Curves

Toni Sellarès
Universitat de Girona

Curves

- Many objects we want to model are not straight.
  - Ex. Text, sketches, etc.
- How can we represent a curve?
  - A large number of points on the curve.
  - Approximate with connected line segments.
    - “piecewise linear approximation”



Accuracy/Space Trade-off

Problem

- Piecewise linear approximations require many pieces to look good (realistic, smooth, etc.).
- Set of individual curve points would take large amounts of storage.

Solution

- Higher-order formulae for coordinates on curve.
- If a simple formula won't work, subdivide curve into pieces that can be represented by simple formulae.
- May still be an approx., but uses much less storage.
- Downside: harder to specify and render.

Curve Representations

- Explicit: y = f(x)
  - Example: $y = x^2$
  - The curve must be a function (only one value of y for each x)
- Parametric: $(x(t),y(t)) = (f(t),g(t))$
  - Example: $(x(t), y(t)) = (\cos t, \sin t)$
  - Easy to specify, modify, control
  - Easy to join curve segments smoothly
- Implicit: F(x,y) = 0
  - Example: $x^2-y^2-r^2=0$
  - Could exist several values of y for each x
  - Need constraints to model just one part of a curve
  - Joining curves together smoothly is difficult
  - Hard to specify, modify, control

Curve examples

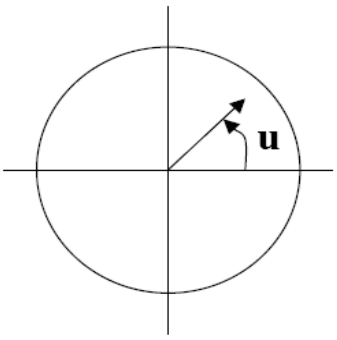
- Two Ways to Define a Circle

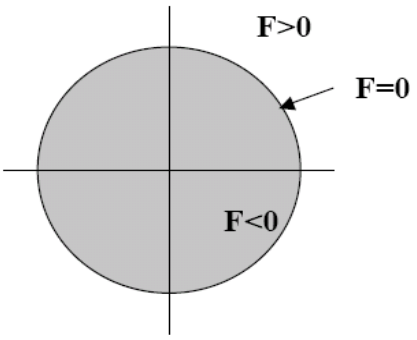
Parametric:

$x(u) = r \cos(u)$
$y(u) = r \sin(u)$

Implicit:

$F(x,y) = x^2 + y^2 - r^2 = 0$





Parametric Equations

Cubic preferred

Balance between *flexibility* and *complexity* in specifying and computing shape.

$$Q(t)=(x(t),y(t),z(t))$$

$$x(t) = a_x t^3 + b_x t^2 + c_x t + d_x$$
$$y(t) = a_y t^3 + b_y t^2 + c_y t + d_y$$
$$z(t) = a_z t^3 + b_z t^2 + c_z t + d_z$$
$$t \in [0,1]$$

Matricial expression

$$Q(t) = T \circ C$$

Where:

$$T = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \qquad C = \begin{bmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \\ d_x & d_y & d_z \end{bmatrix}$$

Joining Curve Segments

G^0 geometric continuity: Two curve segments join together.

G^1 geometric continuity: The direction of the two segments' tangent vectors are equal at the join point.

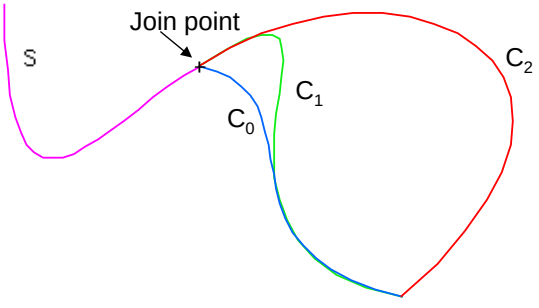
C^0 continuity: Curves share the same point where they join.

C^1 continuity: Tangent vectors of the two segments are equal in magnitude and direction (share the same parametric derivatives).

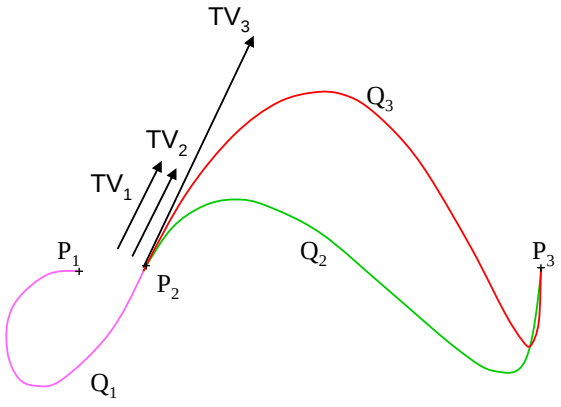
C^2 continuity: Curves share the same parametric second derivatives where they join.

$C^1 \Rightarrow G^1$, unless tangent vector = [0, 0, 0].

Joining Examples

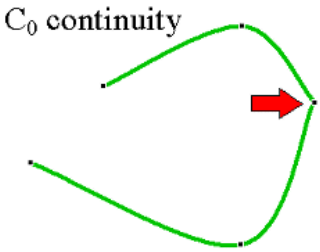


S joins C_0, C_1, and C_2 with C^0, C^1, and C^2 continuity, respectively.

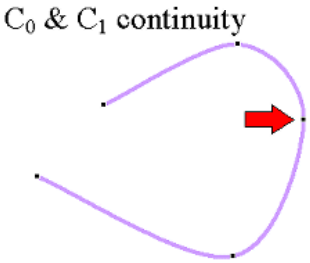


Q_1 and Q_2 are C^1 continuous because their tangents, TV_1 and TV_2, are equal. Q_1 and Q_3 are only G^1 continuous.

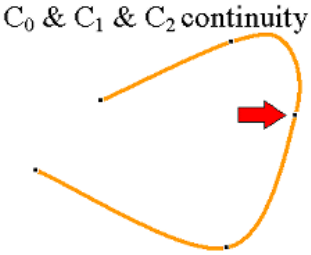
Continuity examples



Continuous in position

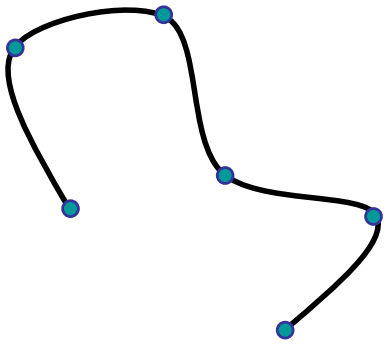


Continuous in position and tangent vector



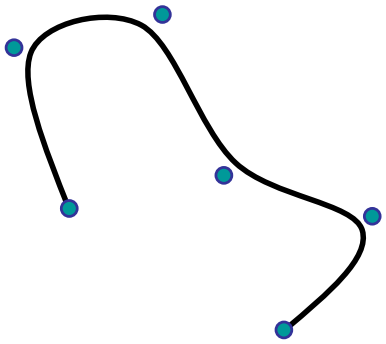
Continuous in position, tangent, and curvature

Curve Fitting: Interpolation and Approximation



Interpolation

curve must pass
through control points



Approximation

curve is influenced
by control points

Curve representations: polynomial bases

- Polynomials are easy to analyze, derivatives remain polynomial, etc.
- Monomial basis $\{1, x, x^2, x^3, \ldots\}$.
  - Coefficients are geometrically meaningless.
  - Manipulation is not robust.
- Number of coefficients = polynomial rank.

Polynomial Interpolation

- An *n*-th degree polynomial fits a curve to n+1 points.
  - Example: fit a second degree curve to three points:
    - **$x(u) = au^2 + bu + c$.**
    - control points to interpolate**:** **(u_1, x_1), (u_2, x_2), (u_3, x_3)**.
    - solve for coefficients **(a, b, c):** 3 linear eqns, 3 unknowns.
  - called Lagrange Interpolation.
  - result is a curve that changes to any control point affects entire curve (non-local) [*this method is poor*].
- We usually want the curve to be as smooth as possible:
  - minimize the wiggles.
  - high-degree polynomials are bad.

Lagrange Interpolation

The *Lagrange interpolating polynomial* is the polynomial of degree n-1 that passes through the n points,

$$(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n),$$

and is given by:

$$P(x) = y_1 \frac{(x - x_2)\cdots(x - x_n)}{(x_1 - x_2)\cdots(x_1 - x_n)} + y_2 \frac{(x - x_1)(x - x_3)\cdots(x - x_n)}{(x_2 - x_1)(x_2 - x_3)\cdots(x_2 - x_n)} + \cdots$$

$$+ y_n \frac{(x - x_1)\cdots(x - x_n)}{(x_n - x_1)\cdots(x_1 - x_n)}$$

$$= \sum_{i=1}^{n} y_i \prod_{j \neq i} \frac{(x - x_j)}{(x_i - x_j)}$$

Splines

A *spline* is a parametric curve defined by *control points:*

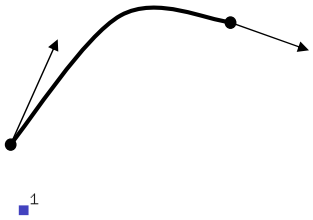
- term “spline” dates from engineering drawing, where a spline was a piece of flexible wood used to draw smooth curves.
- control points are *adjusted by the user* to control shape of curve.
- wood splines:
  - have second-order continuity,
  - pass through the control points.



