[2008][01-1]

Computer aided ship design

Part 1. Curve & Surface

September 2008

Prof. Kyu-Yeul Lee

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

Advanced
Ship
Design
Automation
Laboratory

Ch0. Introduction

Advanced
Ship
Design
Automation
Laboratory

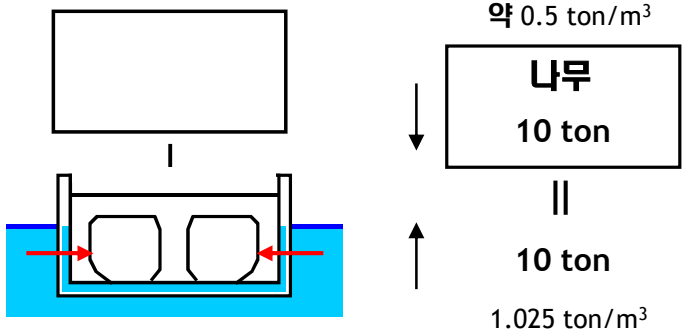
선박설계에서 고려해야 할 점

철의 밀도 = 7.85 ton/m³

■ 선박의 기본적인 요건

1) 물에 떠야 한다 — 선박안정론

→ 선박의 무게 = 밀어낸 물의 무게*



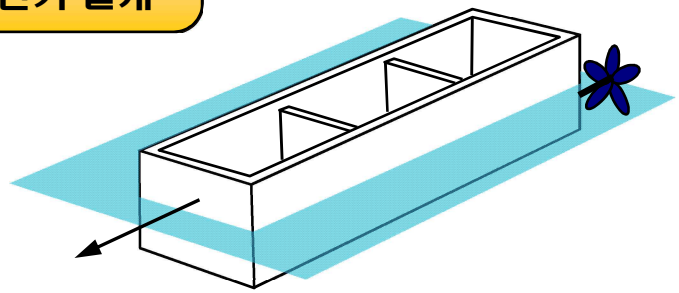
2) 원하는 목적지로 빨리 갈 수 있어야 한다 — 선박형상설계, 선박유체역학, 추진기 설계

→ 형상: 물의 저항이 작은 형태(ex. 유선형)

→ 추진기관: 디젤 엔진, 나선형 프로펠러

3) 짐을 실을 수 있어야 한다 — 선박구획설계

→ 최대한 많은 짐을 실을 수 있도록 내부가 비어있어야 함



4) 튼튼한 그릇으로서의 역할을 해야 한다 — 선박구조역학/ 구조설계.해석

→ 철판(약 10~30 mm두께)과 보강재를 용접한 구조물

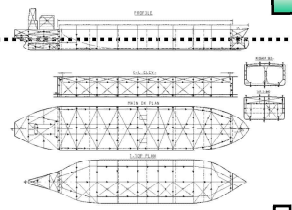
* 아르키메데스의 원리: 유체 속에 있는 물체(부유체)가 받는 부력의 크기는 그 물체가 밀어낸 유체의 무게와 같고 그 방향은 중력과 반대 방향이다.

선박설계 단계

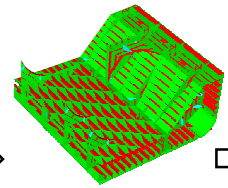
선박 초기(기본) 설계 → 선박 상세 설계 → 선박 생산 설계

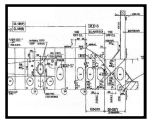
TRIBON 시스템

생산 설계 단계부터 CAD 모델 생성 가능



블록 분할도



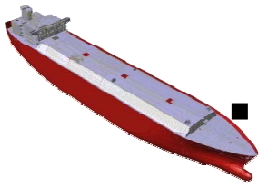


생산 도면 추출

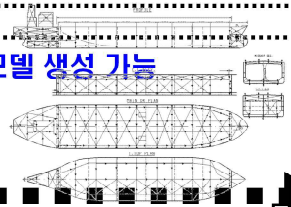
주어진 블록 분할도에 따라 블록별 생산 모델(선체+의장) 생성

IntelliShip 시스템

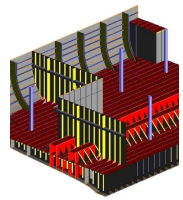
상세 설계 단계부터 CAD 모델 생성 가능



선체 전체에 대한 상세 모델 생성



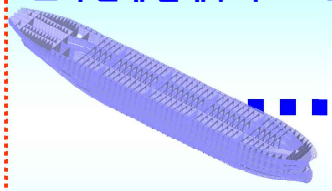
블록 분할도



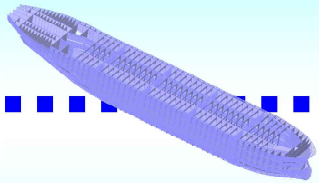
주어진 블록 분할도에 따라 블록별 생산모델(선체+의장) 생성

구조 부재간 연관성을 고려한 선체 모델링 시스템 (본 연구)

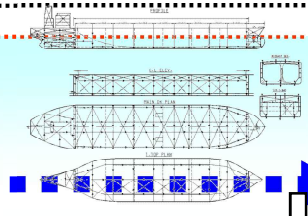
초기 설계 단계부터 CAD 모델 생성 가능



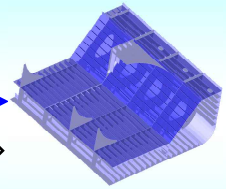
선체 전체에 대한 초기 구조 모델 생성



선체 전체에 대한 상세 구조 모델 생성



블록 분할도



주어진 블록 분할도에 따라 블록별 생산 모델(선체) 생성

* TRIBON 시스템: 조선 전용 CAD 시스템, 스웨덴 TRIBON Solution사 개발

* IntelliShip: M-CAD(Intergraph)의 조선 전용화 시스템, Intergraph사, 삼성 중공업, 덴마크 Odense 조선소, 일본 Hitachi 조선소 공동 개발

선박 초기 설계 단계

개념 설계

주요 요목 결정

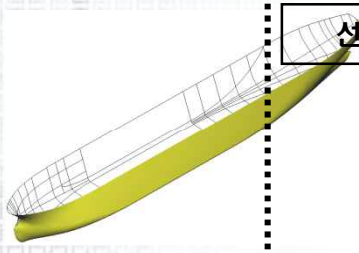
선박 형상 모델링

기준선 선형 입력

선형 변환(variation)

선형 순정(fairing)

선형 모델 완성



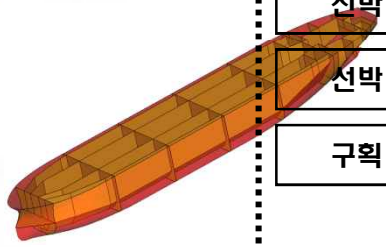
본 연구에서는
선박 형상 모델링을 위해
기존 틀을 사용

선박 구획 모델링

선박 구획 정의

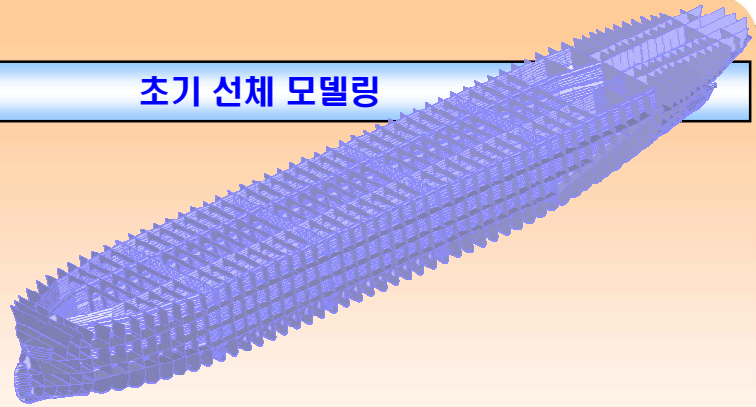
선박 계산 수행

구획 모델 완성

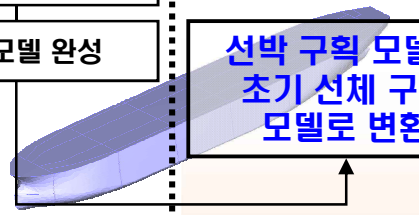


본 연구에서는
선박 구획 모델링을 위해
기존 틀을 사용

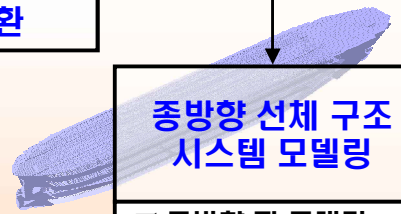
초기 선체 모델링



선박 구획 모델을
초기 선체 구조
모델로 변환

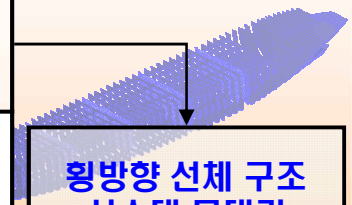


종방향 선체 구조 시스템 모델링



- ☑ 종방향 판 모델링
- ☑ 종방향 보강재 모델링
- ☑ 종방향 부재의 상세 모델링

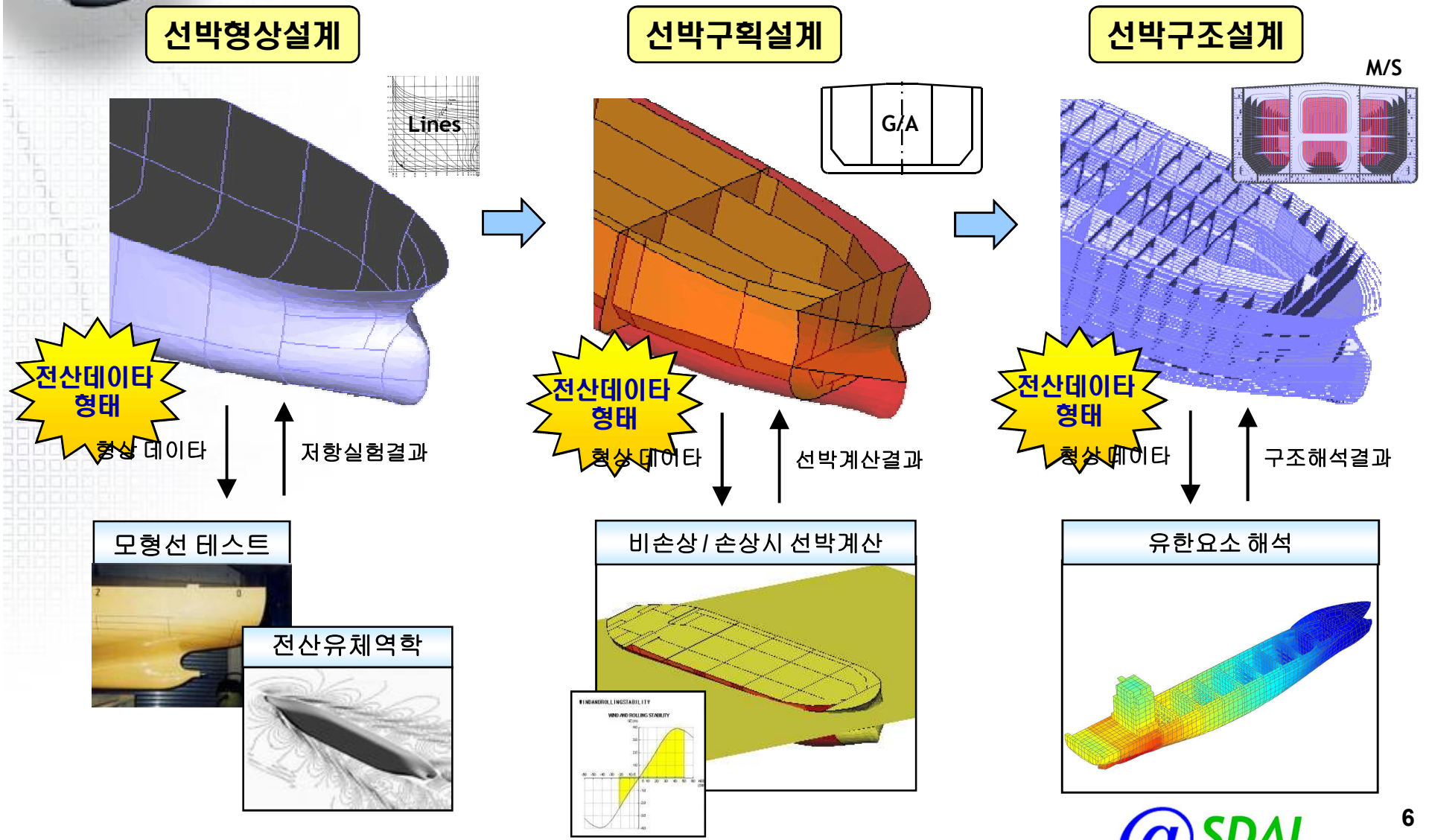
횡방향 선체 구조 시스템 모델링



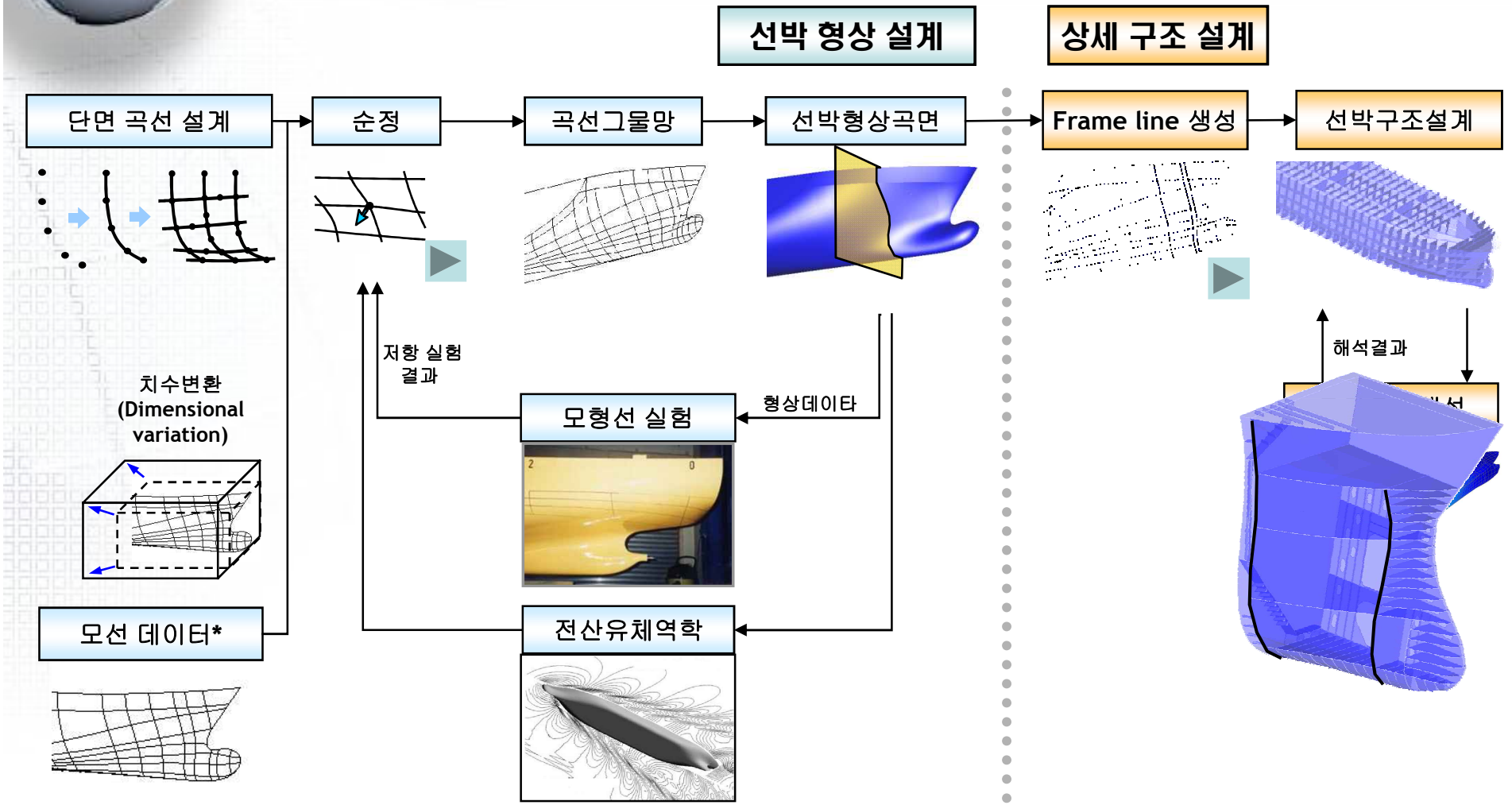
- ☑ 횡방향 판 모델링
- ☑ 횡방향 보강재 모델링
- ☑ 횡방향 부재의 상세 모델링

