Naval Architecture & Ocean Engineering

Computer Aided Ship Design

Part II. Curve and Surface Modeling

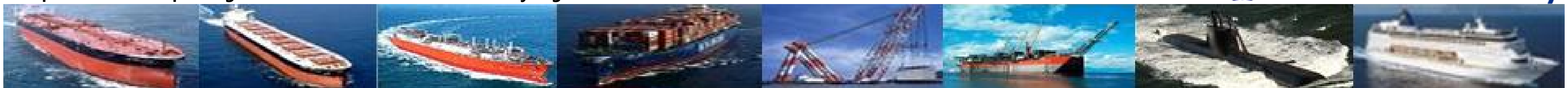
Ch. 2 Bezier Curves

September, 2013

Prof. Myung-Il Roh

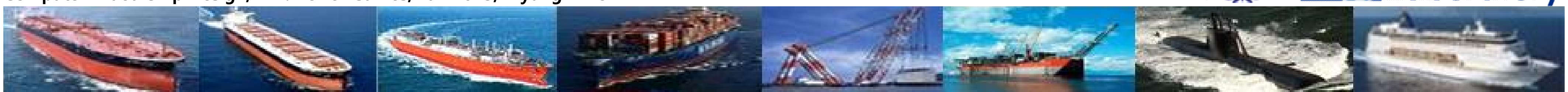
Department of Naval Architecture and Ocean Engineering,
Seoul National University of College of Engineering

SYstem Design Laboratory



Chapter 2. Bezier Curves

SYstem Design Laboratory

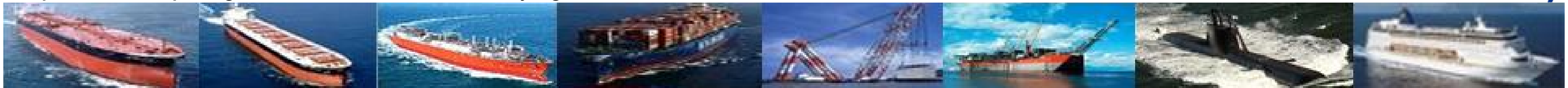


2.1 Parametric Function and Curves

(1) Explicit Function, Implicit Function, Parametric Function

(2) Characteristics of Parametric Function

(3) Expression of General Function Using Parametric Function



Explicit / Implicit / Parametric Function

☑ Explicit function

- A function of the form of y=f(x) is called "explicit function".
- y can be obtained easily if x is given.

$$ex)\quad y = \sqrt{r^2 - x^2}$$

☑ Implicit function

- A function of the form of f(x, y)=0 is called "implicit function".
- It is easy to check that the given point is inside or outside, left or right of the curve.
- Implicit function is not always possible to transform into an explicit form.

$$ex)\quad x^2 + y^2 - r^2 = 0$$

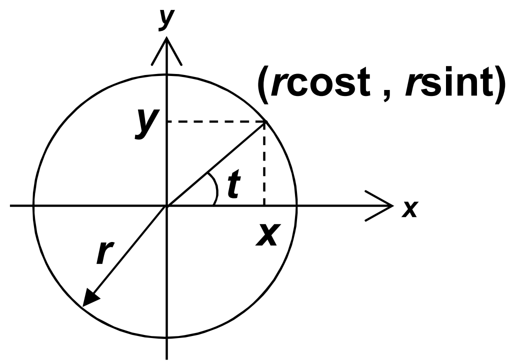
$$ex)\quad (0)^2 + (0)^2 - r^2 < 0$$

$$(r)^2 + (r)^2 - r^2 > 0$$

$$ex)\quad y = \pm\sqrt{r^2 - x^2}$$

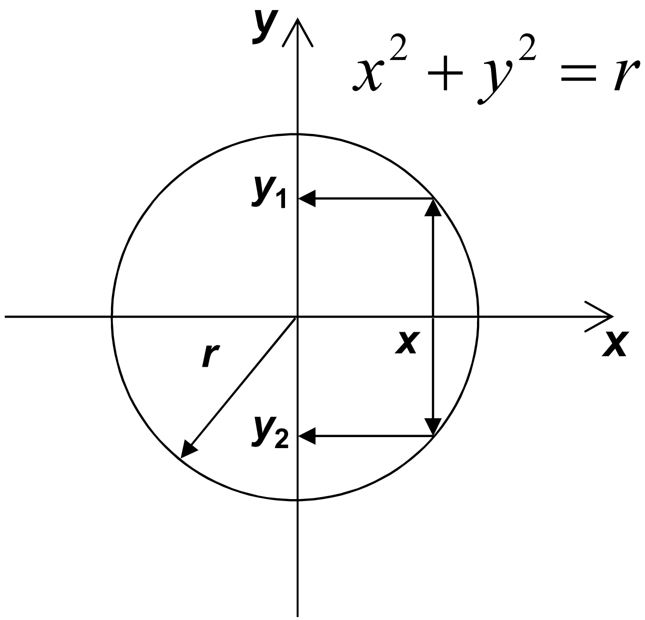
☑ Parametric function

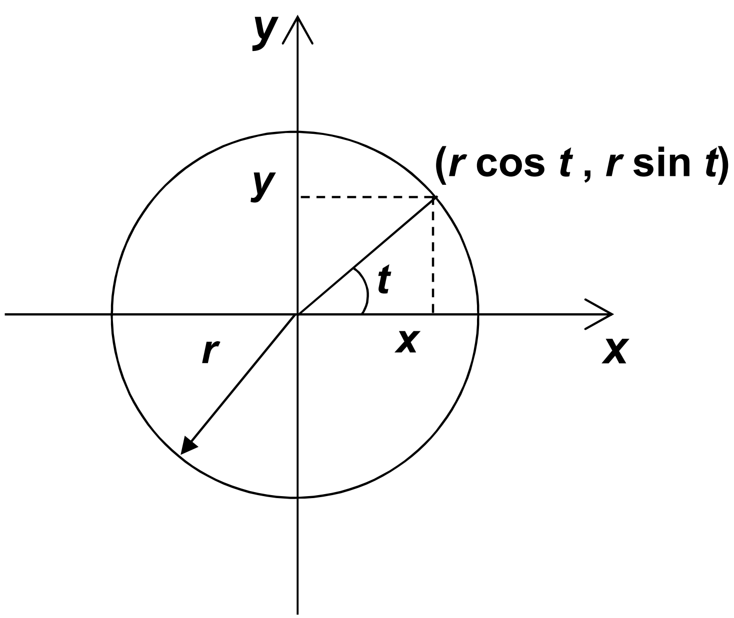
- A function y=f(x) can be expressed by x=f(t), y=g(t) using the parameter 't'. We call it "parametric function".
- Every explicit function is possible to transform into a parametric form.



$$ex)\quad x(t) = r\cos t,\ \ y(t) = r\sin t$$

Characteristics of Parametric Function (1/3)





☑ **General function**

- **y value of more than two can be obtained for an x value (multi-value function).**

$$x^2 + y^2 = r^2 \qquad y = \pm\sqrt{r^2 - x^2}$$

- **It may be difficult to express derivatives.** $\left.\dfrac{dy}{dx}\right|_{x=r} = \infty$

☑ **Parametric function**

- **A parameter value has only one result.**

$$x(t) = r\cos t,\ y(t) = r\sin t$$

- **It is easy to express derivatives.**
  ➨ **Calculate dy/dx dividing each elements: dx/dt, dy/dt**

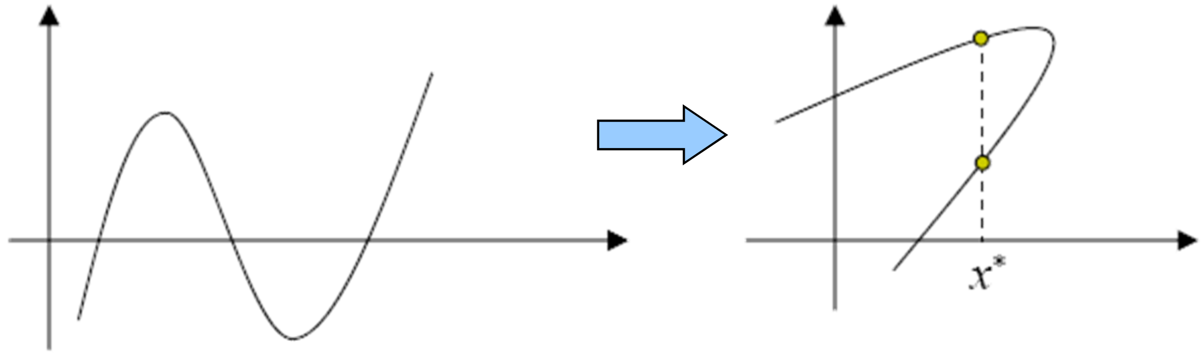
$$\frac{dx}{dt} = -r\sin t,\quad \frac{dy}{dt} = r\cos t$$

Characteristics of Parametric Function (2/3)

☑ Explicit function of y=f(x)

- It is difficult to express as an explicit function* again after original explicit function is rotated.

*dimensional extension



$y = a_0 + a_1 x^1 + a_2 x^2 + a_3 x^3$ $\qquad$ $y = ??$

☑ Implicit function of f(x, y)=0

- Points on the curve can not be calculated in order.
- Dimensional extension is difficult.

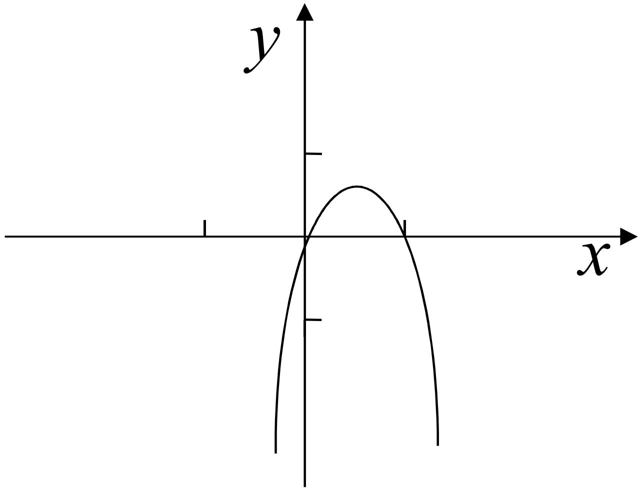
☑ Parametric function of x = f(t), y = g(t)

- Points on the curve can be easily calculated in order by varying the parameters.
- Dimensional extension is easy.
- These are the reasons why parametric function is commonly used for CAD systems.

Characteristics of Parametric Function (3/3)

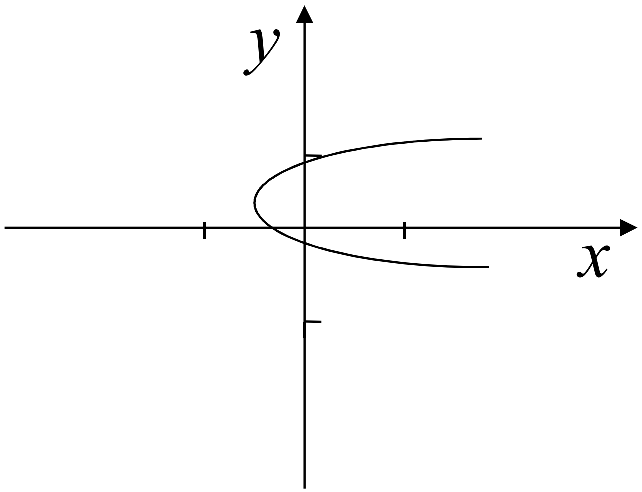
☑ **A curve is defined by the parametric functions as follows:**

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} t \\ 2t - 2t^2 \end{bmatrix}$$



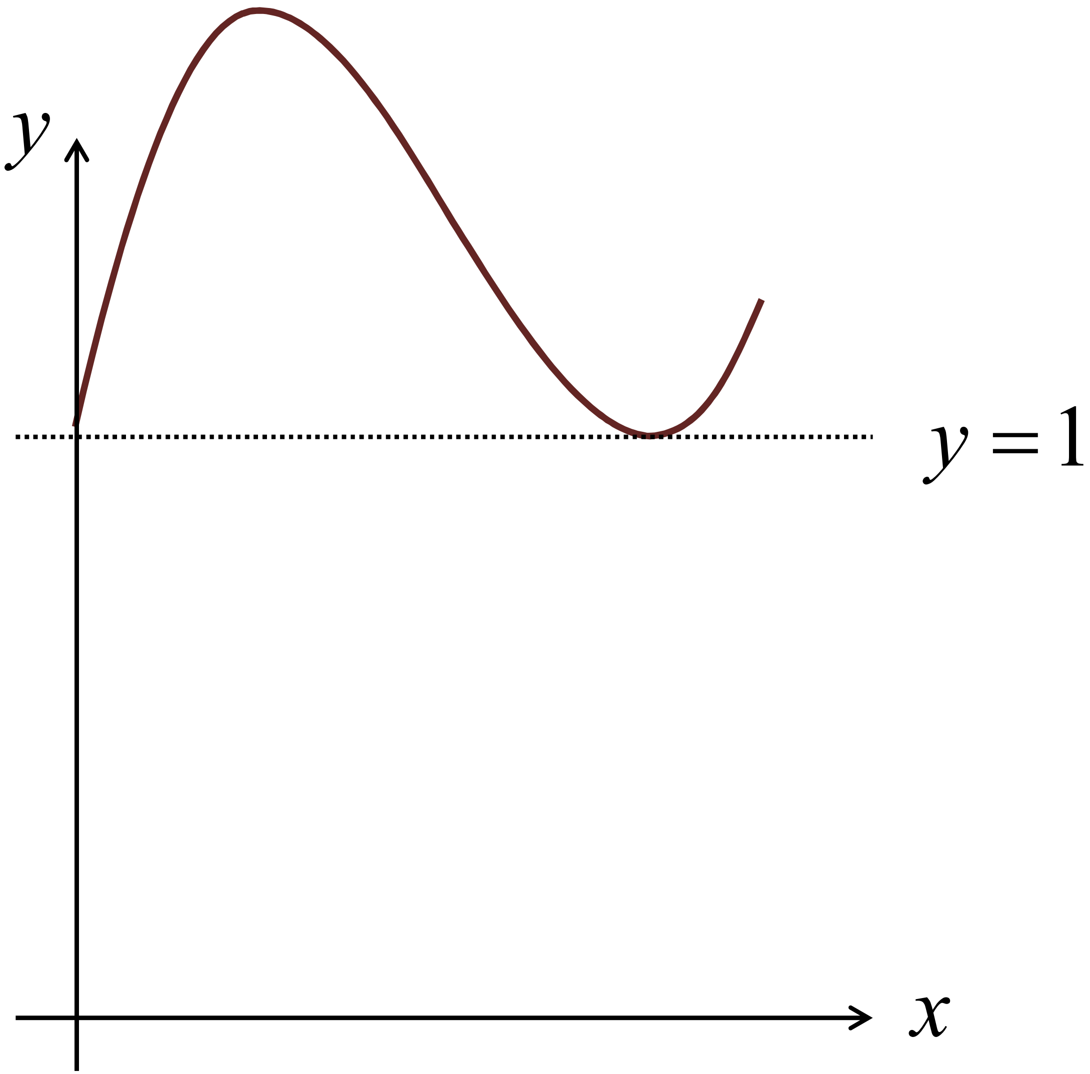
☑ **If the curve is rotated with angle of 90°, geometry('topology') is not changed, only its position vector are changed.**

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -2t + 2t^2 \\ t \end{bmatrix}$$



Expression of General Function by Using Parametric Function (1/6)

Given: $y = 2x^3 - 4x^2 + 2x + 1$



Expression of General Function by Using Parametric Function (2/6)

$$y = 2x^3 - 4x^2 + 2x + 1$$

$$\mathbf{r}(t) = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} t \\ 2t^3 - 4t^2 + 2t + 1 \end{bmatrix}$$

- **From this parametric function with coefficient 2, -4, 2, 1, it is not at all obvious what the function might look like.**

- **Alternatively, we can express the function in another way as follows:**

$$\mathbf{r}(t) = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} (1-t)^3 x_0 + 3t(1-t)^2 x_1 + 3t^2(1-t)x_2 + t^3 x_3 \\ (1-t)^3 y_0 + 3t(1-t)^2 y_1 + 3t^2(1-t)y_2 + t^3 y_3 \end{bmatrix}$$

$$\begin{bmatrix} t \\ 2t^3 - 4t^2 + 2t + 1 \end{bmatrix} = \begin{bmatrix} (1-t)^3 x_0 + 3t(1-t)^2 x_1 + 3t^2(1-t)x_2 + t^3 x_3 \\ (1-t)^3 y_0 + 3t(1-t)^2 y_1 + 3t^2(1-t)y_2 + t^3 y_3 \end{bmatrix}$$

Expression of General Function by Using Parametric Function (3/6)

$$(1-t)^3 x_0 + 3t(1-t)^2 x_1 + 3t^2(1-t)x_2 + t^3 x_3 = t$$

Coefficient of constant:	$x_0 = 0$		$x_0 = 0$
Coefficient of *t*:	$-3x_0 + 3x_1 = 1$	➡	$x_1 = 1/3$
Coefficient of *t*²:	$3x_0 - 6x_1 + 3x_2 = 0$		$x_2 = 2/3$
Coefficient of *t*³:	$-x_0 + 3x_1 - 3x_2 + x_3 = 0$		$x_3 = 1$

$$b_{x_i}^0 = x_i = \frac{i}{n}$$

Linear Precision

* Linear precision: If the control points b_1 and b_2 are evenly spaced on the straight line between b_0 and b_3, the cubic Bezier curve is the linear interpolant between b_0 and b_3.

* Farin, G.E., The Essentials of CAGD, 2000, p.29

Expression of General Function by Using Parametric Function (4/6)

$$(1-t)^3 y_0 + 3t(1-t)^2 y_1 + 3t^2(1-t) y_2 + t^3 y_3 = 2t^3 - 4t^2 + 2t + 1$$

Coefficient of constant: $y_0 = 1$

Coefficient of *t*: $-3y_0 + 3y_1 = 2$

Coefficient of *t*²: $3y_0 - 6y_1 + 3y_2 = -4$

Coefficient of *t*³: $-y_0 + 3y_1 - 3y_2 + y_3 = 2$

$\Rightarrow$

$y_0 = 1$

$y_1 = 5/3$

$y_2 = 1$

$y_3 = 1$

Expression of General Function by Using Parametric Function (5/6)

$x_0 = 0, x_1 = 1/3, x_2 = 2/3, x_3 = 1$

$y_0 = 1, y_1 = 5/3, y_2 = 1, y_3 = 1$

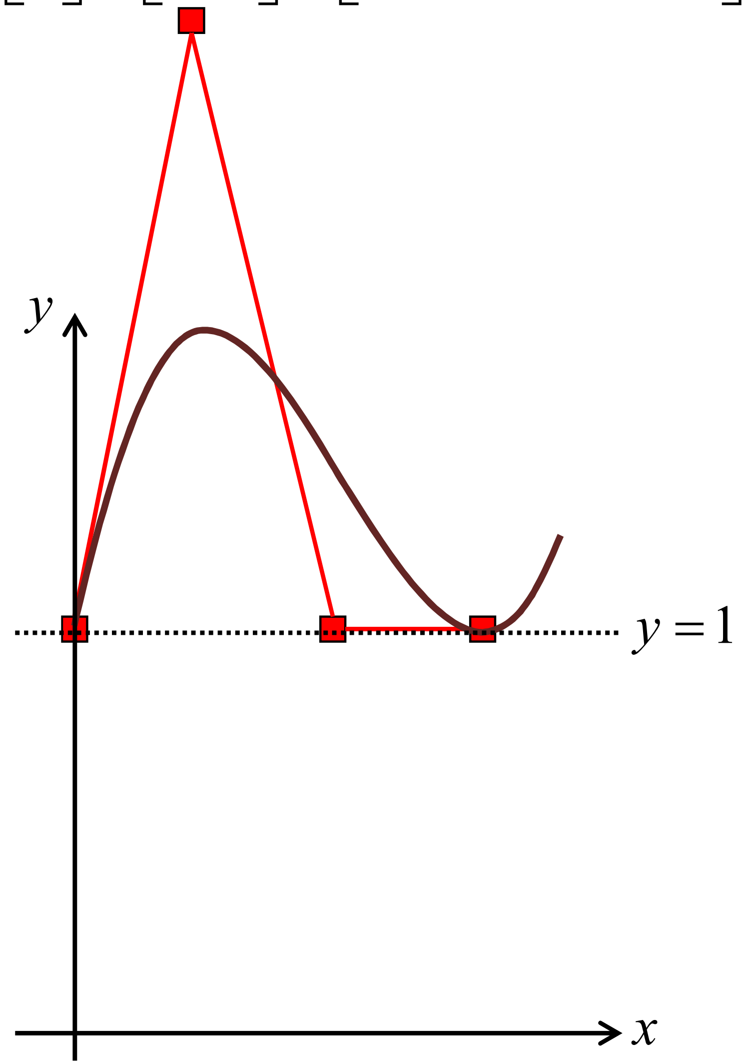
$$y = 2x^3 - 4x^2 + 2x + 1$$

$$\mathbf{r}(t) = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} t \\ 2t^3 - 4t^2 + 2t + 1 \end{bmatrix} = \begin{bmatrix} (1-t)^3 \cdot 0 + 3t(1-t)^2 \cdot \frac{1}{3} + 3t^2(1-t) \cdot \frac{2}{3} + t^3 \cdot 1 \\ (1-t)^3 \cdot 1 + 3t(1-t)^2 \cdot \frac{5}{3} + 3t^2(1-t) \cdot 1 + t^3 \cdot 1 \end{bmatrix}$$

$$= (1-t)^3 \begin{bmatrix} 0 \\ 1 \end{bmatrix} + 3t(1-t)^2 \begin{bmatrix} 1/3 \\ 5/3 \end{bmatrix} + 3t^2(1-t) \begin{bmatrix} 2/3 \\ 1 \end{bmatrix} + t^3 \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$= B_0^3(t) \begin{bmatrix} 0 \\ 1 \end{bmatrix} + B_1^3(t) \begin{bmatrix} 1/3 \\ 5/3 \end{bmatrix} + B_2^3(t) \begin{bmatrix} 2/3 \\ 1 \end{bmatrix} + B_3^3(t) \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$= B_0^3 \mathbf{b}_0 + B_1^3 \mathbf{b}_1 + B_2^3 \mathbf{b}_2 + B_3^3 \mathbf{b}_3$$



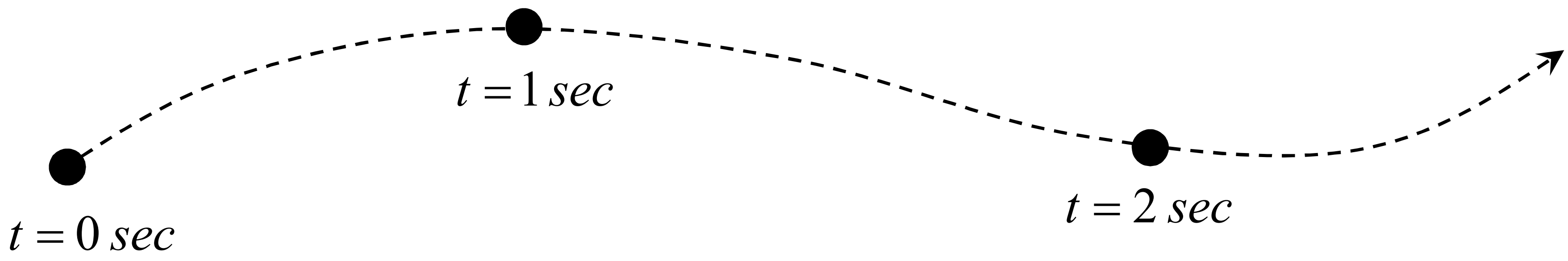
When the points are connected to the line, it shows a similar appearance for the original curve.

