Problem with Hermite/Bezier Curves

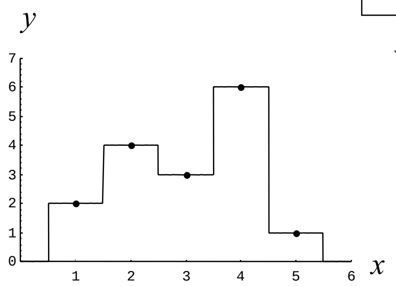
- Piecing together many Hermite or Bezier curves is a hassle
  - Continuity conditions are clumsy to enforce.
- Can move to higher-order Bezier curves – e.g. 20 control points, with a 19th degree polynomial curve. But:
  - Moving any one control point affects the whole curve.
  - It's slow to calculate each point.
- Want LOCAL CONTROL
  - Moving one control point should affect only the immediate vicinity of the curve.
- B-spline curves are a solution

Interlude – Interpolation and Smoothing

Consider a sequence of uniformly-spaced samples, y_0, y_1, ...
How do we interpolate to get a smooth function?



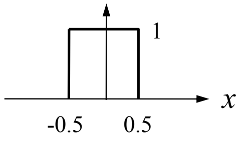
Piecewise Constant
(aka "Nearest Neighbour")



$$\boxed{y(x) = \sum y_i U(x-i)}$$

$$\text{where } U(x) = \begin{cases} 1 & -0.5 \le x < 0.5 \\ 0 & \text{otherwise} \end{cases}$$

[The unit "square pulse" function]



Convolutional Smoothing

- Piecewise constant is not smooth enough
- Common smoothing technique is "convolutional smoothing"
  - Smoothed value at any point is the average of the input function in the vicinity of the point
  - Unweighted average over a fixed interval is called "running mean"
  - Generally have a weight function or *filter* function, *h*(*x*)

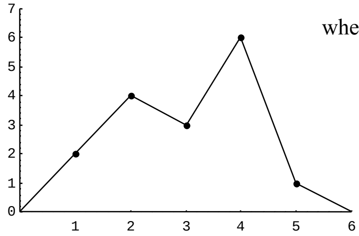
$$f_{smooth}(x) = f * h = \int_{-\infty}^{\infty} f(u)h(x-u)\,du$$

  - Box filtering is convolutional smoothing with square pulse, *h* = *U*

Piecewise Linear Interpolation

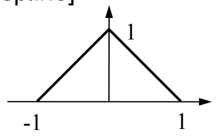
Obtained by “box filtering” nearest-neighbour plot.

$$y(x)=\sum y_i\, L(x-i)$$



where $L(x)=U(x)*U(x)=\begin{cases}1+x & -1\le x<0\\ 1-x & 0\le x<1\\ 0 & \text{otherwise}\end{cases}$

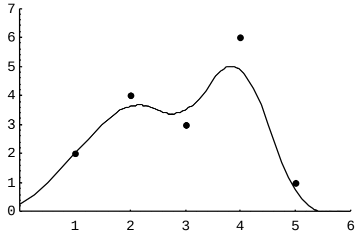
[The “tent” function - aka linear b-spline]



Piecewise Quadratic *Approximation*

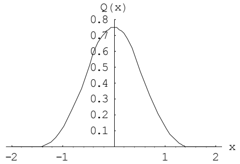
NB: no longer interpolates points

$$y(x)=\sum y_i\, Q(x-i)$$



where $Q(x)=L(x)*U(x)=\begin{cases}\dfrac{(2x+3)^2}{8} & -\dfrac{3}{2}\le x<-\dfrac{1}{2}\\ \dfrac{3}{4}-x^2 & -\dfrac{1}{2}\le x<\dfrac{1}{2}\\ \dfrac{(2x-3)^2}{8} & \dfrac{1}{2}\le x<\dfrac{3}{2}\\ 0 & \text{otherwise}\end{cases}$

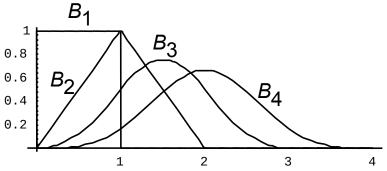
[The “Quadratic B-Spline” function]



The Uniform B-spline Functions — Definition

$$B_{m+1}(x)=B_m(x)*B_1(x)$$

$$B_1(x)=\begin{cases}1 & 0\le x<1\\ 0 & \text{otherwise}\end{cases}$$



- Note change of origin – easier formulae!
- Set of all integer translates of a B-spline function of a given order is a *basis* for a piecewise approximation space of that order.
  - Hence name “B-spline”