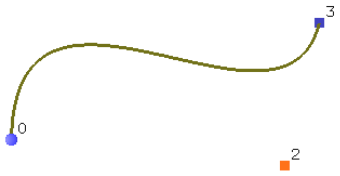
Drawing curves with wood splines

Spline Curve Families

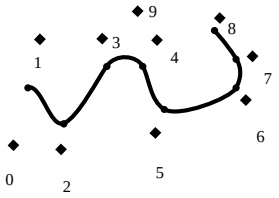
- Hermite cubic
  - Defined by its 2 endpoints and tangent vectors at endpoints.
  - Interpolates all its control points.
  - Special case of Bezier and B-Spline.



- Bezier
  - Interpolates first and last control points.
  - Curve is tangent to first and last segments of control polygon.
  - Easy to subdivide.
  - Curve segment lies within convex hull of control polygon.



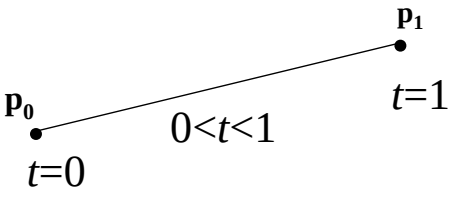
- B-Spline
  - Not guaranteed to interpolate control points.
  - Curve segment lies within convex hull of control polygon.
  - Greater local control than Bezier.



Linear Interpolation

- Linear interpolation (Lerp) is a common technique for generating a new value that is somewhere in between two other values.

- By interpolating between two points p_0 and p_1 by some parameter t we obtain a segment:



Linear Interpolation

Three ways of writing a segment:

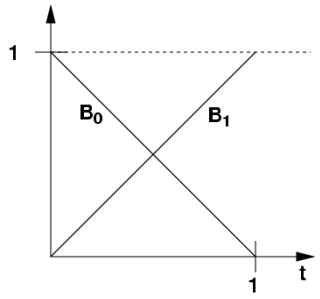
1. Weighted average of the control points: $Q(t) = (1-t)p_0 + tp_1$.

   Basis Functions:

   $B_0(t)=1-t;\ B_1(t)=t$

   $B_0(t)+B_1(t)=1$

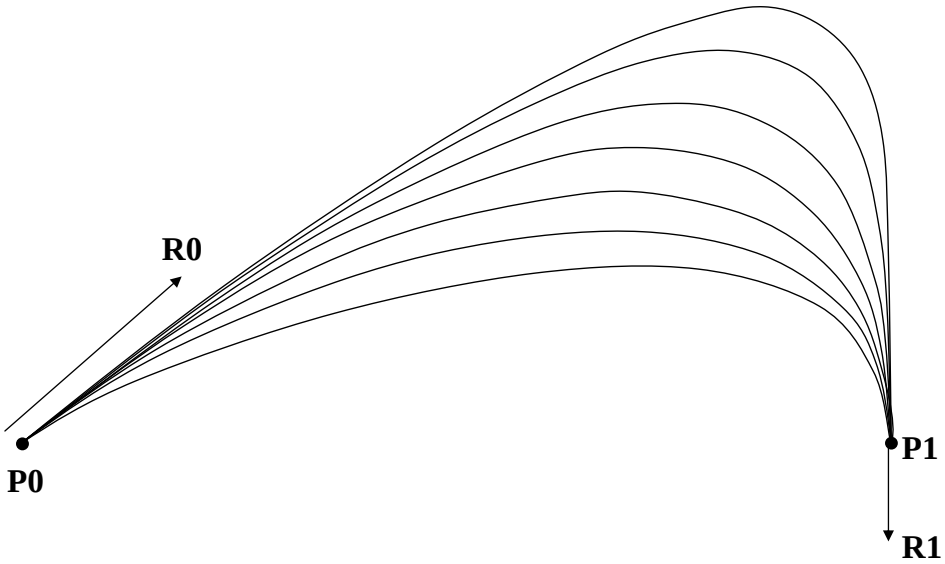
   $Q(t) = B_0(t)p_0 + B_1(t)p_1$



2. Polynomial in t: $Q(t) = (p_1-p_0)t + p_0$
3. Matrix form:

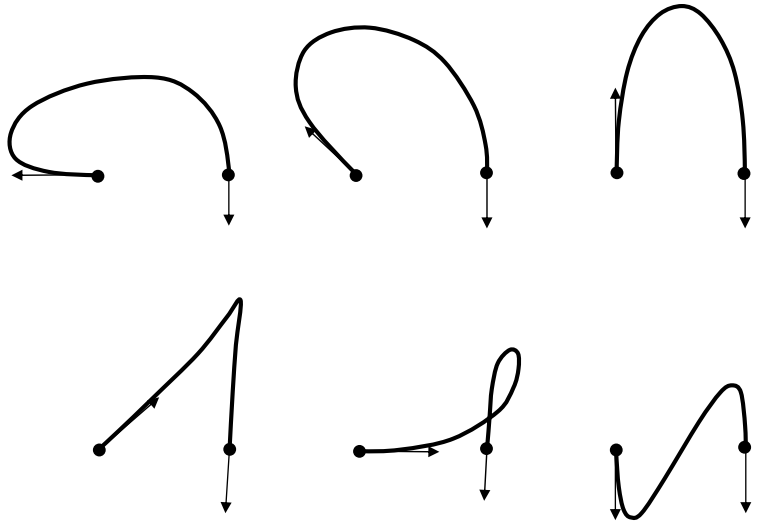
$$Q(t) = \begin{bmatrix} t & 1 \end{bmatrix} \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \end{bmatrix}$$

Hermite Curve Examples



The set of Hermite curves that have the same values for the endpoints P0 and P1, tangent vectors R0 and R1 of the same direction, but with different magnitudes for R0. The magnitude of R1 remains fixed.

Hermite Curve Examples

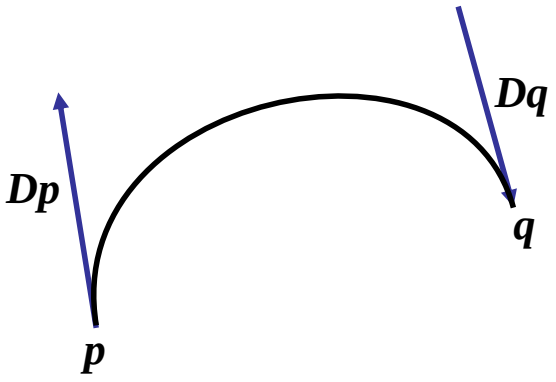


All tangent vector magnitudes are equal, but the direction of the left tangent vector varies.

Hermite Cubic Curves

Is a cubic curve for which the user provides:

The endpoints of the curve: *p*, *q*;
The parametric derivatives of the curve at the endpoints: *Dp, Dq.*



$$\boldsymbol{P}(t) = at^3 + bt^2 + ct + d$$

$$t \text{ in } [0,1]$$

$$\boldsymbol{P}(0) = \boldsymbol{p}$$

$$\boldsymbol{P}(1) = \boldsymbol{q}$$

$$\boldsymbol{P}'(0) = D\boldsymbol{p}$$

$$\boldsymbol{P}'(1) = D\boldsymbol{q}$$

Hermite Coefficients

$$\boldsymbol{P}(t) = at^3 + bt^2 + ct + d$$

$$\boldsymbol{P}(0) = \boldsymbol{p}$$

$$\boldsymbol{P}(1) = \boldsymbol{q}$$

$$\boldsymbol{P}'(0) = D\boldsymbol{p}$$

$$\boldsymbol{P}'(1) = D\boldsymbol{q}$$

$$\boldsymbol{P}(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

$$\boldsymbol{P}'(t) = \begin{bmatrix} 3t^2 & 2t & 1 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

For each coordinate, we have 4 linear equations in 4 unknowns

Boundary Constraint Matrix

$$\boldsymbol{P}(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

$$\boldsymbol{P}'(t) = \begin{bmatrix} 3t^2 & 2t & 1 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

$$\begin{bmatrix} \boldsymbol{p} \\ \boldsymbol{q} \\ \boldsymbol{Dp} \\ \boldsymbol{Dq} \end{bmatrix} = \overbrace{\begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix}}^{\boldsymbol{N_H}} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

