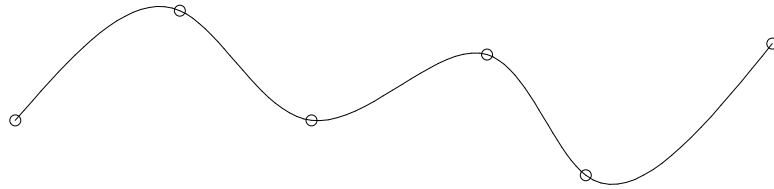
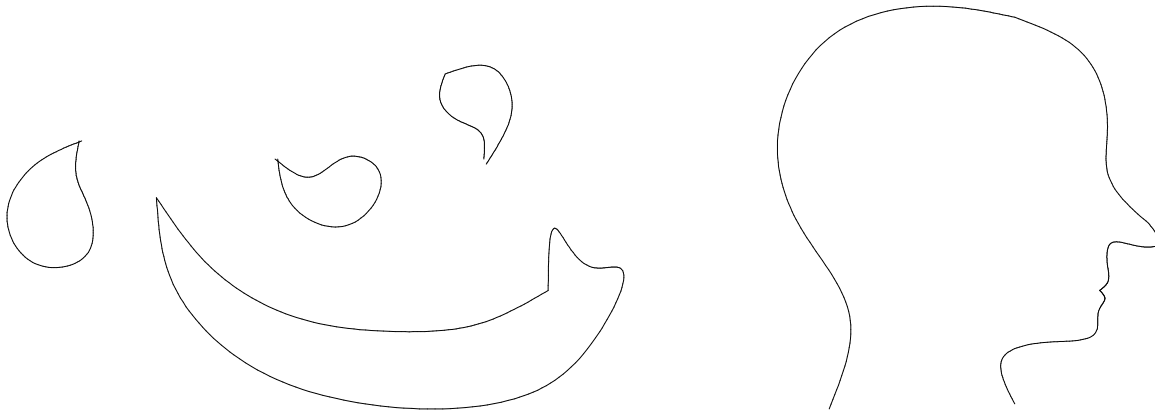


Curve Representations and Rendering

- Consider piecewise curves only. Why?



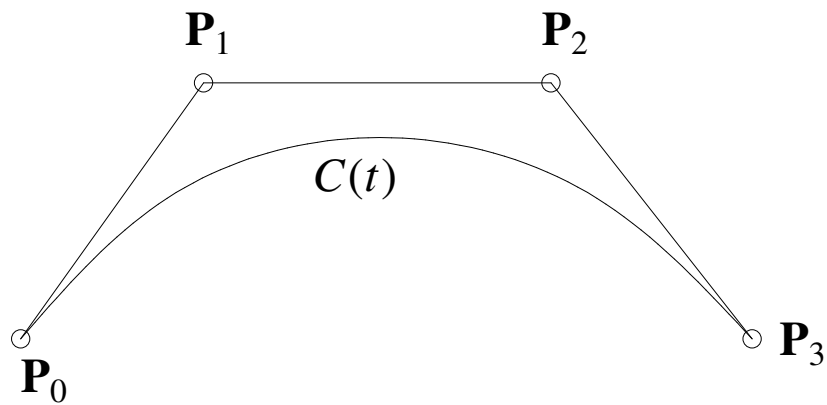
- Used for **font**, **carton character**, **car body**, ..., design/representation



Bezier curve segments of degree 3:

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

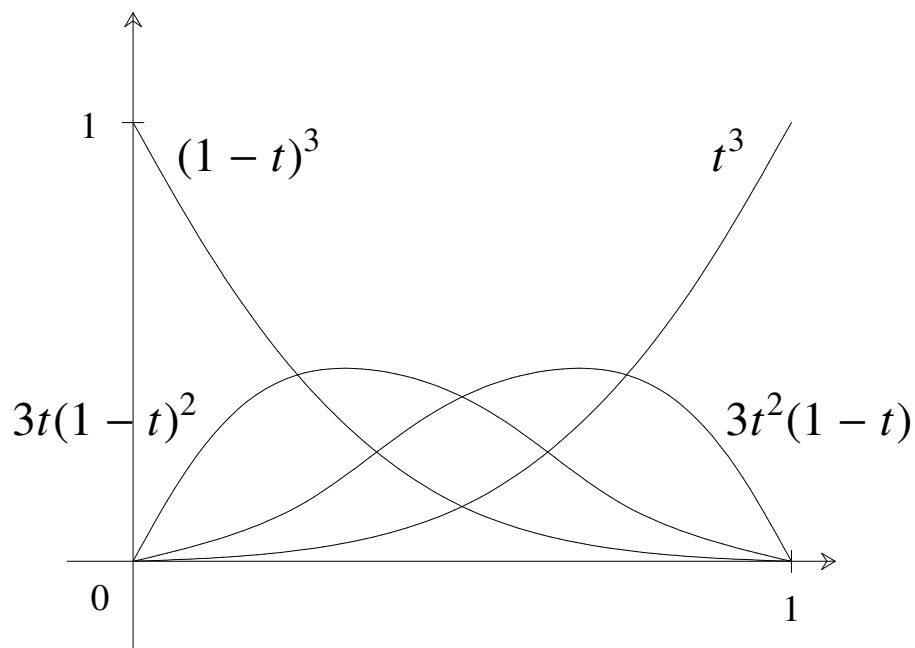
$$0 \leq t \leq 1, \quad \mathbf{P}_i = (x_i, y_i)$$



Matrix form:

$$\begin{aligned} \mathbf{C}(t) &= [1, t, t^2, t^3] \begin{bmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{P}_0 \\ \mathbf{P}_1 \\ \mathbf{P}_2 \\ \mathbf{P}_3 \end{bmatrix} \\ &= \mathbf{T} \cdot \mathbf{M}_B \cdot \mathbf{G} \end{aligned}$$

