CSC 433/533 Computer Graphics

Joshua Levine
josh@email.arizona.edu

Lecture 27 Curves/Surfaces

Dec. 4, 2019

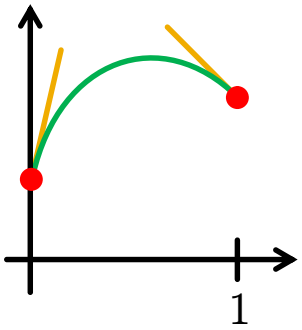
Today's Agenda

- Reminders:
  - TCEs! Please complete! 41.67% done!
  - A07 questions?
- Goals for today: finish curve modeling

Summary from Last Time

- Hermite Curves: specified by 2 control points and 2 tangents vectors
  - Solves for a cubic function and derivatives
- Bézier Curves: specified by 4 control points (the control polygon)
  - Also can be computed w/ de Casteljau's subdivision

Cubic Hermite Interpolation



$$P(t) = at^3 + bt^2 + ct + d$$
$$P'(t) = 3at^2 + 2bt + c$$

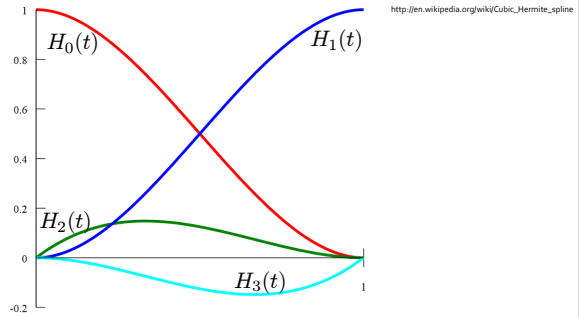
$$P(0) = d = h_0$$
$$P(1) = a + b + c + d = h_1$$
$$P'(0) = c = h_2$$
$$P'(1) = 3a + 2b + c = h_3$$

Matrix Representation

$$\begin{pmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \end{pmatrix} = \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix}$$

$$a = 2h_0 - 2h_1 + h_2 + h_3$$
$$b = -3h_0 + 3h_1 - 2h_2 - h_3$$
$$c = h_2$$
$$d = h_0$$

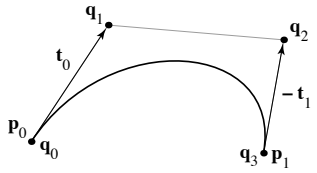
Hermite Basis



$$= h_0(2t^3 - 3t^2 + 1) + h_1(-2t^3 + 3t^2) +$$
$$h_2(t^3 - 2t^2 + t) + h_3(t^3 - t^2)$$

Hermite to Bézier

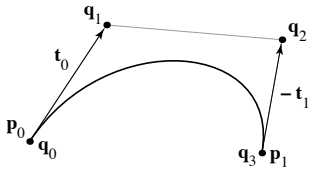
$$\mathbf{p}_0 = \mathbf{q}_0$$
$$\mathbf{p}_1 = \mathbf{q}_3$$
$$\mathbf{t}_0 = 3(\mathbf{q}_1 - \mathbf{q}_0)$$
$$\mathbf{t}_1 = 3(\mathbf{q}_3 - \mathbf{q}_2)$$



$$\begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{v}_0 \\ \mathbf{v}_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 \end{bmatrix} \begin{bmatrix} \mathbf{q}_0 \\ \mathbf{q}_1 \\ \mathbf{q}_2 \\ \mathbf{q}_3 \end{bmatrix}$$

Hermite to Bézier

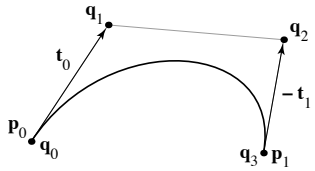
$$\mathbf{p}_0 = \mathbf{q}_0$$
$$\mathbf{p}_1 = \mathbf{q}_3$$
$$\mathbf{t}_0 = 3(\mathbf{q}_1 - \mathbf{q}_0)$$
$$\mathbf{t}_1 = 3(\mathbf{q}_3 - \mathbf{q}_2)$$



$$\begin{bmatrix}\mathbf{a}\\ \mathbf{b}\\ \mathbf{c}\\ \mathbf{d}\end{bmatrix} = \begin{bmatrix}2 & -2 & 1 & 1\\ -3 & 3 & -2 & -1\\ 0 & 0 & 1 & 0\\ 1 & 0 & 0 & 0\end{bmatrix} \begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 0 & 0 & 1\\ -3 & 3 & 0 & 0\\ 0 & 0 & -3 & 3\end{bmatrix} \begin{bmatrix}\mathbf{q}_0\\ \mathbf{q}_1\\ \mathbf{q}_2\\ \mathbf{q}_3\end{bmatrix}$$

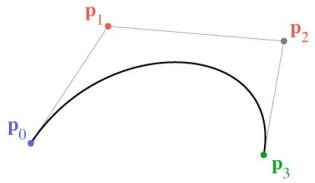
Hermite to Bézier

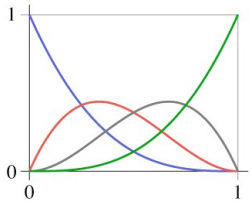
$$\mathbf{p}_0 = \mathbf{q}_0$$
$$\mathbf{p}_1 = \mathbf{q}_3$$
$$\mathbf{t}_0 = 3(\mathbf{q}_1 - \mathbf{q}_0)$$
$$\mathbf{t}_1 = 3(\mathbf{q}_3 - \mathbf{q}_2)$$



$$\begin{bmatrix}\mathbf{a}\\ \mathbf{b}\\ \mathbf{c}\\ \mathbf{d}\end{bmatrix} = \begin{bmatrix}-1 & 3 & -3 & 1\\ 3 & -6 & 3 & 0\\ -3 & 3 & 0 & 0\\ 1 & 0 & 0 & 0\end{bmatrix} \begin{bmatrix}\mathbf{q}_0\\ \mathbf{q}_1\\ \mathbf{q}_2\\ \mathbf{q}_3\end{bmatrix}$$

