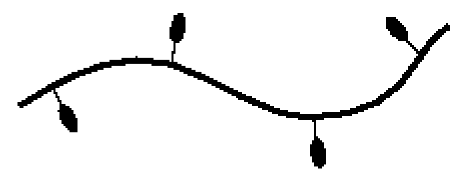
CS 428: Fall 2009

Introduction to Computer Graphics

Parametric curves and surfaces

Curve representation + design

- Loftsman spline
  - Thin strip of wood/metal
  - Shaped by fixed weights – “ducks”
  - Produces (mostly) C^2 curves by minimizing bending energy



- Developed in 60s for industrial design
- Uses in CG
  - Building models
  - Paths of motion + interpolation in animation

Curve and surface representations

- Explicit representations

$$\mathbf{p} : R \to R^d, d = 1,2,3,\ldots \qquad \mathbf{q} : R^2 \to R^d, d = 1,2,3,\ldots$$

$$t \mapsto \mathbf{p}(t) = \big(x(t), y(t), z(t)\big) \qquad (u,v) \mapsto \mathbf{q}(u,v) = \big(x(u,v), y(u,v), z(u,v)\big)$$

$$\mathbf{p}(t) = r \cdot \big(\cos(t), \sin(t), 0\big) \qquad \mathbf{p}(u,v) = r \cdot \big(\cos(u)\cos(v),\ \sin(u)\cos(v),\ \sin(v)\big)$$

$$t \in [0, 2\pi] \qquad (u,v) \in [0,2\pi] \times [-\pi/2, \pi/2]$$

- Implicit representations

$$f : R^2 \to R \qquad g : R^3 \to R$$

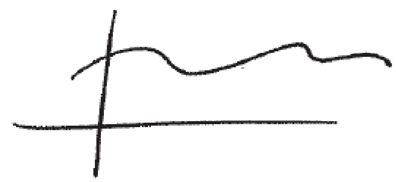
$$K = \{\mathbf{p} \in R^2 : f(\mathbf{p}) = 0\} \qquad K = \{\mathbf{p} \in R^3 : g(\mathbf{p}) = 0\}$$

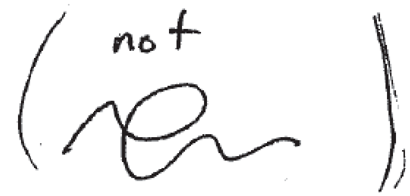
$$f(x,y) = x^2 + y^2 - r^2 \qquad g(x,y,z) = x^2 + y^2 + z^2 - r^2$$

Mathematical curve representations

Explicit

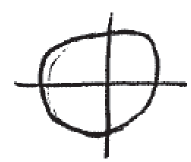
$y = f(x)$





Implicit

$f(x,y,z) = 0$

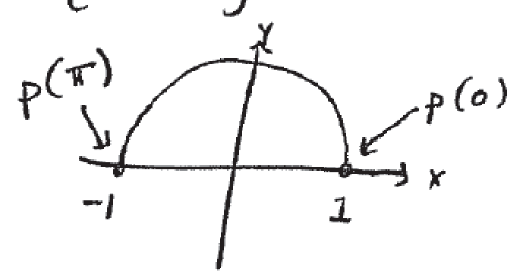


$x^2 + y^2 - 1 = 0$

Parametric

$$p(t) = \begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix}$$

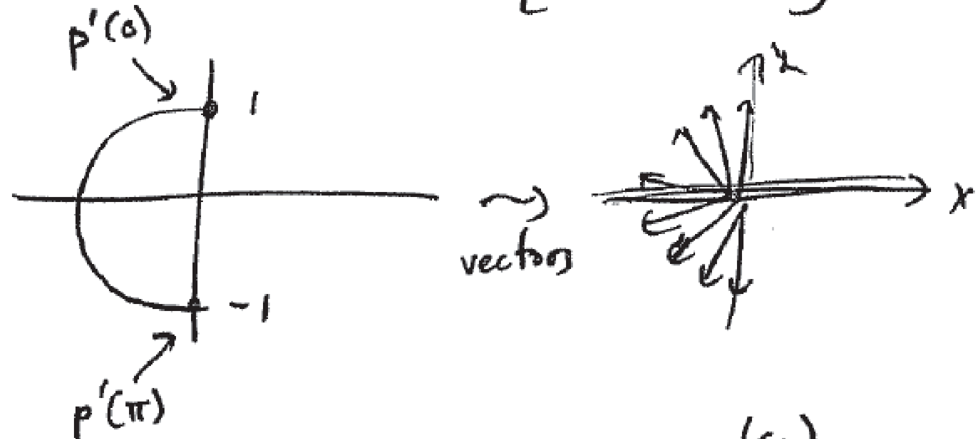
$$p(t) = \begin{bmatrix} \cos(t) \\ \sin t \end{bmatrix} \quad t \in [0, \pi]$$

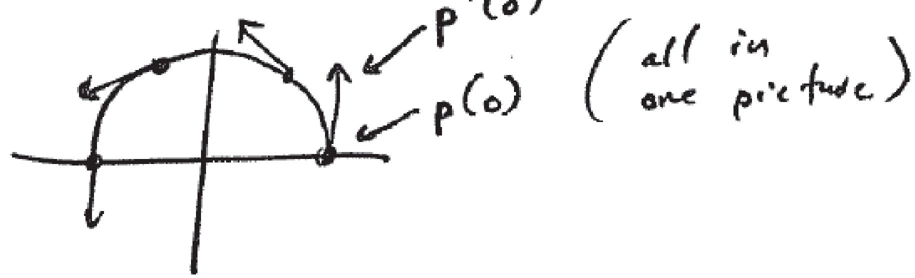


Parametric curve derivatives

- Tangent vector: points in direction of curve as t changes

$$p'(t) = \frac{dp(t)}{dt} = \begin{bmatrix} -\sin t \\ \cos t \end{bmatrix} \qquad t \in [0, \pi]$$

