EBERHARD KARLS UNIVERSITÄT TÜBINGEN

FACULTY OF SCIENCE

Computer Graphics

Computer Graphics (Graphische Datenverarbeitung)

- Splines -

Hendrik Lensch

WS 2021/2022

Corona

- Regular random lookup of the 3G certificates

- Contact tracing: We need to know who is in the class room
  - New ILIAS group for every lecture slot
  - Register via ILIAS or this QR code (only if you are present in this room)

Overview

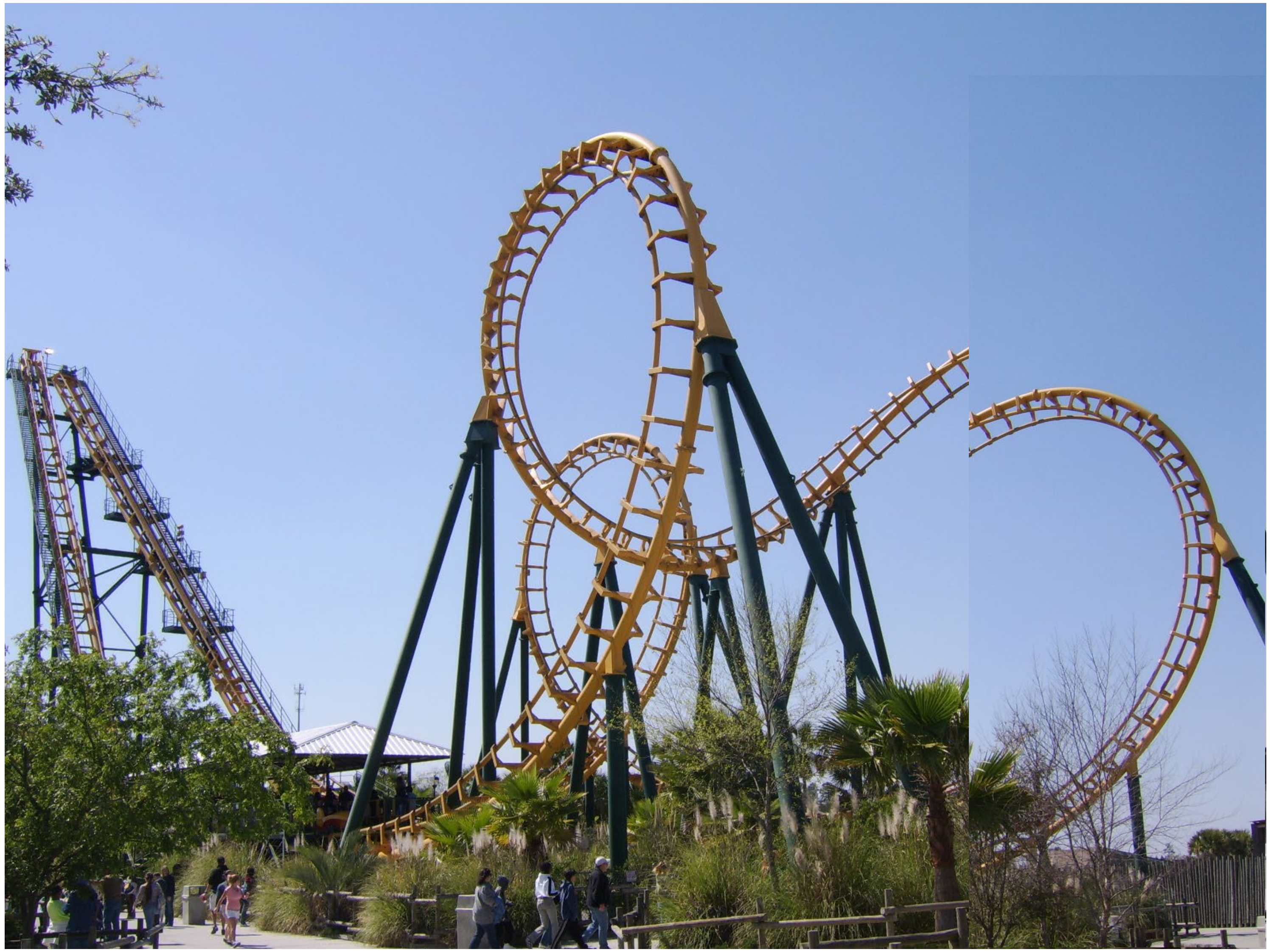
- Last Time
  - Open-GL

- Today
  - Parametric Curves
  - Lagrange Interpolation
  - Hermite Splines
  - Bézier Splines
  - DeCasteljau Algorithm
  - Parameterization

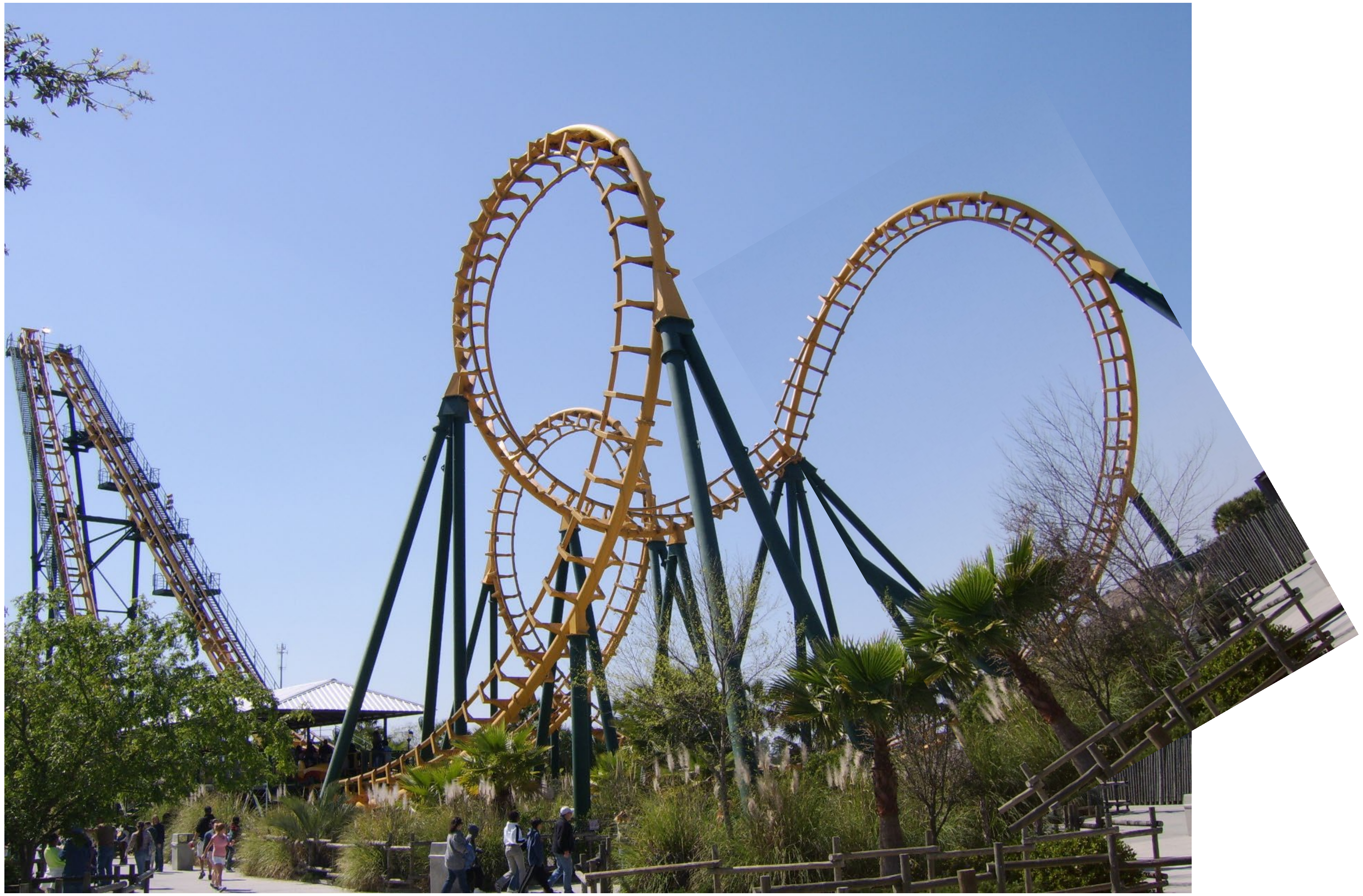
Curves

Roller Coaster

Roller Coaster – not good



Roller Coaster – not good



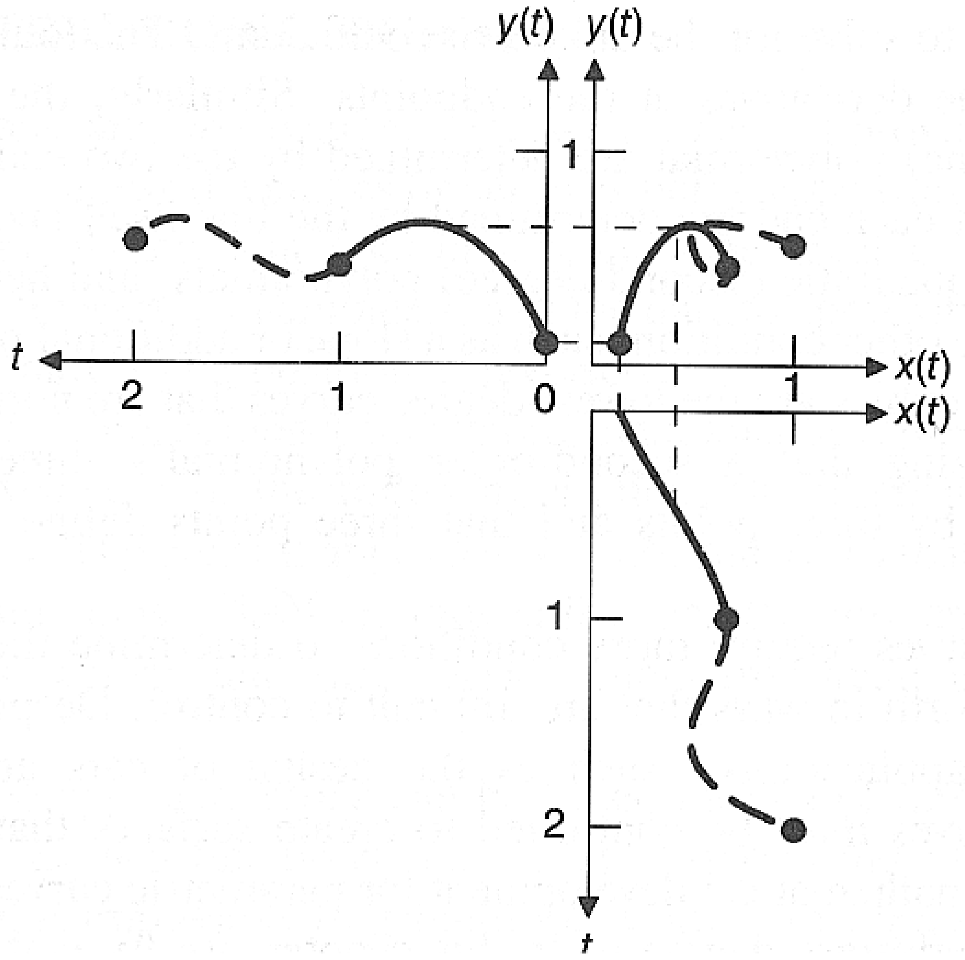
Curves

- Curve descriptions
  - Explicit
    - $y(x) = \pm \mathrm{sqrt}(r^2 - x^2)$, restricted domain
  - Implicit:
    - $x^2 + y^2 = r^2$ unknown solution set
  - Parametric:
    - $x(t) = r \cos(t)$, $y(t) = r \sin(t)$, $t \in [0, 2\pi]$
    - Flexibility and ease of use

- Polynomials
  - Avoids complicated functions (z.B. pow, exp, sin, sqrt)
  - Use simple polynomials of low degree

Parametric curves

- Separate function in each coordinate
  - 3D: f(t)= (x(t), y(t), z(t))



Monomials

- Monomial basis
  - Simple basis: 1, t, t^2, ... (t usually in [0 .. 1])
- Polynomial representation

$$x(t) = 3t^3 + 1t^2 - 2t + 4$$

$$\underline{P}(t) = \left(\underline{x}(t) \quad \underline{y}(t) \quad \underline{z}(t)\right) = \sum_{i=0}^{n} t^i \underline{A}_i$$

