

# parametric spline curves

# curves

---

- used in many contexts
  - fonts
  - animation paths
  - shape modeling
- different representation
  - implicit curves
  - parametric curves
    - mostly used

# implicit representation for 2D curves

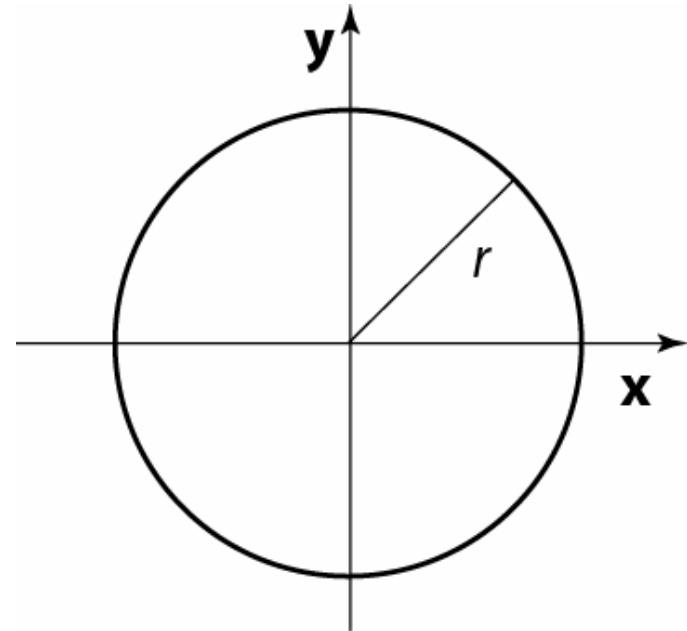
---

- curves can be represented implicitly as

$$f(\mathbf{p}) = f(x, y) = 0$$

- example: circle of radius  $r$  centered at origin

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



# parametric representation for 2D curves

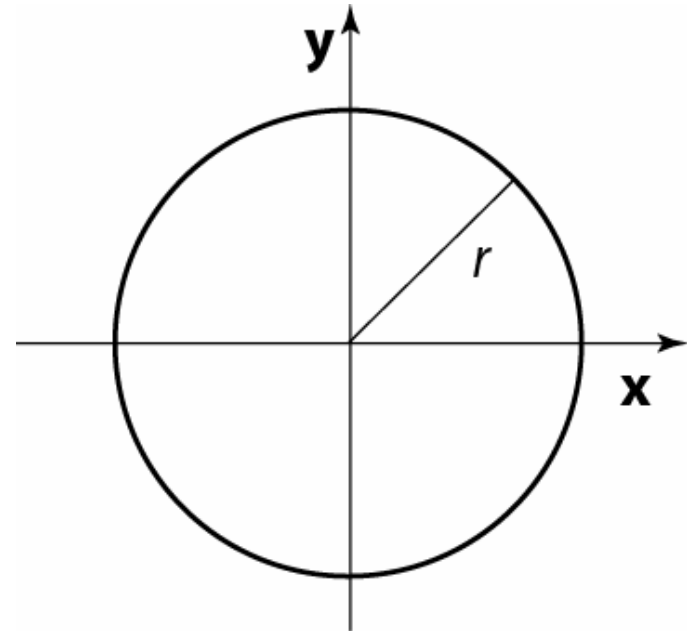
---

- curves can be represented parametrically as

$$\mathbf{p}(u) = \begin{cases} x = f_x(u) \\ y = f_y(u) \end{cases}$$

- example: circle of radius  $r$  centered at origin

$$\begin{cases} x = r \cos(u) \\ y = r \sin(u) \end{cases}$$



# parametric representation of splines

---

- general parametric curve can be written as

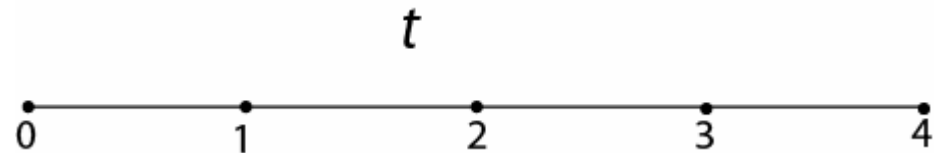
$$\mathbf{p}(t) = f(t) \quad t \in [0, N]$$

- goals when defining  $f$ 
  - smoothness
  - predictable and local control
  - efficiency