Hermite Matrix

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \underbrace{\begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}}_{M_H = (N_H)^{-1}} \underbrace{\begin{bmatrix} \boldsymbol{p} \\ \boldsymbol{q} \\ \boldsymbol{Dp} \\ \boldsymbol{Dq} \end{bmatrix}}_{G_H}$$

$$P(t) = T \circ M_H \circ G_H$$

Hermite Basis Functions

$$\boldsymbol{P}(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} M_H \begin{bmatrix} \boldsymbol{p} \\ \boldsymbol{q} \\ \boldsymbol{Dp} \\ \boldsymbol{Dq} \end{bmatrix} = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{p} \\ \boldsymbol{q} \\ \boldsymbol{Dp} \\ \boldsymbol{Dq} \end{bmatrix}$$

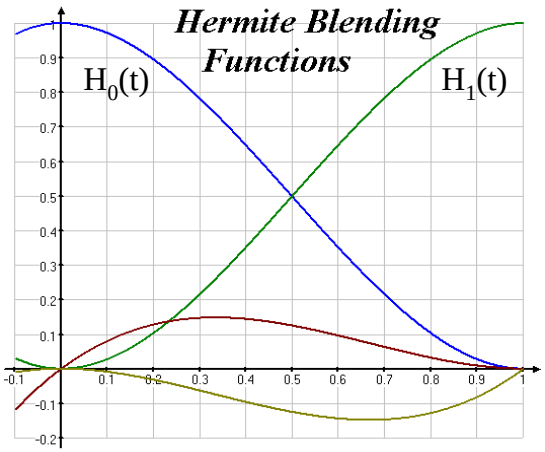
$$\boldsymbol{P}(t) = \boldsymbol{p}\left(2t^3 - 3t^2 + 1\right) + \boldsymbol{q}\left(-2t^3 + 3t^2\right) + \boldsymbol{Dp}\left(t^3 - 2t^2 + t\right) + \boldsymbol{Dq}\left(t^3 - t^2\right)$$

$$\boldsymbol{P}(t) = \boldsymbol{p}H_0(t) + \boldsymbol{q}H_1(t) + \boldsymbol{Dp}H_2(t) + \boldsymbol{Dq}H_3(t)$$

Hermite Basis Functions

- Functions:

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



are called *Hermite Basis or Blending Functions.*

- Observe that: $H_0(t)+H_1(t)+H_2(t)+H_3(t) \neq 1$.

- An Hermite cubic curve can be thought as a higher order extension of linear interpolation:

$$P(t)=H_0(t)p + H_1(t)q+H_2(t)Dp+H_3(t)Dq$$

Displaying Hermite curves

- Evaluate the curve at a fixed set of parameter values and join the points with straight lines.

From Hermite to Bezier curves

- Hermite curves are difficult to use because we usually have control points but not derivatives.
- However, Hermite curves are the basis of the Bezier curves.
- Bezier curves are more intuitive, since we only need to specify control points.

Bezier Curves of degree *d*

- Bezier curve is an *approximation* of given control points.
- Bezier curve of degree *d* is defined over *d+1* control points $\{P_i\}_{i=0,\ldots,d}$.
- Have two formulations:
  - Algebraic (higher order extension of linear interpolation).
  - Geometric.

Bezier Curves: Algebraic Formulation

- The user supplies *d+1 control points*:

$$\boldsymbol{p}_i \; ; \; i=0, \ldots , d.$$

- Write the curve of degree *d* as:

$$Q(t)=\sum_{i=0}^{d} B_i^d(t)\, p_i \qquad B_i^d(t)=\binom{d}{i} t^i (1-t)^{d-i}$$

$$\binom{d}{i}=\begin{cases} d!/\,i!*(d-i)! & when \;\; 0\le i\le d \\ 0 & otherwise \end{cases}$$

- The functions $B_i^d(t)$ are the *Bernstein basis polynomials* or *Bezier blending functions* of degree *d.*

Bezier Curves: Algebraic Formulation

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

$$=(1-t)^d\, p_0 + t(1-t)^{d-1} p_1 + \ldots + t^{d-1(1-t)} p_{d-1} + t^d p_d.$$

- *Q(0)=p_0* and *Q(1)=p_d*, that means the Bézier curve lies on p_0 and p_d.
- *Q'(0)=d(p_1 - p_0)* and *Q'(1)=d(p_d - p_{d-1})* (tangents in start and end points).

Bernstein Basis Polynomials

- They are all positive in interval [0,1],
- Their sum is equal to 1: $\sum_{i=0}^{d} B_i^d(t) = 1 \quad \forall t \in [0,1]$

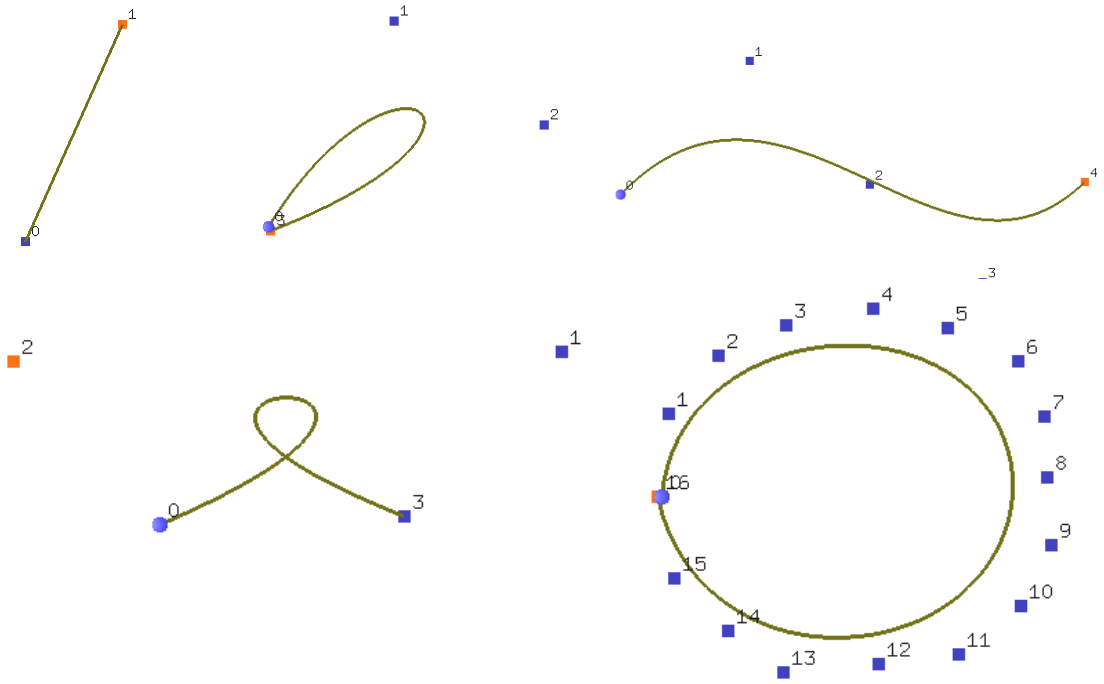
Proof:

$$1 = ((1-t)+t)^d =$$

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

- Symmetry: $B_i^d(t) = B_{d-i}^d(1-t)$
- Recursion: $B_i^d(t) = t \cdot B_{i-1}^{d-1}(t) + (1-t) \cdot B_i^{d-i}(t)$

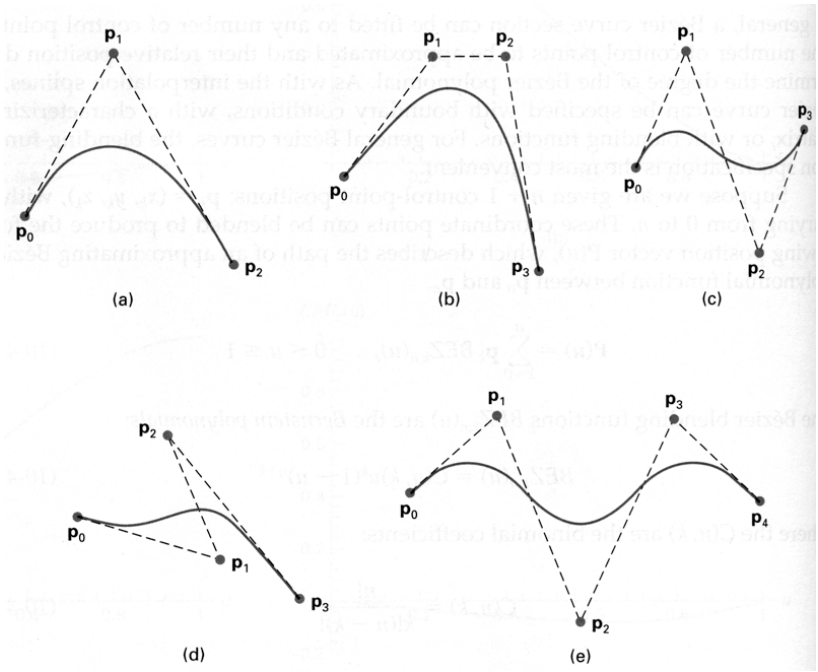
Bezier Curves



Cubic Bezier Curves (d=3)

For cubic Bezier curves, two control points define endpoints, and two control the tangents at the endpoints in a geometric way.