Cox-deBoor Recurrence

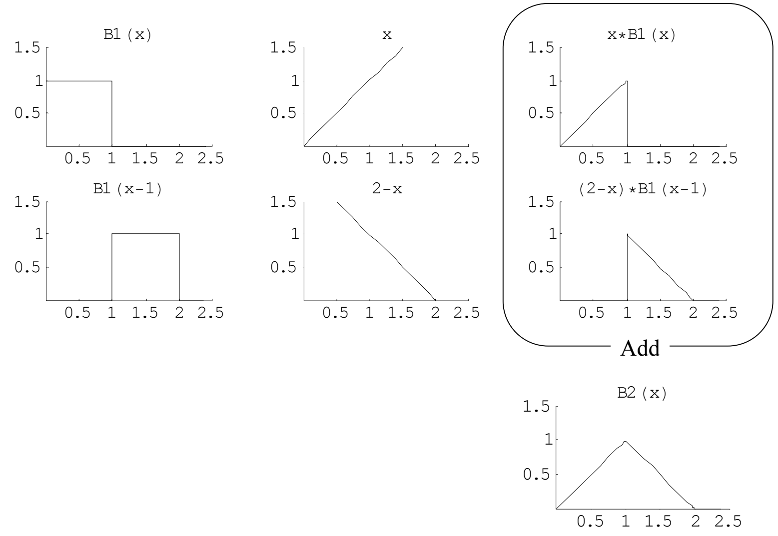
- An alternative, more convenient, recurrence formula is:

$$B_m(x)=\frac{xB_{m-1}(x)+(m-x)B_{m-1}(x-1)}{m-1}$$

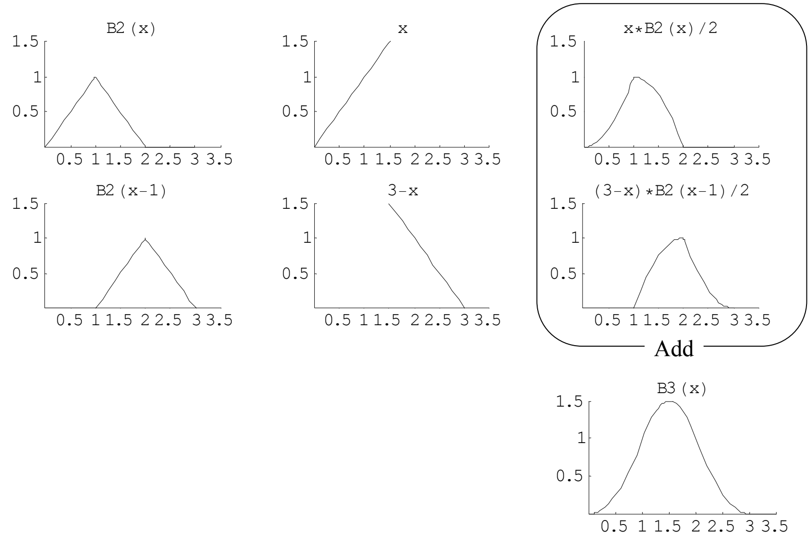
$$B_1(x)=\begin{cases}1 & 0\le x<1\\ 0 & \text{otherwise}\end{cases}$$

- Called “Cox-deBoor recurrence”
  - [Or a special case of it – see later]

Cox-deBoor for B_2



Cox-deBoor for B_3



Back to the Main Thread (Uniform B-spline Curves)

- What is a Uniform B-spline Curve?
- Examples
- Properties of Uniform B-splines
- End-point Replication
- The F&vD Formulation

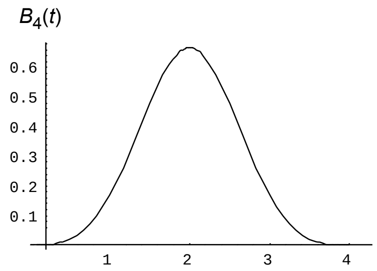
What is a Uniform B-spline Curve?

- A curve in which uniform B-spline functions are used as blending functions. Usually use cubic B-splines, m= 4. With *n* >= 4 control points:

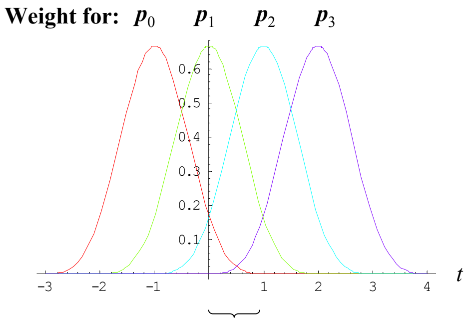
$$\mathbf{p}(t) = \sum_{i=0}^{n-1} \mathbf{p}_i B_4(t-i+3) \quad \text{with } t \text{ in range } [0, n-3]$$

$$B_4(t) = \frac{1}{6}\begin{cases} t^3 & 0 \le t < 1 \\ v(2-t) & 1 \le t < 2 \\ v(t-2) & 2 \le t < 3 \\ (4-t)^3 & 3 \le t < 4 \\ 0 & \text{otherwise} \end{cases}$$