Bézier basis

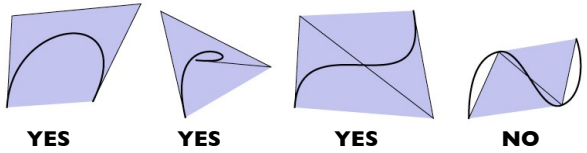




Curve Properties

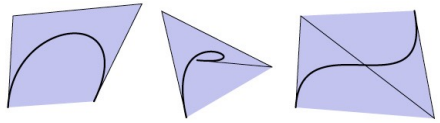
Spline segment properties

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



Convex hull

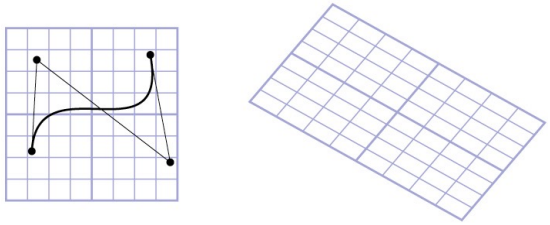
- If basis functions are all positive, the spline has the convex hull property
  - we're still requiring them to sum to 1



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

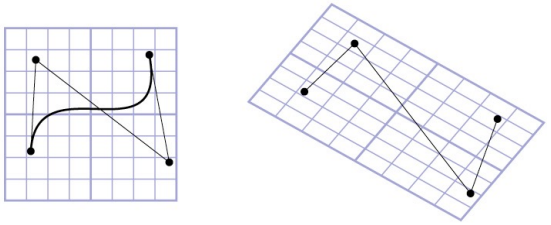
Affine invariance

- Transforming the control points is the same as transforming the curve
  - true for all commonly used splines
  - extremely convenient in practice…



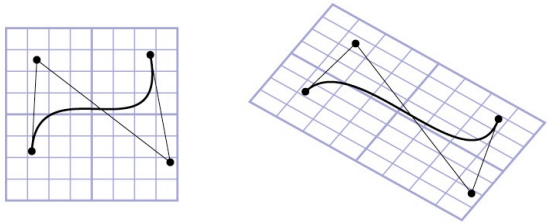
Affine invariance

- Transforming the control points is the same as transforming the curve
  - true for all commonly used splines
  - extremely convenient in practice…



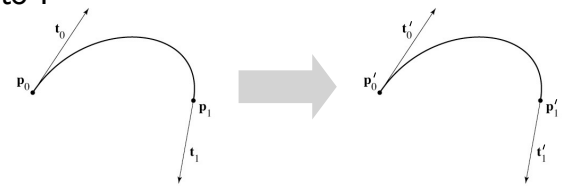
Affine invariance

- Transforming the control points is the same as transforming the curve
  - true for all commonly used splines
  - extremely convenient in practice…



Affine invariance

- Basis functions associated with points should always sum to 1

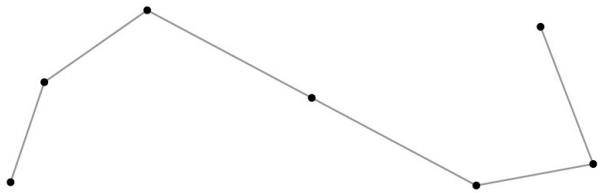


$$\begin{aligned}\mathbf{p}(t) &= b_0\mathbf{p}_0 + b_1\mathbf{p}_1 + b_2\mathbf{v}_0 + b_3\mathbf{v}_1 \\ \mathbf{p}'(t) &= b_0(\mathbf{p}_0 + \mathbf{u}) + b_1(\mathbf{p}_1 + \mathbf{u}) + b_2\mathbf{v}_0 + b_3\mathbf{v}_1 \\ &= b_0\mathbf{p}_0 + b_1\mathbf{p}_1 + b_2\mathbf{v}_0 + b_3\mathbf{v}_1 + (b_0 + b_1)\mathbf{u} \\ &= \mathbf{p}(t) + \mathbf{u}\end{aligned}$$

Interpolating vs. Approximating

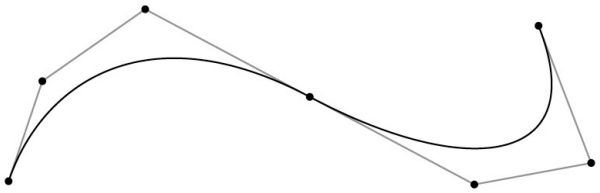
Interpolating Curves

- Pass through their control points



Approximating Curves

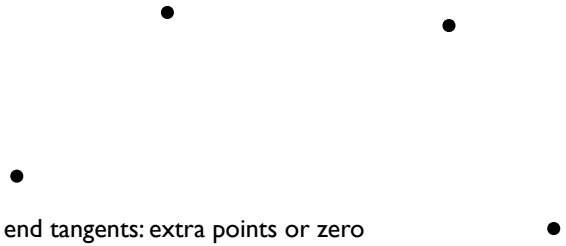
- Merely guided by the control points



Catmull-Rom Interpolation

Hermite to Catmull-Rom

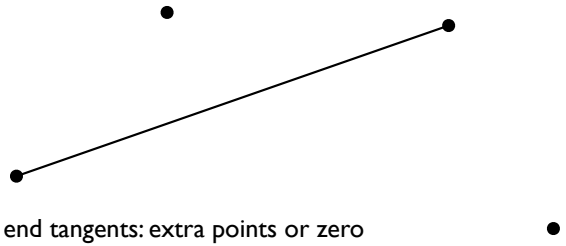
- Have not yet seen any interpolating splines
- Would like to define tangents automatically
  - use adjacent control points



  - end tangents: extra points or zero

Hermite to Catmull-Rom

- Have not yet seen any interpolating splines
- Would like to define tangents automatically
  - use adjacent control points