* 종방향 선체 구조 시스템: 외판 시스템, 갑판 시스템, 거더 시스템, 스트링거 시스템, 종격벽 시스템 등
* 횡방향 선체 구조 시스템: 횡격벽 시스템, 웨브 프레임 시스템

선박의 초기설계 과정


선박형상설계
선박구획설계
선박구조설계
Lines
G/A
M/S
전산데이타
형태
형상 데이타
저항실험결과
선박계산결과
구조해석결과
모형선 테스트
전산유체역학
비손상 / 손상시 선박계산
유한요소 해석

선박형상설계의 위치

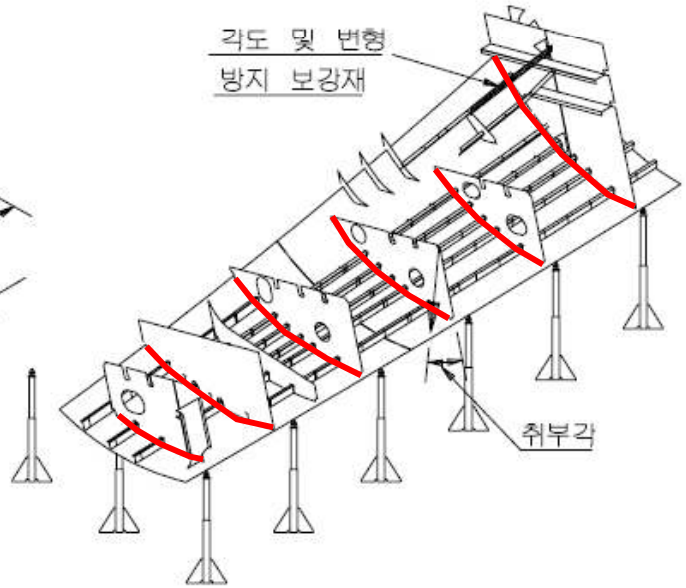


* 선박 설계는 무에서 유를 창조하는 혁신적인 업무라기보다는 실적 자료를 토대로 한 개선

선박형상 곡면의 필요성

■ 초기설계단계에서 생산과 관련된 정보를 조기에 추출가능

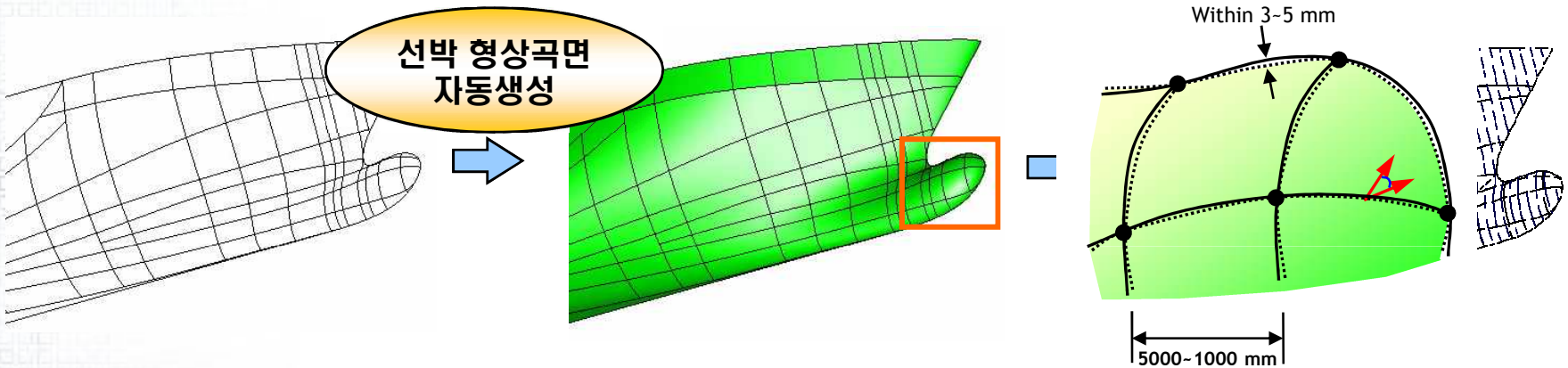
1) 용접길이, 소요시수(man-hour) 계산을 통하여 조기에 공정일정 계획 수립가능
   → 선가 견적, 건조기간 예상에 큰 도움
2) 곡블록을 고정하기 위한 취부(zig) 정보
3) 도장면적 계산을 통한 정확한 도료 물량 계산



선박 형상곡면의 품질 요구사항

초기선형설계 → 상세설계 / 생산설계

Given : 곡선그물망 → Find : 선박 형상곡면 → Frame line 생성



- Non-uniform B-spline 곡선
- 비정규 위상의 곡선그물망

제약조건

- B-spline (or Bezier) 곡면 형식
- 곡선그물망을 정확하게 보간 or
  곡선그물망과 곡면사이의 최대거리<업계 허용오차*
- 순정된 선박 형상곡면: 접평면 연속(G^1 연속)**

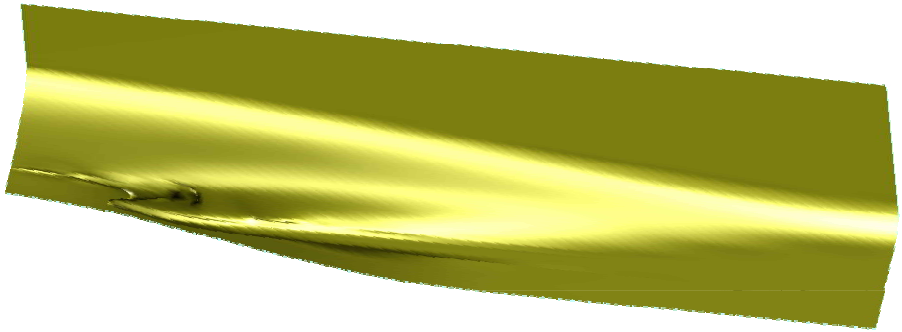
- 선박 형상곡면의 순정도를 검증
- 곡면과 평면과의 교차계산으로 생성가능

* 조선소에서 통용되는 최대 거리 오차는 약 3~5 mm 임

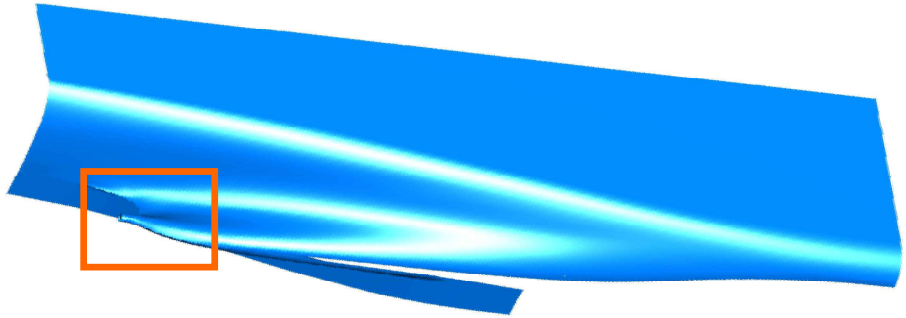
** 조선전용 상세/생산설계 시스템인 IntelliShip은 exact G^1 연속인 곡면을 입력정보로 요구하고 있음

관련 연구 현황

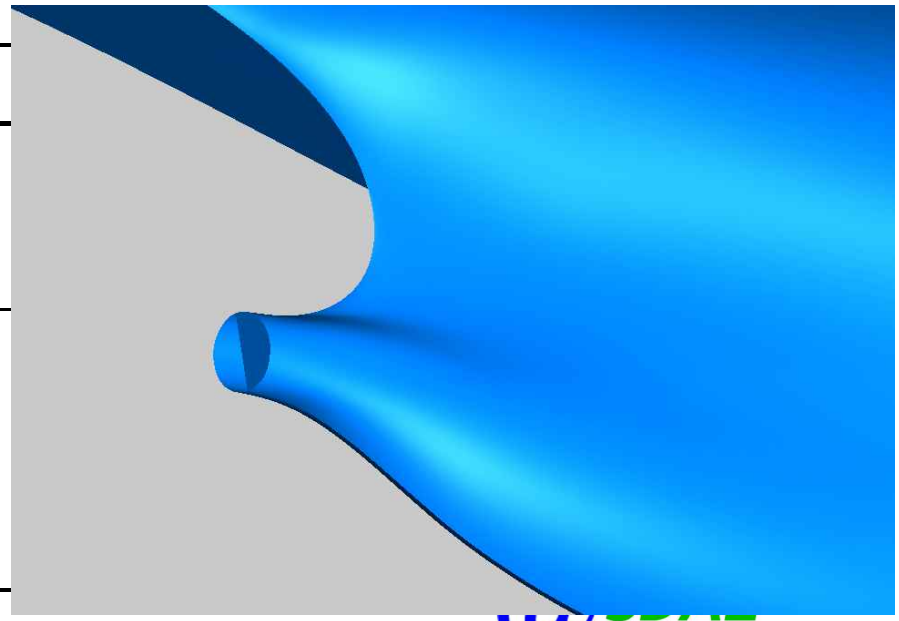
■ **단일 곡면 생성방법**
(Single patch approach)



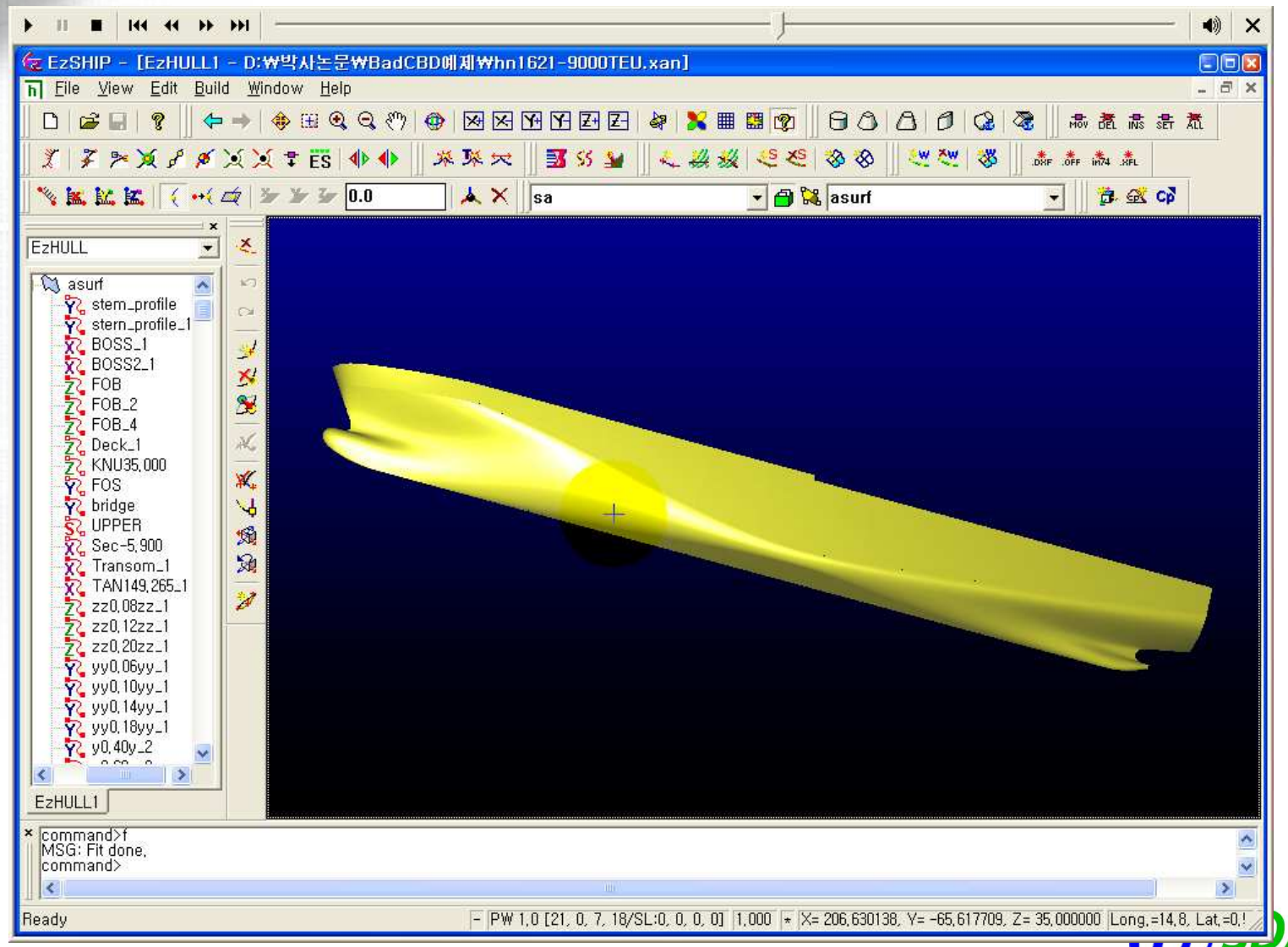
■ **곡면 조각을 이용하는 방법**
(Piecewise patch approach)



방법	단일 곡면 생성방법
장점	• 선박 형상곡면의 표현방법이 간단 • 곡면상의 모든 점에서 수학적으로 2차 미분 연속(C^2)
단점	• 단일 B-spline 곡면으로는 복잡한 형상을 갖는 선수, 선미부분에 → ① 곡선그물망을 정확하게 보간하는 ② 순정된 선박 형상곡면을 생성하기 어려움 • Knuckle curve 형상을 정확하게 표현하기 어려움