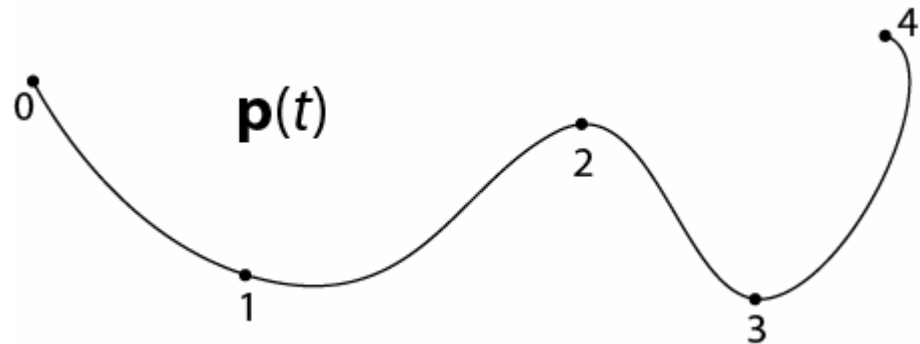
# parametric representation of splines

---

- splines: piecewise parametric polynomials
  - polynomials are smooth
  - controlled by small number of *local control points*
  - discontinuities at integer intervals



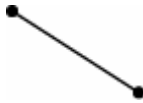
$$\mathbf{p}(t) = f(t) \quad t \in [0, N]$$



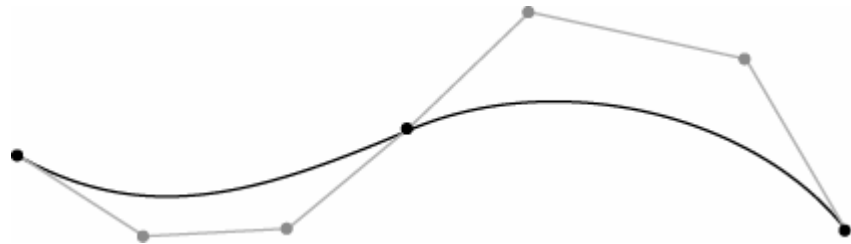
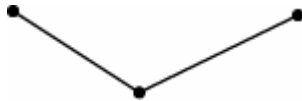
# splines - intuition

---

- define segment by “blending” *control points*



- join segments to form curve



# defining splines

---

- pick segment interpolating function
- impose constraints to define segments
  - i.e. control points that define the spline
- impose constraints to join segments together



# interpolating vs. approximating splines

---

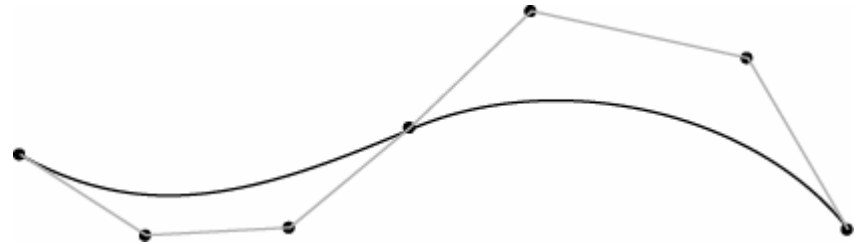
- interpolating

- pass through control points



- approximating

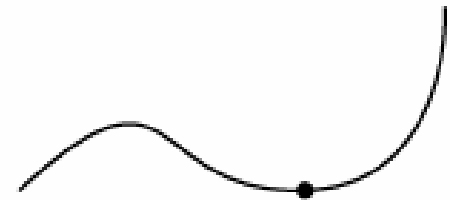
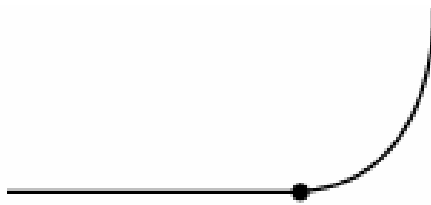
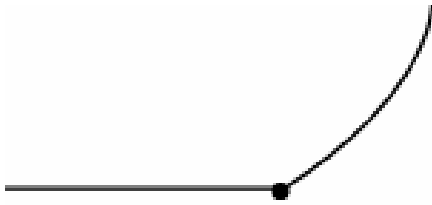
- guided by control points