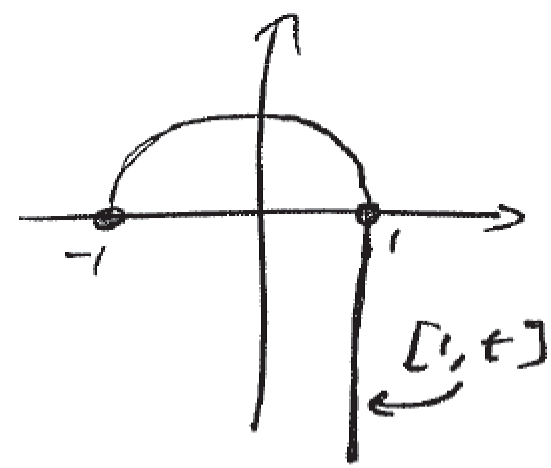




Piecewise definitions

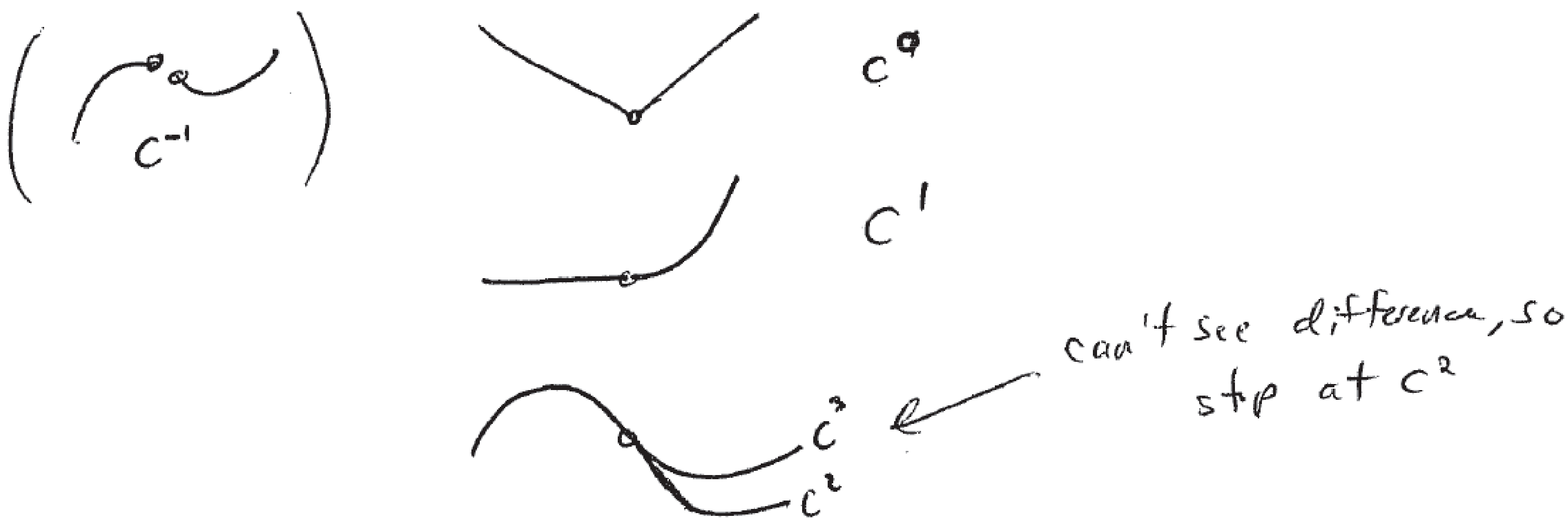
- Piece together curves for varying parameter values

$$p(t) = \begin{cases} \begin{bmatrix} \cos t \\ \sin t \end{bmatrix} & t \in [0, \pi] \\ \begin{bmatrix} 1 \\ t \end{bmatrix} & t < 0 \end{cases}$$



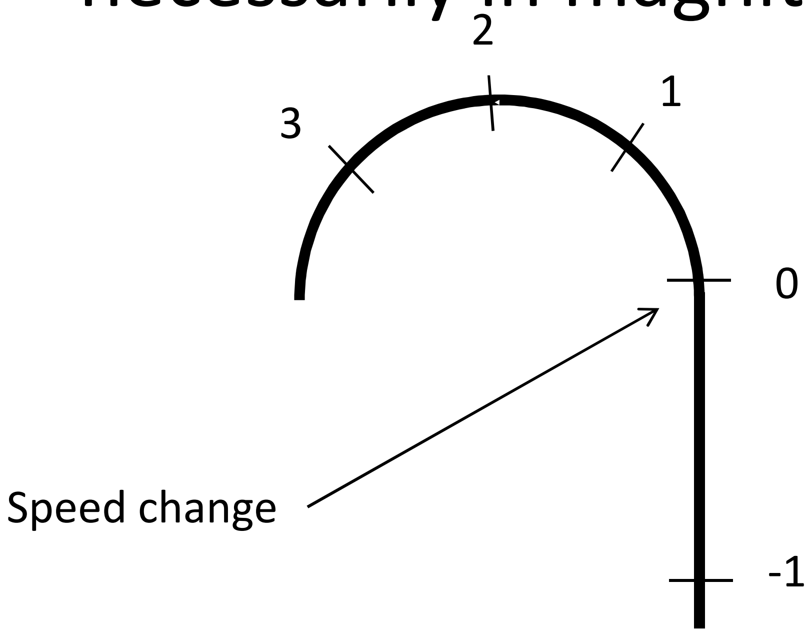
Parametric continuity

- **C^k** – k-th order derivatives exist **and** are continuous (at joints)