Expression of General Function by Using Parametric Function (6/6)

$$\mathbf{r}(t) = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} t \\ 2t^3 - 4t^2 + 2t + 1 \end{bmatrix} = \begin{bmatrix} (1-t)^3 \cdot 0 + 3t(1-t)^2 \cdot \frac{1}{3} + 3t^2(1-t) \cdot \frac{2}{3} + t^3 \cdot 1 \\ (1-t)^3 \cdot 1 + 3t(1-t)^2 \cdot \frac{5}{3} + 3t^2(1-t) \cdot 1 + t^3 \cdot 1 \end{bmatrix}$$

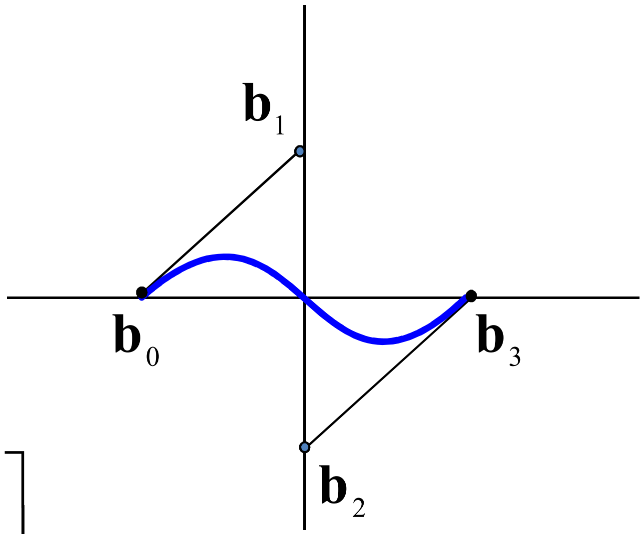
- **If the parameter 't' is time, then r(t) can be regarded as the moving trajectory of a rigid body.**
- **In explicit or implicit function, it is only possible to express the moving trajectory of a rigid body, whereas the parametric function can express the position, velocity, acceleration of r(t) in specific time 't'.**

$\mathrm{r}(t)$: Position of body, $\dot{\mathrm{r}}(t)$: Velocity of body, $\ddot{\mathrm{r}}(t)$: Acceleration of body



"Blending" the Points in Space and Parametric Functions

- Curves can be represented by "blending" the points in space and parametric functions.
- If these points are moved, then the shape of the curve is changed.
- So, these points are called "control points".



$$r(t) = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -(1-t)^3 + t^3 \\ 3(1-t)^2 t - 3(1-t)t^2 \end{bmatrix}$$

$$= (1-t)^3 \begin{bmatrix} -1 \\ 0 \end{bmatrix} + 3(1-t)^2 t \begin{bmatrix} 0 \\ 1 \end{bmatrix} + 3(1-t)t^2 \begin{bmatrix} 0 \\ -1 \end{bmatrix} + t^3 \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$= B_0^3 \begin{bmatrix} -1 \\ 0 \end{bmatrix} + B_1^3 \begin{bmatrix} 0 \\ 1 \end{bmatrix} + B_2^3 \begin{bmatrix} 0 \\ -1 \end{bmatrix} + B_3^3 \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

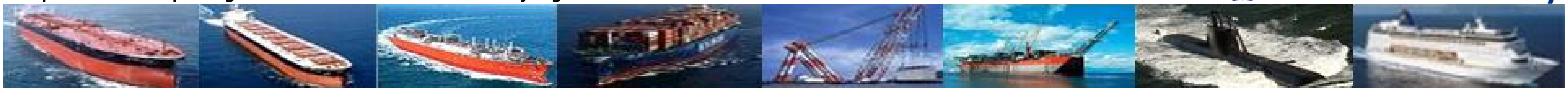
$$= B_0^3 \mathbf{b}_0 + B_1^3 \mathbf{b}_1 + B_2^3 \mathbf{b}_2 + B_3^3 \mathbf{b}_3$$

➡ French engineer P. Bezier at Renault formulated it in 1971.

2.2 Bezier Curves

(1) Definition of Cubic "Bezier" Curves
(2) Characteristics of Bezier Curves
(3) Derivatives of Cubic Bezier Curves
(4) Higher Order Bezier Curves
(5) Derivatives of Higher Order Bezier Curves
(6) Matrix Form of Bezier Curves

SYstem Design Laboratory



Definition of Cubic "Bezier" Curves

$$\mathbf{r}(u) = \begin{bmatrix} x(u) \\ y(u) \\ z(u) \end{bmatrix} = \sum_{i=0}^{D-1} \mathbf{d}_i N_i^n(u)$$

B-spline curve

☑ **Cubic Bezier curve is defined by**

$$if \quad c_1 B_0^3(t) + c_2 B_1^3(t) + c_3 B_2^3(t) + c_4 B_3^3(t) = 0,$$
$$then \quad c_1 = c_2 = c_3 = c_4 = 0$$

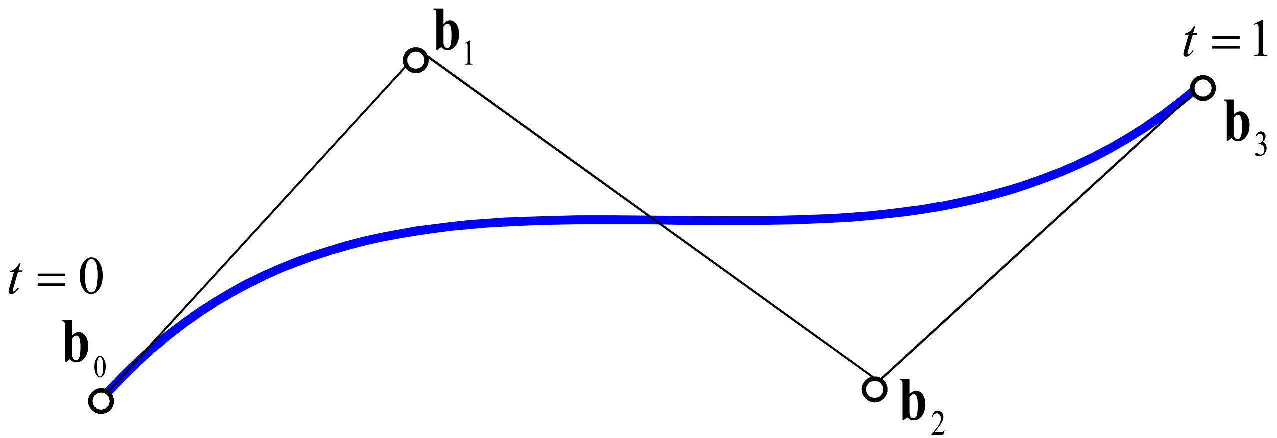
$$\mathbf{r}(t) = \begin{bmatrix} \mathbf{x}(t) \\ \mathbf{y}(t) \end{bmatrix} \; or \; \begin{bmatrix} \mathbf{x}(t) \\ \mathbf{y}(t) \\ \mathbf{z}(t) \end{bmatrix} = (1-t)^3 \mathbf{b}_0 + 3(1-t)^2 t \, \mathbf{b}_1 + 3(1-t)t^2 \, \mathbf{b}_2 + t^3 \, \mathbf{b}_3$$
$$= B_0^3(t) \, \mathbf{b}_0 + B_1^3(t) \, \mathbf{b}_1 + B_2^3(t) \, \mathbf{b}_2 + B_3^3(t) \, \mathbf{b}_3$$

linearly independent

where, $\mathbf{b}_i$: Bezier control points (b_{ix}, b_{iy}) or (b_{ix}, b_{iy} , b_{iz})

$B_i^3(t)$: cubic Bernstein polynomial or Bernstein basis function $\quad \sum_{i=0}^{3} B_i^3(t) = 1, \quad B_i^3(t) \ge 0$

$0 \le t \le 1$: Bezier curve parameter

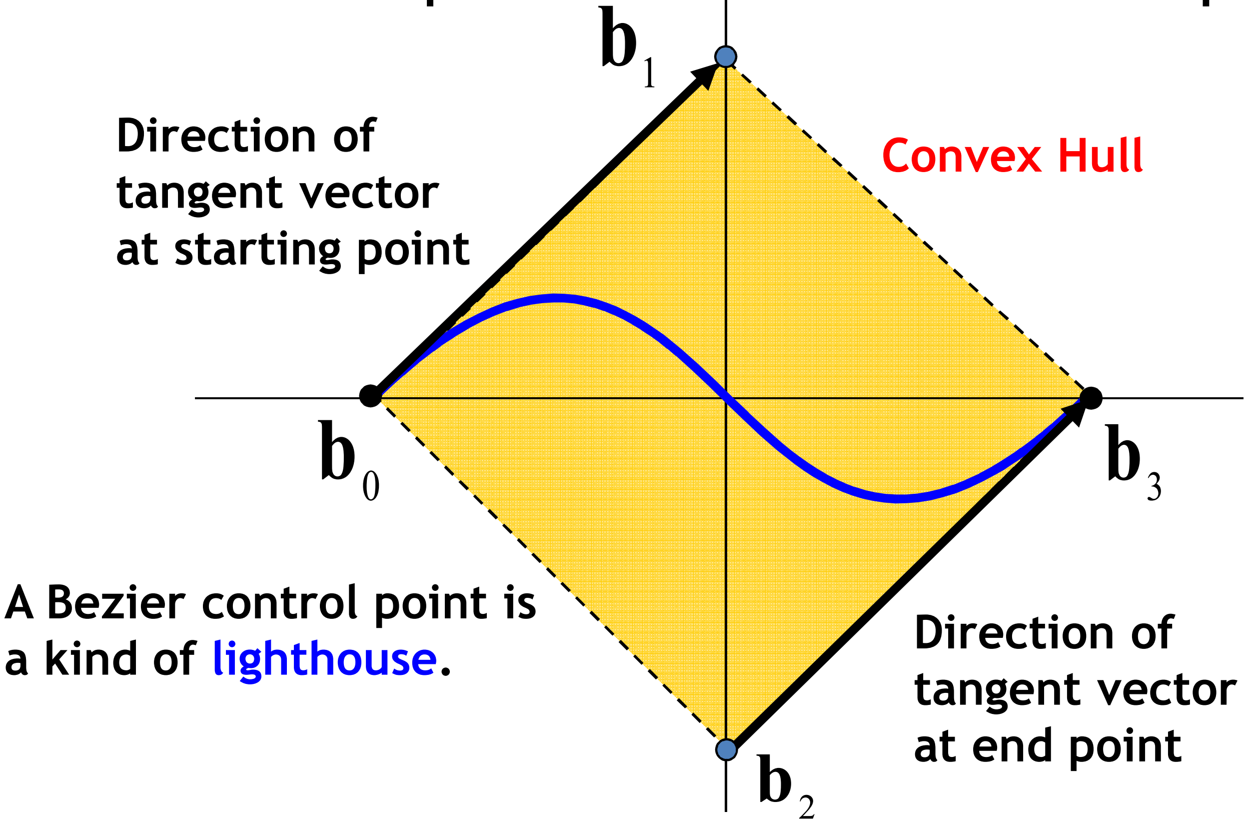


Characteristics of Bezier Curves (1/2)

- Bezier curves are represented in a *convex hull* which is composed of the outer control points[1).

$$\left(\because \sum_{i=0}^{3} B_i^3(t) = 1\right)$$

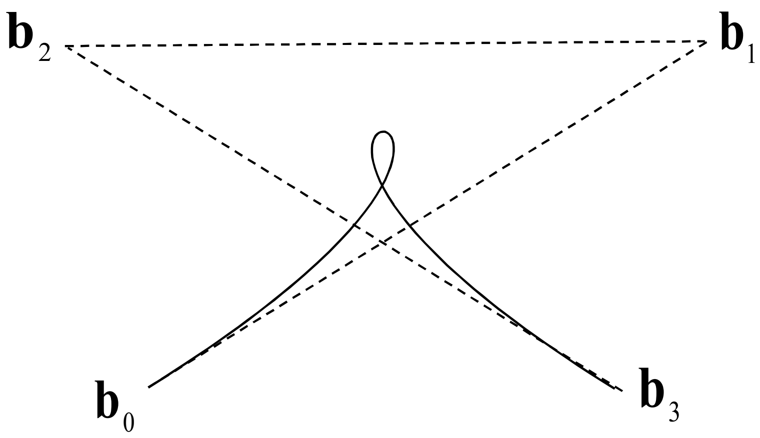
- The direction of tangent vector at the start and end points can be obtained from the first two control points and the last two control points.



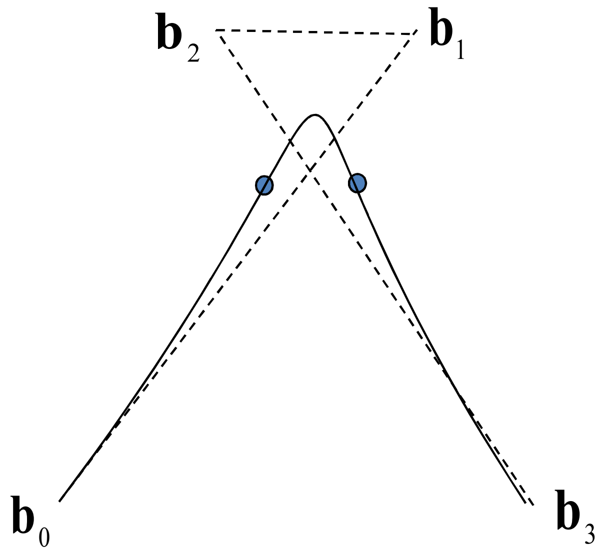
1) Convex Hull Property: For all t, the curve r(t) is in the convex hull of the control polygon.

Characteristics of Bezier Curves (2/2)

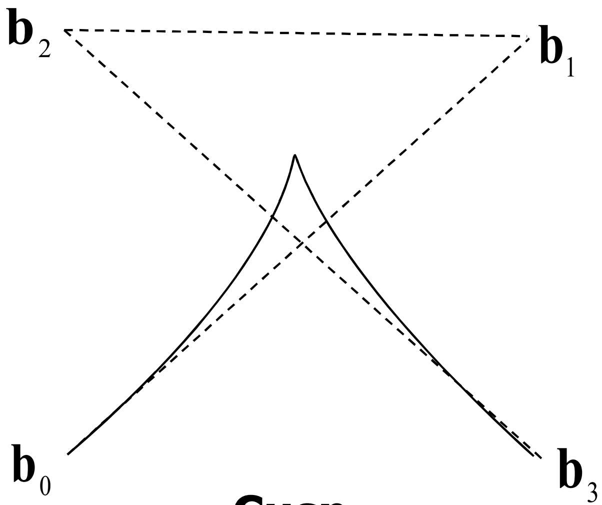
- If the control points are moved, then shape of the curve is changed.



Loop



Two inflection points



Cusp

Derivatives of Cubic Bezier Curves (1/2)

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

☑ **First derivatives: Tangent vector of the curve**

➡ **"Velocity of body at time = t"**

$$\frac{d\mathbf{r}(t)}{dt} = -3(1-t)^2 \mathbf{b}_0 + [3(1-t)^2 - 6(1-t)t]\mathbf{b}_1 + [6(1-t)t - 3t^2]\mathbf{b}_2 + 3t^2 \mathbf{b}_3$$

$$= 3[\mathbf{b}_1 - \mathbf{b}_0](1-t)^2 + 6[\mathbf{b}_2 - \mathbf{b}_1](1-t)t + 3[\mathbf{b}_3 - \mathbf{b}_2]t^2$$

$$= 3\Delta\mathbf{b}_0 (1-t)^2 + 6\Delta\mathbf{b}_1 (1-t)t + 3\Delta\mathbf{b}_2 t^2$$

$$= 3(\Delta\mathbf{b}_0 B_0^2 + \Delta\mathbf{b}_1 B_1^2 + \Delta\mathbf{b}_2 B_2^2)$$

where, $\Delta\mathbf{b}_i = \mathbf{b}_{i+1} - \mathbf{b}_i$ **: forward differences**

Derivatives of Cubic Bezier Curves (2/2)

☑ The derivative of the cubic curve is quadratic curve.

$$\dot{\mathbf{r}}(t) = \frac{d\mathbf{r}(t)}{dt} = 3(\Delta\mathbf{b}_0 B_0^2 + \Delta\mathbf{b}_1 B_1^2 + \Delta\mathbf{b}_2 B_2^2)$$

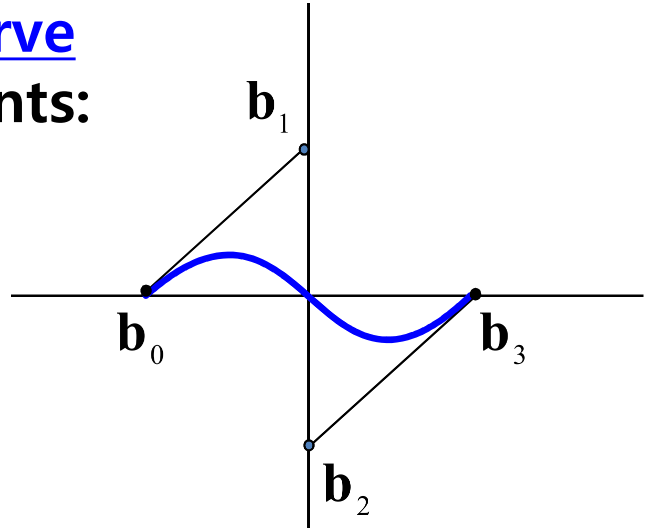
$$= 3\Delta\mathbf{b}_0 (1-t)^2 + 6\Delta\mathbf{b}_1 (1-t)t + 3\Delta\mathbf{b}_2 t^2$$

where, B_i^2 : quadratic Bernstein basis function

☑ Most important tangent vectors at the curve is tangent vectors at starting and end points:

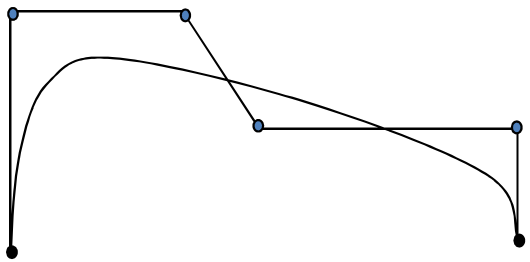
$$\dot{\mathbf{r}}(0) = 3\Delta\mathbf{b}_0 = 3(\mathbf{b_1} - \mathbf{b_o}),$$

$$\dot{\mathbf{r}}(1) = 3\Delta\mathbf{b}_2 = 3(\mathbf{b}_3 - \mathbf{b}_2)$$



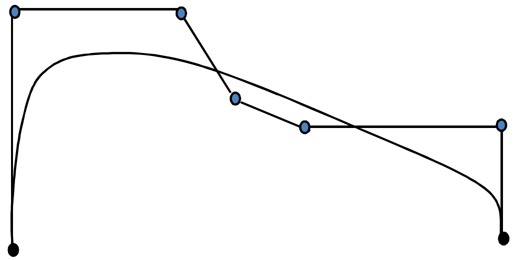
Higher Order Bezier Curves (1/3)

5th-degree Bezier curve



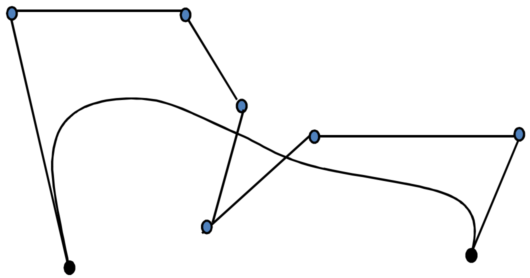
Degree = 5
No of control points = 6

6th-degree Bezier curve



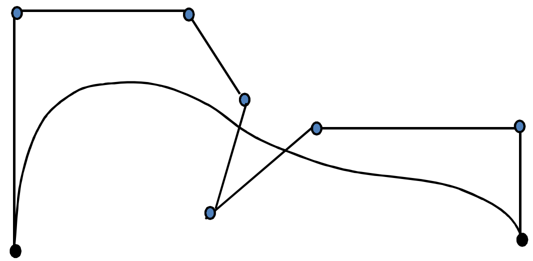
Degree = 6
No of control points = 7

7th-degree Bezier curve



Degree = 7
No of control points = 8

7th-degree Bezier curve



Degree = 7
No of control points = 8

Higher Order Bezier Curves (2/3)

☑ A Bezier curve of degree *n* can be defined by

$$\mathbf{r}(t) = \mathbf{b}_0 B_0^n(t) + \mathbf{b}_1 B_1^n(t) + \ldots\ldots + \mathbf{b}_n B_n^n(t).$$

☑ where, $B_i^n(t)$: Bernstein Polynomial Function.