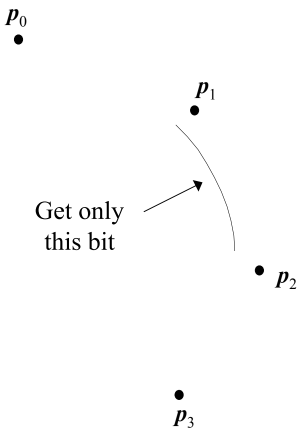
where $v(s)=(3s^3-6s^2+4)$



Example 1: n = 4

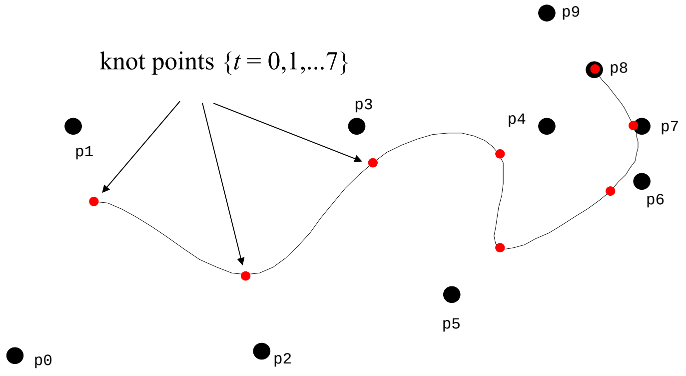


Curve defined only over [0,1]



Get only this bit

Example 2: n = 10



See also F&vD Figs 11.22 and 11.23

The F&vD Formulation

- Have (m+1) control points, $\mathbf{p}_0 \ldots \mathbf{p}_m$ (m >= 3)
- The full curve is made up of (m −2) cubic polynomial curve segments $\mathbf{q}_3 \ldots \mathbf{q}_m$
- Segment $\mathbf{q}_i$ has the B-spline geometry constraint vector

$$\mathbf{G}_{Bs_i} = \begin{pmatrix} \mathbf{p}_{i-3} \\ \mathbf{p}_{i-2} \\ \mathbf{p}_{i-1} \\ \mathbf{p}_i \end{pmatrix}, \quad 3 \le i \le m$$

- Each control point thus affects four of the curve segments.
- Each segment goes from somewhere in the vicinity of $\mathbf{p}_{i-2}$ to somewhere in the vicinity of $\mathbf{p}_{i-1}$
  - UDOO -- where *exactly* does the segment start and end?

The F&vD Formulation (cont.)

- The B-spline basis matrix $\mathbf{M}_{Bs}$ is:

$$\mathbf{M}_{Bs} = \frac{1}{6}\begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{pmatrix}$$

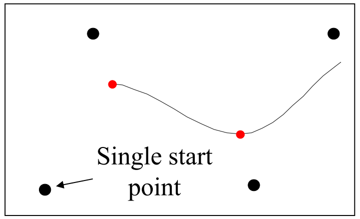
- UDOO:
  - determine the blending functions from this matrix and relate them to the cubic B-spline definition on slide 41.

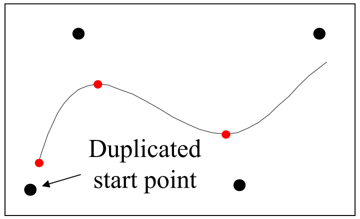
Properties of Uniform B-splines

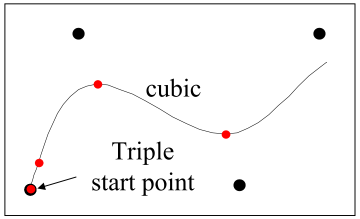
- Assuming all control points distinct, have C^2 continuity (cf. C^1 for Hermite/Bezier)
  - 2nd derivatives match at “knot” points (where the separate curves join).
- Each curve segment lies within convex hull of its associated control points
  - Proof: UDOO
- In general, none of the control points are on the curve, but can replicate control points
  - Particularly first and last

Replicating End-points

Start of example 2 curve with different start-point *multiplicity*.





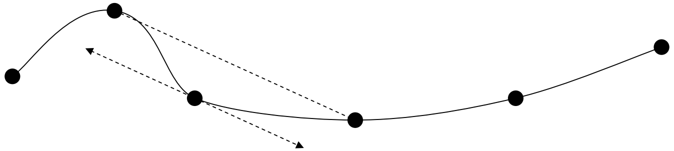


- Can replicate end point similarly to force curve to start and end at first and last points.
- BUT first segment of curve is linear, second is quadratic.
- Not ideal.

Interlude: Catmull-Rom Spline Curve

Gives a smooth curve passing through a set of points (except first and last — have to invent extra points at ends!).