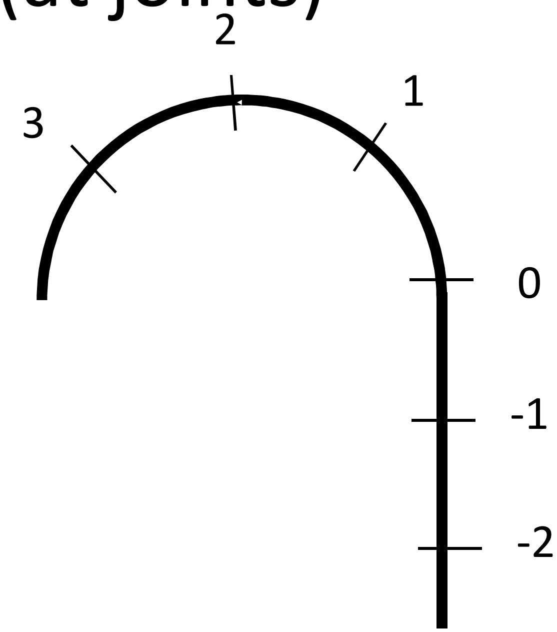


Geometric continuity

- Signed **direction** of k-th derivates agree, not necessarily in magnitude (at joints)



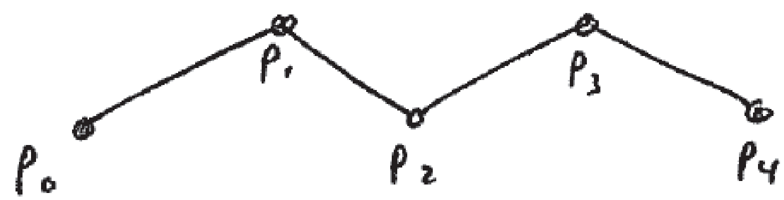
G^1 but not C^1



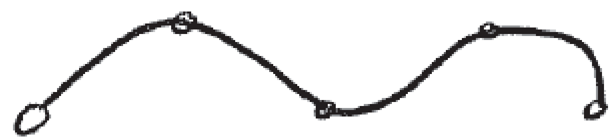
G^1 and C^1

Representation

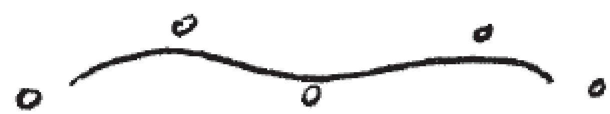
- Generate curve using ordered series of points
  - Control polygon



- Which curve to generate?
  - Interpolating
    - Control points on curve
    - Wiggles, unstable