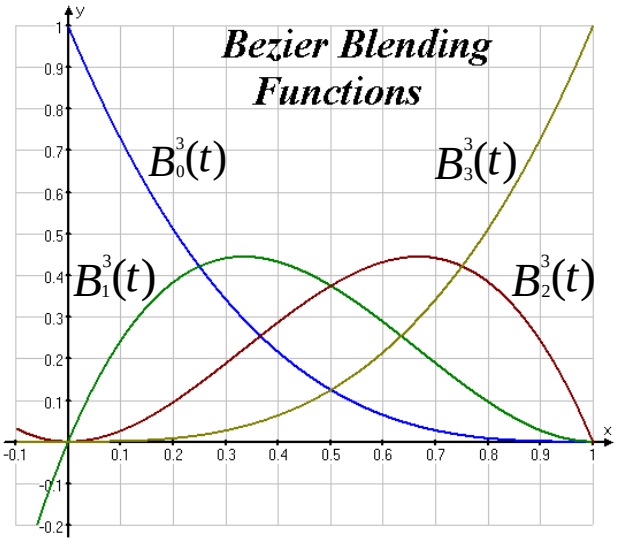
Some cubic Bezier curves



Bernstein Basis Polynomials for *d*=3

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

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



Bézier Matrix for *d=3*

From: $Dp_0 = 3(p_1 - p_0)$

$Dp_3 = 3(p_3 - p_2)$

We can deduce:

$$Q(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \cdot \underbrace{\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}}_{\boldsymbol{M_B}} \cdot \underbrace{\begin{bmatrix} p_{0x} & p_{0y} & p_{0z} \\ p_{1x} & p_{1y} & p_{1z} \\ p_{2x} & p_{2y} & p_{2z} \\ p_{3x} & p_{3y} & p_{3z} \end{bmatrix}}_{\boldsymbol{G_B}}$$

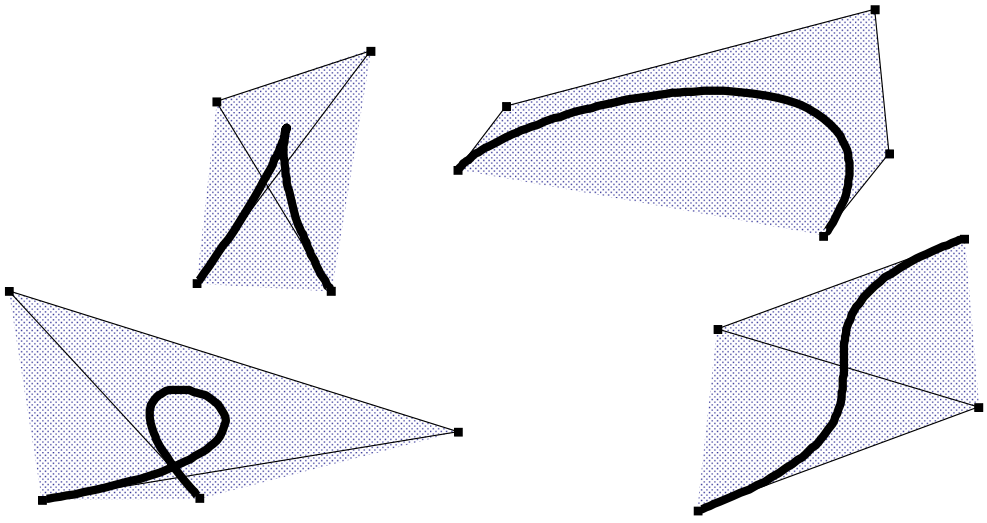
$$Q(t) = T \circ M_B \circ G_B$$

Bezier Cubic Curves Properties

- The first and last control points are interpolated,
- The tangent to the curve at the first control point is along the line joining the first and second control points,
- The tangent at the last control point is along the line joining the second last and last control points,
- The curve lies entirely within the convex hull of its control points:
  - Every point on curve is a linear combination of the control points,
  - The weights of the combination are all positive,
  - The sum of the weights is 1,
  - Therefore, the curve is a convex combination of the control points.
- By specifying multiple coincident points at a vertex, we pull the curve in closer and closer to that vertex.

Convex Hull Property

The properties of the Bernstein polynomials ensure that all Bezier curves lie in the convex hull of their control points.



De Casteljau Algorithm

- Describes the curve as a recursive series of linear interpolations.
- Is useful for providing an intuitive understanding of the geometry involved.
- Provides a means for evaluating Bezier curves.

De Casteljau Algorithm

Select $t \in [0,1]$ value. Then:

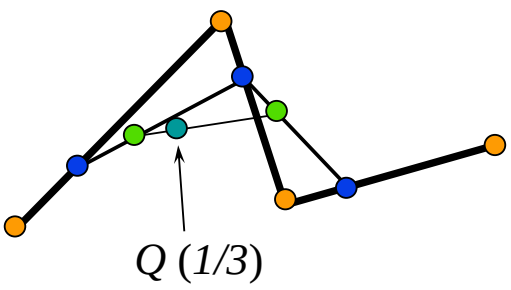
For $i := 0$ to d do $P_i^{[0]}(t) := P_i$;

For $j := 1$ to d do

  For $i :=$ j to d do

    $P_i^{[j]}(t) := (1-t)P_{i-1}^{[j-1]}(t) + tP_i^{[j-1]}(t)$;

$Q(t) := P_d^{[d]}(t)$;

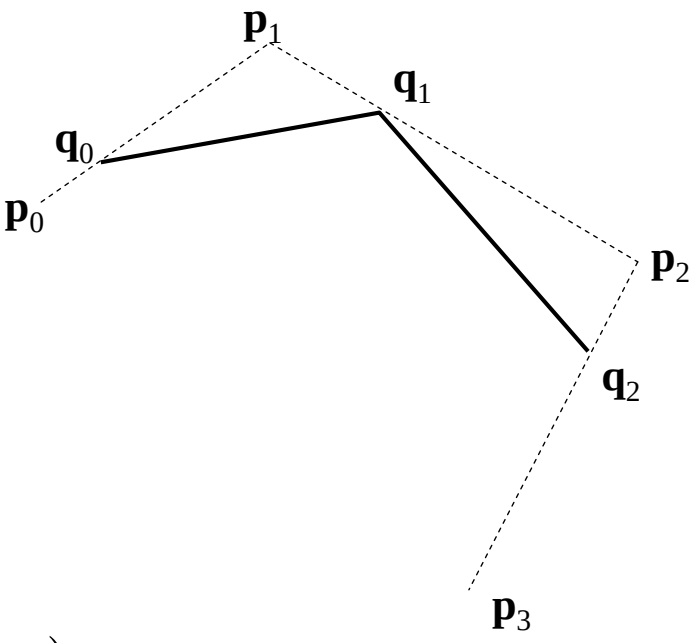


We take points 1/3 of the way

De Casteljau Algorithm for d=3

We start with our original set of points p_0, p_1, p_2 and p_3.

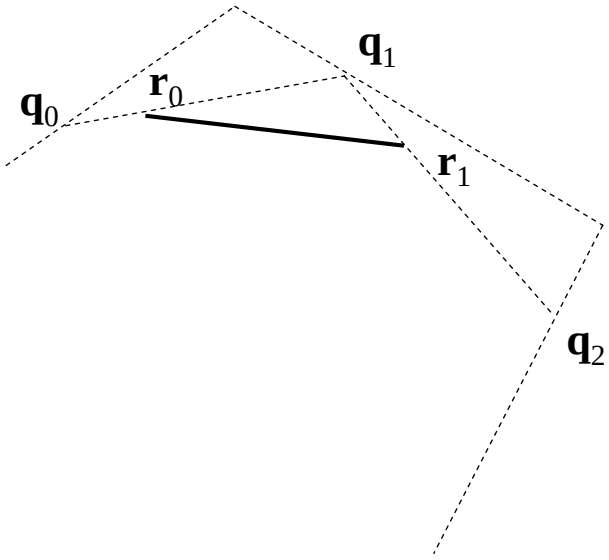
$$\mathbf{q}_0 = Lerp(t, \mathbf{p}_0, \mathbf{p}_1)$$
$$\mathbf{q}_1 = Lerp(t, \mathbf{p}_1, \mathbf{p}_2)$$
$$\mathbf{q}_2 = Lerp(t, \mathbf{p}_2, \mathbf{p}_3)$$