Demonstration



Ch1. 학습 목표

1.1 주어진 점을 지나는 곡선 만들기

1.2 주어진 점을 지나는 곡면 만들기

Advanced
Ship
Design
Automation
Laboratory

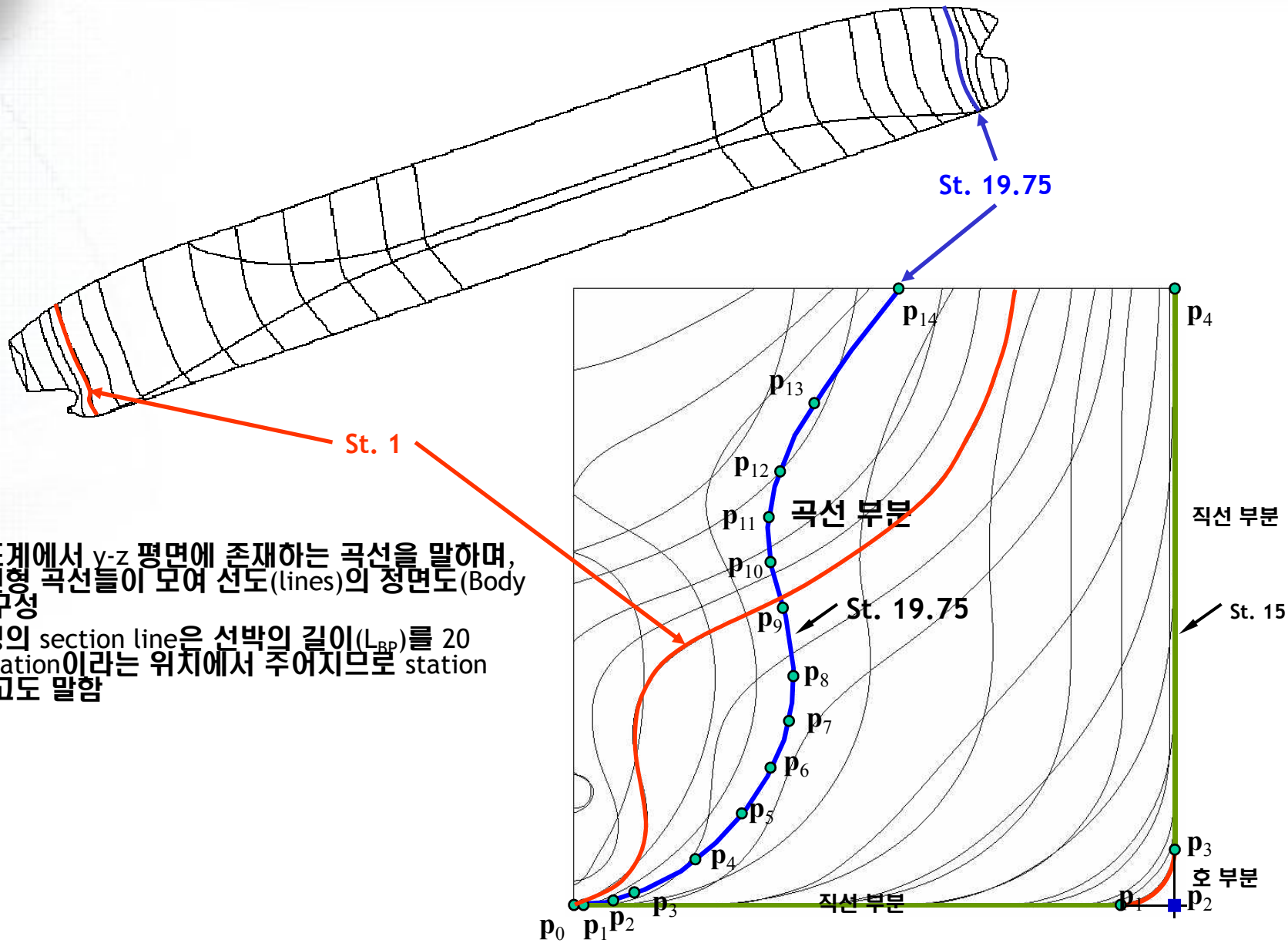
1.1 주어진 점을 지나는 곡선 만들기

1.1.1 **선형 표현을 위한 주요 곡선** – Section line

1.1.2 **선형 표현을 위한 주요 곡선** – Water line **생성**

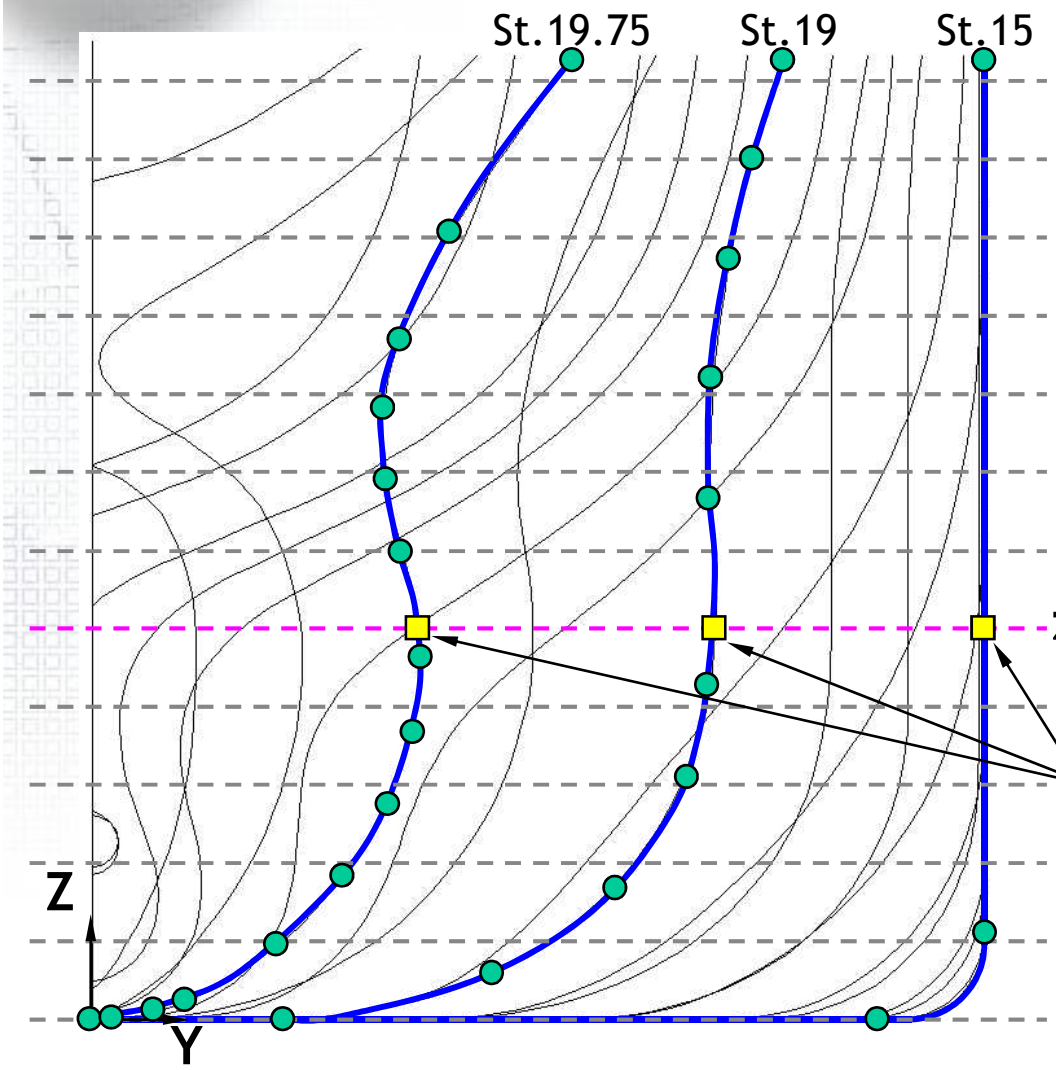
Advanced
Ship
Design
Automation
Laboratory

1.1.1 선형 표현을 위한 주요 곡선들 - Section Line



- 선형 좌표계에서 y-z 평면에 존재하는 곡선을 말하며, 이러한 선형 곡선들이 모여 선도(lines)의 정면도(Body Plan)를 구성
- 보통 선형의 section line은 선박의 길이(L_{BP})를 20 등분한 station이라는 위치에서 주어지므로 station line이라고도 말함

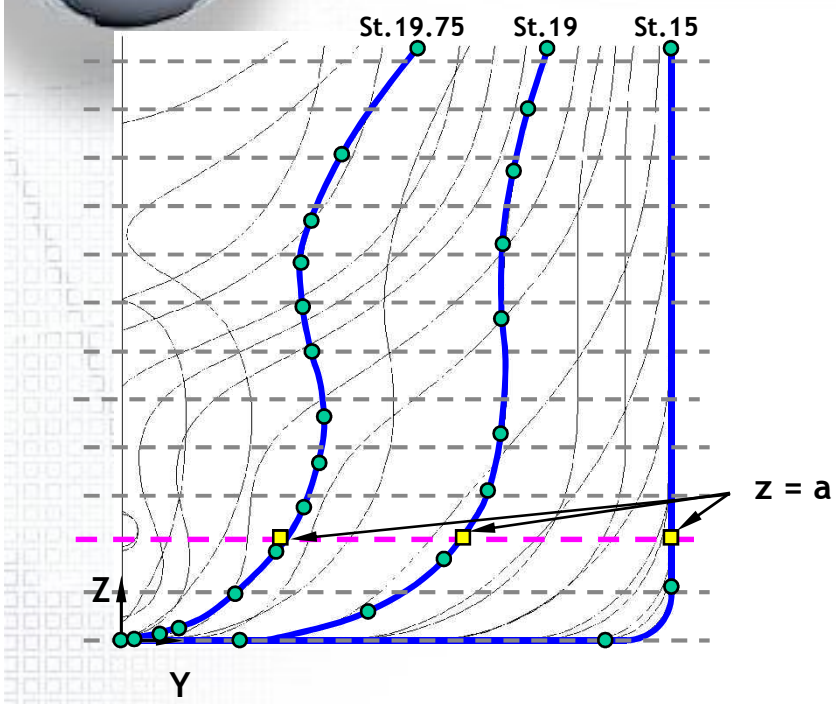
1.1.2 선형 표현을 위한 주요 곡선들 - Water Line 생성 (1)

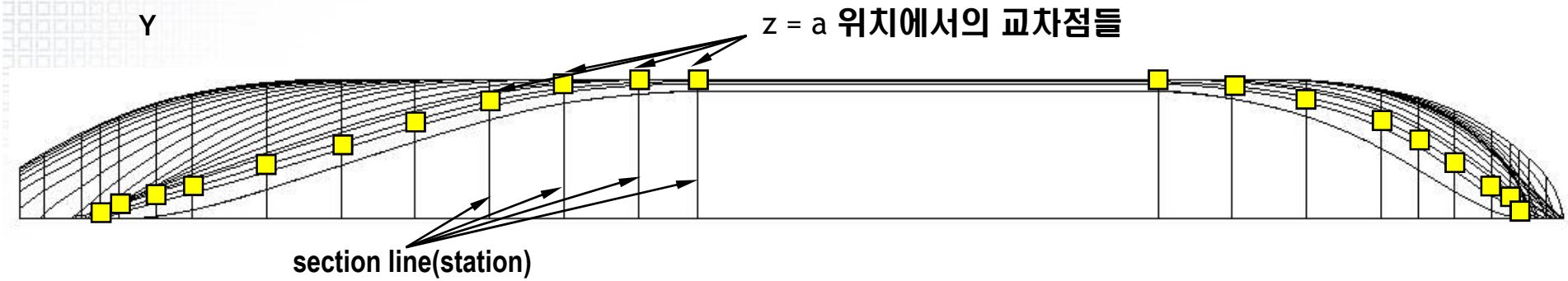


➡ z = a 평면과 주요 곡선 및 모든 section line들과의 교차 계산을 통해 waterline 생성

z = a에서의 waterline 생성을 위한 교차점

1.1.2 선형 표현을 위한 주요 곡선들 - Water Line 생성 (2)



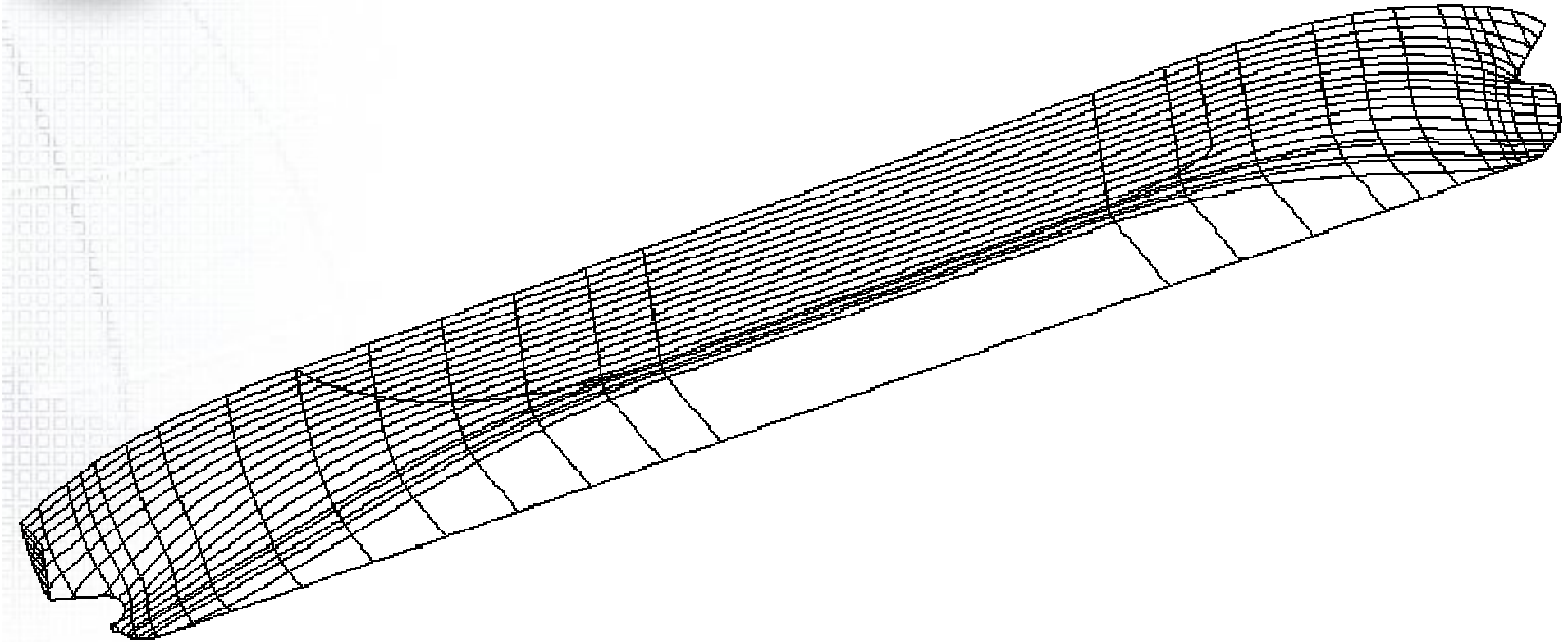


z = a **위치에서의 모든 교차점들을** NURB **곡선을 이용하여 보간**(fitting)
➡ z = a**에서의** waterline **생성**

➡ Waterline **생성**

원하는 z **위치에 대해 위 과정을 반복 수행**

1.1.2 선형 표현을 위한 주요 곡선들 - Water Line 생성 (3)