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i},$$

$$\binom{n}{i} = {}_nC_i = \begin{cases} \dfrac{n!}{i!(n-i)!} & \text{if } 0 \le i \le n \\ \mathbf{0} & \text{else} \end{cases}$$

$$B_i^n(t) = tB_{i-1}^{n-1}(t) + (1-t)B_i^{n-1}(t) \quad \text{with } B_0^0(t) \equiv 1$$

☑ For cubic case(= degree 3), the Bezier curve is

$$\mathbf{r}(t) = \mathbf{b}_0 B_0^3(t) + \mathbf{b}_1 B_1^3(t) + \mathbf{b}_2 B_2^3(t) + \mathbf{b}_3 B_3^3(t).$$

Higher Order Bezier Curves (3/3)

☑ **Bernstein Polynomial Function is defined by;**

For quadratic(2nd-degree)

$$[(1-t)+t]^2 = (1-t)^2 + 2(1-t)t + t^2$$
$$= B_0^2(t) + B_1^2(t) + B_2^2(t),$$

For cubic(3rd-degree)

$$[(1-t)+t]^3 = [(1-t)+t]^2[(1-t)+t]$$
$$= (1-t)^3 + 3(1-t)^2 t + 3(1-t)t^2 + t^3$$
$$= B_0^3(t) + B_1^3(t) + B_2^3(t) + B_3^3(t),$$

For quartic(4th-degree)

$$[(1-t)+t]^4 = [(1-t)+t]^3[(1-t)+t]$$
$$= (1-t)^4 + 4(1-t)^3 t + 6(1-t)^2 t^2 + 4(1-t)t^3 + t^4$$
$$= B_0^4(t) + B_1^4(t) + B_2^4(t) + B_3^4(t) + B_4^4(t)$$

1

1 1

1 2 1

1 3 3 1

1 4 6 4 1

Pascal's triangle

Derivatives of Higher Order Bezier Curves (1/2)

☑ **For cubic case (n = 3),**

$$\dot{\mathbf{r}}(t) = 3[\Delta\mathbf{b}_0 B_0^2 + \Delta\mathbf{b}_1 B_1^2 + \Delta\mathbf{b}_2 B_2^2].$$

☑ **For degree = n,**

$$\dot{\mathbf{r}}(t) = n[\Delta\mathbf{b}_0 B_0^{n-1} + \Delta\mathbf{b}_1 B_1^{n-1} + \ldots\ldots\ldots + \Delta\mathbf{b}_{n-1} B_{n-1}^{n-1}].$$

where $\Delta\mathbf{b}_i = \mathbf{b}_{i+1} - \mathbf{b}_i$ **: forward difference.**

☑ **Bezier Curve ➡ differentiated by more than one by parameter 't'.**

☑ **For the k^{th} times derivative:**

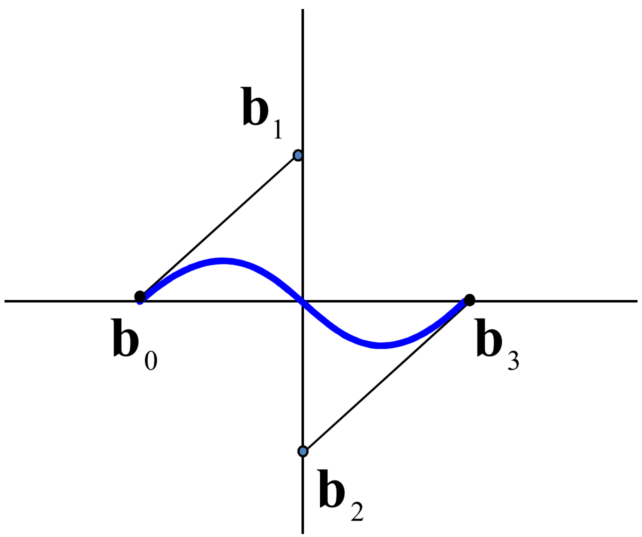
$$\frac{d^k\mathbf{r}(t)}{dt^k} = \frac{n!}{(n-k)!}[\Delta^k\mathbf{b}_0 B_0^{n-k}(t) + \Delta^k\mathbf{b}_1 B_1^{n-k}(t) \ldots\ldots + \Delta^k\mathbf{b}_{n-k} B_{n-k}^{n-k}(t)].$$

Derivatives of Higher Order Bezier Curves (2/2)

where, Δ^k : forward operator.

- we can get $\Delta^k \mathbf{b}_i = \Delta^{k-1}\mathbf{b}_{i+1} - \Delta^{k-1}\mathbf{b}_i$.
  where, $\Delta^0 \mathbf{b}_i = \mathbf{b}_i$.
- for k=2 : $\mathbf{b}_{i+2} - 2\mathbf{b}_{i+1} + \mathbf{b}_i$.
- for k=3 : $\mathbf{b}_{i+3} - 3\mathbf{b}_{i+2} + 3\mathbf{b}_{i+1} - \mathbf{b}_i$.
- for k=4 : $\mathbf{b}_{i+4} - 4\mathbf{b}_{i+3} + 6\mathbf{b}_{i+2} - 4\mathbf{b}_{i+1} + \mathbf{b}_i$.
- the k^{th} derivative of $\mathbf{r}(0)$ and $\mathbf{r}(1)$;

$$\mathbf{r}^k(0) = \frac{n!}{(n-k)!}\Delta^k \mathbf{b}_0 \quad \text{and} \quad \mathbf{r}^k(1) = \frac{n!}{(n-k)!}\Delta^k \mathbf{b}_{n-k}.$$



For n=3, k=2;

$$\begin{aligned}\mathbf{r}^2(0) &= \frac{3!}{(3-2)!}\Delta^2\mathbf{b}_0 \\ &= 6(\Delta^1\mathbf{b}_1 - \Delta^1\mathbf{b}_0) \\ &= 6((\Delta^0\mathbf{b}_2 - \Delta^0\mathbf{b}_1) - (\Delta^0\mathbf{b}_1 - \Delta^0\mathbf{b}_0)) \\ &= 6(\Delta^0\mathbf{b}_2 - 2\Delta^0\mathbf{b}_1 + \Delta^0\mathbf{b}_0) \\ &= 6(\mathbf{b}_2 - 2\mathbf{b}_1 + \mathbf{b}_0)\end{aligned}$$

$$\begin{aligned}\mathbf{r}^2(1) &= \frac{3!}{(3-2)!}\Delta^2\mathbf{b}_1 \\ &= 6(\Delta^1\mathbf{b}_2 - \Delta^1\mathbf{b}_1) \\ &= 6((\Delta^0\mathbf{b}_3 - \Delta^0\mathbf{b}_2) - (\Delta^0\mathbf{b}_2 - \Delta^0\mathbf{b}_1)) \\ &= 6(\Delta^0\mathbf{b}_3 - 2\Delta^0\mathbf{b}_2 + \Delta^0\mathbf{b}_1) \\ &= 6(\mathbf{b}_3 - 2\mathbf{b}_2 + \mathbf{b}_1)\end{aligned}$$

Matrix Form of Bezier Curves (1/3)

☑ Cubic Bezier Curve

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

☑ Applying the dot product to above equation, we can get

$$\mathbf{r}(t) = \begin{bmatrix}\mathbf{b}_0 & \mathbf{b}_1 & \mathbf{b}_2 & \mathbf{b}_3\end{bmatrix}\begin{bmatrix}(1-t)^3 \\ 3(1-t)^2 t \\ 3(1-t)\,t^2 \\ t^3\end{bmatrix}$$

Matrix Form of Bezier Curves (2/3)

☑ The matrix form of the Bezier curve is

$$\mathbf{r}(t)=\begin{bmatrix}\mathbf{b}_0 & \mathbf{b}_1 & \mathbf{b}_2 & \mathbf{b}_3\end{bmatrix}\begin{bmatrix}(1-t)^3 \\ 3(1-t)^2 t \\ 3(1-t)\,t^2 \\ t^3\end{bmatrix}=\begin{bmatrix}\mathbf{b}_0 & \mathbf{b}_1 & \mathbf{b}_2 & \mathbf{b}_3\end{bmatrix}\begin{bmatrix}B_0^3(\mathbf{t}) \\ B_1^3(\mathbf{t}) \\ B_2^3(\mathbf{t}) \\ B_3^3(\mathbf{t})\end{bmatrix}$$

Conversion into the monomial curve: $\mathbf{r}(t)=\mathbf{a}_0+\mathbf{a}_1 t+\mathbf{a}_2 t^2+\mathbf{a}_3 t^3$

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

$$=\begin{bmatrix}\mathbf{a}_0 & \mathbf{a}_1 & \mathbf{a}_2 & \mathbf{a}_3\end{bmatrix}\begin{bmatrix}1 \\ t \\ t^2 \\ t^3\end{bmatrix}$$

Matrix Form of Bezier Curves (3/3)

☑ **The matrix form of the monomial curve is**

$$\mathbf{r}(t) = \mathbf{a}_0 + \mathbf{a}_1 t + \mathbf{a}_2 t^2 + \mathbf{a}_3 t^3 = \begin{bmatrix} \mathbf{a_0} & \mathbf{a_1} & \mathbf{a_2} & \mathbf{a_3} \end{bmatrix} \begin{bmatrix} 1 \\ t \\ t^2 \\ t^3 \end{bmatrix}$$

Transformation into the Bezier form:

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

$$= B_0^3(t)\,\mathbf{b}_0 + B_1^3(t)\,\mathbf{b}_1 + B_2^3(t)\,\mathbf{b}_2 + B_3^3(t)\,\mathbf{b}_3$$

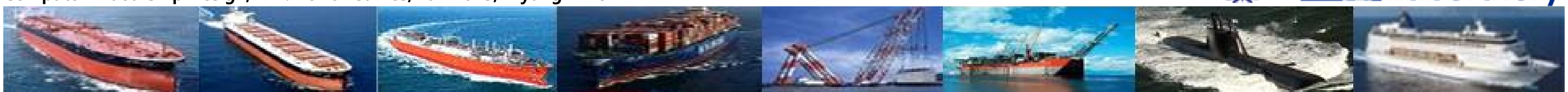
$$= \begin{bmatrix} \mathbf{b_0} & \mathbf{b_1} & \mathbf{b_2} & \mathbf{b_3} \end{bmatrix} \begin{bmatrix} 1 & -3 & 3 & -1 \\ 0 & 3 & -6 & 3 \\ 0 & 0 & 3 & -3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ t \\ t^2 \\ t^3 \end{bmatrix}$$

$$\therefore \begin{bmatrix} \mathbf{b_0} & \mathbf{b_1} & \mathbf{b_2} & \mathbf{b_3} \end{bmatrix} = \begin{bmatrix} \mathbf{a_0} & \mathbf{a_1} & \mathbf{a_2} & \mathbf{a_3} \end{bmatrix} \begin{bmatrix} 1 & -3 & 3 & -1 \\ 0 & 3 & -6 & 3 \\ 0 & 0 & 3 & -3 \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1}$$

2.3 Degree Elevation and Reduction of Bezier Curves

(1) Degree Elevation
(2) Repeated Degree Elevation
(3) Degree Reduction

SYstem Design Laboratory



Degree Elevation (1/6)

☑ Objective

- To connect curves with different degree, we have to change the degree of the curves to be same.

Ex) 3^{rd}-degree Bezier curve + 4^{th}-degree Bezier curve
➡ 4^{th}-degree Bezier curve + 4^{th}-degree Bezier curve

- For free curve design by using more control points (Number of Bezier control points = degree+1)

Degree Elevation (2/6)

☑ 2nd-degree Bezier curve ➡ 3rd-degree Bezier curve

$$\mathbf{r}(t) = (1-t)^2\mathbf{b}_0 + 2(1-t)t\mathbf{b}_1 + t^2\mathbf{b}_2 \qquad \times\ [t + (1-t)]$$

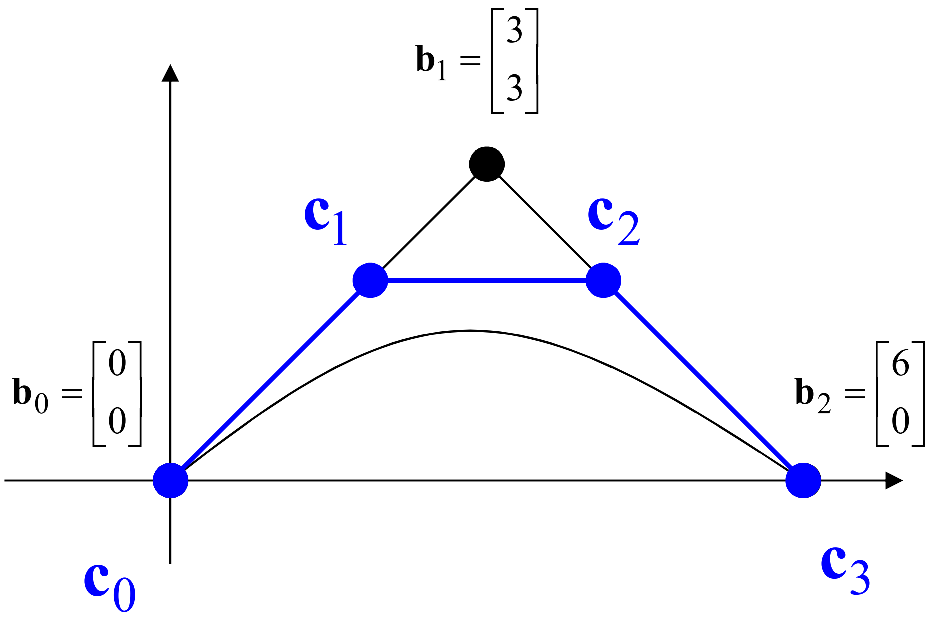
$$\mathbf{r}(t) = [t(1-t)^2 + (1-t)^3]\mathbf{b}_0 + 2[t^2(1-t) + (1-t)^2 t]\mathbf{b}_1 + [t^3 + t^2(1-t)]\mathbf{b}_2$$

$$\mathbf{r}(t) = (1-t)\boxed{} + 3(1-t)^2 t\boxed{} + 3(1-t)t^2\boxed{}] + t^3\boxed{}$$

▭ : New control points

⇨ **The original 2nd-degree Bezier curve may also be written as a 3rd-degree Bezier curve with new control points.**

Degree Elevation (3/6)



$$\mathbf{c}_0 = \mathbf{b}_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

$$\mathbf{c}_1 = [\frac{1}{3}\mathbf{b}_0 + \frac{2}{3}\mathbf{b}_1] = \begin{bmatrix} 2 \\ 2 \end{bmatrix},$$

$$\mathbf{c}_2 = [\frac{2}{3}\mathbf{b}_1 + \frac{1}{3}\mathbf{b}_2] = \begin{bmatrix} 4 \\ 2 \end{bmatrix},$$

$$\mathbf{c}_3 = \mathbf{b}_2 = \begin{bmatrix} 6 \\ 0 \end{bmatrix}$$

2nd-degree Bezier curve

$$\mathbf{r}(t) = (1-t)^2\mathbf{b}_0 + 2(1-t)t\mathbf{b}_1 + t^2\mathbf{b}_2$$

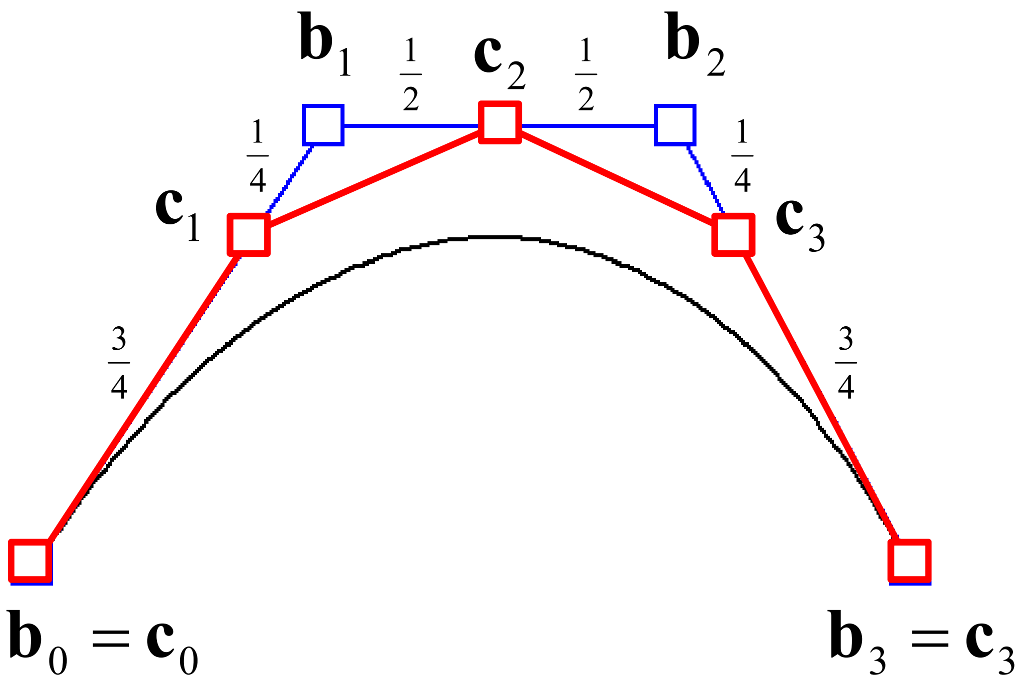
3rd-degree Bezier curve

$$\mathbf{r}(t) = (1-t)^3\mathbf{b}_0 + 3(1-t)^2 t[\frac{1}{3}\mathbf{b}_0 + \frac{2}{3}\mathbf{b}_1] + 3(1-t)t^2[\frac{2}{3}\mathbf{b}_1 + \frac{1}{3}\mathbf{b}_2] + t^3\mathbf{b}_2$$

Degree Elevation (4/6)

☑ Degree elevation of a n-degree Bezier curve with control point $\mathbf{b}_0,\ldots,\mathbf{b}_n$ to a n+1-degree Bezier curve

$$\mathbf{c}_0 = \mathbf{b}_0,$$
$$\vdots$$
$$\mathbf{c}_i = \frac{i}{n+1}\mathbf{b}_{i-1} + \left(1 - \frac{i}{n+1}\right)\mathbf{b}_i,$$
$$\vdots$$
$$\mathbf{c}_{n+1} = \mathbf{b}_n$$



Degree Elevation: 3rd-degree ➡ 4th-degree

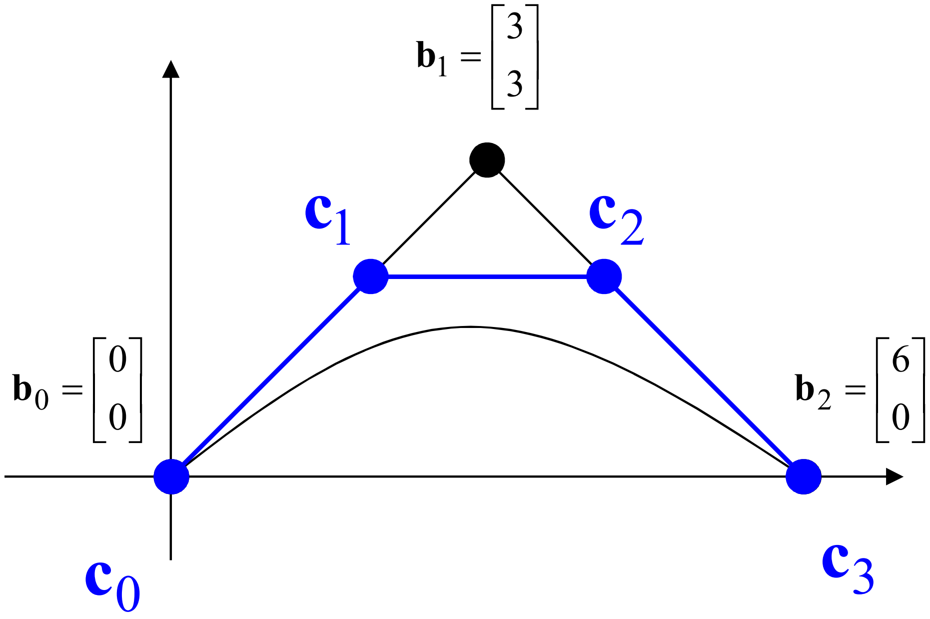
Degree Elevation (5/6)

$$n+2\ \ rows\ \left\{ \overbrace{\begin{bmatrix} 1 & & & & \\ * & * & & & \\ & * & * & & \\ & & \vdots & \vdots & \\ & & & * & * \\ & & & & 1 \end{bmatrix}}^{n+1\ \ colums} \right. \begin{bmatrix} \mathbf{b}_0 \\ \vdots \\ \mathbf{b}_n \end{bmatrix} = \begin{bmatrix} \mathbf{c}_0 \\ \vdots \\ \mathbf{c}_{n+1} \end{bmatrix}$$

$$\mathbf{DB} = \mathbf{C}$$

Degree Elevation (6/6)

Example)



$$\mathbf{c}_0 = \mathbf{b}_0,$$

$$\mathbf{c}_1 = [\frac{1}{3}\mathbf{b}_0 + \frac{2}{3}\mathbf{b}_1],$$

$$\mathbf{c}_2 = [\frac{2}{3}\mathbf{b}_1 + \frac{1}{3}\mathbf{b}_2],$$

$$\mathbf{c}_3 = \mathbf{b}_2$$

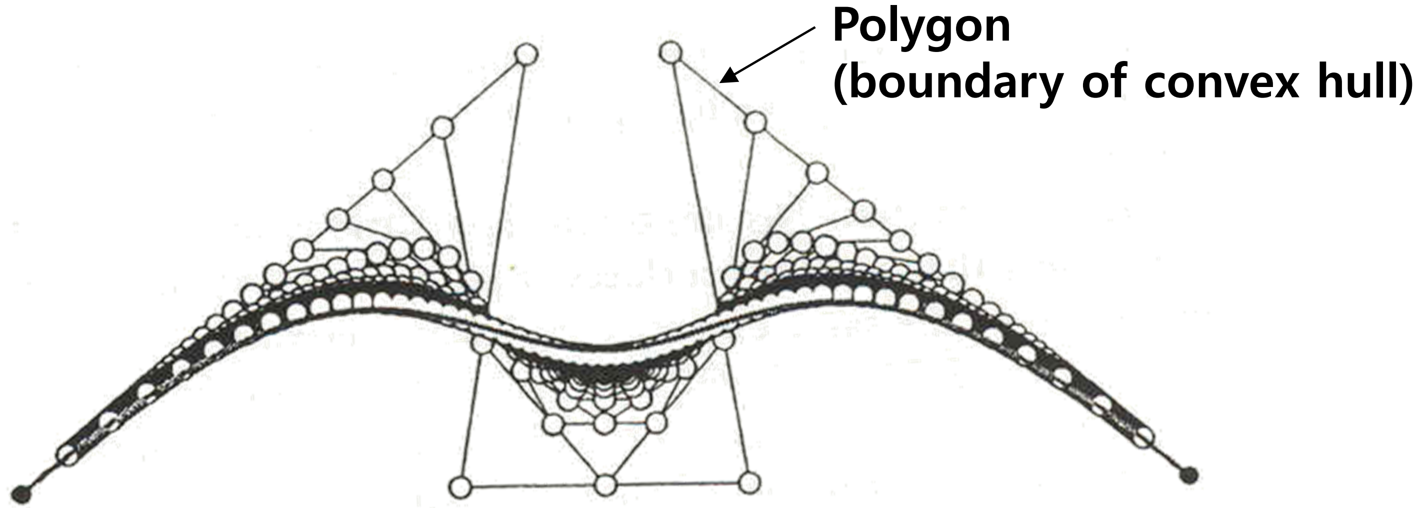
$$\mathbf{DB} = \mathbf{C}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 1/3 & 2/3 & 0 \\ 0 & 2/3 & 1/3 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 3 & 3 \\ 6 & 0 \end{bmatrix} = \mathbf{C}$$

$$\therefore \mathbf{C} = \begin{bmatrix} 0 & 0 \\ 2 & 2 \\ 4 & 2 \\ 6 & 0 \end{bmatrix}$$