- $\mathbf{P}_i = (x_i, y_i)$ are called **control points**
- The polygon $\mathbf{P}_0\mathbf{P}_1\mathbf{P}_2\mathbf{P}_3$ is called the **control polygon**
- The weights $(1-t)^3$, $3t(1-t)^2$, $3t^2(1-t)$, and t^3 are called **blending functions**



Notes:

- Blending functions are always non-negative
- Blending functions always sum to 1
- $C(0) = \mathbf{P}_0$; $C(1) = \mathbf{P}_3$

(A Bezier curve starts at \mathbf{P}_0 and ends at \mathbf{P}_3)

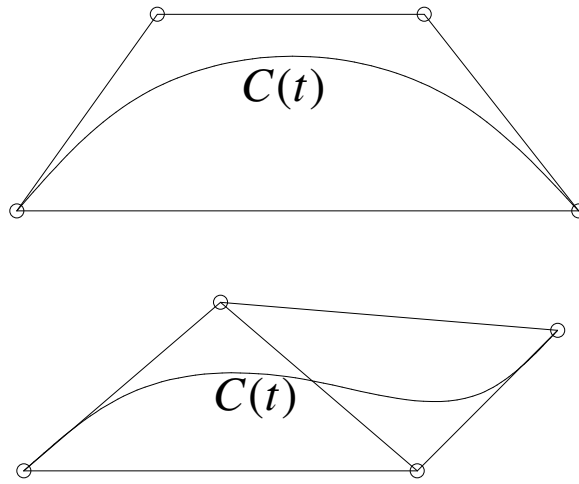
- $C'(0) = 3(\mathbf{P}_1 - \mathbf{P}_0)$; $C'(1) = 3(\mathbf{P}_3 - \mathbf{P}_2)$

(A Bezier curve is tangent to the control polygon at the endpoints)

- $C''(0) = 6(\mathbf{P}_2 - 2\mathbf{P}_1 + \mathbf{P}_0)$; $C''(1) = 6(\mathbf{P}_3 - 2\mathbf{P}_2 + \mathbf{P}_1)$

- Bezier curve segments satisfy **convex hull property**

i.e., a Bezier curve segment is always contained in the **convex hull** of its control points



- Bezier curves have intuitive appeal for interactive users

A recurrence formula:

For any $t \in [0, 1]$, a cubic Bezier curve defined as follows

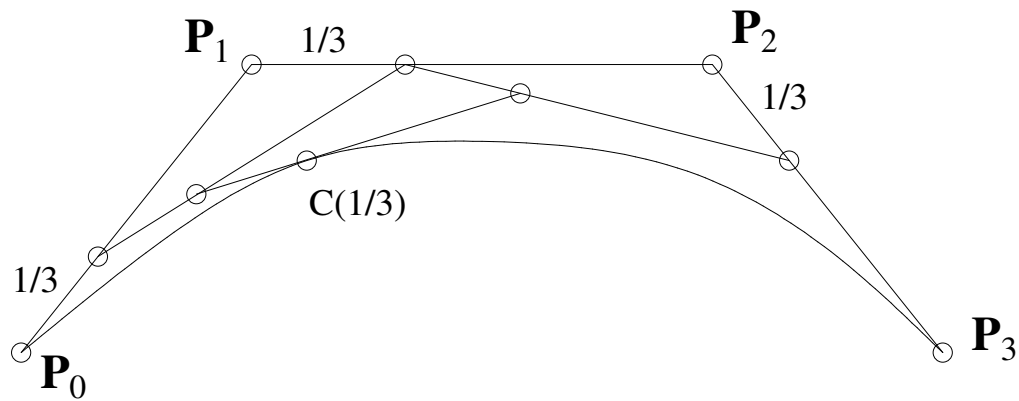
$$\mathbf{C}(t) = (1 - t)^3 \mathbf{P}_0 + 3t(1 - t)^2 \mathbf{P}_1 + 3t^2(1 - t) \mathbf{P}_2 + t^3 \mathbf{P}_3$$

satisfies the following recurrence formula:

$$\begin{aligned} C(t) &= (1 - t) [(1 - t)^2 \mathbf{P}_0 + 2t(1 - t) \mathbf{P}_1 + t^2 \mathbf{P}_2] \\ &\quad + t [(1 - t)^2 \mathbf{P}_1 + 2t(1 - t) \mathbf{P}_2 + t^2 \mathbf{P}_3] \\ &= (1 - t) [(1 - t) [(1 - t) \mathbf{P}_0 + t \mathbf{P}_1] \\ &\quad + t [(1 - t) \mathbf{P}_1 + t \mathbf{P}_2]] \\ &\quad + t [(1 - t) [(1 - t) \mathbf{P}_1 + t \mathbf{P}_2] \\ &\quad + t [(1 - t) \mathbf{P}_2 + t \mathbf{P}_3]] \end{aligned}$$

Applications of the recurrence formula:

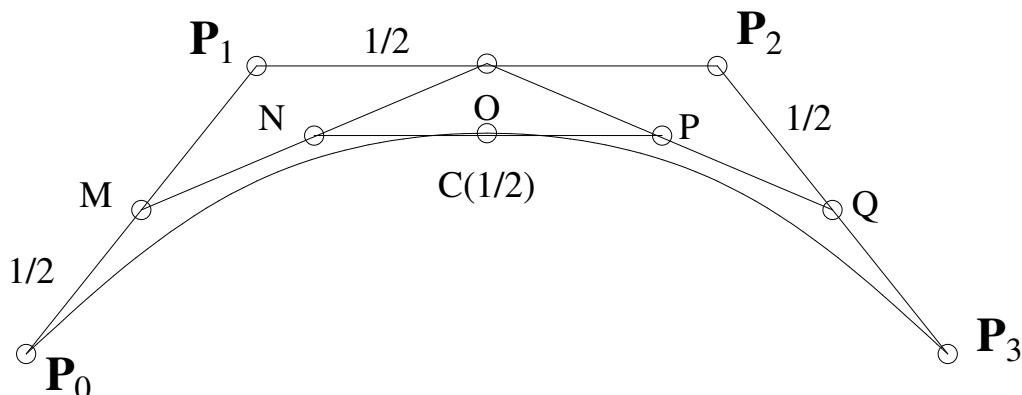
- Curve computation



e.g., for $t = 1/3$, we can find $f(1/3)$ using the above approach:

$$\begin{aligned} C\left(\frac{1}{3}\right) &= \frac{2}{3} \left[\frac{2}{3} \left(\frac{2}{3} P_0 + \frac{1}{3} P_1 \right) + \frac{1}{3} \left(\frac{2}{3} P_1 + \frac{1}{3} P_2 \right) \right] \\ &\quad + \frac{1}{3} \left[\frac{2}{3} \left(\frac{2}{3} P_1 + \frac{1}{3} P_2 \right) + \frac{1}{3} \left(\frac{2}{3} P_2 + \frac{1}{3} P_3 \right) \right] \end{aligned}$$

- Curve subdivision



Recursively subdivide the control polygons at the midpoints, We will not only get $C(1/2)$, but also control points P_0, M, N, O for the left half of $C(t)$ and control points O, P, Q, P_3 for the right half of $C(t)$, i.e., if $C_1(t)$ and $C_2(t)$ are defined as follows:

$$C_1(t) \equiv (1-t)^3 P_0 + 3t(1-t)^2 M + 3t^2(1-t) N + t^3 O$$

$$C_2(t) \equiv (1-t)^3 O + 3t(1-t)^2 P + 3t^2(1-t) Q + t^3 P_3$$

where $0 \leq t \leq 1$

then \mathbf{C}_1 equals left half of $\mathbf{C}(t)$ and \mathbf{C}_2 equals right half of $\mathbf{C}(t)$, i.e.,

$$\left\{ \mathbf{C}_1(t) \mid 0 \leq t \leq 1 \right\} = \left\{ \mathbf{C}(t) \mid 0 \leq t \leq 1/2 \right\}$$

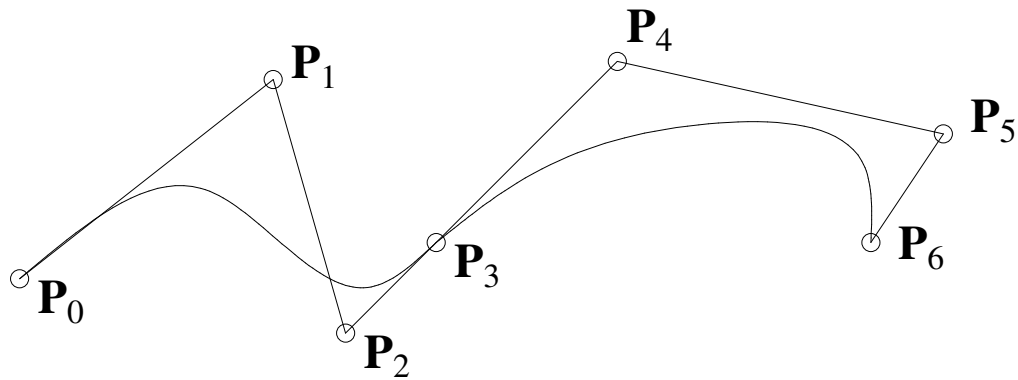
$$\left\{ \mathbf{C}_2(t) \mid 0 \leq t \leq 1 \right\} = \left\{ \mathbf{C}(t) \mid 1/2 \leq t \leq 1 \right\}$$

By recursively subdividing a Bezier curve, we can simply use the control points of the subsegments to approximate the curve.

Furthermore, this also provides us with an approximation method to find the intersection of two cubic Bezier curves.

Composite Bezier Curves

- Bezier curve segments can be joined together to form complicated shapes



P_0 , P_1 , P_2 , and P_3 are control points of the 1st segment

P_3 , P_4 , P_5 , and P_6 are control points of the 2nd segment

P_2 , P_3 , and P_4 are collinear (to guarantee smooth joint)

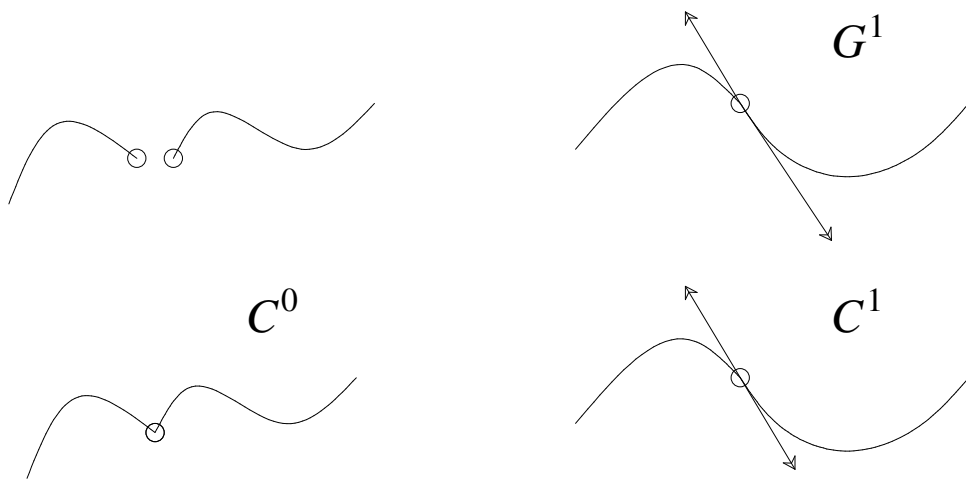
- Smoothness (continuity) at Join Points:

C^0 : the endpoints coincide

G^1 : tangents have the same slope

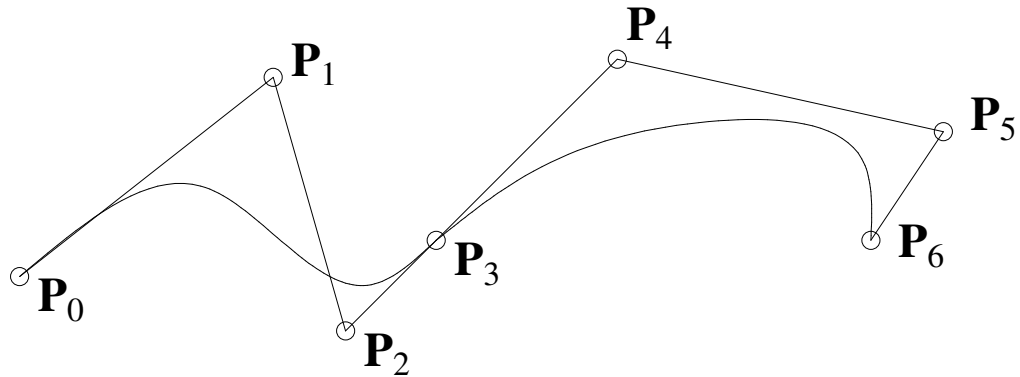
C^1 : the first derivatives on both segments match at join point

C^2 : nth derivatives on both segments match at join point

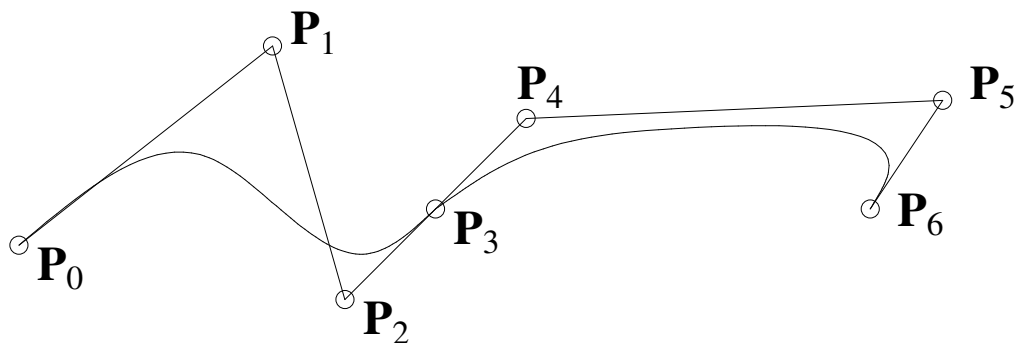


$\mathbf{P}_0, \mathbf{P}_1, \mathbf{P}_2$, and \mathbf{P}_3 : control points of the 1st segment

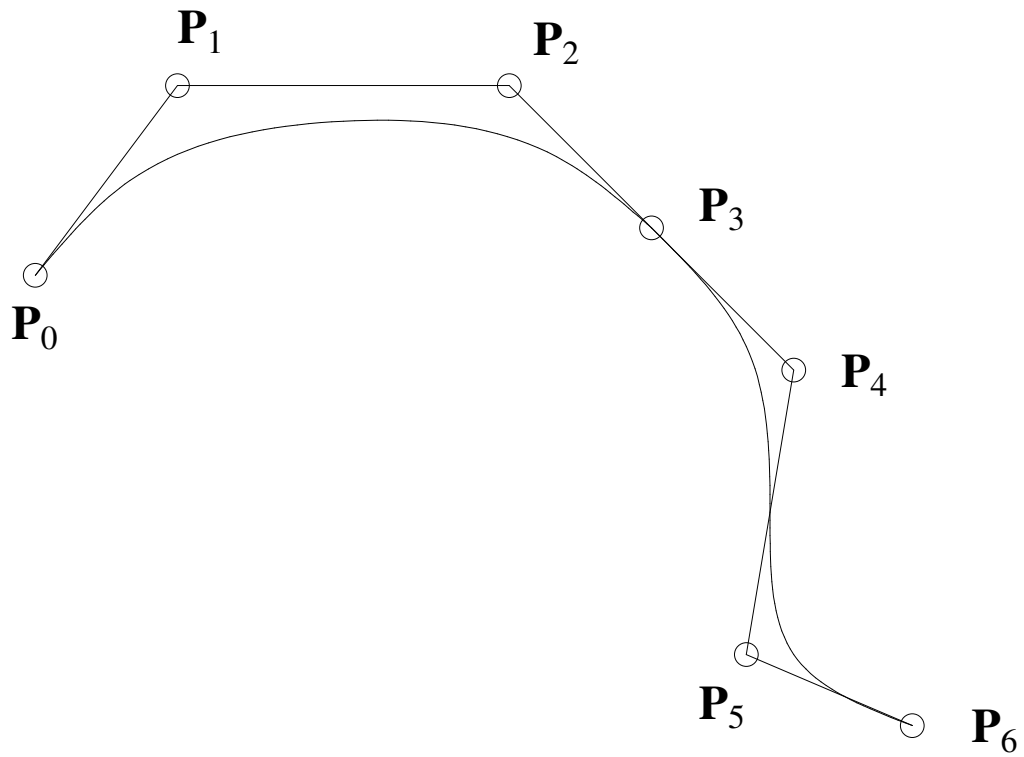
$\mathbf{P}_3, \mathbf{P}_4, \mathbf{P}_5$, and \mathbf{P}_6 : control points of the 2nd segment