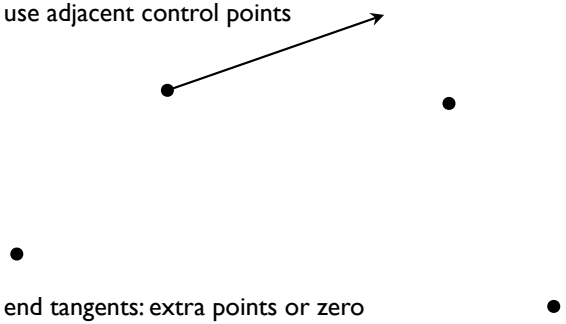
  - end tangents: extra points or zero

Cornell CS4620 Fall 2017 • Lecture 16

© 2017 Steve Marschner • 51

Hermite to Catmull-Rom

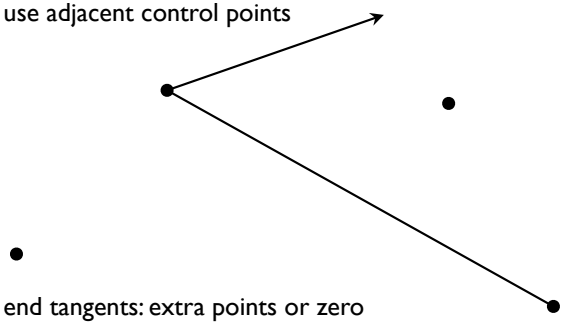
- Have not yet seen any interpolating splines
- Would like to define tangents automatically
  - use adjacent control points



  - end tangents: extra points or zero

Hermite to Catmull-Rom

- Have not yet seen any interpolating splines
- Would like to define tangents automatically
  - use adjacent control points

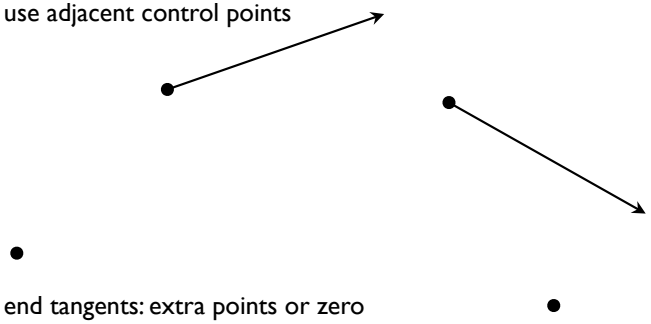


  - end tangents: extra points or zero

Cornell CS4620 Fall 2017 • Lecture 16 © 2017 Steve Marschner • 51

Hermite to Catmull-Rom

- Have not yet seen any interpolating splines
- Would like to define tangents automatically
  - use adjacent control points

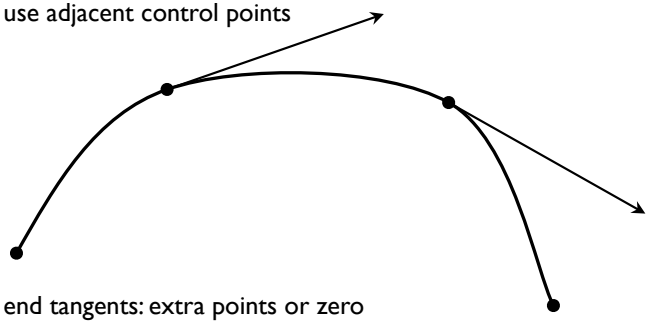


  - end tangents: extra points or zero

Cornell CS4620 Fall 2017 • Lecture 16 © 2017 Steve Marschner • 51

Hermite to Catmull-Rom

- Have not yet seen any interpolating splines
- Would like to define tangents automatically
  - use adjacent control points



  - end tangents: extra points or zero

Cornell CS4620 Fall 2017 • Lecture 16 © 2017 Steve Marschner • 51

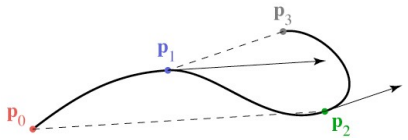
Hermite to Catmull-Rom

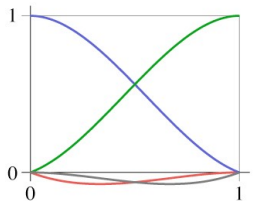
- Tangents are $(\mathbf{p}_{k+1} - \mathbf{p}_{k-1}) / 2$
  - scaling based on same argument about collinear case

$$\mathbf{p}_0 = \mathbf{q}_k$$
$$\mathbf{p}_1 = \mathbf{q}_k + 1$$
$$\mathbf{v}_0 = 0.5(\mathbf{q}_{k+1} - \mathbf{q}_{k-1})$$
$$\mathbf{v}_1 = 0.5(\mathbf{q}_{k+2} - \mathbf{q}_K)$$

$$\begin{bmatrix} \mathbf{a} \\ \mathbf{b} \\ \mathbf{c} \\ \mathbf{d} \end{bmatrix} = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -.5 & 0 & .5 & 0 \\ 0 & -.5 & 0 & .5 \end{bmatrix} \begin{bmatrix} \mathbf{q}_{k-1} \\ \mathbf{q}_k \\ \mathbf{q}_{k+1} \\ \mathbf{q}_{k+2} \end{bmatrix}$$

Catmull-Rom basis





Catmull-Rom splines

- Our first example of an interpolating spline
- Like Bézier, equivalent to Hermite
  - in fact, all splines of this form are equivalent
- First example of a spline based on just a control point sequence
- Does not have convex hull property

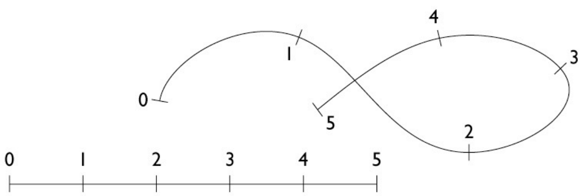
From Curves to Splines

Defining spline curves

- At the most general they are parametric curves

$$S = \{\mathbf{f}(t) \mid t \in [0, N]\}$$

- For splines, **f**(*t*) is piecewise polynomial
  - for this lecture, the discontinuities are at the integers