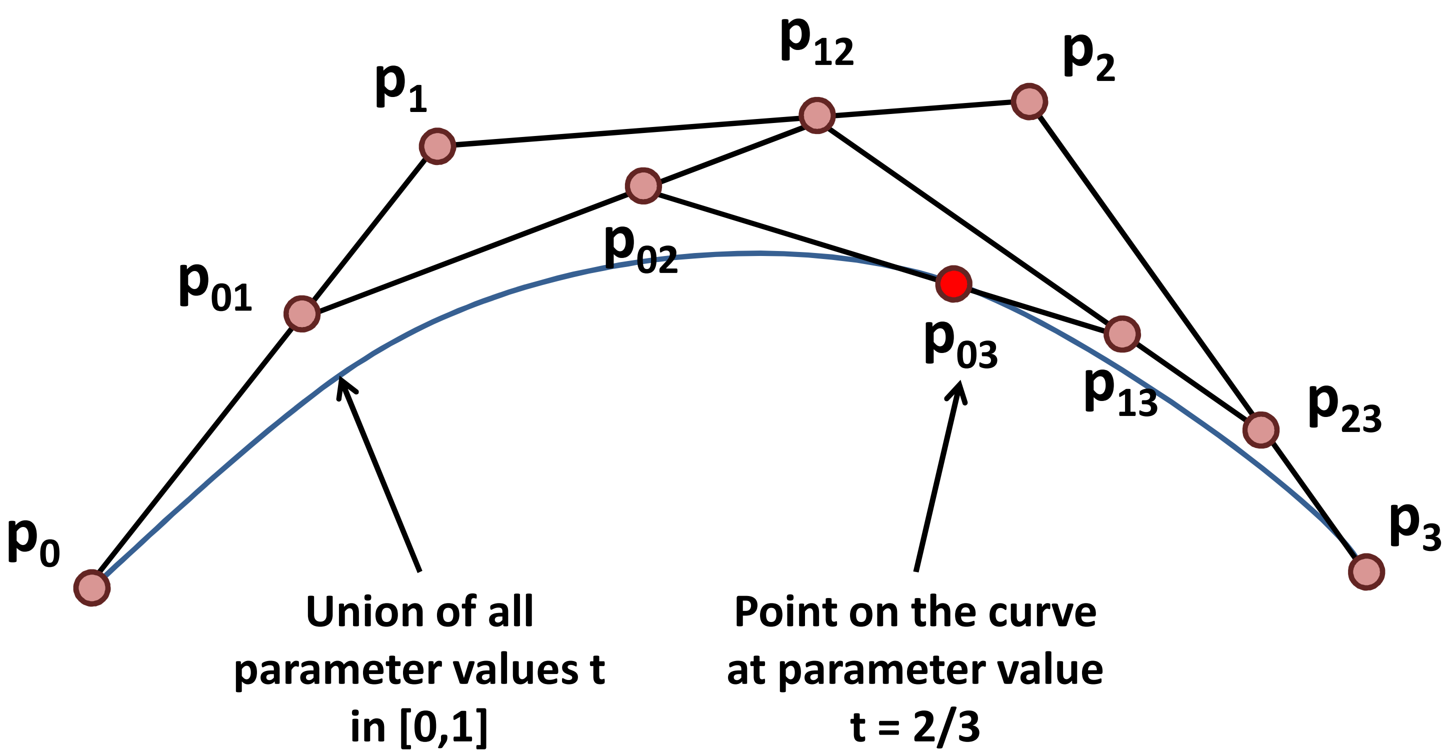


  - Approximating
    - Control points "close" to curve
    - Stable



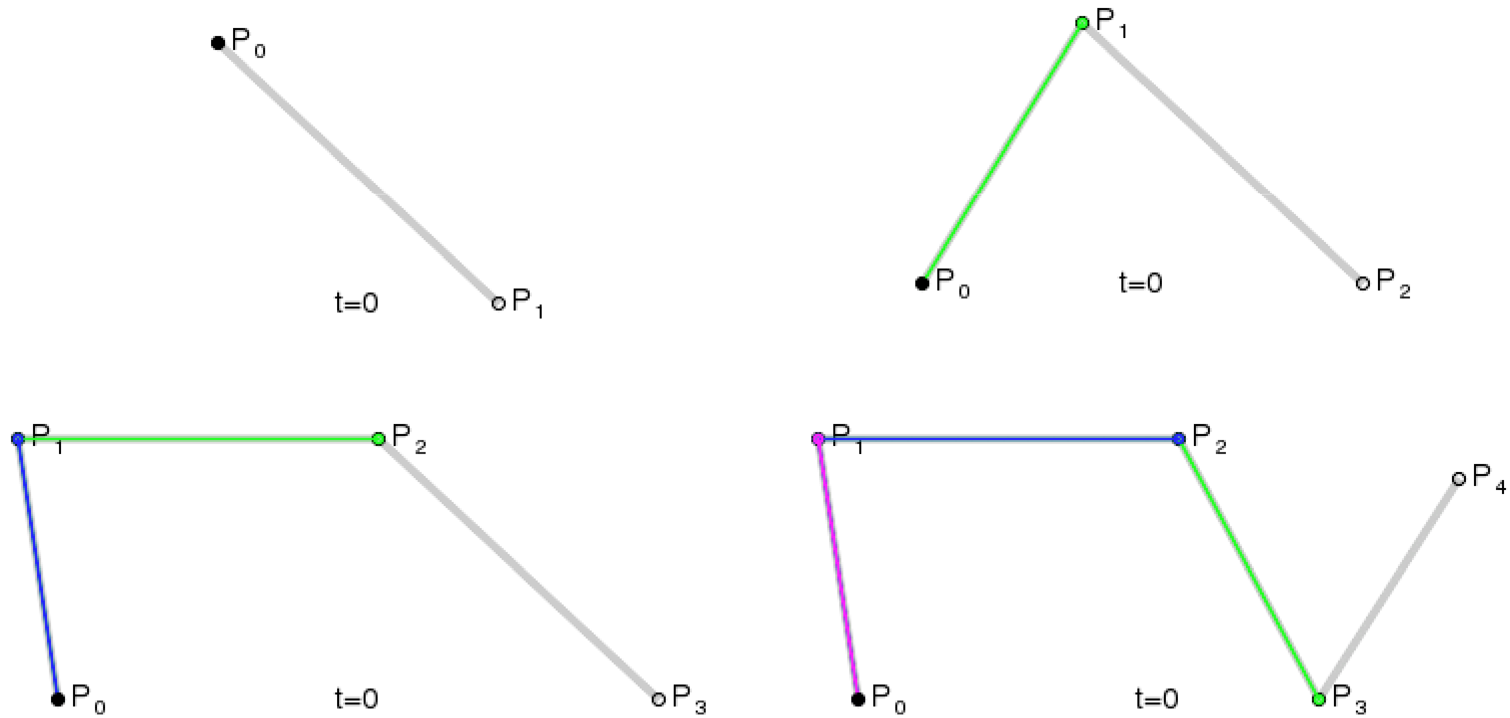
Bézier curves

- De Casteljau algorithm (example for t=2/3)



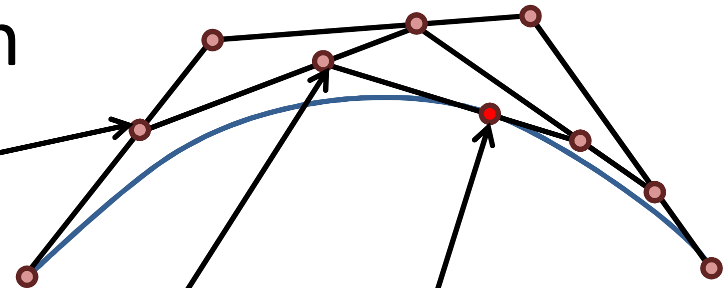
Bézier curves

- Animation of Bézier curves (from Wikipedia)



Bézier curves

- Mathematical construction



$$p_{01}(t) = (1-t)\,p_0 + t\,p_1$$

$$p_{12} = \vdots$$

$$p_{23} = \vdots$$

$$p_{02}(t) = (1-t)\,p_{01}(t) + t\,p_{12}(t)$$

$$p_{13}(t) = \dots$$

$$p_{03}(t) = (1-t)\,p_{02}(t) + t\,p_{13}(t)$$ **= p(t)**

Bézier curves

- Mathematical construction

$$P_{03}(t) = (1-t)p_{02}(t) + t\,p_{13}(t)$$

$$= \underbrace{p_0} + \underbrace{3(p_1 - p_0)}\,t + \underbrace{\cdots}\,t^2 + \underbrace{\cdots}\,t^3$$

$$p(t) = \;= \underbrace{(1-t)^3}_{B_0^3(t)}p_0 + \underbrace{3(1-t)^2 t}_{B_1^3(t)}p_1 + \underbrace{3(1-t)t^2}_{B_2^3(t)}p_2 + \underbrace{t^3}_{B_3^3(t)}p_3$$

