Geometric Modeling

Summer Semester 2012

Polynomial Spline Curves

Piecewise Polynomials · Splines Bases · Properties

UNIVERSITÄT DES SAARLANDES

CLUSTER OF EXCELLENCE

max planck institut informatik

Announcements

HISPOS Registration

- *Important:* Hispos registration is now open
- Do not forget to register for geometric modeling

Room Change

- On *Tuesday, June 26th*,
  the lecture will be held in
  Buiding E1 7 (Cluster MMCI Building), *Room 0.02*

Exam Topics

- Section "*Spectral Graph Methods for 3D Shape Analysis*"
  is not relevant for the exam.

Today...

Topics:

- Introduction: Geometric Modeling
- Mathematical Background
- Interpolation & Approximation
- Splines
  - Polynomial Spline Curves
  - Blossoming and Polar Forms
  - Rational Splines
  - Spline Surfaces
- Meshes

Today...

Topics:

- Introduction: Geometric Modeling
- Mathematical Background
- Interpolation & Approximation
- Splines
  - Polynomial Spline Curves
  - Blossoming and Polar Forms
  - Rational Splines
  - Spline Surfaces
- Meshes

Today

Polynomial Spline Curves

- Piecewise cubic interpolation
- Splines with local control
  - Hermite Splines
  - Bezier Splines
  - Non-Uniform B-Splines
  - Uniform B-Splines

Polynomial Spline Curves

Piecewise Cubic Interpolation

What we have so far...

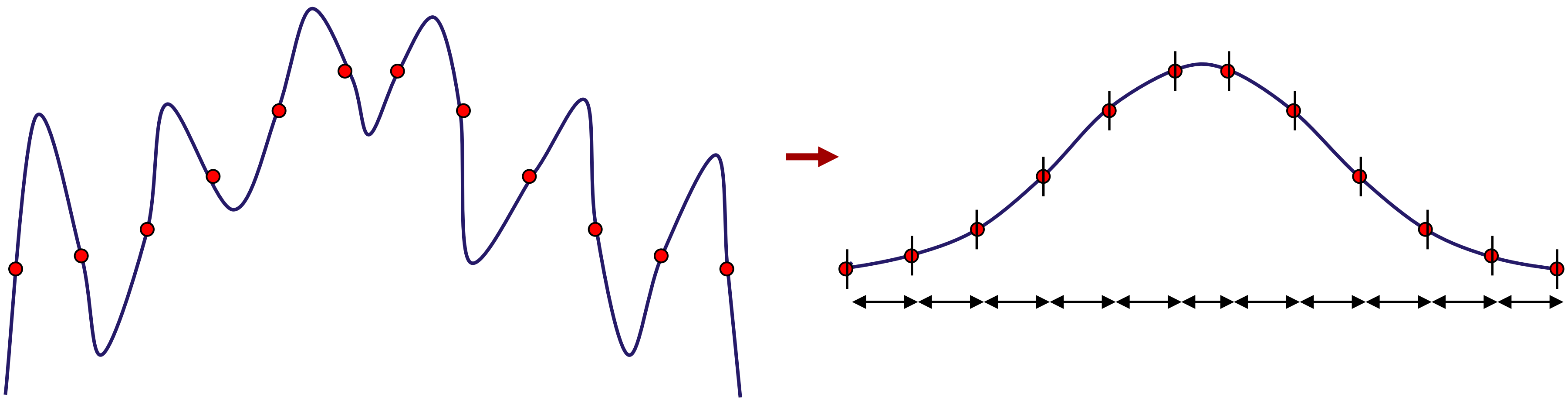
What we have so far:

- Given a basis, we can interpolate and approximate points
  - Curves, surfaces, higher dimensional objects
  - Functions (heightfields) and parametric objects
  - Differential properties can be prescribed as well

Problem:

- We need a suitable basis
- Polynomial bases don't work for large degree (say ≥ 10)
  - Monomials – numerical nightmare
  - Orthogonal polynomials: Runge's phenomenon still limits applicability

Piecewise Polynomials



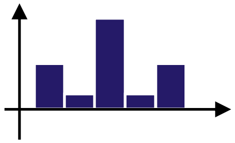
Key Idea:

- Polynomials of high degree don't work
- Therefore: Use piecewise polynomials of low degree
- What is a good degree to use?

Choosing the Degree...

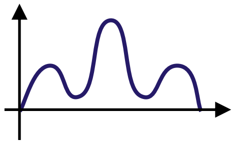
Candidates:

- *d* = 0 (piecewise constant): not smooth

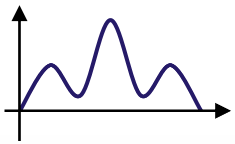


- *d* = 1 (piecewise linear): not smooth enough

- *d* = 2 (piecewise quadratic): constant 2nd derivative, still too inflexible



- *d* = 3 (piecewise cubic): degree of choice for computer graphics applications

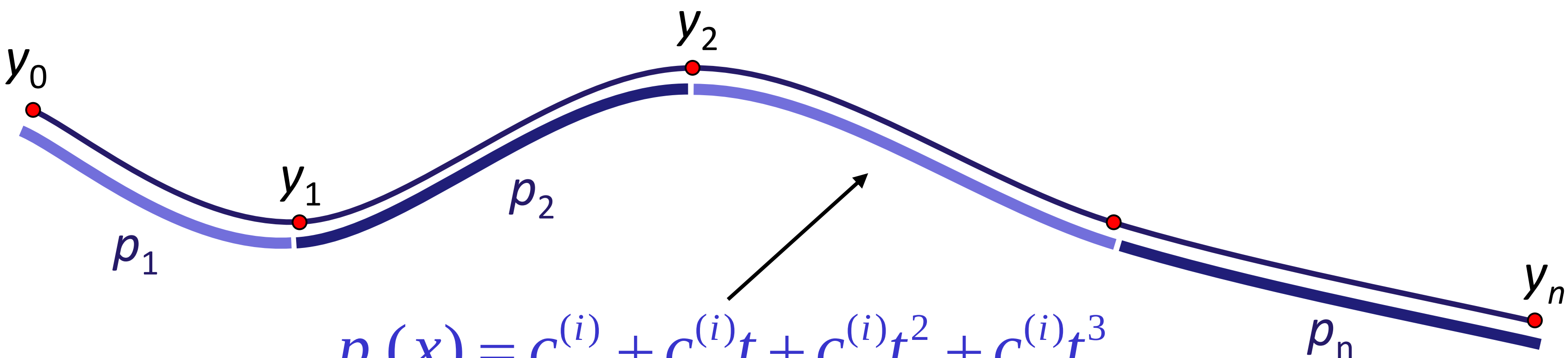


Cubic Splines

Cubic piecewise polynomials:

- We can attain C^2 continuity without fixing the second derivative throughout the curve
- C^2 continuity is perceptually important
  - We can see second order shading discontinuities (esp.: reflective objects)
  - Motion: continuous *position*, *velocity* & *acceleration*
    Discontinuous acceleration noticeable (object/camera motion)
- One more argument for cubics:
  - Among all C^2 curves that interpolate a set of points (and obey to the same end conditions), a piecewise cubic curve has the least integral acceleration (“smoothest curve you can get”).

Piecewise Cubic Interpolation

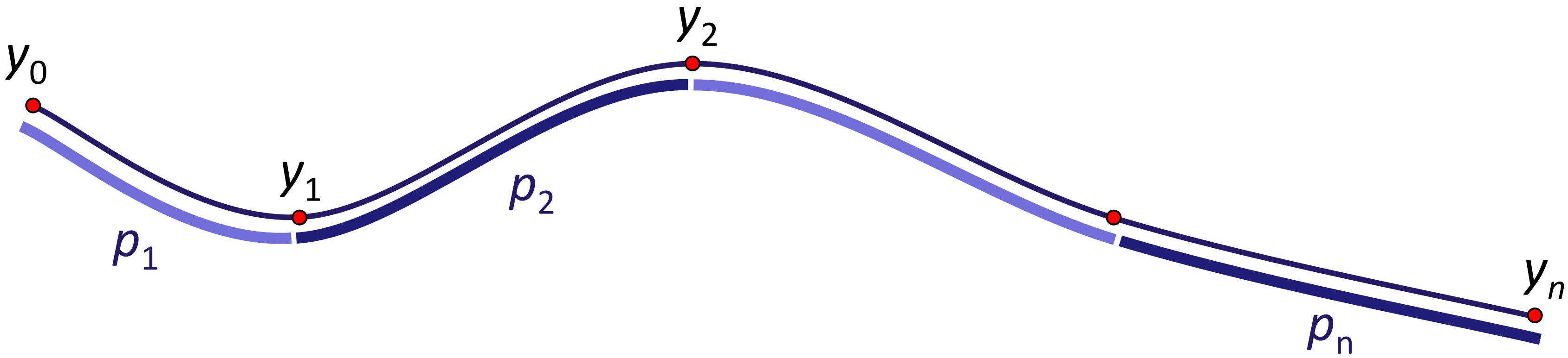


$$p_i(x) = c_0^{(i)} + c_1^{(i)} t + c_2^{(i)} t^2 + c_3^{(i)} t^3$$

Setup:

- (n+1) control points $y_0 \ldots y_n$ (to be interpolated)
- For simplicity: assume uniform spacing $t_0 \ldots t_n = (0,1,2,\ldots,n)$
- n cubic polynomial pieces $p_1 \ldots p_n$ parametrized over [0...1]
- Multidimensional case: solve problem for each axis (x,y,z)

Conditions



(4n degrees of freedom)

$\forall i = 1 \ldots n : p_i(0) = y_{i-1}$ (*n* conditions)

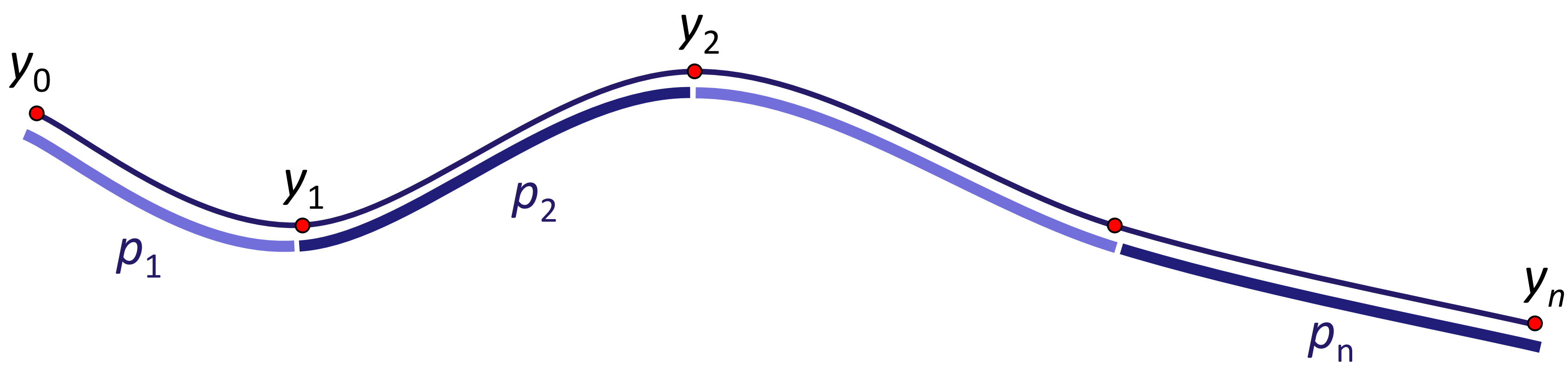
$\forall i = 1 \ldots n : p_i(1) = y_i$ (*n* conditions)

$\forall i = 2 \ldots n : \frac{d}{dt} p_i(0) = \frac{d}{dt} p_{i-1}(1)$ (*n*-1 conditions)

$\forall i = 2 \ldots n : \frac{d^2}{dt^2} p_i(0) = \frac{d^2}{dt^2} p_{i-1}(1)$ (*n*-1 conditions)

2 dimensional null space (so far)

Conditions



$\forall i = 1 \ldots n : p_i(0) = y_{i-1}$ (*n* conditions)

$\forall i = 1 \ldots n : p_i(1) = y_i$ (*n* conditions)

$\forall i = 2 \ldots n : \frac{d}{dt} p_i(0) = \frac{d}{dt} p_{i-1}(1)$ (*n*-1 conditions)

$\forall i = 2 \ldots n : \frac{d^2}{dt^2} p_i(0) = \frac{d^2}{dt^2} p_{i-1}(1)$ (*n*-1 conditions)

additional:

$$\frac{d^2}{dt^2} p_1(0) = 0$$

$$\frac{d^2}{dt^2} p_n(1) = 0$$

alternative:

cyclic boundary conditions
(closed curves)

Numerical Solution

Solving the system of equations:

- Band matrix, bandwidth O(1)
- Can be solved in O(*n*) time & space for *n* variables

$$\begin{pmatrix} * & * & * & & & & & \\ * & * & * & * & & & & \\ * & * & * & * & * & & & \\ & * & * & * & * & * & & \\ & & * & * & * & * & * & \\ & & & * & * & * & * & * \\ & & & & * & * & * & * \\ & & & & & * & * & * \end{pmatrix}$$

Cubics Minimize Acceleration

Theorem:

- Given n data points (y_i, t_i) to interpolate and fixed end conditions (either prescribed 1st derivative, or zero second derivative), a piecewise cubic interpolant minimizes the energy: $E(f)=\int_0^n f''(t)^2\,dt$
- This means: A cubic spline curve has the least square acceleration.
- Related to elastic energy: Hooke's elastic energy of a straight line is given by: $E(\mathbf{f})=\int_0^n \lambda \left\| \boldsymbol{\kappa}2[\mathbf{f}](t) \right\|^2 dt$
- I.e.: cubic spline interpolation approximates elastic beams.

Proof: Cubics Minimize Acceleration

Cubic spline: $c(t)$

Another C^2 interpolating curve: $a(t)$

Residual: $d(t) = a(t) - c(t)$.

Energy functional:

$$E(a) = \int_0^n a''(t)^2\,dt$$

$$= \int_0^n \left[c''(t) + d''(t)\right]^2 dt$$

$$= \int_0^n c''(t)^2\,dt + 2\int_0^n c''(t)d''(t)\,dt + \int_0^n d''(t)^2\,dt$$

Proof: Cubics Minimize Acceleration

Cubic spline: $c(t)$

Another C^2 interpolating curve: $a(t)$

Residual: $d(t) = a(t) - c(t)$.

Integration by parts:

$$\int_a^b a(t)b'(t)dt = \left[a(t)b(t)\right]_{t=a}^{t=b} - \int_a^b a'(x)b(x)dt$$

Energy functional:

$$E(a) = \int_0^n c''(t)^2 dt + 2\int_0^n c''(t)d''(t)dt + \int_0^n d''(t)^2 dt$$

Integration by parts:

$$\int_0^n c''(t)d''(t)dt = \left[c''(t)d'(t)\right]_0^n - \int_0^n c'''(t)d'(t)dt$$

$$= \left[c''(t)\left(a'(t) - c'(t)\right)\right]_{t=0}^{t=n} - \int_0^n c'''(t)d'(t)dt$$

$$= \underbrace{c''(n)}_{\substack{0 \text{ for} \\ c''(n)=0}}\underbrace{\left(a'(n) - c'(n)\right)}_{\substack{0 \text{ if identical first} \\ \text{order end cond.}}} - \underbrace{c''(0)}_{\substack{0 \text{ for} \\ c''(n)=0}}\underbrace{\left(a'(0) - c'(0)\right)}_{\substack{0 \text{ if identical first} \\ \text{order end cond.}}} - \int_0^n c'''(t)d'(t)dt$$

Proof: Cubics Minimize Acceleration

Cubic spline: $c(t)$

Another C^2 interpolating curve: $a(t)$

Residual: $d(t) = a(t) - c(t)$.

Integration by parts:

$$\int_a^b a(t)b'(t)dt = \left[a(t)b(t)\right]_{t=a}^{t=b} - \int_a^b a'(x)b(x)dt$$

Energy functional:

$$E(a) = \int_0^n c''(t)^2 dt + 2\int_0^n c''(t)d''(t)dt + \int_0^n d''(t)^2 dt$$

Middle term (cont.):

$$\int_0^n c''(t)d''(t)dt = \int_0^n \underbrace{c'''(t)}_{\text{piecewise const.}} d'(t)dt$$

$$= \sum_{i=0}^{n-1} c'''(i+0.5) \underbrace{\left[d(t)\right]_{t=i}^{t=i+1}}_{=0\,(\text{interpolation})}$$

$$= 0$$

Proof: Cubics Minimize Acceleration

Cubic spline: $c(t)$

Another C^2 interpolating curve: $a(t)$

Residual: $d(t) = a(t) - c(t)$.

Energy functional:

$$E(a) = \underbrace{\int_0^n c''(t)^2\,dt}_{\text{cubic spline}} + \underbrace{\int_0^n d''(t)^2\,dt}_{\text{additional energy: positive}}$$

Positive additional energy:

Any function that differs in second derivative from *c* will have higher energy.

$\Rightarrow$ *c* is a minimal function in terms of *E*.

Polynomial Spline Curves

Spline Bases with Local Control

So what's missing?

So we have solved our problem – what's left to do?

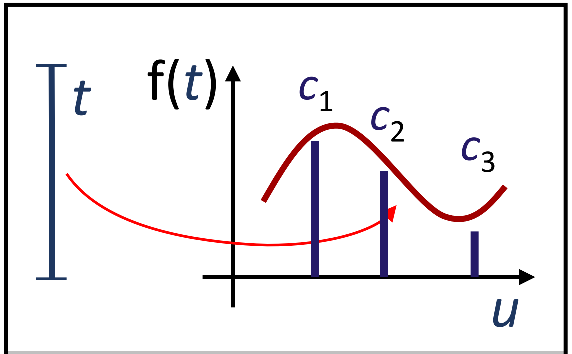
- Target area: *interactive geometric modeling*
- Shape of the entire curve depends on all control points
- Changing one control point can affect the whole curve
- Not a big issue for algorithmic curve control
  - Fitting curves to data, optimizing curves according to some objective function, etc...
- But not acceptable for modeling by humans
- "User interface problem": We want *local control*.

Notation

Function design problem:

- Function $f: \mathbb{R} \to \mathbb{R}$

  $f(t) = c_1 b_1(t) + c_2 b_2(t) + c_3 b_3(t) + \ldots$
- Coefficients $c_1, c_2, c_3, \ldots \in \mathbb{R}$

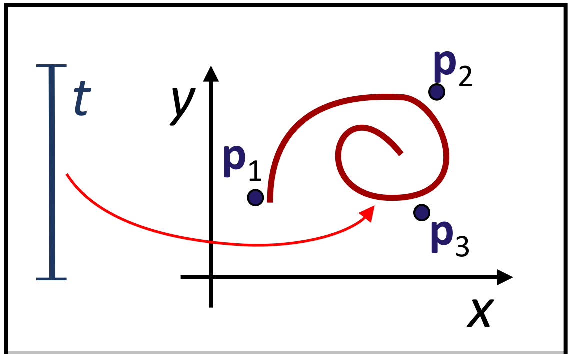


function graph

Curve design problem

- Function $\mathbf{f}: \mathbb{R} \to \mathbb{R}^n$

  $\mathbf{f}(t) = b_1(t)\mathbf{p}_1 + b_2(t)\mathbf{p}_2 + b_3(t)\mathbf{p}_3 + \ldots$
- “Control points” $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \ldots \in \mathbb{R}^n$



parametric curve

New Idea

Problem: Again the basis...

- We want a basis such that:
  - Coefficients / control points have intuitive meaning
- Let the user edit the control points
  - Intuitive response
  - Manually controllable

Desirable Properties

Useful requirements for a spline basis:

- Well behaved curve
  - Smooth basis functions
- Local control
  - Basis functions with compact support
- Affine invariance:
  - Applying an affine map $\mathbf{x} \rightarrow \mathbf{A}\mathbf{x} + \mathbf{b}$ on
    - control points
    - curve

    should have the same effect
  - In particular: rotation, translation
  - Otherwise: interactive curve editing very difficult

Desirable Properties

Useful requirements for a spline basis:

- Convex hull property:
  - The curve is in the convex hull of its control points
  - Avoids at least too weird oscillations
- Advantages
  - Computational advantages (recursive intersection tests)
  - More predictable behavior

Summary

Useful properties

- Smoothness
- Local control / support
- Affine invariance
- Convex hull property

Affine Invariance

Affine Invariance:

- Affine map: $\mathbf{x} \rightarrow \mathbf{A}\mathbf{x} + \mathbf{b}$
- **Part I:** Linear invariance – we get this automatically
  - Linear approach: $\mathbf{f}(t) = \sum_{i=1}^{n} b_i(t)\mathbf{p}_i = \sum_{i=1}^{n} b_i(t) \begin{pmatrix} p_i^{(x)} \\ p_i^{(y)} \\ p_i^{(z)} \end{pmatrix}$
  - Therefore: $\mathbf{A}(\mathbf{f}(t)) = \mathbf{A}\left(\sum_{i=1}^{n} b_i(t)\mathbf{p}_i\right) = \sum_{i=1}^{n} b_i(t)(\mathbf{A}\mathbf{p}_i)$

Affine Invariance

Affine Invariance:

- Affine map: **x** → **Ax** + **b**
- **Part II:** Translational invariance – need some brains
  - $$\sum_{i=1}^{n} b_i(t)(\mathbf{p}_i + \mathbf{b}) = \sum_{i=1}^{n} b_i(t)\mathbf{p}_i + \sum_{i=1}^{n} b_i(t)\mathbf{b} = f(t) + \underbrace{\left(\sum_{i=1}^{n} b_i(t)\right)}_{\text{must sum to one}}\mathbf{b}$$
  - For translational invariance, the sum of the basis functions must be one *everywhere* (for all parameter values *t* that are used).
  - This is called "*partition of unity property*".
  - The $\mathbf{b}_i$ form an "*affine combination*" of the control points $\mathbf{p}_i$.
  - This is very important for human modeling.