1.2 주어진 점을 지나는 곡면 만들기

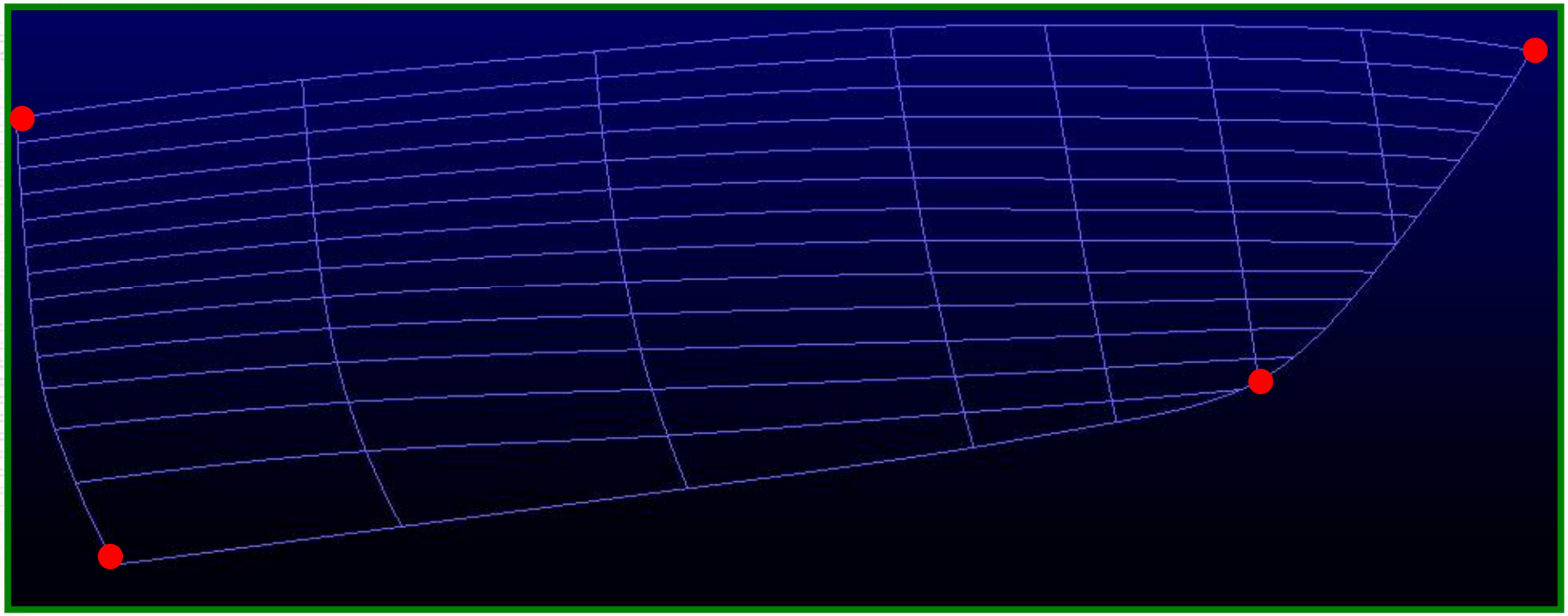
1.2.1 요트 형상의 선박형상곡면 생성

1.2.2 구상선수를 갖는 선박형상곡면 생성

Advanced
Ship
Design
Automation
Laboratory

1.2.1 요트 형상의 선박형상 곡면 생성 (1)

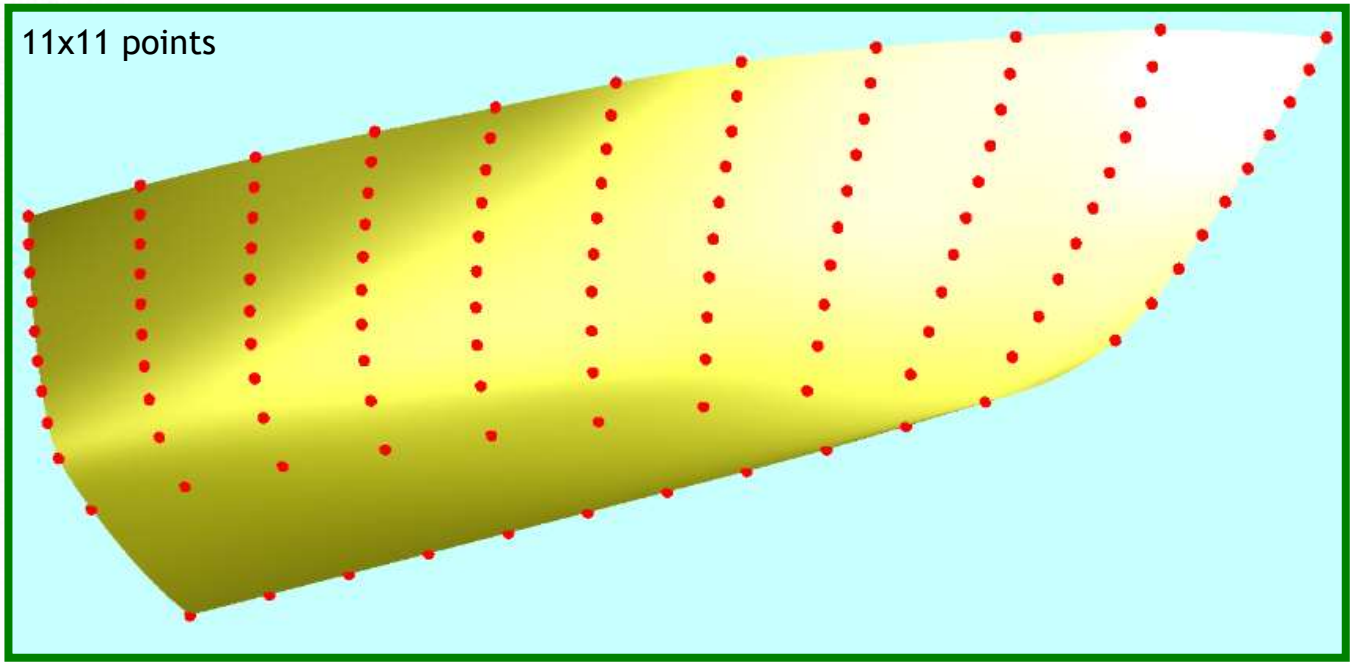
* 출처 : 서울대 조선해양공학과 2005년 2학년 교과목 『 조선해양공학계획』 강좌 중에 학생들이 설계한 선형

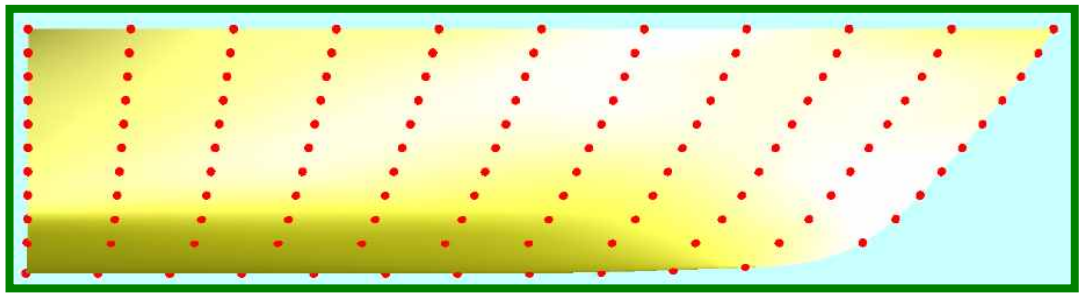


사각형 패치의 꼭지점 결정

1.2.1 요트 형상의 선박형상 곡면 생성 (2)

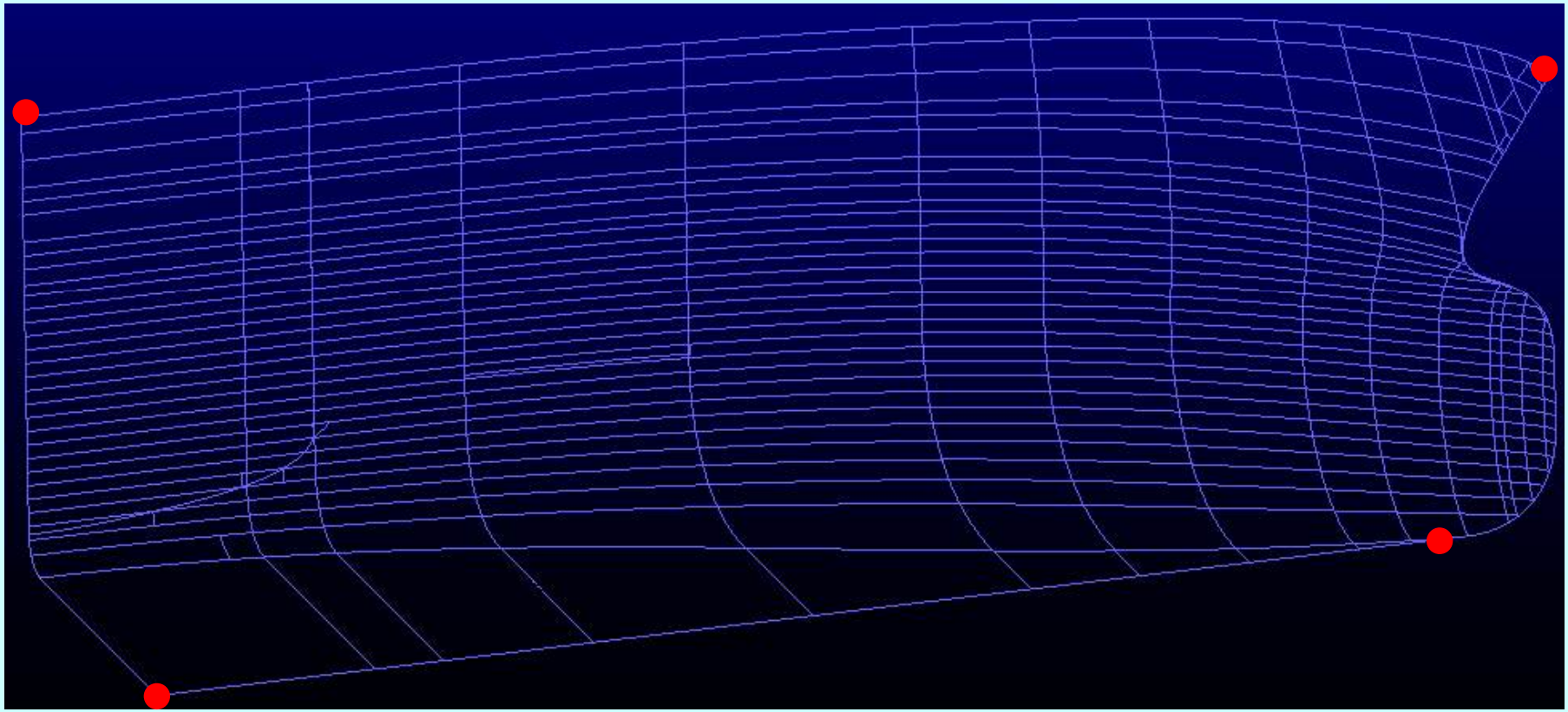
❑ 점들의 x좌표 사이의 거리가 거의 일정하도록 점을 추출한 후
→ 이 점 data로부터 선형곡면을 생성한 결과





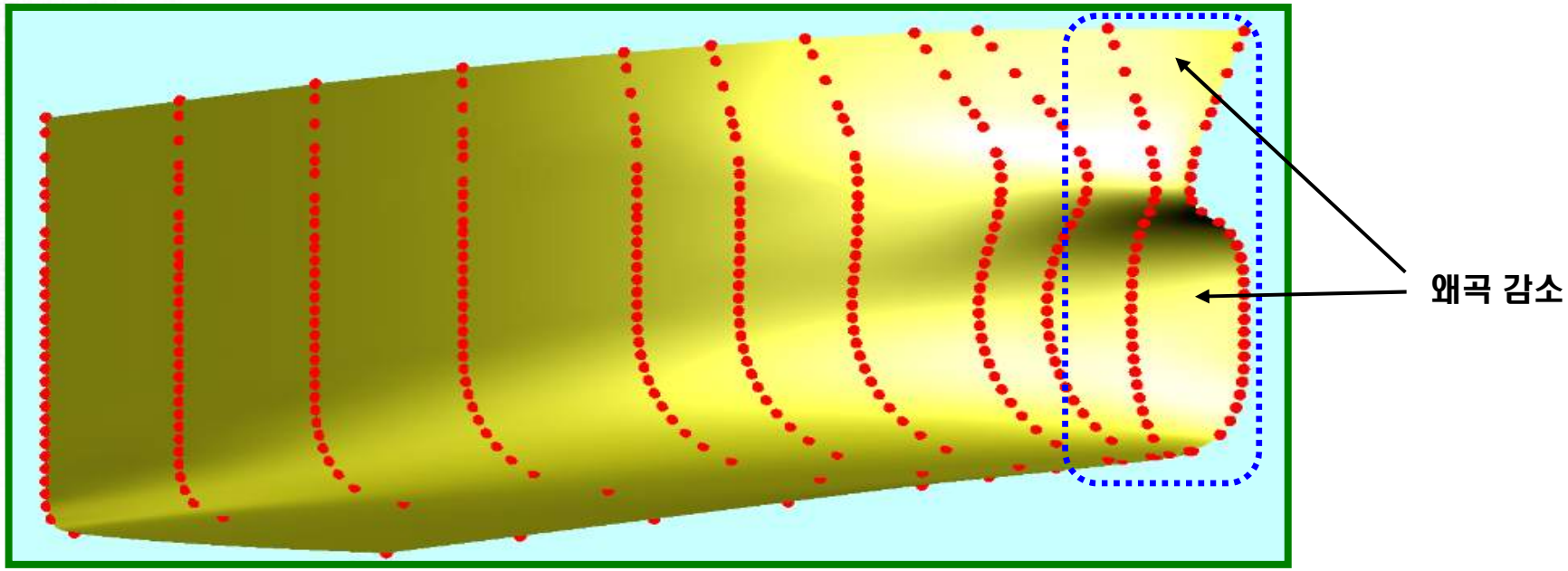
1.2.2 구상선수를 갖는 선박형상곡면 생성 (1)

선수부



1.2.2 구상선수를 갖는 선박형상곡면 생성 (2)

❑ 주어진 곡선그물망 이외의 보조선을 생성하여 곡선 보간에 적합한 점 data를 생성한 후, 점 data로부터 선수부 선형곡면을 생성한 결과



Ch 2. 곡선(Curves)

2.1 Parametric function/curves

2.2 Bezier Curves

2.3 B-spline Curves

Advanced
Ship
Design
Automation
Laboratory

2.1 Parametric function/ curves

2.1.1 양함수/음함수/매개변수 함수

2.1.2 매개변수 함수의 특징

2.1.3 일반함수의 매개변수 함수 표현

2.1.4 매개변수 함수의 직관적 표현방법

Advanced
Ship
Design
Automation
Laboratory

2.1.1 Explicit / Implicit / Parametric function

- Explicit function(**양함수식**)
  - **함수가** y=f(x) **의 형태로 나타내어지면** **양함수**라고 함
  - x**가 주어지면** y**를 쉽게 구할 수 있음**

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

- Implicit function(**음함수식**)
  - **변수가** x, y **두 개일 때, 함수** f(x, y) = 0 **와 같은 형태로 표현되는 식을** **음함수**라고 함
  - **주어진 점이 곡선의 내부 또는 외부, 왼쪽 또는 오른쪽)에 있는지 판단하기 쉬움**
  - f(x, y) = 0**인 함수를** y**에 대해 풀어서** y = f(x)**의 형태가 얻어지면 양함수로 변환이 가능함. 모든 음함수가 양함수로 변환될 수 있는 것은 아님**