Repeated Degree Elevation

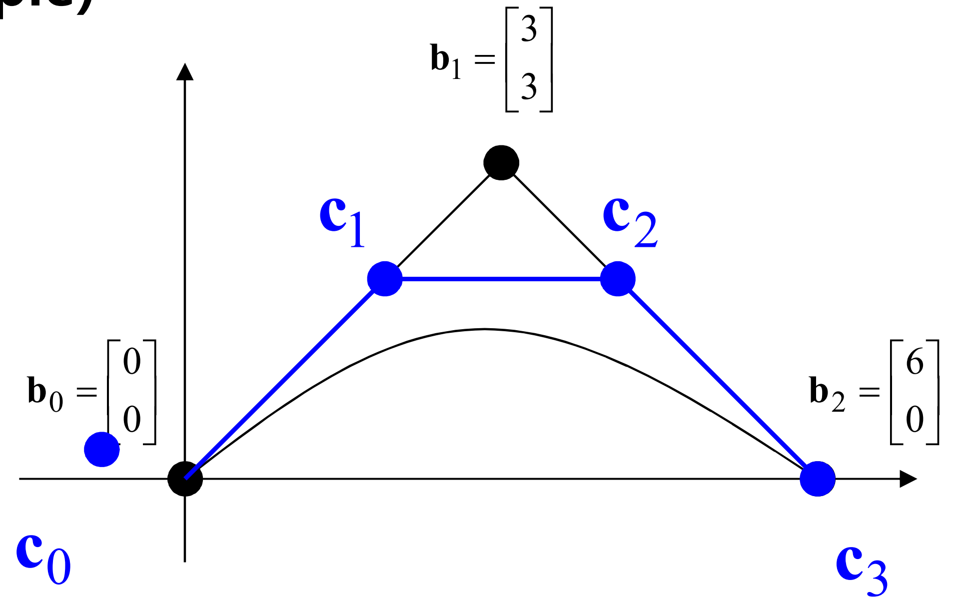
☑ If we repeat the degree elevation, the polygon approaches the curve.



Repeated degree elevation
: a sequence of polygons approaching the curve

Degree Reduction

Example)



$$\mathbf{c}_0 = \mathbf{b}_0,$$
$$\mathbf{c}_1 = [\frac{1}{3}\mathbf{b}_0 + \frac{2}{3}\mathbf{b}_1],$$
$$\mathbf{c}_2 = [\frac{2}{3}\mathbf{b}_1 + \frac{1}{3}\mathbf{b}_2],$$
$$\mathbf{c}_3 = \mathbf{b}_2$$

$$\mathbf{DB} = \mathbf{C}$$

$$\mathbf{D} = \begin{bmatrix} 1 & 0 & 0 \\ 1/3 & 2/3 & 0 \\ 0 & 2/3 & 1/3 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{C} = \begin{bmatrix} 0 & 0 \\ 2 & 2 \\ 4 & 2 \\ 6 & 0 \end{bmatrix}$$

$$\mathbf{D}^T\mathbf{DB} = \mathbf{D}^T\mathbf{C}$$

$$\therefore \mathbf{B} = \left(\mathbf{D}^T\mathbf{D}\right)^{-1}\mathbf{D}^T\mathbf{C}$$

$$\mathbf{D}^T\mathbf{D} = \frac{1}{9}\begin{bmatrix} 10 & 2 & 0 \\ 2 & 8 & 2 \\ 0 & 2 & 10 \end{bmatrix}, \quad \mathbf{D}^T\mathbf{C} = \frac{1}{3}\begin{bmatrix} 2 & 2 \\ 12 & 8 \\ 22 & 2 \end{bmatrix}, \quad \therefore \mathbf{B} = \begin{bmatrix} 0 & 0 \\ 3 & 3 \\ 6 & 0 \end{bmatrix}$$

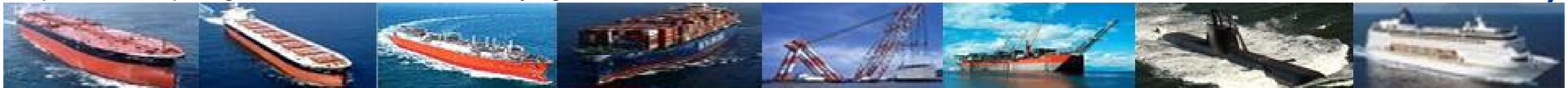
2.4 de Casteljau Algorithm

(1) de Casteljau Algorithm and Bezier Curves

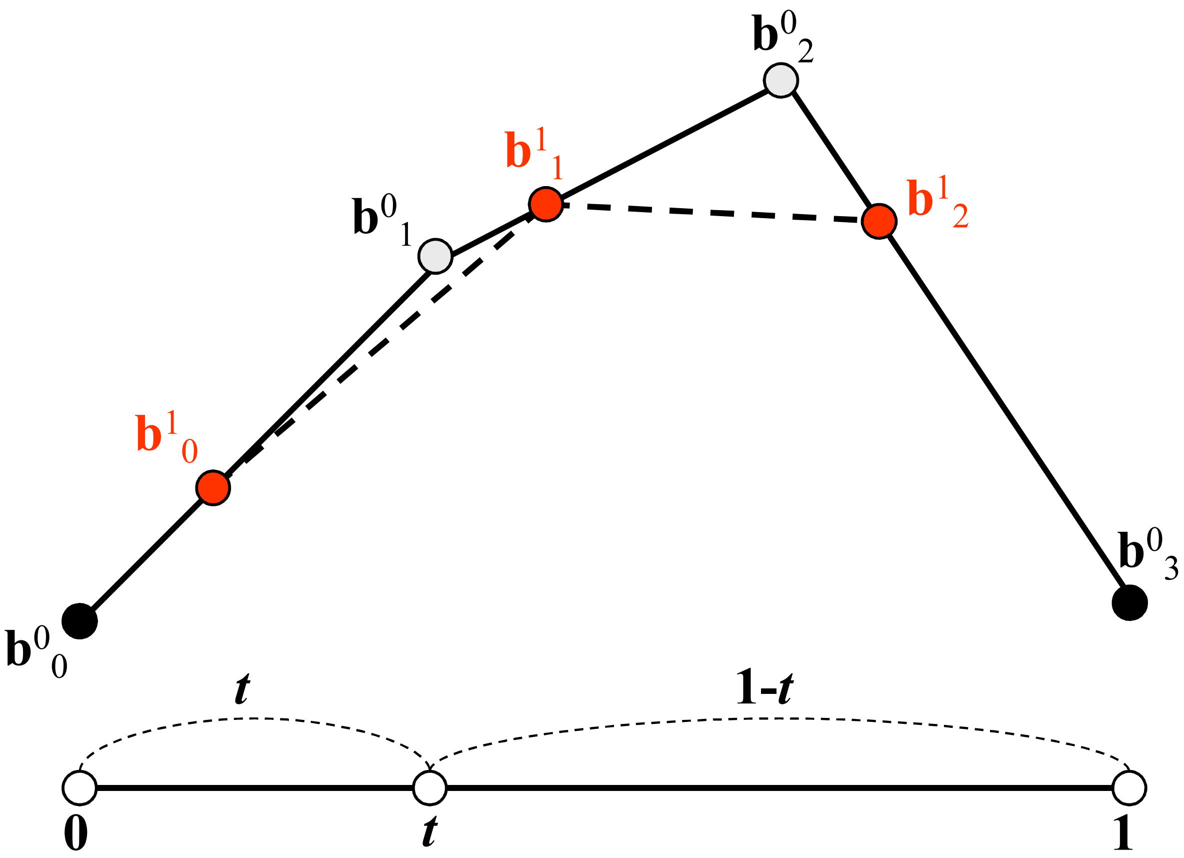
(2) Point on the Bezier Curve

(3) Division into Two Bezier Curves at the Point

(4) Comparison Between the de Casteljau Algorithm and Bezier Curves



de Casteljau Algorithm and Bezier Curves (1/2)



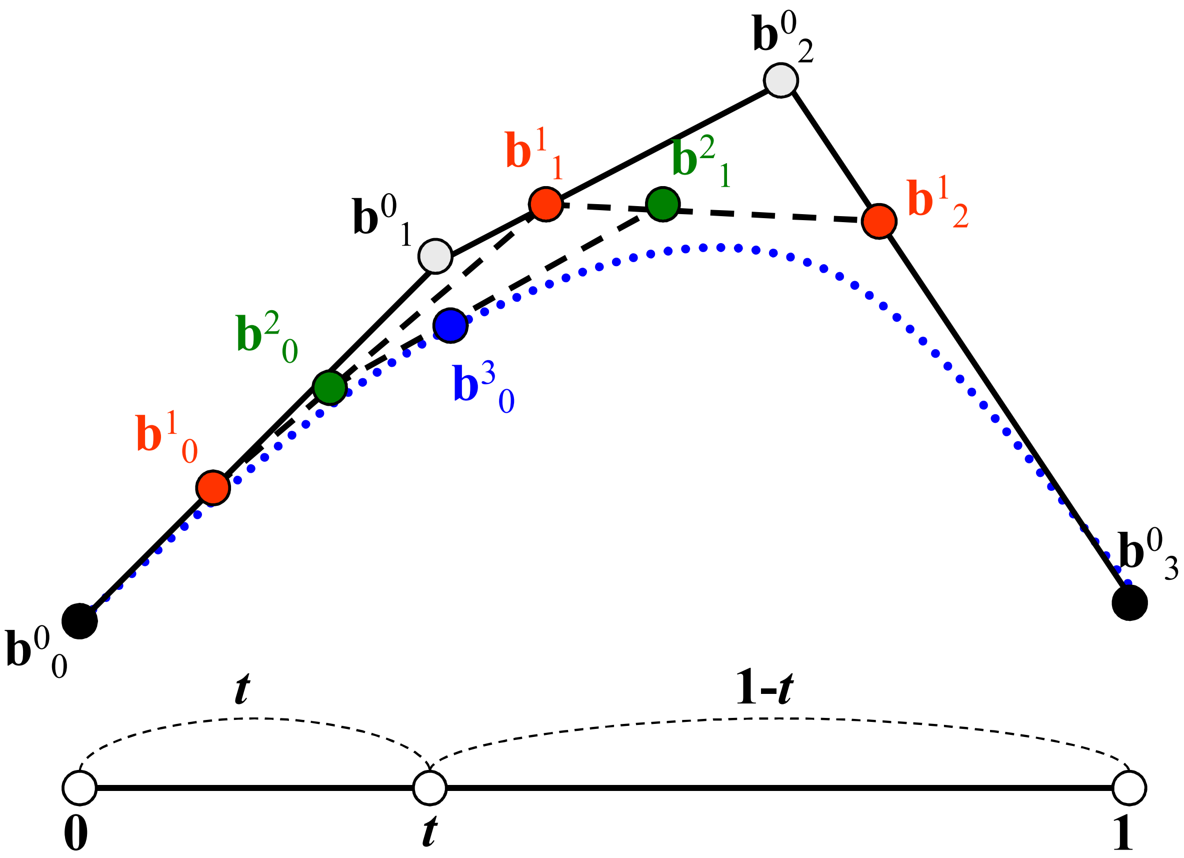
Linear interpolation

$$\mathbf{b}_0^1(t) = (1-t)\mathbf{b}_0^0 + t\mathbf{b}_1^0$$

$$\mathbf{b}_1^1(t) = (1-t)\mathbf{b}_1^0 + t\mathbf{b}_2^0$$

$$\mathbf{b}_2^1(t) = (1-t)\mathbf{b}_2^0 + t\mathbf{b}_3^0$$

de Casteljau Algorithm and Bezier Curves (2/2)



Linear interpolation

$$\mathbf{b}_0^1(t) = (1-t)\mathbf{b}_0^0 + t\mathbf{b}_1^0$$

$$\mathbf{b}_1^1(t) = (1-t)\mathbf{b}_1^0 + t\mathbf{b}_2^0$$

$$\mathbf{b}_2^1(t) = (1-t)\mathbf{b}_2^0 + t\mathbf{b}_3^0$$

$$\mathbf{b}_0^2(t) = (1-t)\mathbf{b}_0^1 + t\mathbf{b}_1^1$$

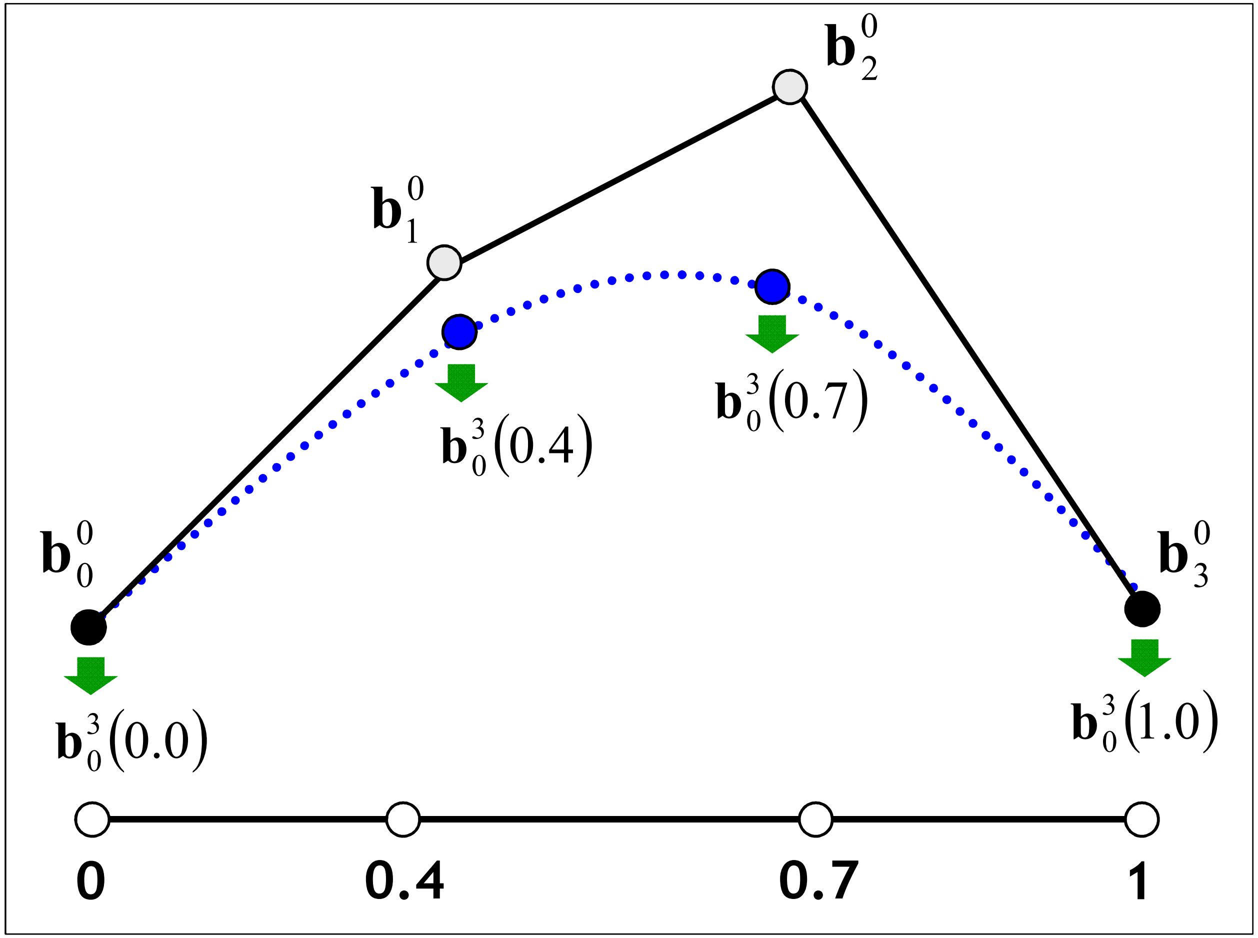
$$\mathbf{b}_1^2(t) = (1-t)\mathbf{b}_1^1 + t\mathbf{b}_2^1$$

$$\mathbf{b}_0^3(t) = (1-t)\mathbf{b}_0^2 + t\mathbf{b}_1^2$$

The same functions with 3rd-degree Bezier curves!!!

$$\mathbf{b}_0^3(t) = (1-t)^3\mathbf{b}_0^0 + 3t(1-t)^2\mathbf{b}_1^0 + 3t^2(1-t)\mathbf{b}_2^0 + t^3\mathbf{b}_3^0$$

Example of de Casteljau Algorithm



Given

$\mathbf{b}_0^0,\ \mathbf{b}_1^0,\ \mathbf{b}_2^0,\ \mathbf{b}_3^0$

Find

Points on Bezier curve at t = 0.0, 0.4, 0.7, 1.0

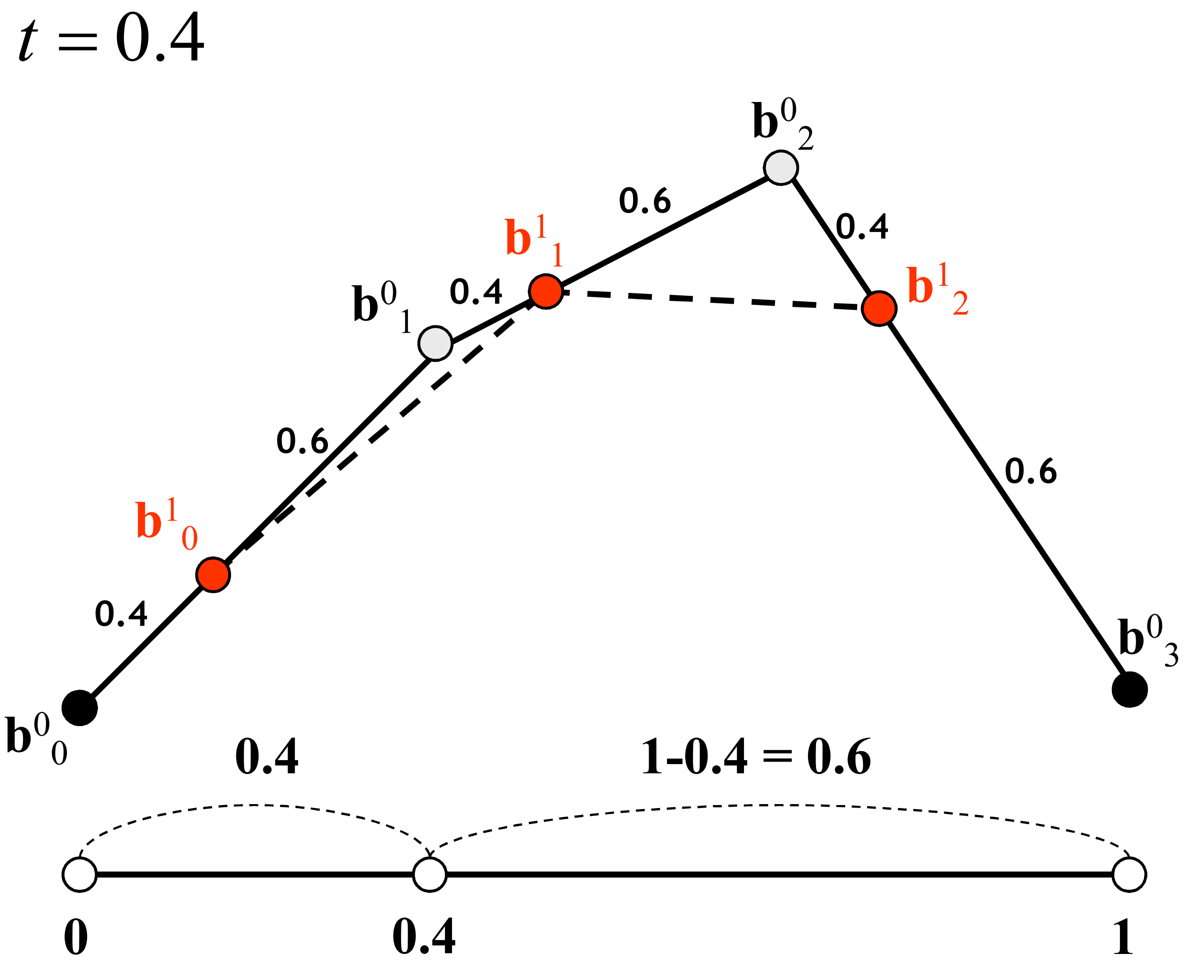
$$\mathbf{b}_0^3(0.0) = (1-0.0)^3\mathbf{b}_0^0 + 3\cdot 0.0(1-0.0)^2\mathbf{b}_1^0 + 3\cdot 0.0^2(1-0.0)\mathbf{b}_2^0 + 0.0^3\mathbf{b}_3^0 = \mathbf{b}_0^0$$

$$\mathbf{b}_0^3(0.4) = (1-0.4)^3\mathbf{b}_0^0 + 3\cdot 0.4(1-0.4)^2\mathbf{b}_1^0 + 3\cdot 0.4^2(1-0.4)\mathbf{b}_2^0 + 0.4^3\mathbf{b}_3^0$$

$$\mathbf{b}_0^3(0.7) = (1-0.7)^3\mathbf{b}_0^0 + 3\cdot 0.7(1-0.7)^2\mathbf{b}_1^0 + 3\cdot 0.7^2(1-0.7)\mathbf{b}_2^0 + 0.7^3\mathbf{b}_3^0$$

$$\mathbf{b}_0^3(0.0) = (1-1.0)^3\mathbf{b}_0^0 + 3\cdot 1.0(1-1.0)^2\mathbf{b}_1^0 + 3\cdot 1.0^2(1-1.0)\mathbf{b}_2^0 + 1.0^3\mathbf{b}_3^0 = \mathbf{b}_3^0$$