- G^1 -continuity: $\mathbf{P}_2, \mathbf{P}_3$, and \mathbf{P}_4 are collinear
(See the above example)
- C^1 -continuity: $\mathbf{P}_2, \mathbf{P}_3$, and \mathbf{P}_4 are collinear and \mathbf{P}_3 is the midpoint of $\mathbf{P}_2\mathbf{P}_4$

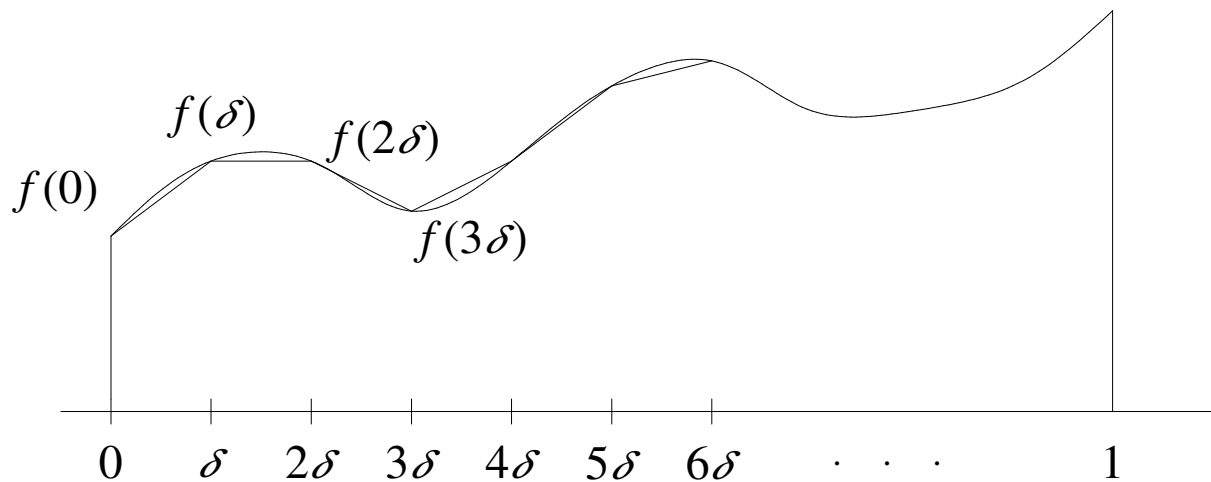


- C^2 -continuity:
 - \mathbf{P}_2 , \mathbf{P}_3 , and \mathbf{P}_4 are collinear
 - \mathbf{P}_3 is the midpoint of $\mathbf{P}_2\mathbf{P}_4$
 - $\mathbf{P}_5 = \mathbf{P}_1 + 4(\mathbf{P}_3 - \mathbf{P}_2)$



Forward Differencing

- Another technique to render a (cubic) curve
- Each component of a cubic Bezier curve is a polynomial of degree 3. Hence, the question is: how to efficiently compute points of a cubic polynomial $f(t)$ at $0, \delta, 2\delta, 3\delta, \dots, 1$?



- **Forward differencing:** only three additions are needed to compute a new point

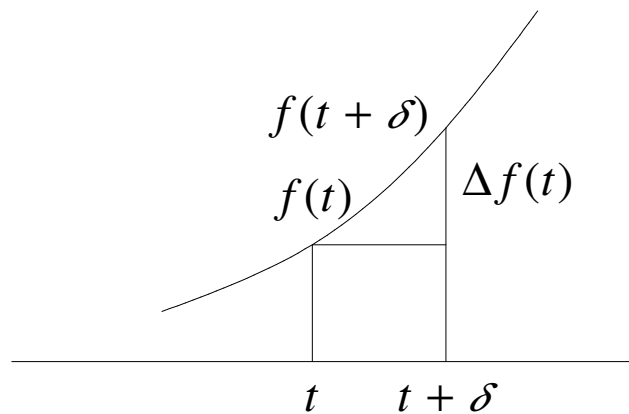
Let

$$f(t) = a + bt + ct^2 + dt^3, \quad t \in [0, 1] \quad (\%)$$

and $\delta > 0$ given. Define

$$\Delta f(t) = f(t + \delta) - f(t), \quad t \in [0, 1] \quad (*)$$

Δ is called a **forward differencing operator**



If we know $f(t)$ and $\Delta f(t + \delta)$ then from (*) we have

$$f(t + \delta) = f(t) + \Delta f(t) \quad (**)$$

Equation (**) shows that if we know $f(0)$ and $\Delta f(0)$ then we can compute $f(\delta)$ as follows:

$$f(\delta) = f(0) + \Delta f(0)$$

If we know $f(\delta)$ and $\Delta f(\delta)$ then we can compute $f(2\delta)$ as follows:

$$f(2\delta) = f(\delta) + \Delta f(\delta)$$

If we know $f(2\delta)$ and $\Delta f(2\delta)$ then we can compute $f(3\delta)$ as follows:

$$f(3\delta) = f(2\delta) + \Delta f(2\delta)$$

$$\begin{array}{ccccccc}
 f(0) & \longrightarrow & f(\delta) & \longrightarrow & f(2\delta) & \longrightarrow & f(3\delta) & \longrightarrow & f(4\delta) & \dots \\
 \hline
 \Delta f(0) & & \Delta f(\delta) & & \Delta f(2\delta) & & \Delta f(3\delta) & & \Delta f(4\delta)
 \end{array}$$

Note that

$$\Delta f(t) = (b\delta + c\delta^2 + d\delta^3) + (2c\delta + 3d\delta^2)t + (3d\delta)t^2, (***)$$

a polynomial of degree 2. So $\Delta f(\delta)$, $\Delta f(2\delta)$, ... are computable.