Where: $Lerp(t, p, q) = (1-t)p + tq$

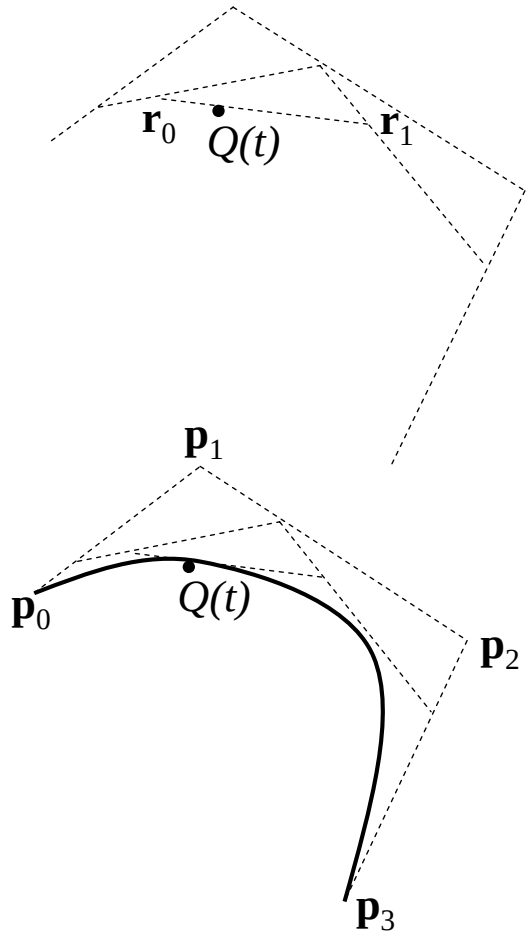
De Casteljau Algorithm

$$\mathbf{r}_0 = Lerp(t, \mathbf{q}_0, \mathbf{q}_1)$$
$$\mathbf{r}_1 = Lerp(t, \mathbf{q}_1, \mathbf{q}_2)$$



De Casteljau Algorithm

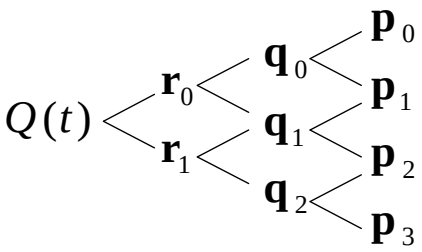
$$Q(t) = Lerp(t, \mathbf{r}_0, \mathbf{r}_1)$$



Bezier curve

Recursive Linear Interpolation

$$Q(t) = Lerp(t, \mathbf{r}_0, \mathbf{r}_1) \quad \begin{matrix} \mathbf{r}_0 = Lerp(t, \mathbf{q}_0, \mathbf{q}_1) \\ \mathbf{r}_1 = Lerp(t, \mathbf{q}_1, \mathbf{q}_2) \end{matrix} \quad \begin{matrix} \mathbf{q}_0 = Lerp(t, \mathbf{p}_0, \mathbf{p}_1) \\ \mathbf{q}_1 = Lerp(t, \mathbf{p}_1, \mathbf{p}_2) \\ \mathbf{q}_2 = Lerp(t, \mathbf{p}_2, \mathbf{p}_3) \end{matrix} \quad \begin{matrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{matrix}$$



Expanding the Lerps

$$\mathbf{q}_0 = Lerp(t, \mathbf{p}_0, \mathbf{p}_1) = (1-t)\mathbf{p}_0 + t\mathbf{p}_1$$
$$\mathbf{q}_1 = Lerp(t, \mathbf{p}_1, \mathbf{p}_2) = (1-t)\mathbf{p}_1 + t\mathbf{p}_2$$
$$\mathbf{q}_2 = Lerp(t, \mathbf{p}_2, \mathbf{p}_3) = (1-t)\mathbf{p}_2 + t\mathbf{p}_3$$

$$\mathbf{r}_0 = Lerp(t, \mathbf{q}_0, \mathbf{q}_1) = (1-t)((1-t)\mathbf{p}_0 + t\mathbf{p}_1) + t((1-t)\mathbf{p}_1 + t\mathbf{p}_2)$$
$$\mathbf{r}_1 = Lerp(t, \mathbf{q}_1, \mathbf{q}_2) = (1-t)((1-t)\mathbf{p}_1 + t\mathbf{p}_2) + t((1-t)\mathbf{p}_2 + t\mathbf{p}_3)$$

$$Q(t) = Lerp(t, \mathbf{r}_0, \mathbf{r}_1) = (1-t)((1-t)((1-t)\mathbf{p}_0 + t\mathbf{p}_1) + t((1-t)\mathbf{p}_1 + t\mathbf{p}_2))$$
$$+ t((1-t)((1-t)\mathbf{p}_1 + t\mathbf{p}_2) + t((1-t)\mathbf{p}_2 + t\mathbf{p}_3))$$

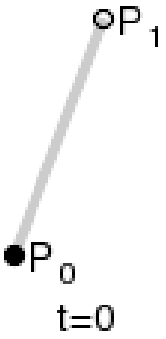
Bernstein Polynomial Form

$$Q(t) = (1-t)((1-t)((1-t)\mathbf{p}_0 + t\mathbf{p}_1) + t((1-t)\mathbf{p}_1 + t\mathbf{p}_2))$$
$$+ t((1-t)((1-t)\mathbf{p}_1 + t\mathbf{p}_2) + t((1-t)\mathbf{p}_2 + t\mathbf{p}_3))$$

$$Q(t) = (1-t)^3\mathbf{p}_0 + 3(1-t)^2 t\mathbf{p}_1 + 3(1-t)t^2\mathbf{p}_2 + t^3\mathbf{p}_3$$

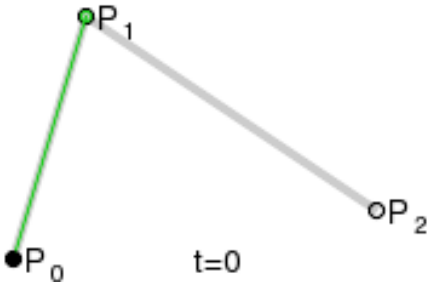
De Casteljau Algorithm

d=1



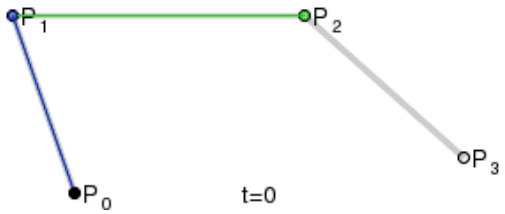
Animations from *Wikipedia*

d=2



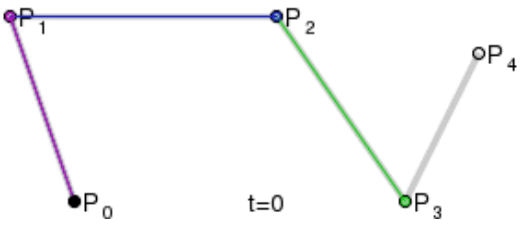
De Casteljau Algorithm

d=3



Animations from *Wikipedia*

d=4



Drawing Algorithms

Evaluate points and draw lines.

Possibilities:

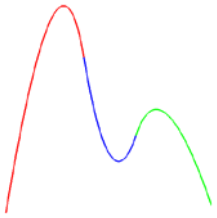
1. Evaluate recursively Bernstein Polynomials.
2. Compute the geometric matrix.
3. Use the recursive algorithm of De Casteljau.

Bezier Curves Drawbacks

- Are hard to control and hard to work with.
- The intermediate points don't have obvious effect on shape.
- For large sets of points – curve deviates far from the points.
- Degree corresponds to number of control points.
- Changing any control point can change the whole curve:
  - We want *local support:* each control point only influences nearby portion of curve.

Piecewise Curves

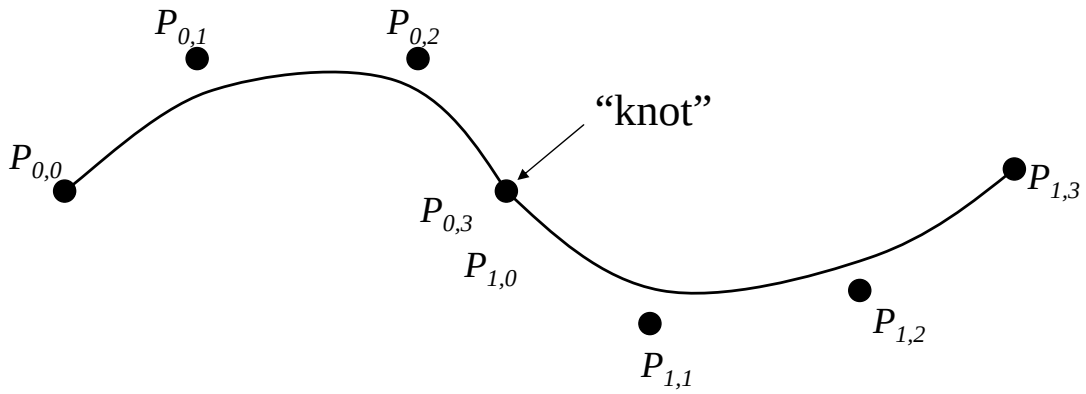
- A single cubic Hermite o Bezier curve can only capture a small class of curves:
  - At most 2 inflection points.
- One solution is to raise the degree:
  - Allows more control, at the expense of more control points and higher degree polynomials.
  - Control is not *local*, one control point influences entire curve.
- Alternate, most common solution is to join pieces of cubic curve together into *piecewise cubic curves:*
  - Total curve can be broken into pieces, each of which is cubic.
  - *Local control*: each control point only influences a limited part of the curve.
  - Interaction and design is much easier.