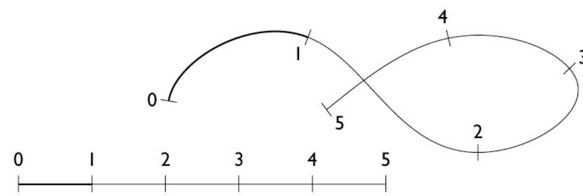


Defining spline curves

- At the most general they are parametric curves

$$S = \{\mathbf{f}(t) \mid t \in [0, N]\}$$

- For splines, **f**(*t*) is piecewise polynomial
  - for this lecture, the discontinuities are at the integers



Cornell CS4620 Fall 2017 • Lecture 16

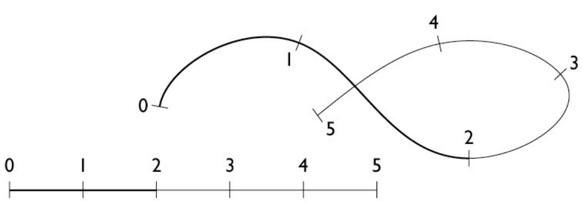
© 2017 Steve Marschner • 5

Defining spline curves

- At the most general they are parametric curves

$$S = \{\mathbf{f}(t) \mid t \in [0, N]\}$$

- For splines, **f**(*t*) is piecewise polynomial
  - for this lecture, the discontinuities are at the integers



Cornell CS4620 Fall 2017 • Lecture 16

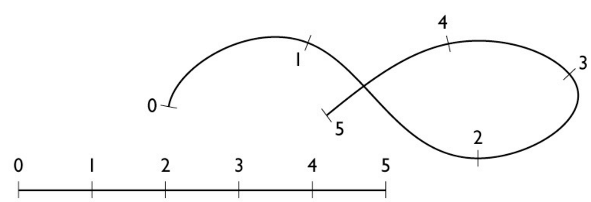
© 2017 Steve Marschner • 5

Defining spline curves

- At the most general they are parametric curves

$$S = \{\mathbf{f}(t) \mid t \in [0, N]\}$$

- For splines, **f**(*t*) is piecewise polynomial
  - for this lecture, the discontinuities are at the integers



Cornell CS4620 Fall 2017 • Lecture 16

© 2017 Steve Marschner • 5

Defining spline curves

- Generally **f**(*t*) is a piecewise polynomial
  - for this lecture, the discontinuities are at the integers
  - e.g., a cubic spline has the following form over [k, k + 1]:

$$x(t) = a_x t^3 + b_x t^2 + c_x t + d_x$$
$$y(t) = a_y t^3 + b_y t^2 + c_y t + d_y$$

  - Coefficients are different for every interval

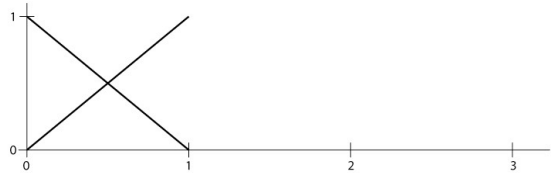
Local Control

Control

- Local control
  - changing control point only affects a limited part of spline
  - without this, splines are very difficult to use
  - many likely formulations lack this
    - natural spline
    - polynomial fits

Trivial example: piecewise linear

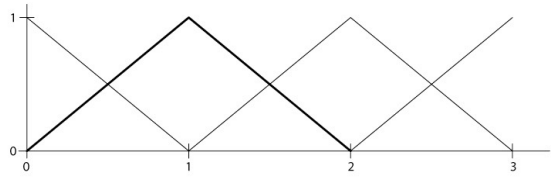
- Basis function formulation: "function times point"
  - basis functions: contribution of each point as *t* changes



  - can think of them as blending functions glued together
  - this is just like a reconstruction filter!

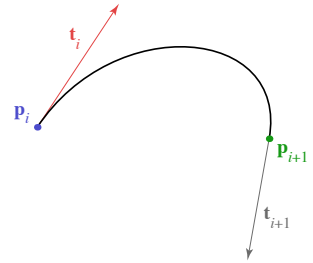
Trivial example: piecewise linear

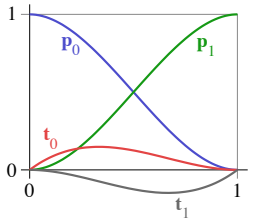
- Basis function formulation: "function times point"
  - basis functions: contribution of each point as *t* changes



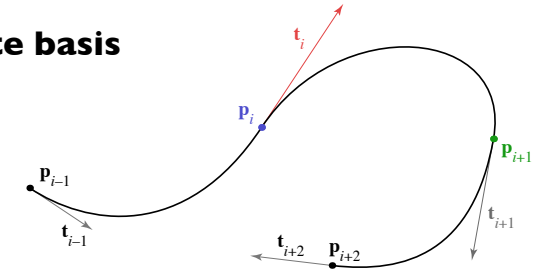
  - can think of them as blending functions glued together
  - this is just like a reconstruction filter!

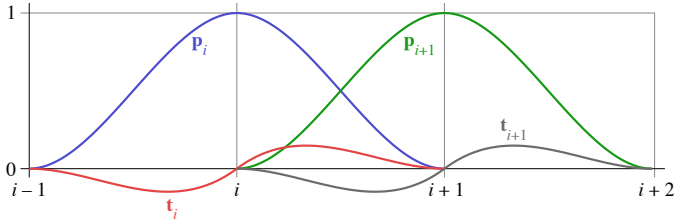
Hermite basis



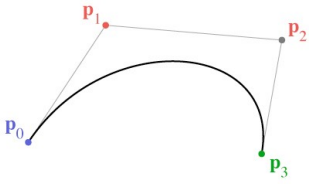


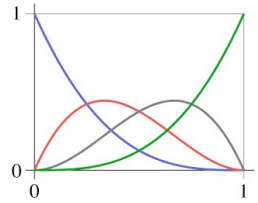
Hermite basis





Bézier basis



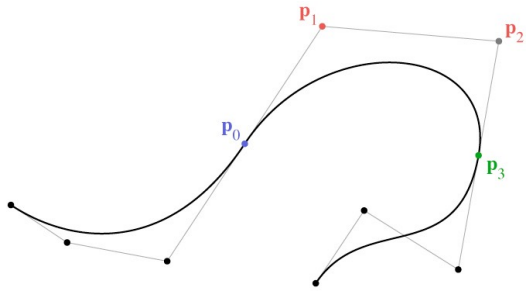


Chaining Bézier splines

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

Chaining Bézier splines

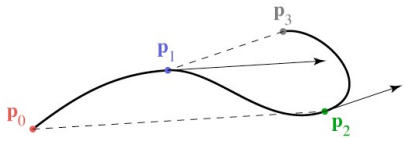
- No continuity built in
- Achieve C^1 using collinear control points