# smoothness

---

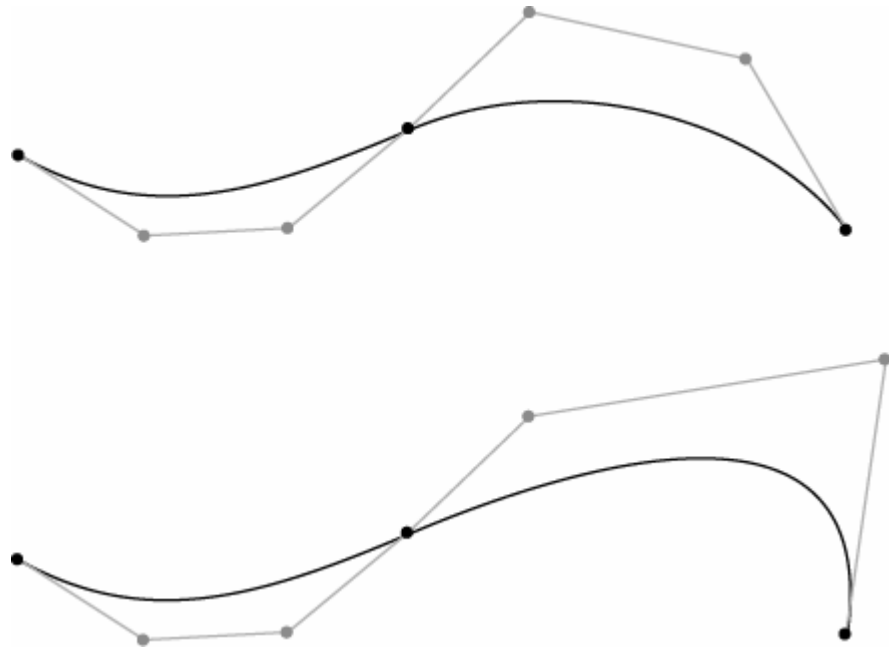
- smoothness described by degree of continuity
  - $C^0$ : same position at each side of joints
  - $C^1$ : same tangent at each side of joints
  - $C^2$ : same curvature at each side of joints
  - $C^n$ :  $n$ -th derivative defined at joints



# control

---

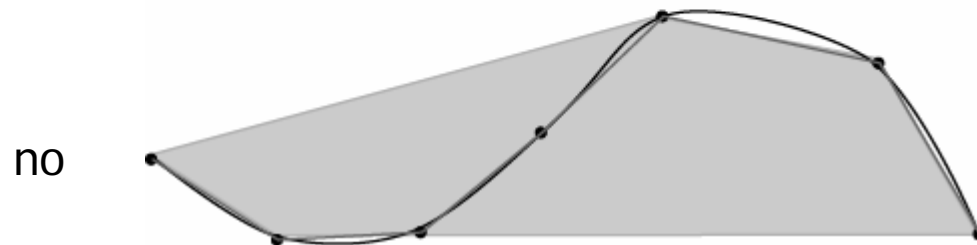
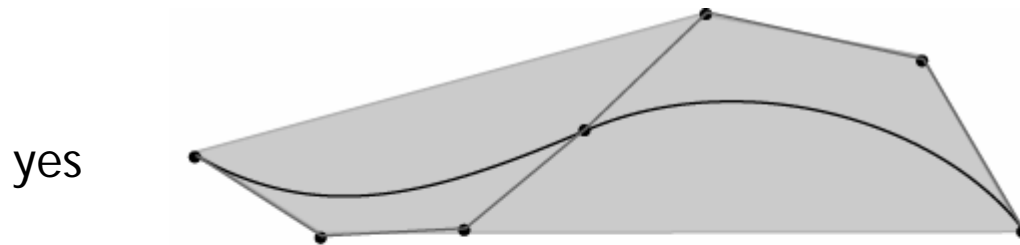
- local control
  - changing control points only affect locally the curve
    - easy to control
  - true for all splines



# control

---

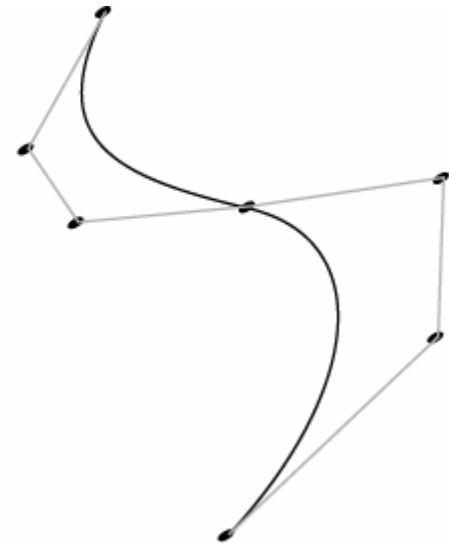
- convex hull property
  - convex hull: smallest convex region enclosing all points
    - predictable behavior
    - more efficient operations
  - only some splines



# efficiency

---

- affine invariance
  - transforming the spline same as transforming controls
  - efficient algorithms, esp. combined with convex hull
  - true for all used splines



# piecewise linear splines

---

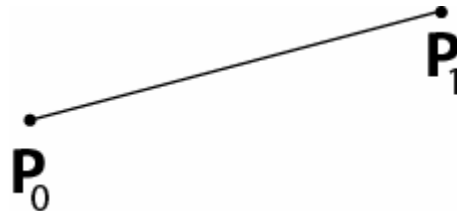
- each segment is a linear function

$$\mathbf{p}(t) = t\mathbf{a} + \mathbf{b} \quad t \in [0,1]$$

- impose endpoint constraints

$$\begin{cases} \mathbf{p}(0) = \mathbf{p}_0 \\ \mathbf{p}(1) = \mathbf{p}_1 \end{cases} \Rightarrow \begin{cases} \mathbf{a} = \mathbf{p}_1 - \mathbf{p}_0 \\ \mathbf{b} = \mathbf{p}_0 \end{cases}$$

$$\mathbf{p}(t) = \mathbf{p}_0 + t(\mathbf{p}_1 - \mathbf{p}_0) \quad t \in [0,1]$$