Convex Hull Property

Convex combinations:

- A convex combination of a set of points $\{\mathbf{p}_1,...,\mathbf{p}_n\}$ is any point of the form:
  - $\sum_{i=1}^{n} \lambda_i \mathbf{p}_i$ with: $\sum_{i=1}^{n} \lambda_i = 1$ and $\forall i = 1..n: \lambda_i \geq 0, \lambda_i \leq 1$
  - (Remark: $\lambda_i \leq 1$ is redundant)
- The set of all admissible convex combinations forms the convex hull of the point set
  - Easy to see (simple exercise): This convex hull is the smallest set that contains all points $\{\mathbf{p}_1,...,\mathbf{p}_n\}$ and every complete straight line between two elements of the set.

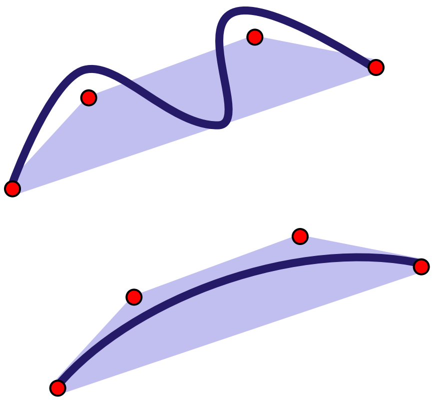
Convex Hull Property

Accordingly:

- If we have this property:

  $$\forall t \in \Omega : \sum_{i=1}^{n} b_i(t) = 1 \ \text{ and } \forall t \in \Omega : \forall i = 1..n : b_i(t) \geq 0$$

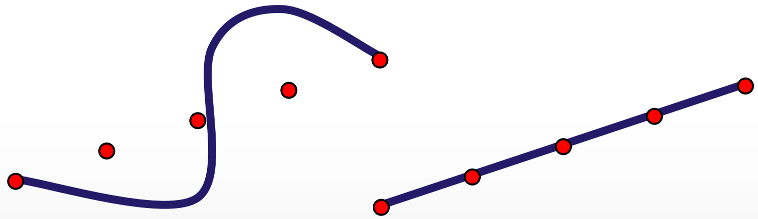
  the constructed curves / surfaces will be:
  - Affine invariant (translations, linear maps)
  - Be restricted to the convex hull of the control points
- Corollary: Curves with this property will have *linear precision*, i.e.: if all control points lie on a straight line, the curve is a straight line segment, too.
- Surfaces with planar control points will be flat, too.

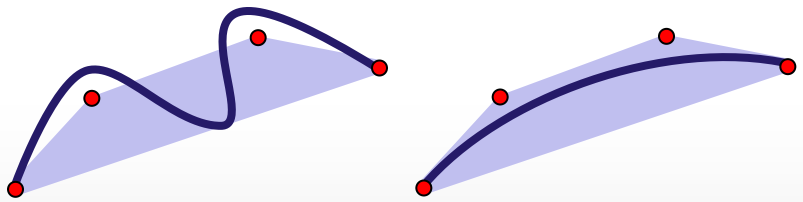


Convex Hull Property

Convex Hull Property:

- Very useful property
  - Avoids at least the worst oscillations (no escape from convex hull, unlike polynomial interpolation through control points)
  - Linear precision property is intuitive (people expect this)
  - Can be used fast range checks
    - Test for intersection with convex hull first, then the object.
    - Recursive intersection algorithms in conjunction with subdivision rules (more on this later)





Spline Techniques

Spline bases we will look at in this lecture:

- Hermite interpolation
- Bezier curves & surfaces
- Uniform B-splines
- Non-uniform B-splines
- [NURBS: Non-uniform rational B-splines]
  (not linear, more on this later)

Spline Techniques

Two views:

- Linear algebra: polynomial function spaces
  - Basis changes
  - Derivatives and continuity conditions
- Geometry: Successive linear interpolation (“blossoming”)
  - Construct polynomial spline curves by repeated linear interpolation of control points
  - More intuitive explanations of properties
  - Mathematical formalism: Blossoming and polar forms

This part of the lecture will deal with the linear algebra view.
Blossoming gets a separate chapter...

Polynomial Spline Curves

Hermite Splines

Hermite Splines

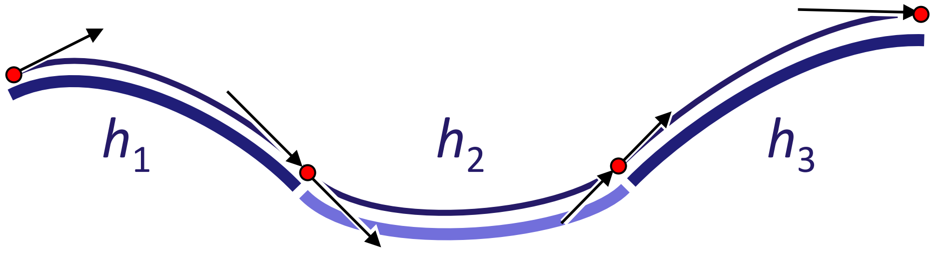
Overview:

- Simple spline technique, easy to implement
- Has some shortcomings
- We will look at C^1 cubic Hermite splines as an example

Key Idea:

- Specify position and derivatives at the endpoints of each segment
- Come up with a rule to match them easily
- Precompute basis for this purpose

Illustration



$$h_i(t) = c_0^{(i)} + c_1^{(i)}t + c_2^{(i)}t^2 + c_3^{(i)}t^3$$

For each segment $\mathbf{h}_i(t)$ we know:

- Positions: $\mathbf{h}_i(0)$, $\mathbf{h}_i(1)$
- Derivatives: $\partial_t \mathbf{h}_i(0)$, $\partial_t \mathbf{h}_i(1)$

Hermite Basis

Linear system: (one dimension, one segment)

$$h(t) = c_0 + c_1 t + c_2 t^2 + c_3 t^3$$

$$h'(t) = c_1 + 2c_2 t + 3c_3 t^2$$

$$h(0) = p_0 \quad \Rightarrow \quad c_0 = p_0$$

$$h(1) = p_1 \quad \Rightarrow \quad c_0 + c_1 + c_2 + c_3 = p_1$$

$$h'(0) = m_0 \quad \Rightarrow \quad c_1 = m_0$$

$$h'(1) = m_1 \quad \Rightarrow \quad c_1 + 2c_2 + 3c_3 = m_1$$

$$\rightarrow \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 3 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} p_0 \\ p_1 \\ m_0 \\ m_1 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 3 & -2 & -1 \\ 2 & -2 & 1 & 1 \end{pmatrix} \begin{pmatrix} p_0 \\ p_1 \\ m_0 \\ m_1 \end{pmatrix} = \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix}$$

Hermite Basis

Solution:

$$\mathbf{f}(t) = [1, t, t^2, t^3] \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 3 & -2 & -1 \\ 2 & -2 & 1 & 1 \end{pmatrix} \begin{pmatrix} p_0 \\ p_1 \\ m_0 \\ m_1 \end{pmatrix}$$

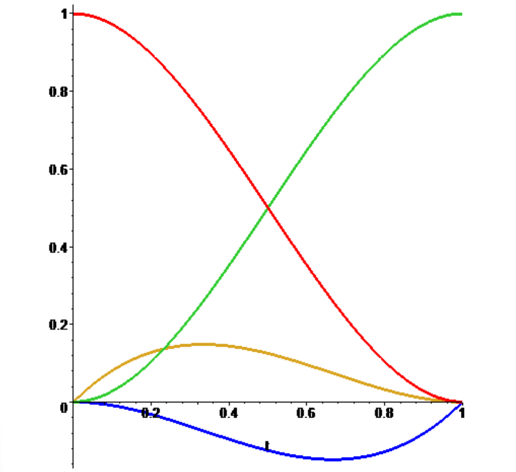
Basis Functions:

$$h_{p_0}(t) = 1 - 3t^2 + 2t^3$$

$$h_{p1}(t) = 3t^2 - 2t^3$$

$$h_{m_0}(t) = t - 2t^2 + t^3$$

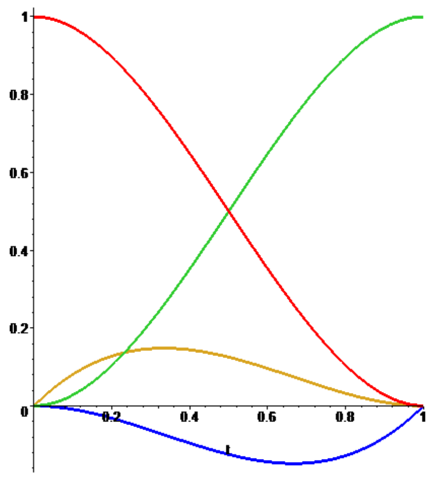
$$h_{m_1}(t) = -t^2 + t^3$$



Hermite Basis

Properties:

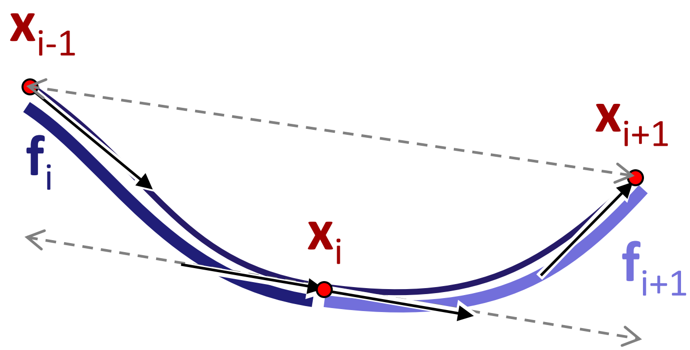
- h_{p_0} and h_{p_1} sum to one.
  (affine invariant w.r.t. position)
- Curve might leave convex hull of control points



Open question:

- How to specify derivatives?

Illustration



Simple rule for derivatives:

- Derivatives:

$$\partial_t \mathbf{f}_i(0) := \frac{\mathbf{x}_i - \mathbf{x}_{i-2}}{2}, \quad (i \in \{2..n\})$$

$$\partial_t \mathbf{f}_i(1) := \frac{\mathbf{x}_{i+1} - \mathbf{x}_{i-1}}{2}, \quad (i \in \{1..n-1\})$$

$$\partial_t \mathbf{f}_1(0) := \mathbf{x}_1 - \mathbf{x}_0$$

$$\partial_t \mathbf{f}_n(1) := \mathbf{x}_n - \mathbf{x}_{n-1}$$

- “Catmull-Rom Spline”

Properties

Properties of this spline construction:

- Interpolates original points
- Local control
- C_1 continuous
- Affine invariant
- No convex hull property
  - Tends to “overshoot”
  - This can be really nasty in practice

Polynomial Spline Curves

Bezier Curves

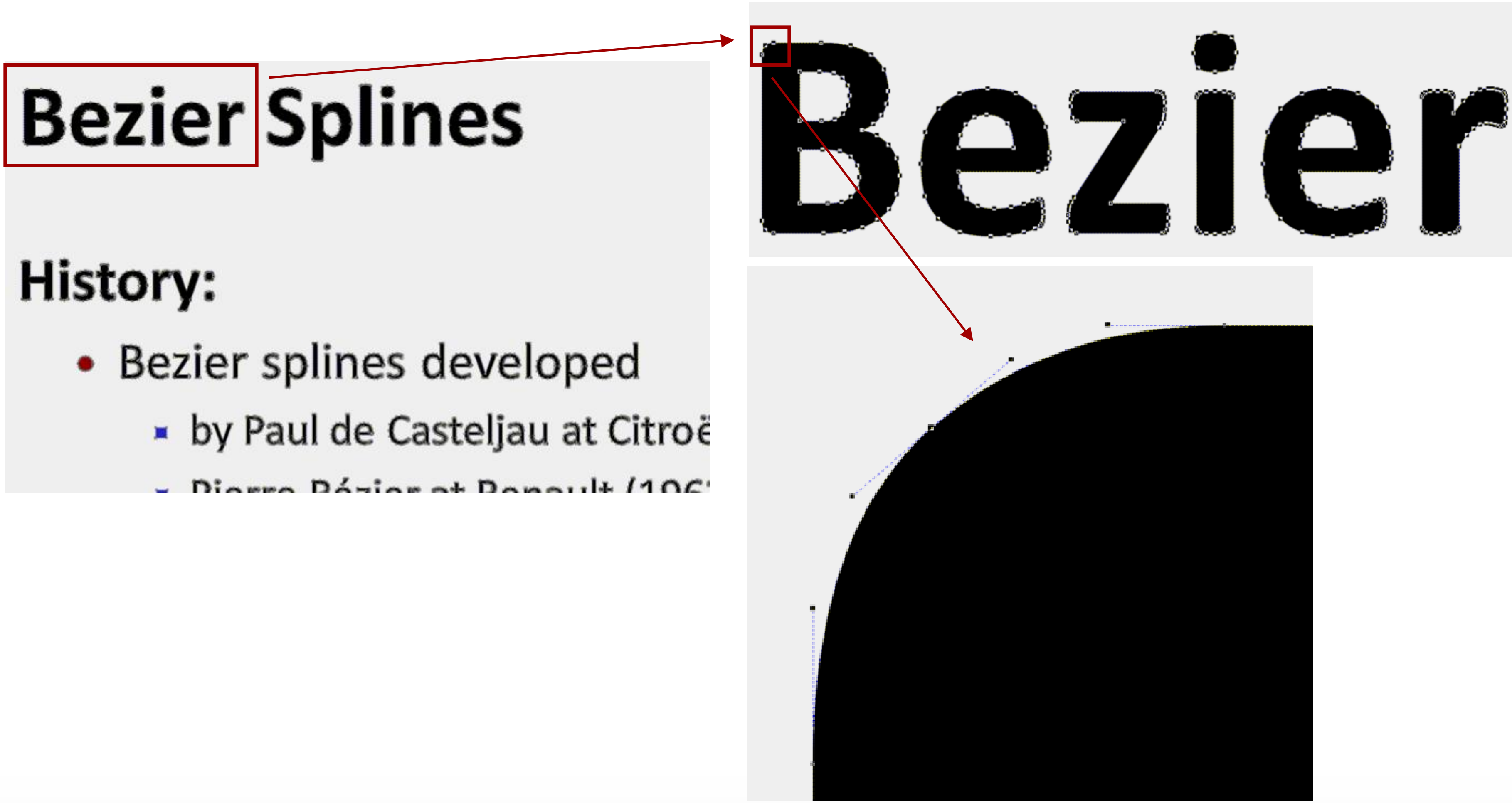
Bezier Splines

History:

- Bezier splines developed
  - by Paul de Casteljau at Citroën (1959)
  - Pierre Bézier at Renault (1962)

  for designing smooth free-form parts in automotive design applications.
- Today: The standard tool for 2D curve editing,
  Cubic 2D Bezier curves are used almost everywhere:
  - Postscript, PDF, Truetype (quadratic curves), Windows GDI...
  - Corel Draw, Powerpoint, Illustrator, ...
- Widely used in 3D curve & surface modeling as well

All You See is Bezier Curves...



Bernstein Basis

Bezier splines use the Bernstein basis:

- Bernstein basis of degree *n*: $B = \left\{B_0^{(n)}, B_1^{(n)}, \ldots, B_n^{(n)}\right\}$

$$B_i^{(n)}(t) := \binom{n}{i} t^i (1-t)^{n-i} = B_{i-th\text{ basis function}}^{(\text{degree}\,n)}$$

- Each basis function is a polynomial of degree *n*.
- The basis functions form a partition of unity

$$1 = (1-t+t) = (t+(1-t))^n = \sum_{i=0}^{n} \binom{n}{i} t^i (1-t)^{n-i} = \sum_{i=0}^{n} B_i^{(n)}(t)$$

(binomial theorem)

- For $t \in [0..1]$, the basis functions are positive ($B_i^{(n)}(t) \geq 0$).

Examples

The first three Bernstein bases:

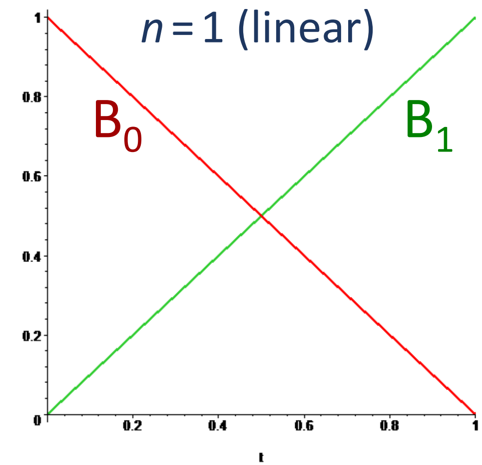
$$B_0^{(0)} := 1$$

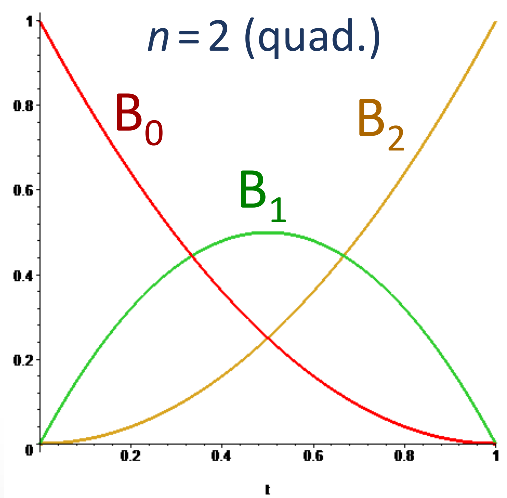
$$B_0^{(1)} := (1-t) \qquad B_1^{(1)} := t$$

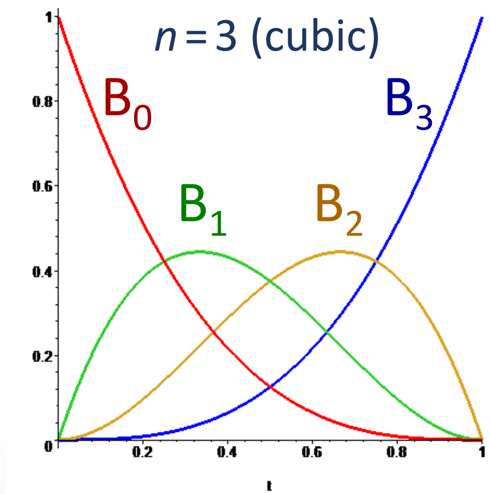
$$B_0^{(2)} := (1-t)^2 \qquad B_1^{(2)} := 2t(1-t) \qquad B_2^{(2)} := t^2$$

$$B_0^{(3)} := (1-t)^3 \qquad B_1^{(3)} := 3t(1-t)^2$$

$$B_2^{(3)} := 3t^2(1-t) \qquad B_3^{(3)} := t^3$$



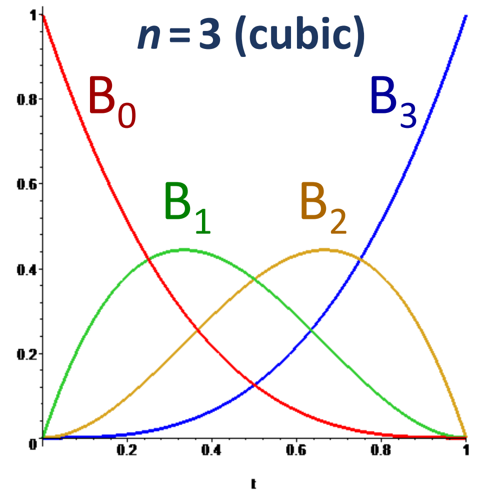


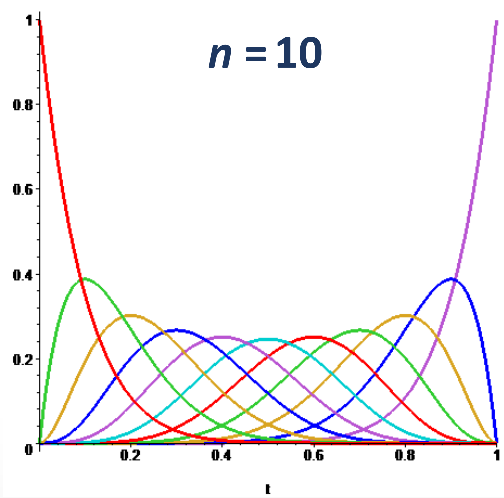


Bernstein Basis

Bernstein basis properties:

- $B = \left\{B_0^{(n)}, B_1^{(n)}, \ldots, B_n^{(n)}\right\}, B_i^{(n)}(t) := \binom{n}{i} t^i (1-t)^{n-i}$
- Basis for polynomials of degree *n*.
- Each basis function $B_i^{(n)}$ has its maximum at i/n.
- Recursive computation:
  $B_i^{(n)}(t) := (1-t) B_i^{(n-1)}(t) + t\, B_{i-1}^{(n-1)}(t)$
  with $B_0^0(t) = 1, B_i^n(t) = 0 \text{ for } i \notin \{0 \ldots n\}$
- Symmetry: $B_i^n(t) = B_{n-i}^n(1-t)$

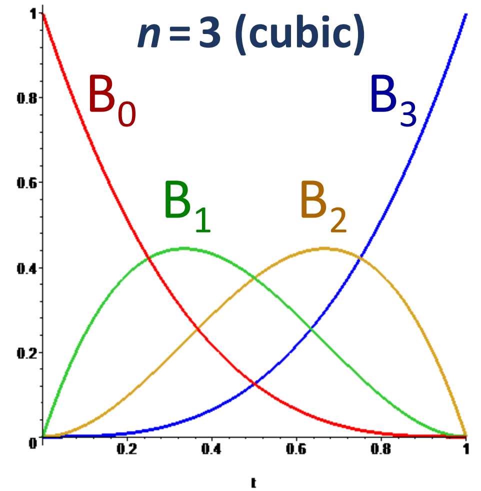


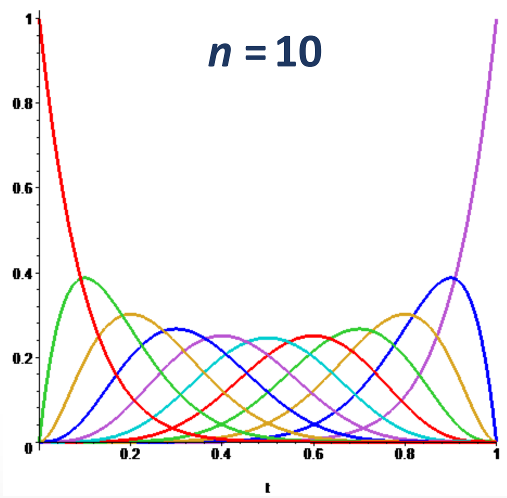


Bezier Curves

Bezier curves Properties:

- Curves: $\mathbf{f}(t) = \sum_{i=0}^{n} \mathbf{p}_i B_i^{(n)}(t)$
- Considering the interval $t \in [0..1]$
- Bezier curves are affine invariant.
- Bezier curves are contained in the convex hull of the control points.
- The influence of the control points is moving along the curve with index *i*. Largest influence at $t = i/n$.
- However: A single curve segment has no fully local control.

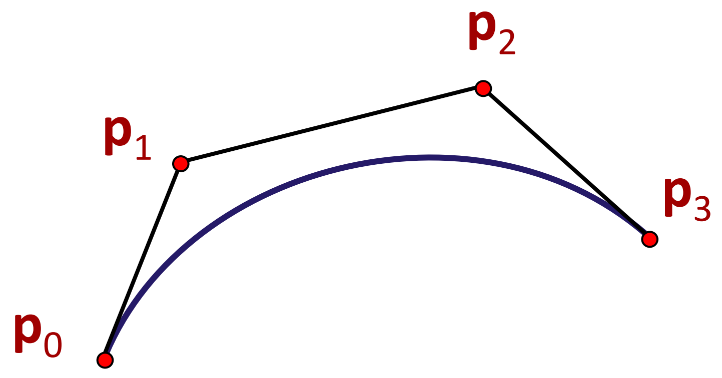


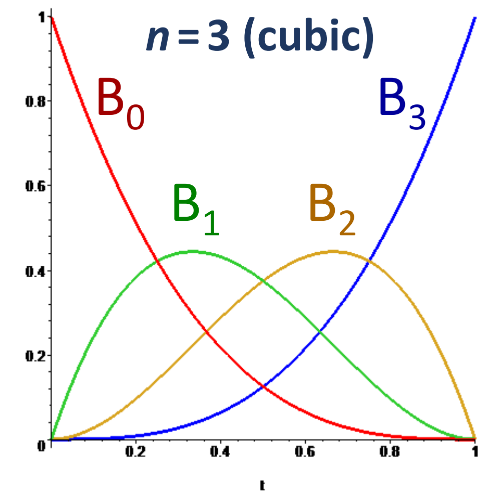


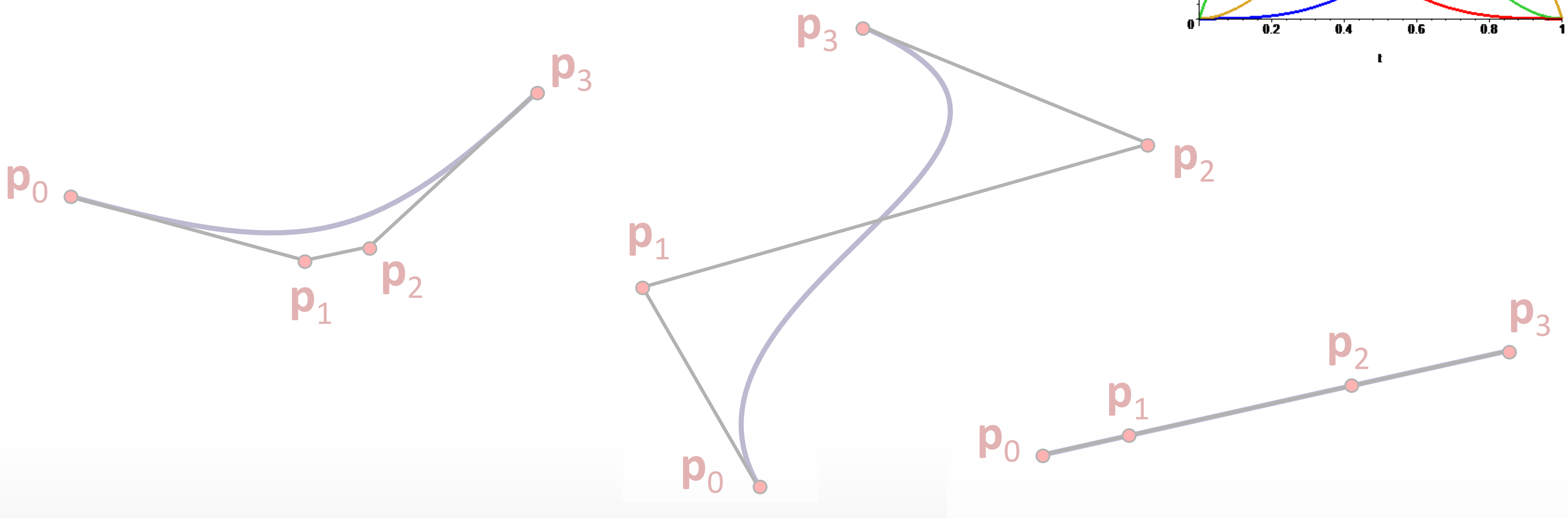
Bezier Curves: Examples

Bezier Curves:

- $\mathbf{f}(t) = \sum_{i=0}^{n} \mathbf{p}_i B_i^{(n)}$







Matrix Form

Matrix Notation: Bezier $\rightarrow$ Monomials

$$\mathbf{f}(t) = \begin{bmatrix} 1 & t & t^2 \end{bmatrix} \begin{pmatrix} 1 & 0 & 0 \\ -2 & 2 & 0 \\ 1 & -2 & 1 \end{pmatrix} \begin{pmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \end{pmatrix}$$

(quadratic case)

$$\mathbf{f}(t) = \begin{bmatrix} 1 & t & t^2 & t^3 \end{bmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{pmatrix} \begin{pmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{pmatrix}$$

(cubic case)

Format Conversion

Conversion: Compute Bezier coefficients from monomial coefficients

$$\begin{pmatrix} c_0^{(Bez.)} \\ c_1^{(Bez.)} \\ c_2^{(Bez.)} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 2 & 0 \\ 1 & -2 & 1 \end{pmatrix}^{-1} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \end{pmatrix}$$

(quadratic case)

$$\begin{pmatrix} c_0^{(Bez.)} \\ c_1^{(Bez.)} \\ c_2^{(Bez.)} \\ c_3^{(Bez.)} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{pmatrix}^{-1} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix}$$

(cubic case)

Format Conversion

Conversion: quadratic to cubic

$$\begin{pmatrix} c_0^{(3)} \\ c_1^{(3)} \\ c_2^{(3)} \\ c_3^{(3)} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{pmatrix}^{-1} \begin{pmatrix} 1 & 0 & 0 & 0 \\ -2 & 2 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} c_0^{(2)} \\ c_1^{(2)} \\ c_2^{(2)} \\ 0 \end{pmatrix}$$

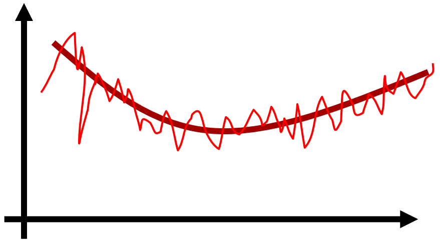
Convert to monomials and back to Bezier coefficients.
(Other degrees similar)

Example Application: Output of TrueType fonts in Postscript.

The Other Way Round...

Reducing the degree:

- Exact solution is not always possible
- Approximate solution: least-squares (function approximation)
- System of normal equations:



$$\begin{pmatrix} \langle \tilde{b}_1, \tilde{b}_1 \rangle & \cdots & \langle \tilde{b}_n, \tilde{b}_1 \rangle \\ \vdots & \ddots & \vdots \\ \langle \tilde{b}_1, \tilde{b}_n \rangle & \cdots & \langle \tilde{b}_n, \tilde{b}_n \rangle \end{pmatrix} \begin{pmatrix} \tilde{c}_1 \\ \vdots \\ \tilde{c}_n \end{pmatrix} = \begin{pmatrix} \langle \tilde{b}_1(x), f \rangle \\ \vdots \\ \langle \tilde{b}_n(x), f \rangle \end{pmatrix}, \quad f(t) := \sum_{i=1}^{m} c_i b_i(t)$$

$$\Leftrightarrow \begin{pmatrix} \langle \tilde{b}_1, \tilde{b}_1 \rangle & \cdots & \langle \tilde{b}_n, \tilde{b}_1 \rangle \\ \vdots & \ddots & \vdots \\ \langle \tilde{b}_1, \tilde{b}_n \rangle & \cdots & \langle \tilde{b}_n, \tilde{b}_n \rangle \end{pmatrix} \begin{pmatrix} \tilde{c}_1 \\ \vdots \\ \tilde{c}_n \end{pmatrix} = \sum_{i=1}^{m} c_i \begin{pmatrix} \langle \tilde{b}_1(x), b_i(t) \rangle \\ \vdots \\ \langle \tilde{b}_n(x), b_i(t) \rangle \end{pmatrix}$$

Cubic → Quadratic Case

Reducing the degree: Cubic → Quadratic

$$\begin{pmatrix} \frac{1}{7} & \frac{1}{14} & \frac{1}{35} \\ \frac{1}{14} & \frac{3}{35} & \frac{9}{140} \\ \frac{1}{35} & \frac{9}{140} & \frac{3}{35} \end{pmatrix} \begin{pmatrix} \tilde{c}_0 \\ \tilde{c}_1 \\ \tilde{c}_2 \end{pmatrix} = c_0 \begin{pmatrix} \frac{1}{6} \\ \frac{1}{15} \\ \frac{1}{60} \end{pmatrix} + c_1 \begin{pmatrix} \frac{1}{10} \\ \frac{1}{10} \\ \frac{1}{20} \end{pmatrix} + c_2 \begin{pmatrix} \frac{1}{20} \\ \frac{1}{10} \\ \frac{1}{10} \end{pmatrix} + c_3 \begin{pmatrix} \frac{1}{60} \\ \frac{1}{15} \\ \frac{1}{6} \end{pmatrix}$$

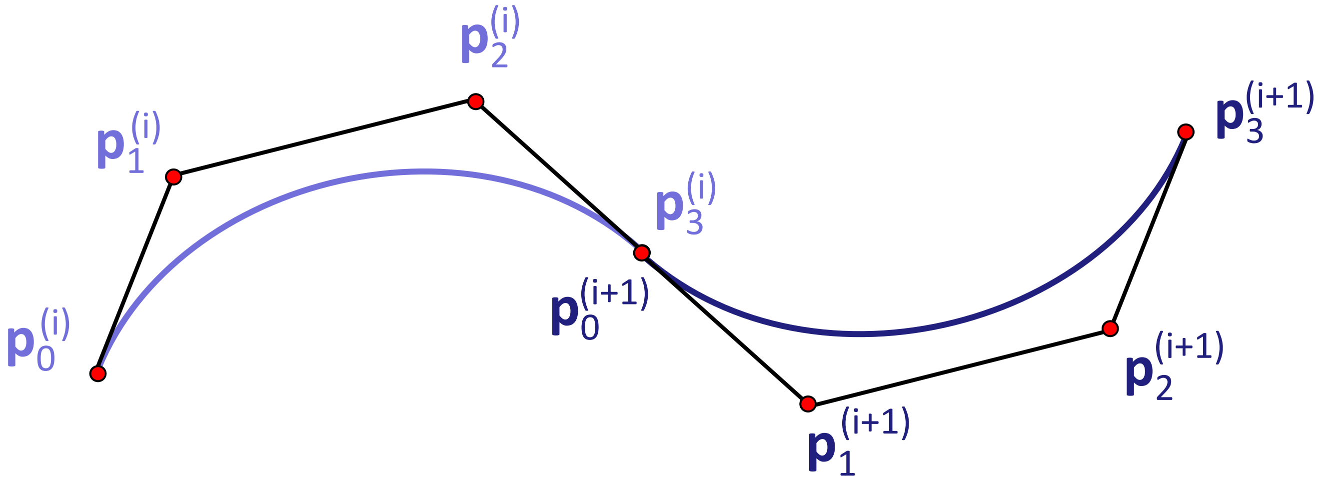
Polynomial Spline Curves

Bezier Splines

Bezier Splines

Local control: Bezier splines

- Concatenate several curve segments
- Question: Which constraints to place upon the control points in order to get C^{-1}, C^0, C^1, C^2 continuity?



Derivatives

Bernstein basis properties:

- Derivatives:

$$\frac{d}{dt}B_i^{(n)}(t) = \binom{n}{i}\left(it^{\{i-1\}}(1-t)^{n-i} - (n-i)t^{\{i\}}(1-t)^{\{n-i-1\}}\right)$$

$$= \frac{n!}{(n-i)!\,i!}it^{\{i-1\}}(1-t)^{n-i} - \frac{n!}{(n-i)!\,i!}(n-i)t^{i}(1-t)^{\{n-i-1\}}$$

$$= n\left[\binom{n-1}{i-1}t^{\{i-1\}}(1-t)^{n-i} - \binom{n-1}{i}t^{i}(1-t)^{\{n-i-1\}}\right]$$

$$= n\left[B_{i-1}^{(n-1)}(t) - B_i^{(n-1)}(t)\right]$$

($\{k\} = k$ if $k > 0$, zero otherwise)

Derivatives

Bernstein basis properties:

- Derivatives:

$$\frac{d^2}{dt^2}B_i^{(n)}(t) = \frac{d}{dt}\binom{n}{i}\left(it^{\{i-1\}}(1-t)^{n-i} - (n-i)t^i(1-t)^{\{n-i-1\}}\right)$$

$$= \binom{n}{i}\Big(\{i-1\}it^{\{i-2\}}(1-t)^{n-i} - i(n-i)t^{\{i-1\}}(1-t)^{\{n-i-1\}}$$

$$- i(n-i)t^{\{i-1\}}(1-t)^{\{n-i-1\}} + \{n-i-1\}(n-i)t^{\{i\}}(1-t)^{\{n-i-2\}}\Big)$$

$$= n(n-1)\left[B_{i-2}^{(n-2)}(t) - 2B_{i-1}^{(n-2)}(t) + B_i^{(n-2)}(t)\right]$$

($\{k\} = k$ if $k > 0$, zero otherwise)

Bezier Curve Properties

Important for continuous concatenation:

- Function value at {0,1}:

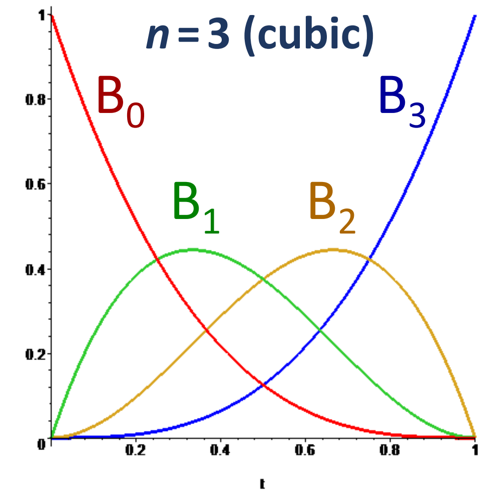
$$\mathbf{f}(t) = \sum_{i=0}^{n} \binom{n}{i} t^i (1-t)^{n-i} \mathbf{p}_i$$

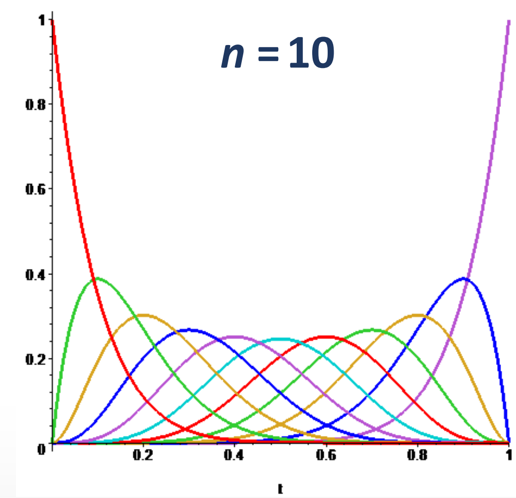
$$\longrightarrow \mathbf{f}(0) = \mathbf{p}_0$$

$$\mathbf{f}(1) = \mathbf{p}_n$$

- First derivative vector at {0,1}

- Second derivative vector at {0,1}





Bezier Curve Properties

Important for continuous concatenation:

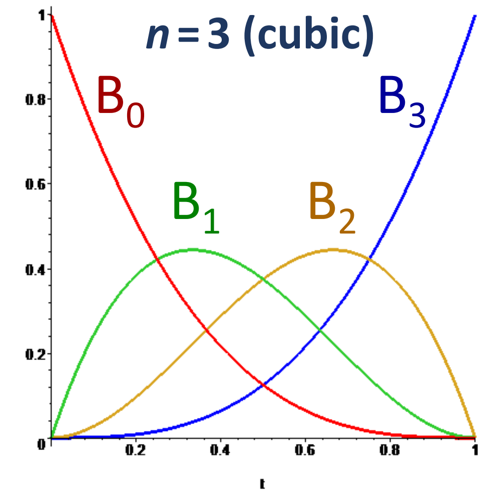
- Function value at {0,1}:

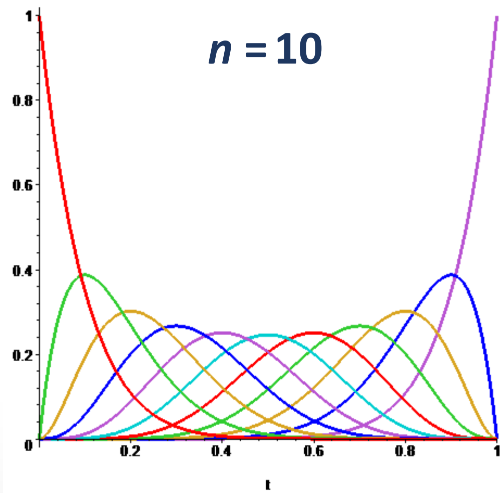
  $\mathbf{f}(0) = \mathbf{p}_0, \ \mathbf{f}(1) = \mathbf{p}_n$

- First derivative vector at {0,1}:

  $\frac{d}{dt}\mathbf{f}(t)$

- Second derivative vector at {0,1}





Bezier Curve Properties

Important for continuous concatenation:

- Function value at {0,1}:

$$\mathbf{f}(0) = \mathbf{p}_0,\ \mathbf{f}(1) = \mathbf{p}_n$$

- First derivative vector at {0,1}:

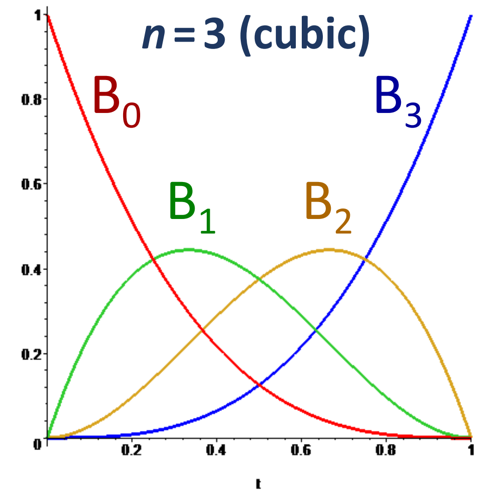
$$\frac{d}{dt}\mathbf{f}(t) = n\sum_{i=0}^{n}\left[B_{i-1}^{(n-1)}(t) - B_{i}^{(n-1)}(t)\right]\mathbf{p}_i$$

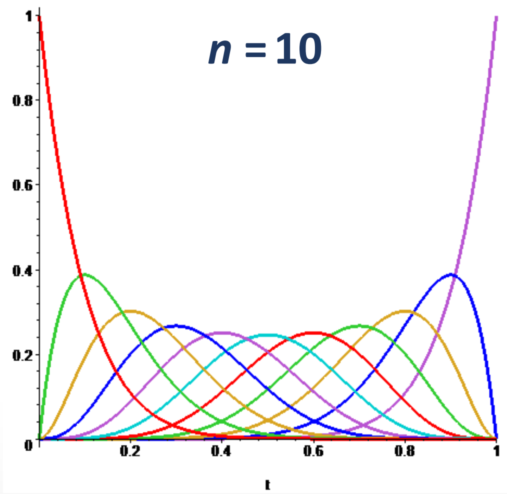
$$= \left(n\left[-B_0^{(n-1)}(t)\right]\mathbf{p}_0 + \left[B_0^{(n-1)}(t) - B_1^{(n-1)}(t)\right]\mathbf{p}_1 + \ldots\right.$$

$$\left.\ldots + \left[B_{n-2}^{(n-1)}(t) - B_{n-1}^{(n-1)}(t)\right]\mathbf{p}_{n-1} + \left[B_{n-1}^{(n-1)}(t)\right]\mathbf{p}_n\right)$$

$$\mathbf{f}'(0) = n(\mathbf{p}_1 - \mathbf{p}_0) \quad \mathbf{f}'(1) = n(\mathbf{p}_n - \mathbf{p}_{n-1})$$

- Second derivative vector at {0,1}





Bezier Curve Properties

Important for continuous concatenation:

- Function value at {0,1}:

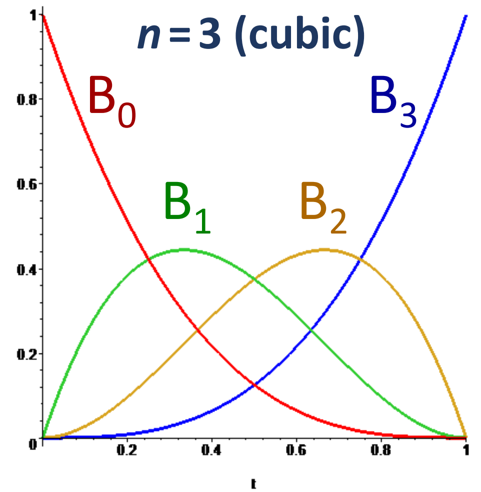
$$\mathbf{f}(0) = \mathbf{p}_0$$
$$\mathbf{f}(1) = \mathbf{p}_n$$

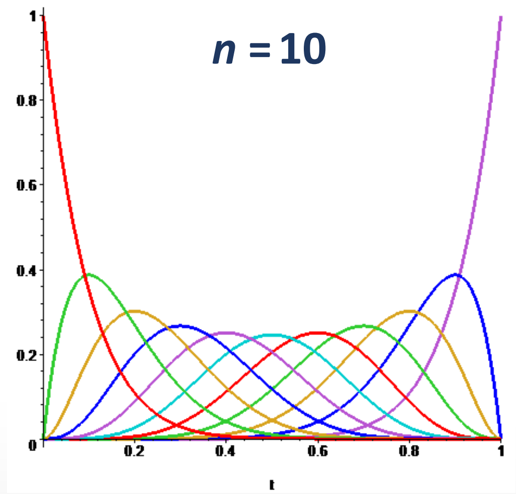
- First derivative vector at {0,1}:

$$\mathbf{f}'(0) = n[\mathbf{p}_1 - \mathbf{p}_0]$$
$$\mathbf{f}'(1) = n[\mathbf{p}_n - \mathbf{p}_{n-1}]$$

- Second derivative vector at {0,1}:

$$\mathbf{f}''(0) = n(n-1)[\mathbf{p}_0 - 2\mathbf{p}_1 + \mathbf{p}_2]$$
$$\mathbf{f}''(1) = n(n-1)[\mathbf{p}_n - 2\mathbf{p}_{n-1} + \mathbf{p}_{n-2}]$$

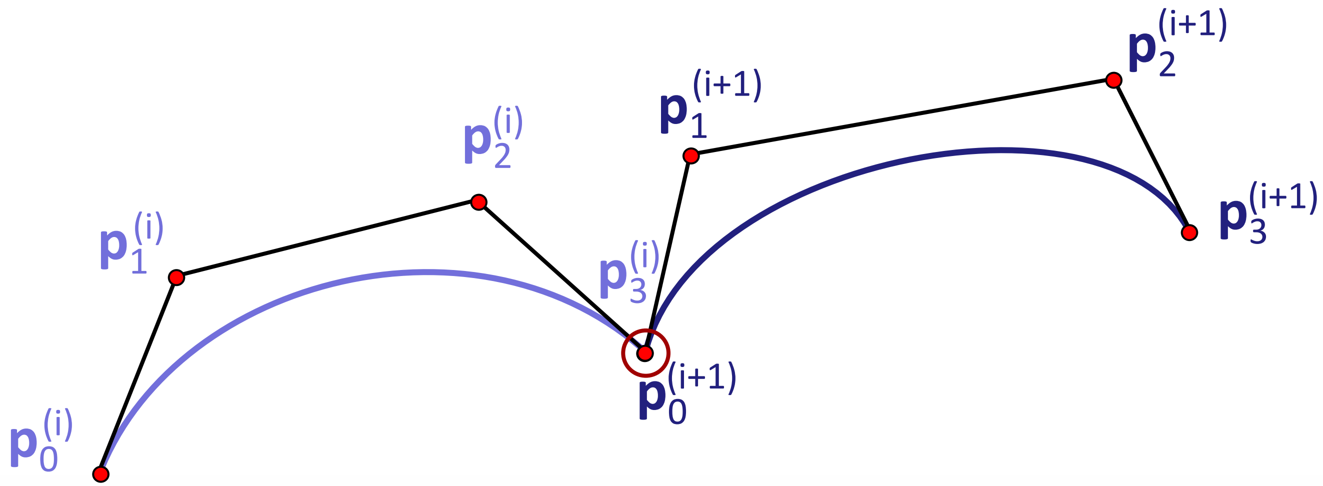




Bezier Spline Continuity

Rules for Bezier spline continuity:

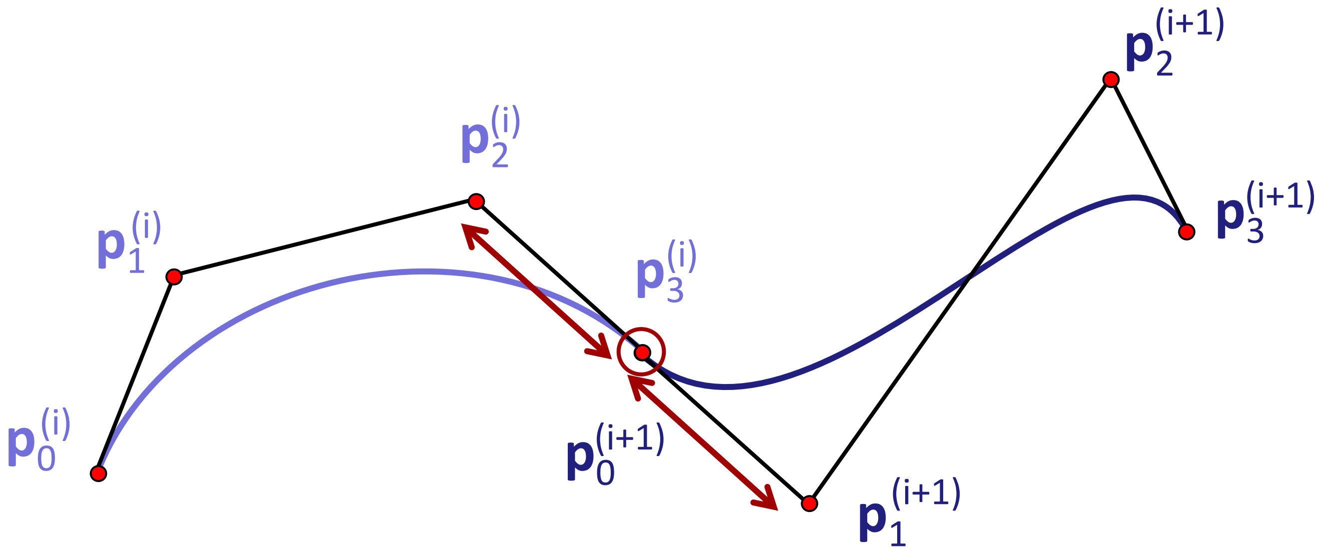
- C^0 continuity:
  - Each spline segment interpolates the first and last control point
  - Therefore: Points of neighboring segments have to coincide for C^0 continuity.



Bezier Spline Continuity

Rules for Bezier spline continuity:

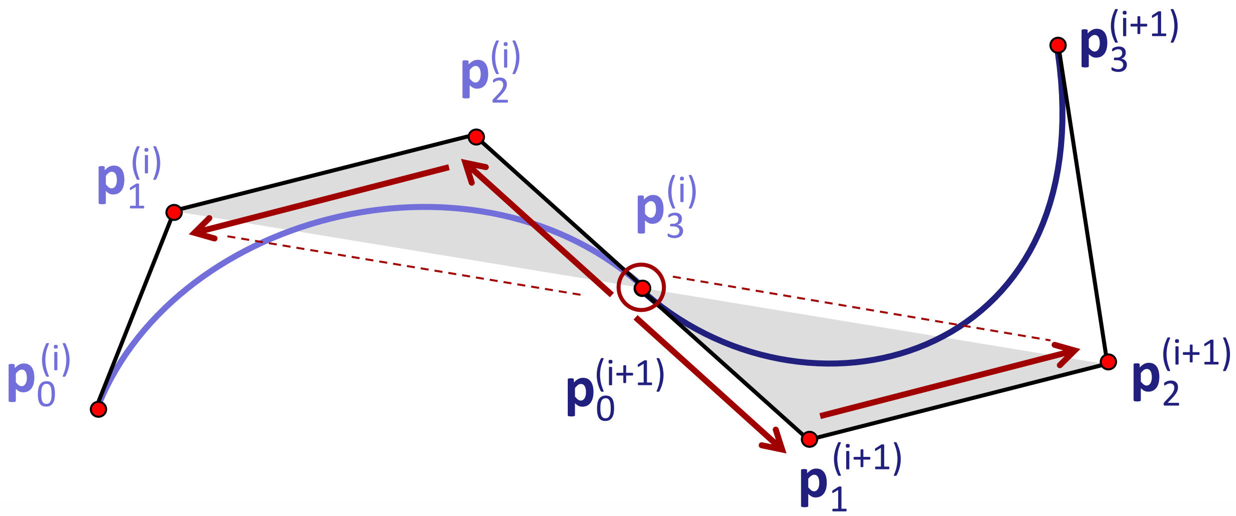
- Additional requirement for C^1 continuity:
  - Tangent vectors are proportional to differences $\mathbf{p}_1 - \mathbf{p}_0$, $\mathbf{p}_n - \mathbf{p}_{n\text{-}1}$
  - Therefore: These vectors must be identical for C^1 continuity



Bezier Spline Continuity

Rules for Bezier spline continuity:

- Additional requirement for C^2 continuity:
  - d^2/dt^2 vectors are prop. to $(\mathbf{p}_2 - 2\mathbf{p}_1 + \mathbf{p}_0)$, $(\mathbf{p}_n - 2\mathbf{p}_{n-1} + \mathbf{p}_{n-2})$
  - Tangents must be the same (C^2 implies C^1)
  - Therefore: Triangle of first / last three points must be the same

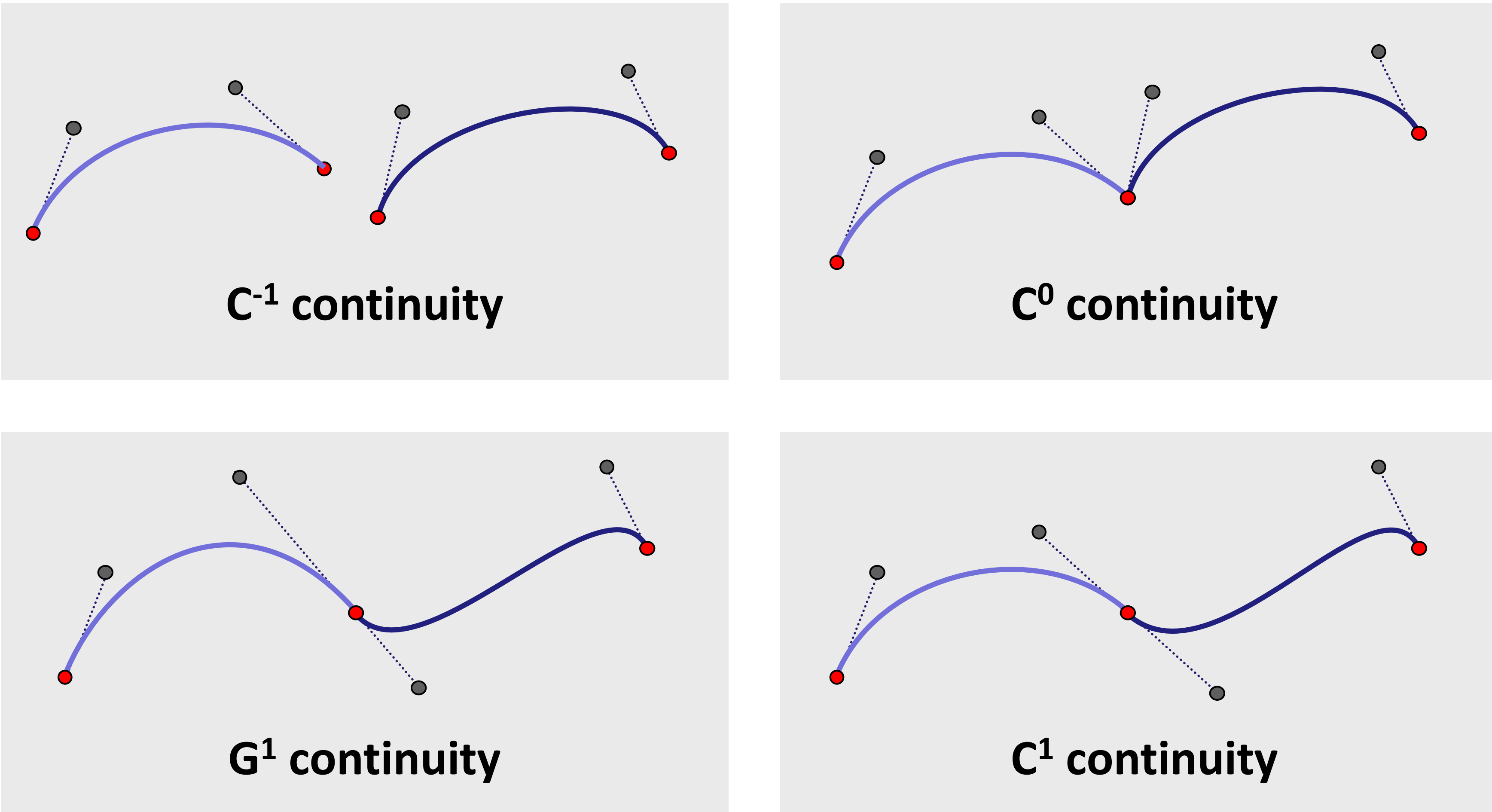


In Practice

In practice:

- Everyone is using cubic Bezier curves
- Higher degrees rare (CAD/CAM applications)
- Typically: “points & handles” interface
- Four modes:
  - Discontinuous (two curves)
  - C^0 Continuous (points meet)
  - Tangent direction continuous (handles point into the same direction, but different length)
    (“G^1 continuous”, more on this shortly)
  - C^1 Continuous (handle points have symmetric vectors)
- C^2 is rarely supported (too restrictive, no local control)

Bezier Curve Editing



Geometric Continuity

Parametric Continuity:

- C^0, C^1, C^2... continuity.
- Does a particle moving on this curve have a smooth trajectory (position, velocity, acceleration,...)?
- Useful for animation (object movement, camera paths)

Geometric Continuity:

- Is the curve itself smooth?
- C.f.: differential geometry – parametrization independent measures
- More relevant for modeling (curve design)

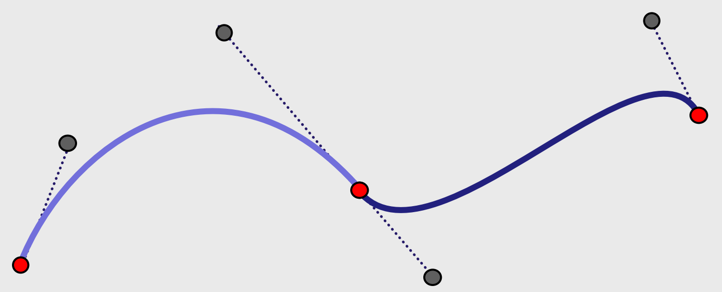
Geometric Continuity

Geometric Continuity:

- $G^0 = C^0$: position varies continuously
- G^1: tangent directions varies continuously
  - In other words: the normalized tangent varies continuously
  - Equivalently: The curve can be reparametrized so that it becomes C^1.
  - Also equivalent: A unit speed parametrization would be C^1.
- G^2: curvature varies continuously
  - Equivalently: The curve can be reparametrized so that it becomes C^2.
  - Also equivalent: A unit speed parametrization would be C^2.

Geometric Continuity for Bezier Splines

This means:



G^1 continuity

This Bezier curve is G^1: It can be reparametrized to become C^1. (Just increase the speed for the second segment by ratio of tangent vector lengths.)