Degree (= Order – 1)

Coefficients $\in$ R³

Monomials

  - Coefficients can be determined from a sufficient number of constraints (e.g. interpolation of given points)
    - Given (n+1) parameter values t_i and points P_i
    - Solution of a linear system in the A_i − possible, but inconvenient
- Matrix representation

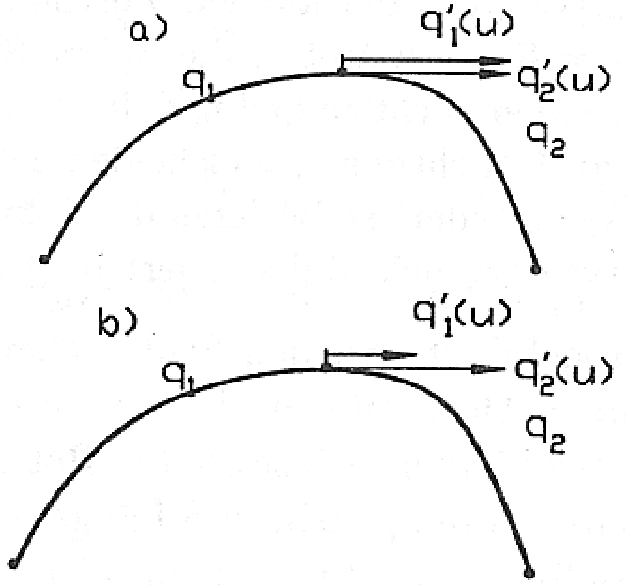
$$P(t) = \left(x(t) \;\; y(t) \;\; z(t)\right) = T(t)A \; = [t^n \quad t^{n-1} \quad \cdots \quad 1] \begin{bmatrix} A_{x,n} & A_{y,n} & A_{z,n} \\ A_{x,n-1} & A_{y,n-1} & A_{z,n-1} \\ & \vdots & \\ A_{x,0} & A_{y,0} & A_{z,0} \end{bmatrix}$$

Derivatives

- Derivative = tangent vector
  - Polynomial of degree (n-1)

$$P'(t) = \left(x'^{(t)} \;\; y'^{(t)} \; z'^{(t)}\right) = T'(t)A \;= [nt^{n-1} \quad (n-1)t^{n-2} \quad \cdots \quad 1 \quad 0]\begin{bmatrix} A_{x,n} & A_{y,n} & A_{z,n} \\ A_{x,n-1} & A_{y,n-1} & A_{z,n-1} \\ & \vdots & \\ A_{x,0} & A_{y,0} & A_{z,0} \end{bmatrix}$$

- Continuity and smoothness between parametric curves
  - $C^0 = G^0$ = same point
  - Parametric continuity C^1
    - Tangent vectors are identical
  - Geometric continuity G^1
    - Same direction of tangent vectors
  - Similar for higher derivatives



More on Continuity

- at one point:

- Geometric Continuity:
  - G^0: curves are joined
  - G^1: first derivatives are proportional at joint point, same direction but not necessarily same length
  - G^2: first and second derivatives are proportional

- Parametric Continuity:
  - C^0: curves are joined
  - C^1: first derivative equal
  - C^2: first and second derivatives are equal. If t is the time, this implies the acceleration is continuous.
  - C^n: all derivatives up to and including the nth are equal.

Lagrange Interpolation

- Interpolating basis functions
  - Lagrange polynomials for a set of parameters T={t_0, ..., t_n}

$$L_i^n(t) = \prod_{\substack{j=o \\ i \neq j}}^{n} \frac{t - t_j}{t_i - t_j}, \qquad L_i^n(t_j) = \delta_{ij} = \begin{cases} 1 & i = j \\ 0 & \end{cases}$$

- Properties
  - Good for interpolation at given parameter values
    - At each t_i: One basis function = 1, all others = 0
  - Polynomial of degree n (n factors linear in t)
- Lagrange Curves
  - Use Lagrange Polynomials with point coefficients

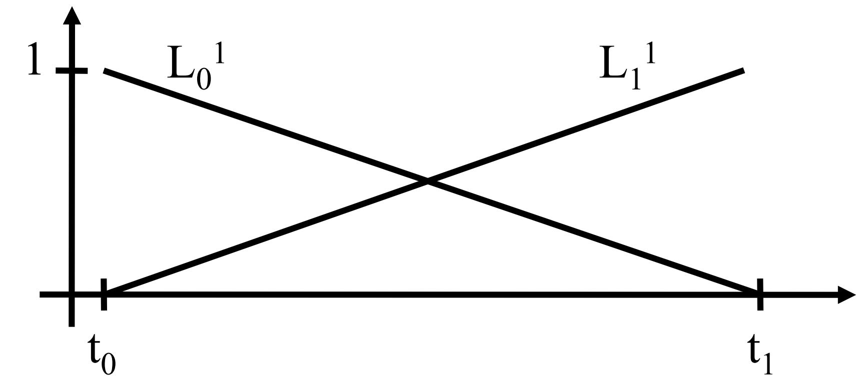
$$\underline{P}(t) = \sum_{i=0}^{n} L_i^n(t) \underline{P}_i$$

Lagrange Interpolation

- Simple Linear Interpolation
  - T={t_0, t_1}

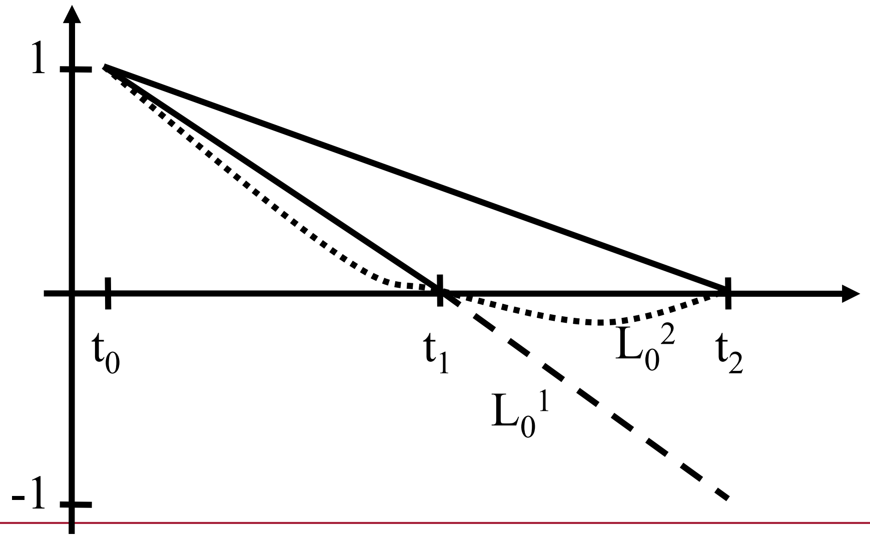
$$L_0^1(t) = \frac{t - t_1}{t_0 - t_1}$$

$$L_1^1(t) = \frac{t - t_0}{t_1 - t_0}$$



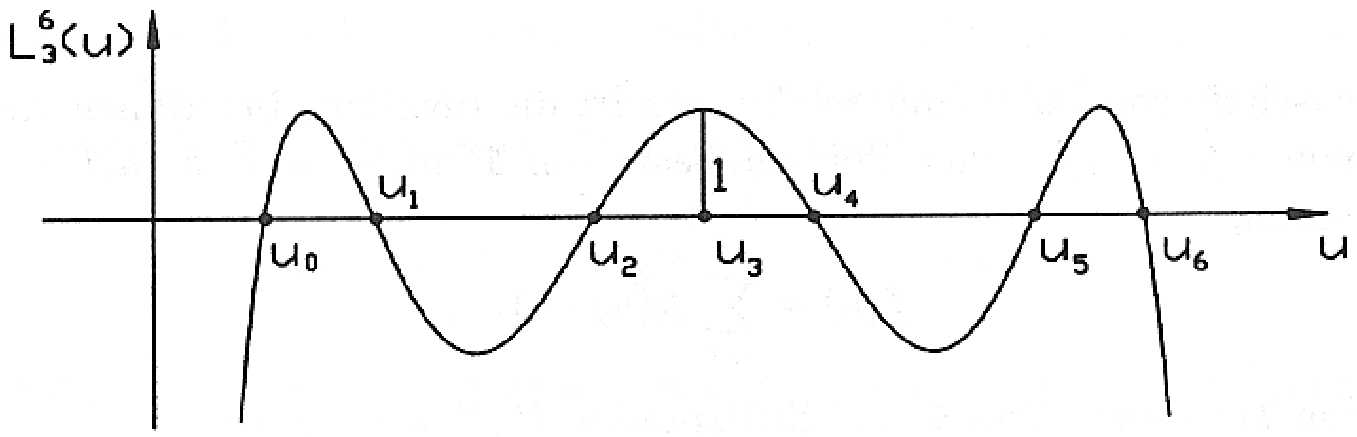
- Simple Quadratic Interpolation
  - T={t_0, t_1, t_2}

$$L_0^2(t) = \frac{t - t_1}{t_0 - t_1}\frac{t - t_2}{t_0 - t_2}$$



Problems

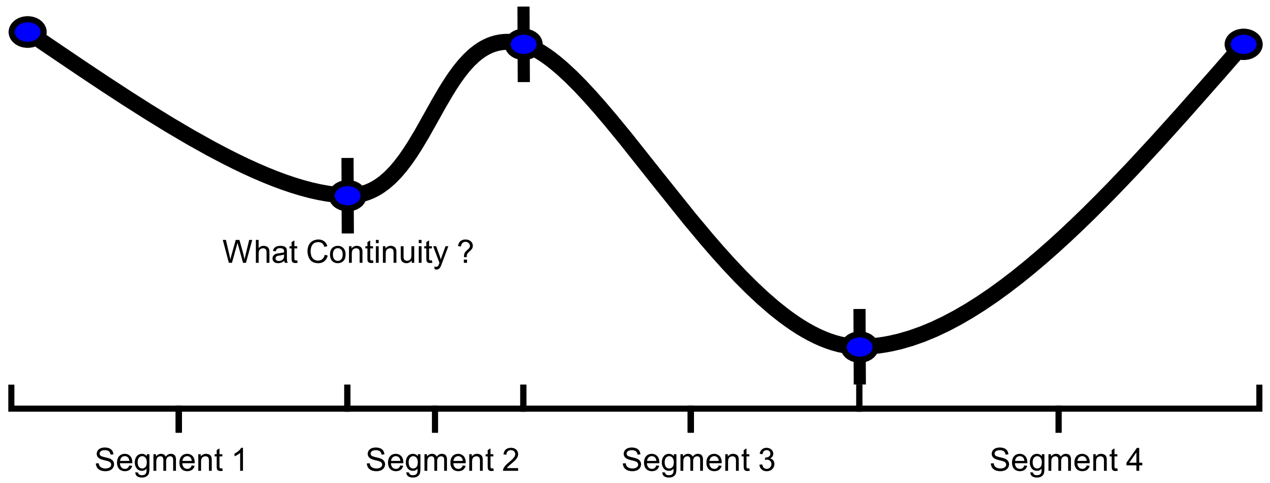
- Problems with a single polynomial
  - Degree depends on the number of interpolation constraints
  - Strong overshooting for high degree (n > 7)
  - Problems with smooth joints
  - Numerically unstable
  - No local changes



Splines

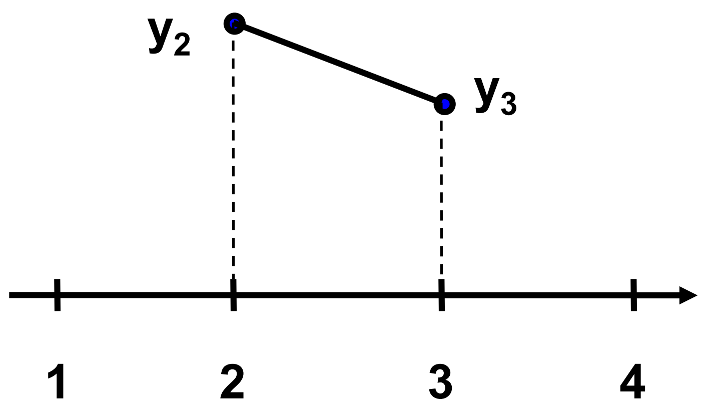
Splines

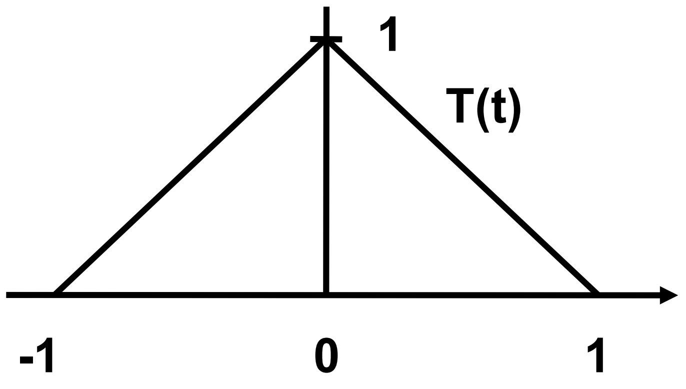
- Functions for interpolation & approximation
  - Standard curve and surface primitives in geometric modeling
  - Key frame and in-betweens in animations
  - Filtering and reconstruction of images
- Historically
  - Name for a tool in ship building
    - Flexible metal strip that tries to stay straight
  - Within computer graphics:
    - Piecewise polynomial function