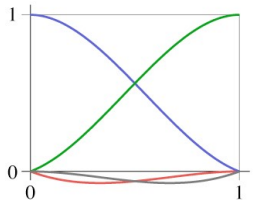


Chaining spline segments

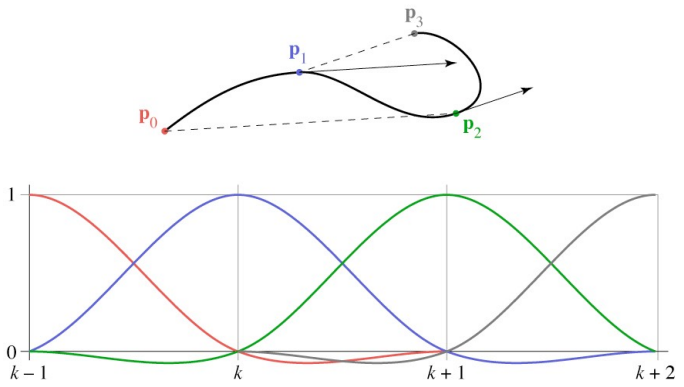
- Hermite curves are convenient because they can be made long easily
- Bézier curves are convenient because their controls are all points
  - but it is fussy to maintain continuity constraints
  - and they interpolate every 3rd point, which is a little odd
- We derived Bézier from Hermite by defining tangents from control points
  - a similar construction leads to the interpolating *Catmull-Rom* spline

Catmull-Rom basis





Catmull-Rom basis



B-splines

B-splines

- We may want more continuity than C^1
- We may not need an interpolating spline
- B-splines are a clean, flexible way of making long splines with arbitrary order of continuity
- Various ways to think of construction
  - a simple one is convolution
  - relationship to sampling and reconstruction

Deriving the B-Spline

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

Efficient construction of any B-spline

- B-splines defined for all orders
  - order *d*: degree *d* – 1
  - order *d*: *d* points contribute to value
- One definition: Cox-deBoor recurrence

$$b_1 = \begin{cases} 1 & 0 \le u < 1 \\ 0 & \text{otherwise} \end{cases}$$

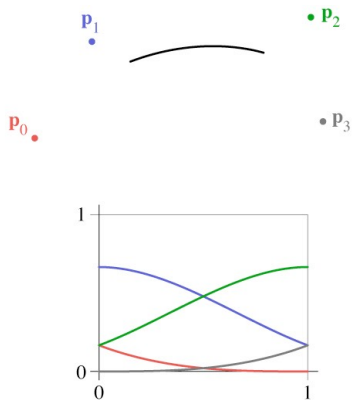
$$b_d = \frac{t}{d-1} b_{d-1}(t) + \frac{d-t}{d-1} b_{d-1}(t-1)$$

Note: This equation is simplified a bit from the complete version with knots

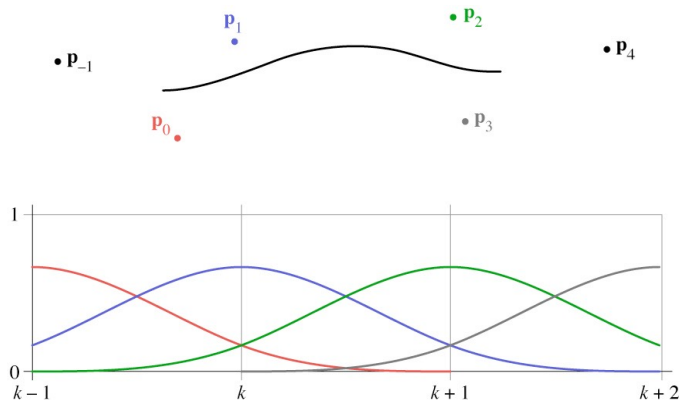
Cubic B-spline matrix

$$\mathbf{f}_i(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \cdot \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{p}_{i-1} \\ \mathbf{p}_i \\ \mathbf{p}_{i+1} \\ \mathbf{p}_{i+2} \end{bmatrix}$$

Cubic B-spline basis



Cubic B-spline basis



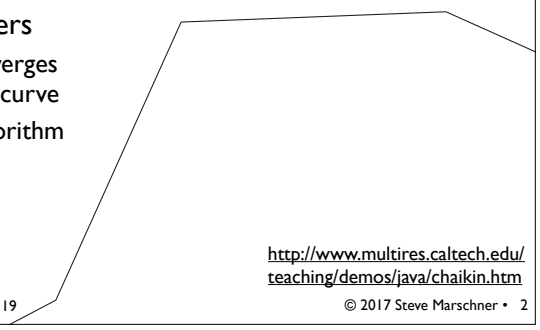
Other types of B-splines

- Nonuniform B-splines
  - discontinuities not evenly spaced
  - allows control over continuity or interpolation at certain points
  - e.g. interpolate endpoints (commonly used case)
- Nonuniform Rational B-splines (NURBS)
  - ratios of nonuniform B-splines: $x(t) / w(t)$; $y(t) / w(t)$
  - key properties:
    - invariance under perspective as well as affine
    - ability to represent conic sections exactly