Polynomial Spline Curves

Uniform Cubic B-Splines

Literature

Literature:

- *An Introduction to Splines for use in Computer Graphics and Geometric Modeling*
  Richard H. Bartels, John C. Beatty, Brian A. Barsky
  Morgan Kaufmann 1987

  (now hard to get, waited several month on Amazon)

Overview

Uniform cubic B-splines

- This is a special case of general B-splines
  - (which additionally provide arbitrary degree and general and non-uniform parametrization)
- We look at this first to get an intuition for the basic ideas and concepts for B-splines
- Some derivations are left out – will be shown later for the general case

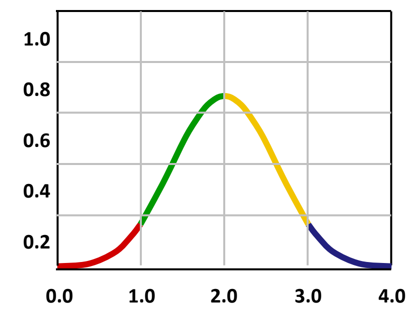
Overview

Improvement over cubic Bezier splines:

- C^2 continuity is easily attainable
- We will use only one type of basis functions
  - Shifted in the domain to create curves with multiple segments
  - This principle is conceptually easier to apply in general modeling problems (e.g. as a basis for finite elements, for PDEs or variational problems)

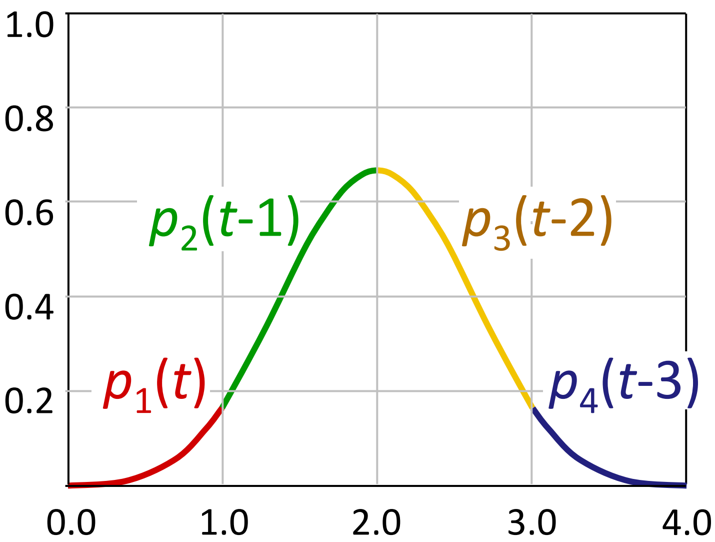
Key Ideas

Key Ideas:

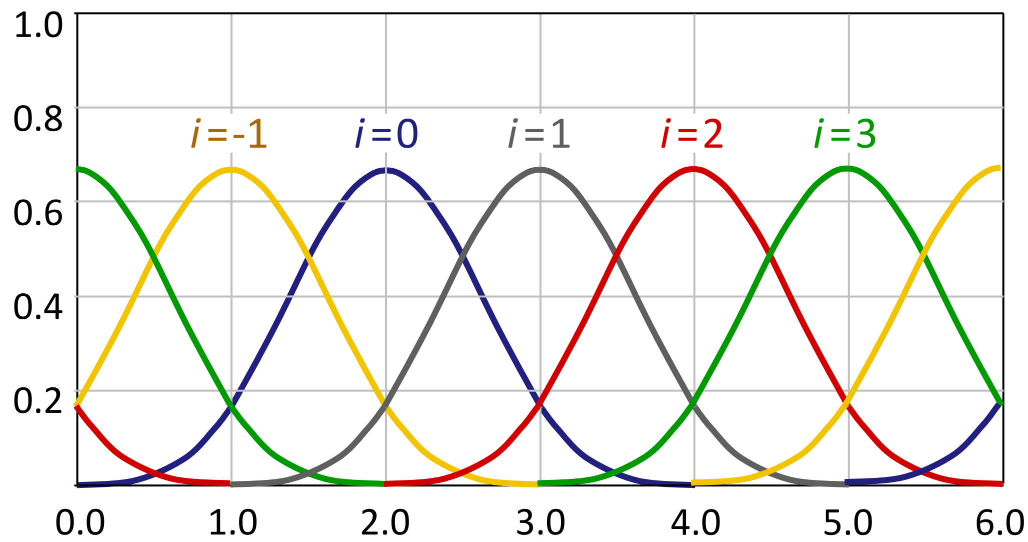


- We design one basis function $b(t)$
- Properties:
  - $b(t)$ is C^2 continuous.
  - $b(t)$ is piecewise polynomial, degree 3 (cubic).
  - $b(t)$ has local support.
  - Overlaying shifted $b(t + i)$ forms a partition of unity.
  - $b(t) \geq 0$ for all t
- In short: We build-in all the desirable properties into the basis. Linear combinations will inherit these.

Shifted Basis Functions



basis function $b(t)$



shifted basis functions $b(t-i)$ for [0..6]

Basis function:

- Consists of four polynomial parts $p_1 \ldots p_4$.
- Shifted basis $b(t-i)$: spacing of 1.
- Each interval to be used must be overlapped by 4 different b_i.

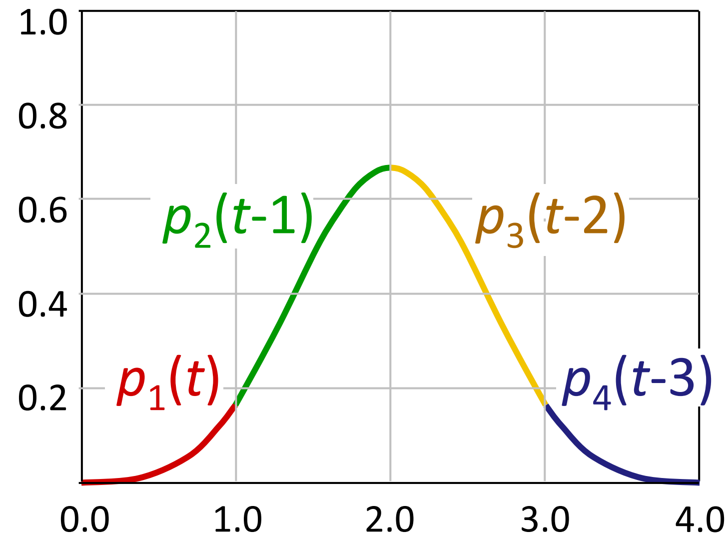
Basis Function

$$p_1(t) = \frac{1}{6}t^3$$

$$p_2(t) = \frac{1}{6}\left(1 + 3t + 3t^2 - 3t^3\right)$$

$$p_3(t) = \frac{1}{6}\left(4 - 6t^2 + 3t^3\right)$$

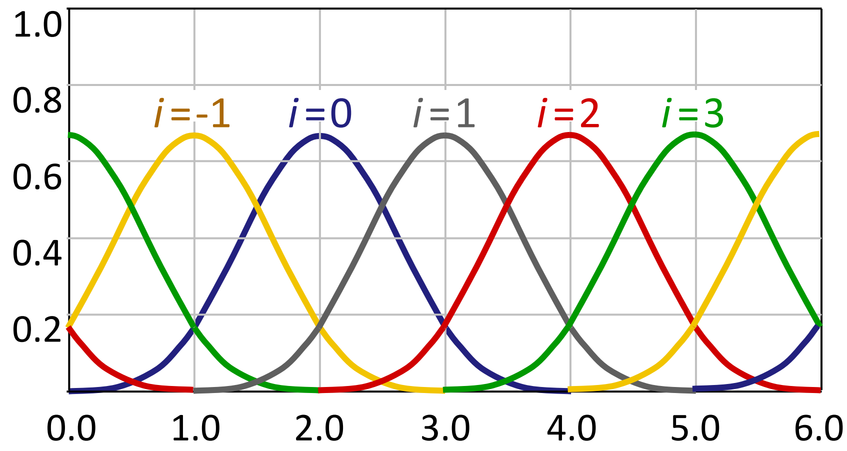
$$p_4(t) = \frac{1}{6}\left(1 - 3t + 3t^2 - t^3\right)$$



basis function $b(t)$

$$b(t) = \begin{cases} 0 & \text{if } t < 0 \\ p_1(t) & \text{if } 0 < t \le 1 \\ p_2(t-1) & \text{if } 1 < t \le 2 \\ p_3(t-2) & \text{if } 2 < t \le 3 \\ p_4(t-3) & \text{if } 3 < t \le 4 \\ 0 & \text{if } t > 4 \end{cases} = \begin{cases} 0 & \text{if } t < 0 \\ \frac{1}{6}t^3 & \text{if } 0 < t \le 1 \\ \frac{1}{6}\left(1 + 3(t-1) + 3(t-1)^2 - 3(t-1)^3\right) & \text{if } 1 < t \le 2 \\ \frac{1}{6}\left(4 - 6(t-2)^2 + 3(t-2)^3\right) & \text{if } 2 < t \le 3 \\ \frac{1}{6}\left(1 - 3(t-3) + 3(t-3)^2 - (t-3)^3\right) & \text{if } 3 < t \le 4 \\ 0 & \text{if } t > 4 \end{cases}$$

Creating Curves



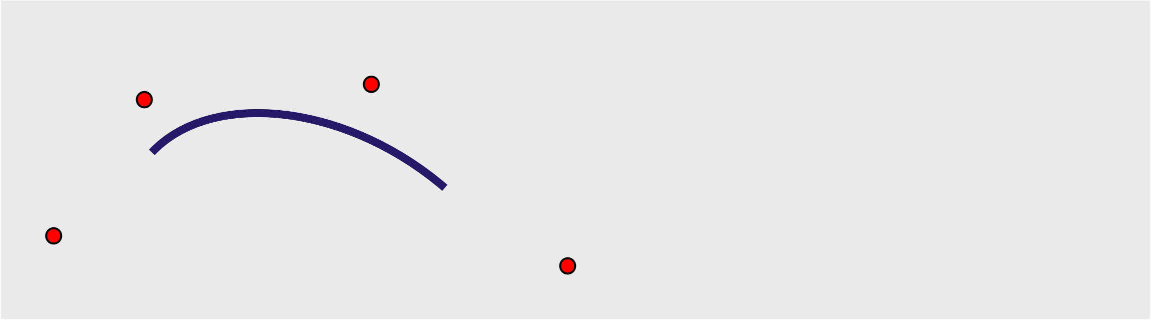
Creating uniform B-spline curves:

- Choose parameter interval $[n_{start}\text{-}2, n_{end}+2,]$, $n_{start} < n_{end} \in \mathbb{Z}$
- Use all shifted basis functions that overlap this interval:

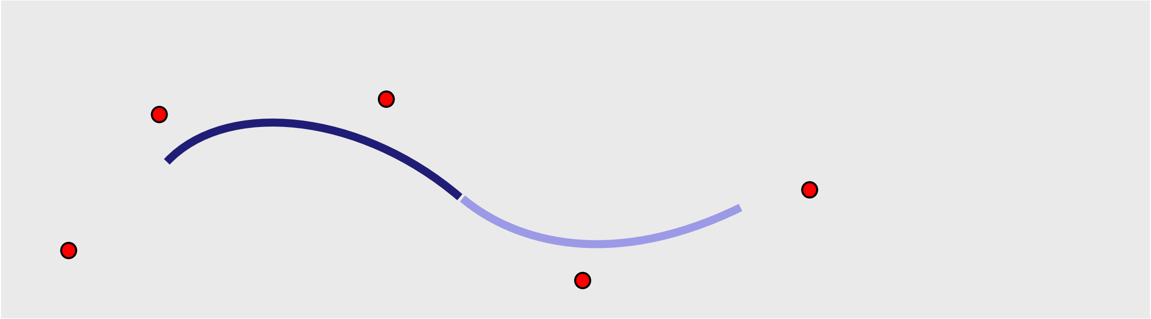
  $\mathbf{B} = \{b(t - [n_{start}\text{-}1]), \ldots, b(t - [n_{end}+1])\}$

- Form linear combinations: $\mathbf{f}(t) = \sum_{i=n_{start}-1}^{n_{end}+1} b(t-i)\mathbf{p}_i$

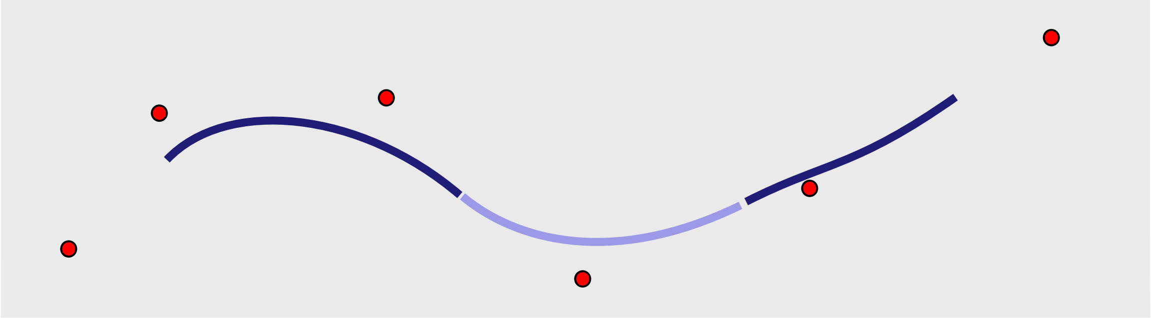
Uniform B-Spline Curves



one segment



two segments



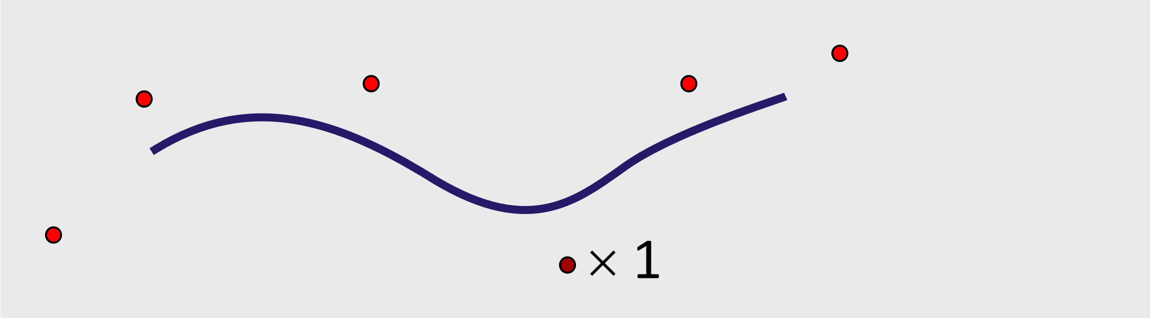
three segments

Discontinuities

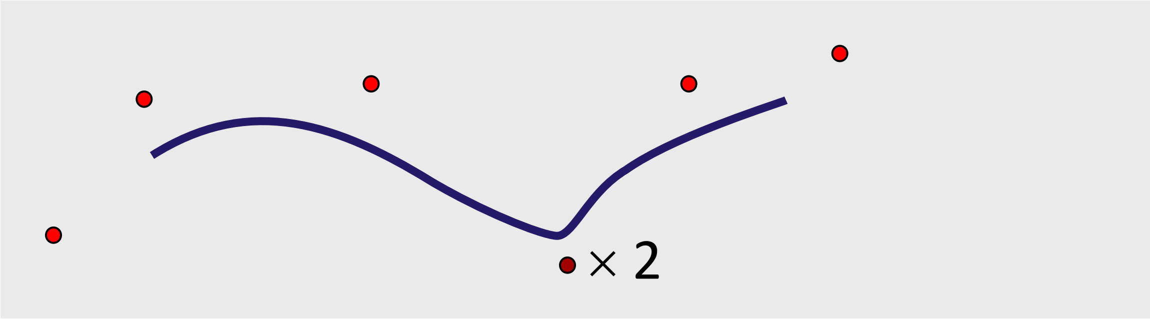
Continuity Control

- Easier than with Bezier curves
- The parametric function is always C^2, by construction
- However: We can create curves with lower geometric smoothness
  - This will lead to a degenerate (non-regular) parametrization
  - This problem is fixed with general, non-uniform B-Splines
- Basic idea: Double control points
  - Single points: G^2 curve
  - Double points: G^1 curve
  - Triple points: G^0 curve

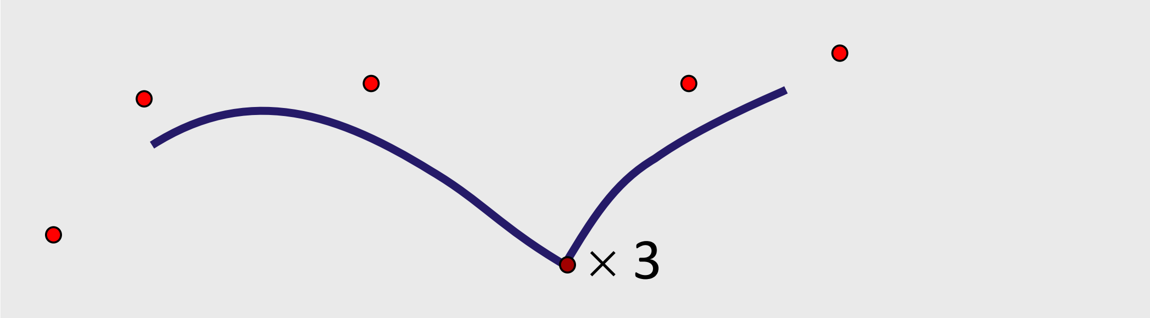
Continuity Control



single points

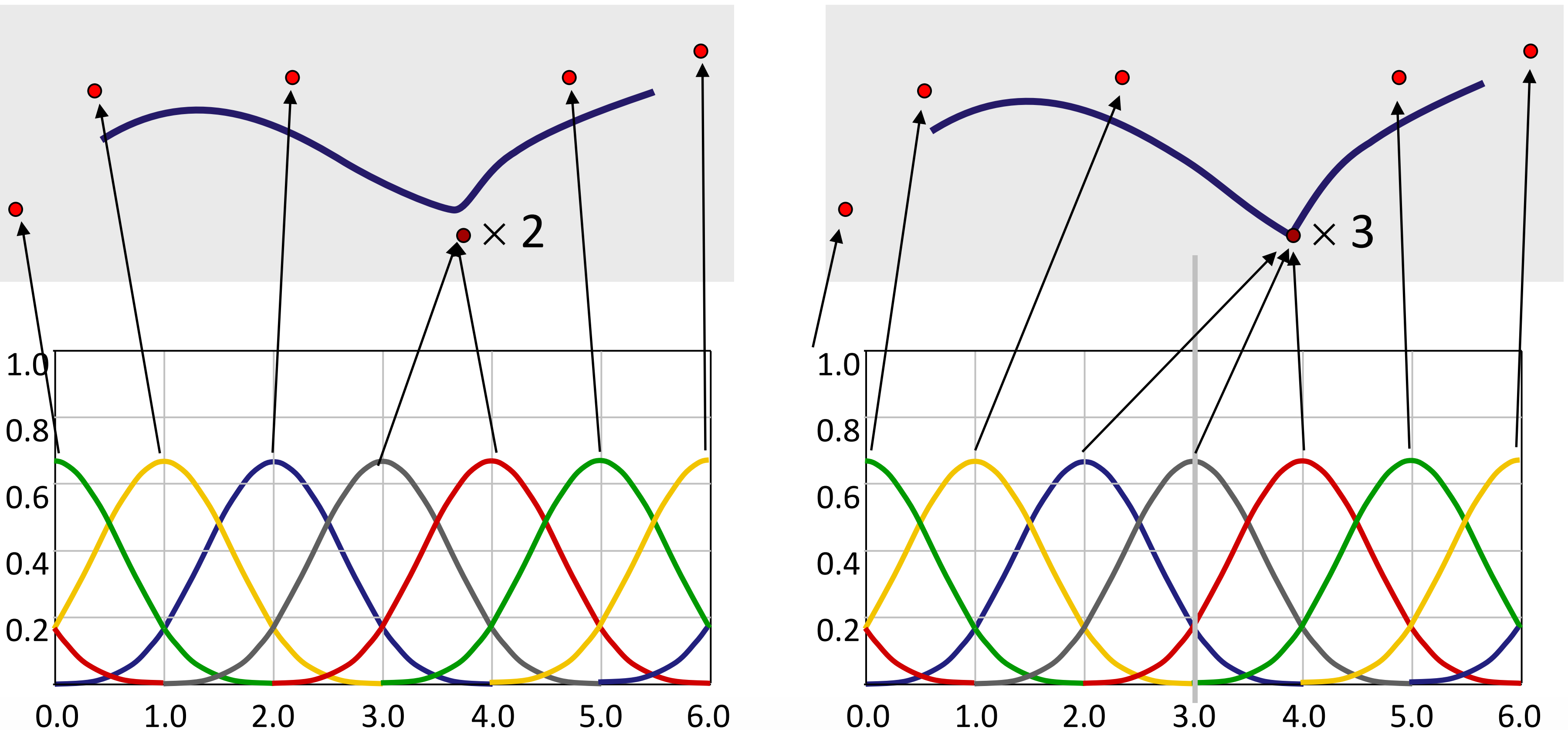


double point



triple point
(interpolates that point)

Illustration

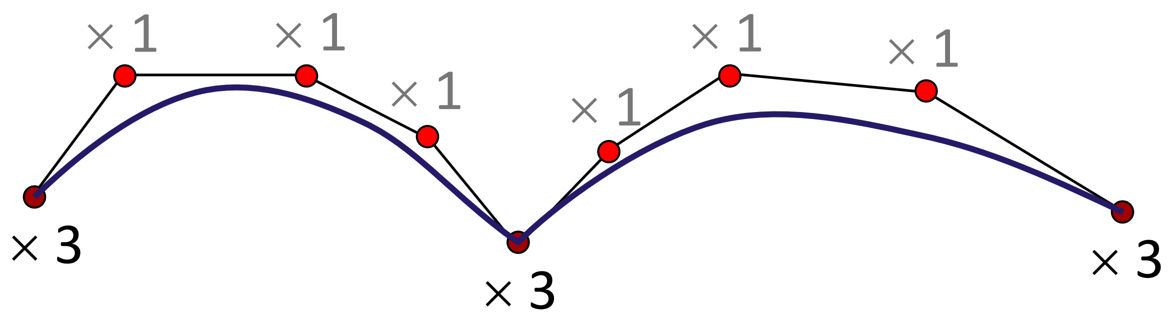


End Conditions

Problem:

- We need at least 4 points for one spline segment, 5 for two and so on.
- Means: We need $s + 3$ control points for s segments (rather than $s + 1$), two more than in spline interpolation.
- This is inconvenient...
- Simple solution:
  - Use double or triple end points
  - Triple end points will be interpolated
  - We will get along with $s + 1$ control points

Knot Sequences



Specifying a uniform, cubic B-Spline curve:

- A set of control points $\mathbf{p}_1, \ldots, \mathbf{p}_n$.
- Knot multiplicities $(i_1, \ldots, i_k)$, $i_j \in \{1,2,3\}$, $\Sigma i_j = n$.
  - For example (3,1,1,1,3,1,1,1,3)
    - Creates one sharp corner in the middle of the spline
    - Interpolates the end points

Conversion to Bezier Basis

Uniform B-Splines can be converted to the Bezier Format:

- Cubic polynomial pieces
- Each can be represented as Bezier segment
- Just a basis change...