- Multiple segments (like B-spline)
- Each segment is a Hermite (or a Bezier!)
- Parametric tangent at point $\mathbf{p}_i$ is $(\mathbf{p}_{i+1} - \mathbf{p}_{i-1})/2$
- Easy to implement
  - Like uniform B-spline, but with a different basis matrix
    - UDOO: Deduce basis matrix
  - Or can draw as multiple Hermites/Beziers
- No convex hull property – can be “unstable”



Non-uniform B-splines

- Rationale
- The Knot Vector
- The Generalised Cox-deBoor Recurrence Formula
- The End-Point-Interpolating B-splines
- Example

Rationale

- Want to specify exact start and end points
- Replicating start and end points 3 times gives linear end segments
  - Unsatisfactory

The Knot Vector

- With previous B-splines, had knots at uniform intervals in *t*
  - *Knot vector* (values of *t* at the knots) was (0,1,2,3,....)
- We now generalise to allow arbitrary (non-decreasing) knot vector $\{t_k\}$
- Spacing between knots determines length of corresponding segment of curve
  - So by replicating knots at start and end we can shrink the linear and quadratic segments to zero ☺

The Generalised Cox-deBoor Recurrence Formula

Notation change: use $B_{i,j}(t)$ for the *j*-th order blending function for control point $\mathbf{p}_i$.

$$B_{i,j}(t) = \frac{t-t_i}{t_{i+j-1}-t_i}B_{i,j-1}(t) + \frac{t_{i+j}-t}{t_{i+j}-t_{i+1}}B_{i+1,j-1}(t)$$

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

If denominator zero, make the term zero too (!)

First term is the corresponding lower-order term multiplied by an “up-ramp”
Second term is the next-in-sequence lower-order term mutliplied by a “down-ramp”.

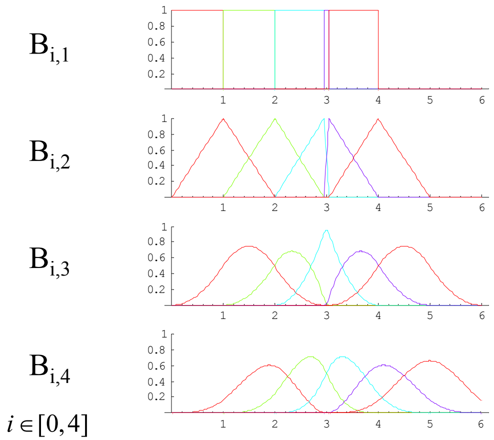
UDOO: Show that this reduces to the earlier version for uniform knots $t_k=k$

Still have convex hull property for any segment of curve: $\sum_i B_{i,j}(t) = 1$

A Repeated Knot (almost!)

Knot vector = {0,1,2,2.95,3.05,4,5,....)

(Repeated knots separated slightly for clarity)



The quadratic B-spline passes through $\mathbf{p}_2$ when multiplicity = 2. Cubic would pass through it if multiplicity = 3.

A Repeated Root at the Start

- Knot vector = {0,0.05,1,2,3,4,...} [again, repeated roots separated slightly]

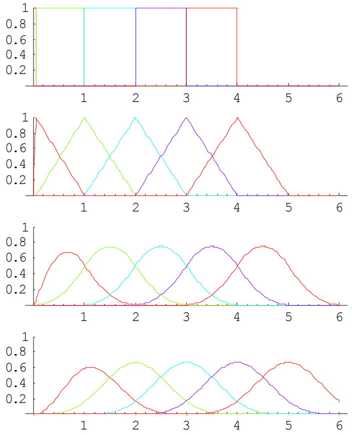
$B_{i,1}$

$B_{i,2}$

$B_{i,3}$

$B_{i,4}$

$i \in [0,4]$



Curve interpolates $\mathbf{p}_0$

A Multiplicity-3 Root at the Start

- Knot vector = {0,0,0,1,2,3,4,...} [truly equal knots now]