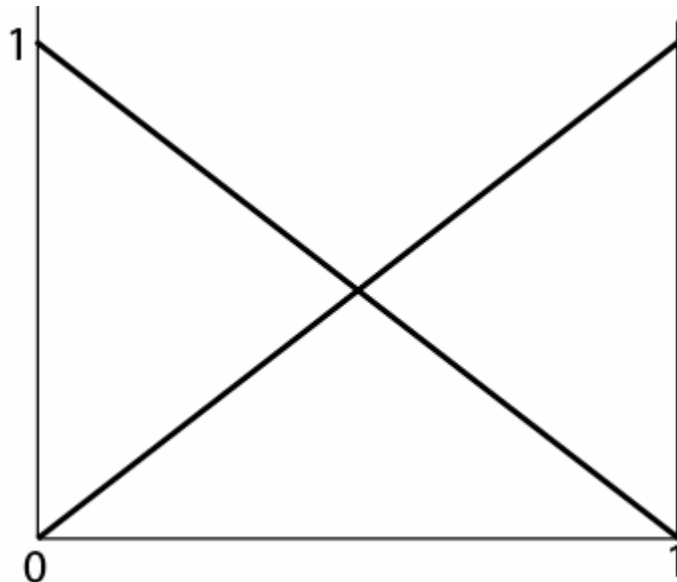
# point blending interpretation

---

- can interpret as blending of points

$$\mathbf{p}(t) = (1-t)\mathbf{p}_0 + t\mathbf{p}_1 = b_0(t)\mathbf{p}_0 + b_1(t)\mathbf{p}_1 \quad t \in [0,1)$$

- blending functions do not depend on points
  - different intervals only change control points



# matrix notation

---

- write blending functions more conveniently

$$\mathbf{p}(t) = (1-t)\mathbf{p}_0 + t\mathbf{p}_1$$

$$\mathbf{p}(t) = \begin{bmatrix} t & 1-t \end{bmatrix} \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \end{bmatrix}$$



# joining line segments

---

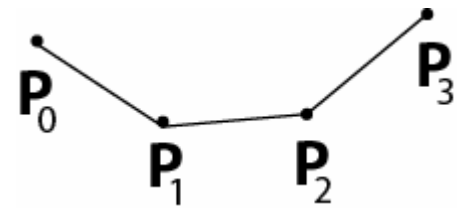
- impose  $C^0$  continuity at joints

- first segment  $\mathbf{p}^0(t) \rightarrow \begin{cases} \mathbf{p}^0(0) = \mathbf{p}_0^0 \\ \mathbf{p}^0(1) = \mathbf{p}_1^0 \end{cases}$

- second segment  $\mathbf{p}^1(t) \rightarrow \begin{cases} \mathbf{p}^1(0) = \mathbf{p}_0^1 \\ \mathbf{p}^1(1) = \mathbf{p}_1^1 \end{cases}$

- implies

$$\mathbf{p}^0(1) = \mathbf{p}^1(0) \rightarrow \mathbf{p}_1^0 = \mathbf{p}_0^1$$



- general formula

- appropriately rename control points

$$\mathbf{p}(t) = b_0(t-k)\mathbf{p}_k + b_1(t-k)\mathbf{p}_{k+1} \quad t \in [0, N], k = \text{floor}(t)$$

# Hermite splines

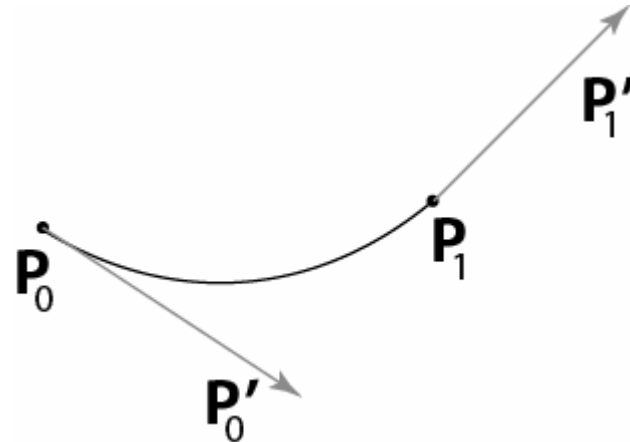
---

- each segment is a cubic polynomial function

$$\mathbf{p}(t) = \mathbf{a}t^3 + \mathbf{b}t^2 + \mathbf{c}t + \mathbf{d}$$

- impose endpoints and tangents constraints

$$\begin{cases} \mathbf{p}(0) = \mathbf{p}_0 \\ \mathbf{p}(1) = \mathbf{p}_1 \\ \mathbf{p}'(0) = \mathbf{p}'_0 \\ \mathbf{p}'(1) = \mathbf{p}'_1 \end{cases}$$