Piecewise Bezier Curves

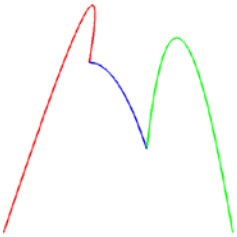
If complicated curves are to be generated, they can be formed by piecing together several Bézier sections.

The curve segments join at *knots*.



Achieving Continuity

- For Hermite curves:
  - the user specifies the derivatives, so C^1 is achieved simply by sharing points and derivatives across the *knot*.
- For Bezier curves:
  - They interpolate their endpoints, so C^0 is achieved by sharing control points.
  - The parametric derivative is a constant multiple of the vector joining the first/last 2 control points.
  - So C^1 is achieved by setting $P_{0,3}=P_{1,0}=P$, and making $P_{0,2}$ and P and $P_{1,1}$ collinear, with $P-P_{0,2}=P_{1,1}-P$.
  - C^2 comes from further constraints on $P_{0,1}$ and $P_{1,2}$



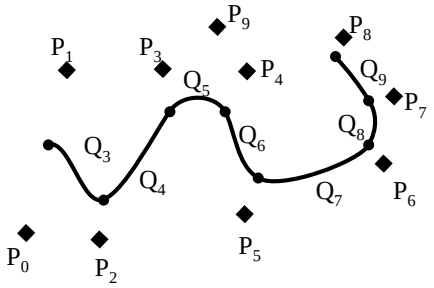
C_0 continuity

Why more curves?

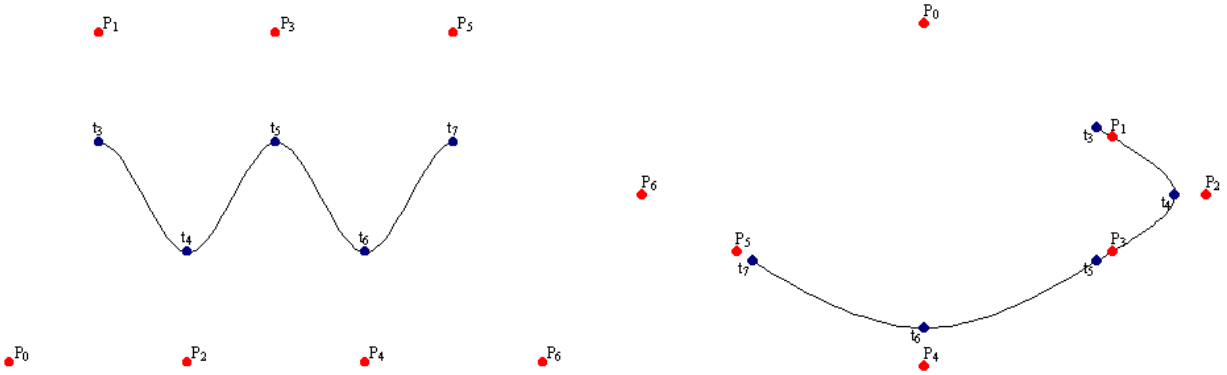
- Bezier and Hermite curves have global influence:
  - One could create a Bezier curve that required 15 points to define the curve…
    - Moving any one control point would affect the entire curve.
  - Piecewise Bezier or Hermite don't suffer from this, but they don't enforce derivative continuity at join points.
- B-Splines consist of curve segments whose polynomial coefficients depend on just a few control points:
  - Local control

Cubic B-Spline: Geometric Definition

- Approximates *n+1* control points: $P_0, P_1, \ldots, P_n$, $n >= 3$.
- Consists of *n –2* cubic polynomial curve segments: $Q_3, Q_4, \ldots, Q_n$.
- Defined using a non-decreasing sequence of *n-1* parameters: $t_3, \ldots, t_{n+1}$ (knots).
- It is uniform if the elements of the knot sequence are uniformly spaced: $t_{i+1}-t_i=c$, $i=3,\ldots,n$ (without loss of generality, we can assume $t_3=0$ *and* $t_{i+1}-t_i=1$), otherwise it is non-uniform.
- Curve segment Q_i is defined by the control points P_{i-3}, P_{i-2}, P_{i-1} and P_i over the knot interval $[t_i,t_{i+1})$ (there exist local control).



Uniform Cubic B-Splines



Uniform Cubic B-Splines

To determine the equation of the curve segment $Q_3(t)$, t in $[t_3,t_4)$:

- Each control point affects 4 curve segments.
- Formulate 16 equations to solve the 16 unknowns, that correspond to 4 polynomial of degree 3.
- The 16 equations enforce the C_0, C_1, and C_2 continuity between adjoining curve segments.

We obtain:

$$Q_3(t) = \sum_{k=0}^{3} B_{k,4}(t) P_k =$$

$$= \frac{1}{6}\left(1 - 3t + 3t^2 - t^3\right)P_0 + \frac{1}{6}\left(4 - 6t^2 + 3t^3\right)P_1 + \frac{1}{6}\left(1 + 3t + 3t^2 - 3t^3\right)P_2 + \frac{1}{6}\left(t^3\right)P_3$$

for t in $[t_3, t_4)$.

Uniform Cubic B-Splines

From:

$$Q_3(t) = \sum_{k=0}^{3} B_{k,4}(t) P_k =$$

$$= \frac{1}{6}\left(1 - 3t + 3t^2 - t^3\right)P_0 + \frac{1}{6_1}\left(4 - 6t^2 + 3t^3\right)P_1 + \frac{1}{6}\left(1 + 3t + 3t^2 - 3t^3\right)P_2 + \frac{1}{6}\left(t^3\right)P_3$$

We obtain the matricial expression:

$$Q_3(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}$$

Uniform Cubic B-Splines

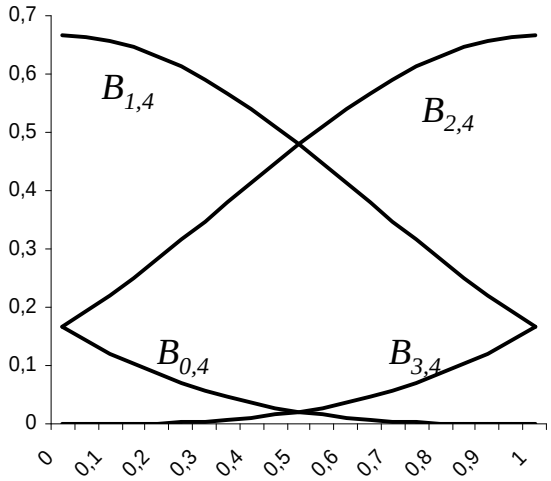
Basis or blending cubic functions for curve segment $Q_3(t)$:

$$B_{0,4}(t) = \frac{1}{6}\left(1 - 3t + 3t^2 - t^3\right)$$

$$B_{1,4}(t) = \frac{1}{6_1}\left(4 - 6t^2 + 3t^3\right)$$

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

$$B_{3,4}(t) = \frac{1}{6}\left(t^3\right)$$



Basis functions for $Q_3(t)$, t in $[t_3, t_4]=[0,1]$.

Uniform Cubic B-Splines

It is not difficult to see that for the curve segment $Q_i(t)$, t in $[t_i, t_{i+1})$, $i=3,\ldots,n$, we have:

$$Q_i(t) = \sum_{k=i-3}^{i} P_k B_{k,4}(t)$$

where:

$$B_{i,4}(t)=B_{0,4}(t-i).$$

Uniform Cubic B-Splines

- The blending functions sum to one, and are positive everywhere.
- The curve lies inside the convex hull of the control points.
- The curve does not interpolate its endpoints.
- Uniform B-splines are C^2.
- Moving a control point has a *local effect*.

Uniform B-Splines: Algebraic Definition

A B-spline curve $Q(t)$, is defined by:

$$Q(t) = \sum_{i=0}^{n} B_{i,d}(t) P_i$$

where:

- The P_i, $i = 0,1,...,n$ are the $n+1$ control points.
- d is the *order* of the polynomial segments of the B-Spline curve. That means that the curve is made up of piecewise polynomial segments of degree $d-1$.
- The order d is independent of the number of control points ($n+1$).
- $B_{i,d}$ are the uniform B-Spline *basis* or *blending functions* of degree $d-1$.