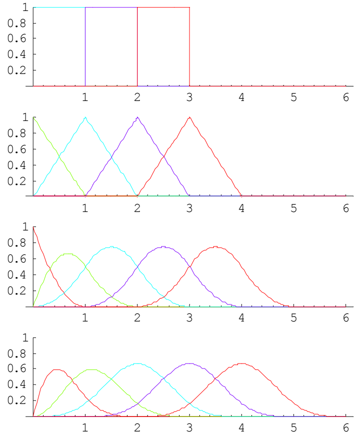
$B_{i,1}$

$B_{i,2}$

$B_{i,3}$

$B_{i,4}$

$i \in [0,4]$



Curve interpolates $\mathbf{p}_0$

A Multiplicity-4 Root at the Start

- Knot vector = {0,0,0,0,1,2,3,4,...}

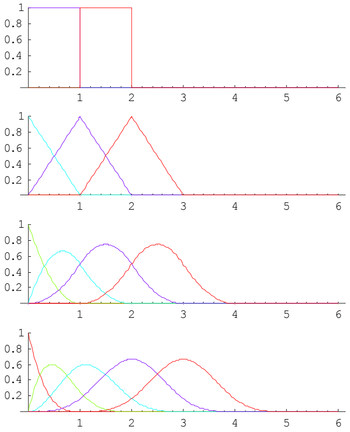
$B_{i,1}$

$B_{i,2}$

$B_{i,3}$

$B_{i,4}$

$i \in [0,4]$



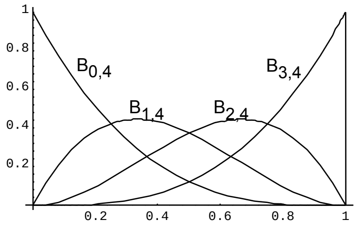
Curve interpolates $\mathbf{p}_0$

The End-Point Interpolating Cubic B-Splines

- From previous slides, see that a multiplicity 4 knot at the start allows us to interpolate the start point.
- Similarly at the end.
- Hence, can set up “end-point interpolating” B-splines
  - With *n* control points {p_0, $p_{1,...,}$ p_{n-1}} ,knot vector is {0,0,0,0,1,2,3,4,....*n*-4,*n*-3, *n*-3,*n*-3,*n*-3}
  - Called “the standard knot vector”

Knot vector (0,0,0,0,1,1,1,1)

- Need 4 control points (one curve segment)
- Blending functions

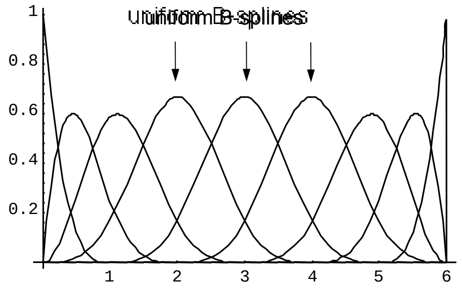


Warning: Fig. 11.26 in the *first* printing of F&vD, showing derivation of these, is nonsense. Fixed in later printings.

- These are exactly the cubic Bezier functions!!

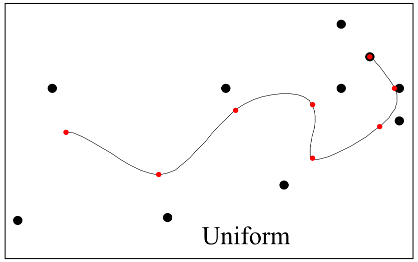
Knot vector (0,0,0,0,1,2,3,4,5,6,6,6,6)

- Implies 9 control points (6 curve segments)
- Blending functions:

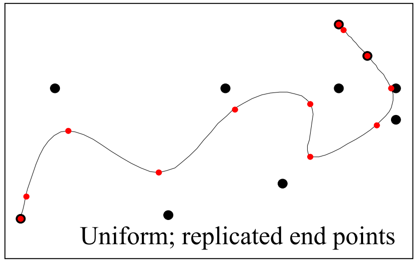


Middle functions same as uniform B-splines

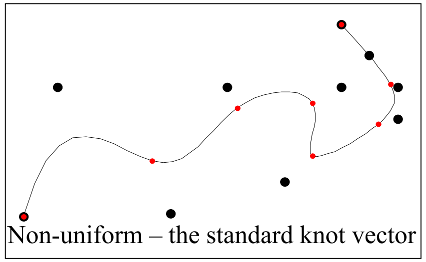
Example



Uniform



Uniform; replicated end points



Non-uniform – the standard knot vector

NB: SLIDE CHANGED FROM VERSION IN HANDOUT

Non-uniform Rational B-splines
[NURBS]

- NURBS are effectively non-uniform B-splines defined in homogeneous coordinates.
- Each control point has 4 components: $\tilde{P}_k = (x_k, y_k, z_k, w_k)$
- "Rational" because after weighting by the B-spline functions and projecting back to 3-space we get [UDOO]:

$$\mathbf{p}(t) = \frac{\sum_k \mathbf{p}_k B_{k,m}}{\sum_k w_k B_{k,m}} = \boxed{\frac{\sum_k w_k \mathbf{q}_k B_{k,m}}{\sum_k w_k B_{k,m}}}$$

The usual text-book form

$$\text{where } \mathbf{q}_k = \left(\frac{x_k}{w_k}, \frac{y_k}{w_k}, \frac{z_k}{w_k}\right) = \left(x_k', y_k', z_k'\right)$$

Benefits of NURBS

- Can represent conic sections, e.g. circle, with quadratic NURBS
  - – UDOO: Prove that the quadratic Bezier with the following 2D homogeneous coordinates defines a 2D quarter circle: (0,1,1), (√2/2, √2/2, √2/2), (1,0,1)
- Are a superset of all other curves studied so far
  - – e.g. for uniform B-splines, set $w_k = 1$, choose uniform knot sequence. For Bezier curve .... [UDOO]

Cool B-spline applet:
http://www.cs.technion.ac.il/~cs234325/Homepage/Applets/applets/bspline/html/

Parametric Bi-Cubic Surfaces

Surfaces (2D) involve two parameters rather than one.
“Bi-cubic” means that each of the parameters is a cubic.