Example of de Casteljau Algorithm
- de Casteljau Algorithm at t = 0.4 (1/3)



Linear interpolation

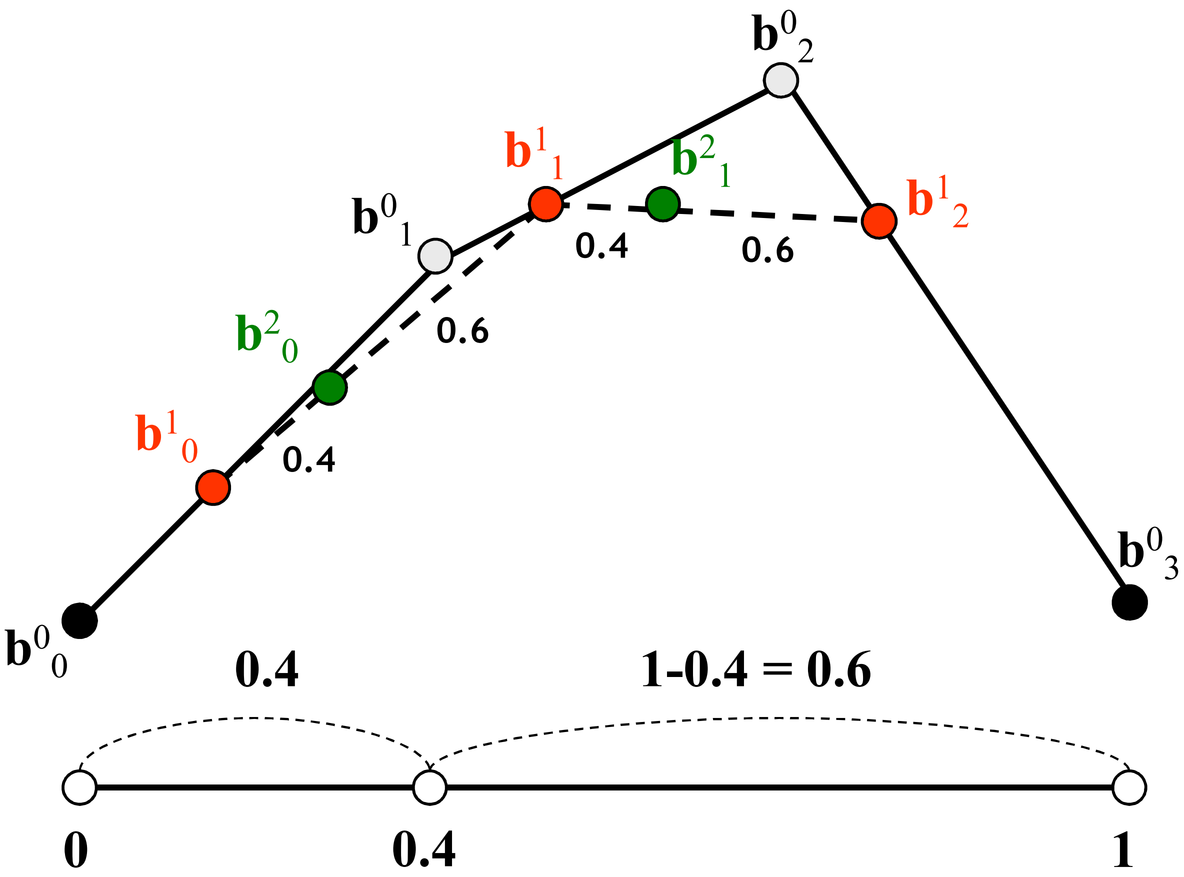
$$\mathbf{b}_0^1(0.4) = (1-0.4)\mathbf{b}_0^0 + 0.4\mathbf{b}_1^0$$

$$\mathbf{b}_1^1(0.4) = (1-0.4)\mathbf{b}_1^0 + 0.4\mathbf{b}_2^0$$

$$\mathbf{b}_2^1(0.4) = (1-0.4)\mathbf{b}_2^0 + 0.4\mathbf{b}_3^0$$

Example of de Casteljau Algorithm
- de Casteljau Algorithm at t = 0.4 (2/3)

$t = 0.4$



Linear interpolation

$$\mathbf{b}_0^1(0.4) = (1-0.4)\mathbf{b}_0^0 + 0.4\mathbf{b}_1^0$$

$$\mathbf{b}_1^1(0.4) = (1-0.4)\mathbf{b}_1^0 + 0.4\mathbf{b}_2^0$$

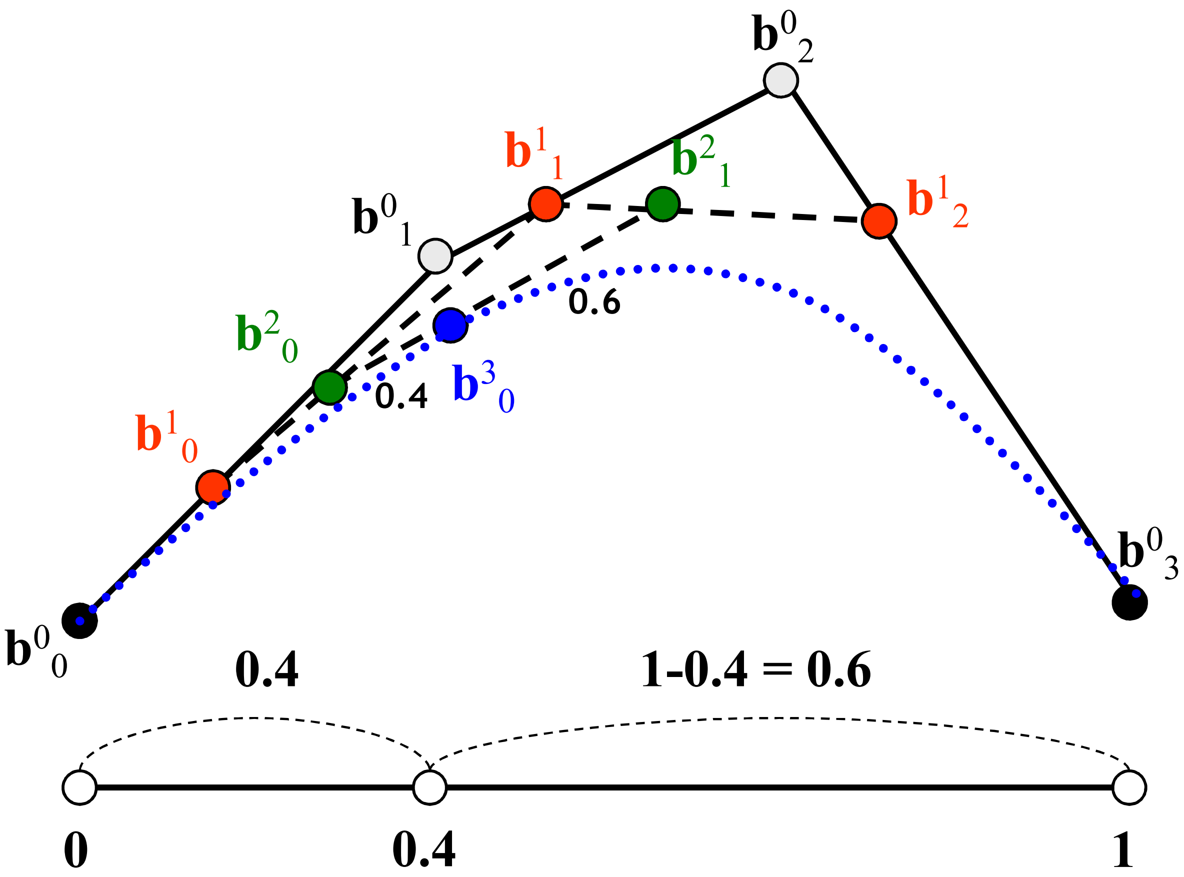
$$\mathbf{b}_2^1(0.4) = (1-0.4)\mathbf{b}_2^0 + 0.4\mathbf{b}_3^0$$

$$\mathbf{b}_0^2(0.4) = (1-0.4)\mathbf{b}_0^1 + 0.4\mathbf{b}_1^1$$

$$\mathbf{b}_1^2(0.4) = (1-0.4)\mathbf{b}_1^1 + 0.4\mathbf{b}_2^1$$

Example of de Casteljau Algorithm
- de Casteljau Algorithm at t = 0.4 (3/3)

$t = 0.4$



Linear interpolation

$$\mathbf{b}_0^1(0.4) = (1-0.4)\mathbf{b}_0^0 + 0.4\mathbf{b}_1^0$$

$$\mathbf{b}_1^1(0.4) = (1-0.4)\mathbf{b}_1^0 + 0.4\mathbf{b}_2^0$$

$$\mathbf{b}_2^1(0.4) = (1-0.4)\mathbf{b}_2^0 + 0.4\mathbf{b}_3^0$$

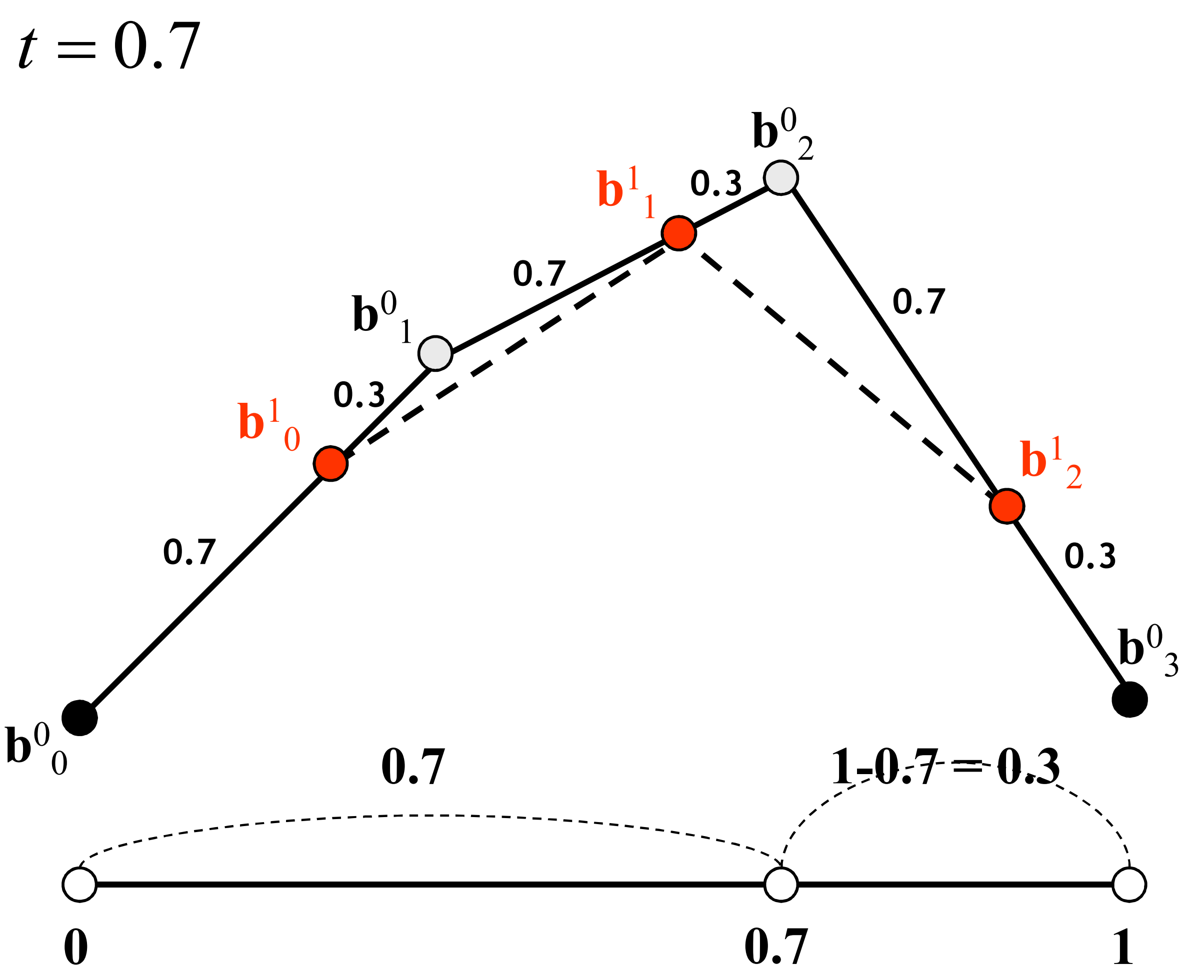
$$\mathbf{b}_0^2(0.4) = (1-0.4)\mathbf{b}_0^1 + 0.4\mathbf{b}_1^1$$

$$\mathbf{b}_1^2(0.4) = (1-0.4)\mathbf{b}_1^1 + 0.4\mathbf{b}_2^1$$

$$\mathbf{b}_0^3(0.4) = (1-0.4)\mathbf{b}_0^2 + 0.4\mathbf{b}_1^2$$

$$\mathbf{b}_0^3(0.4) = (1-0.4)^3\mathbf{b}_0^0 + 3\cdot 0.4(1-0.4)^2\mathbf{b}_1^0 + 3\cdot 0.4^2(1-0.4)\mathbf{b}_2^0 + 0.4^3\mathbf{b}_3^0$$

Example of de Casteljau Algorithm
- de Casteljau Algorithm at t = 0.7 (1/3)



Linear interpolation

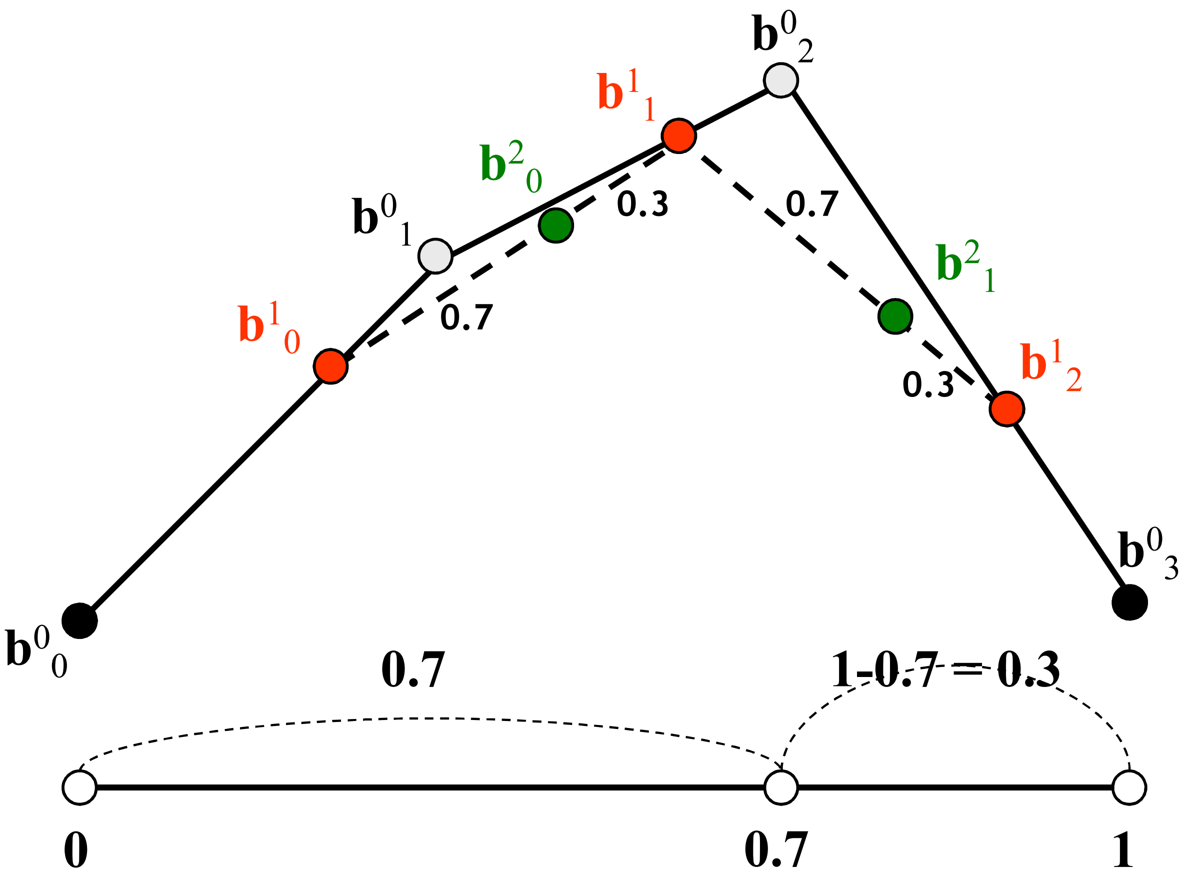
$$\mathbf{b}_0^1(0.7) = (1-0.7)\mathbf{b}_0^0 + 0.7\mathbf{b}_1^0$$

$$\mathbf{b}_1^1(0.7) = (1-0.7)\mathbf{b}_1^0 + 0.7\mathbf{b}_2^0$$

$$\mathbf{b}_2^1(0.7) = (1-0.7)\mathbf{b}_2^0 + 0.7\mathbf{b}_3^0$$

Example of de Casteljau Algorithm
- de Casteljau Algorithm at t = 0.7 (2/3)

$t = 0.7$



Linear interpolation

$$\mathbf{b}_0^1(0.7) = (1-0.7)\mathbf{b}_0^0 + 0.7\mathbf{b}_1^0$$

$$\mathbf{b}_1^1(0.7) = (1-0.7)\mathbf{b}_1^0 + 0.7\mathbf{b}_2^0$$

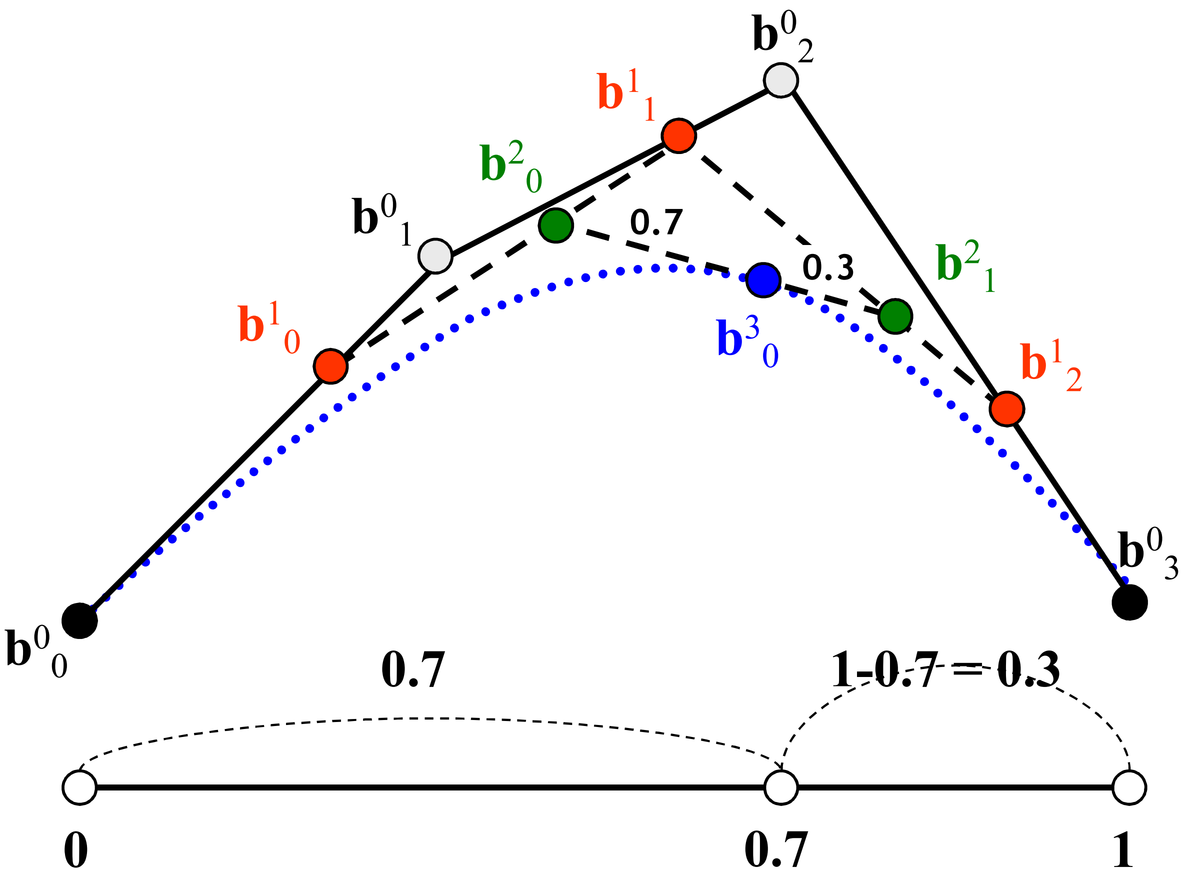
$$\mathbf{b}_2^1(0.7) = (1-0.7)\mathbf{b}_2^0 + 0.7\mathbf{b}_3^0$$

$$\mathbf{b}_0^2(0.7) = (1-0.7)\,\mathbf{b}_0^1 + 0.7\mathbf{b}_1^1$$

$$\mathbf{b}_1^2(0.7) = (1-0.7)\,\mathbf{b}_1^1 + 0.7\mathbf{b}_2^1$$

Example of de Casteljau Algorithm
- de Casteljau Algorithm at t = 0.7 (3/3)

$t = 0.7$



Linear interpolation

$$\mathbf{b}_0^1(0.7) = (1-0.7)\mathbf{b}_0^0 + 0.7\mathbf{b}_1^0$$

$$\mathbf{b}_1^1(0.7) = (1-0.7)\mathbf{b}_1^0 + 0.7\mathbf{b}_2^0$$

$$\mathbf{b}_2^1(0.7) = (1-0.7)\mathbf{b}_2^0 + 0.7\mathbf{b}_3^0$$

$$\mathbf{b}_0^2(0.7) = (1-0.7)\,\mathbf{b}_0^1 + 0.7\mathbf{b}_1^1$$

$$\mathbf{b}_1^2(0.7) = (1-0.7)\,\mathbf{b}_1^1 + 0.7\mathbf{b}_2^1$$

$$\mathbf{b}_0^3(0.7) = (1-0.7)\,\mathbf{b}_0^2 + 0.7\,\mathbf{b}_1^2$$

$$\mathbf{b}_0^3(0.7) = (1-0.7)^3\mathbf{b}_0^0 + 3\cdot 0.7(1-0.7)^2\mathbf{b}_1^0 + 3\cdot 0.7^2(1-0.7)\mathbf{b}_2^0 + 0.7^3\mathbf{b}_3^0$$

[Appendix] Parameter Transformation

☑ The affine map for the interval of $t \in [0,1] \rightarrow u \in [a,b]$,

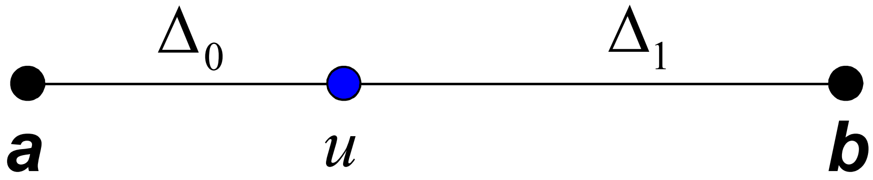
☑ Change the interval of [a, b] to the interval of [0, 1]

$$t = \frac{u-a}{b-a} \quad \text{and} \quad 1-t = \frac{b-u}{b-a}$$

where, u: global parameter, t: local parameter

☑ The process of changing interval is called parameter transformation.