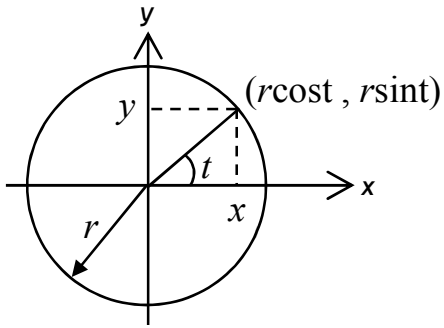
$ex)\quad x^2+y^2-r^2=0$

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

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

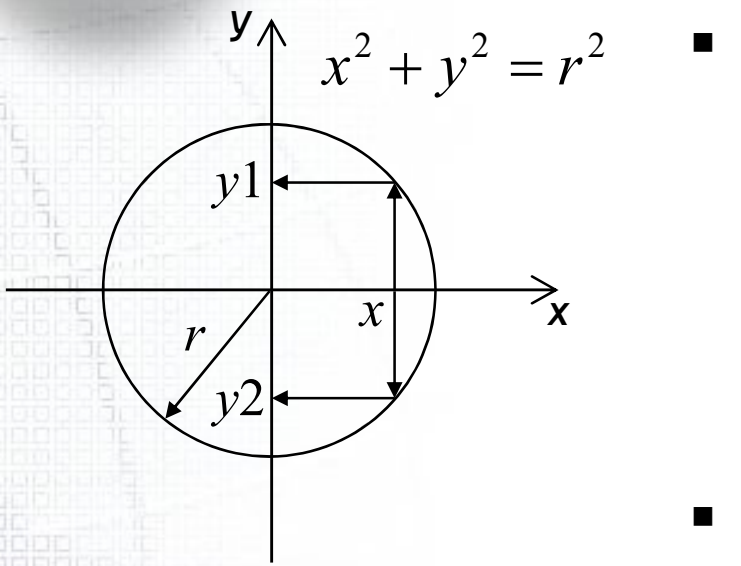
$ex)\quad y=\pm\sqrt{r^2-x^2}$

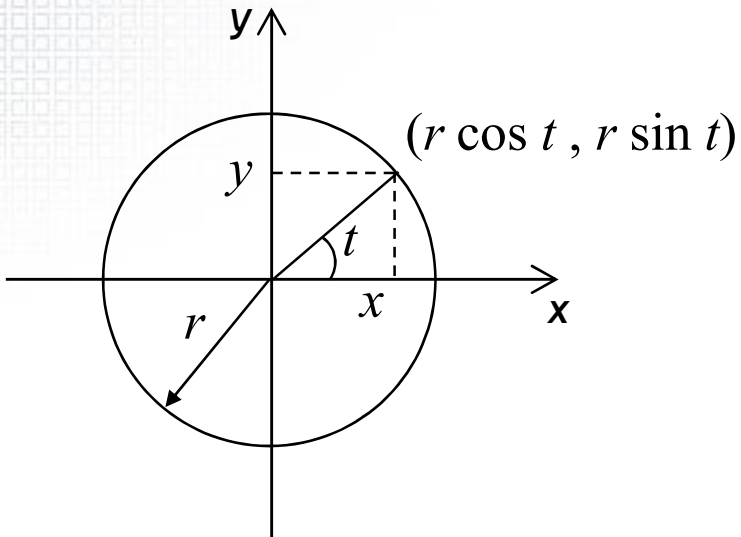
- Parametric function(**매개 변수 함수식**)
  - 2**변수 함수가 매개 변수** t**를 이용하여,** x = f(t), y = g(t)**로 나타내어지면 이를** **매개 변수 함수**라고 함
  - **즉, 매개 변수** t**를 매개로 하여** x**와** y**를 표현하는 형식**
  - **모든 양함수식은 매개 변수 함수식으로 변환이 가능함**



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

2.1.2 매개변수 함수(Parametric function)의 특징 (1)





- **일반적인 함수**
  - **하나의 x 값에 대해 y 값이 두 개이상 나올 수 있음** (multi-value function)

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

  - **미분 계수를 표현하기 어려운 경우가 있음** $\left.\frac{dy}{dx}\right|_{,x=r} = \infty$

- **매개 변수 함수**
  - **하나의 매개 변수 값에 대해 하나의 값만이 나옴**

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

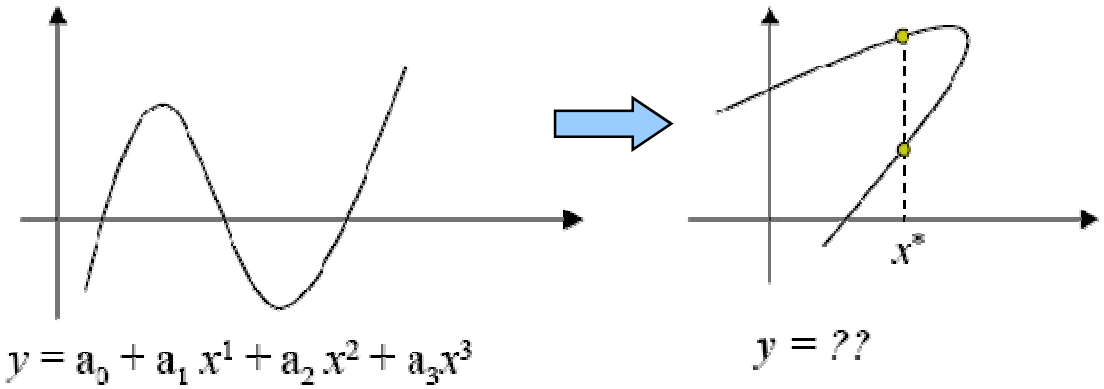
  - **미분 계수를 표현하기 쉬움**
    →dy/dx를 각 요소별로 나누어서 계산함: dy/dt, dx/dt

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

2.1.2 매개변수 함수(Parametric function)의 특징 (2)

- y = f(x)의 양함수식
  - 원래의 양함수 곡선을 회전,이동 등을 통하여 변형한 후, 이를 다시 양함수 곡선으로 표현* 하기 어려움

    *차원확장



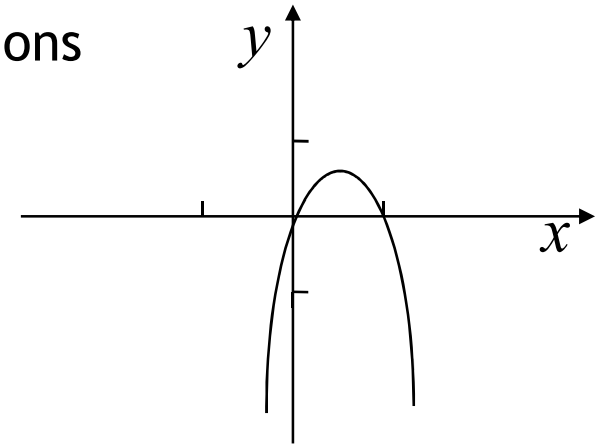
- f(x, y) = 0의 음함수식
  - 곡선 상의 점을 순차적으로 계산할 수 없다
  - 차원 확장이 어렵다.

- x = f(t), y = g(t)의 매개 변수식
  - 매개 변수를 통해 곡선 상의 점을 순차적으로 계산할 수 있다.
  - 차원을 쉽게 확장할 수 있다.
    - ➡ CAD 시스템에서 매개 변수식을 많이 사용하는 이유

2.1.2 매개변수 함수(Parametric function)의 특징 (3)

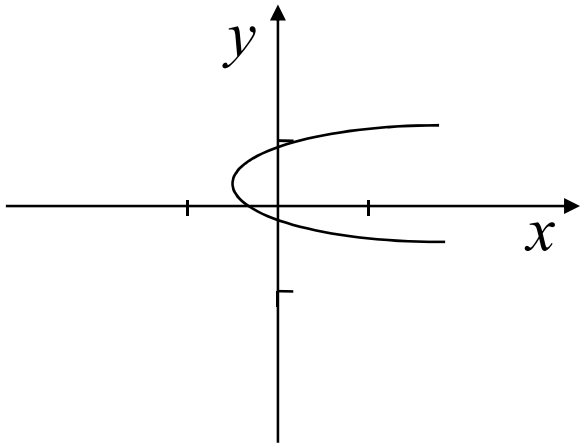
- The curve is defined by parametric functions

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



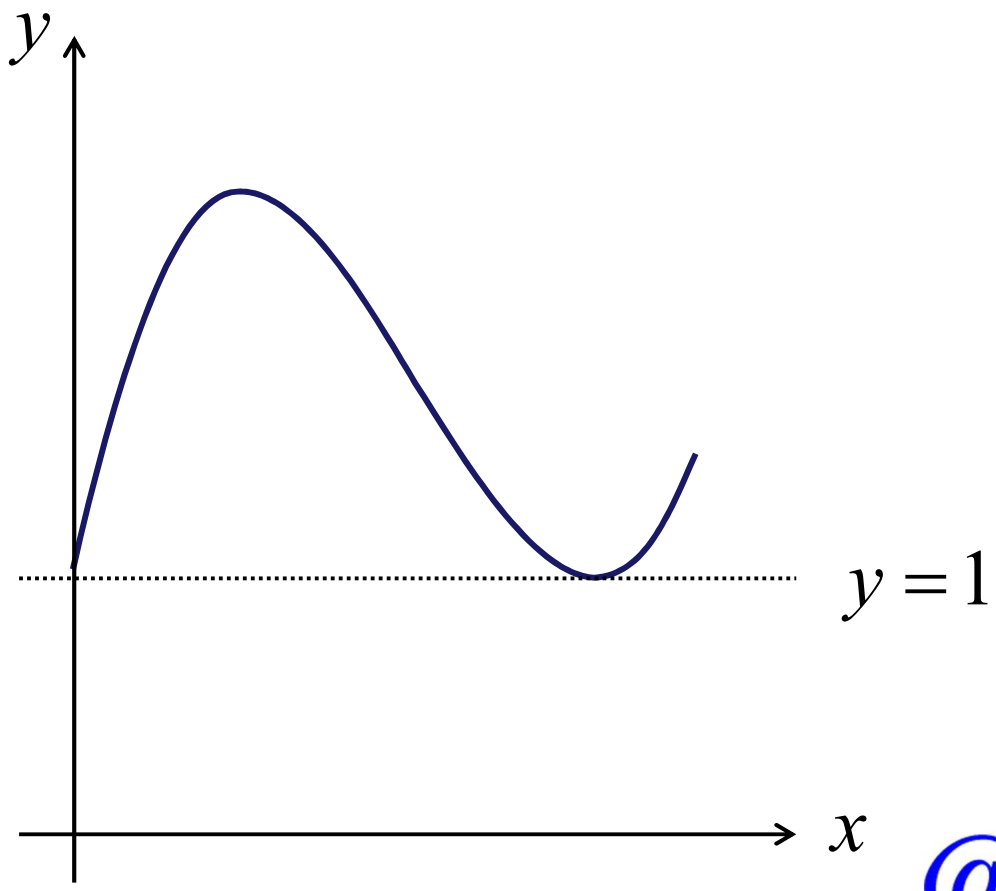
- If the curve is rotated by 90°,
  형상은 변하지 않고, 위치만 변함

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



2.1.3 일반 함수의 매개변수 함수 표현 (1)

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

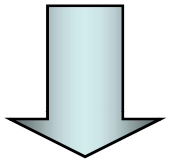


2.1.3 일반 함수의 매개변수 함수 표현 (2)

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

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

- 이러한 함수식과 계수 2, -4, 2, 1 으로는 그래프의 모양을 "직관적" 으로 예상하기 어려움



- 함수식을 아래와 같은 형태로 표현할 수 있다면

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

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

2.1.3 일반 함수의 매개변수 함수 표현 (3)

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

상수의 계수:	$x_0 = 0$	$x_0 = 0$
t의 계수:	$-3x_0 + 3x_1 = 1$	$x_1 = 1/3$
t^2의 계수:	$3x_0 - 6x_1 + 3x_2 = 0$	$x_2 = 2/3$
t^3의 계수:	$-x_0 + 3x_1 - 3x_2 + x_3 = 0$	$x_3 = 1$

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

Linear Precision

Gerald E. Farin, The Essentials of CAGD, 2000, p. 29.

• *Linear precision*: If the control points $\mathbf{b}_1$ and $\mathbf{b}_2$ are evenly spaced on the straight line between $\mathbf{b}_0$ and $\mathbf{b}_3$, the cubic Bezier curve is the linear interpolant between $\mathbf{b}_0$ and $\mathbf{b}_3$.

2.1.3 일반 함수의 매개변수 함수 표현 (4)

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

상수의 계수: $y_0 = 1$

t의 계수: $-3y_0 + 3y_1 = 2$

t^2의 계수: $3y_0 - 6y_1 + 3y_2 = -4$

t^3의 계수: $-y_0 + 3y_1 - 3y_2 + y_3 = 2$

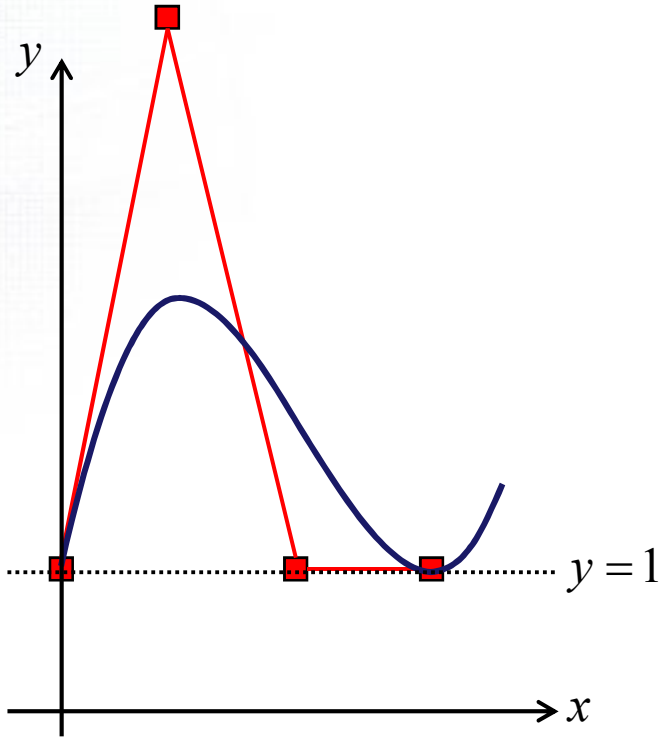
$$\Rightarrow \quad y_0 = 1,\ y_1 = 5/3,\ y_2 = 1,\ y_3 = 1$$

2.1.3 일반 함수의 매개변수 함수 표현 (5)

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

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

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



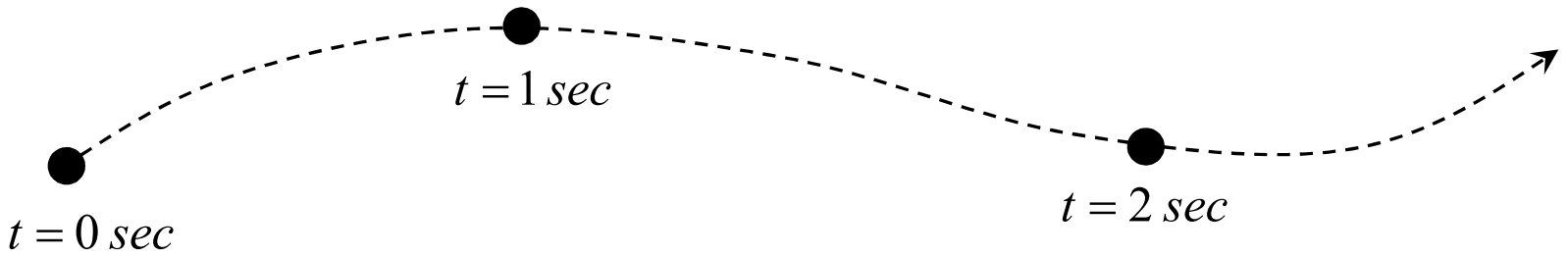
- 새로운 함수들에 곱해지는 계수들을 점으로 가시화 하면, 처음 점과 마지막 점은 곡선을 지나고,
- 점들을 직선으로 연결하면, 곡선의 모양과 비슷한 형태임을 알 수 있다.

2.1.3 일반 함수의 매개변수 함수 표현 (6)

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

- **매개변수 t를 시간이라고 생각하면, $\mathbf{r}(t)$는 어떠한 물체(rigid body)가 이동하는 궤적(trajectory)을 표현한 것으로 생각할 수 있다.**
- **양함수, 음함수 함수식에서는 이동하는 물체의 궤적의 형상만 표현할 수 있지만, 매개변수 함수식으로는 이동하는 물체의 궤적의 형상뿐만 아니라 특정시각 t 에서의 위치 $\mathbf{r}(t)$의 세부항목을 함께 살펴볼 수 있다.**

$\mathbf{r}(t)$: 물체의 궤적, $\dot{\mathbf{r}}(t)$: 물체의 속도, $\ddot{\mathbf{r}}(t)$: 물체의 가속도



2.1.4 매개 변수 함수의 '직관적' 표현 방법

- For any polynomial degree, on the cubic case $n = 3$

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

위와 같이 어떠한 함수도 매개 변수 형태로 표현할 수 있음. 그러나 r(t)가 다항식으로 표현되는 경우는 직관적으로 곡선의 형상을 예상하기 어려움

<문제 제기> 어떻게 하면 함수를 직관적으로 알아 볼 수 있게 표현할 수 있을까?