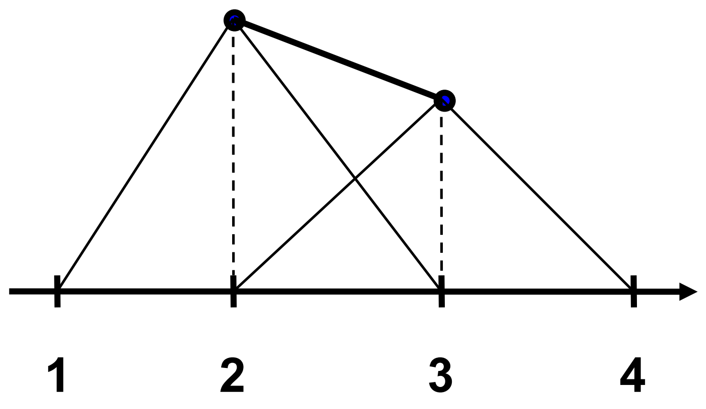


Linear Interpolation

- Hat functions and linear splines
- Piecewise linear function



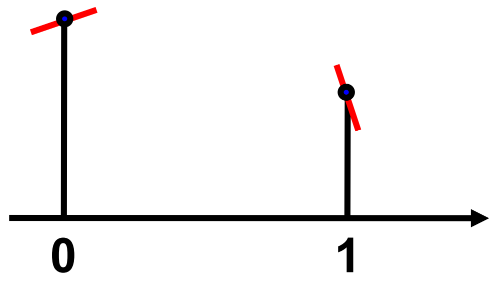




$$P(t) = T_2(t)y_2 + T_3(t)y_3$$

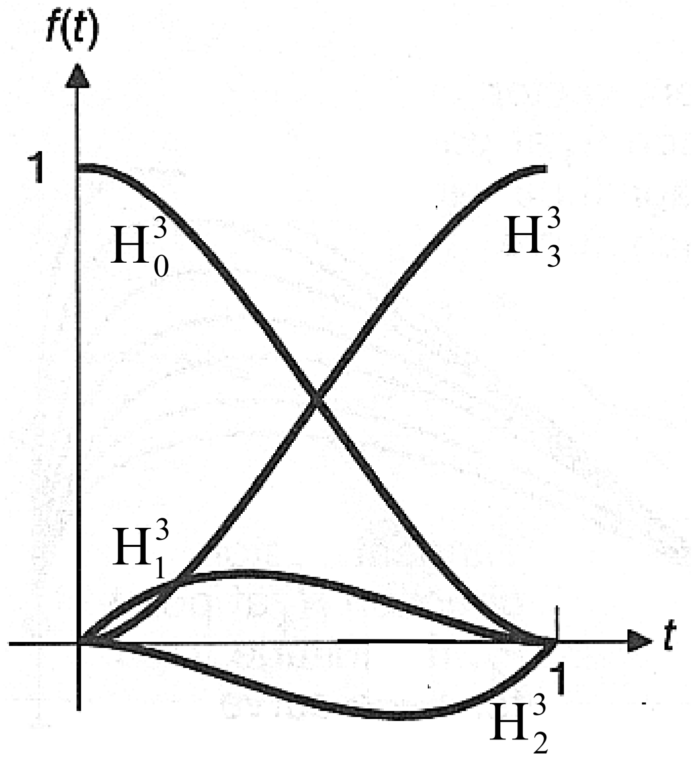
Hermite Interpolation

- Hermite Basis (cubic)
  - Interpolation of position P and tangent P´ information for t= {0, 1}



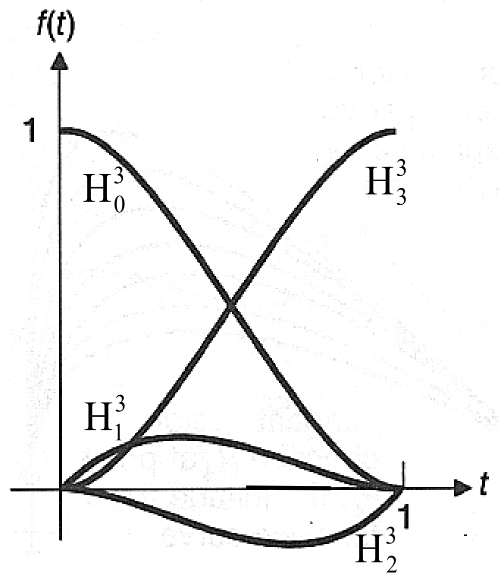
  - Basis functions

$$H_0^3(t) = (1-t)^2(1+2t)$$
$$H_1^3(t) = t(1-t)^2$$
$$H_2^3(t) = -t^2(1-t)$$
$$H_3^3(t) = (3-2t)t^2$$



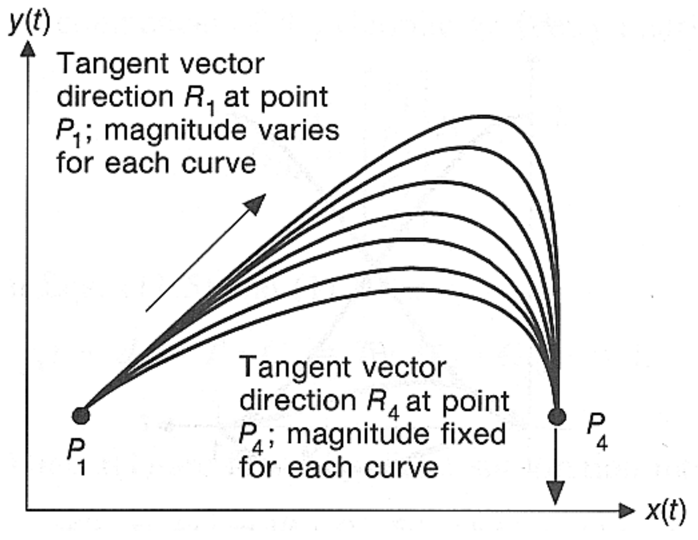
Hermite Interpolation

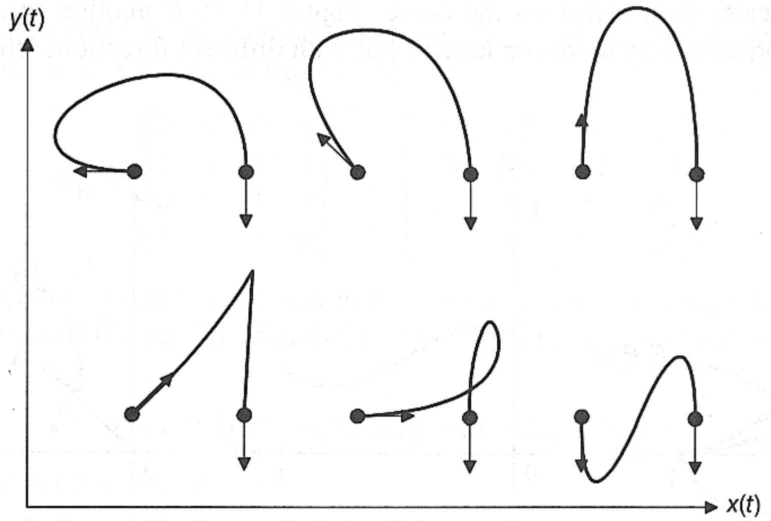
- Properties of Hermite Basis Functions
  - H_0 (H_3) interpolates smoothly from 1 to 0 (1 to 0)
  - H_0 and H_3 have zero derivative at t= 0 and t= 1
    - No contribution to derivative (H_1, H_2)
  - H_1 and H_2 are zero at t= 0 and t= 1
    - No contribution to position (H_0, H_3)
  - H_1 (H_2) has slope 1 at t= 0 (t= 1)
    - Unit factor for specified derivative vector
- Hermite polynomials
  - P_0, P_1 are positions $\in R^3$
  - P'_0, P'_1 are derivatives (tangent vectors) $\in R^3$

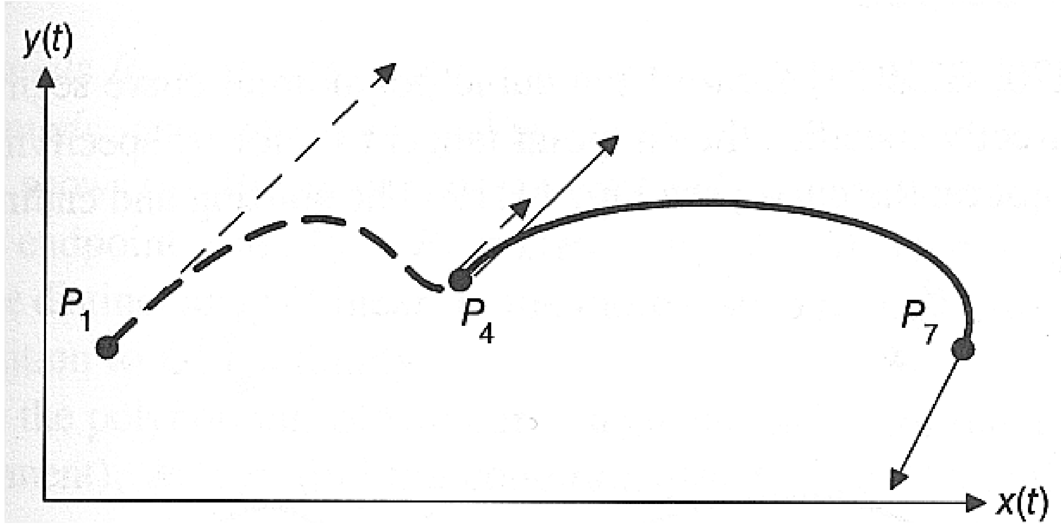


$$\underline{P}(t) = P_0 H_0^3(t) + P_0{}' H_1^3(t) + P_1{}' H_2^3(t) + P_1 H_3^3(t)$$

Examples: Hermite Interpolation


y(t)
Tangent vector direction R_1 at point P_1; magnitude varies for each curve
Tangent vector direction R_4 at point P_4; magnitude fixed for each curve
P_1
P_4
x(t)





Matrix Representation

- Matrix representation

$$P(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} A_{x,n} & A_{y,n} & A_{z,n} \\ A_{x,n-1} & A_{y,n-1} & A_{z,n-1} \\ & \vdots & \\ A_{x,0} & A_{y,0} & A_{z,0} \end{bmatrix}$$

$$= \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \underbrace{\begin{bmatrix} M_{11} & M_{12} & M_{13} & M_{14} \\ M_{21} & \ddots & & \\ & & & \\ & & & \end{bmatrix}}_{\text{basis matrix M(4x4)}} \underbrace{\begin{bmatrix} G_{x,3} & G_{y,3} & G_{z,3} \\ G_{x,2} & G_{y,2} & G_{z,2} \\ G_{x,1} & G_{y,1} & G_{z,1} \\ G_{x,0} & G_{y,0} & G_{z,0} \end{bmatrix}}_{\text{geometry matrix G(4x3)}}$$

$$= \underbrace{\begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} M_{11} & M_{12} & M_{13} & M_{14} \\ M_{21} & \ddots & & \\ & & & \\ & & & \end{bmatrix}}_{\text{basis functions}} \begin{bmatrix} P_0^T \\ P_1^T \\ P_2^T \\ P_3^T \end{bmatrix}$$