But, instead of using (***) directly, is there a way to compute $\Delta f(\delta)$, $\Delta f(2\delta)$, $\Delta f(3\delta)$, ... more efficiently? **YES**

If we define

$$\Delta^2 f(t) \equiv \Delta(\Delta f(t)) = \Delta f(t + \delta) - \Delta f(t) \quad (\#)$$

then $\Delta f(t + \delta)$ can be computed as follows if $\Delta f(t)$ and $\Delta^2 f(t)$ are known to us

$$\Delta f(t + \delta) = \Delta f(t) + \Delta^2 f(t) \quad (\#\#)$$

For instance, if $\Delta f(0)$ and $\Delta^2 f(0)$ are known to us then we can compute $\Delta f(\delta)$ as follows:

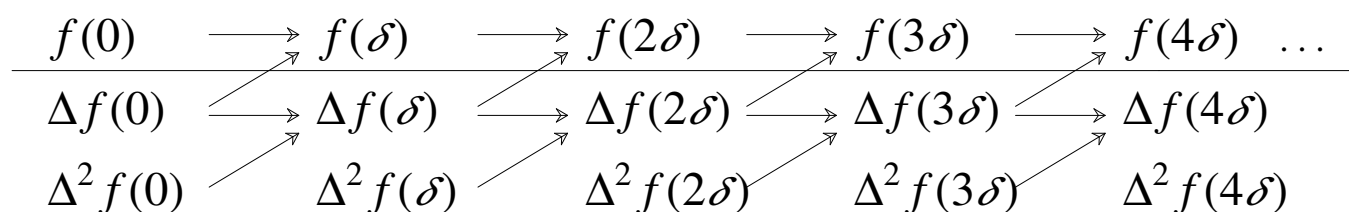
$$\Delta f(\delta) = \Delta f(0) + \Delta^2 f(0)$$

If $\Delta f(\delta)$ and $\Delta^2 f(\delta)$ are known to us then we can compute $\Delta f(2\delta)$ as follows:

$$\Delta f(2\delta) = \Delta f(\delta) + \Delta^2 f(\delta)$$

If $\Delta f(2\delta)$ and $\Delta^2 f(2\delta)$ are known to us then we can compute $\Delta f(3\delta)$ as follows:

$$\Delta f(3\delta) = \Delta f(2\delta) + \Delta^2 f(2\delta)$$



Note that

$$\Delta^2 f(t) = (2c\delta^2 + 6d\delta^3) + (6d\delta^2)t, \quad (\#\#\#)$$

a polynomial of degree 1. So $\Delta^2 f(0)$, $\Delta^2 f(\delta)$, $\Delta^2 f(2\delta)$, ... are computable.

But, again, is there a way to compute $\Delta^2 f(\delta)$, $\Delta^2 f(2\delta)$, $\Delta^2 f(3\delta)$, ... more efficiently, instead of using $(\#\#\#)$? **YES**

Define

$$\Delta^3 f(t) \equiv \Delta(\Delta^2 f(t)) = \Delta^2 f(t + \delta) - \Delta^2 f(t) \quad (\&)$$

From $(\#\#\#)$, we have

$$\Delta^3 f(t) \equiv 6d\delta^3, \quad (\&\&)$$

a constant.

Equation (&) shows that if $\Delta^2 f(t)$ and $\Delta^3 f(t)$ are known to us then $\Delta^2 f(t + \delta)$ can be computed as their sum.

$$\Delta^2 f(t + \delta) = \Delta^2 f(t) + \Delta^3 f(t) \quad (\&\&\&)$$

For instance, if $\Delta^2 f(0)$ and $\Delta^3 f(0)$ are known to us then we can compute $\Delta^2 f(\delta)$ as follows:

$$\Delta^2 f(\delta) = \Delta^2 f(0) + \Delta^3 f(0)$$

If $\Delta^2 f(\delta)$ and $\Delta^3 f(\delta)$ are known to us then we can compute $\Delta^2 f(2\delta)$ as follows:

$$\Delta^2 f(2\delta) = \Delta^2 f(\delta) + \Delta^3 f(\delta)$$

If $\Delta^2 f(2\delta)$ and $\Delta^3 f(2\delta)$ are known to us then we can compute $\Delta^2 f(3\delta)$ as follows:

$$\Delta^2 f(3\delta) = \Delta^2 f(2\delta) + \Delta^3 f(2\delta)$$

Note that, since $\Delta^3 f(t)$ is a constant (see (&&)), we have

$$\Delta^3 f(0) = \Delta^3 f(\delta) = \Delta^3 f(2\delta) = \Delta^3 f(3\delta) = \dots = 6d\delta^3.$$

This value, $6d\delta^3$, has to be computed only once, for $\Delta^3 f(0)$.

Hence, if $\delta = 1/n$, we can generate $n + 1$ points

$$f(0), f(\delta), f(2\delta), \dots, f((n-1)\delta), f(n\delta) = f(1)$$

on $f(t)$ as follows.

$$\begin{array}{ccccccccc}
 f(0) & \longrightarrow & f(\delta) & \longrightarrow & f(2\delta) & \longrightarrow & f(3\delta) & \longrightarrow & f(4\delta) & \dots \\
 \hline
 \Delta f(0) & \longrightarrow & \Delta f(\delta) & \longrightarrow & \Delta f(2\delta) & \longrightarrow & \Delta f(3\delta) & \longrightarrow & \Delta f(4\delta) \\
 \Delta^2 f(0) & \longrightarrow & \Delta^2 f(\delta) & \longrightarrow & \Delta^2 f(2\delta) & \longrightarrow & \Delta^2 f(3\delta) & \longrightarrow & \Delta^2 f(4\delta) \\
 \Delta^3 f(0) & \longrightarrow & \Delta^3 f(\delta) & \longrightarrow & \Delta^3 f(2\delta) & \longrightarrow & \Delta^3 f(3\delta) & \longrightarrow & \Delta^3 f(4\delta)
 \end{array}$$

One needs to compute the values of $f(0)$, $\Delta f(0)$, $\Delta^2 f(0)$, and $\Delta^3 f(0)$ using (%), (**), (###), and (&&), respectively, first.

$$f(0) = a$$

$$\Delta f(0) = b\delta + c\delta^2 + d\delta^3$$

$$\Delta^2 f(0) = 2c\delta^2 + 6d\delta^3$$

$$\Delta^3 f(0) = 6d\delta^3$$

These terms requires several multiplications. But every subsequent point then requires 3 additions to compute only.