- The polynomial in terms of a combination of points;

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

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

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

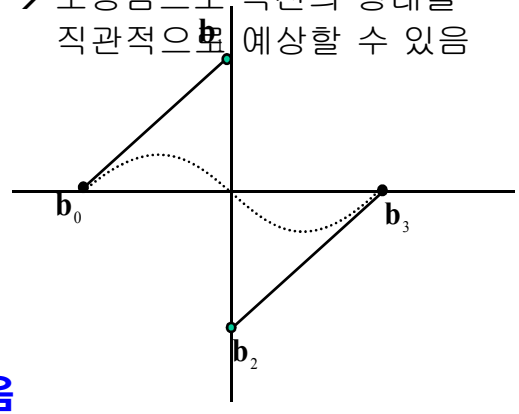
$$= B_0^3 \mathbf{b}_0 + B_1^3 \mathbf{b}_1 + B_2^3 \mathbf{b}_2 + B_3^3 \mathbf{b}_3$$

→ **프랑스 르노자동차의 엔지니어인 P. Bezier가 1971년 발견**

- 시작 조정점과 끝 조정점은 곡선상에 위치하고,

조정점들을 연결한 직선의 곡선의 모양과 비슷한 형상이다

→ 조정점으로 곡선의 형태를 직관적으로 예상할 수 있음



공간 상의 점(point)들을 3차 함수들과 "blending"하여 곡선을 표현할 수 있음
또한, 이러한 점을 움직이면 곡선의 형상이 변하므로, 곡선의 조정점(control points)이라고 함

2.2 Bezier curves

2.2.1 Definition & Characteristics of Bezier curves

2.2.2 Degree Elevation/Reduction of Bezier curves

2.2.3 de Casteljau algorithm

2.2.4 Bezier Curve Interpolation / Approximation

Advanced
Ship
Design
Automation
Laboratory

2.2.1 Definition & Characteristics of Bezier curves

2.2.1.1 Definition of Bezier curves
2.2.1.2 1st Derivatives of Cubic Bezier curves
2.2.1.3 Characteristics of Bezier curves
2.2.1.4 Higher order Bezier curves
2.2.1.5 1st Derivatives of higher order Bezier curves
2.2.1.6 Matrix form of Bezier curves
2.2.1.7 Sample code of Bezier curve class

2.2.1.1 Definition of cubic "Bezier" curves

- The cubic Bezier curve is defined by

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

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

linearly independent

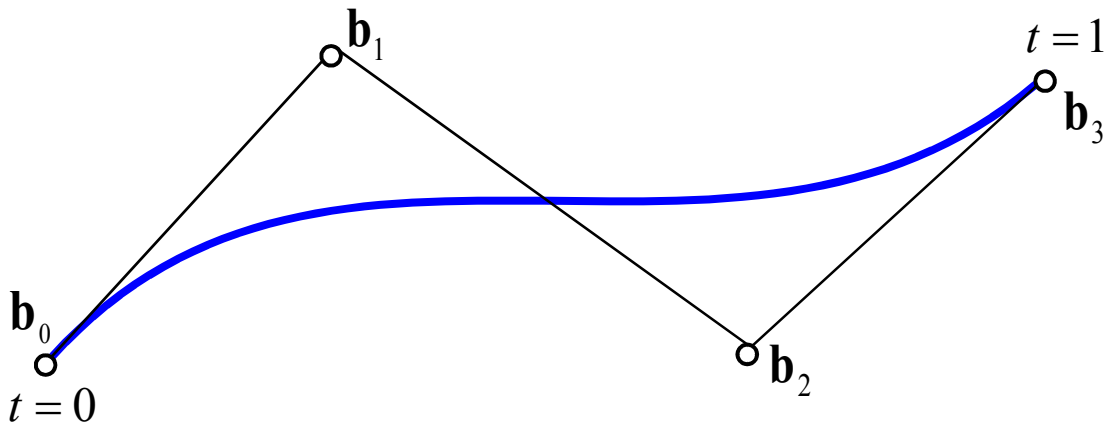
$$if \quad c_1 B_0^3(t) + c_2 B_1^3(t) + c_3 B_2^3(t) + c_4 B_3^3(t) = 0,$$
$$then \quad c_1 = c_2 = c_3 = c_4 = 0$$

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

$$\sum_{i=0}^{3} B_i^3(t) = 1, \quad B_i^3(t) \geq 0$$

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



2.2.1.2 1st Derivatives of Cubic Bezier Curves (1)

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

- First derivatives: Tangent vector of the curve
  : "시간 t에서의 물체의 속도"

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

$$= 3[\mathbf{b}_1 - \mathbf{b}_0](1-t)^2 + 6[\mathbf{b}_2 - \mathbf{b}_1](1-t)t + 3[\mathbf{b}_3 - \mathbf{b}_2]t^2$$
$$= 3\Delta\mathbf{b}_0(1-t)^2 + 6\Delta\mathbf{b}_1(1-t)t + 3\Delta\mathbf{b}_2 t^2$$
$$= 3(\Delta\mathbf{b}_0 B_0^2 + \Delta\mathbf{b}_1 B_1^2 + \Delta\mathbf{b}_2 B_2^2)$$

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

2.2.1.2 1st Derivatives of Cubic Bezier Curves(2)

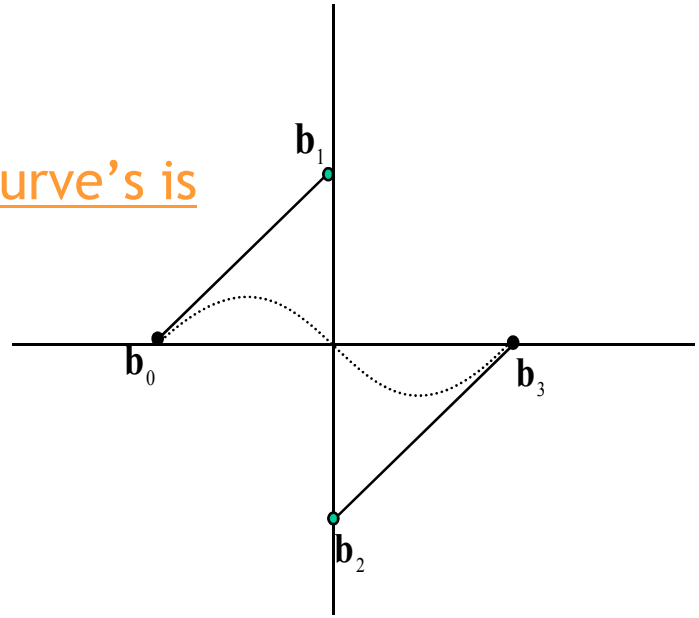
- The derivative of the cubic curve is quadratic curve.

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

  - where, B_i^2 : quadratic Bernstein basis function.

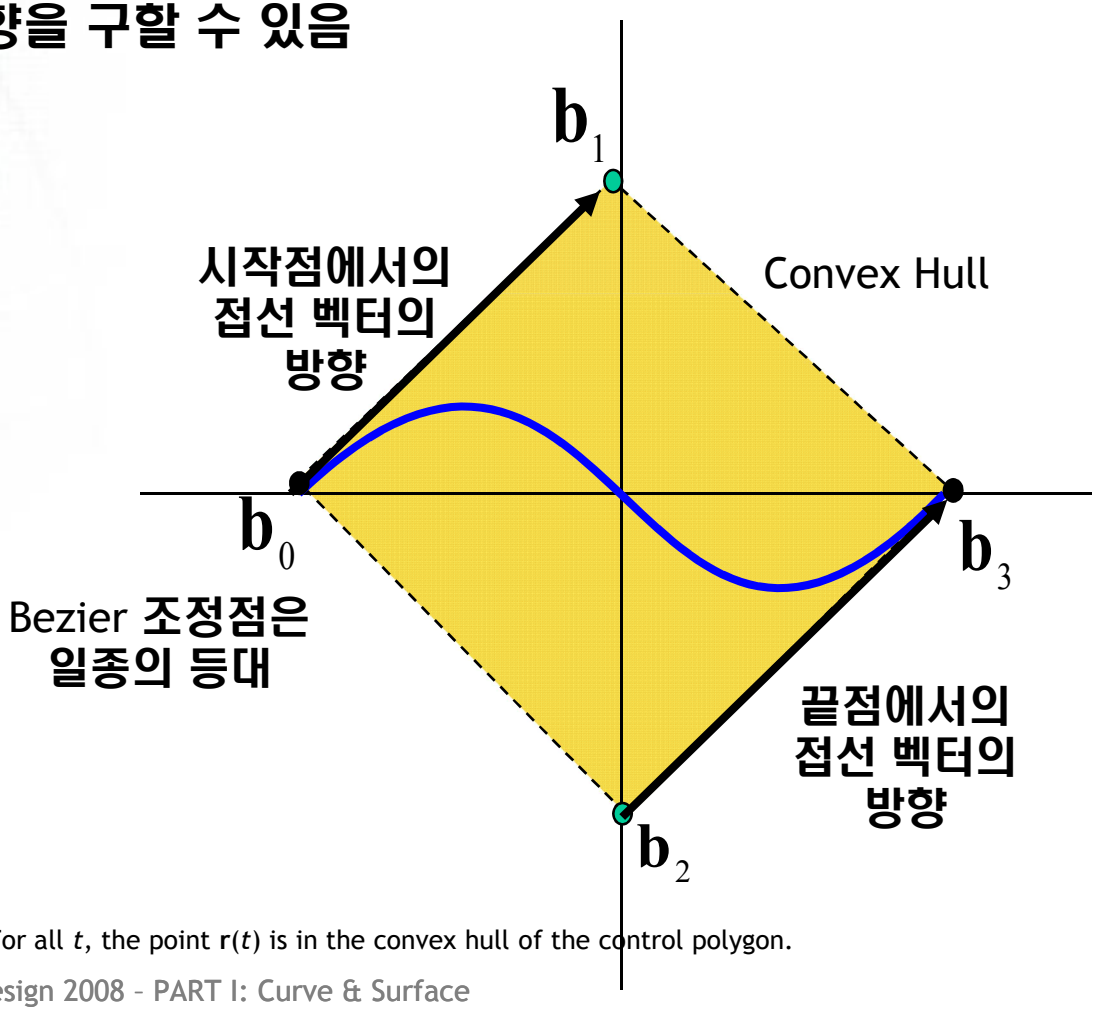
- Most important tangent vectors at the curve's is Endpoints tangent vectors:

$$\dot{\mathbf{r}}(0) = 3\Delta\mathbf{b}_0 = 3(\mathbf{b}_1 - \mathbf{b}_0),$$
$$\dot{\mathbf{r}}(1) = 3\Delta\mathbf{b}_2 = 3(\mathbf{b}_3 - \mathbf{b}_2)$$



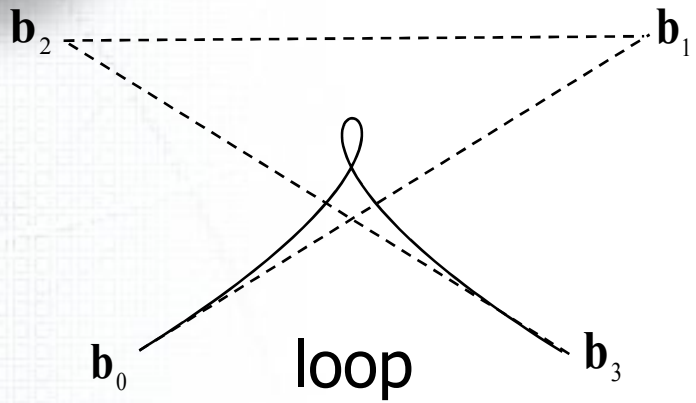
2.2.1.3 Characteristics of Bezier Curves (1)

- Bezier 곡선은 최외각 조정점들로 구성되는 Convex Hull 내에 존재함[1] $\left(\because \sum_{i=0}^{3} B_i^3(t) = 1\right)$
- 처음 두 개의 조정점과 마지막 두 개의 조정점으로부터 시작점 및 끝점에서의 접선 벡터의 방향을 구할 수 있음

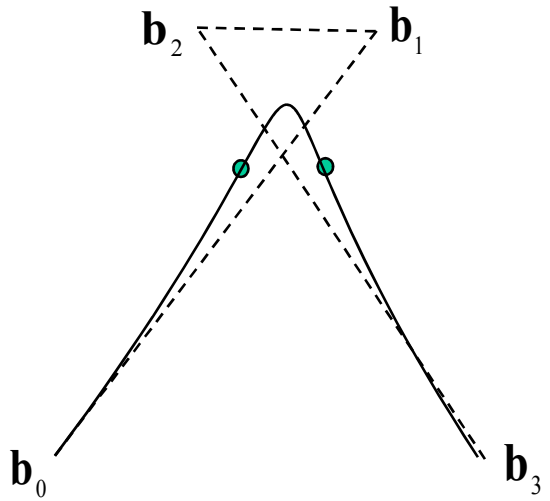


1) *Convex Hull Property*: For all t, the point $\mathbf{r}(t)$ is in the convex hull of the control polygon.

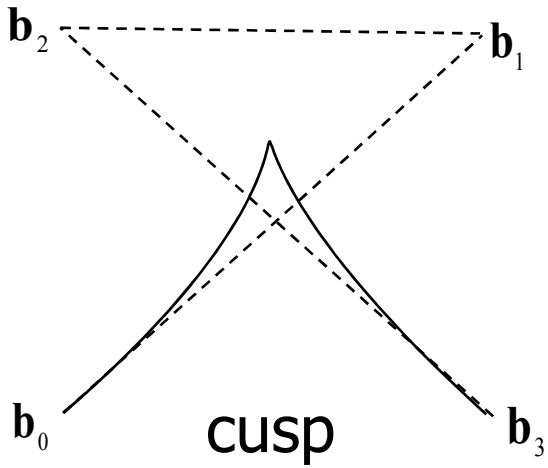
2.2.1.3 Characteristics of Bezier Curves (2)



loop



Two inflection points



cusp

2.2.1.4 Higher order Bezier Curves (1)

- 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.