[Appendix] Linear Interpolation on [a, b]



$$u - a : b - u = \Delta_0 : \Delta_1$$

$$\Delta_0 (b - u) = \Delta_1 (u - a)$$

$$(\Delta_0 + \Delta_1) u = \Delta_1 a + \Delta_0 b$$

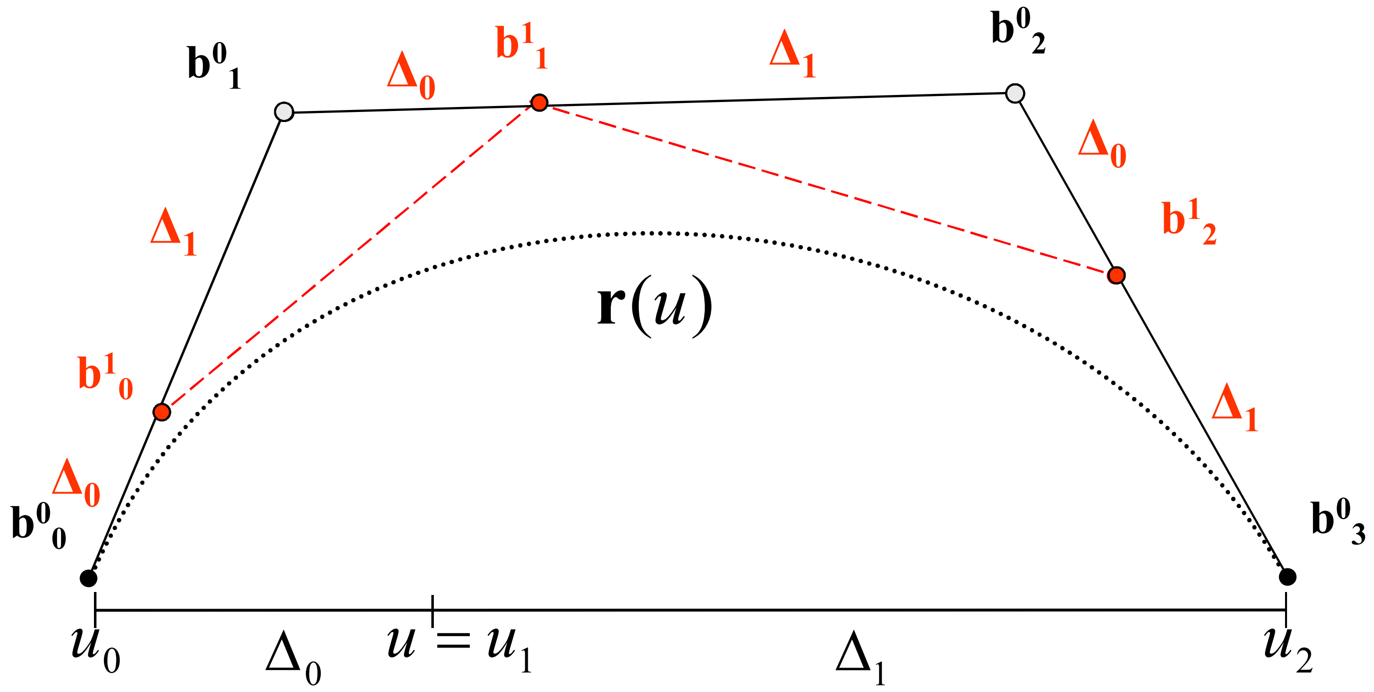
$$u = \frac{\Delta_1 a + \Delta_0 b}{\Delta_0 + \Delta_1}$$

$$\therefore u = \frac{\Delta_1}{\Delta_0 + \Delta_1} a + \frac{\Delta_0}{\Delta_0 + \Delta_1} b$$

$$ratio(a, u, b) = \frac{\Delta_0}{\Delta_1}$$

Point on the Bezier Curve (1/3)

The interval of the parameter u is given by $[u_0, u_2]$. For given four control points, construct the point on the curve at $u = u_1$ by using de Casteljau Algorithm.



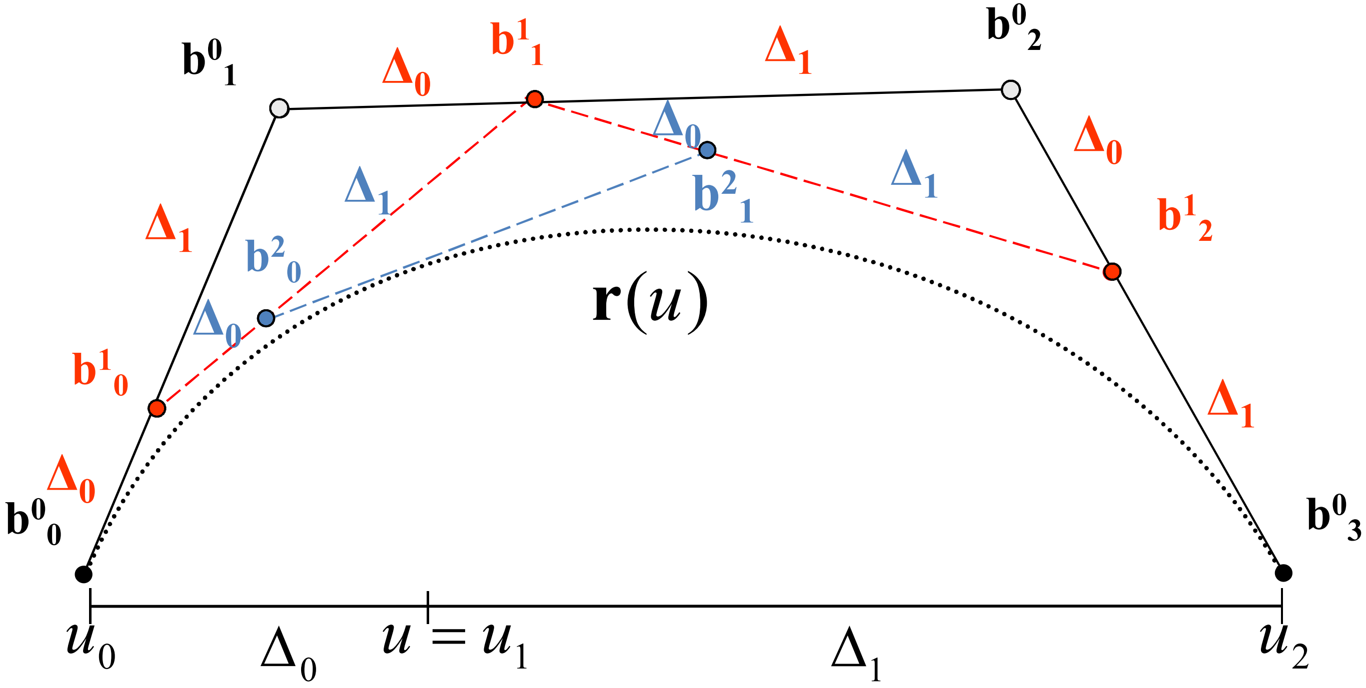
$$\mathbf{b}_1^1(u) = \frac{\Delta_1}{\Delta}\mathbf{b}_1^0 + \frac{\Delta_0}{\Delta}\mathbf{b}_2^0$$

$$\mathbf{b}_2^1(u) = \frac{\Delta_1}{\Delta}\mathbf{b}_2^0 + \frac{\Delta_0}{\Delta}\mathbf{b}_3^0$$

$$\Delta = u_2 - u_0, \quad \Delta_1 = u_2 - u_1, \quad \Delta_0 = u_1 - u_0, \quad \Delta = \Delta_0 + \Delta_1$$

$$\text{ratio}(b_0^2, b_0^3, b_1^2) = \frac{u - u_0}{u_2 - u} = \frac{\Delta_0}{\Delta_1}$$

Point on the Bezier Curve (2/3)



$$\mathbf{b}_0^1(u) = \frac{\Delta_1}{\Delta}\mathbf{b}_0^0 + \frac{\Delta_0}{\Delta}\mathbf{b}_1^0$$

$$\mathbf{b}_1^1(u) = \frac{\Delta_1}{\Delta}\mathbf{b}_1^0 + \frac{\Delta_0}{\Delta}\mathbf{b}_2^0$$

$$\mathbf{b}_2^1(u) = \frac{\Delta_1}{\Delta}\mathbf{b}_2^0 + \frac{\Delta_0}{\Delta}\mathbf{b}_3^0$$

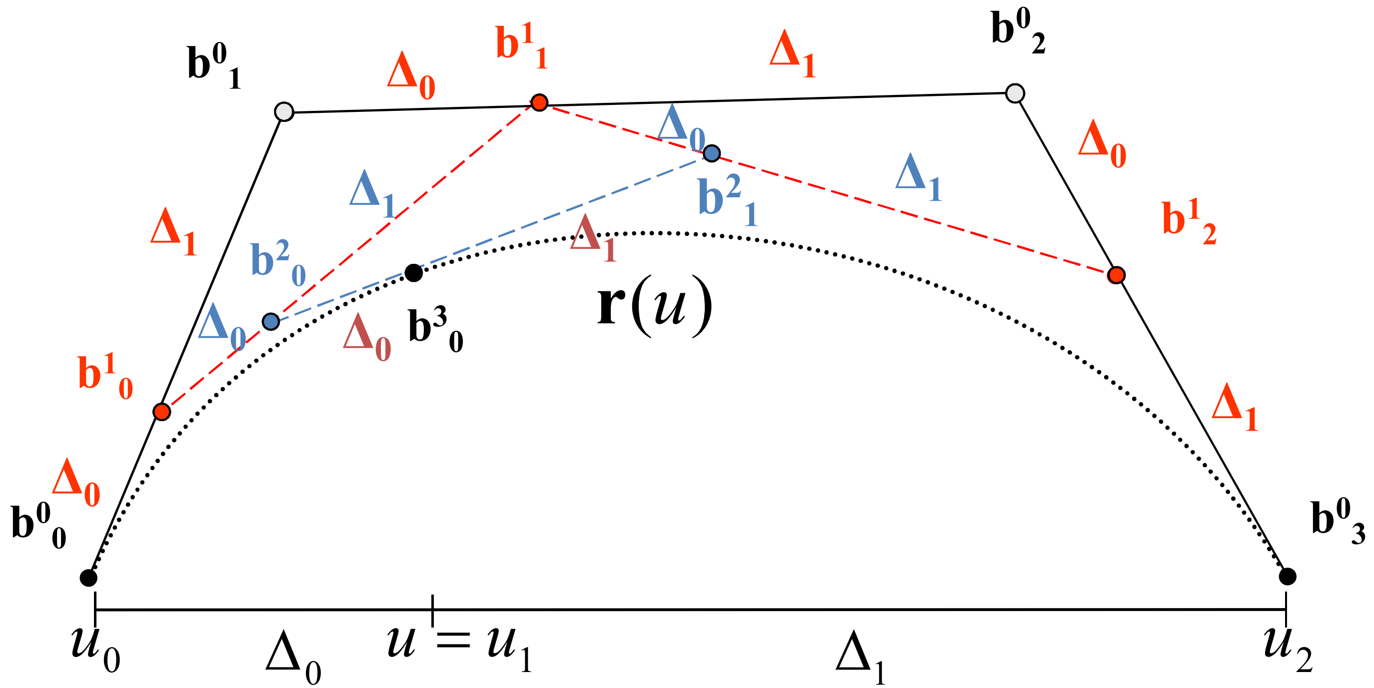
$$\mathbf{b}_0^2(u) = \frac{\Delta_1}{\Delta}\mathbf{b}_0^1 + \frac{\Delta_0}{\Delta}\mathbf{b}_1^1$$

$$\mathbf{b}_1^2(u) = \frac{\Delta_1}{\Delta}\mathbf{b}_1^1 + \frac{\Delta_0}{\Delta}\mathbf{b}_2^1$$

$$\Delta = u_2 - u_0, \quad \Delta_1 = u_2 - u_1, \quad \Delta_0 = u_1 - u_0, \quad \Delta = \Delta_0 + \Delta_1$$

$$\text{ratio}(b_0^2, b_0^3, b_1^2) = \frac{u - u_0}{u_2 - u} = \frac{\Delta_0}{\Delta_1}$$

Point on the Bezier Curve (3/3)



$$\Delta = u_2 - u_0, \quad \Delta_1 = u_2 - u_1, \quad \Delta_0 = u_1 - u_0, \quad \Delta = \Delta_0 + \Delta_1$$

$$\text{ratio}(b_0^2, b_0^3, b_1^2) = \frac{u - u_0}{u_2 - u} = \frac{\Delta_0}{\Delta_1}$$

$$\mathbf{b}_0^1(u) = \frac{u_2 - u}{u_2 - u_0}\mathbf{b}_0^0 + \frac{u - u_0}{u_2 - u_0}\mathbf{b}_1^0 = \frac{\Delta_1}{\Delta}\mathbf{b}_0^0 + \frac{\Delta_0}{\Delta}\mathbf{b}_1^0$$

$$\mathbf{b}_1^1(u) = \frac{\Delta_1}{\Delta}\mathbf{b}_1^0 + \frac{\Delta_0}{\Delta}\mathbf{b}_2^0$$

$$\mathbf{b}_2^1(u) = \frac{\Delta_1}{\Delta}\mathbf{b}_2^0 + \frac{\Delta_0}{\Delta}\mathbf{b}_3^0$$

$$\mathbf{b}_0^2(u) = \frac{\Delta_1}{\Delta}\mathbf{b}_0^1 + \frac{\Delta_0}{\Delta}\mathbf{b}_1^1$$

$$\mathbf{b}_1^2(u) = \frac{\Delta_1}{\Delta}\mathbf{b}_1^1 + \frac{\Delta_0}{\Delta}\mathbf{b}_2^1$$

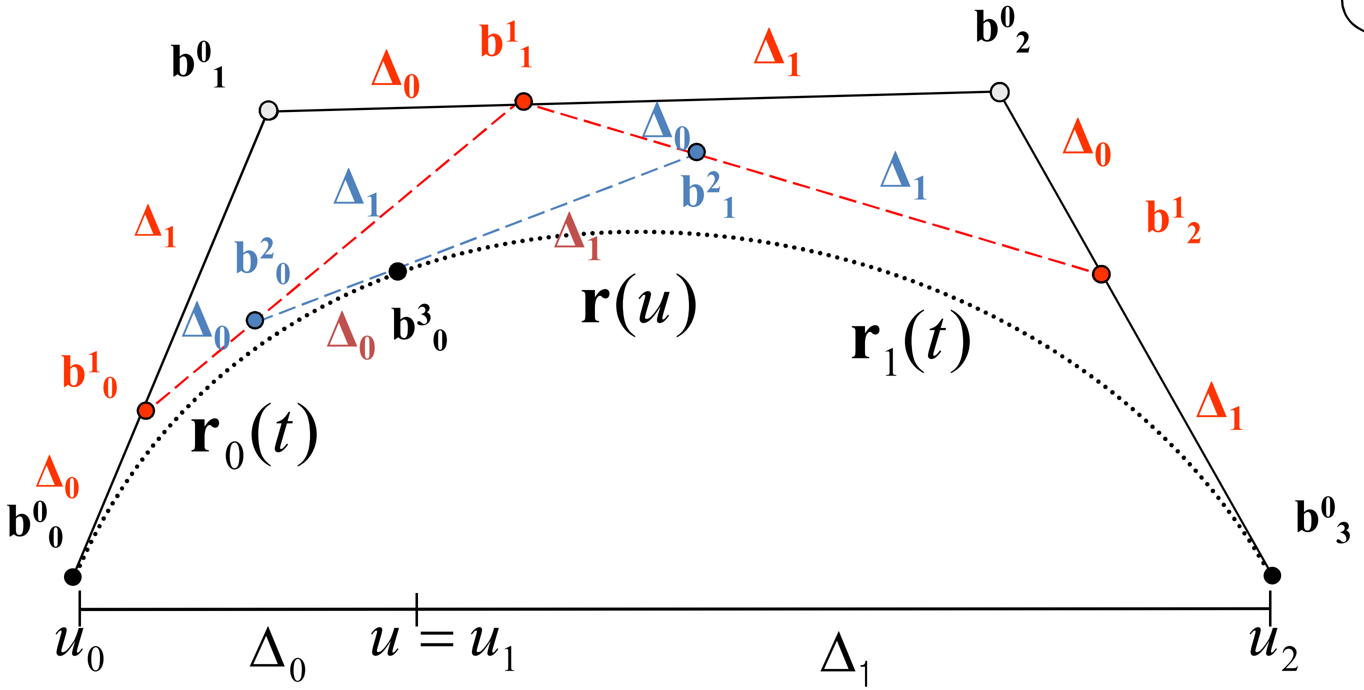
$$\mathbf{b}_0^3(u) = \frac{\Delta_1}{\Delta}\mathbf{b}_0^2 + \frac{\Delta_0}{\Delta}\mathbf{b}_1^2$$

Let $t = \dfrac{u - u_0}{u_2 - u_0}$

Identical with 3rd degree Bezier curves!!!

$$\mathbf{b}_0^3(u) = \mathbf{r}(t) = (1-t)^3\mathbf{b}_0 + 3(1-t)^2 t\mathbf{b}_1 + 3(1-t)t^2\mathbf{b}_2 + t^3\mathbf{b}_3$$

Division into Two Bezier Curves at the Point



$\Delta = u_2 - u_0$, $\Delta_1 = u_2 - u_1$, $\Delta_0 = u_1 - u_0$, $\Delta = \Delta_0 + \Delta_1$

$$ratio(b_0^2, b_0^3, b_1^2) = \frac{u - u_0}{u_2 - u} = \frac{\Delta_0}{\Delta_1}$$

① (1) The evaluation of a point on the Bezier curve at $u = u_1$ generates two sets of Bezier control points.

(2) Bezier control points $\mathbf{b}^0{}_0$, $\mathbf{b}^1{}_0$, $\mathbf{b}^2{}_0$, $\mathbf{b}^3{}_0$, with parameter interval of Δ_0 represent the left curve $\mathbf{r}_0(t)$, and Bezier control points $\mathbf{b}^3{}_0$, $\mathbf{b}^2{}_1$, $\mathbf{b}^1{}_2$, $\mathbf{b}^0{}_3$ with parameter interval of Δ_1 represent the right curve $\mathbf{r}_1(t)$.