Matrix Representation

- For cubic Hermite interpolation we obtain by evaluating [t^3 t^2 t^1 t^0] or [$3t^2$ $2t^1$ $1t^0$ 0] (derivative):

$$\begin{aligned} P_0^T &= (0 \quad 0 \quad 0 \quad 1)\mathbf{M}_H\,\mathbf{G}_H \\ P_1^T &= (1 \quad 1 \quad 1 \quad 1)\mathbf{M}_H\,\mathbf{G}_H \\ P_0'^T &= (0 \quad 0 \quad 1 \quad 0)\mathbf{M}_H\,\mathbf{G}_H \\ P_1'^T &= (3 \quad 2 \quad 1 \quad 0)\mathbf{M}_H\,\mathbf{G}_H \end{aligned} \quad \textbf{or} \quad \begin{pmatrix} P_0^T \\ P_1^T \\ P_0'^T \\ P_1'^T \end{pmatrix} = \mathbf{G}_H = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix} \mathbf{M}_H \mathbf{G}_H$$

- Solution:
  - Two matrices must multiply to identity

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

Matrix Representation

- For cubic Hermite interpolation we obtain by evaluating $[t^3\ t^2\ t^1\ t^0]$ or $[3t^2\ 2t^1\ 1t^0\ 0]$ (derivative):

$$\begin{aligned} P_0^T &= (0\quad 0\quad 0\quad 1)\mathbf{M}_H\,\mathbf{G}_H \\ P_1^T &= (1\quad 1\quad 1\quad 1)\mathbf{M}_H\,\mathbf{G}_H \\ P_0'^T &= (0\quad 0\quad 1\quad 0)\mathbf{M}_H\,\mathbf{G}_H \\ P_1'^T &= (3\quad 2\quad 1\quad 0)\mathbf{M}_H\mathbf{G}_H \end{aligned}$$

or

$$\begin{pmatrix} P_0^T \\ P_1^T \\ P_0'^T \\ P_1'^T \end{pmatrix} = \mathbf{G}_H = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix} \mathbf{M}_H \mathbf{G}_H$$

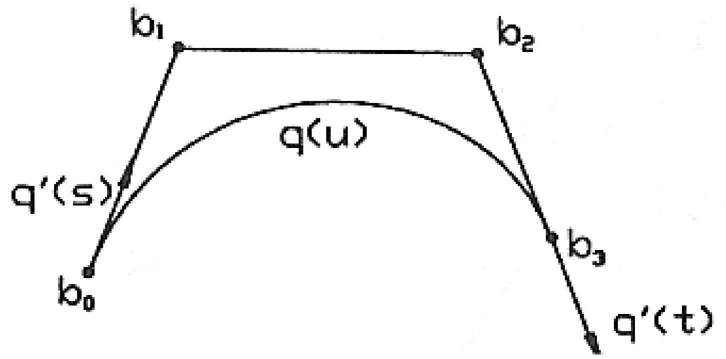
- Solution:
  - Two matrices must multiply to identity

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

$$\begin{aligned} H_0^3(t) &= (1-t)^2(1+2t) \\ &= 2t^3 - 3t^2 + 1 \\ H_1^3(t) &= t(1-t)^2 \\ H_2^3(t) &= -t^2(1-t) \\ H_3^3(t) &= (3-2t)t^2 \end{aligned}$$

Bézier

- Bézier Basis [deCasteljau´59, Bézier´62]
  - Different curve representation
  - Start and end point
  - 2 point that are approximated by the curve (cubics)
  - $P'_0 = 3(b_1 - b_0)$ and $P'_1 = 3(b_3 - b_2)$
    - Factor 3 due to derivative of t^3



$$G_H = \begin{bmatrix} P_0^T \\ P_1^T \\ P'^T_0 \\ P'^T_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 \end{bmatrix} \begin{bmatrix} b_0^T \\ b_1^T \\ b_2^T \\ b_3^T \end{bmatrix} = M_{HB} G_B$$

Basis transformation

- Transformation - Bézier to Hermite
  - P(t)=T M_H G_H = T M_H (M_{HB} G_B) = T ($M_H M_{HB}$) G_B = T M_B G_B

$$M_B = M_H M_{HB} = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

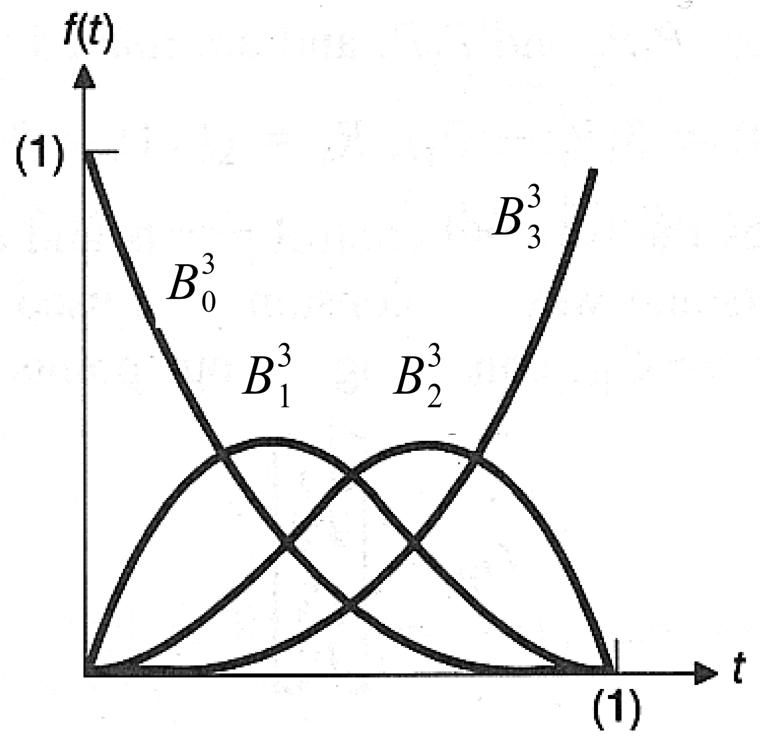
- Bézier Curves & Basis Functionss

$$\mathrm{P} = \sum_{i=0}^{3} B_i^3(t) b_i =$$

$$(1-t)^3 b_0 + 3t(1-t)^2 b_1 + 3t^2(1-t) b_2 + t^3 b_3$$

$$\mathrm{P(t)} = \sum_{i=0}^{n} B_i^n(t) b_i$$

$$with\ Basisfunctions \quad B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}$$

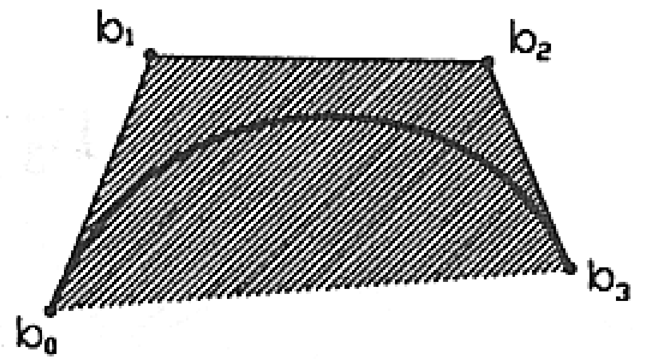


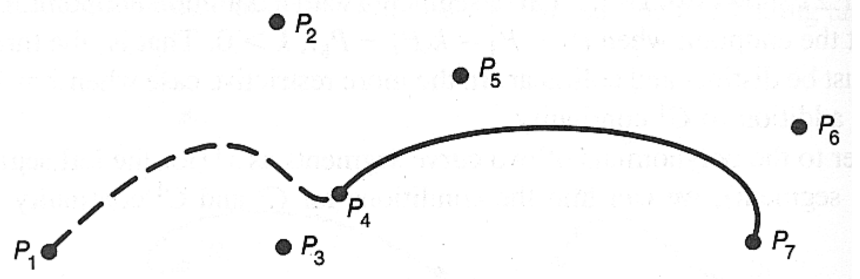
  - Basis functions: Bernstein polynomials

Properties: Bézier

- Advantages:
  - End point interpolation
  - Tangents explicitly specified
  - Smooth joints are simple
    - P_3, P_4, P_5 collinear $\rightarrow$ G^1 continuous
  - Geometric meaning of control points
  - Affine invariance

    $\forall \sum B_i(t) = 1$
  - Convex hull property
    - For $0<t<1$: $B_i(t) \geq 0$
  - Symmetry: $B_i(t) = B_{n-i}(1-t)$
- Disadvantages
  - Smooth joints need to be maintained explicitly
    - Automatic in B-Splines (and NURBS)





EBERHARD KARLS UNIVERSITÄT TÜBINGEN

DeCasteljau Algorithm

DeCasteljau Algorithm

- Direct evaluation of the basis functions
  - Simple but expensive
- Use recursion
  - Recursive definition of the basis functions

$$B_i^n(t) = tB_{i-1}^{n-1}(t) + (1-t)B_i^{n-1}(t) = \begin{pmatrix} n \\ i \end{pmatrix} t^i (1-t)^{n-i}$$

  - Inserting this once yields:

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

  - with the new Bézier points given by the recursion

$$b_i^k(t) = t\, b_{i+1}^{k-1}(t) + (1-t)b_i^{k-1}(t) \quad \text{a} \quad b_i^0(t) = b_i$$

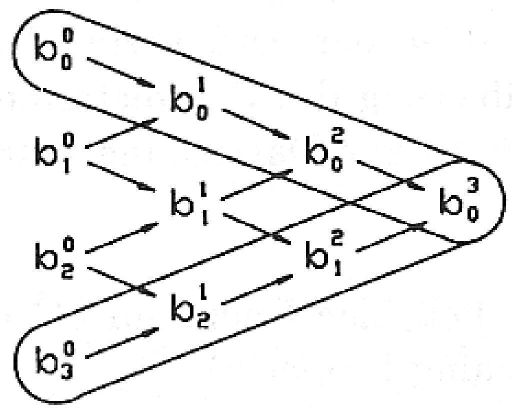
DeCasteljau Algorithm

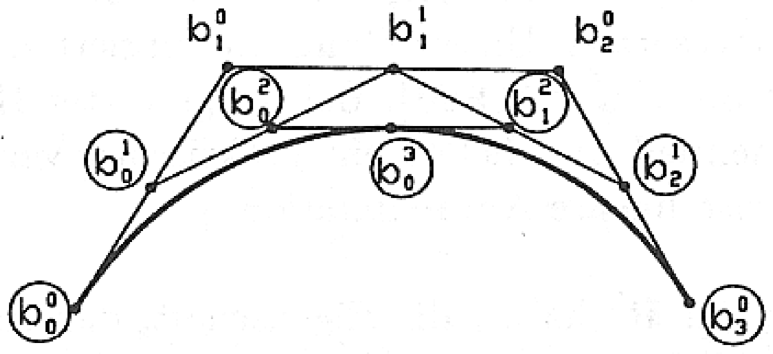
- DeCasteljau-Algorithm:
  - Recursive degree reduction of the Bezier curve by using the recursion formula for the Bernstein polynomials

$$P(t) = \sum_{i=0}^{n} b_i^0 B_i^n(t) = \sum_{i=0}^{n-1} b_i^1 B_i^{n-1}(t) = \cdots = b_i^n(t) \cdot 1$$

- Example:
  - t= 0.5

$$b_i^k(t)=\mathrm{t}b_{i+1}^{k-1}(t) + (1-\mathrm{t})b_i^{k-1}(t)$$





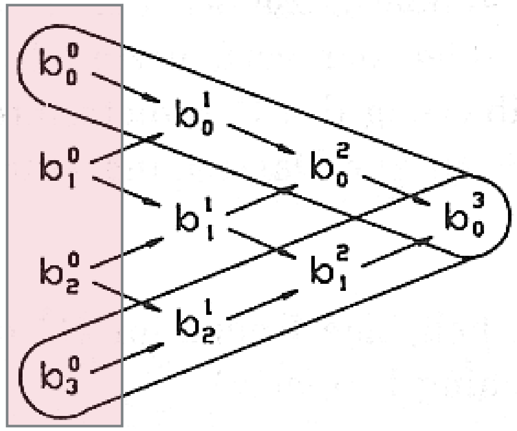
DeCasteljau Algorithm

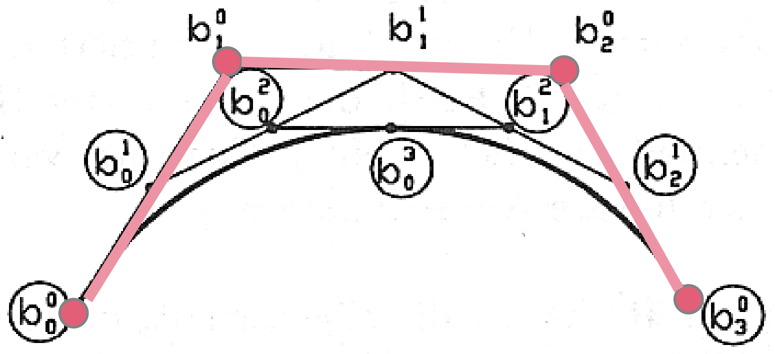
- DeCasteljau-Algorithm:
  - Recursive degree reduction of the Bezier curve by using the recursion formula for the Bernstein polynomials

$$P(t) = \sum_{i=0}^{n} b_i^0 B_i^n(t) = \sum_{i=0}^{n-1} b_i^1 B_i^{n-1}(t) = \cdots = b_i^n(t) \cdot 1$$