Corner Cutting

Introduction: corner cutting

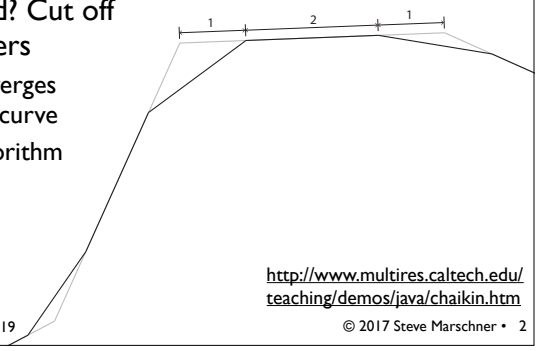
- Piecewise linear curve too jagged for you? Lop off the corners!
  - results in a curve with twice as many corners
- Still too jagged? Cut off the new corners
  - process converges to a smooth curve
  - Chaikin's algorithm



http://www.multires.caltech.edu/teaching/demos/java/chaikin.htm

Introduction: corner cutting

- Piecewise linear curve too jagged for you? Lop off the corners!
  - results in a curve with twice as many corners
- Still too jagged? Cut off the new corners
  - process converges to a smooth curve
  - Chaikin's algorithm



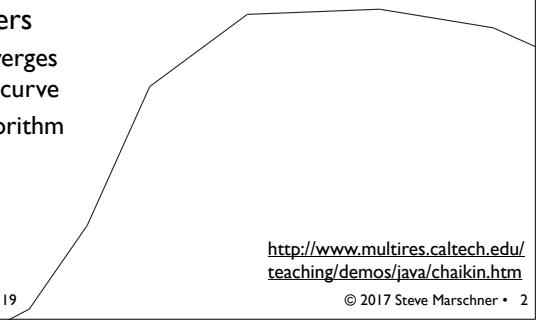
http://www.multires.caltech.edu/teaching/demos/java/chaikin.htm

Cornell CS4620 Spring 2017 • Lecture 19

© 2017 Steve Marschner • 2

Introduction: corner cutting

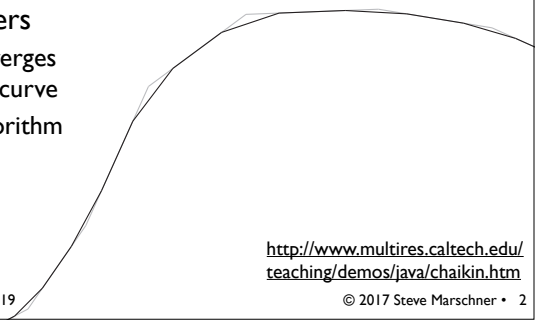
- Piecewise linear curve too jagged for you? Lop off the corners!
  - results in a curve with twice as many corners
- Still too jagged? Cut off the new corners
  - process converges to a smooth curve
  - Chaikin's algorithm



http://www.multires.caltech.edu/teaching/demos/java/chaikin.htm

Introduction: corner cutting

- Piecewise linear curve too jagged for you? Lop off the corners!
  - results in a curve with twice as many corners
- Still too jagged? Cut off the new corners
  - process converges to a smooth curve
  - Chaikin's algorithm

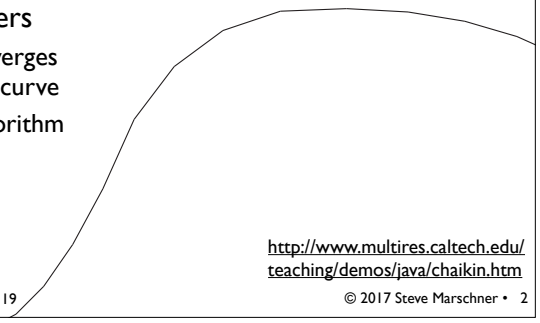


http://www.multires.caltech.edu/teaching/demos/java/chaikin.htm

Cornell CS4620 Spring 2017 • Lecture 19 © 2017 Steve Marschner • 2

Introduction: corner cutting

- Piecewise linear curve too jagged for you? Lop off the corners!
  - results in a curve with twice as many corners
- Still too jagged? Cut off the new corners
  - process converges to a smooth curve
  - Chaikin's algorithm

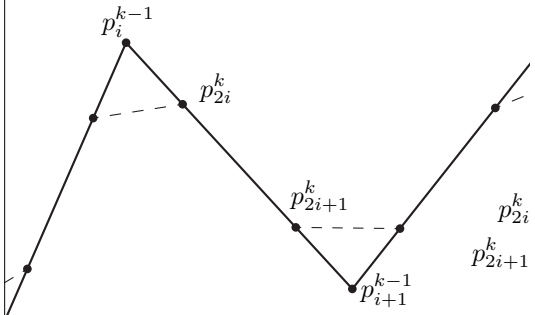


http://www.multires.caltech.edu/teaching/demos/java/chaikin.htm

Cornell CS4620 Spring 2017 • Lecture 19 © 2017 Steve Marschner • 2

Corner cutting in equations

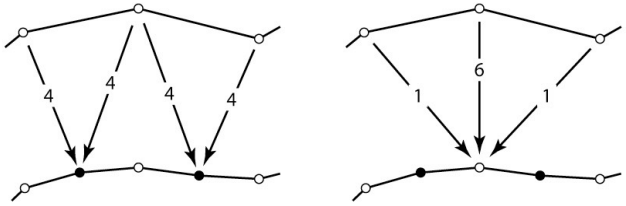
- New points are linear combinations of old ones
- Different treatment for odd-numbered and even-numbered points.



$$p_{2i}^k = (3p_i^{k-1} + p_{i+1}^{k-1})/4$$
$$p_{2i+1}^k = (p_i^{k-1} + 3p_{i+1}^{k-1})/4$$

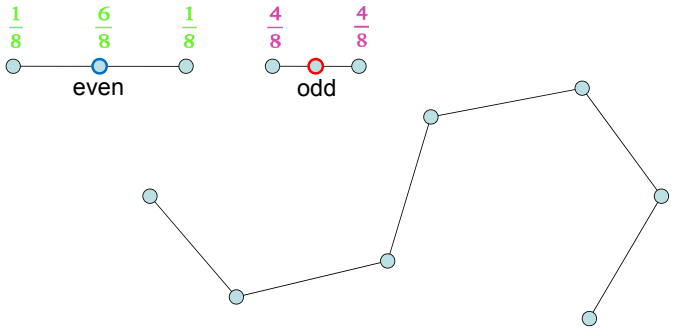
Subdivision for B-splines

- Control vertices of refined spline are linear combinations of the c.v.s of the coarse spline