For instance, once we have the values of $f(0)$, $\Delta f(0)$, $\Delta^2 f(0)$, and $\Delta^3 f(0)$ in the first column of the above table, we can then compute the values of the items in the second column by adding $\Delta f(0)$ to $f(0)$ to get $f(\delta)$, adding $\Delta^2 f(0)$ to $\Delta f(0)$ to get $\Delta f(\delta)$, adding $\Delta^3 f(0)$ to $\Delta^2 f(0)$ to get $\Delta^2 f(\delta)$, and setting $\Delta^3 f(\delta) = \Delta^3 f(0)$.

The values of the items in the third column are determined using a similar approach, i.e., adding $\Delta f(\delta)$ to $f(\delta)$ to get $f(2\delta)$, adding $\Delta^2 f(\delta)$ to $\Delta f(\delta)$ to get $\Delta f(2\delta)$, adding $\Delta^3 f(\delta)$ to $\Delta^2 f(\delta)$ to get $\Delta^2 f(2\delta)$, and setting $\Delta^3 f(2\delta) = \Delta^3 f(\delta)$.

Each of this iteration process requires three additions.

Note that only the items above the line in the above table are needed in the rendering process of the curve. The items below the line are used to find the items above the line.

Forward differencing is the most efficient curve rendering technique. However, since numerical errors will be propagated all the way from $f(0)$ to the last term, $f(n\delta) = f(1)$, it is **not numerically stable**.

Cubic Uniform B-Spline Curves

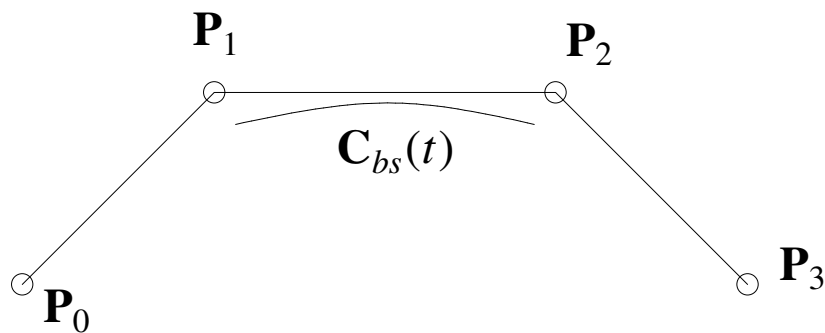
- A curve representation with **local property**

A Cubic Uniform B-Spline Curve segment

For four given control points \mathbf{P}_0 , \mathbf{P}_1 , \mathbf{P}_2 and \mathbf{P}_3 , a cubic uniform B-spline curve segment is defined as follows:

$$\mathbf{C}_{bs}(t) = \frac{(1-t)^3}{6} \mathbf{P}_0 + \frac{(4-6t^2+3t^3)}{6} \mathbf{P}_1 + \frac{(1+3t+3t^2-3t^3)}{6} \mathbf{P}_2 + \frac{t^3}{6} \mathbf{P}_3$$