Bernstein polynomials

$$B_k^d(t) = \binom{d}{k} t^k (1-t)^{d-k}$$

So

Bézier curves

$$p(t) = \sum_{i=0}^{d} p_i\, B_i^d(t)$$

binomial coefficient: $\binom{d}{k} = \dfrac{d!}{k!\,(d-k)!}$

$d = (\#\text{ of control points} - 1)$
$= \text{degree of curve } (3 = \text{cubic})$

Bézier curves

- Matrix form

$$p(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \underbrace{\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & 6 & 3 & \\ -3 & 3 & & \\ 1 & & & \end{bmatrix}}_{\text{Bézier matrix}} \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix}$$

- Easy to show

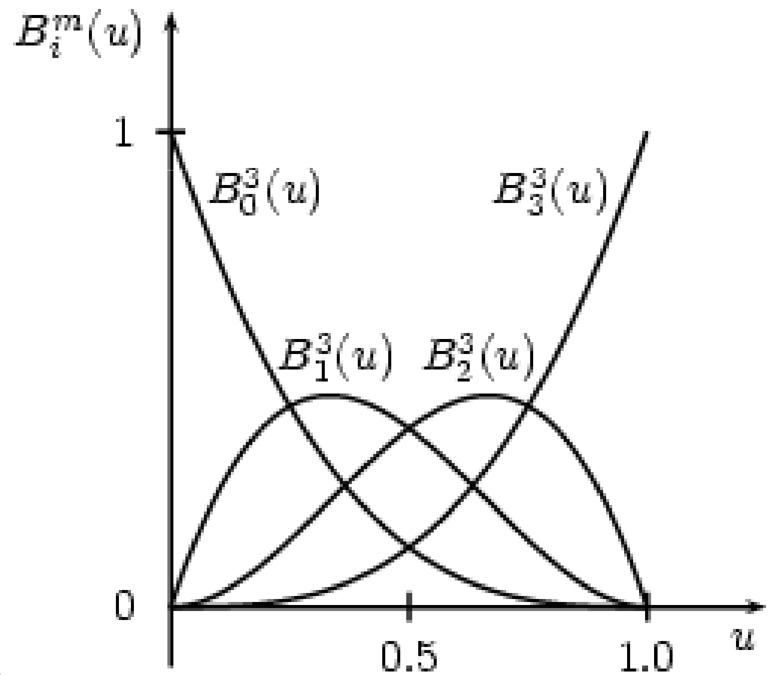
$$\sum_{i=0}^{d} B_i^d(t) = \underbrace{B_0^3(t) + B_1^3(t) + \dots B_3^3(t)}_{\text{example for } d=3} = 1$$

  - So this is an **affine** combination!

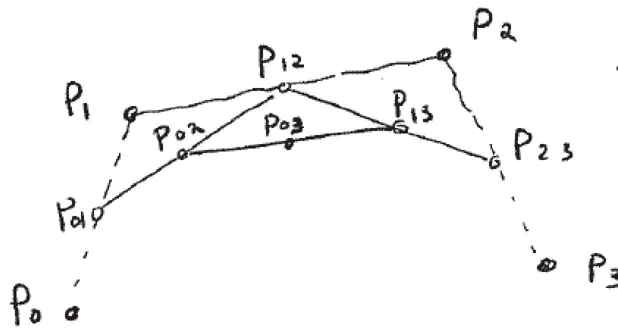
Bézier curves

- Bernstein polynomials as basis functions
- Form a basis of cubic polynomials that map $[0,1] \rightarrow \Re$
- Sum to 1 everywhere
- $p(0) = p_0$ and $p(1) = p_3$
- Affine invariance!

$$\mathbf{T}\left(\sum_{i=0}^{n} p_i B_i^d(t)\right) = \sum_{i=0}^{n} \left(\mathbf{T} p_i\right) B_i^d(t)$$



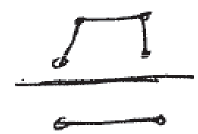
Drawing Bézier curves



← using $t = \frac{1}{2}$

draw (P_0, P_1, P_2, P_3)

{

if $\dfrac{|P_0 - P_1| + |P_1 - P_2| + |P_2 - P_3|}{|P_0 - P_3|} < (1+\varepsilon)$ ←  } if flat enough

line (P_0, P_3)

else { $P_{01} = \dots$

... draw $(P_0, P_{01}, P_{02}, P_{03})$ ← 

draw $(P_{03}, P_{13}, P_{23}, P_3)$ ←

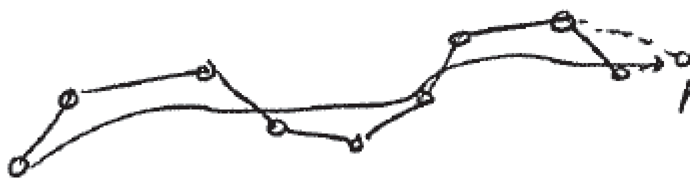
} divide + conquer

}

}

Towards B-splines

- Using a degree (N-1) curve with N points gets expensive and unstable with increasing N
- No local control, since each basis function is nonzero in [0,1]

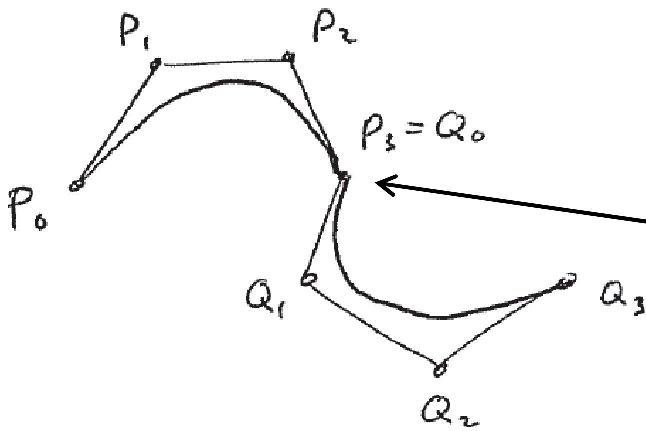


entire curve moves (a little)

- Possible solution: piecewise curves
  - Formed from degree 3 Bézier curves
  - How to join them to obtain C^k continuity?