Basis Change

$$M_{UB\rightarrow Mn}=\frac{1}{6}\begin{pmatrix}0 & 1 & 4 & 1\\ 0 & 3 & 0 & -3\\ 0 & 3 & -6 & 3\\ 1 & -3 & 3 & -1\end{pmatrix}$$

$$M_{Bez\rightarrow Mn}=\begin{pmatrix}1 & 0 & 0 & 0\\ -3 & 3 & 0 & 0\\ 3 & -6 & 3 & 0\\ -1 & 3 & -3 & 1\end{pmatrix}$$

$$M_{UB\rightarrow Bez}=\left(M_{Bez\rightarrow Mn}\right)^{-1}M_{UB\rightarrow Mn}$$

$$\begin{pmatrix}\tilde{\mathbf{b}}_i^{(1)}\\ \tilde{\mathbf{b}}_i^{(2)}\\ \tilde{\mathbf{b}}_i^{(3)}\\ \tilde{\mathbf{b}}_i^{(4)}\end{pmatrix}=M_{UB\rightarrow Bez}\begin{pmatrix}\mathbf{b}_i^{(1)}\\ \mathbf{b}_{i-1}^{(2)}\\ \mathbf{b}_{i-2}^{(3)}\\ \mathbf{b}_{i-3}^{(4)}\end{pmatrix}$$

Uniform B-Spline:

$$p_1(t)=\frac{1}{6}t^3$$

$$p_2(t)=\frac{1}{6}\left(1+3t+3t^2-3t^3\right)$$

$$p_3(t)=\frac{1}{6}\left(4-6t^2+3t^3\right)$$

$$p_4(t)=\frac{1}{6}\left(1-3t+3t^2-t^3\right)$$

Bezier-Spline:

$$B_0^{(3)}:=(1-t)^3 \qquad B_1^{(3)}:=3t(1-t)^2$$

$$B_2^{(3)}:=3t^2(1-t) \qquad B_3^{(3)}:=t^3$$

Where does the basis come from?

How do we construct a B-Spline basis?

- Derivation of uniform cubic B-Splines?
- Generalizations:
  - General degree?
  - Non-uniform parametrization?

Three Approaches:

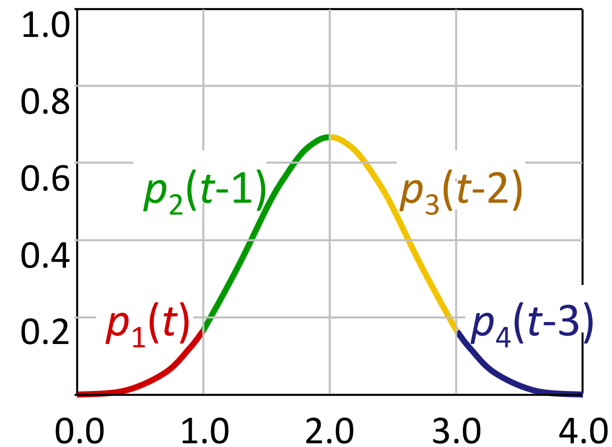
Three ways to get the basis:

1. The elementary approach:
   - Derive a linear system of equations, solve
2. Repeated convolution:
   - d-fold convolution of box functions
3. de-Boor Recursion:
   - Repeated linear interpolation

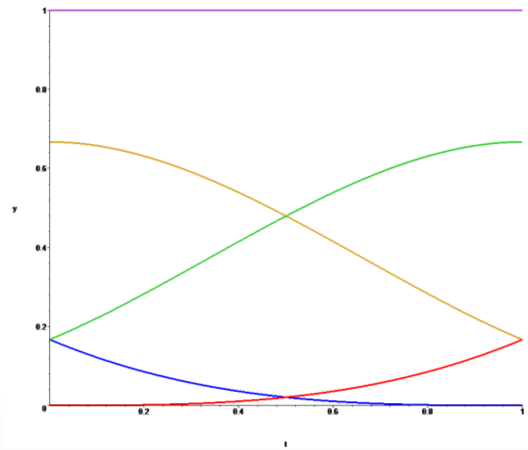
The Elementary Approach

Cubic Uniform B-Spline Basis:

- One basis function, just shifted
- Consists of four pieces p_1, p_2, p_3, p_4.
- We just need the coefficients of the pieces
- Setting up a linear system...



basis function $b(t)$



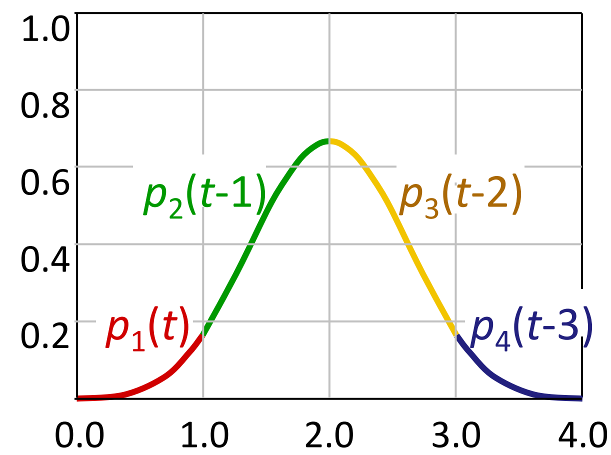
pieces $p_i(t)$

The Elementary Approach

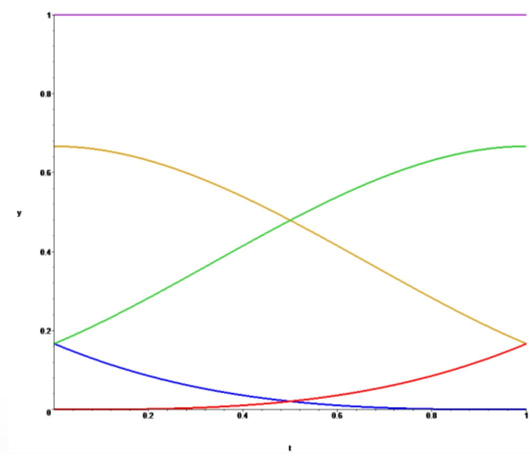
Linear system:

$$
\begin{array}{ccc}
p_1(0) = 0 & p_1'(0) = 0 & p_1''(0) = 0 \\
p_1(1) = p_2(0) & p_1'(1) = p_2'(0) & p_1''(1) = p_2''(0) \\
p_2(1) = p_3(0) & p_2'(1) = p_3'(0) & p_2''(1) = p_3''(0) \\
p_3(1) = p_4(0) & p_3'(1) = p_4'(0) & p_3''(1) = p_4''(0) \\
p_4(1) = 0 & p_4'(1) = 0 & p_4''(1) = 0
\end{array}
$$

$$p_0\left(\frac{1}{2}\right) + p_1\left(\frac{1}{2}\right) + p_2\left(\frac{1}{2}\right) + p_3\left(\frac{1}{2}\right) = 1$$



basis function $b(t)$



pieces $p_i(t)$

The Elementary Approach

Normalization:

$$p_1(0)=0 \qquad p_1'(0)=0 \qquad p_1''(0)=0$$
$$p_1(1)=p_2(0) \qquad p_1'(1)=p_2'(0) \qquad p_1''(1)=p_2''(0)$$
$$p_2(1)=p_3(0) \qquad p_2'(1)=p_3'(0) \qquad p_2''(1)=p_3''(0)$$
$$p_3(1)=p_4(0) \qquad p_3'(1)=p_4'(0) \qquad p_3''(1)=p_4''(0)$$
$$p_4(1)=0 \qquad p_4'(1)=0 \qquad p_4''(1)=0$$
$$p_0\left(\frac{1}{2}\right)+p_1\left(\frac{1}{2}\right)+p_2\left(\frac{1}{2}\right)+p_3\left(\frac{1}{2}\right)=1$$

- Completely determines the p_i.
- Turns out to hold everywhere in [0,1]
- Not yet clear why
- But: if it is possible, our conditions are sufficient
- So we have to show that it is possible

Positivity:

- Not enforced; we get this accidentally (simplicity)
- Same argument: if it's possible, the conditions are sufficient

Properties

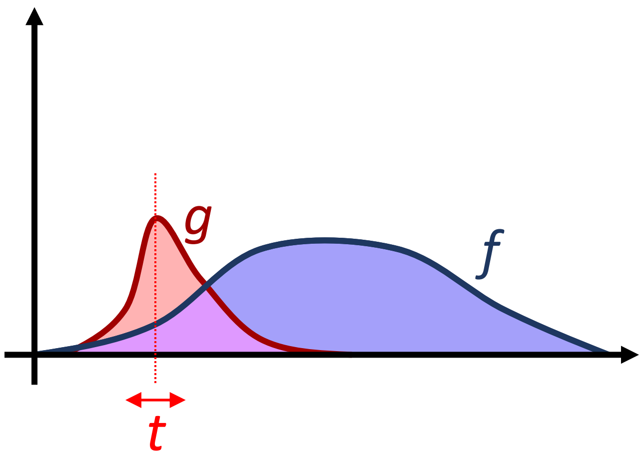
Minimal Support:

- We have 16 conditions (cubic case)
  - 15 for smoothness, one for normalization
  - Easy to see: linear independent
- Need 4 polynomial segments to get sufficiently many degrees of freedom
- Consequence: Any C^2 function with 3 or less polynomial segments, and the zero function everywhere else, must be the zero function.
  - 15 linear independent constraints, homogeneous.
  - Zero vector is the only solution.
- Therefore: We have *minimal support*.

Repeated Convolution

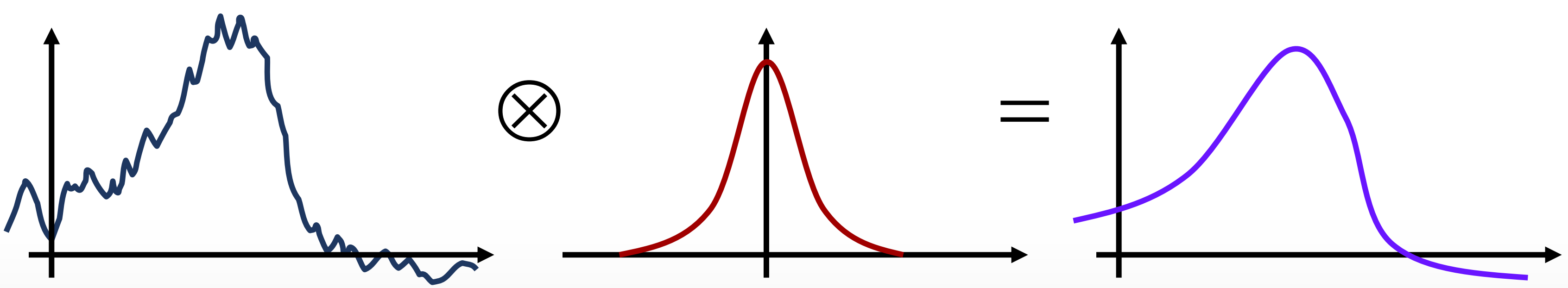
Convolution:

- Weighted average of functions
- Definition:



$$f(t) \otimes g(t) = \int_{-\infty}^{\infty} f(x) g(x-t) dx$$

Example:



Repeated Convolution

A Different Derivation:

- We start with 0th degree basis functions
- Increase smoothness by convolution

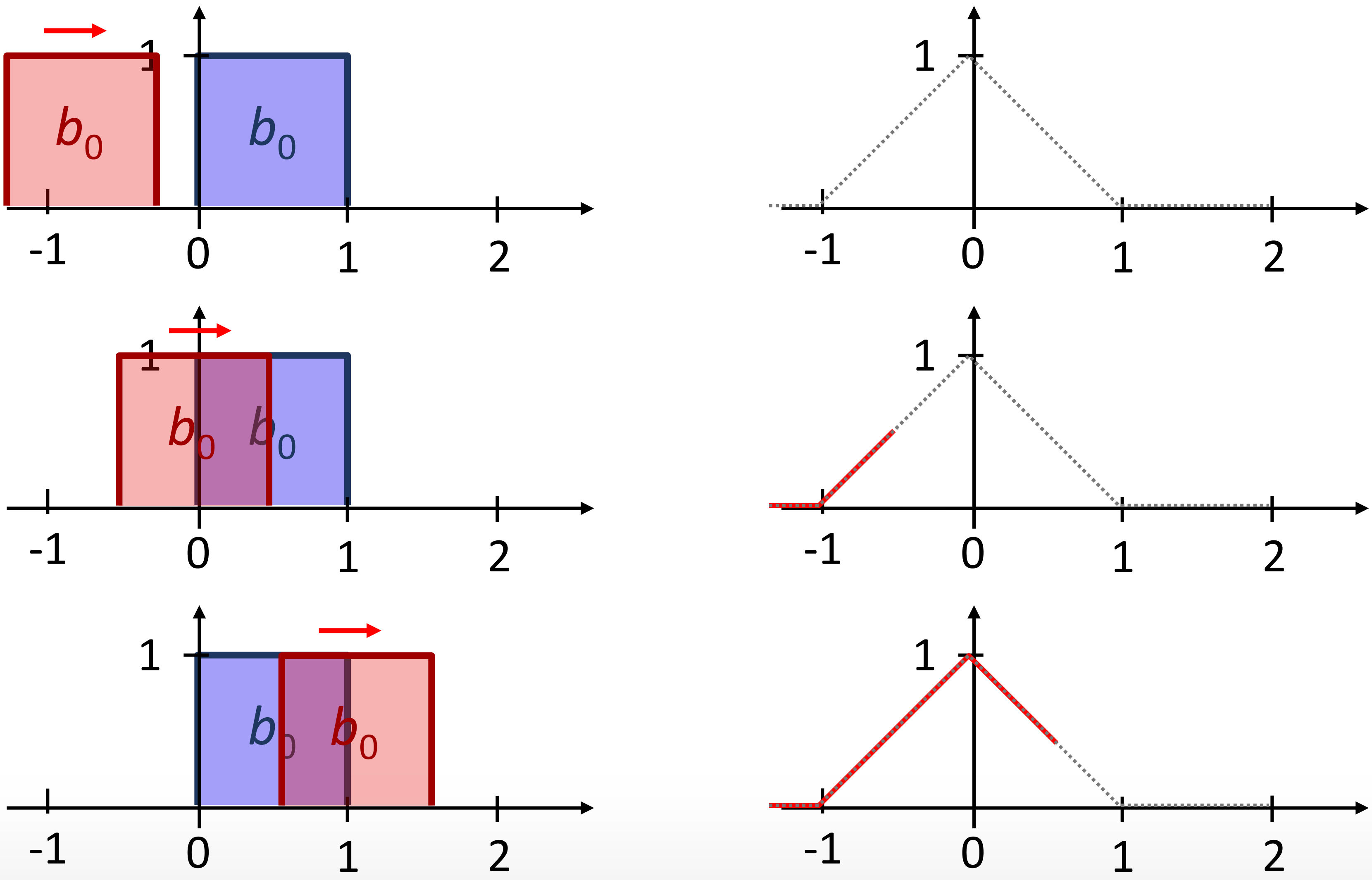
Degree-zero B-Spline:

$$b^{(0)}(t) = \begin{cases} 1, \text{ if } \ t \in [0 \ldots 1) \\ 0, \text{ otherwise} \end{cases}$$

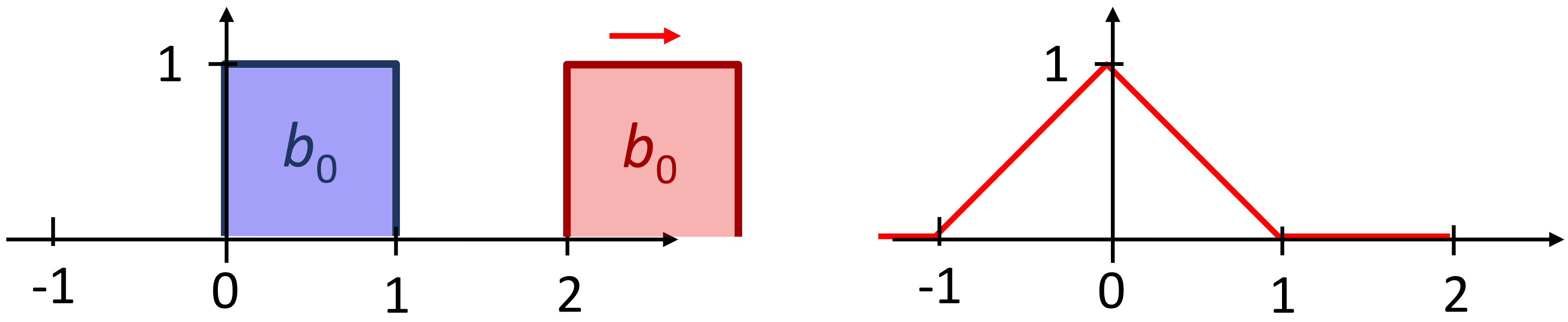
General-degree B-Spline:

$$b^{(i)}(t) = b^{(i)}(t) \otimes b^{(0)}(t) = \int_{-\infty}^{\infty} b^{(i)}(x) b^{(0)}(x-t) dx$$

Illustration



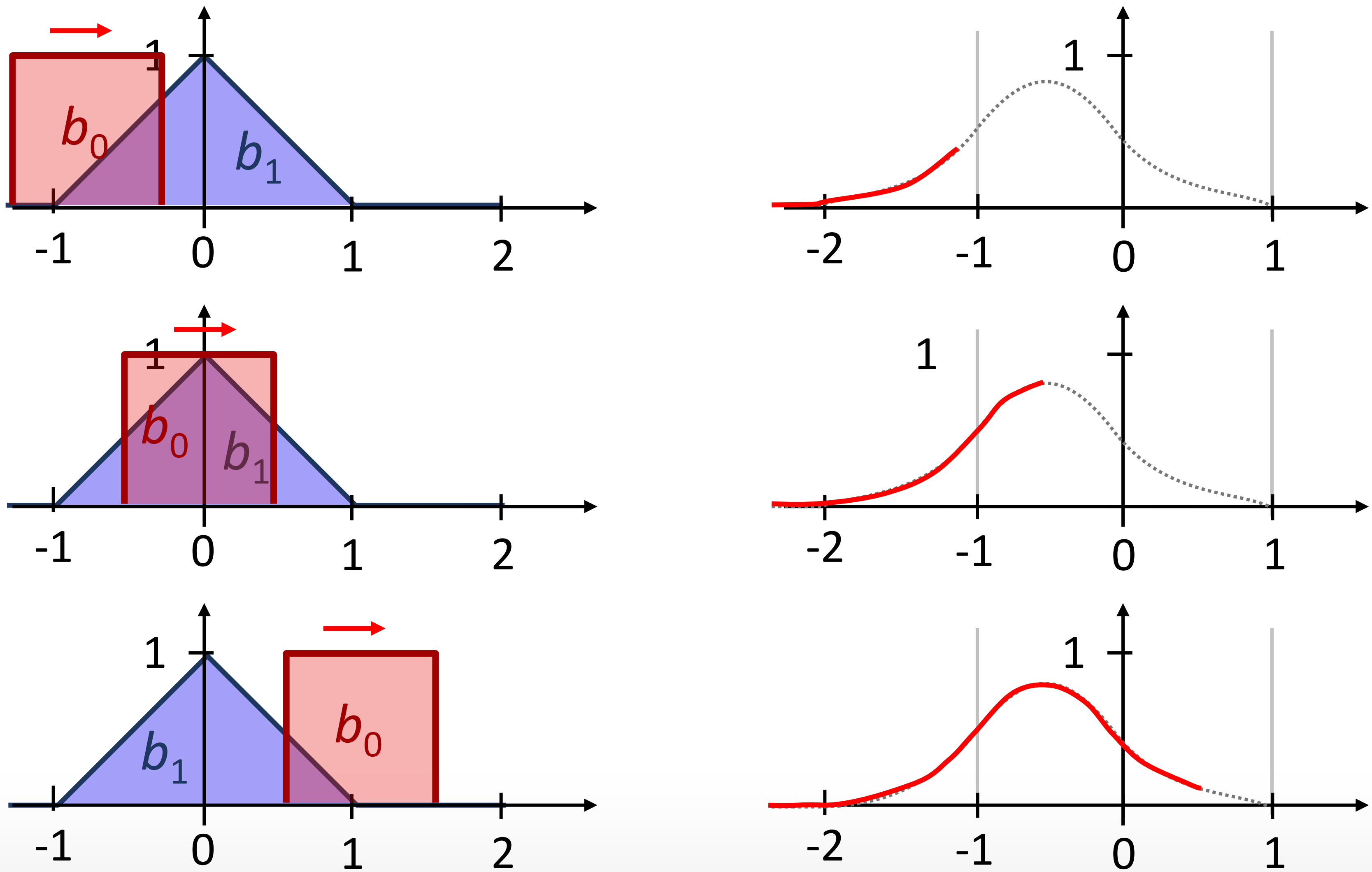
Illustration



Result:

- Piecewise linear B-spline basis function
- Each convolution with b_0 increases the continuity by 1.