# Hermite splines

---

$$\mathbf{p}(t) = \mathbf{a}t^3 + \mathbf{b}t^2 + \mathbf{c}t + \mathbf{d}$$

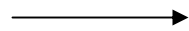
$$\mathbf{p}'(t) = 3\mathbf{a}t^2 + 2\mathbf{b}t + \mathbf{c}$$

$$\mathbf{p}(0) = \mathbf{d}$$

$$\mathbf{p}(1) = \mathbf{a} + \mathbf{b} + \mathbf{c} + \mathbf{d}$$

$$\mathbf{p}'(0) = \mathbf{c}$$

$$\mathbf{p}'(1) = 3\mathbf{a} + 2\mathbf{b}$$



$$\mathbf{a} = 2\mathbf{p}_0 - 2\mathbf{p}_1 + \mathbf{p}'_0 + \mathbf{p}'_1$$

$$\mathbf{b} = -3\mathbf{p}_0 + 3\mathbf{p}_1 - 2\mathbf{p}'_0 - \mathbf{p}'_1$$

$$\mathbf{c} = \mathbf{p}'_0$$

$$\mathbf{d} = \mathbf{p}_0$$

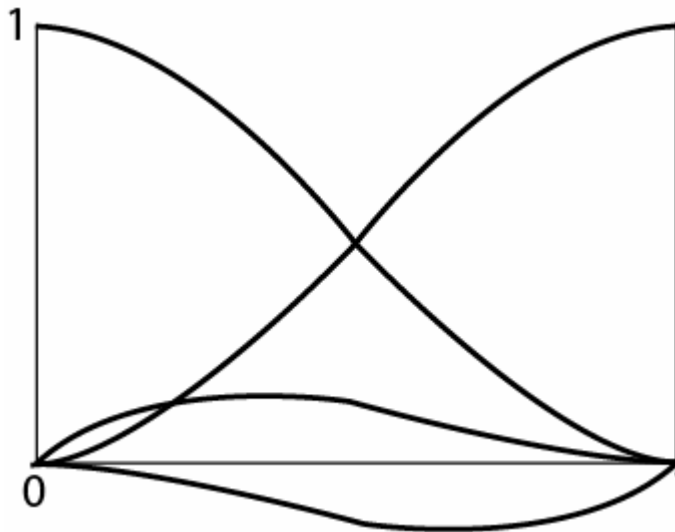
# Hermite

---

- matrix formulation

$$\mathbf{p}(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 & 2 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}'_0 \\ \mathbf{p}'_1 \end{bmatrix}$$

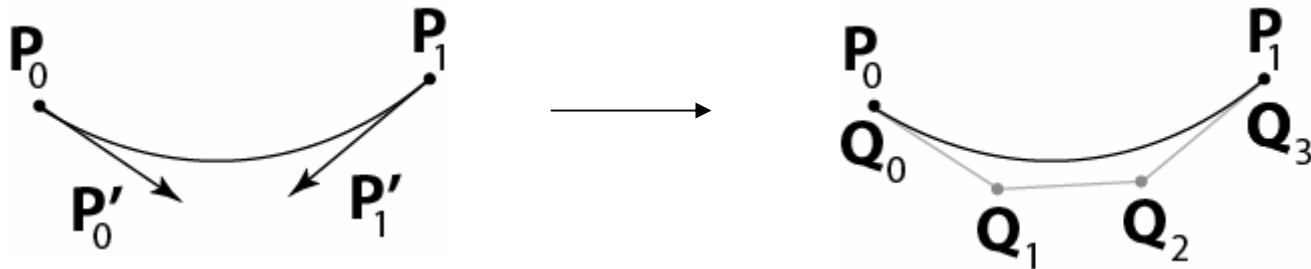
- blending functions



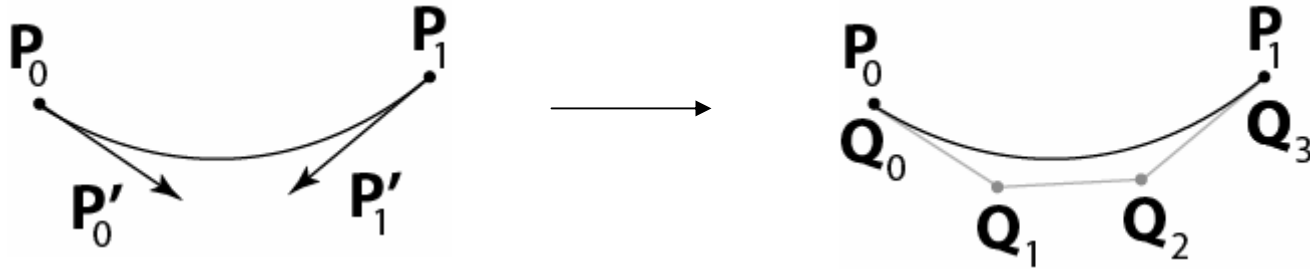
# Bezier splines

---

- Hermite splines has points and vectors controls
  - would like to use just points
  - insight: specify tangents as difference of points
    - choose appropriate scaling value, see later