B-Spline Basis Functions

Given a non-decreasing *knot sequence* of *n+d+1* parameters $t_0, \ldots, t_{n+d}$, *we* define:

$$B_{i,1}(t) = \begin{cases} 1 & t_i \le t < t_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

$$B_{i,d}(t) = \left(\frac{t - t_i}{t_{i+d-1} - t_k} \right) B_{i,d-1}(t) + \left(\frac{t_{i+d} - t}{t_{i+d} - t_{i+1}} \right) B_{i+1,d-1}(t)$$

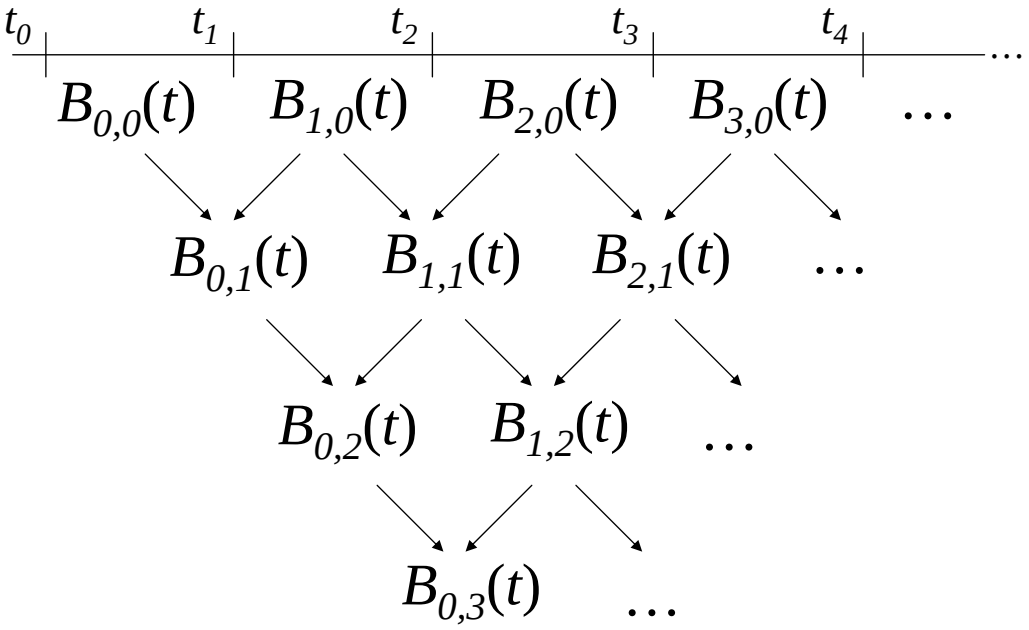
for $d > 1$ and t in $[t_{d-1,n+1})$.

If the *denominator* terms on the right hand side of the last equation are *zero* or the subscripts are *out of the range* of the summation limits, then the associated fraction is not evaluated and the term becomes zero (avoiding 0/0 expressions).

We will concentrate in uniform B-splines and, without loss of generality, we will assume $t_0 = -3$ *and* $t_{i+1} - t_i = 1$.

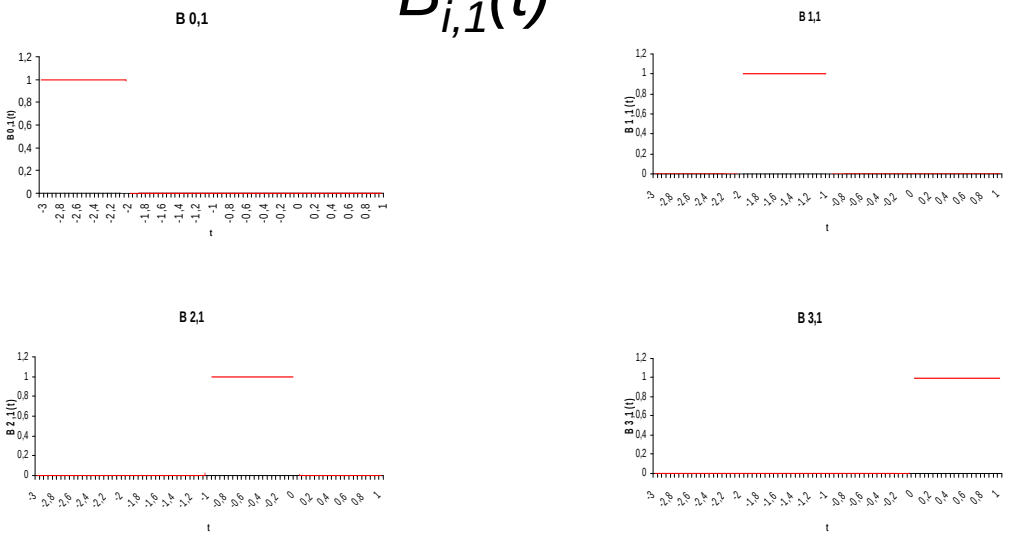
Observe that for a cubic (d=4) B-Spline now we have n+5 parameters: $t_0, \ldots, t_{n+4}$.

B-Splines Basis Functions Computation



Blending functions for *d=1*

$$B_{i,1}(t)$$



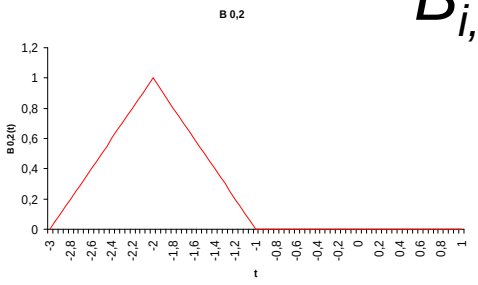
$B_{i,1}(t)$ has support (is non zero) on interval [i-3,i-2).

$B_{i,1}(t)$ is a constant function.

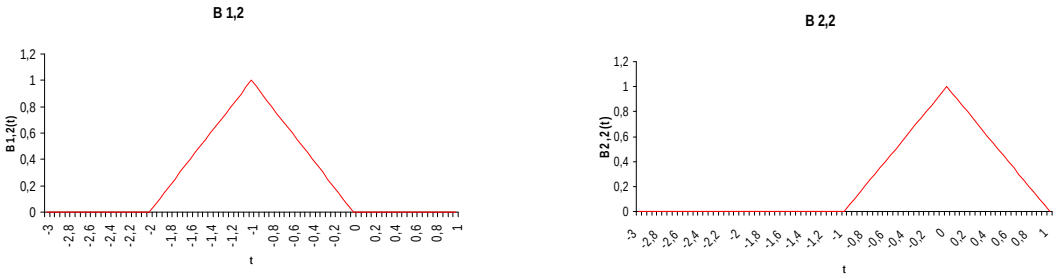
$B_{i+1,1}(t)$ is just $B_{i,1}(t)$ shifted one unit to the right, because the uniform election of parameter knots. We can write: $B_{i,1}(t)=B_{0,1}(t-i)$.

Blending functions for *d=2*

$$B_{i,2}(t)$$



$$B_{0,2}(t)=\begin{cases} t+3 & -3\le t<-2 \\ -1-t & -2\le t<-1 \end{cases}$$



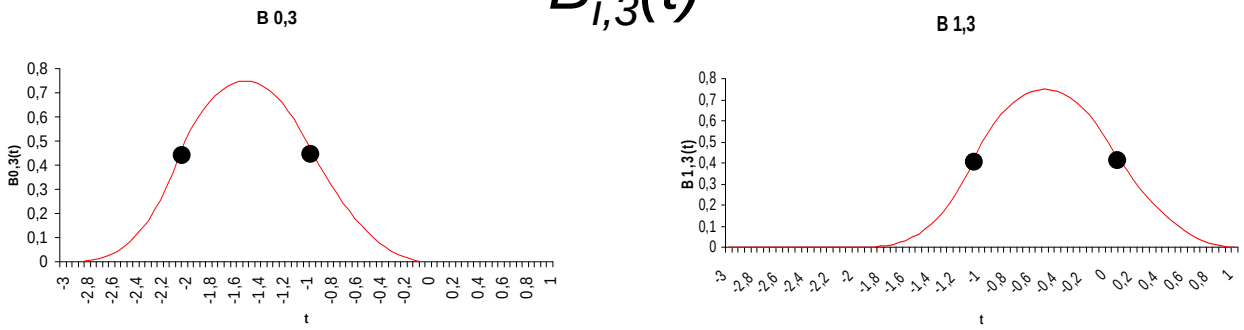
$B_{i,2}(t)$ has support on interval [i-3,i-1).

$B_{i,2}(t)$ is piecewise linear, made up of two linear segments joined continuously.

We have: $B_{i,2}(t)=B_{0,2}(t-i)$.