- Example:
  - t= 0.5

$$b_i^k(t) = t b_{i+1}^{k-1}(t) + (1-t) b_i^{k-1}(t)$$





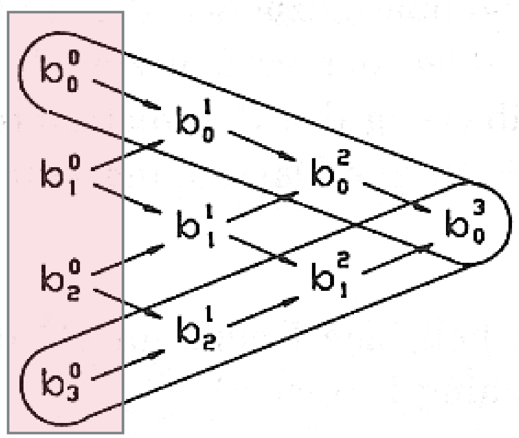
DeCasteljau Algorithm

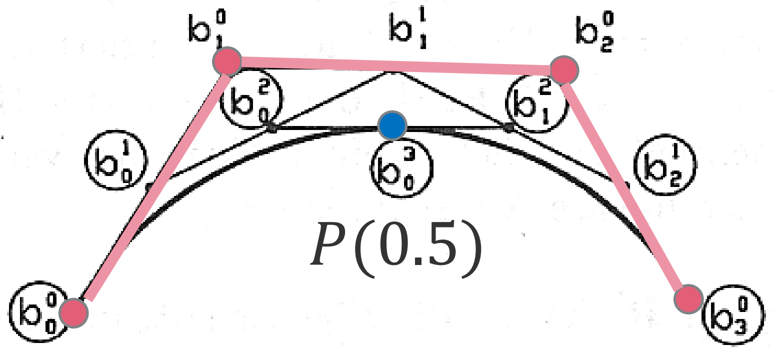
- DeCasteljau-Algorithm:
  - Recursive degree reduction of the Bezier curve by using the recursion formula for the Bernstein polynomials

$$P(t) = \sum_{i=0}^{n} b_i^0 B_i^n(t) = \sum_{i=0}^{n-1} b_i^1 B_i^{n-1}(t) = \cdots = b_i^n(t) \cdot 1$$

- Example:
  - t= 0.5

$$b_i^k(t) = t b_{i+1}^{k-1}(t) + (1-t) b_i^{k-1}(t)$$





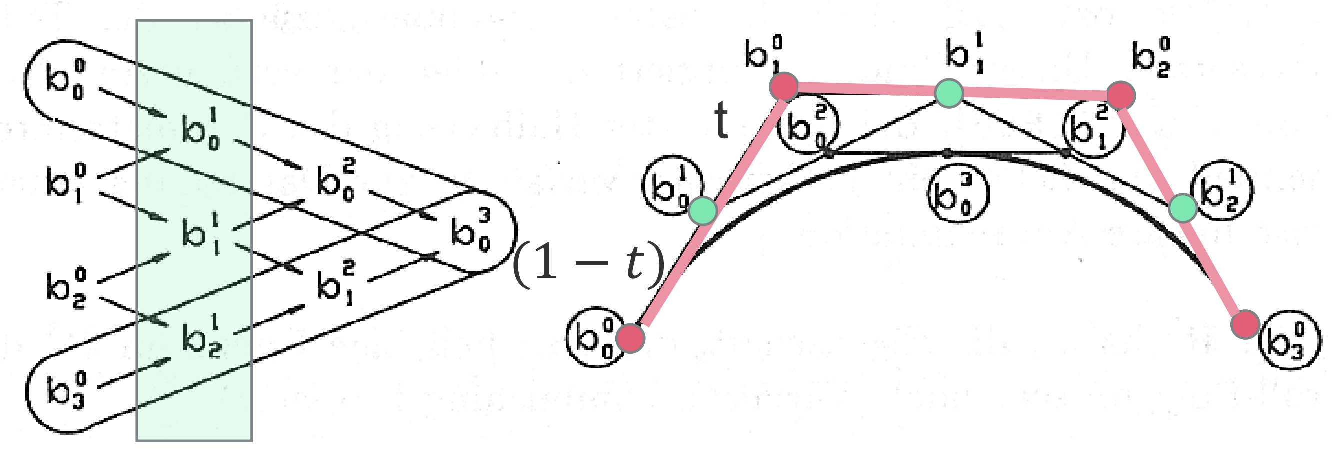
DeCasteljau Algorithm

- DeCasteljau-Algorithm:
  - Recursive degree reduction of the Bezier curve by using the recursion formula for the Bernstein polynomials

$$P(t) = \sum_{i=0}^{n} b_i^0 B_i^n(t) = \sum_{i=0}^{n-1} b_i^1 B_i^{n-1}(t) = \cdots = b_i^n(t) \cdot 1$$

- Example:
  - t= 0.5

$$b_i^k(t) = t b_{i+1}^{k-1}(t) + (1 - t) b_i^{k-1}(t)$$



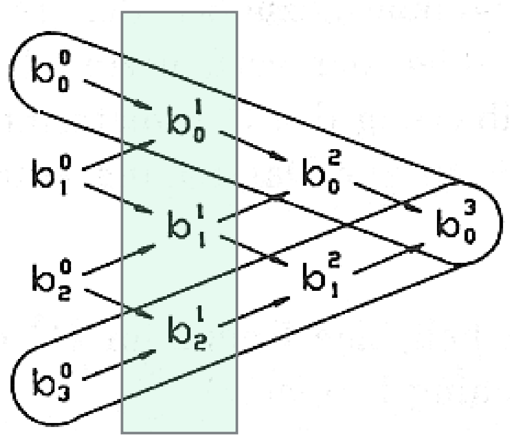
DeCasteljau Algorithm

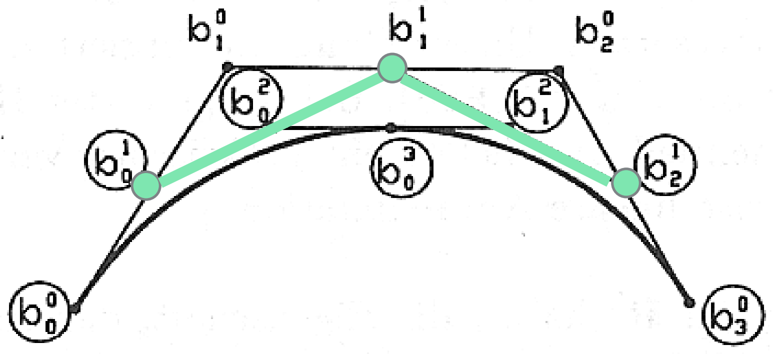
- DeCasteljau-Algorithm:
  - Recursive degree reduction of the Bezier curve by using the recursion formula for the Bernstein polynomials

$$P(t) = \sum_{i=0}^{n} b_i^0 B_i^n(t) = \sum_{i=0}^{n-1} b_i^1 B_i^{n-1}(t) = \cdots = b_i^n(t) \cdot 1$$

- Example:
  - t= 0.5

$$b_i^k(t) = t b_{i+1}^{k-1}(t) + (1-t) b_i^{k-1}(t)$$





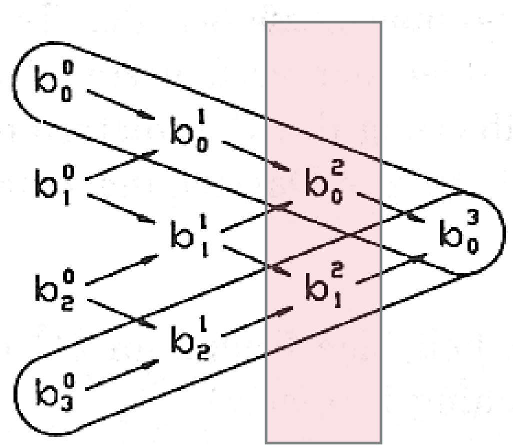
DeCasteljau Algorithm

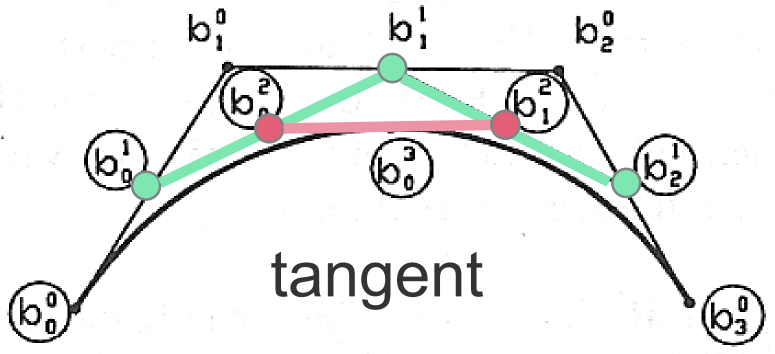
- DeCasteljau-Algorithm:
  - Recursive degree reduction of the Bezier curve by using the recursion formula for the Bernstein polynomials

$$P(t) = \sum_{i=0}^{n} b_i^0 B_i^n(t) = \sum_{i=0}^{n-1} b_i^1 B_i^{n-1}(t) = \cdots = b_i^n(t) \cdot 1$$

- Example:
  - t= 0.5

$$b_i^k(t) = t b_{i+1}^{k-1}(t) + (1-t) b_i^{k-1}(t)$$





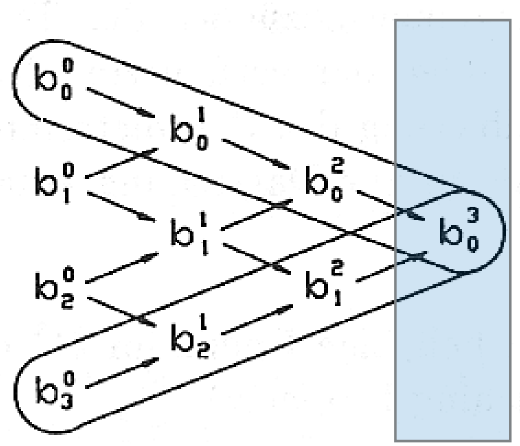
DeCasteljau Algorithm

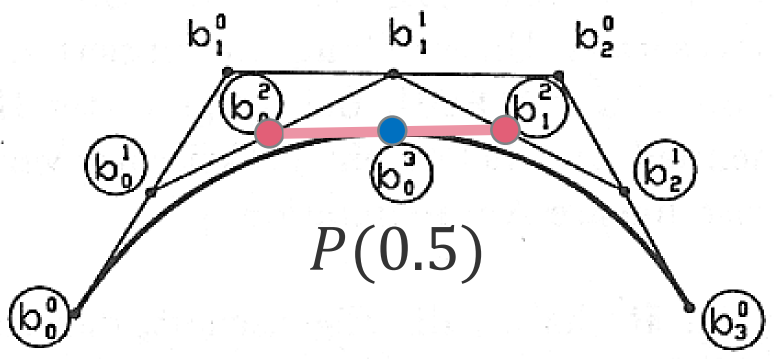
- DeCasteljau-Algorithm:
  - Recursive degree reduction of the Bezier curve by using the recursion formula for the Bernstein polynomials

$$P(t) = \sum_{i=0}^{n} b_i^0 B_i^n(t) = \sum_{i=0}^{n-1} b_i^1 B_i^{n-1}(t) = \cdots = b_i^n(t) \cdot 1$$