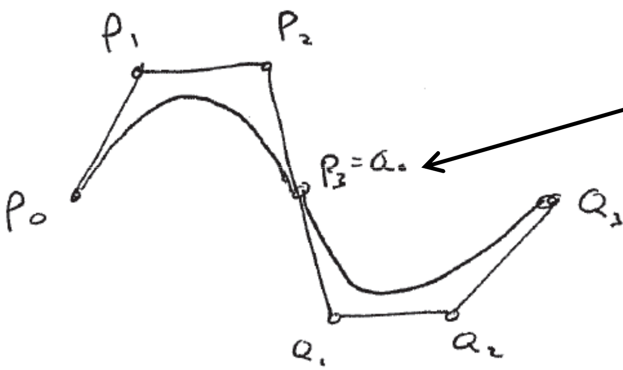
Bézier splines

Continuity

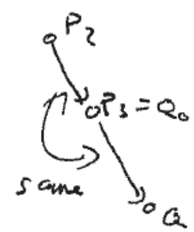


C^0 – rely on endpoint interpolation

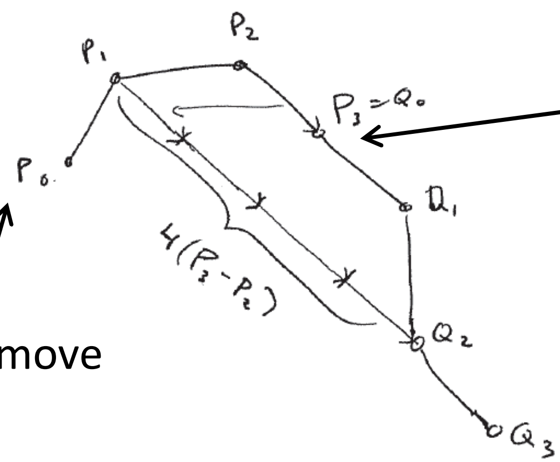
$P_3 = Q_0$



C^1 – want



$Q_1 = P_3 + (P_3 - P_2)$



C^2 –

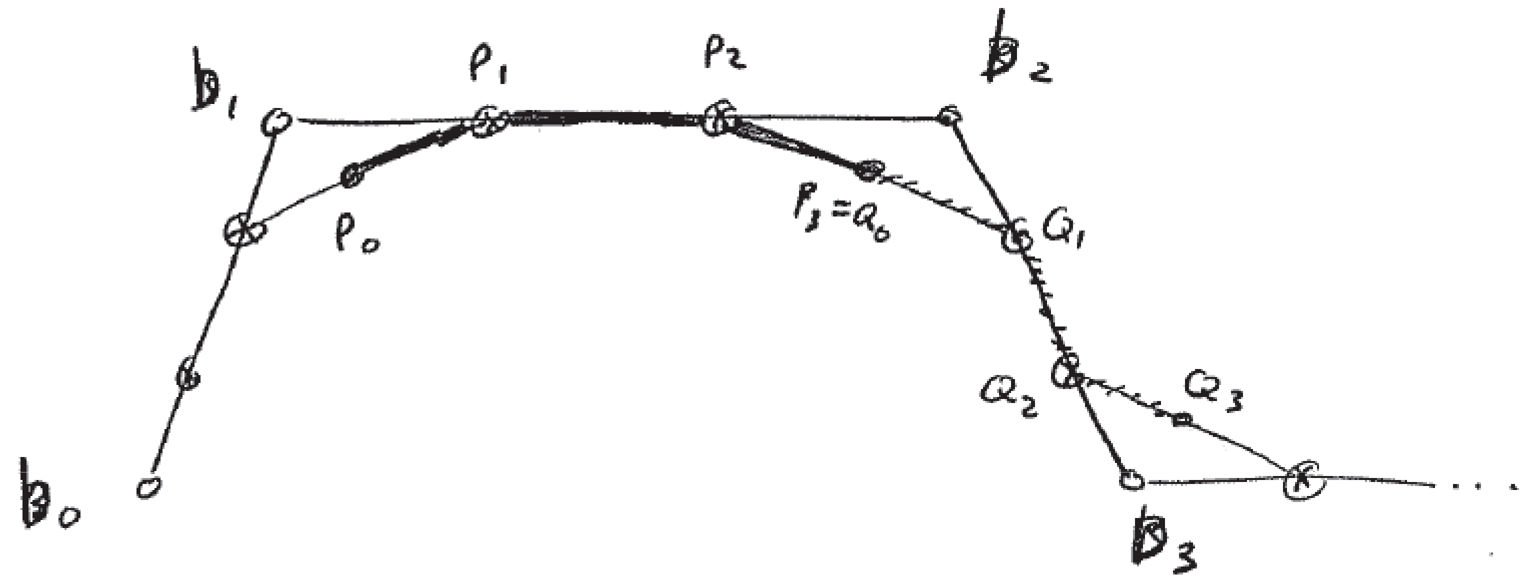
$Q_2 = P_1 + 4(P_3 - P_2)$

Still free to move

Still free to move

More constraints

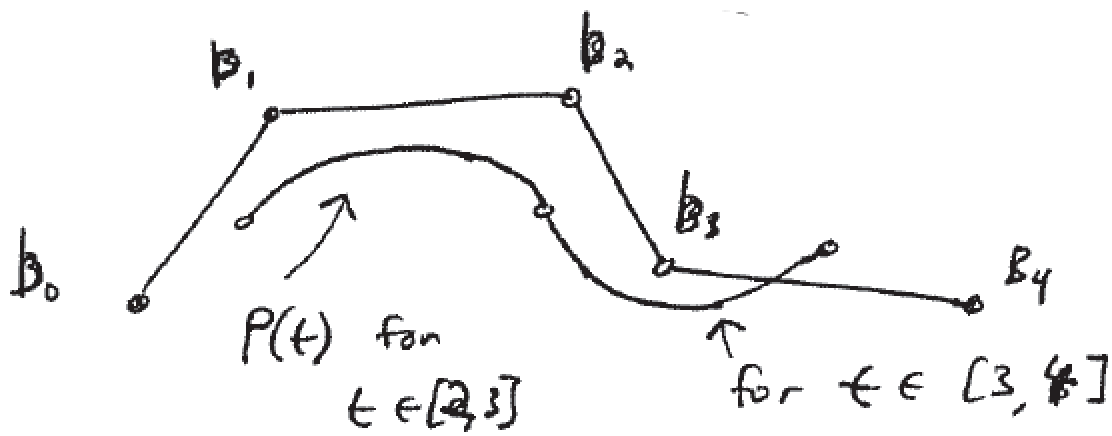
de Boor algorithm



- For cubic curves
  - Split each edge in de Boor polygon into 1/3's
  - Connect across corners, and split in half
- Has local control
  - Moving b_i only effects nearby Bézier curves

B-splines

- Piecewise polynomial of degree d with C^{d-1} continuity, specified by n+1 control points $\{b_k \mid k \in 0,\ldots,n\}$ and a knot vector $\{t_k \mid k \in 0,\ldots,n+d\}$ that contains n+d+1 values