Blending functions for *d=3*

$$B_{i,3}(t)$$



$$B_{0,3}(t)=\frac{1}{2}\begin{cases}(t+3)^2 & -3\le t<-2\\ -2t^2-6t-3 & -2\le t<-1\\ t^2 & -1\le t<0\end{cases}$$

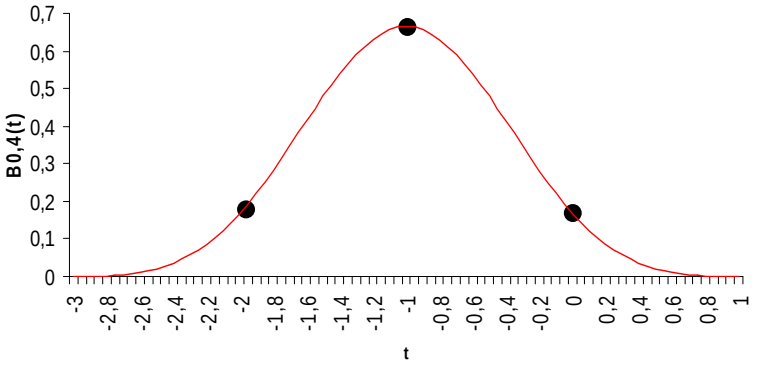
$B_{i,3}(t)$ has support on interval $[i-3,i)$.

$B_{i,3}(t)$ is a piecewise quadratic curve made up of three parabolic segments that are joined continuously.

We have: $B_{i,3}(t)=B_{0,3}(t-i)$.

Blending functions for *d=4* (cubic)

$$B_{0,4}(t)$$

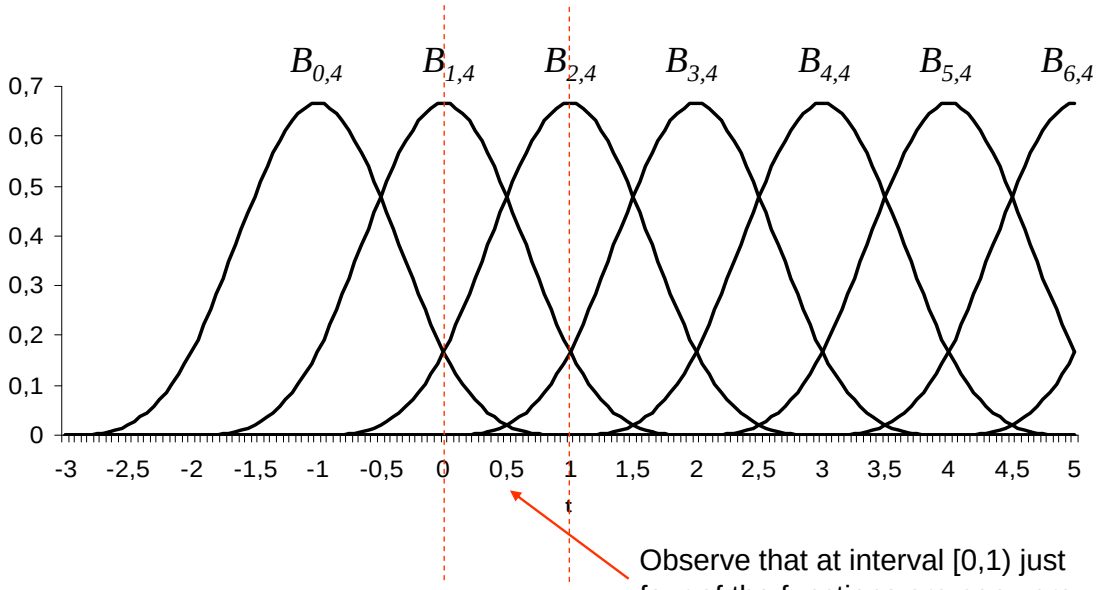


$$B_{0,4}(t)=\frac{1}{6}\begin{cases}(t+3)^3 & -3\le t<-2\\ -3t^3-15t^2-21t-5 & -2\le t<-1\\ 3t^3+3t^2-3t+1 & -1\le t<0\\ (1-t)^3 & 0\le t<1\end{cases}$$

$B_{i,4}(t)$ has support on interval $[i-3,i+1)$.

$B_{i,4}(t)$ is a piecewise cubic curve made up of four cubic segments that are joined continuously.

Uniform Cubic B-spline Blending Functions



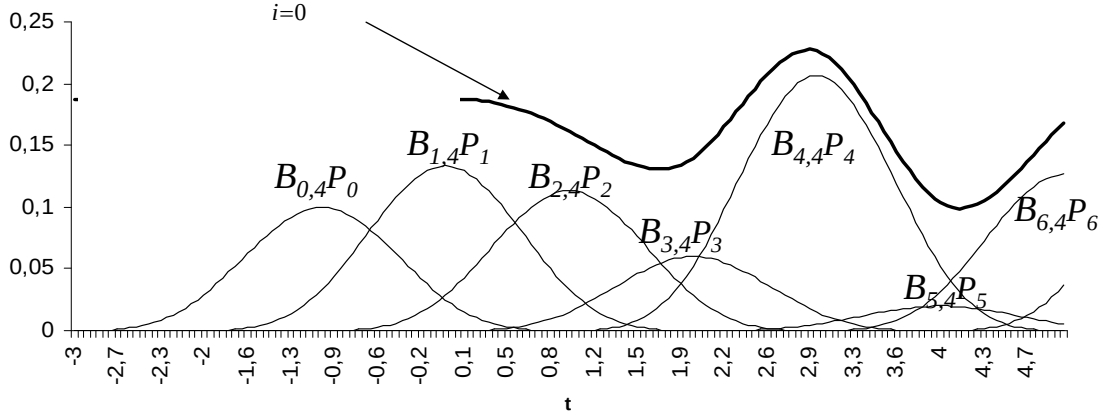
We have: $B_{i,4}(t)=B_{0,4}(t-i)$.

Observe that at interval [0,1) just four of the functions are non-zero: $B_{0,4}(t)$, $B_{1,4}(t)$, $B_{0,4}(t)$ and $B_{1,4}(t)$.

They are the four functions that determine the curve segment Q_3.

Example of Uniform Cubic B-Spline ($n=6$)

$$Q(t)=\sum_{i=0}^{n} B_{i,4}(t)\,P_i$$



The curve can't start until there are 4 basis functions active.

Uniform Cubic B-spline at Arbitrary t

- The interval from an integer parameter value *i* to *i+1* is essentially the same as the interval from 0 to 1:
  - The parameter value is offset by *i.*
  - A different set of control points is needed.
- To evaluate a uniform cubic B-spline at an arbitrary parameter value t:
  - Find the greatest integer less than or equal to *t: i* = floor(*t*)
  - Evaluate:

$$Q(t) = \sum_{k=0}^{3} B_{k,4}(t-i)P_{i+k}$$

- Valid parameter range: $0 \le t < n-3$, where n is the number of control points

Cubic B-Splines Shape Modification Using Control Points

Closed curve

- To obtain a closed curve, repeat the control points P_0, P_1, P_2 at the end of the sequence: P_0, P_1, P_2, ... , P_0, P_1, P_2.

Point interpolation

- On can force the curve pass through a control point by giving that point a multiplicity 3: $P_i = P_{i+1} = P_{i+2}$ (however, tangent discontinuity may result).
- In particular, to force endpoints interpolation, let: $P_0 = P_1 = P_2$ and $P_{n-2} = P_{n-1} = P_n$.

NURBS: Non-Uniform Rational B-Splines

NURBS are defined as:

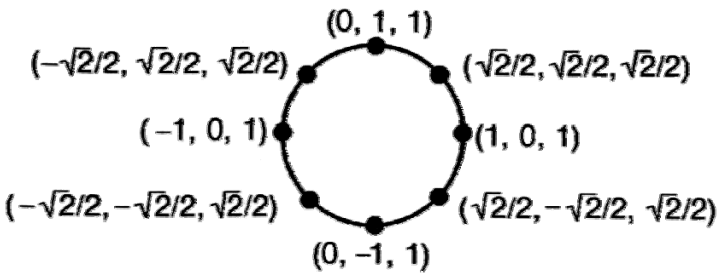
$$Q(t) = \frac{\sum_{i=0}^{n} w_i B_{i,d}(t) P_i}{\sum_{j=0}^{n} w_j B_{j,d}(t)}$$

where every point control point P_i has associated a weight ω_i.

NURBS

NURBS can be made to pass arbitrarily far or near to a control point by changing the corresponding weight.

All conic sections (a circle for example) can be modelled exactly:



Constructing a 2D-circle with NURBS

NURBS inherit all advantages of B-Splines (locality, ...) while extending the liberty of modelling.

NURBS

- If all w_i are set to the value 1, we obtain standard B-Splines.
- A Bezier curve is a special case of a B-Spline, so NURBS can also represent Bezier curves.
- Today NURBS are standard for modelling.

Curves

Toni Sellarès
Universitat de Girona