- Example:
  - t= 0.5

$$b_i^k(t)=\mathrm{t}b_{i+1}^{k-1}(t) + (1-\mathrm{t})b_i^{k-1}(t)$$





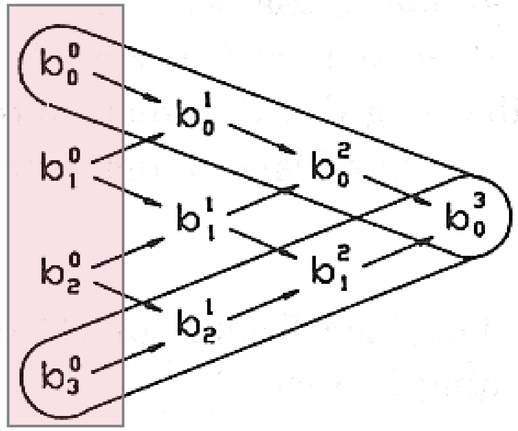
DeCasteljau Algorithm

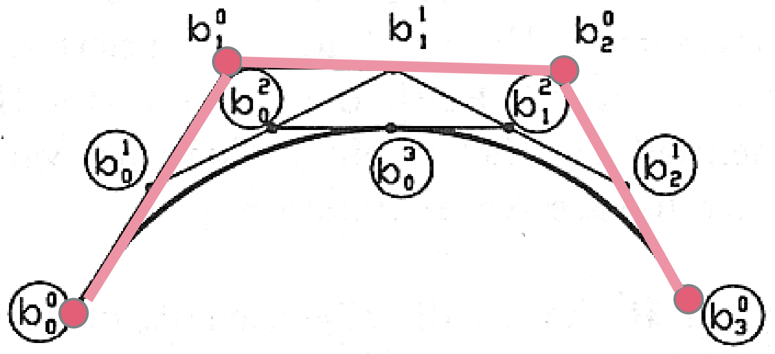
- DeCasteljau-Algorithm:
  - Recursive degree reduction of the Bezier curve by using the recursion formula for the Bernstein polynomials

$$P(t) = \sum_{i=0}^{n} b_i^0 B_i^n(t) = \sum_{i=0}^{n-1} b_i^1 B_i^{n-1}(t) = \cdots = b_i^n(t) \cdot 1$$

- Example:
  - t= 0.25

$$b_i^k(t) = t b_{i+1}^{k-1}(t) + (1-t) b_i^{k-1}(t)$$





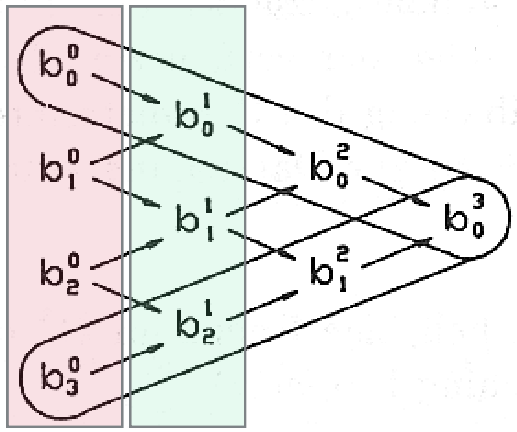
DeCasteljau Algorithm

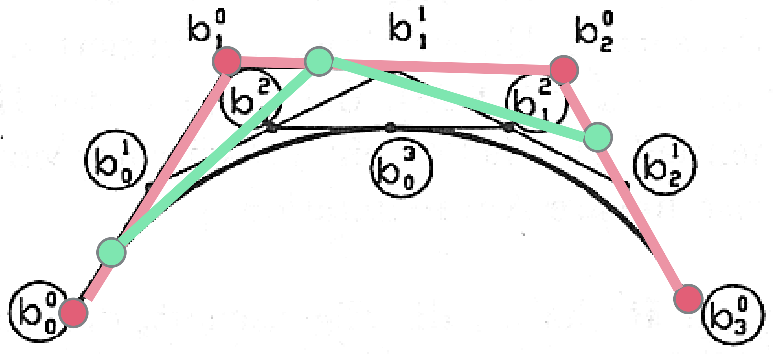
- DeCasteljau-Algorithm:
  - Recursive degree reduction of the Bezier curve by using the recursion formula for the Bernstein polynomials

$$P(t) = \sum_{i=0}^{n} b_i^0 B_i^n(t) = \sum_{i=0}^{n-1} b_i^1 B_i^{n-1}(t) = \cdots = b_i^n(t) \cdot 1$$

- Example:
  - t= 0.25

$$b_i^k(t) = t b_{i+1}^{k-1}(t) + (1-t) b_i^{k-1}(t)$$





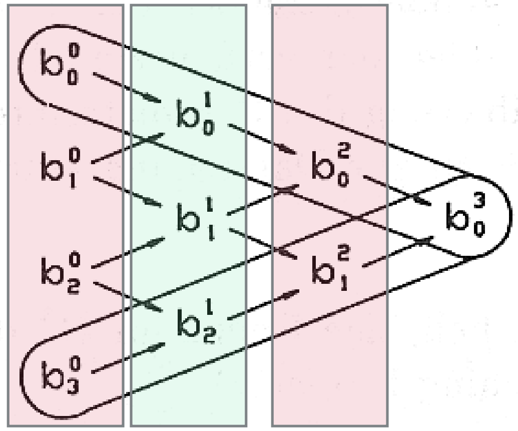
DeCasteljau Algorithm

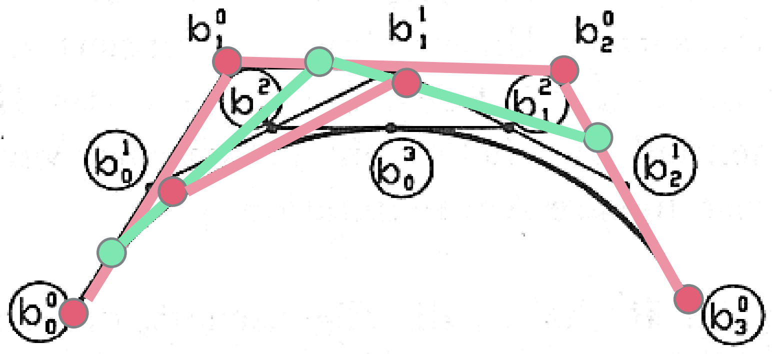
- DeCasteljau-Algorithm:
  - Recursive degree reduction of the Bezier curve by using the recursion formula for the Bernstein polynomials

$$P(t) = \sum_{i=0}^{n} b_i^0 B_i^n(t) = \sum_{i=0}^{n-1} b_i^1 B_i^{n-1}(t) = \cdots = b_i^n(t) \cdot 1$$

- Example:
  - t= 0.25

$$b_i^k(t) = t b_{i+1}^{k-1}(t) + (1-t) b_i^{k-1}(t)$$





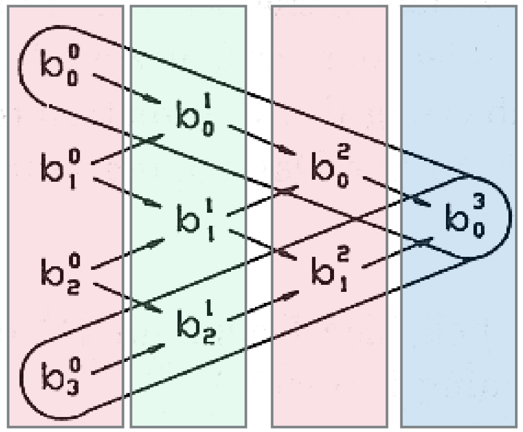
DeCasteljau Algorithm

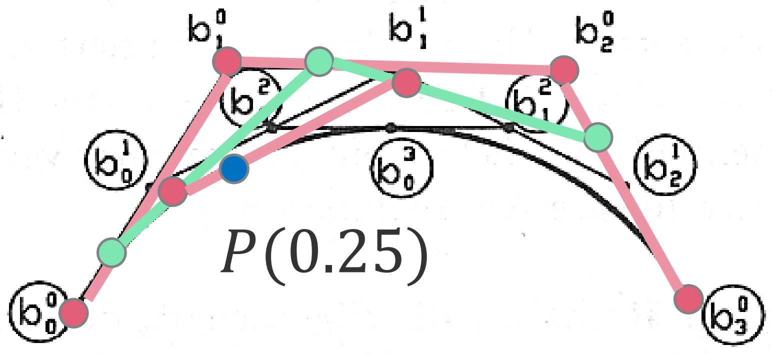
- DeCasteljau-Algorithm:
  - Recursive degree reduction of the Bezier curve by using the recursion formula for the Bernstein polynomials

$$P(t) = \sum_{i=0}^{n} b_i^0 B_i^n(t) = \sum_{i=0}^{n-1} b_i^1 B_i^{n-1}(t) = \cdots = b_i^n(t) \cdot 1$$

- Example:
  - t= 0.25

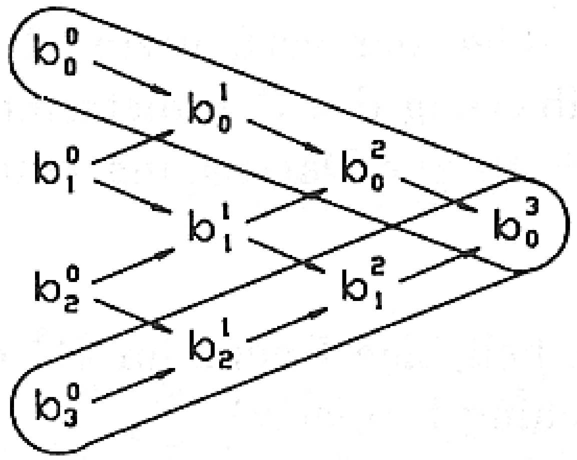
$$b_i^k(t) = t b_{i+1}^{k-1}(t) + (1-t) b_i^{k-1}(t)$$

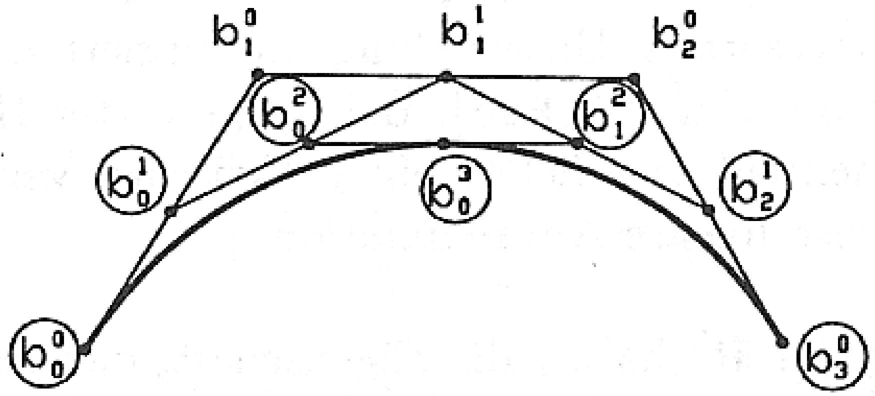




DeCasteljau Algorithm

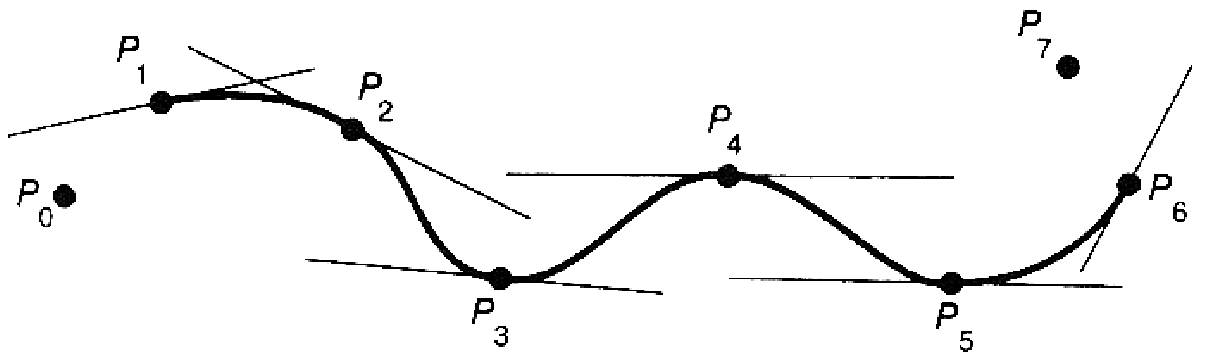
- Subdivision using the deCasteljau-Algorithm
  - Take boundaries of the deCasteljau triangle as new control points for left/right portion of the curve
- Extrapolation
  - Backwards subdivision
    - Reconstruct triangle from one side