②
$$\mathbf{r}(t) = (1-t)^3\mathbf{b}_0^0 + 3(1-t)^2 t\mathbf{b}_1^0 + 3(1-t)t^2\mathbf{b}_2^0 + t^3\mathbf{b}_3^0,\ t = \frac{u - u_0}{u_2 - u_0}$$

$$\mathbf{r}_0(t) = (1-t)^3\mathbf{b}_0^0 + 3(1-t)^2 t\mathbf{b}_0^1 + 3(1-t)t^2\mathbf{b}_0^2 + t^3\mathbf{b}_0^3,\ t = \frac{u - u_0}{u_1 - u_0}$$

$$\mathbf{r}_1(t) = (1-t)^3\mathbf{b}_0^3 + 3(1-t)^2 t\mathbf{b}_1^2 + 3(1-t)t^2\mathbf{b}_2^1 + t^3\mathbf{b}_3^0,\ t = \frac{u - u_1}{u_2 - u_1}$$

Comparison Between the de Casteljau Algorithm and Bezier Curves

☑ de Casteljau algorithm: "Constructive Approach"

- Input: $\mathbf{b}_i$ (Bezier control points)
- Processor: Sequentially n-times "linear interpolation"
- Output: Point on the n^{th}-degree Bezier curve

☑ Bezier curve: "Bernstein Function Evaluation Approach"

- Input: $\mathbf{b}_i$ (Bezier control points)
- Processor: Curve by "blending" the control points($\mathbf{b}_i$) and Bernstein basis functions
- Output: the n^{th}-degree Bezier curve

2.5 Bezier Curve Interpolation and Approximation

(1) Introduction to Curve Interpolation

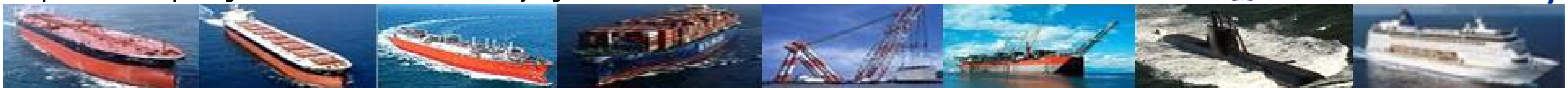
(2) Cubic Bezier Curve Interpolation

(3) Bezier Curve Interpolation Beyond Cubics

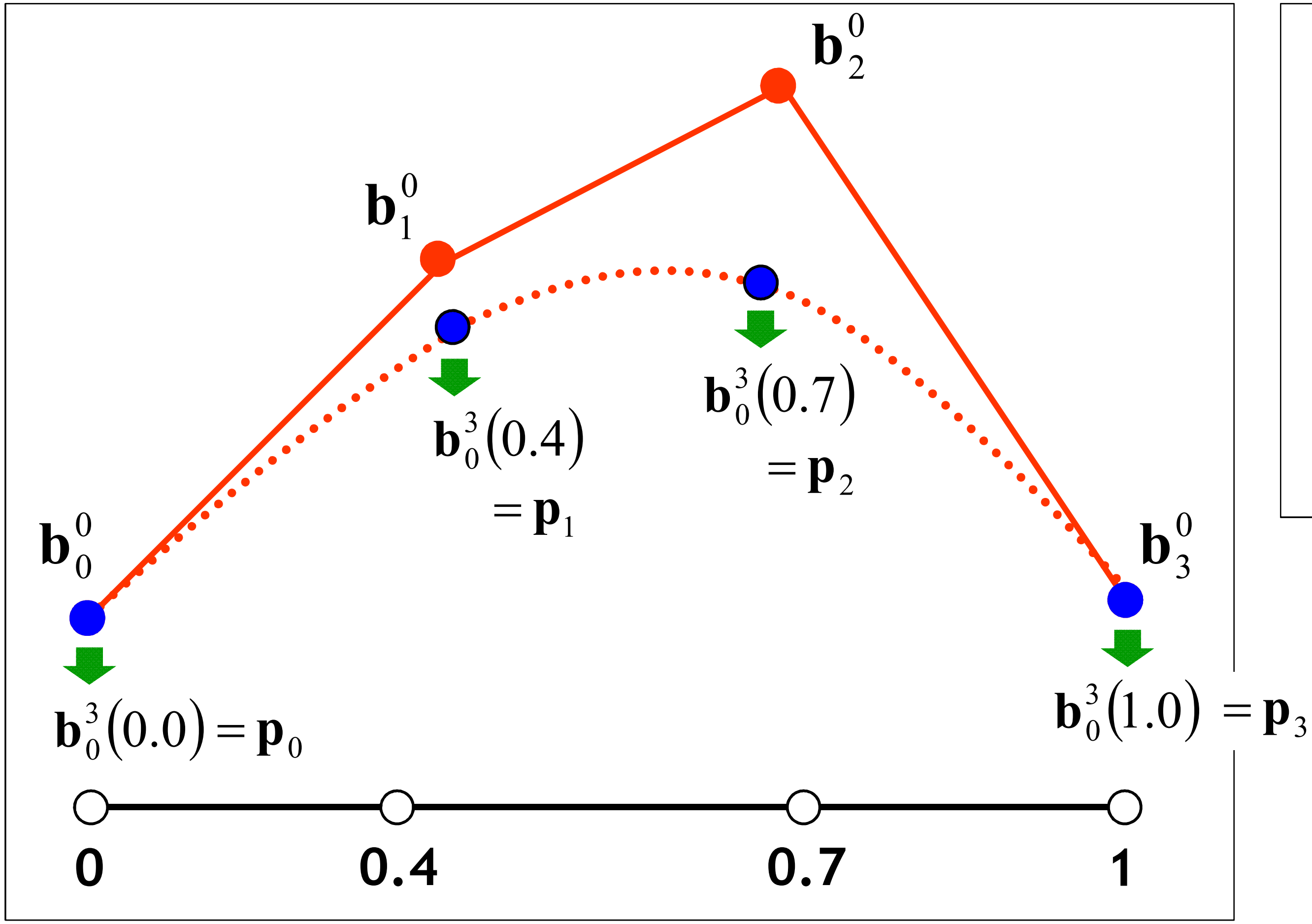
(4) Bezier Curve Approximation

(5) Chord Length Parameters

SYstem Design Laboratory



Points on the Cubic Bezier Curve at Parameter *t*



Given

$\mathbf{b}_0^0,\ \mathbf{b}_1^0,\ \mathbf{b}_2^0,\ \mathbf{b}_3^0$

Find

Points on the Bezier curve at t = 0.0, 0.4, 0.7, 1.0

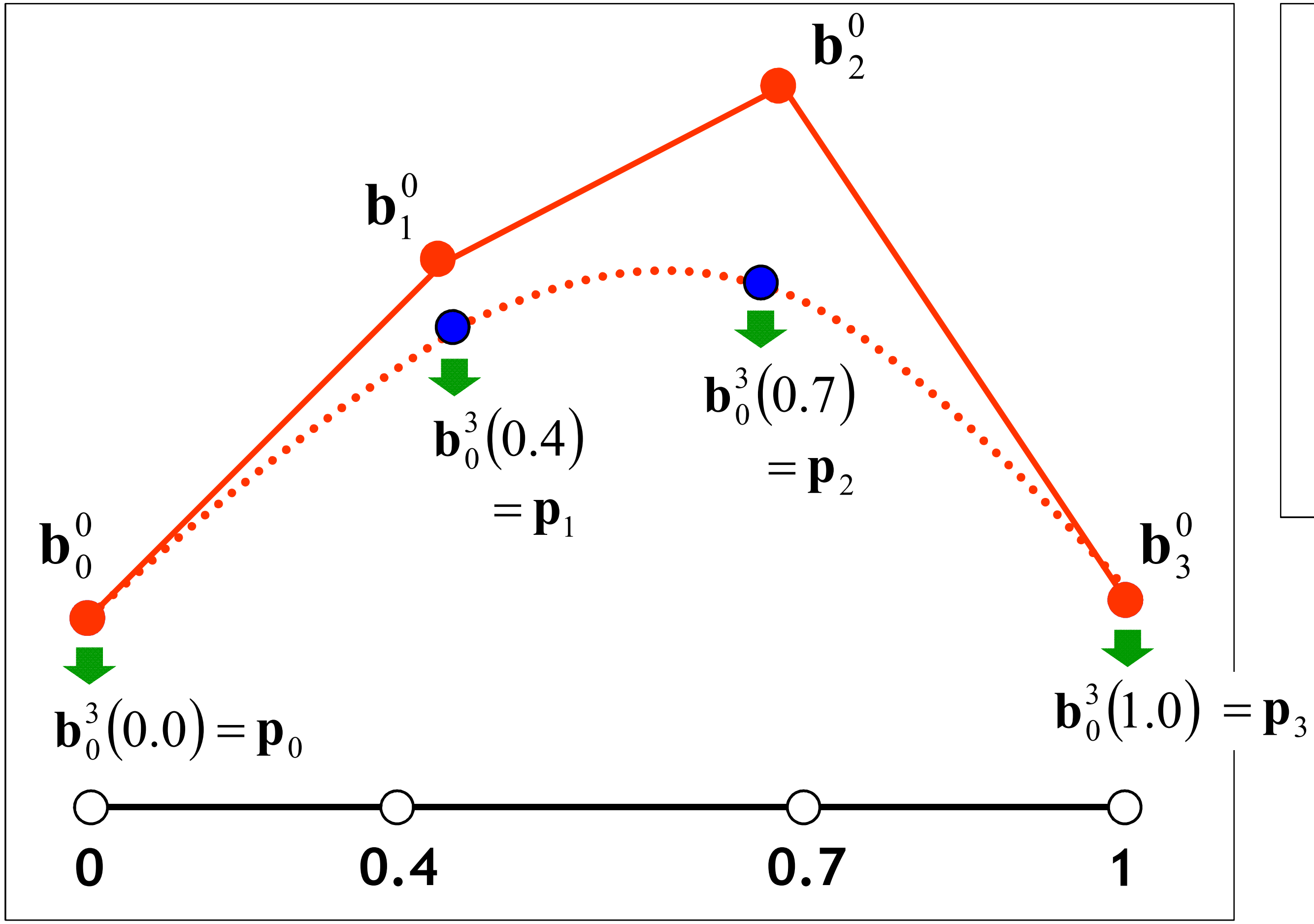
$$\mathbf{b}_0^3(0.0) = (1-0.0)^3\mathbf{b}_0^0 + 3\cdot 0.0(1-0.0)^2\mathbf{b}_1^0 + 3\cdot 0.0^2(1-0.0)\mathbf{b}_2^0 + 0.0^3\mathbf{b}_3^0 = \mathbf{b}_0^0$$

$$\mathbf{b}_0^3(0.4) = (1-0.4)^3\mathbf{b}_0^0 + 3\cdot 0.4(1-0.4)^2\mathbf{b}_1^0 + 3\cdot 0.4^2(1-0.4)\mathbf{b}_2^0 + 0.4^3\mathbf{b}_3^0$$

$$\mathbf{b}_0^3(0.7) = (1-0.7)^3\mathbf{b}_0^0 + 3\cdot 0.7(1-0.7)^2\mathbf{b}_1^0 + 3\cdot 0.7^2(1-0.7)\mathbf{b}_2^0 + 0.7^3\mathbf{b}_3^0$$

$$\mathbf{b}_0^3(0.0) = (1-1.0)^3\mathbf{b}_0^0 + 3\cdot 1.0(1-1.0)^2\mathbf{b}_1^0 + 3\cdot 1.0^2(1-1.0)\mathbf{b}_2^0 + 1.0^3\mathbf{b}_3^0 = \mathbf{b}_3^0$$

Curve Interpolation



Given

Points on the Bezier curve at t = 0.0, 0.4, 0.7, 1.0 $(\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3)$

Find: Cubic Bezier Curve

$\mathbf{b}_0^0,\ \mathbf{b}_1^0,\ \mathbf{b}_2^0,\ \mathbf{b}_3^0$

If we have given fitting points P_i and we wish to pass a curve through them, called "curve interpolation".

We may choose among many kinds of curves.

If we use a cubic Bezier curve as an interpolation curve,

➡ "cubic Bezier curve interpolation"

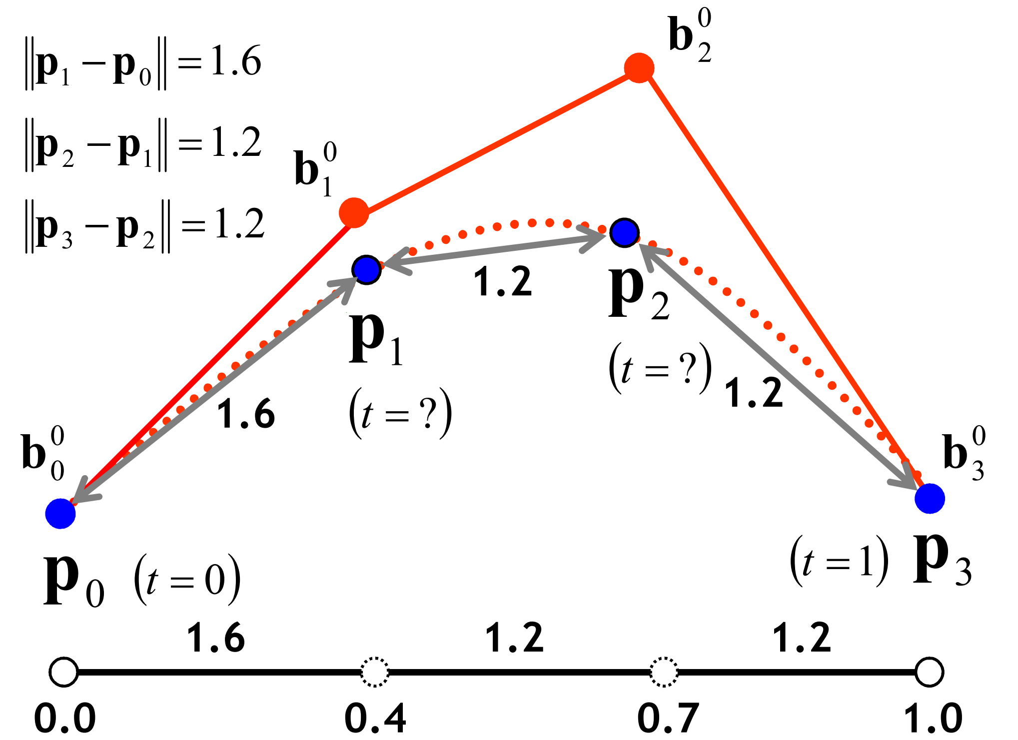
$$\mathbf{b}_0^3(0.0) = (1-0.0)^3\mathbf{b}_0^0 + 3\cdot 0.0(1-0.0)^2\mathbf{b}_1^0 + 3\cdot 0.0^2(1-0.0)\mathbf{b}_2^0 + 0.0^3\mathbf{b}_3^0 = \mathbf{b}_0^0$$

$$\mathbf{b}_0^3(0.4) = (1-0.4)^3\mathbf{b}_0^0 + 3\cdot 0.4(1-0.4)^2\mathbf{b}_1^0 + 3\cdot 0.4^2(1-0.4)\mathbf{b}_2^0 + 0.4^3\mathbf{b}_3^0$$

$$\mathbf{b}_0^3(0.7) = (1-0.7)^3\mathbf{b}_0^0 + 3\cdot 0.7(1-0.7)^2\mathbf{b}_1^0 + 3\cdot 0.7^2(1-0.7)\mathbf{b}_2^0 + 0.7^3\mathbf{b}_3^0$$

$$\mathbf{b}_0^3(0.0) = (1-1.0)^3\mathbf{b}_0^0 + 3\cdot 1.0(1-1.0)^2\mathbf{b}_1^0 + 3\cdot 1.0^2(1-1.0)\mathbf{b}_2^0 + 1.0^3\mathbf{b}_3^0 = \mathbf{b}_3^0$$

Set of Parameter Using Chord Length



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

Given

Points on the Bezier curve
at t = 0.0, 0.4, 0.7, 1.0
$(\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3)$

Find

$\mathbf{b}_0^0,\ \mathbf{b}_1^0,\ \mathbf{b}_2^0,\ \mathbf{b}_3^0$

- Every point on a Bezier curve has a parameter value t; in order to solve interpolation problem, we have to assign a parameter value t_i to every point $\mathbf{P}_i$.

$$0 = t_0 < t_1 < t_2 < t_3 = 1$$

- A natural choice is to associate the ratio of distances between each $\mathbf{P}_i$.

$$t_0 = 0.0,\quad t_3 = 1.0$$

$$t_1 = \frac{1.6}{1.6+1.2+1.2} = 0.4$$

$$t_2 = \frac{1.6+1.2}{1.6+1.2+1.2} = 0.7$$

Set parameter t using chord length

- Then, we want a cubic Bezier curve such that:

$$\mathbf{r}(t_i) = \mathbf{p}_i;\quad i = 0, 1, 2, 3$$

Cubic Bezier Curve Interpolation (1/3)

☑ The cubic Bezier curve of the form is defined by

$$\mathbf{r}(t) = B_0^3(t)\mathbf{b}_0 + B_1^3(t)\mathbf{b}_1 + B_2^3(t)\mathbf{b}_2 + B_3^3(t)\mathbf{b}_3.$$

☑ All interpolation conditions are

$$\mathbf{p}_0 = B_0^3(t_0)\mathbf{b}_0 + B_1^3(t_0)\mathbf{b}_1 + B_2^3(t_0)\mathbf{b}_2 + B_3^3(t_0)\mathbf{b}_3,$$
$$\mathbf{p}_1 = B_0^3(t_1)\mathbf{b}_0 + B_1^3(t_1)\mathbf{b}_1 + B_2^3(t_1)\mathbf{b}_2 + B_3^3(t_1)\mathbf{b}_3,$$
$$\mathbf{p}_2 = B_0^3(t_2)\mathbf{b}_0 + B_1^3(t_2)\mathbf{b}_1 + B_2^3(t_2)\mathbf{b}_2 + B_3^3(t_2)\mathbf{b}_3,$$
$$\mathbf{p}_3 = B_0^3(t_3)\mathbf{b}_0 + B_1^3(t_3)\mathbf{b}_1 + B_2^3(t_3)\mathbf{b}_2 + B_3^3(t_3)\mathbf{b}_3.$$

➡ 4 Unknown Vectors($\mathbf{b}_0$, $\mathbf{b}_1$, $\mathbf{b}_2$, $\mathbf{b}_3$) and 4 Vector Equations

➡ "Determinate Problem"

Cubic Bezier Curve Interpolation (2/3)

- ☑ To find the solution of these four equations for four unknowns, we can write in matrix form as below.

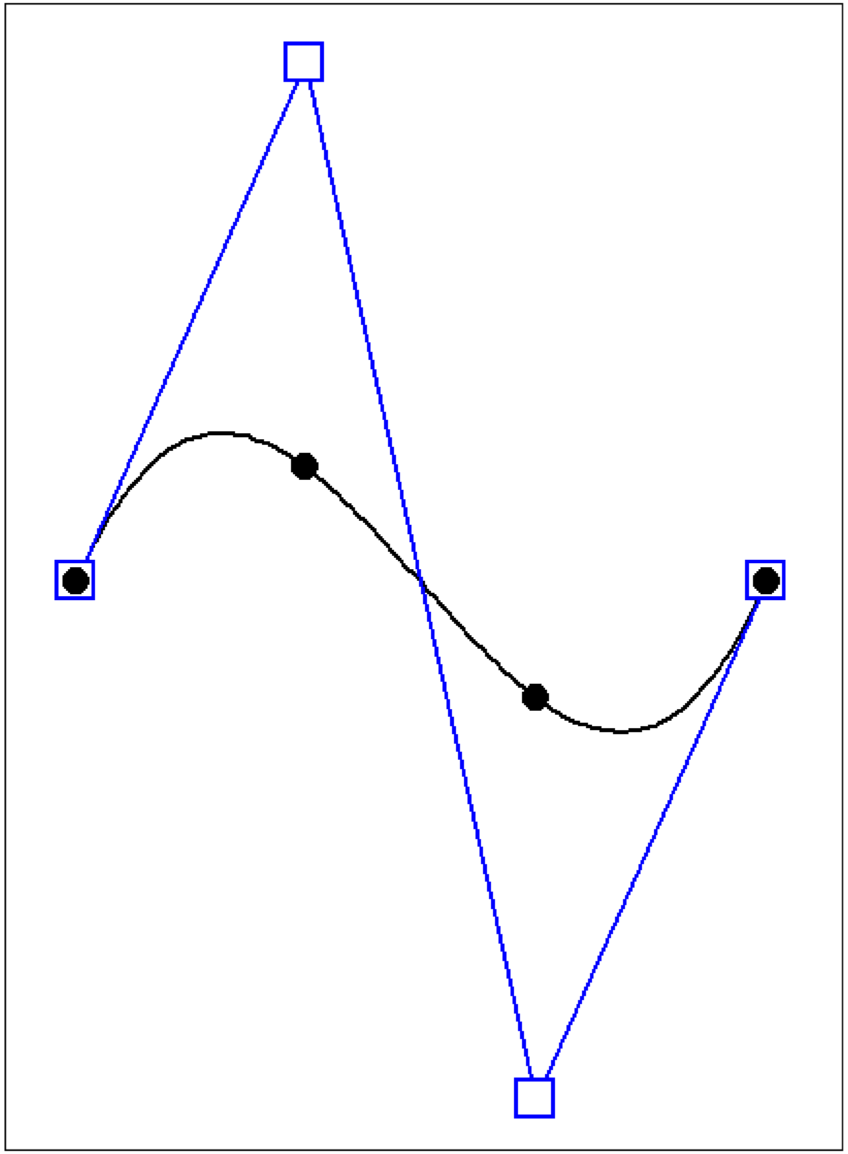
$$\begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{bmatrix} = \begin{bmatrix} B_0^3(t_0) & B_1^3(t_0) & B_2^3(t_0) & B_3^3(t_0) \\ B_0^3(t_1) & B_1^3(t_1) & B_2^3(t_1) & B_3^3(t_1) \\ B_0^3(t_2) & B_1^3(t_2) & B_2^3(t_2) & B_3^3(t_2) \\ B_0^3(t_3) & B_1^3(t_3) & B_2^3(t_3) & B_3^3(t_3) \end{bmatrix} \begin{bmatrix} \mathbf{b}_0 \\ \mathbf{b}_1 \\ \mathbf{b}_2 \\ \mathbf{b}_3 \end{bmatrix}$$

- ☑ To abbreviate the above form is $\mathbf{P} = \mathbf{MB}$.
- ☑ Then, the solution is $\mathbf{B} = \mathbf{M}^{-1}\mathbf{P}$.
- ☑ Although it looks like the solution to one linear system but it is the two or three systems depending on the dimensionality of the $\mathbf{P}_\mathbf{i}$.

$$ex)\ \mathbf{p}_0 = \begin{bmatrix} x_0 & y_0 \end{bmatrix}^T \ or\ \begin{bmatrix} x_0 & y_0 & z_0 \end{bmatrix}^T$$

Cubic Bezier Curve Interpolation (3/3)

Cubic Bezier Interpolation



Bezier Curve Interpolation Beyond Cubics (1/3)

- ☑ Polynomial interpolation can also works for more than four data points.

- ☑ Given: Points $\mathbf{p}_0, \cdots, \mathbf{p}_m$ and corresponding parameter values $0 = t_0 < t_1 < \ldots < t_{m-1} < t_m = 1.$

- ☑ If we choose a Bezier curve of degree n for interpolation, we have "m+1 vector equations" for "n+1 unknown vectors".

- ☑ $n > m$: underdetermined system. We need *additional conditions* to solve the interpolation problem.

- ☑ $n = m$: determinate linear system ➡ "Interpolation problem"

- ☑ $n < m$: over-determined system ➡ "Approximation problem"

Bezier Curve Interpolation Beyond Cubics (2/3)

☑ Given: Points $\mathbf{p}_0, \ldots, \mathbf{p}_m$ and
corresponding parameter values $0 = t_0 < t_1 < \ldots < t_{m-1} < t_m = 1.$

☑ If we use a Bezier curve of degree n $(= m)$,
we have a linear system: $\mathbf{P} = \mathbf{MB}$.