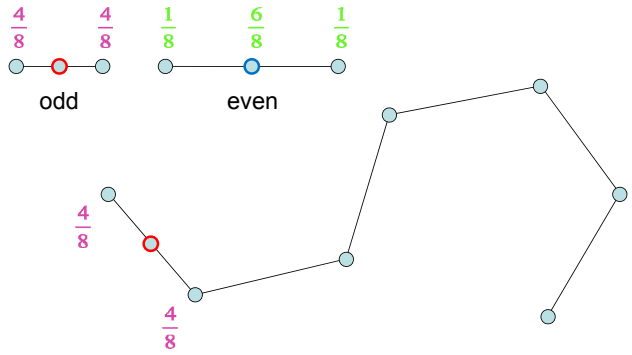
ODD **EVEN**

Cubic B-Spline



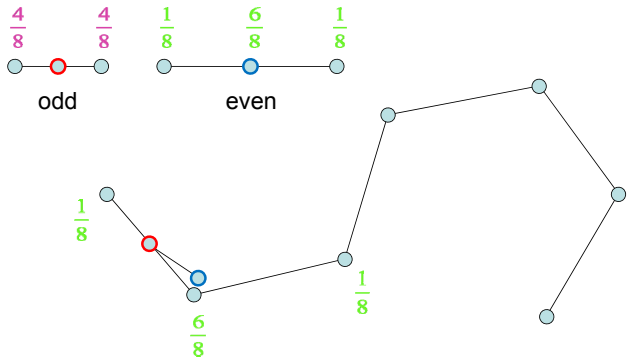
[Stanford CS468 Fall 2010 slides]

Cubic B-Spline



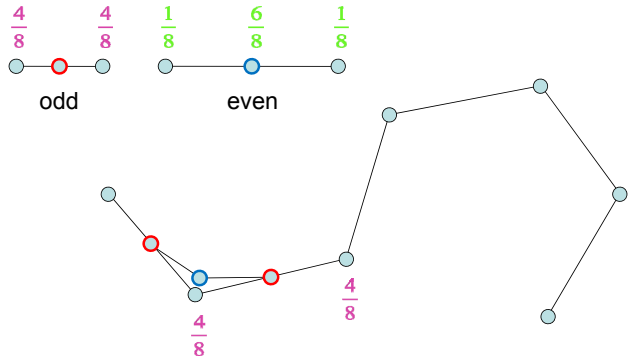
[Stanford CS468 Fall 2010 slides]

Cubic B-Spline



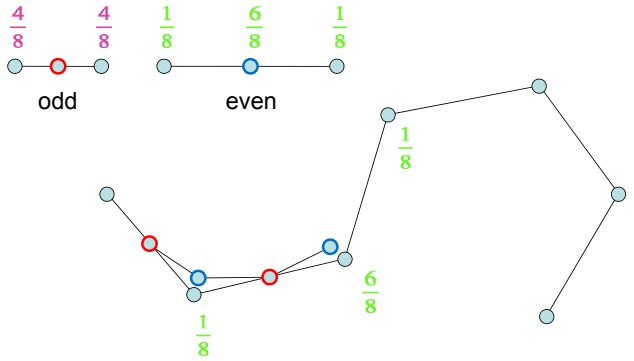
[Stanford CS468 Fall 2010 slides]

Cubic B-Spline



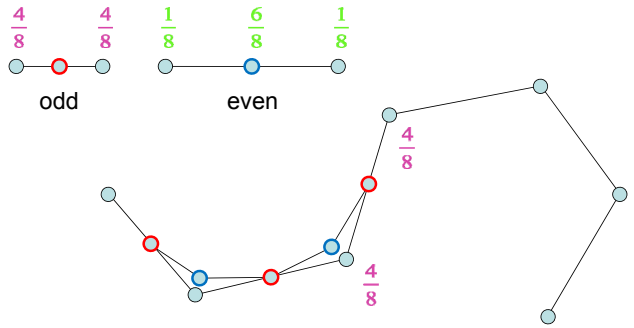
[Stanford CS468 Fall 2010 slides]

Cubic B-Spline



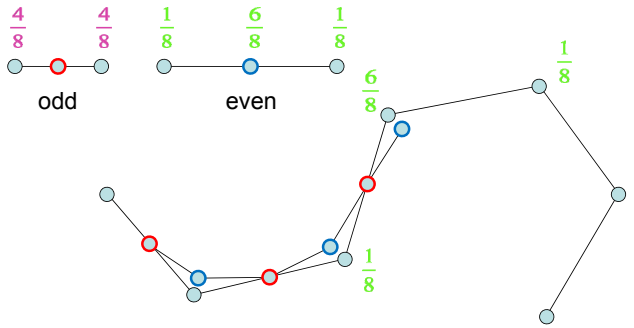
[Stanford CS468 Fall 2010 slides]

Cubic B-Spline



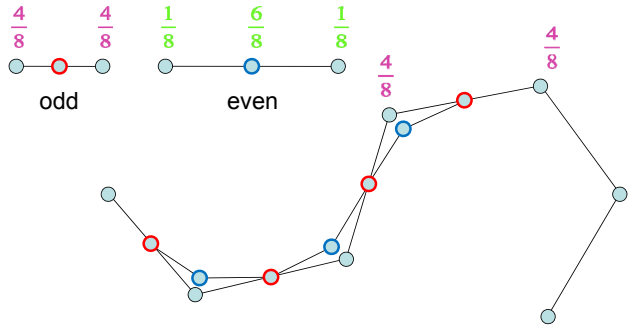
[Stanford CS468 Fall 2010 slides]

Cubic B-Spline



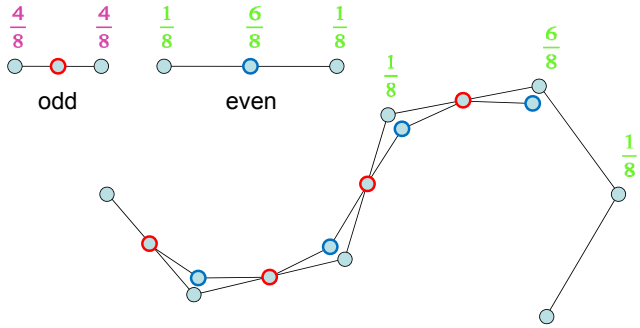
[Stanford CS468 Fall 2010 slides]