  - For now, knot vector is {0,1,2,3,4,5,6}



B-splines
Definition

- Recursive definition of B-spline basis functions

$$p(t) = \sum_{k=0}^{n} b_k N_k^d(t)$$

$$N_k^0(t) = \begin{cases} 1 \ \text{for } t_k \le t < t_{k+1} \\ 0 \ \text{otherwise} \end{cases}$$

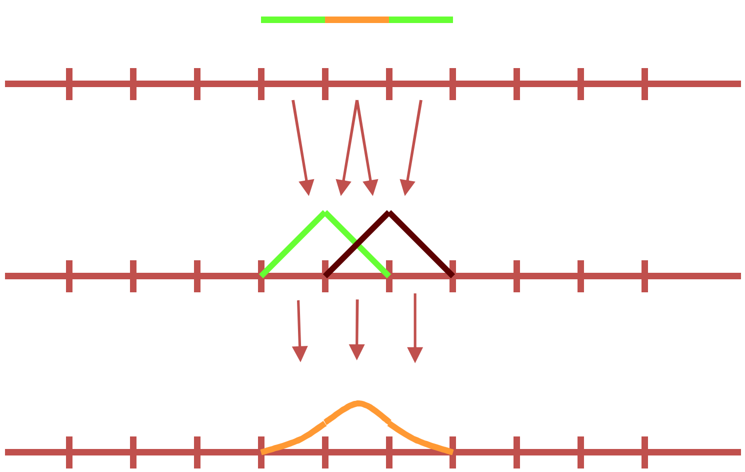
$$N_k^d(t) = \frac{t - t_k}{t_{k+d} - t_k} N_k^{d-1}(t) + \frac{t_{k+d+1} - t}{t_{k+d+1} - t_{k+1}} N_{k+1}^{d-1}(t)$$

- When the knots are equidistant, they are **uniform**, otherwise **non-uniform**

B-splines

Construction

- $N_k^d(t)$ is constructed from piecewise polynomials of degree d over t



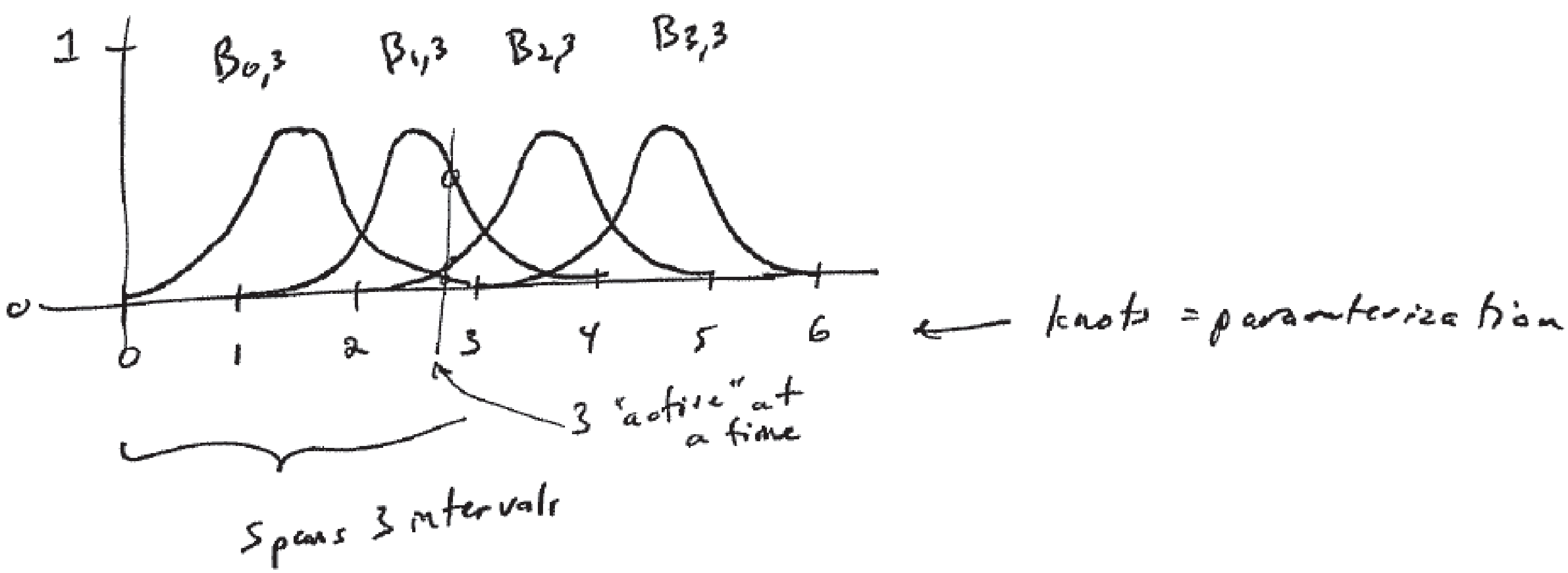
$$N_k^0(t) = \begin{cases} 1 & \text{for } t_k \le t < t_{k+1} \\ 0 & \text{otherwise} \end{cases}$$