- From Curves to Surfaces
- A Matrix Formulation
- Bezier Surfaces
- Tensor Product Form
- Joining Bezier Patches
- B-Spline Surfaces
- Displaying Bi-cubic Patches

From Curves to Surfaces

- The equation

$$\mathbf{p}(t) = \mathbf{T}.\mathbf{M}.\mathbf{G}$$

defines 3D curves

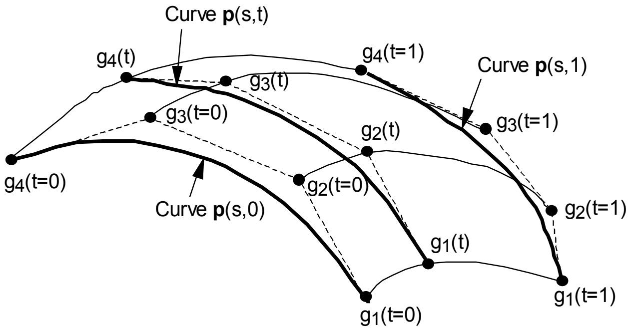
- Changing the parameter t to s (so that we think of the parameter as a “distance” rather than a “time”) gives, instead

$$\mathbf{p}(s) = \mathbf{S}.\mathbf{M}.\mathbf{G}$$

- Assume that each $\mathbf{g_i}$ is a point in 3-space (forget about Hermites from now on), which is moving in time, t, i.e. is $\mathbf{g_i}(t)$.

From Curves to Surfaces (cont.)

- The curve $\mathbf{p}(s,t)$ thus traces out a surface.

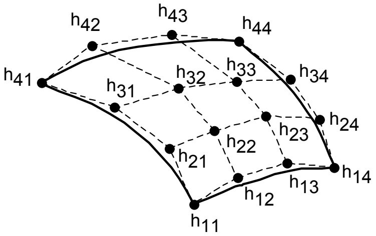


A Matrix Formulation

- Suppose the ith control point is $\mathbf{g}_i(t) = \mathbf{T}.\mathbf{M}.\mathbf{H}_i$

  where $\mathbf{H}_i = [\mathbf{h}_{i1}\ \mathbf{h}_{i2}\ \mathbf{h}_{i3}\ \mathbf{h}_{i4}]^T$.
- Taking the transpose, and using the general result that $(A.B.C)^T = C^T.B^T.A^T$ [and the fact that $\mathbf{g}_i^T(t) = \mathbf{g}_i(t)$, since it is a single element of the geometry vector] gives

  $\mathbf{g}_i(t) = [\mathbf{h}_{i1}\ \mathbf{h}_{i2}\ \mathbf{h}_{i3}\ \mathbf{h}_{i4}].\mathbf{M}^T.\mathbf{T}^T$



- Hence $\mathbf{G} = \mathbf{H}.\mathbf{M}^T.\mathbf{T}^T$
- So equation for surface is

  $\mathbf{p}(s, t) = \mathbf{S}.\mathbf{M}.\mathbf{H}.\mathbf{M}^T.\ \mathbf{T}^T$

Bezier Surfaces

- We have $\mathbf{p}(s, t) = \mathbf{S}.\mathbf{M}.\mathbf{H}.\mathbf{M}^T.\mathbf{T}^T$. For Bezier surfaces, just use

$$\mathbf{M} = \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & 6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

Tensor Product Form

- Previous equation is normally written in tensor product ("blending function") form, obtained by multiplying out the above:

$$\mathrm{p}(s,t) = \sum_{i=1}^{4} \sum_{j=1}^{4} B_i(s) B_j(t) h_{ij}$$

where $B_i(x)$ is the ith cubic Bernstein polynomial:

$B_1 = (1-x)^3$, $B_2 = 3x(1-x)^2$, $B_3 = 3x^2(1-x)$, $B_4 = x^3$

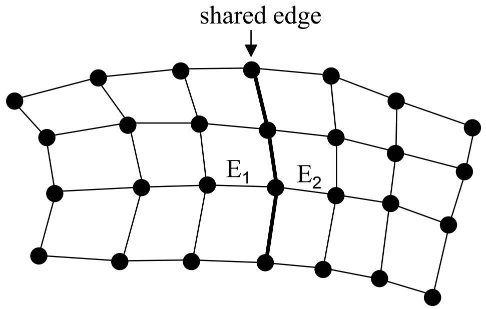
- Some texts claim that this form of the equation is more numerically stable, though slower to evaluate.

Joining Bezier Patches

For G^1 continuity, need

- 4 control points in common
- Colinearity of each of the four groups of three control points that cross the boundary.

UDOO: Deduce C^1 continuity condition.



B-Spline Surfaces

- Just use B-spline version of matrix.
- No special conditions needed for continuity – automatically get C^2 continuity everywhere (unless have duplicated control points).