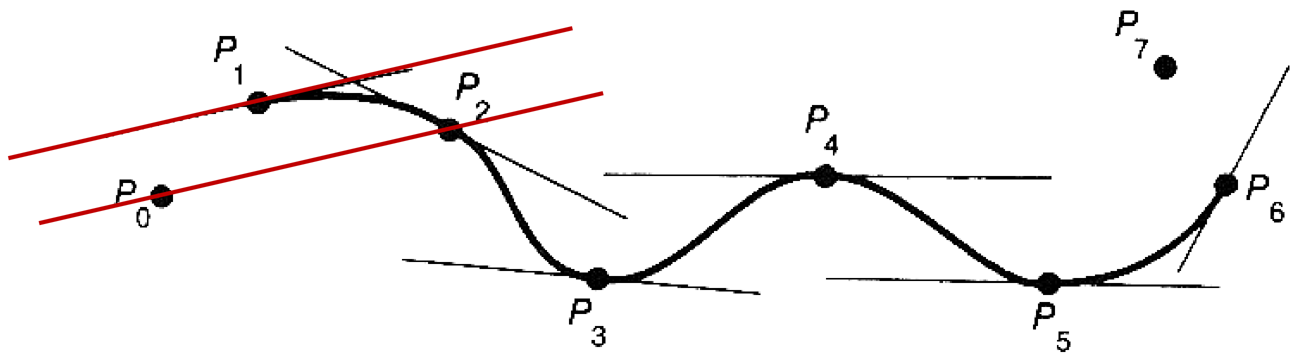
Catmull-Rom-Splines

- Goal
  - Smooth (C^1)-joints between (cubic) spline segments
- Algorithm
  - Tangents given by neighboring points P_{i-1} P_{i+1}
  - Construct (cubic) Hermite segments
- Advantage
  - Arbitrary number of control points
  - Interpolation without overshooting
  - Local control



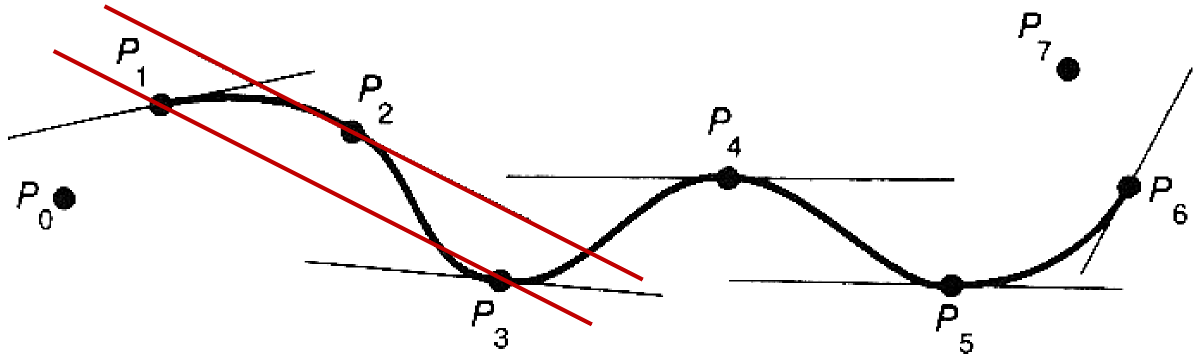
Catmull-Rom-Splines

- Goal
  - Smooth (C^1)-joints between (cubic) spline segments
- Algorithm
  - Tangents given by neighboring points P_{i-1} P_{i+1}
  - Construct (cubic) Hermite segments
- Advantage
  - Arbitrary number of control points
  - Interpolation without overshooting
  - Local control



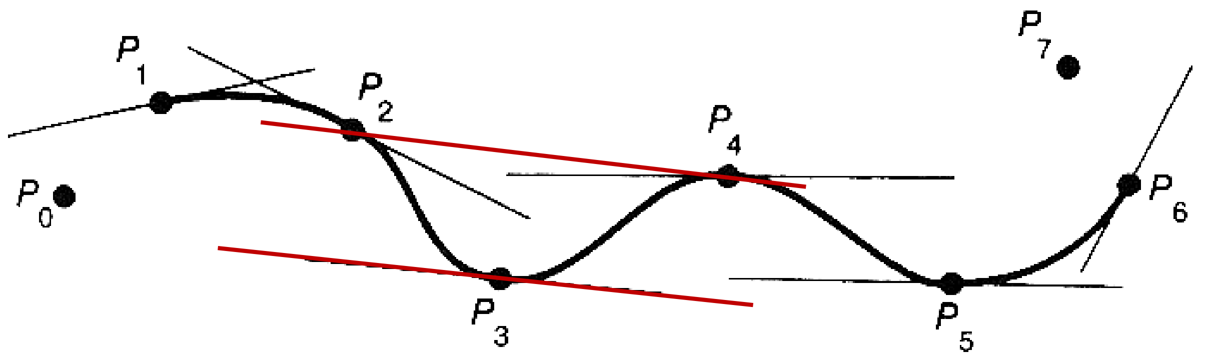
Catmull-Rom-Splines

- Goal
  - Smooth (C^1)-joints between (cubic) spline segments
- Algorithm
  - Tangents given by neighboring points P_{i-1} P_{i+1}
  - Construct (cubic) Hermite segments
- Advantage
  - Arbitrary number of control points
  - Interpolation without overshooting
  - Local control



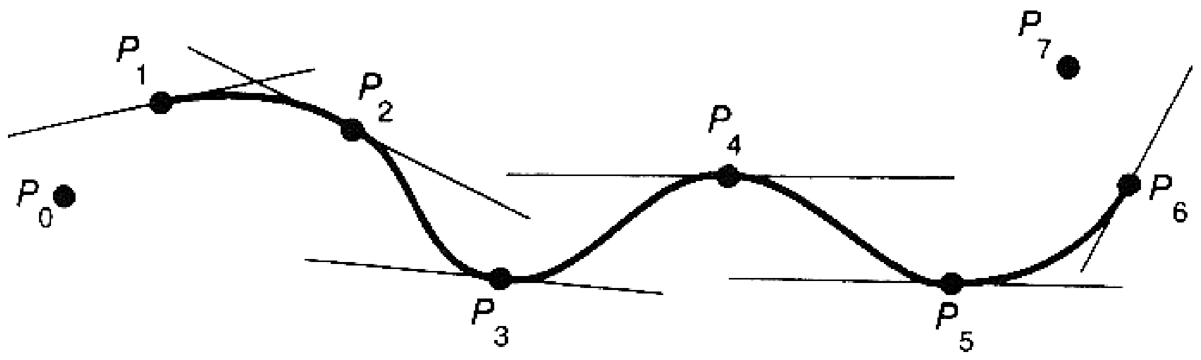
Catmull-Rom-Splines

- Goal
  - Smooth (C^1)-joints between (cubic) spline segments
- Algorithm
  - Tangents given by neighboring points P_{i-1} P_{i+1}
  - Construct (cubic) Hermite segments
- Advantage
  - Arbitrary number of control points
  - Interpolation without overshooting
  - Local control



Catmull-Rom-Splines

- Goal
  - Smooth (C^1)-joints between (cubic) spline segments
- Algorithm
  - Tangents given by neighboring points P_{i-1} P_{i+1}
  - Construct (cubic) Hermite segments
- Advantage
  - Arbitrary number of control points
  - Interpolation without overshooting
  - Local control



Matrix Representation

- Catmull-Rom-Spline
  - Piecewise polynomial curve
  - Four control points per segment
  - For n control points we obtain (n-3) polynomial segments

$$\underline{P}^i(t) = T\mathbf{M}_{C}\,G_{RC} = T\,\frac{1}{2}\begin{bmatrix}-1 & 3 & -3 & 1\\ 2 & -5 & 4 & 1\\ -1 & 0 & 1 & 0\\ 0 & 2 & 0 & 0\end{bmatrix}\begin{bmatrix}\underline{P}_i^T\\ \underline{P}_{i+1}^T\\ \underline{P}_{i+2}^T\\ \underline{P}_{i+3}^T\end{bmatrix}$$

- Application
  - Smooth interpolation of a given sequence of points
  - Key frame animation, camera movement, etc.
  - Only G^1-continuity
  - Control points should be equidistant in time

B-Splines

Choice of Parameterization

- Problem
  - Often only the control points are given
  - How to obtain a suitable parameterization t_i ?
- Example: Chord-Length Parameterization

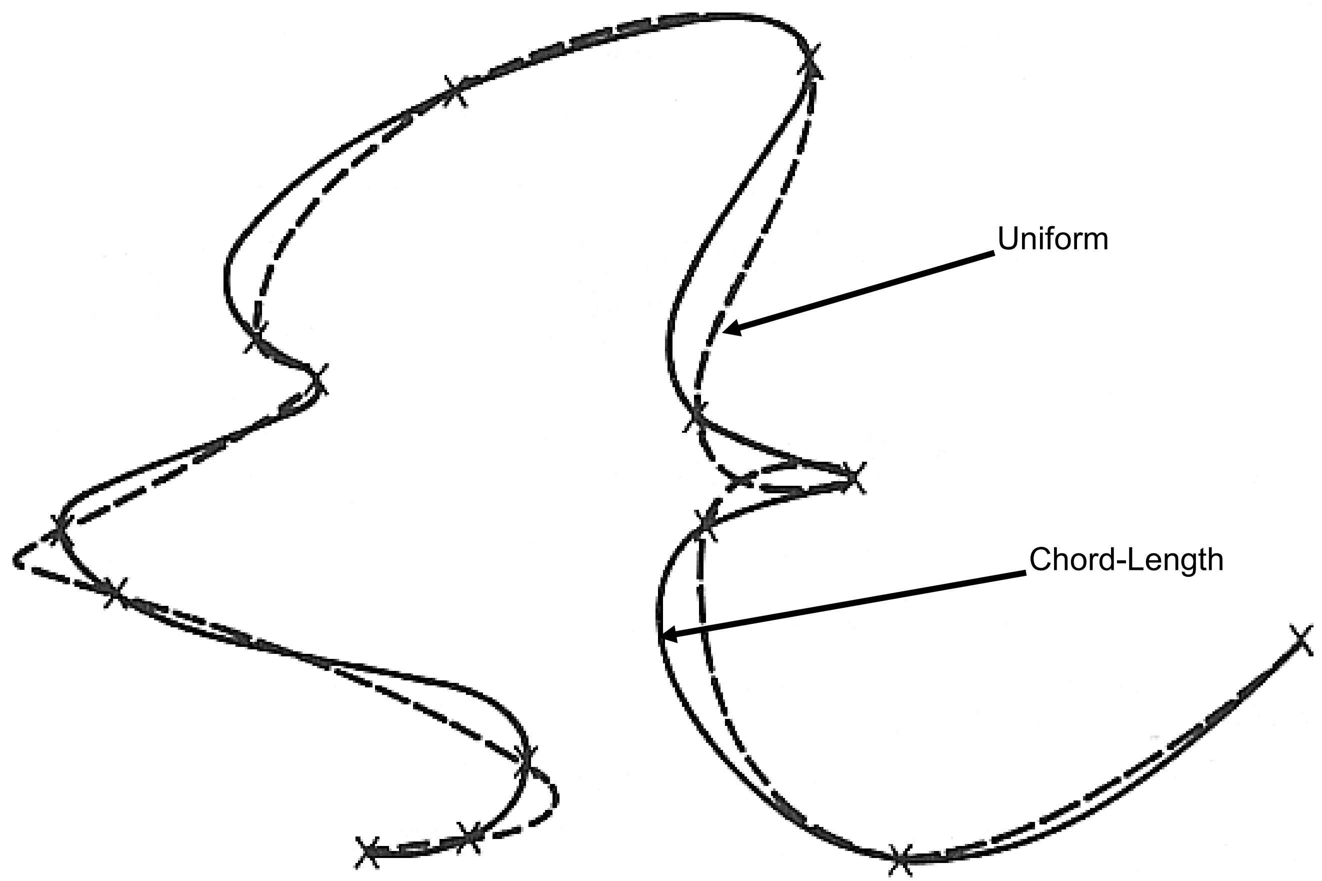
$$t_0 = 0$$

$$t_i = \sum_{j=1}^{i} dist(P_i - P_{i-1})$$

  - Arbitrary up to a constant factor
- Warning
  - Distances are not affine invariant !
  - Shape of curves changes under transformations !!