# Bezier splines



$$\mathbf{p}_0 = \mathbf{q}_0$$

$$\mathbf{p}_1 = \mathbf{q}_3$$

$$\mathbf{p}'_0 = 3(\mathbf{q}_1 - \mathbf{q}_0)$$

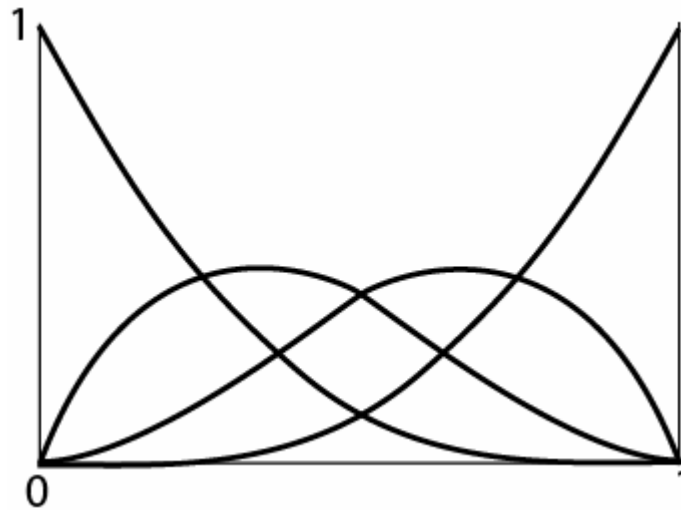
$$\mathbf{p}'_1 = 3(\mathbf{q}_3 - \mathbf{q}_2)$$

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

# Bezier splines

---

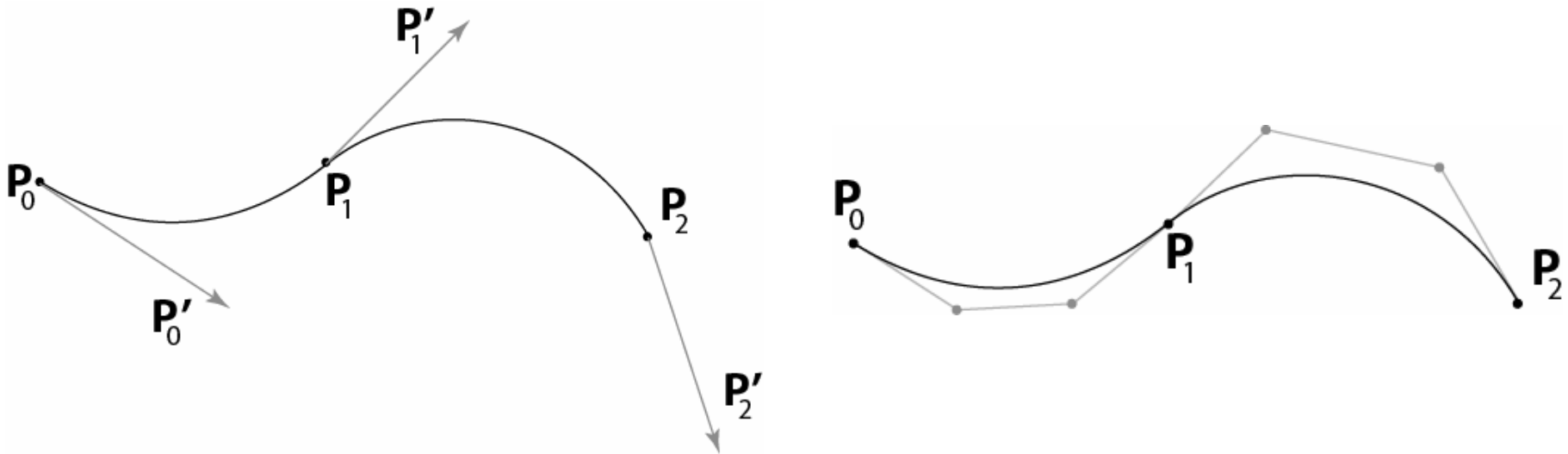
- blending functions



# piecewise cubic splines – smoothness

---

- $C^1$  at joints by imposing equal tangents
  - Hermite: same tangents
  - Bezier: collinear control points
    - geometric continuity if length of tangent differs