Displaying Patches: Wireframe Grid

Simplest algorithm:

```
for s = 0 to 1 by delta_s
    MoveToPoint3d(p(s,0));
    for t = 0 to 1 by delta_t
        LineToPoint3d(p(s,t));
    end for
end for

for t = 0 to 1 by delta_t
    MoveToPoint3d(p(0,t));
    for s = 0 to 1 by delta_s
        LineToPoint3d(p(s,t));
    end for
end for
```

Displaying Patches: Adaptive Subdivision

- To polygonise patch, recursively subdivide patch into 4 until some flatness criterion satisfied (or to some fixed depth).
- Adaptive schemes tend to introduce "cracks":
  - See F&vD Fig. 11.49.
  - Can fix (with difficulty) by forcing extra vertex to lie in plane of neighbour.
- For shading, also need vertex normals.
  - Get from cross-product of two parametric tangent vectors $\partial \mathbf{p}/\partial s$ and $\partial \mathbf{p}/\partial t$. UDOO.

Subdivision Algorithms

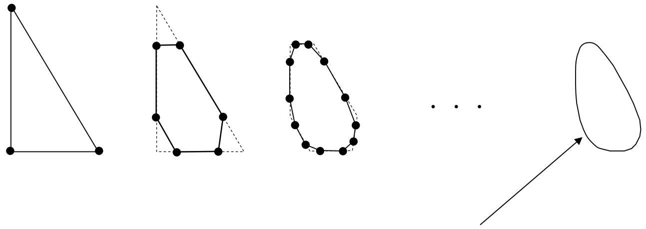
- Another approach to representing a smooth surface
- Start with a coarse polyhedron
- Repeatedly subdivide faces according to some rule

- Limit surface is smooth
- Very popular in recent years
- References:
  - Siggraph 2000 Course Notes: http://mrl.nyu.edu/~dzorin/sig00course/
  - Marcus Gross' course:
    - http://cgg.unibe.ch/teaching/lectures/ss03/ag/subdgross.pdf
    - Above images taken from there
  - Some demos and code available from http://www.subdivision.org

Subdividing a Polygon

- Chaikin's algorithm:

```
for each edge
    insert vertices at ¼ and ¾ points
discard original vertices
```

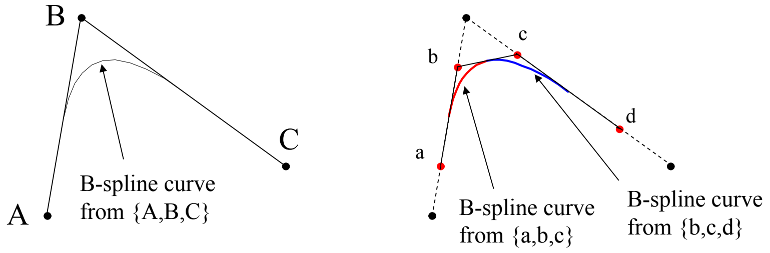


Limit curve is the quadratic B-spline defined by the original control polygon!

Proof of B-spline property

(Idea only)

- Consider an open control polygon ABC
  - Draw its quadratic B-spline segment $\mathbf{p}(t) = \begin{pmatrix} t^2 & t & 1 \end{pmatrix} \frac{1}{2} \begin{pmatrix} 1 & -2 & 1 \\ -2 & 2 & 0 \\ 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} A \\ B \\ C \end{pmatrix}$
- Subdivide to abcd (Chaikin's algorithm)
  - Easy to show that the B-spline segments due to abc and bcd together equal the segment from ABC. [UDOO]

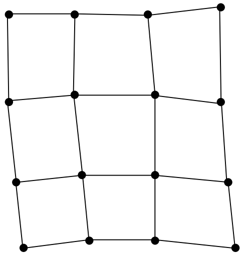


Proof of B-spline property (cont'd)

- Similarly for further subdivisions
  - B-spline curve remains unchanged
- In limit, curve = control polygon

Doo-Sabin Subdivision[1]

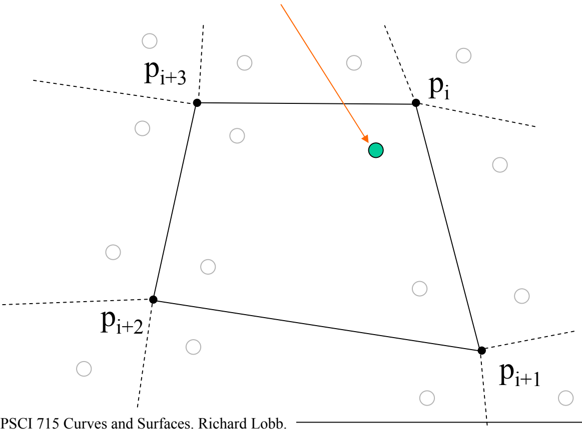
- [A slight variant on *quadratic* Catmull-Clark subdivision]
- 2D equivalent of Chaikin's algorithm
- First consider a regular [i.e. all nodes have valence 4] quadrilateral grid of control points.