Illustration



Smoothness

Convolution with a box filter increases smoothness:

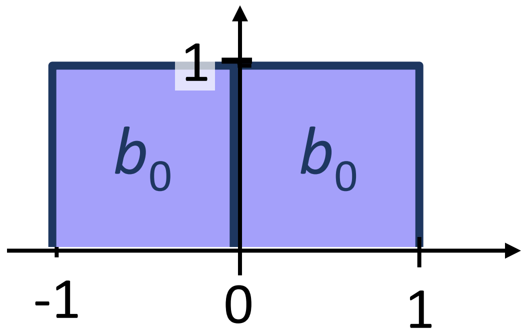
Function f that is $C^{k\text{-}1}$ at $t = t_{0,}$ and C^k everywhere else:

- $f_\otimes := f \otimes b_0 = \int_t^{t+1} f(t)dt = F(t+1) - F(t)$
- $\Delta D^i := \frac{d^i}{dt^i}\Big|_- f(t) - \frac{d^i}{dt^i}\Big|_+ f(t), \quad \Delta D^i f(t) = 0 \quad \text{for } i = 1..k-1$
- $\frac{d^j}{dt^j} f_\otimes(t) = \frac{d^{j-1}}{dt^{j-1}}\left(\frac{d}{dt} f_\otimes(t)\right) = \frac{d^{j-2}}{dt^{j-2}}\left(\frac{d}{dt}\left(F(t+1) - F(t)\right)\right)$

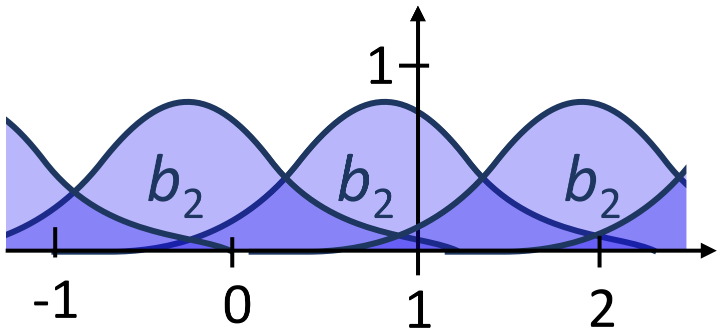
  $= \frac{d^{j-2}}{dt^{j-2}}\left(f(t+1) - f(t)\right)$
- $\Delta D^j f_\otimes(t) = \Delta D^{j-1}\left(f(t+1) - f(t)\right)$

Partition of Unity

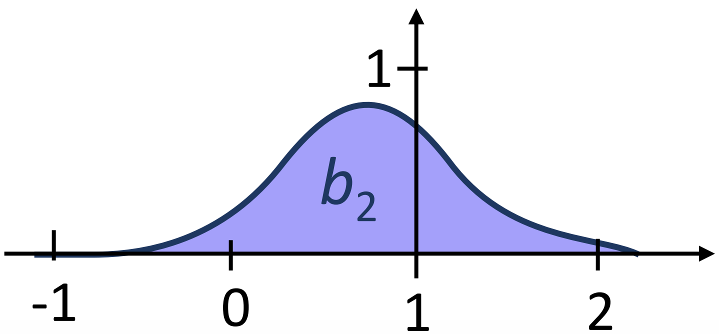
Proof by Induction:



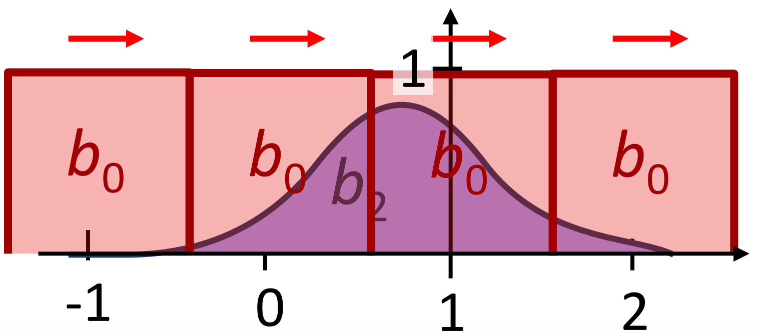
degree zero: **OK**



degree *d*: *given*



degree *d*...



degree *d*+1: *follows*

Other Properties

Positivity:

- By definition

Continuity:

- Smoothness increasing property: k-fold convolution is C^k.

Piecewise polynomial:

- Easy to see: Polynomial in each interval
- Degree k for k-fold convolution.

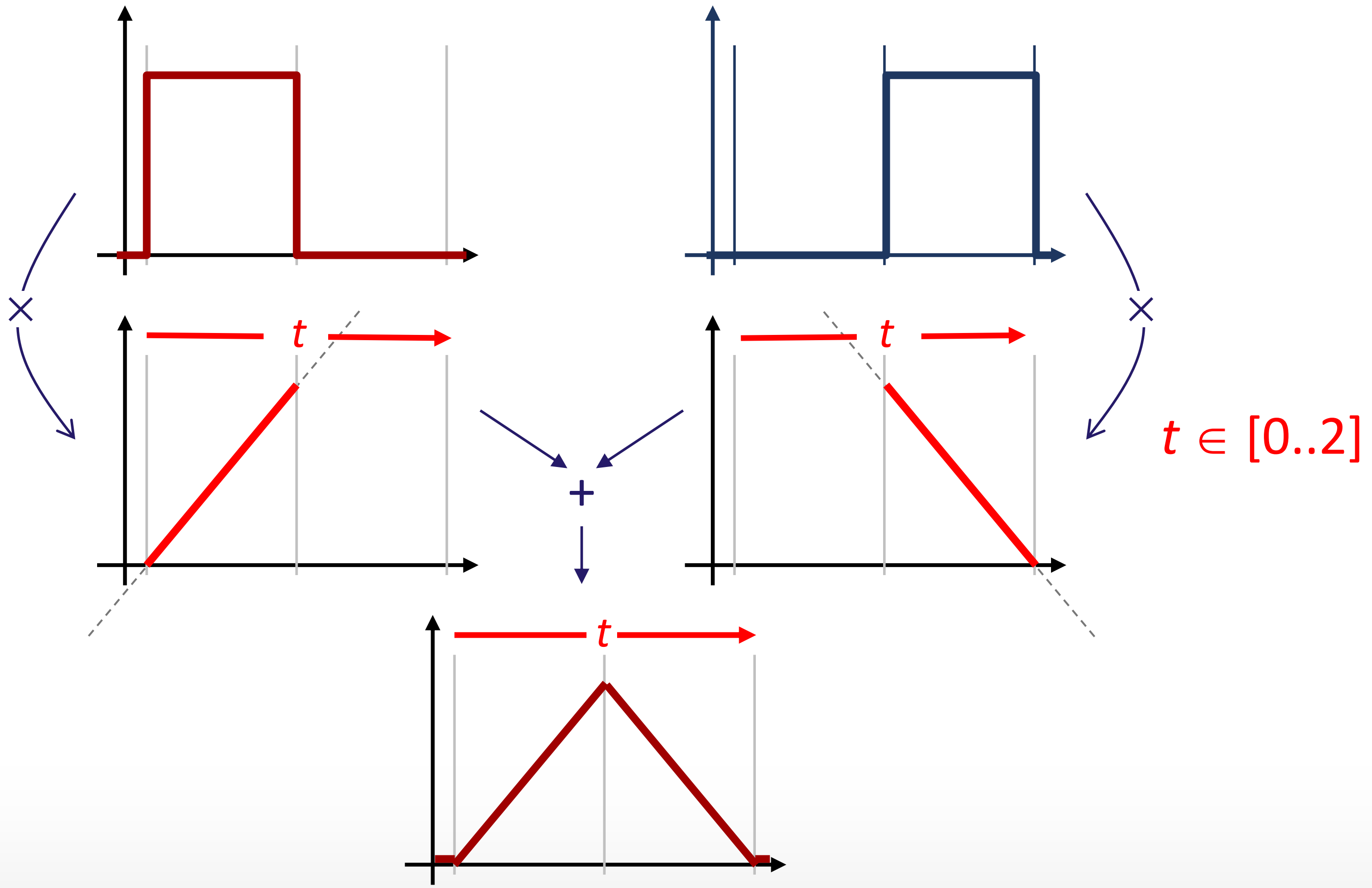
Consequences

Consequences:

- The constructed functions are identical to the explicitly constructed ones (limited degrees of freedom)
- This means, the explicitly constructed basis has the same properties (partition of unity, positivity)

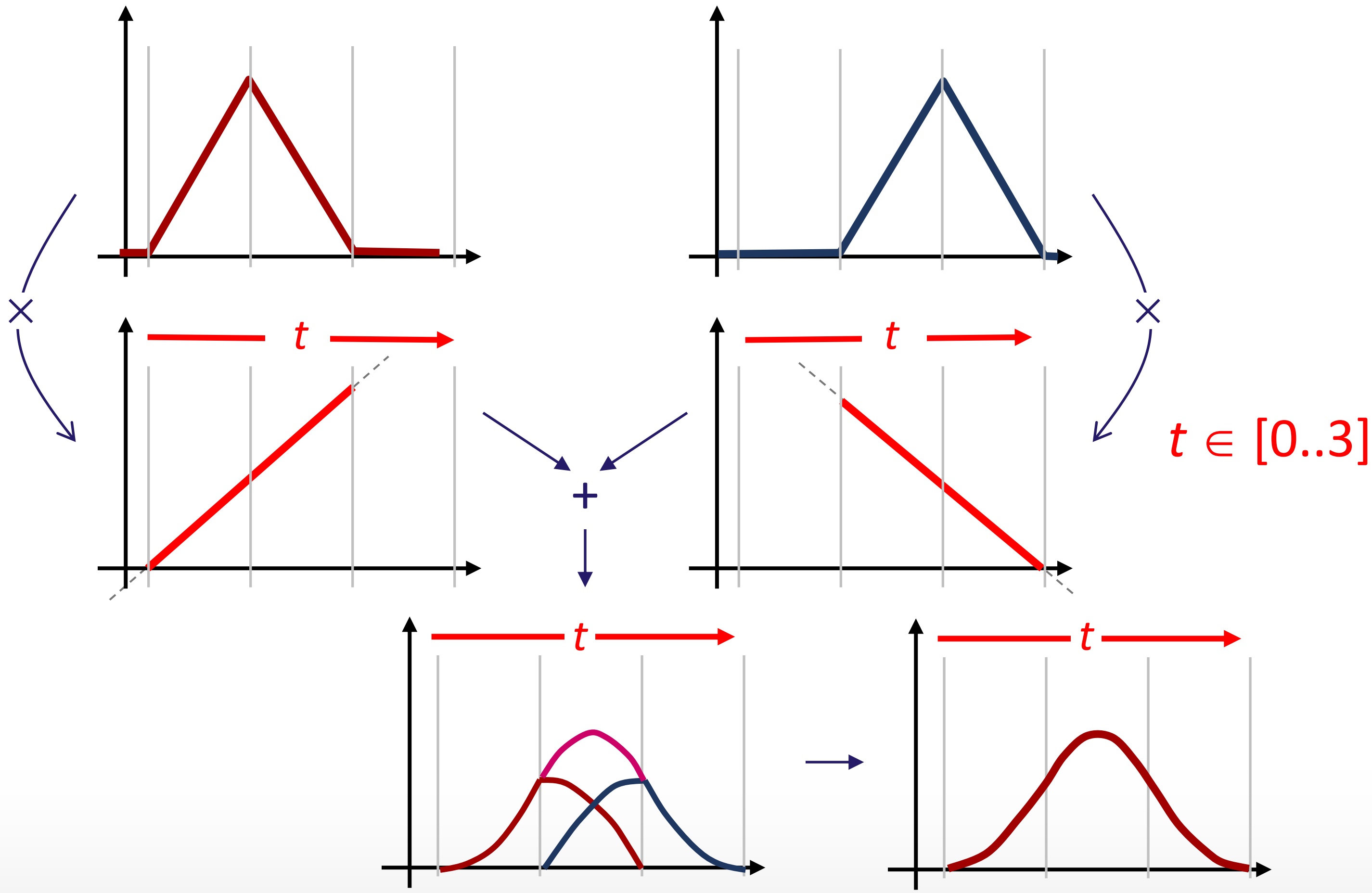
Repeated Linear Interpolation

Another way to increase smoothness:

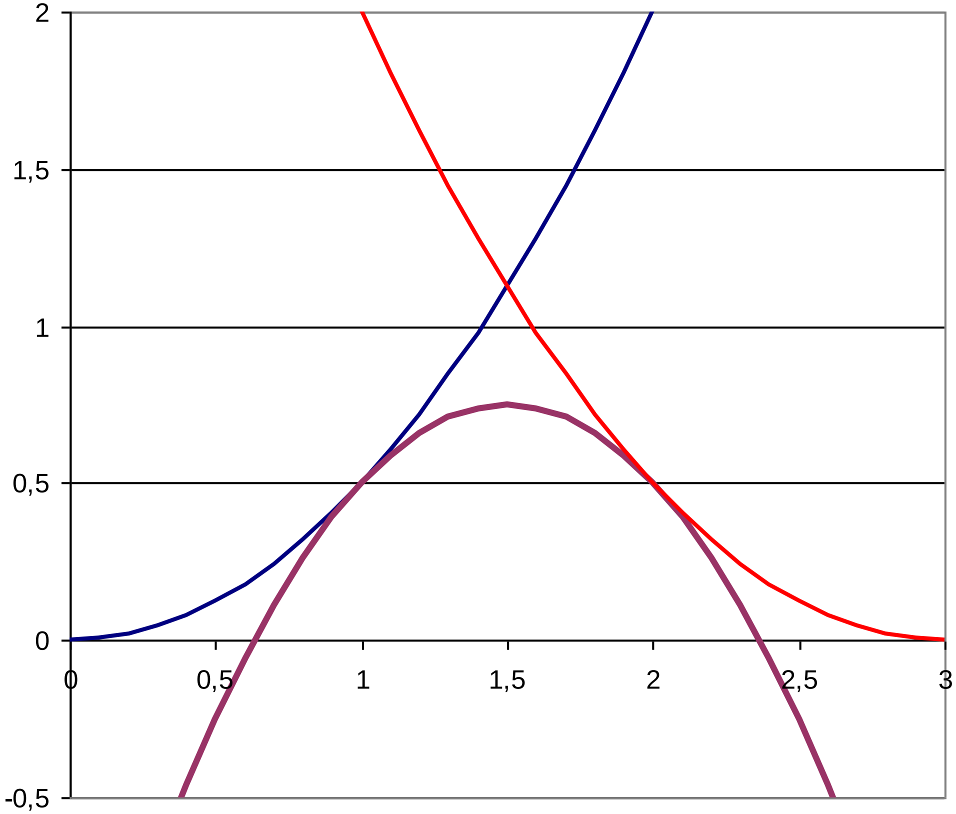


Repeated Linear Interpolation

Another way to increase smoothness:

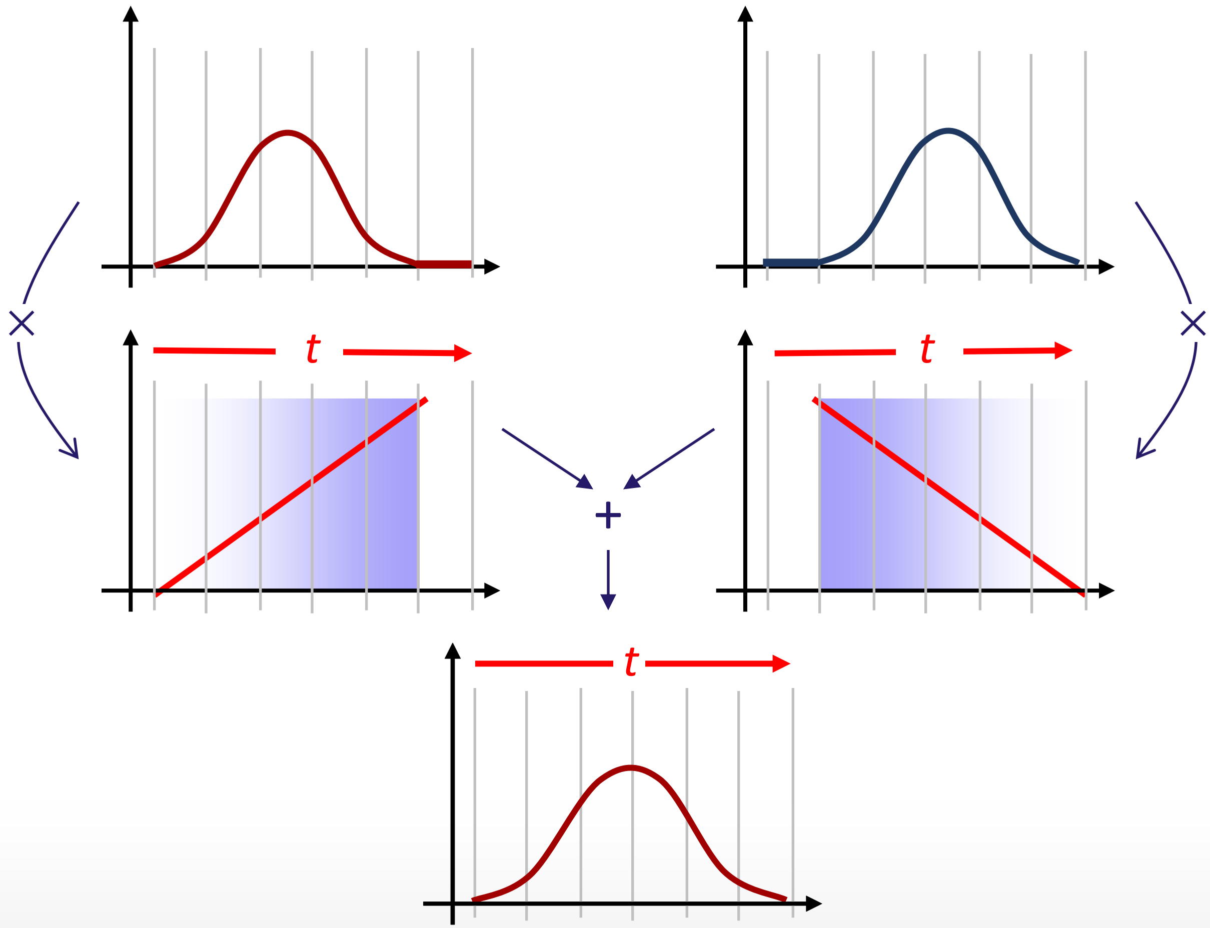


Plot of the Three Pieces



Repeated Linear Interpolation

General Principle:

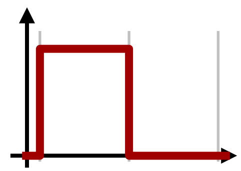


$t \in [0..d+1]$

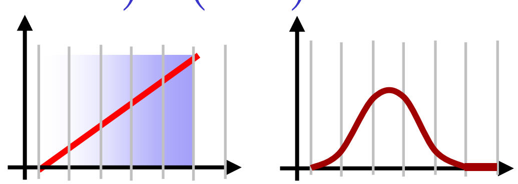
De Boor Recursion

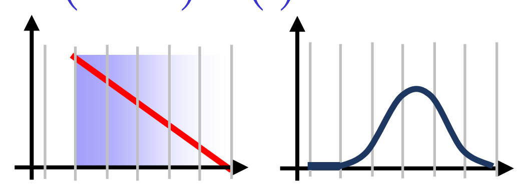
The uniform B-spline basis of degree *d* is given as:

$$N_i^0(t) = \begin{cases} 1, & \text{if } i-1 \le t < i \\ 0, & \text{otherwise} \end{cases}$$



$$N_i^d(t) = \frac{t-(i-1)}{(i+d-1)-(i-1)} N_i^{d-1}(t) + \frac{(i+d)-t}{(i+d)-(i)} N_{i+1}^{d-1}(t)$$





$$= \frac{t-i+1}{d} N_i^{d-1}(t) + \frac{i+d-t}{d} N_i^{d-1}(t)$$

Connection

Three constructions:

- It can be shown that this construction is equivalent to the previous two.
- Rough idea:
  - Convolution with a box filter increases degree by one (multiplication with linear weight in de Boor formula)
  - Antiderivative of box filter: evaluate at t, $t+1$ (corresponds to overlaying N_i, N_{i+1})
  - Need to show that the coefficients match (induction)

Polynomial Spline Curves

Non-Uniform B-Splines

Generalization

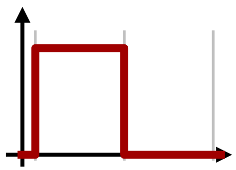
De Boor formula can be generalized:

- Arbitrary parameter sequences ("*knot sequence*") $(t_0, t_1, \ldots, t_n)$ with $t_0 \leq t_1 \leq \ldots \leq t_n$
  - Not necessary to have uniform spacing
  - Will give us more flexibility in curve design.
- Specifying one parameter value multiple times is permitted, e.g. (0, 0, 0, 1, 2, 3, 4, 4, 4, 5, 6, 7, 8, 8, 8)
  - This has a similar effect as multiple nodes in simple uniform B-Splines
  - Will avoid irregular parametrization (means: will create a basis that itself is less smooth, not just the traced out curve)

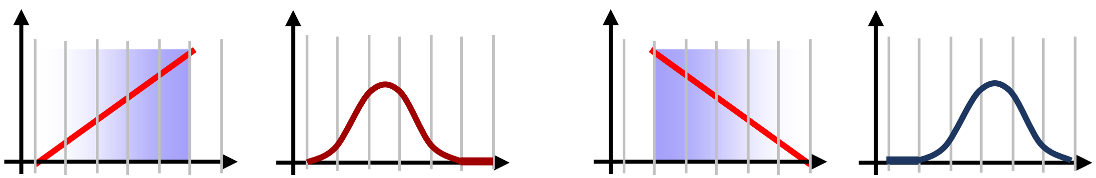
General De Boor Recursion

Generalized Formula:

$$N_i^0(t) = \begin{cases} 1, & \text{if } t_{i-1} \leq t < t_i \\ 0, & \text{otherwise} \end{cases}$$



$$N_i^d(t) = \frac{t - t_{i-1}}{t_{i+d-1} - t_{i-1}} N_i^{d-1}(t) \qquad + \qquad \frac{t_{i+d} - t}{t_{i+d} - t_i} N_{i+1}^{d-1}(t)$$



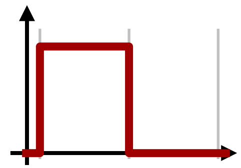
Remark:

- If a knot value is repeated *d* times, the denominator may vanish
- In this case: The fraction is treated as a zero

Uniform Case:

For comparison – the uniform case:

$$N_i^0(t) = \begin{cases} 1, & \text{if } i-1 \le t < i \\ 0, & \text{otherwise} \end{cases}$$



$$N_i^d(t) = \frac{t-(i-1)}{(i+d-1)-(i-1)} N_i^{d-1}(t) \qquad + \qquad \frac{(i+d)-t}{(i+d)-(i)} N_{i+1}^{d-1}(t)$$

B-Splines

Constructing a Spline Curve:

- Choose a degree d.
- Choose a *knot sequence* $(t_0, t_1, \ldots, t_n)$ with $t_0 \leq t_1 \leq \ldots \leq t_n$ $(n \geq d - 1)$
- Choose control points $\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_m$ $(m = n - d + 1)$
- Form the spline curve:

$$\mathbf{f}(t) = \sum_{i=0}^{n} N_i^d(t)\mathbf{p}_i$$

De Boor Algorithm:

- This evaluation can be expanded explicitly...

De Boor Algorithm

$$f(t) = \sum_{i=0}^{n} N_i^d(t)\mathbf{p}_i$$

$$= \sum_{i=0}^{n}\left(\frac{t-t_{i-1}}{t_{i+d-1}-t_{i-1}} N_i^{d-1}(t)\right)\mathbf{p}_i + \sum_{i=0}^{n}\left(\frac{t_{i+d}-t}{t_{i+d}-t_i} N_{i+1}^{d-1}(t)\right)\mathbf{p}_i$$

$$= \sum_{i=0}^{n}\left(\frac{t-t_{i-1}}{t_{i+d-1}-t_{i-1}} N_i^{d-1}(t)\right)\mathbf{p}_i + \sum_{i=1}^{n+1}\left(\frac{t_{i-1+d}-t}{t_{i+d-1}-t_{i-1}} N_i^{d-1}(t)\right)\mathbf{p}_{i-1}$$

$$= \sum_{i=0}^{n+1}\left(\left[\frac{t-t_{i-1}}{t_{i+d-1}-t_{i-1}} N_i^{d-1}(t)\right]\mathbf{p}_i + \left[\frac{t_{i-1+d}-t}{t_{i+d-1}-t_{i-1}} N_i^{d-1}(t)\right]\mathbf{p}_{i-1}\right)$$

$$= \sum_{i=0}^{n+1}\frac{(t-t_{i-1})\mathbf{p}_i + (t_{i-1+d}-t)\mathbf{p}_{i-1}}{t_{i+d-1}-t_{i-1}} N_i^{d-1}(t) \qquad \mathbf{p}_i^1 := \frac{(t-t_{i-1})\mathbf{p}_i + (t_{i-1+d}-t)\mathbf{p}_{i-1}}{t_{i+d-1}-t_{i-1}}$$

$$= \sum_{i=0}^{n+1} N_i^{d-1}(t)\mathbf{p}_i^1$$

De Boor Algorithm

$$f(t) = \sum_{i=0}^{n+1} N_i^{d-1}(t)\mathbf{p}_i^{(1)}$$

$$= \sum_{i=0}^{n+j} N_i^{d-1-j}(t)\mathbf{p}_i^{(j)} \quad \text{with: } \mathbf{p}_i^{(j)} = \frac{(t - t_{i-1})}{t_{i+d-j-1} - t_{i-1}}\mathbf{p}_i^{(j-1)} + \frac{(t_{i-1+d-j} - t)}{t_{i+d-j-1} - t_{i-1}}\mathbf{p}_{i-1}^{(j-1)}$$

$$= \sum_{i=0}^{n+d} N_i^{0}(t)\mathbf{p}_i^{(d)}$$

This means:

- For $t \in [t_{i-1},...,t_i)$, we obtain the function value $f(t)$.
- We can write this as an algorithm...

De Boor Algorithm

De Boor Algorithm:

- We want to evaluate $f(t)$ for a $t \in [t_{i-1},...,t_i)$
- We compute:

$$\mathbf{p}_i^{(0)} = \mathbf{p}_i$$

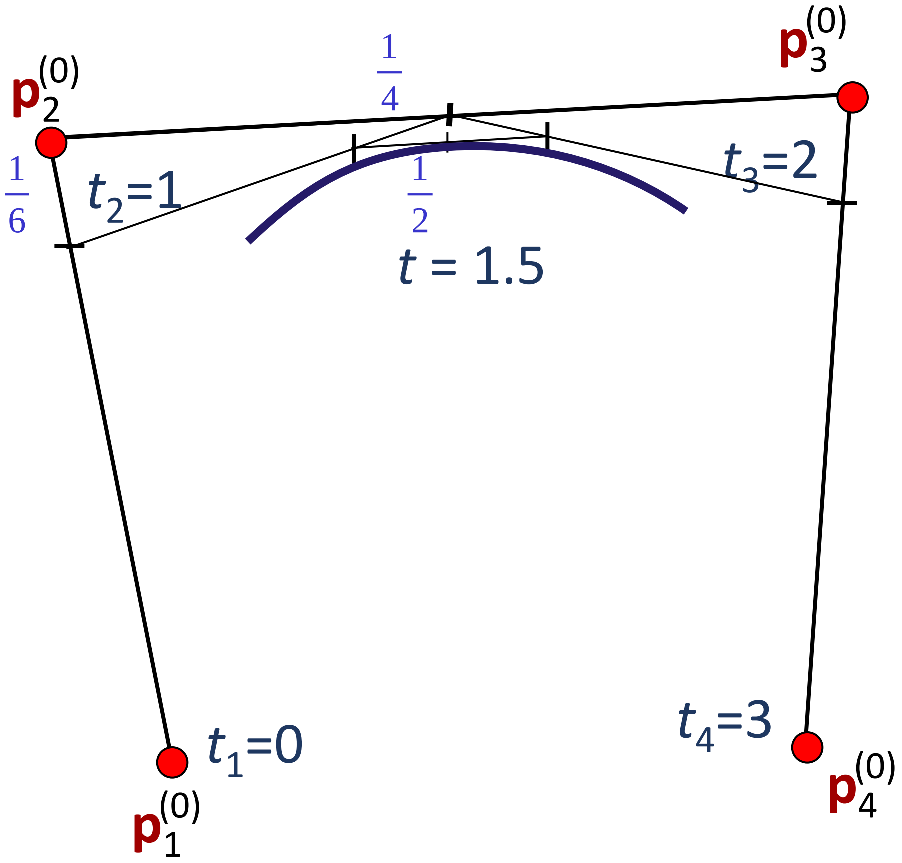
For increasing j:

$$\mathbf{p}_i^{j}(t) = \frac{(t - t_{i-1})}{t_{i+d-j-1} - t_{i-1}} \mathbf{p}_i^{j-1}(t) + \frac{(t_{i-1+d-j} - t)}{t_{i+d-j-1} - t_{i-1}} \mathbf{p}_{i-1}^{j-1}(t)$$

$$= \alpha_i^{(j)} \mathbf{p}_i^{j-1}(t) + \left(1 - \alpha_i^{(j)}\right) \mathbf{p}_{i-1}^{j-1}(t), \quad \alpha_i^{(j)} = \frac{(t - t_{i-1})}{t_{i+d-j-1} - t_{i-1}}$$

Output $\mathbf{p}_i^{d-1}(t)$

Example



$\mathbf{p}_i^{(0)} = \mathbf{p}_i$

For increasing j:

$$\mathbf{p}_i^j(t) = \frac{(t - t_{i-1})}{t_{i+d-j-1} - t_{i-1}} \mathbf{p}_i^{j-1}(t) + \frac{(t_{i-1+d-j} - t)}{t_{i+d-j-1} - t_{i-1}} \mathbf{p}_{i-1}^{j-1}(t)$$

$$= \alpha_i^{(j)} \mathbf{p}_i^{j-1}(t) + \left(1 - \alpha_i^{(j)}\right) \mathbf{p}_{i-1}^{j-1}(t), \quad \alpha_i^{(j)} = \frac{(t - t_{i-1})}{t_{i+d-j-1} - t_{i-1}}$$

Output $\mathbf{p}_i^{d-1}(t)$

Data Flow

Data Flow:

$$
\begin{array}{ccccccccc}
\mathbf{p}_{i-d}^{(0)} & & & & & & & & \\
\mathbf{p}_{i-d+1}^{(0)} & \overset{\searrow}{\rightarrow} & \mathbf{p}_{i-d+1}^{(1)} & & & & & & \\
\vdots & & \vdots & & \ddots & & & & \\
\mathbf{p}_{i-1}^{(0)} & \overset{\searrow}{\rightarrow} & \mathbf{p}_{i-1}^{(1)} & \overset{\searrow}{\rightarrow} & \cdots & & \mathbf{p}_{i-1}^{(d-2)} & & \\
\mathbf{p}_{i}^{(0)} & \overset{\searrow}{\rightarrow} & \mathbf{p}_{i}^{(1)} & \overset{\searrow}{\rightarrow} & \cdots & & \mathbf{p}_{i}^{(d-2)} & \overset{\searrow}{\rightarrow} & \mathbf{p}_{i}^{(d-1)}
\end{array}
$$

De Boor Algorithm

Some nice properties:

- $\mathbf{p}_i^j(t) = \alpha \mathbf{p}_i^{j-1}(t) + (1-\alpha)\mathbf{p}_{i-1}^{j-1}(t) \quad (\text{for some } \alpha)$
- The algorithm forms only convex combinations of the original data points.
  - Affine invariance follows directly
  - Numerically stable – no cancelation or error amplification problems
  - Much better than transforming to monomial basis
  - But slower: O(d^2) instead O(d) multiplications with Horner scheme in monomial basis

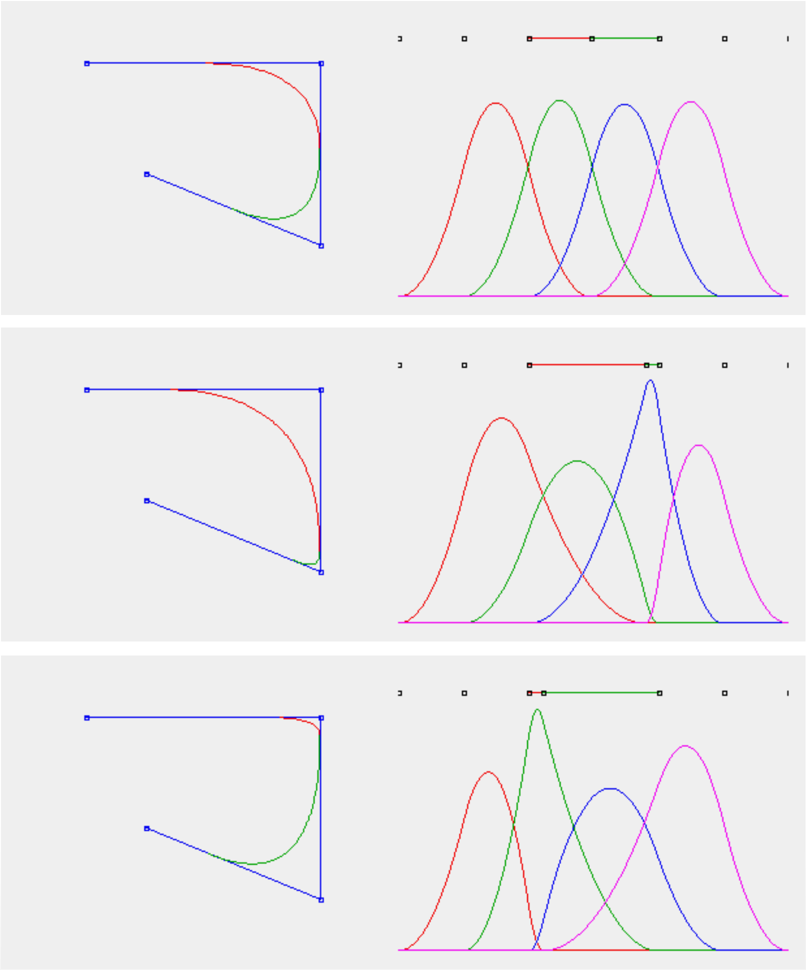
Benefits of Non-Uniform B-Splines

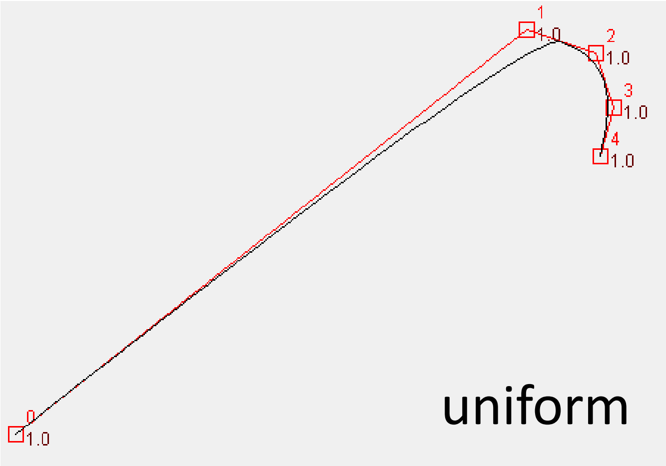
Improvements over the uniform case:

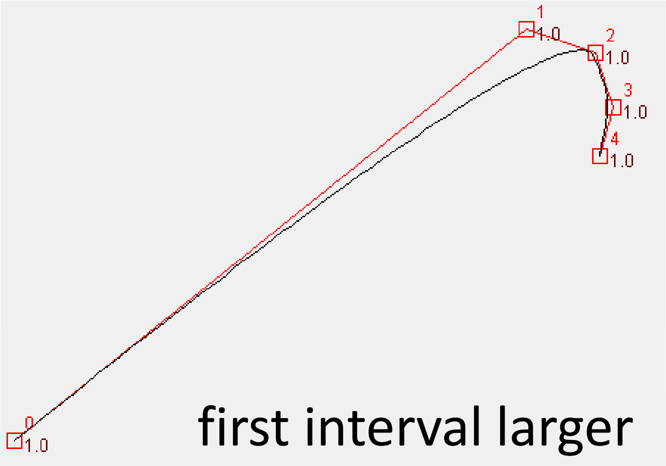
- We can choose the parameter intervals freely
  - Typically: Use distance between control points as knot distance
  - Allows better adaptation to distance between control points
  - Curves tend to "overshoot" less (in particular: problem for B-Spline interpolation; no convex hull property there)
  - Achieve more uniform speed for applications in animation
- No irregularities
  - Reduced smoothness, start/end conditions build into the computed bases
  - Advantages for rendering (no stopping at sharp corners)

Examples

The effect of parameter spacing:

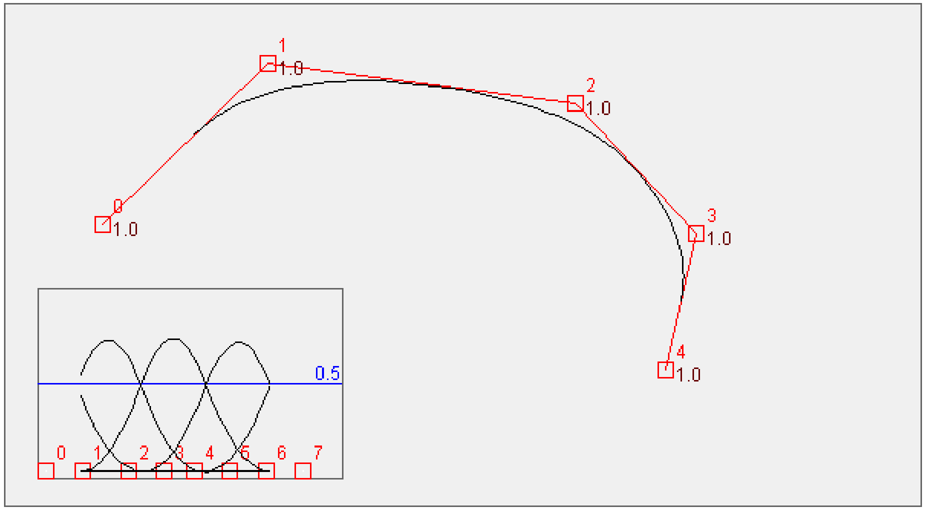


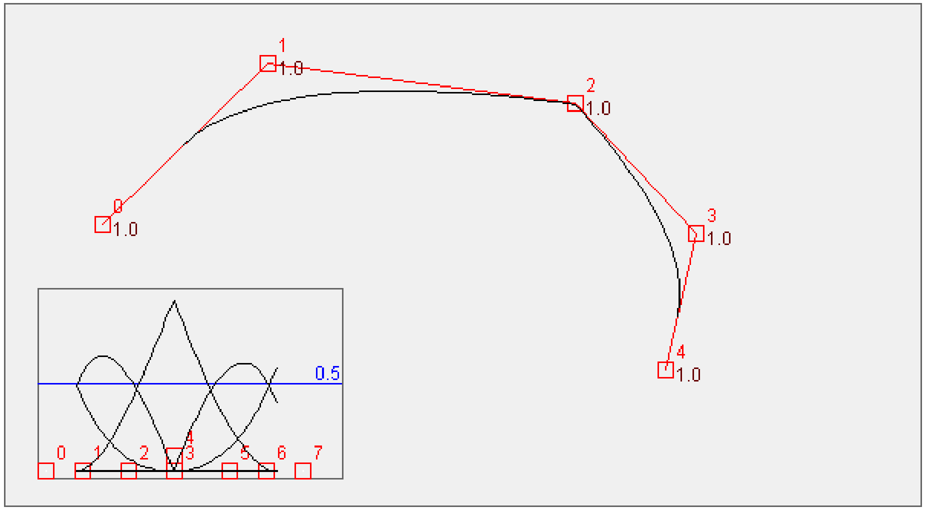




Examples

Multiplying Knots





A Closer Look at B-Splines

Need some more tools & properties:

- Proof various properties
- Operations: Knot insertion, degree elevation, etc.
- Convert to alternative bases (e.g. Bezier, monomials)

Problem:

- Indexing nightmare
- We need a better formalism to understand what's really going on: blossoming & polars
- We will look at that tool first, then go into the details again