Parameterization

- Chord-Length versus uniform Parameterization
  - Analog: Think P(t) as a moving object with mass that may overshoot



B-Splines

- Goal
  - Spline curve with local control and high continuity
- Given
  - Degree: n
  - Control points: $P_0, \ldots, P_m$ (Control polygon, $m \geq n+1$)
  - Knots: $t_0, \ldots, t_{m+n+1}$ (Knot vector, weakly monotonic)
  - The knot vector defines the parametric locations where segments join
- B-Spline Curve

$$\underline{P}(t) = \sum_{i=0}^{m} N_i^n(t)\underline{P}_i$$

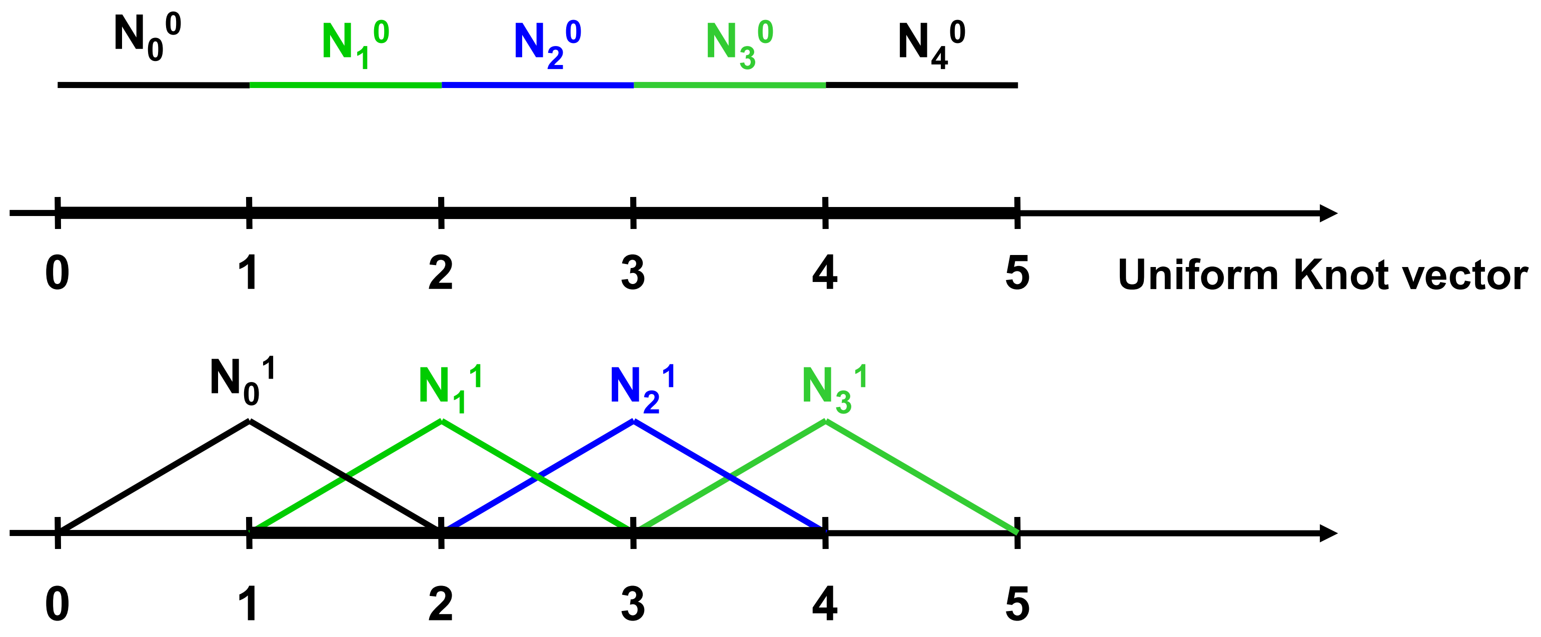
  - Continuity:
    - C_{n-1} at simple knots
    - C_{n-k} at knot with multiplicity k

B-Spline Basis Functions

- Recursive Definition

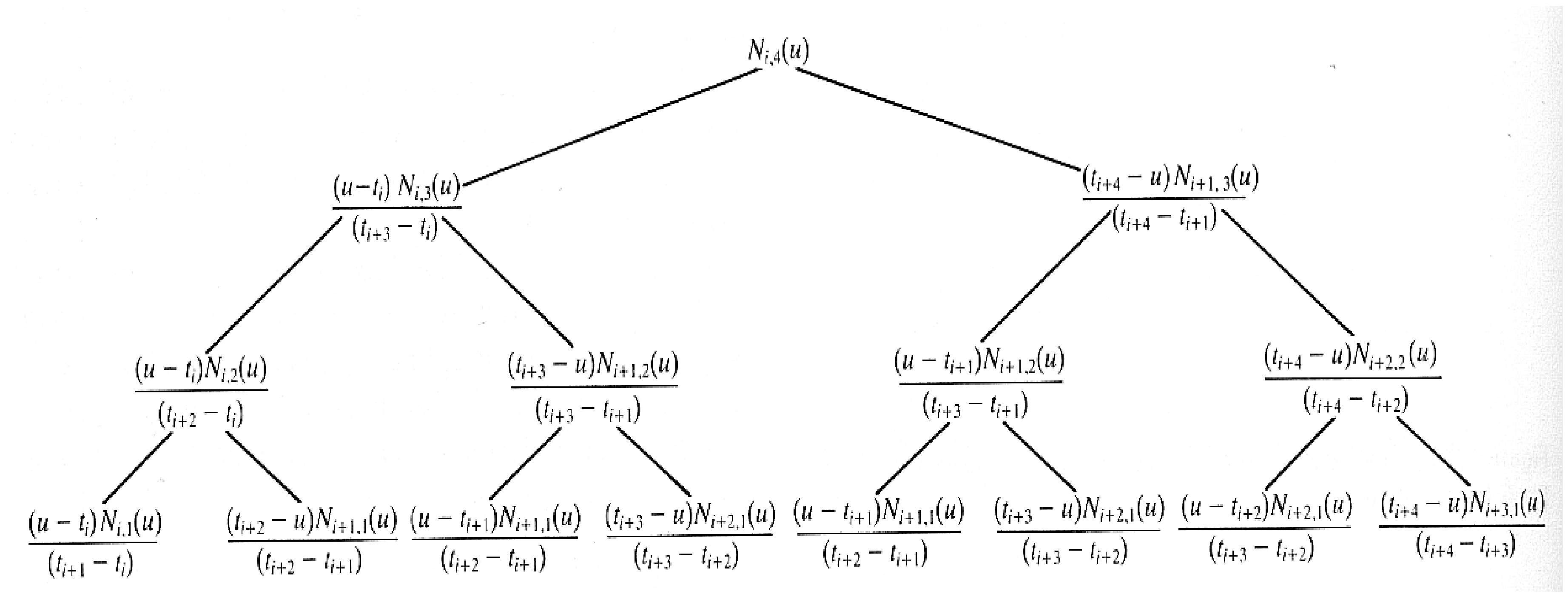
$$N_i^0(t) = \begin{cases} 1 & \text{if } t_i < t < t_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

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



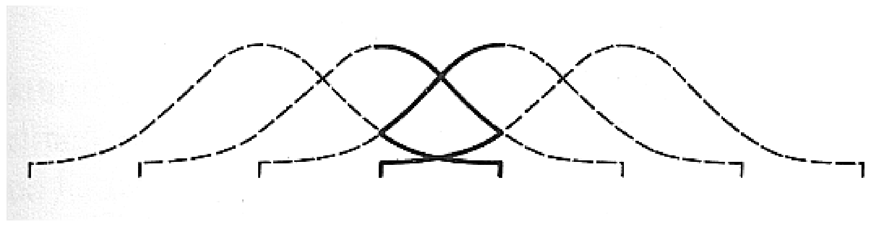
B-Spline Basis Functions

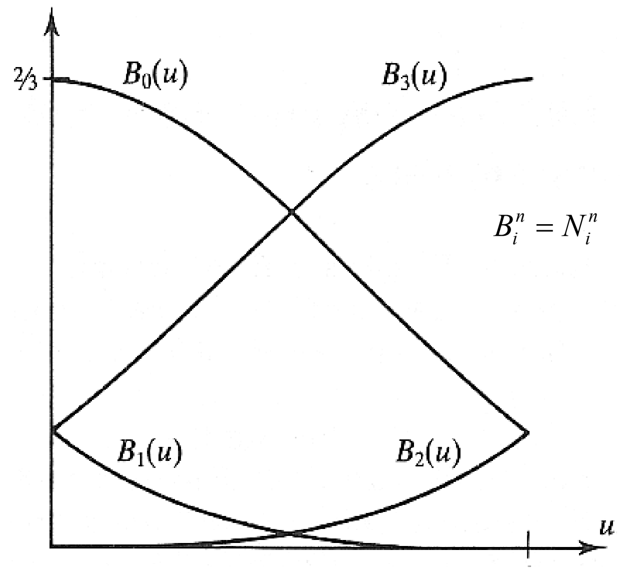
- Recursive Definition
  - Degree increases in every step
  - Support increases by one knot interval


$N_{i,4}(u)$
$\frac{(u-t_i)\,N_{i,3}(u)}{(t_{i+3}-t_i)}$
$\frac{(t_{i+4}-u)N_{i+1,3}(u)}{(t_{i+4}-t_{i+1})}$
$\frac{(u-t_i)N_{i,2}(u)}{(t_{i+2}-t_i)}$
$\frac{(t_{i+3}-u)N_{i+1,2}(u)}{(t_{i+3}-t_{i+1})}$
$\frac{(u-t_{i+1})N_{i+1,2}(u)}{(t_{i+3}-t_{i+1})}$
$\frac{(t_{i+4}-u)N_{i+2,2}(u)}{(t_{i+4}-t_{i+2})}$
$\frac{(u-t_i)N_{i,1}(u)}{(t_{i+1}-t_i)}$
$\frac{(t_{i+2}-u)N_{i+1,1}(u)}{(t_{i+2}-t_{i+1})}$
$\frac{(u-t_{i+1})N_{i+1,1}(u)}{(t_{i+2}-t_{i+1})}$
$\frac{(t_{i+3}-u)N_{i+2,1}(u)}{(t_{i+3}-t_{i+2})}$
$\frac{(u-t_{i+1})N_{i+1,1}(u)}{(t_{i+2}-t_{i+1})}$
$\frac{(t_{i+3}-u)N_{i+2,1}(u)}{(t_{i+3}-t_{i+2})}$
$\frac{(u-t_{i+2})N_{i+2,1}(u)}{(t_{i+3}-t_{i+2})}$
$\frac{(t_{i+4}-u)N_{i+3,1}(u)}{(t_{i+4}-t_{i+3})}$

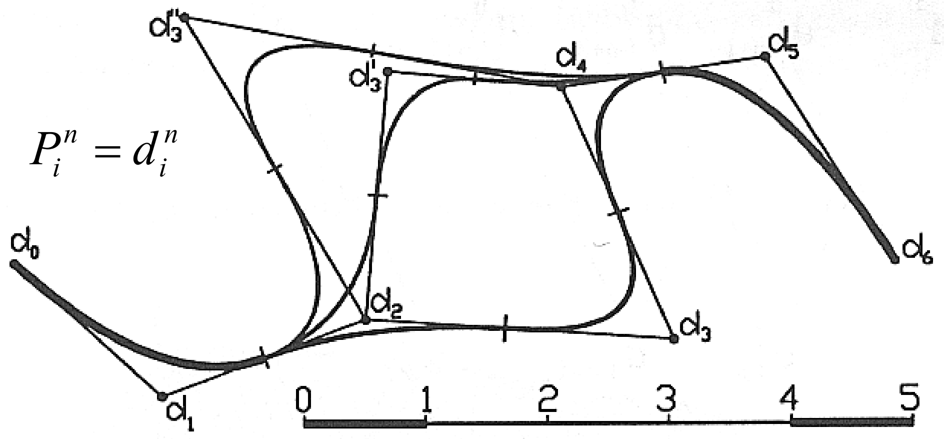
B-Spline Basis Functions

- Uniform Knot Vector
  - All knots at integer locations
    - UBS: Uniform B-Spline
  - Example: cubic B-Splines





- Local Support = Localized Changes
  - Basis functions affect only (n+1) Spline segments
  - Changes are localized



Degree 2

Summary

- Interpolating polynomials hard to control
- Splines: Curves as piece-wise polynomial functions
- Matrices for basis transformations / calculation of derivatives
- DeCasteljau Algorithm for efficient evaluation
- B-Splines can control the parameterization

Written Exam

- Tuesday, 22.02.2022
- 08:00 (s.t.) – 11:00, N10!

- Bring a Pen and a Ruler

- No calculator allowed.

- Resit exam: **Thursday, 31.03.2022!**