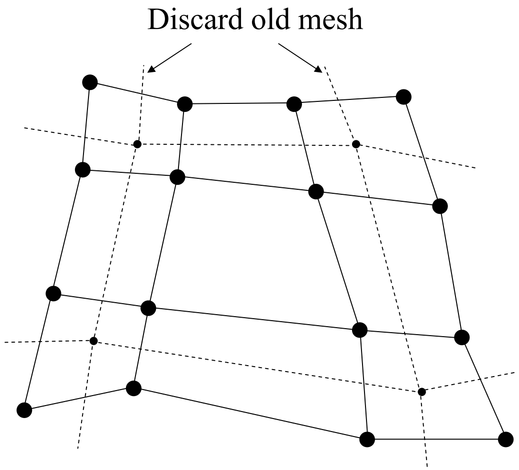
[1] DOO, D., AND SABIN, M.
Behaviour of recursive division surfaces near extraordinary points.
Computer-Aided Design 10 (Sept. 1978), 356-360.

Step 1: Insert new face vertices

- For each vertex p_i in each face $\{p_i, p_{i+1}, p_{i+2}, p_{i+3}\}$, compute a new vertex $p_i' = (9p_i + 3p_{i+1} + p_{i+2} + 3p_{i+3})/16$



Step 2: Connect to create new mesh



- Get one new face for each:
  - Face
  - Edge
  - Vertex
- Effect is to “cut off” all vertices and edges.

Extraordinary vertices

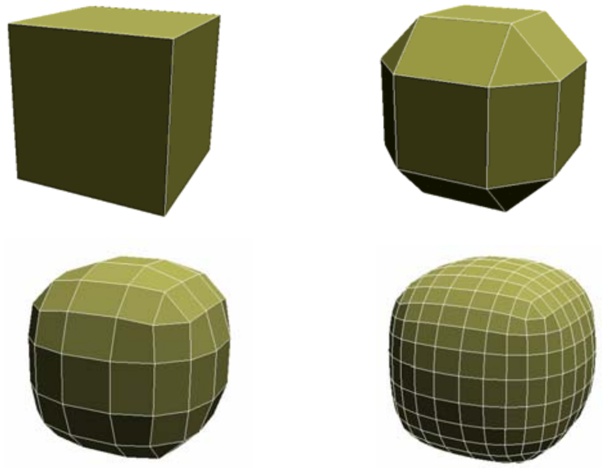
- Can extend algorithm to handle non-valence-four nodes, e.g. if mesh is a closed polyhedron.
  - These are called *extraordinary vertices*
- Algorithm is same except the rule for a new vertex is now:
  - For face with *m* vertices $\{p_i, p_{i+1}, p_{i+2}, p_{i+3} \ldots p_{i+m-1}\}$, compute a new vertex:

$$p_i' = \sum_{k=0}^{m-1} w_k p_{i+k}$$

$$\text{where } w_k = \begin{cases} \dfrac{1}{4} + \dfrac{5}{4m} & k = 0 \\ \dfrac{3 + 2\cos(2k\pi/m)}{4m} & \text{otherwise} \end{cases}$$

- NB: Gives same weights as before if *m* = 4.

Example: Doo-Sabin subdivision of cube

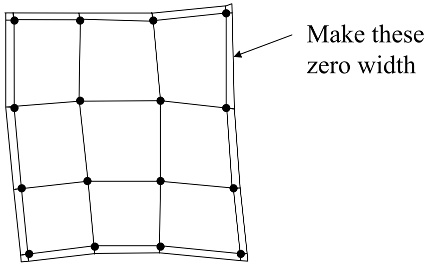


Limit Surface Properties

- For regular quadrilateral mesh it's a quadratic B-spline
  - C^1 continuous
  - Proof follows same general idea as for Chaikin's algorithm
- For general mesh it's C^1 continuous everywhere *except* at a finite number of points arising from each original extraordinary point.
- So Doo-Sabin subdivision is a *generalization* of quadratic B-spline surfaces.

Mesh Boundary

- If mesh is a polyhedron there's no open boundary.
- But what if there *is* an open boundary?
- As with B-spline curves, surface is smaller than control mesh.
- Since centroids of original faces lie on limit surface can stop mesh "shrinking inwards" by adding extra degenerate quadrilaterals around boundary.



Other subdivision algorithms

- Doo-Sabin is just one of many algorithms.
- Some other important ones (see Siggraph course notes):

1. Catmull-Clark cubic subdivision
   - Quadrilateral mesh
   - C^2 continuous (but C^1 at finite number of extraordinary points).
   - Generalizes cubic B-spline
2. Loop subdivision
   - Triangular mesh
   - C^2 continuous (but C^1 at finite number of extraordinary points).
   - Approximating (like B-splines)
3. Modified butterfly subdivision
   - Triangular mesh
   - C^2 continuous (but C^1 at finite number of extraordinary points).
   - Interpolates mesh control points

Fine structures

- Can modify subdivision algorithms to incorporate creases and variable radius bends.
- See: "Subdivision surfaces in Character Animation", De Rose, Kass, Truong. Reprinted in Siggraph 2000 Course Notes.