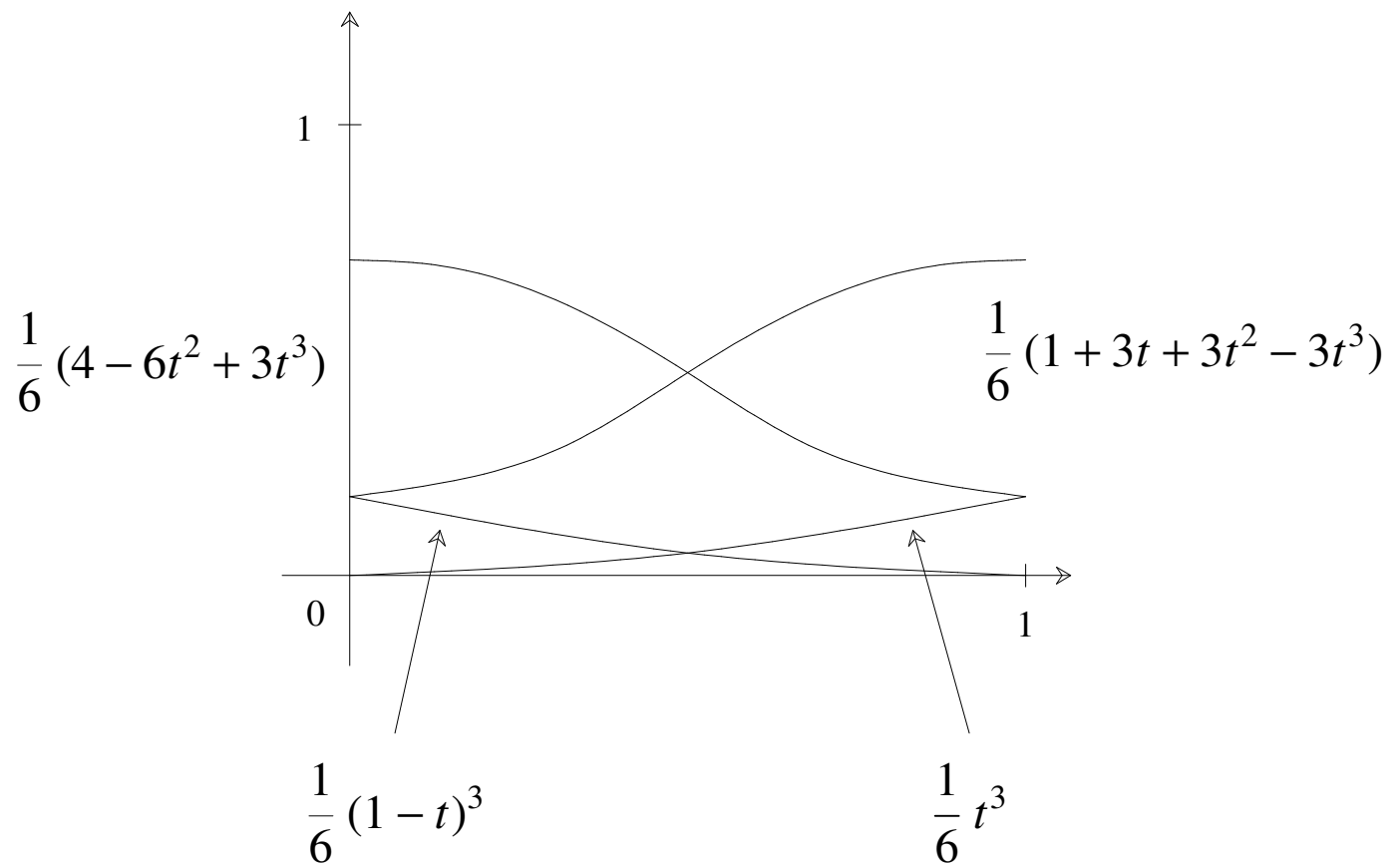
$$0 \leq t \leq 1$$



Matrix form

$$\mathbf{C}_{bs}(t) = [1, t, t^2, t^3] \frac{1}{6} \begin{bmatrix} 1 & 4 & 1 & 0 \\ -3 & 0 & 3 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{P}_0 \\ \mathbf{P}_1 \\ \mathbf{P}_2 \\ \mathbf{P}_3 \end{bmatrix} = \mathbf{T} \cdot \mathbf{M}_{bs} \cdot \mathbf{G}$$

Blending functions



Properties of B-spline blending functions

- Non-negative
- Sum = 1
- Hence, again, a B-spline curve segment is always contained in the **convex hull** of its control points.
- However, $\mathbf{C}_{bs}(0) \neq \mathbf{P}_0$ and $\mathbf{C}_{bs}(1) \neq \mathbf{P}_3$. Actually

$$\mathbf{C}_{bs}(0) = \frac{1}{6} \mathbf{P}_0 + \frac{2}{3} \mathbf{P}_1 + \frac{1}{6} \mathbf{P}_2$$

$$\mathbf{C}_{bs}(1) = \frac{1}{6} \mathbf{P}_1 + \frac{2}{3} \mathbf{P}_2 + \frac{1}{6} \mathbf{P}_3$$

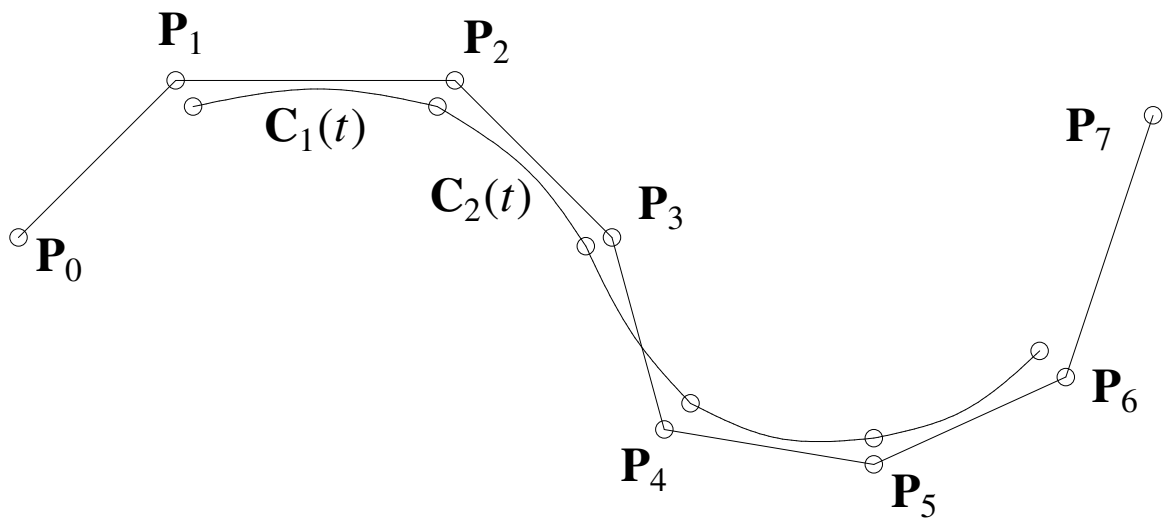
A Cubic Uniform B-Spline Curve

Given a set of n control points, one can define a cubic (uniform) B-spline curve with $(n - 3)$ segments.

The first segment, $C_1(t)$, is defined by the first four control points: P_0, P_1, P_2, P_3 .

The second segment, $C_2(t)$, is defined by the second four control points: P_1, P_2, P_3, P_4

The last one, $C_{n-3}(t)$, by $P_{n-3}, P_{n-2}, P_{n-1}, P_n$.



Properties/Advantages of a B-spline curve

- Local property (changing one control point will affect at most four segments)
- C^2 continuity at the joints
- Compact form for multiple segments
- Can use multiple control points to achieve exact point interpolation