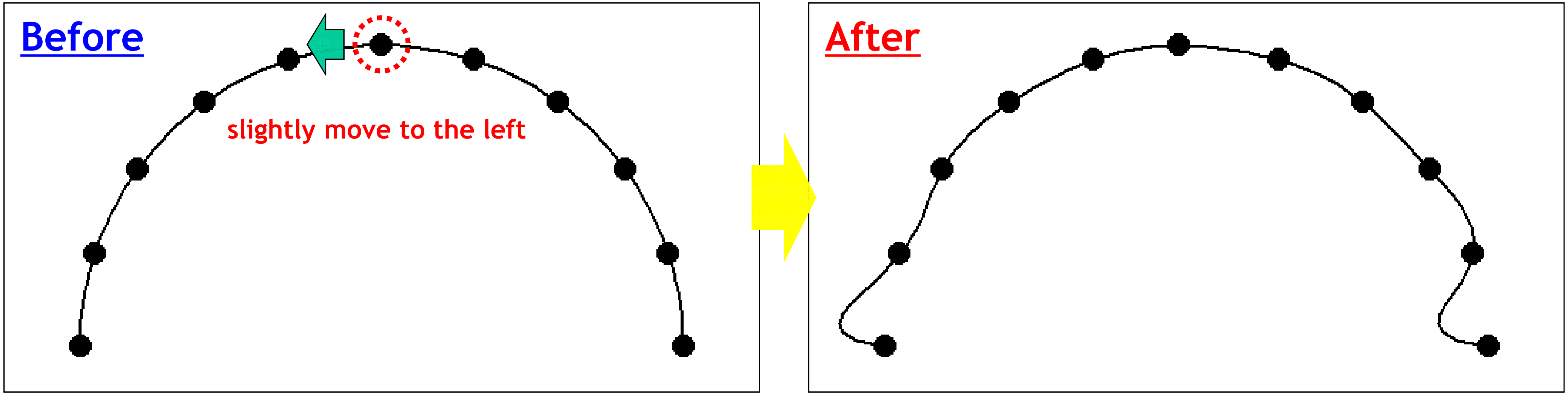
$$\begin{bmatrix} B_0^n(t_0) & \cdots & B_n^n(t_0) \\ \vdots & & \vdots \\ B_0^n(t_m) & & B_n^n(t_m) \end{bmatrix} \begin{bmatrix} \mathbf{b}_0 \\ \vdots \\ \mathbf{b}_n \end{bmatrix} = \begin{bmatrix} \mathbf{p}_0 \\ \vdots \\ \mathbf{p}_m \end{bmatrix}$$

☑ $\mathbf{M}$ is an $(m+1)\times(m+1)$ matrix with elements; $e_{ij} = B_j^m(t_i)$

☑ It can be solved with any linear solver.

☑ Polynomial interpolation does not provide satisfied result for higher degrees. Figure in the next slide should be convincing enough.

Bezier Curve Interpolation Beyond Cubics (3/3)



Data from a circle

One point is slightly modified.

- The processes of a small change in data can lead large change in the interpolating curve is called ill-conditioned.

- Different polynomial forms will give the identical result.

Bezier Curve Approximation (1/5)

- ☑ One is given more data points than should be interpolated by a polynomial curve (i.e., number of data points > degree of curve).
  - ➡ We can solve the problem by interpolating with a higher degree Bezier curve, but higher degree interpolation becomes ill-conditioned.

- ☑ In such cases, an approximating curve will be needed, which does not pass through the data points exactly; rather it passes near them.
  - ■ The best technique to find such curves
  - ➡ "Least squares approximation"

Bezier Curve Approximation (2/5)

- ☑ **Given: Points** $\mathbf{p}_0, \ldots, \mathbf{p}_m$ **and corresponding parameter values** $0 = t_0 < t_1 < \ldots < t_{m-1} < t_m = 1.$

- ☑ **We wish to find a polynomial curve $r(t)$ of a given degree n ($< m$) such that**

$$\sum_{i=1}^{m} \left\| \mathbf{p}_i - \mathrm{r}(t_i) \right\| \rightarrow minimize \quad (or) \quad \mathbf{p}_i = \mathrm{r}(t_i); \quad i = 0, 1, \ldots, m$$

- ☑ **The polynomial curve can have the Bezier form as bellows.**

$$\mathrm{r}(t) = \mathbf{b}_0 B_0^n(t) + \mathbf{b}_1 B_1^n(t) + \ldots\ldots\ldots + \mathbf{b}_n B_n^n(t)$$

Bezier Curve Approximation (3/5)

☑ We would like the following to hold:

$$\mathbf{p}_0 = \mathbf{b}_0 B_0^n(t_0) + \cdots\cdots + \mathbf{b}_n B_n^n(t_0)$$
$$\mathbf{p}_1 = \mathbf{b}_0 B_0^n(t_1) + \cdots\cdots + \mathbf{b}_n B_n^n(t_1)$$
$$\vdots \qquad \vdots \qquad \vdots$$
$$\mathbf{p}_m = \mathbf{b}_0 B_0^n(t_m) + \cdots\cdots + \mathbf{b}_n B_n^n(t_m)$$

⇒

$$\begin{bmatrix} B_0^n(t_0) & \cdots & B_n^n(t_0) \\ \vdots & & \vdots \\ B_0^n(t_m) & & B_n^n(t_m) \end{bmatrix} \begin{bmatrix} \mathbf{b}_0 \\ \vdots \\ \mathbf{b}_n \end{bmatrix} = \begin{bmatrix} \mathbf{p}_0 \\ \vdots \\ \mathbf{p}_m \end{bmatrix}$$

(n+1)*(2 or 3) Unknowns < (m+1)*(2 or 3) Equations ⇐ $\mathbf{MB} = \mathbf{P}$

Where, n < m

$$ex)\ \mathbf{p}_0 = [x_0 \quad y_0]^T \quad or \quad [x_0 \quad y_0 \quad z_0]^T$$

Bezier Curve Approximation (4/5)

☑ Multiply both sides by $\mathbf{M}^T$

$$\mathbf{M}^T\mathbf{M}\mathbf{B} = \mathbf{M}^T\mathbf{P}$$ ⬅ Normal equation

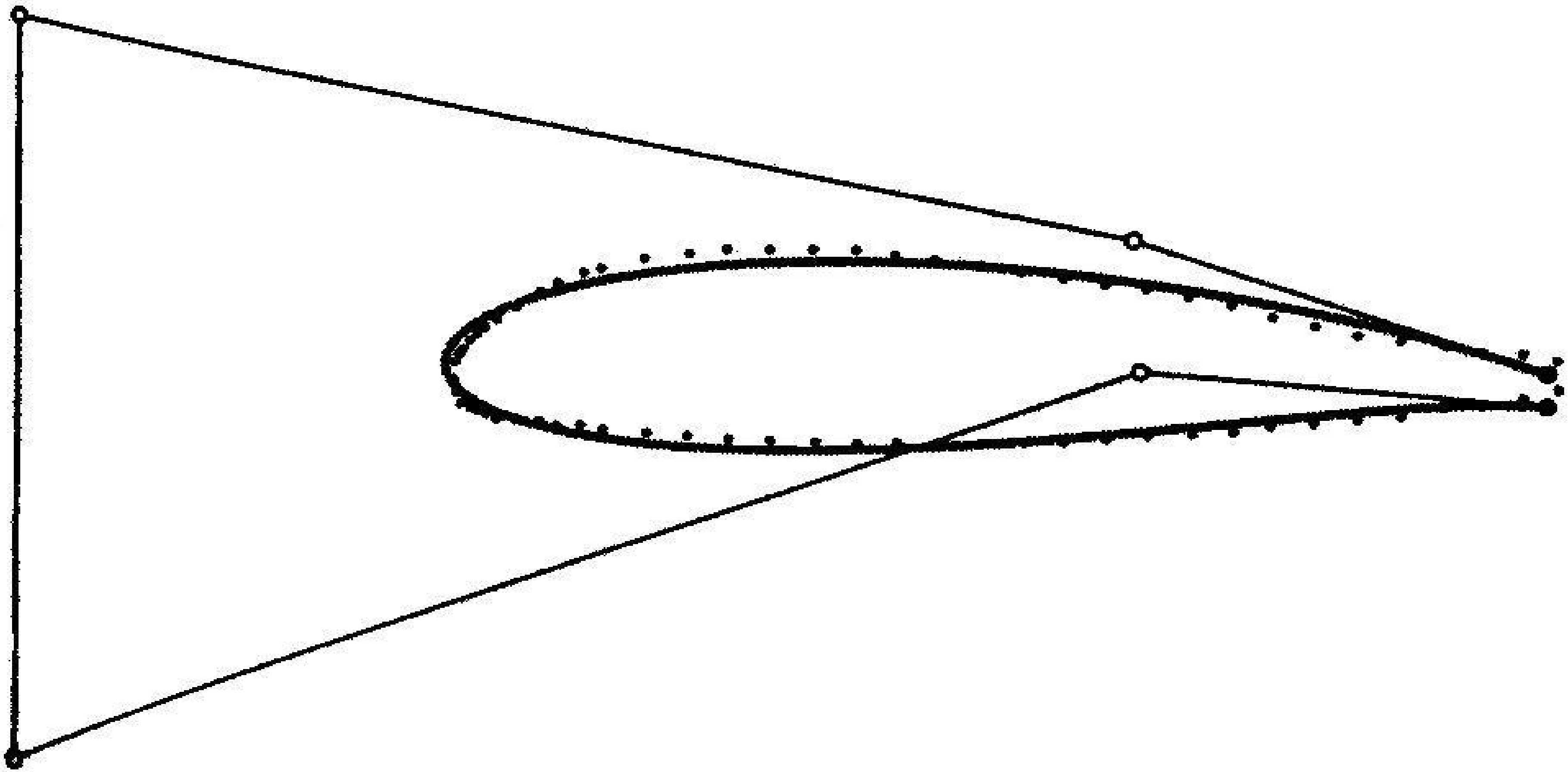
where $\mathbf{M}^T\mathbf{M}$ is a square and symmetric matrix, which is always invertible.

☑ The curve B minimizes the sum of the $\left\|\mathbf{p}_i - \mathrm{r}(t_i)\right\|, \quad i = 0,1,\ldots,m$

$$\therefore\ \mathbf{B} = \left(\mathbf{M}^T\mathbf{M}\right)^{-1}\mathbf{M}^T\mathbf{P}.$$

☑ Note that any modification of the t_i would result in an entirely different solution.

Bezier Curve Approximation (5/5)



<Least square approximation to a wing>

A 5^{th}-degree(quantic) Bezier curve with chord length parameters was assigned to the input points.

Chord Length Parameters (1/2)

☑ In both interpolation and approximation curve, in practice, the parameter value t_i are not normally given, and have to be made up.

☑ There are two types to be made up.

(1) Uniform sets of parameters

- If there are $(m+1)$ points p_i, then set $t_i = i/l$.

(2) Chord length parameters

- If the distance between two points is relatively large, then their parameter values should also be fairly different.

$$t_0 = 0$$

$$t_1 = t_0 + \|\mathbf{p}_1 - \mathbf{p}_0\|$$

$$\vdots$$

$$t_l = t_{l-1} + \|\mathbf{p}_l - \mathbf{p}_{l-1}\|$$

Chord Length Parameters (2/2)

☑ If desired (it makes no difference to the interpolation or approximation result), the parameters may be normalized by scaling the parameters to live between zero and one.

$$t_i = \frac{t_i - t_0}{t_m - t_0}.$$

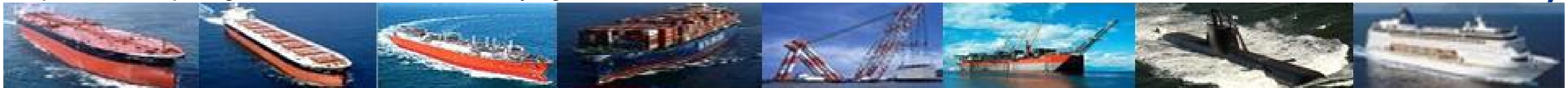
☑ In general, the chord length parameterization method is superior to the uniform method, because it takes into account the geometry of the data.

[Appendix] Programming for Bezier Curve

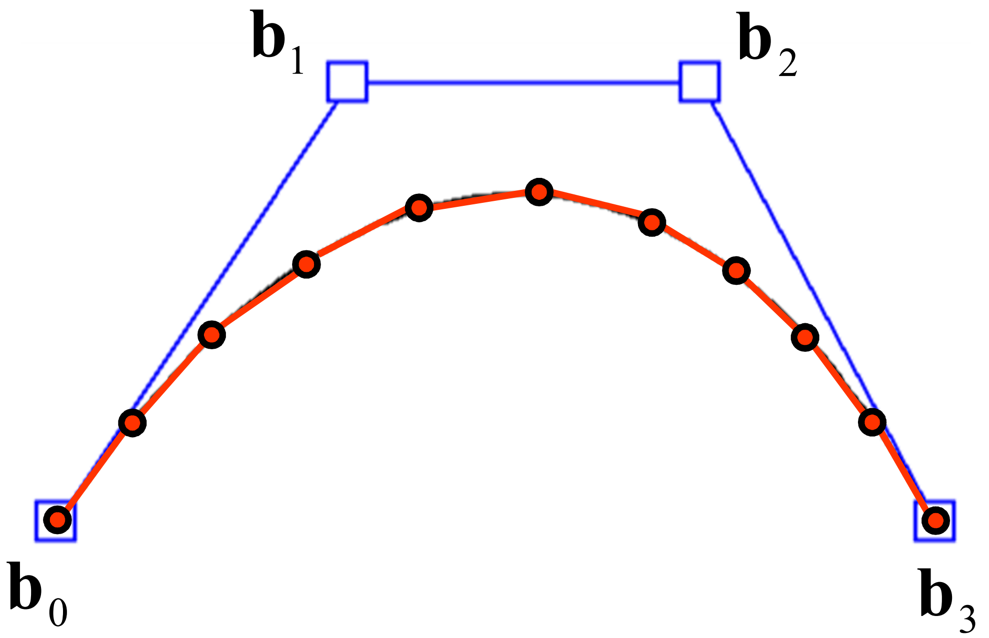
(1) Sample Code for Bezier Curve Class

(2) Sample Code of de Casteljau Algorithm

(3) Sample Code of Curve Interpolation and Approximation



Programming for Bezier Curve Class



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

1) A Bezier curve is defined by

- Degree
- Control Point

<u>Member Variables of Bezier Curve Class</u>
int n: Degree of the Bezier Curve
Vector* m_ControlPoint: Control Points
int m_nControlPoint: Number of Control Points

2) Calculation of Bernstein Polynomial

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i},$$

$$\binom{n}{i} = {}_nC_i = \begin{cases} \dfrac{n!}{i!(n-i)!} & \textbf{if } 0 \le i \le n \\ \mathbf{0} & \textbf{else} \end{cases}$$

3) Construction of the Bezier Curve

- Construct the curve divided by line segment.
- After divide a parameter t(0~1) into n equal parts, find the points on the curve at the each point to be divided.
- Visualize the curve by connecting points with straight lines.

Sample Code for Bezier Curve Class (1/3)

```
#ifndef __BezierCurve_h__
#define __BezierCurve_h__

#include “vector.h”

class BezierCurve {
public:
    int n;  // Degree of Bezier Curve
    Vector* m_ControlPoint;   int m_nControlPoint;
    BezierCurve();
    ~BezierCurve();

    void SetDegree(int degree);
    void SetControlPoint(Vector* pControlPoint, int nC
    Vector CalcPoint(double t);
    double B (int i, double t);                    // Bernstein Polynomial
};
#endif
```

Member Variables

int n: Degree of the Bezier Curve

Vector* m_ControlPoint: Control Points

int m_nControlPoint: Number of Control Points

Sample Code for Bezier Curve Class (2/3)

```
BezierCurve::BezierCurve () {
   m_ControlPoint = 0;  n= 0;
   m_nControlPoint = 0;
}

BezierCurve::~BezierCurve () {
   if(m_ControlPoint) delete[] m_ControlPoint;
}

void BezierCurve::SetControlPoint(Vector* pControlPoint, int nControlPoint) {
   SetDegree( nControlPoint-1 );
   if(m_ControlPoint) delete[] m_ControlPoint;
   m_ControlPoint = new Vector[nControlPoint];
   for(int i=0; i < nControlPoint; i++) {
      m_ControlPoint[i] = pControlPoint[i];
   }
}

void BezierCurve::SetDegree(int degree){
   n = degree;
}
```

Sample Code for Bezier Curve Class (3/3)

```
Vector BezierCurve:: CalcPoint(double t) {
    Vector PointOnCurve(0,0,0);
    if ( t < 0.0 || t > 1.0 ) {
        return PointOnCurve;
    }
    for(int i = 0; i < m_nControlPoint; i++){
        PointOnCurve = PointOnCurve + m_ControlPoint[i] * B(i,t);
    }
    return PointOnCurve;
}

double BezierCurve:: B (int i, double t) {
    double result = 0;
    // Calculate ith Berstein Polynomial at parameter t
    result = comb(n, i) * pow(t, i) * pow(1.0 - t, n-i);
    return result;
}
```

$$\mathbf{r}(t) = \mathbf{b}_0 B_0^n(t) + \mathbf{b}_1 B_1^n(t) + \ldots\ldots + \mathbf{b}_n B_n^n(t)$$

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i},$$

$$\binom{n}{i} = {}_nC_i = \begin{cases} \dfrac{n!}{i!(n-i)!} & \textbf{if } 0 \le i \le n \\ \mathbf{0} & \textbf{else} \end{cases}$$

Sample Code of de Casteljau Algorithm (1/2)

```
#ifndef __BezierCurve_h__
#define __BezierCurve_h__

#include "vector.h"

class BezierCurve {
public:
    int m_nDegree;
    Vector* m_ControlPoint;   int m_nControlPoint;
    BezierCurve();
    ~BezierCurve();

    void SetDegree(int nDegree);
    void SetControlPoint(Vector* pControlPoint, int nControlPoint);
    Vector CalcPoint(double t);
    Vector deCasteljau(double t);  // Calculation of the point on curve by de Casteljau algorithm
    double B (int i, double t);
};
#endif
```

Sample Code of de Casteljau Algorithm (2/2)

```
Vector BezierCurve:: deCasteljau (double t) {
    Vector* TmpControlPoint = new Vector [m_nControlPoint];
    for(int i = 0; i < m_nControlPoints; i++) TmpControlPoint[i] = m_ControlPoint[i];

    for(i = 1; i < m_nControlPoint; i++){
        for(int j = 0; j < m_nDegree - i; j++){
            TmpControlPoint[j] = (1-t)*TmpControlPoint[j]  + t*TmpControlPoint[j+1];
            //      b_j^i                    b_j^(i-1)                  b_(j+1)^(i-1)
        }
    }

    Vector result = TmpControlPoint[0];  // b_0^3
    delete[] TmpControlPoint;
    return result;
}
```

$$
\begin{array}{cccc}
\mathbf{b}_0^0 & \mathbf{b}_0^1 & \mathbf{b}_0^2 & \mathbf{b}_0^3 \\
\mathbf{b}_1^0 & \mathbf{b}_1^1 & \mathbf{b}_1^2 & \\
\mathbf{b}_2^0 & \mathbf{b}_2^1 & & \\
\mathbf{b}_3^0 & & &
\end{array}
$$

Sample Code of Curve Interpolation and Approximation (1/4)

```
#include "vector.h"

class BezierCurve {
public:
    int m_nDegree;
    Vector* m_ControlPoint;   int m_nControlPoint;
    ......
    void SetDegree(int nDegree);
    void SetControlPoint(Vector* pControlPoint, int nControlPoint);
    Vector CalcPoint(double t);
    double B (int i, double t);
    int Approximation(int nDegree, int nType, Vector* FittingPoint, int nPoint);
    int Interpolation(int nType, Vector* FittingPoint, int nPoint);
    void Parameterization(int nType, Vector* FittingPoint, int nPoint, double* t);
};
```

Sample Code of Curve Interpolation and Approximation (2/4)

```
void BezierCurve:: Parameterization (int nType, Vector* FittingPoint, int nPoint, double* t){
    // Assume t is allocated out of function
    if( nType == 1) {                              // Uniform set
        for (int i = 0; i < nPoint; i++)
                        t[i] = 1./(nPoint-1);
    } else if ( nType == 2) {                      // Chord length
        t[0] = 0.;
        for (int i=0; i < nPoint-1; i++)
            t[i+1] = t[i] + (FittingPoint[i+1] - FittingPoint[i]).Magnitude();
        double t0 = t[0], tm = t[nPoint-1];
        for (int i=0; i < nPoint; i++)
            t[i] = (t[i] -  t0)/(tm - t0);         // Normalize
    }
}
```

Sample Code of Curve Interpolation and Approximation (3/4)

```
int BezierCurve:: Approximation(int nDegree, int nType, Vector* FittingPoint, int nPoint){
    m_nDegree = nDegree;
    m_nControlPoint = m_nDegree+1;
    if(m_ControlPoint) = delete[] m_ControlPoint;
    m_ControlPoint = new Vector[m_nControlPoint];

    double* t = new double[nPoint];
    Parameterization(nType, FittingPoint, nPoint, t);

    // Solve the normal equation
    ....
   delete[] t;
}
```

Sample Code of Curve Interpolation and Approximation (4/4)

```
int BezierCurve:: Interpolation(int nType, Vector* FittingPoint, int nPoint){
    …

    double** M = new double*[nNumOfPoint];
    for (i=0; i<nNumOfPoint; i++) M[i] = new double[nNumOfPoint];

    for (i=0; i<nNumOfPoint; i++)    {
        for (j=0; j<nNumOfPoint; j++)    {
            M[i][j] = B(j, t[i]);
        }
    }

    // Solve MB = P
    GaussElimination(nNumOfPoint, M, p_x, b_x);
    GaussElimination(nNumOfPoint, M, p_y, b_y);
    GaussElimination(nNumOfPoint, M, p_z, b_z);
    ….

}
```

$$\begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{bmatrix} = \begin{bmatrix} B_0^3(t_0) & B_1^3(t_0) & B_2^3(t_0) & B_3^3(t_0) \\ B_0^3(t_1) & B_1^3(t_1) & B_2^3(t_1) & B_3^3(t_1) \\ B_0^3(t_2) & B_1^3(t_2) & B_2^3(t_2) & B_3^3(t_2) \\ B_0^3(t_3) & B_1^3(t_3) & B_2^3(t_3) & B_3^3(t_3) \end{bmatrix} \begin{bmatrix} \mathbf{b}_0 \\ \mathbf{b}_1 \\ \mathbf{b}_2 \\ \mathbf{b}_3 \end{bmatrix}$$

$$\mathbf{P} = \mathbf{MB}$$

$$\mathbf{B} = \mathbf{M}^{-1}\mathbf{P}$$