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

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

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

- For cubic case, the Bezier curve as:

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

2.2.1.4 Higher order Bezier Curves (2)

- Bernstein Polynomial Function:

$$\begin{aligned}[(1-t)+t]^2 &= (1-t)^2 + 2(1-t)t + t^2 \\ &= B_0^2(t) + B_1^2(t) + B_2^2(t),\end{aligned}$$

$$\begin{aligned}[(1-t)+t]^3 &= [(1-t)+t]^2[(1-t)+t] \\ &= (1-t)^3 + 3(1-t)^2 t + 3(1-t)t^2 + t^3 \\ &= B_0^3(t) + B_1^3(t) + B_2^3(t) + B_3^3(t),\end{aligned}$$

$$\begin{aligned}[(1-t)+t]^4 &= [(1-t)+t]^3[(1-t)+t] \\ &= (1-t)^4 + 4(1-t)^3 t + 6(1-t)^2 t^2 + 4(1-t)t^3 + t^4 \\ &= B_0^4(t) + B_1^4(t) + B_2^4(t) + B_3^4(t) + B_4^4(t)\end{aligned}$$

1

1 1

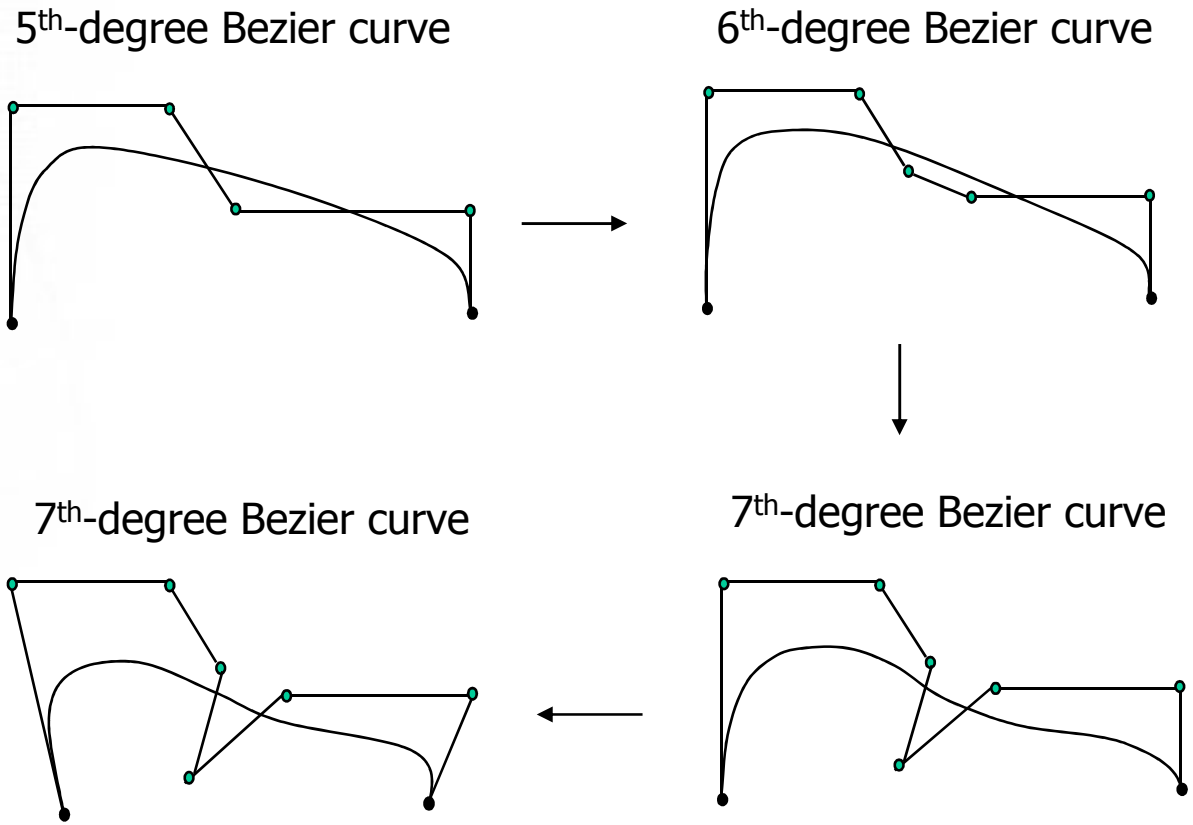
1 2 1

1 3 3 1

1 4 6 4 1

파스칼의 삼각형

2.2.1.4 Higher order Bezier Curves (3)



2.2.1.5 Derivatives of Higher Order Bezier Curves (1)

- 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'.

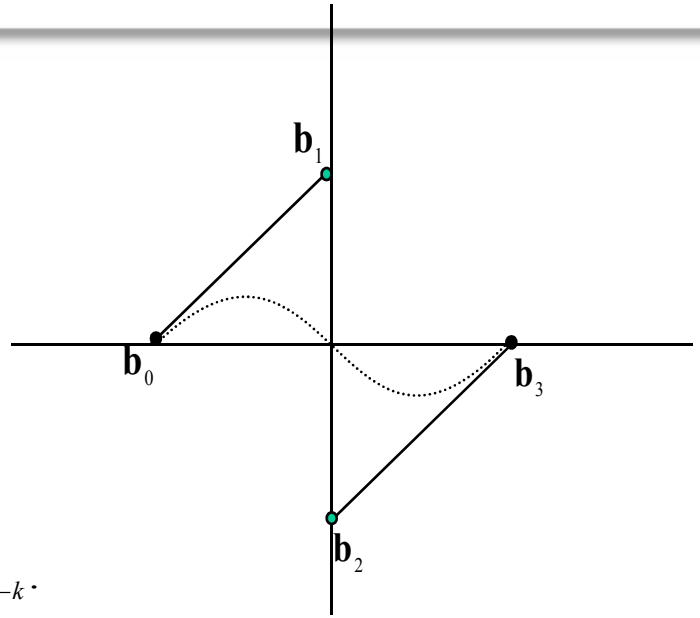
- For the k^{th} times derivative:

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

2.2.1.5 Derivatives of Higher Order Bezier Curves (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;

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

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

2.2.1.6 Matrix form of Bezier curves(1)

- 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;

$$\mathbf{r}(t) = \begin{bmatrix}\mathbf{b_0} & \mathbf{b_1} & \mathbf{b_2} & \mathbf{b_3}\end{bmatrix}\begin{bmatrix}(1-t)^3 \\ 3(1-t)^2 t \\ 3(1-t)\,t^2 \\ t^3\end{bmatrix}$$

2.2.1.6 Matrix form of Bezier curves(2)

- The Matrix form of Bezier Curve is

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

Conversion to the monomial form: $\mathbf{r}(t)=\mathbf{a}_0+\mathbf{a}_1t+\mathbf{a}_2t^2+\mathbf{a}_3t^3$

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

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

2.2.1.6 Matrix form of Bezier curves(3)

- The Matrix form of Monomial Curve is

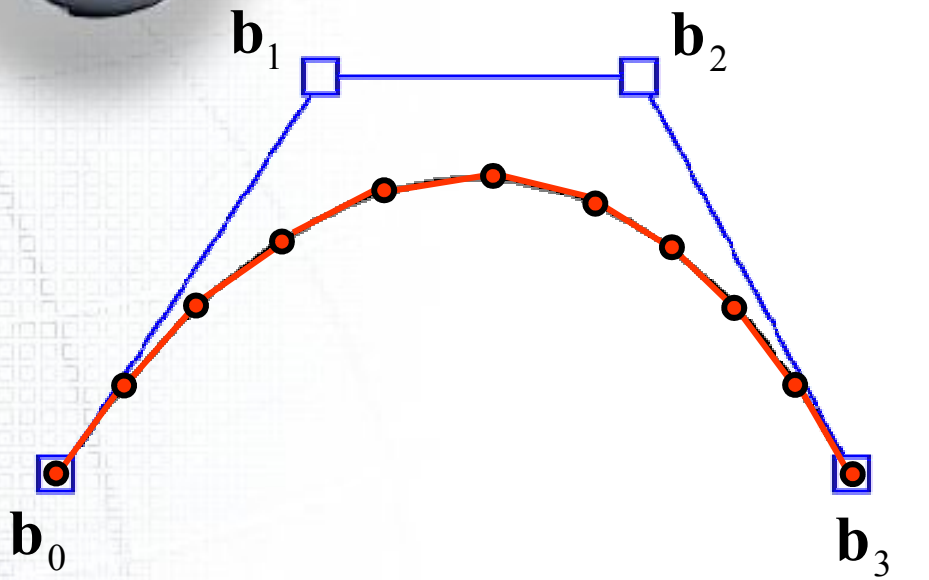
$$\mathbf{r}(t) = \mathbf{a}_0 + \mathbf{a}_1 t + \mathbf{a}_2 t^2 + \mathbf{a}_3 t^3 = \begin{bmatrix} \mathbf{a}_0 & \mathbf{a}_1 & \mathbf{a}_2 & \mathbf{a}_3 \end{bmatrix} \begin{bmatrix} 1 \\ t \\ t^2 \\ t^3 \end{bmatrix}$$

Conversion to the Bezier form:

$$\begin{aligned} \mathbf{r}(t) &= (1-t)^3 \mathbf{b}_0 + 3(1-t)^2 t \mathbf{b}_1 + 3(1-t) t^2 \mathbf{b}_2 + t^3 \mathbf{b}_3 \\ &= B_0^3(t)\,\mathbf{b}_0 + B_1^3(t)\,\mathbf{b}_1 + B_2^3(t)\,\mathbf{b}_2 + B_3^3(t)\,\mathbf{b}_3 \\ &= \begin{bmatrix} \mathbf{b}_0 & \mathbf{b}_1 & \mathbf{b}_2 & \mathbf{b}_3 \end{bmatrix} \begin{bmatrix} 1 & -3 & 3 & -1 \\ 0 & 3 & -6 & 3 \\ 0 & 0 & 3 & -3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ t \\ t^2 \\ t^3 \end{bmatrix} \end{aligned}$$

$$\therefore \begin{bmatrix} \mathbf{b}_0 & \mathbf{b}_1 & \mathbf{b}_2 & \mathbf{b}_3 \end{bmatrix} = \begin{bmatrix} \mathbf{a}_0 & \mathbf{a}_1 & \mathbf{a}_2 & \mathbf{a}_3 \end{bmatrix} \begin{bmatrix} 1 & -3 & 3 & -1 \\ 0 & 3 & -6 & 3 \\ 0 & 0 & 3 & -3 \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1}$$

2.2.1.7 Programming 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) Bezier Curve의 구성

- Degree
- Control Point

Member Variables of Bezier Curve Class

int n: degree of Bezier Curve
Vector* m_ControlPoint: Control Point
int m_nControlPoint: the number of Control Point

2) Bernstein Polynomial 계산

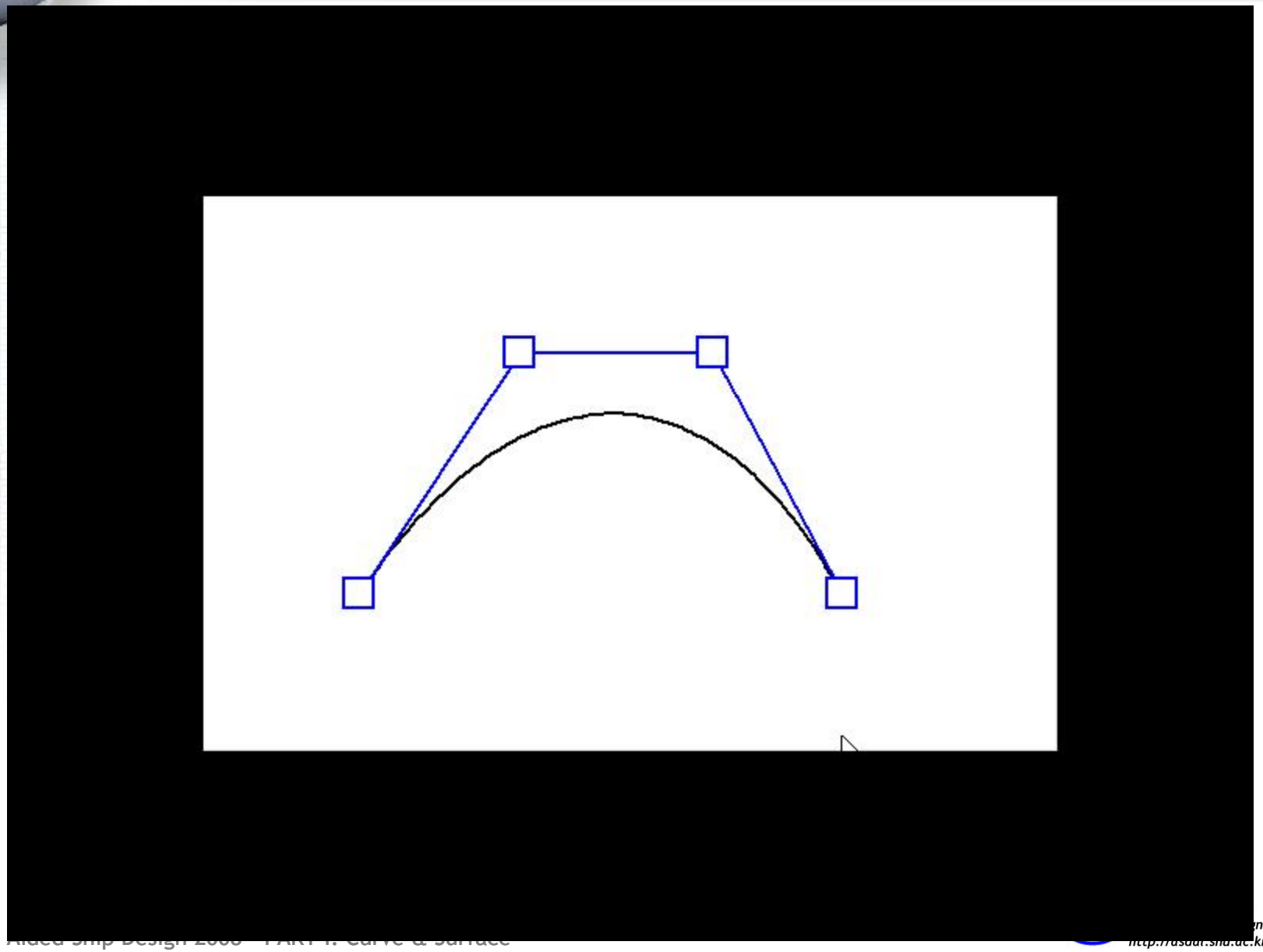
$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i},$$

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

3) Bezier Curve 작도

- 곡선을 Line Segment로 나누어 작도
- Parameter t를 0~1까지 n등분하여 각 t에 대한 곡선 상의 점을 구함
- 위에서 구한 점을 직선으로 연결하여 곡선을 표현

2.2.1.7 Programming Bezier Curve class



2.2.1.7 Sample code of Bezier Curve class(1)

```
#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 nControlPoint);
   Vector CalcPoint(double t);
   double B (int i, double t);                    // Bernstein Polynomial
};
#endif
```

Member Variables

int n: degree of Bezier Curve
Vector* m_ControlPoint: Control Point
int m_nControlPoint: the number of Control Point