Cubic B-Spline



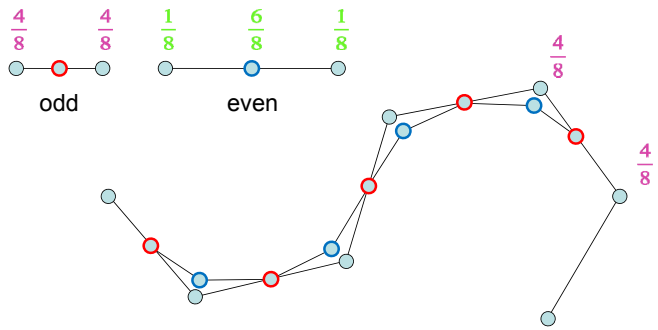
[Stanford CS468 Fall 2010 slides]

Cubic B-Spline



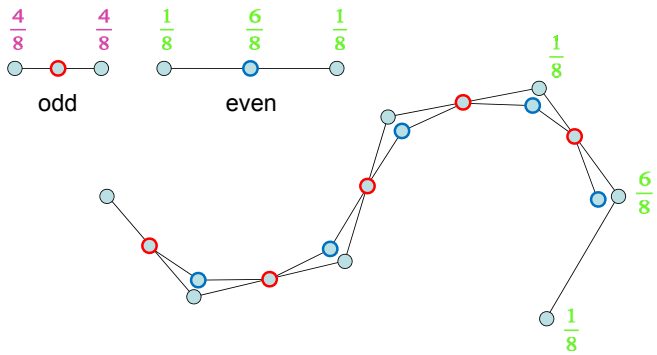
[Stanford CS468 Fall 2010 slides]

Cubic B-Spline



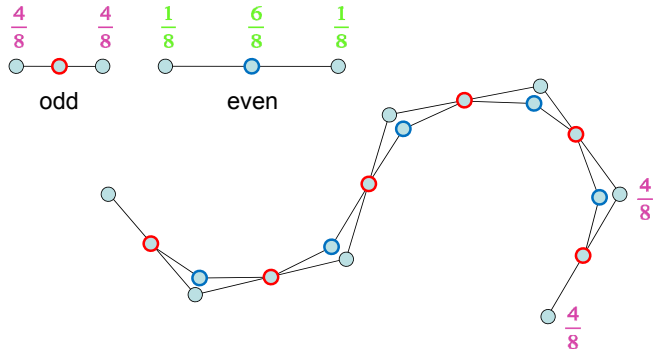
[Stanford CS468 Fall 2010 slides]

Cubic B-Spline



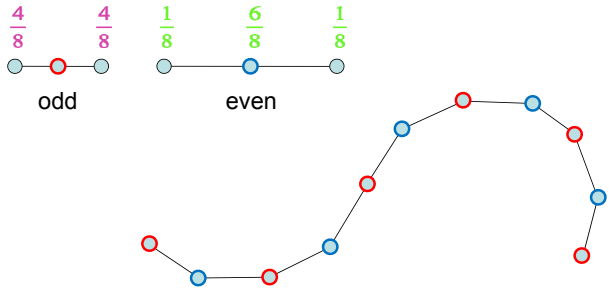
[Stanford CS468 Fall 2010 slides]

Cubic B-Spline



[Stanford CS468 Fall 2010 slides]

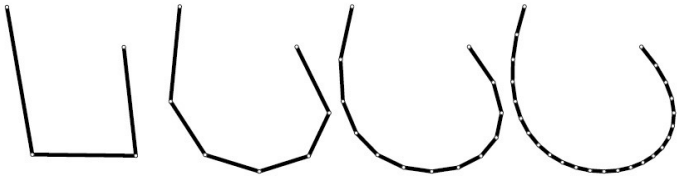
Cubic B-Spline



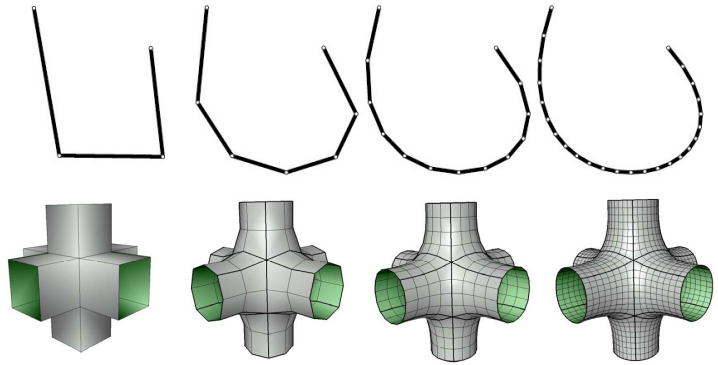
[Stanford CS468 Fall 2010 slides]

Subdivision curves

- Key idea: let go of the polynomials as the definition of the curve, and let the refinement rule define the curve
- Curve is defined as the *limit of a refinement process*
  - properties of curve depend on the rules
  - some rules make polynomial curves, some don't
  - complexity shifts from implementations to proofs



From curves to surfaces



[Schröder & Zorin SIGGRAPH 2000 course 23]

[Stanford CS468 Fall 2010 slides]

Generalizing from curves to surfaces

- Two parts to subdivision process
- Subdividing the mesh (computing new topology)
  - For curves: replace every segment with two segments
  - For surfaces: replace every face with some new faces
- Positioning the vertices (computing new geometry)
  - For curves: two rules (one for *odd* vertices, one for *even*)
    - New vertex's position is a weighted average of positions of old vertices that are nearby along the sequence
  - For surfaces: two kinds of rules (still called odd and even)
    - New vertex's position is a weighted average of positions of old vertices that are nearby in the mesh

Lec28 Required Reading

- FOCG, Ch. 22

Reminder: Assignment 07

Assigned: Wednesday, Nov. 27
Written Due: Monday, Dec. 9, 4:59:59 pm
Program Due: Wednesday, Dec. 11, 4:59:59 pm