# piecewise cubic splines – control

---

- local control
  - comes from the formulation by segments
  - for each segment, curve defined by 4 control points
- convex hull
  - when blending positions

$$b_i(t) \geq 0 \quad \text{and} \quad \sum_{i=0}^3 b_i(t) = 1$$

# piecewise cubic splines – affine invariance

---

- affine invariance
  - affine is combination of linear and translation
  - blending functions sum to 1

$$\begin{aligned} X(\mathbf{p}(t)) &= M\mathbf{p}(t) + \mathbf{t} = M\left(\sum_{i=0}^3 b_i(t)\mathbf{p}_i\right) + \mathbf{t} = \\ &= \sum_{i=0}^3 b_i(t)M\mathbf{p}_i + \sum_{i=0}^3 b_i(t)\mathbf{t} = \\ &= \sum_{i=0}^3 b_i(t)M(\mathbf{p}_i + \mathbf{t}) = \sum_{i=0}^3 b_i(t)MX(\mathbf{p}_i) \end{aligned}$$

# Bezier splines

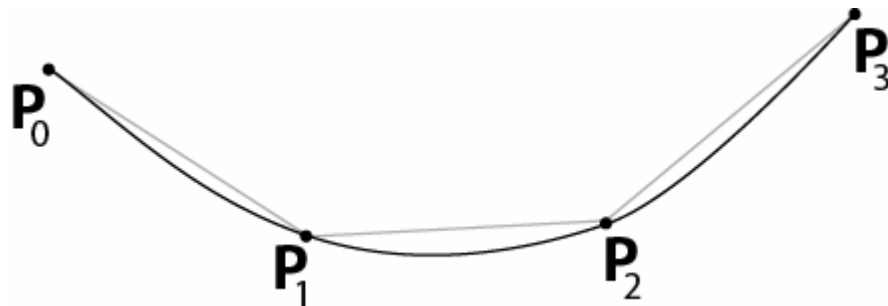
---

- widely used, especially in 2D
  - primitive in PDF
- represent  $C^1$  and  $C^0$  curves with corners
- easily add point at any position

# Catmull-Rom splines

---

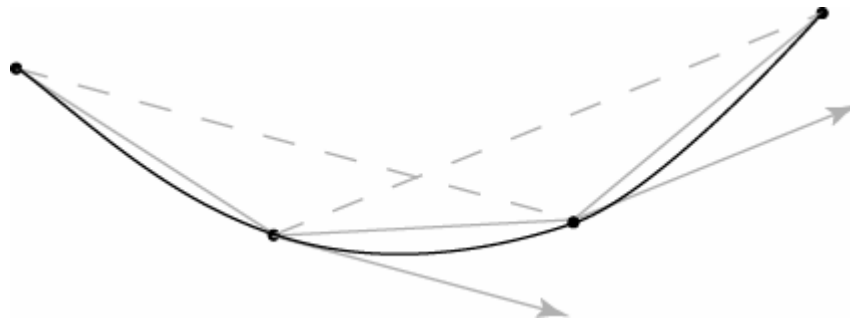
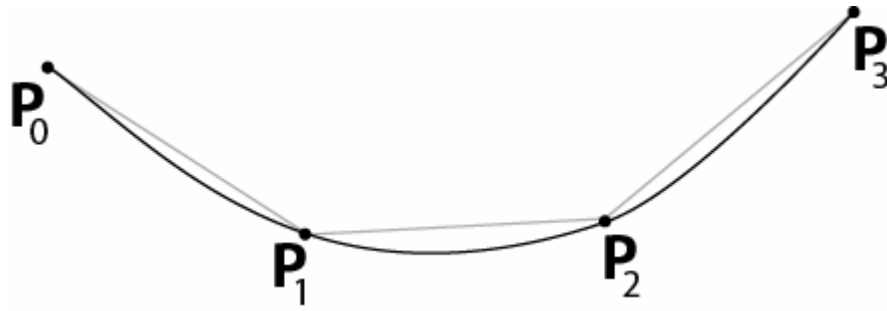
- interpolating spline
  - no convex hull property
- as Hermite, derivatives automatically determined
  - using adjacent control points
  - end tangent using either adding point or zeros



# Catmull-Rom splines

---

$$\mathbf{p}'_k = (\mathbf{p}_{k+1} - \mathbf{p}_{k-1}) / 2$$



# drawing splines

---

- approximate with a sequence of line segments
  - efficiency: fast evaluation, small number of segments
  - guarantees on accuracy
- approaches
  - uniform subdivision in  $t$  (fast)
  - recursive subdivision (small number of segments)