$$N_k^d(t) = \frac{t - t_k}{t_{k+d} - t_k} N_k^{d-1}(t) + \frac{t_{k+d+1} - t}{t_{k+d+1} - t_{k+1}} N_{k+1}^{d-1}(t)$$

- Local support of $N_k^d(t)$, meaning $N_k^d(t) = 0$ for $t \notin [t_k, t_{k+d+1}]$

B-splines

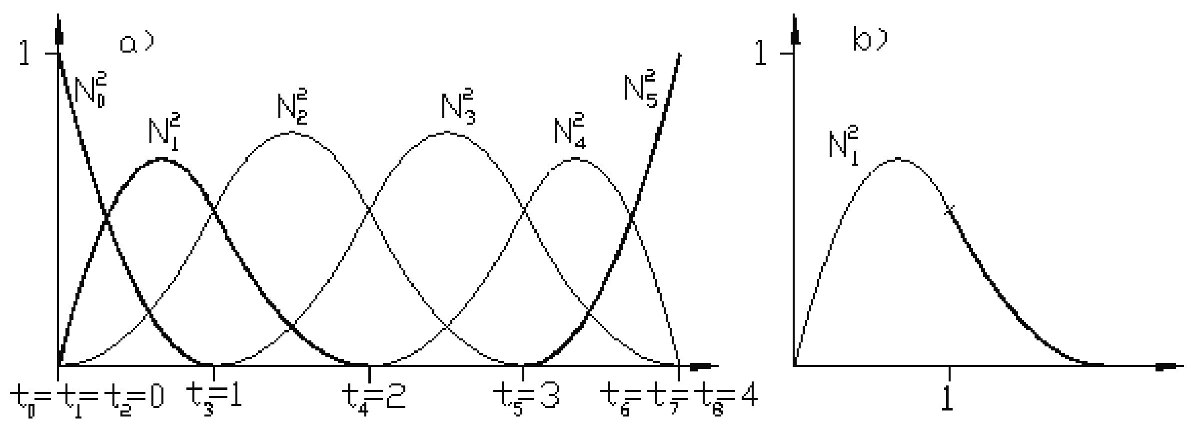
- Quadratic B-spline basis



B-splines

Knot multiplicity

- If t_j is a simple knot such that $t_{j-1} \neq t_j \neq t_{j+1}$ then $N_k^d(t_j)$ is C^{d-1} continuous
- For a knot $s = t_{j+1} = \ldots = t_{j+\mu}$ of multiplicity μ the B-Splines N_k^d of degree d are $C^{d-\mu}$ continuous

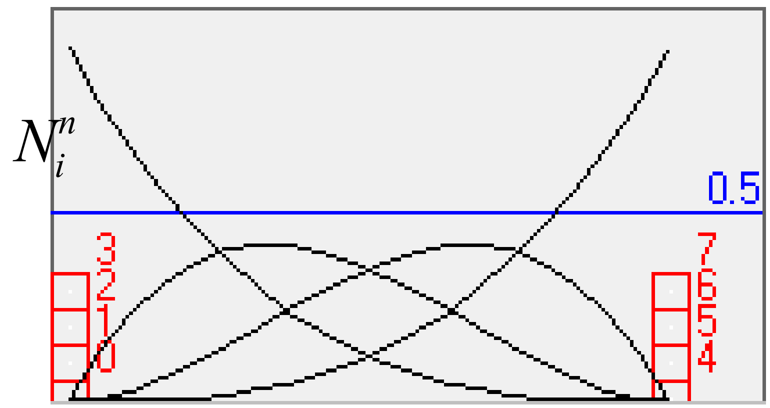


B-splines

Bernstein polynomials

- B-splines N_k^d contain the Bernstein polynomials B_i^d as a special case of knot multiplicity

$$\mathbf{T} = \left(\underbrace{0,\ldots,0}_{d+1}, \underbrace{1,\ldots,1}_{d+1} \right)$$



Rational B-spline

$$p(t) = \frac{\sum_{k=0}^{n} w_k b_k B_{k,d}(t)}{\sum_{k=0}^{n} w_k B_{k,d}(t)}$$

- Like having [x, y, z, w] with varying weights
- Useful for building exact conics (circle, etc.)
- Projective invariance

B-spline surfaces

- Tensor product surfaces
  - Also works for Bézier curves/splines (Bézier patches)

$$P(u,v) = \sum_{k_u=0}^{n_u} \sum_{k_v=0}^{n_v} \mathbf{b}_{k_u,k_v} \underbrace{B_{k_u,d_u}(u)\, B_{k_v,d_v}(v)}_{\text{tensor product surface}}$$

control points "mesh"