# uniform subdivision

---

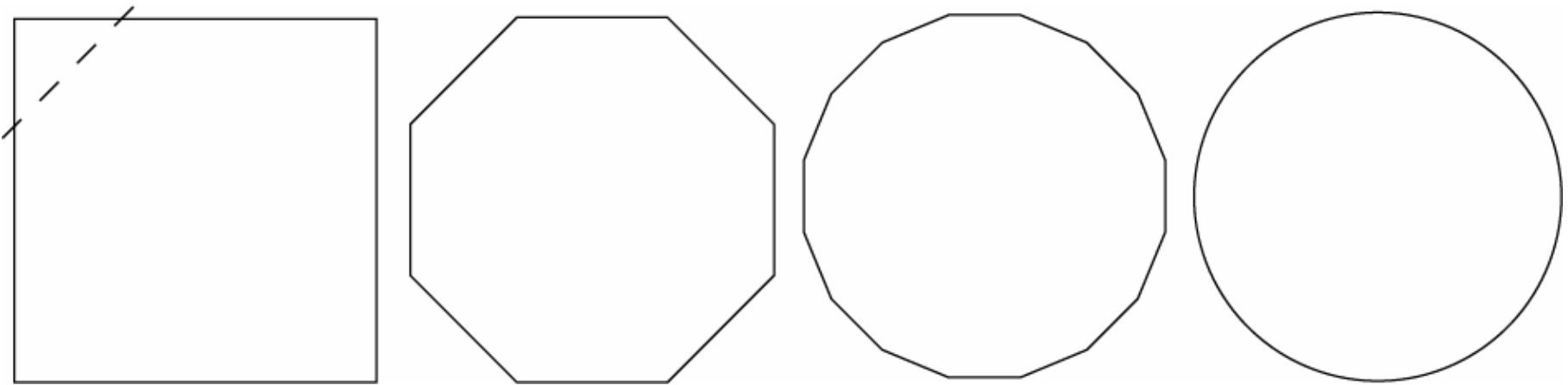
- evaluate spline at fixed  $t$  intervals
  - can be done efficiently



# adaptive subdivision - Bezier

---

- recursively subdivide spline
- until line segments approximate well curve

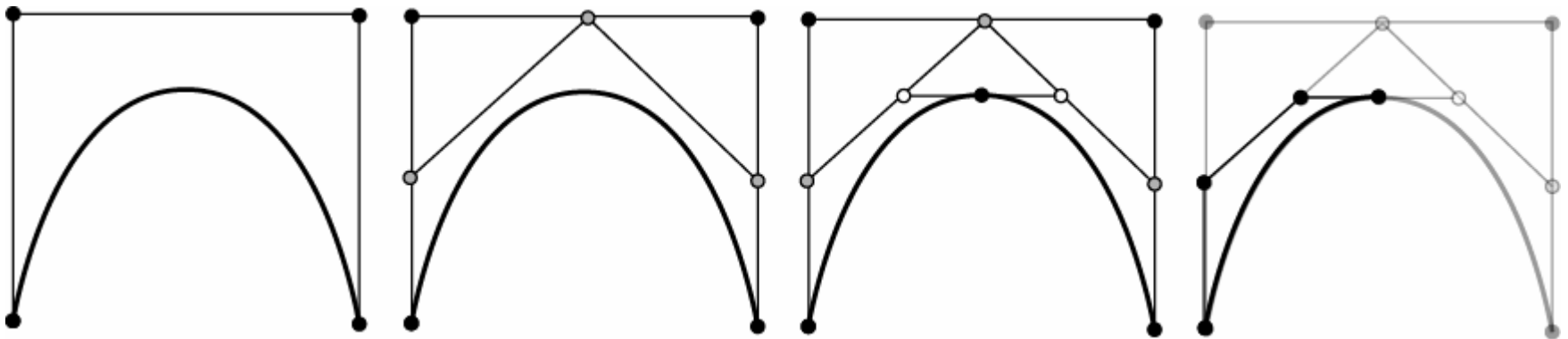




# De Casteljau algorithm - Bezier

---

- recursively do
  - connect midpoints of the control polygons
  - connect midpoints of the new segments
  - the midpoint of this last segment is on the curve
  - and splits the curve in two Bezier segments
- stop when control polygon is close to collinear



# B-Splines

---

- would like  $C^2$  continuity at joints
  - give up interpolation
- impose 3 continuity constraints at joints

$$\mathbf{p}(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{q}_{k-1} \\ \mathbf{q}_k \\ \mathbf{q}_{k+1} \\ \mathbf{q}_{k+2} \end{bmatrix}$$

# other splines

---

- many other types
- non-uniform B-splines
  - discontinuities not evenly spaces
- non-uniform rational B-splines (NURBS)
  - ratios of non-uniform B-splines
  - invariance under perspective
  - can represent conic sections exactly
  - often used in 3D

# spline equivalence

---

- all splines seen so far are equivalent
  - represented by 4x4 matrices
- can convert control points from one to other
  - algorithms can be based on the most efficient
  - UIs can be based on the most user-friendly

# 2D vs. 3D splines

---

- often use 2D splines in 3D
  - by projecting onto a plane
- 3D parametric splines have same formulation
  - just use 3D vectors vs. 2D ones