2.2.1.7 Sample code of Bezier Curve class(2)

```
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;
}
```

2.2.1.7 Sample code of Bezier Curve class(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;
}
```

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

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

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

2.2.2 Degree Elevation / Reduction of Bezier curves

2.2.2.1 Degree Elevation

2.2.2.2 Degree Reduction

2.2.2.3 Repeated Degree Elevation

Advanced
Ship
Design
Automation
Laboratory

2.2.2.1 Degree Elevation (1)

- **목적**
  - **서로 다른 차수의 곡선을 같은 차수로 연결할 때 사용**
    (3**차** Bezier **곡선** + 4**차** Bezier **곡선** → 4**차** Bezier **곡선** + 4**차** Bezier **곡선**)
  - **보다 많은 조정점으로 곡선을 보다 자유롭게 설계**
    (Bezier **조정점의 개수** = **차수**+1)

- **참고**:

  n **차** B-spline **곡선은 주어진 놋트 상에서 부드럽게 연결된**
  n **차** ***Bezier 곡선의 집합*** **이라고 볼 수 있다.**
  따라서, B-spline **곡선의 차수 증가 방법은**
  Bezier **곡선의 차수 증가 방법으로 설명될 수 있다.**

2.2.2.1 Degree Elevation (2)

- 2차 Bézier curve -> 3차 Bézier curve

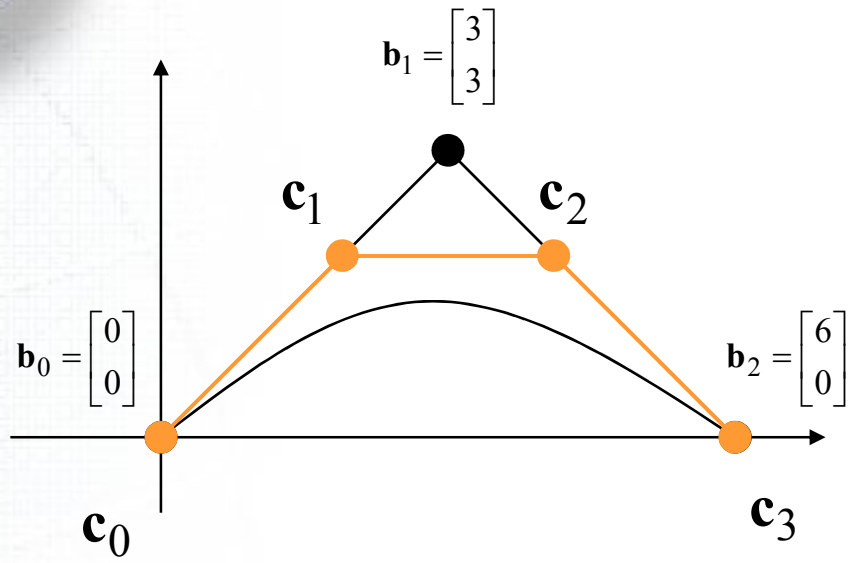
$$\mathbf{r}(t) = (1-t)^2\mathbf{b}_0 + 2(1-t)t\mathbf{b}_1 + t^2\mathbf{b}_2 \quad \times [t + (1-t)]$$

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

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

⇨ 즉, ⬚를 새로운 control point로 갖는 3차 Bézier curve

2.2.2.1 Degree Elevation (3)



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

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

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

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

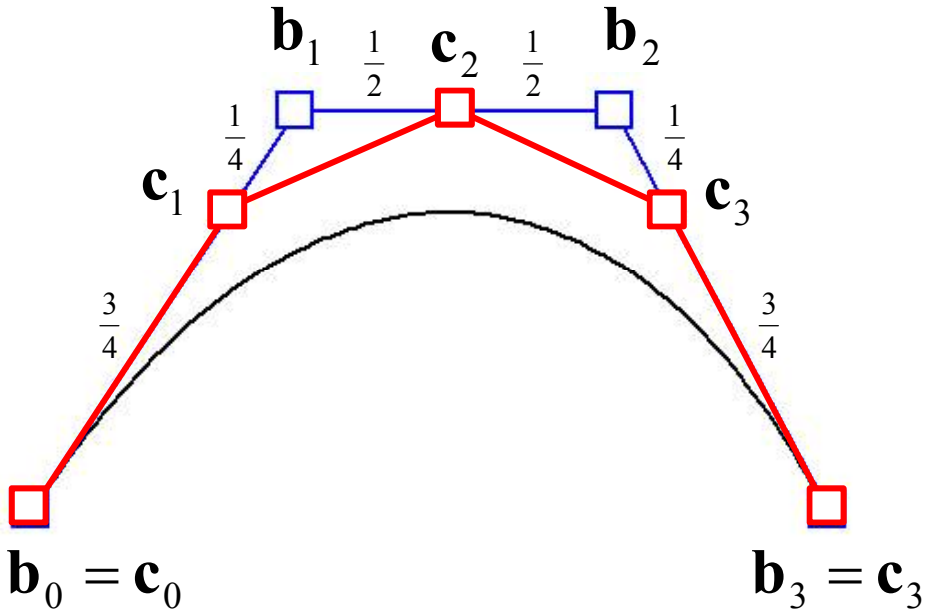
$$\mathbf{r}(t) = (1-t)^2\mathbf{b}_0 + 2(1-t)t\mathbf{b}_1 + t^2\mathbf{b}_2$$

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

2.2.2.1 Degree Elevation (4)

- $\mathbf{b}_0,\ldots,\mathbf{b}_n$ 를 control point로 가지는 n차 Bézier curve를 n+1차로 degree elevation하면

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



Degree Elevation: 3차 → 4차

2.2.2.1 Degree Elevation (5)

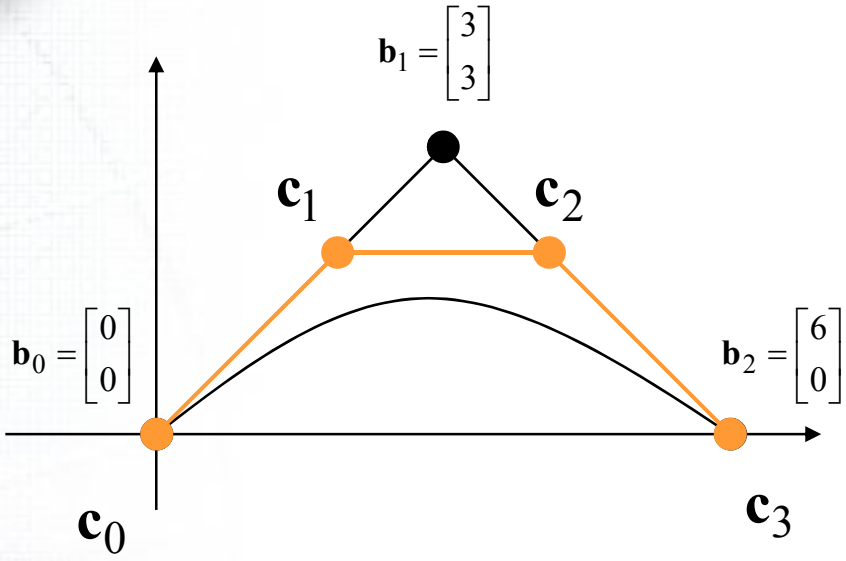
- $\mathbf{b}_0,\ldots,\mathbf{b}_n$ 를 control point로 가지는 n차 Bézier curve를 n+1차로 degree elevation하면

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

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

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

2.2.2.1 Degree Elevation (6)



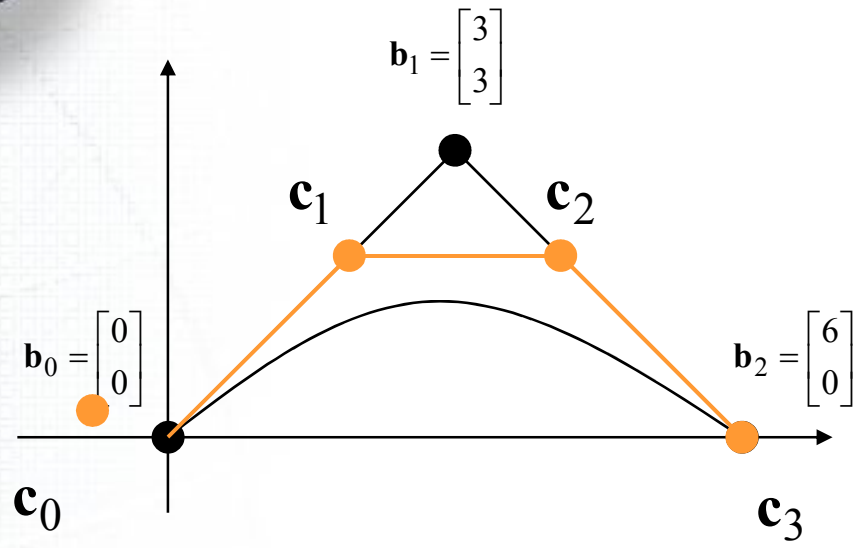
$$\mathbf{c}_0 = \mathbf{b}_0,$$
$$\mathbf{c}_1 = [\frac{1}{3}\mathbf{b}_0 + \frac{2}{3}\mathbf{b}_1],$$
$$\mathbf{c}_2 = [\frac{2}{3}\mathbf{b}_1 + \frac{1}{3}\mathbf{b}_2],$$
$$\mathbf{c}_3 = \mathbf{b}_2$$

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

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

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

2.2.2.2 Degree Reduction



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

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

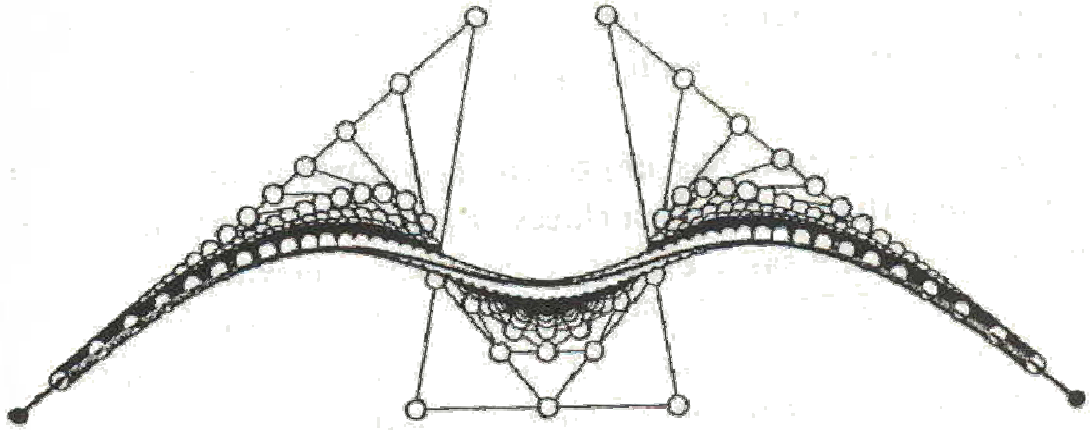
$$\mathbf{D} = \begin{bmatrix} 1 & 0 & 0 \\ 1/3 & 2/3 & 0 \\ 0 & 2/3 & 1/3 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{C} = \begin{bmatrix} 0 & 0 \\ 2 & 2 \\ 4 & 2 \\ 6 & 0 \end{bmatrix}$$

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

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

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

2.2.2.3 Repeated Degree Elevation



무한히 반복하면 polygon이 curve에 근접해간다.

2.2.3 de Casteljau algorithm

2.2.3.1 de Casteljau algorithm & Bezier curves

2.2.3.2 Parameter Transformation

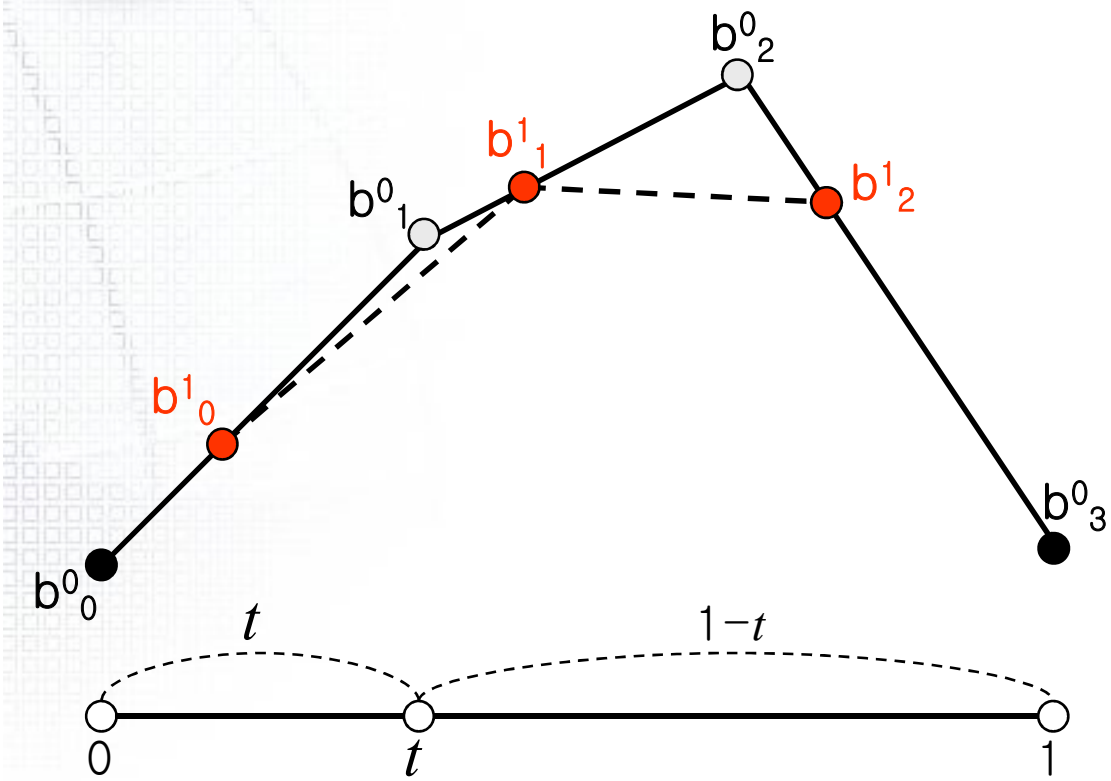
2.2.3.3 Linear Interpolation on [a,b]

2.2.3.4 de Casteljau algorithm at $u=u_1$ of $[u_0, u_2]$

2.2.3.5 de Casteljau algorithm**의 특징**

2.2.3.6 Sample code of de Casteljau algorithm

2.2.3.1 de Casteljau algorithm & Bezier curves (1)



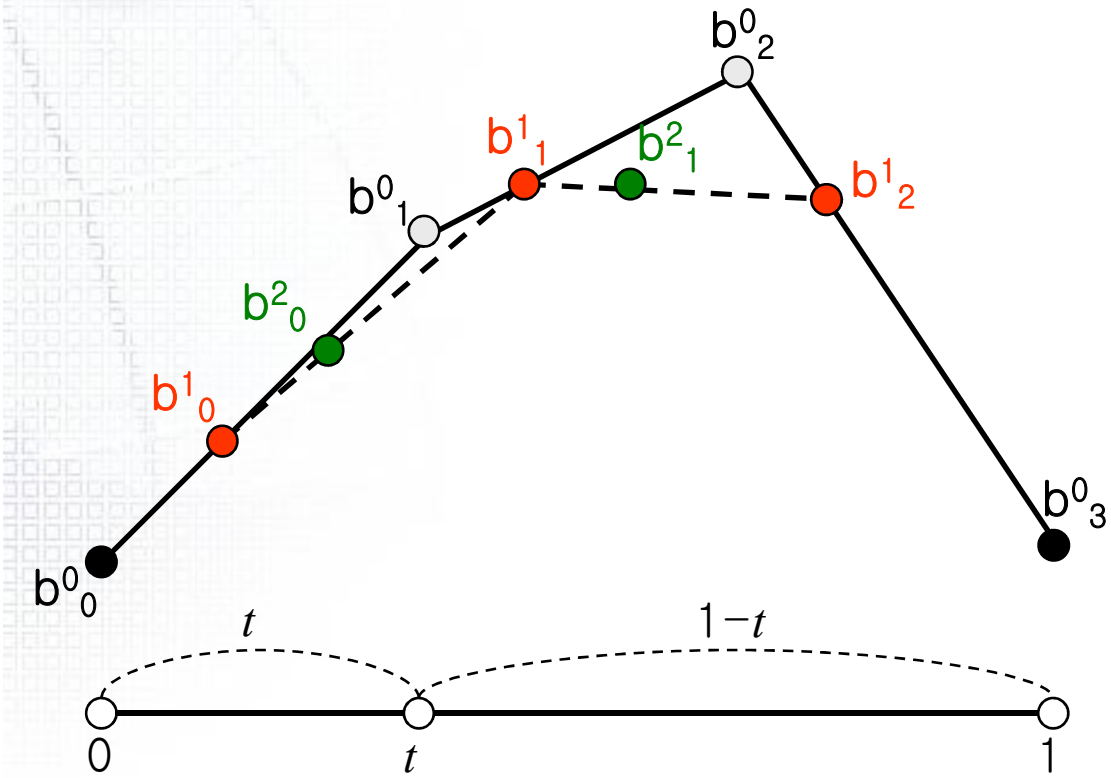
Linear interpolation

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

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

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

2.2.3.1 de Casteljau algorithm & Bezier curves (2)

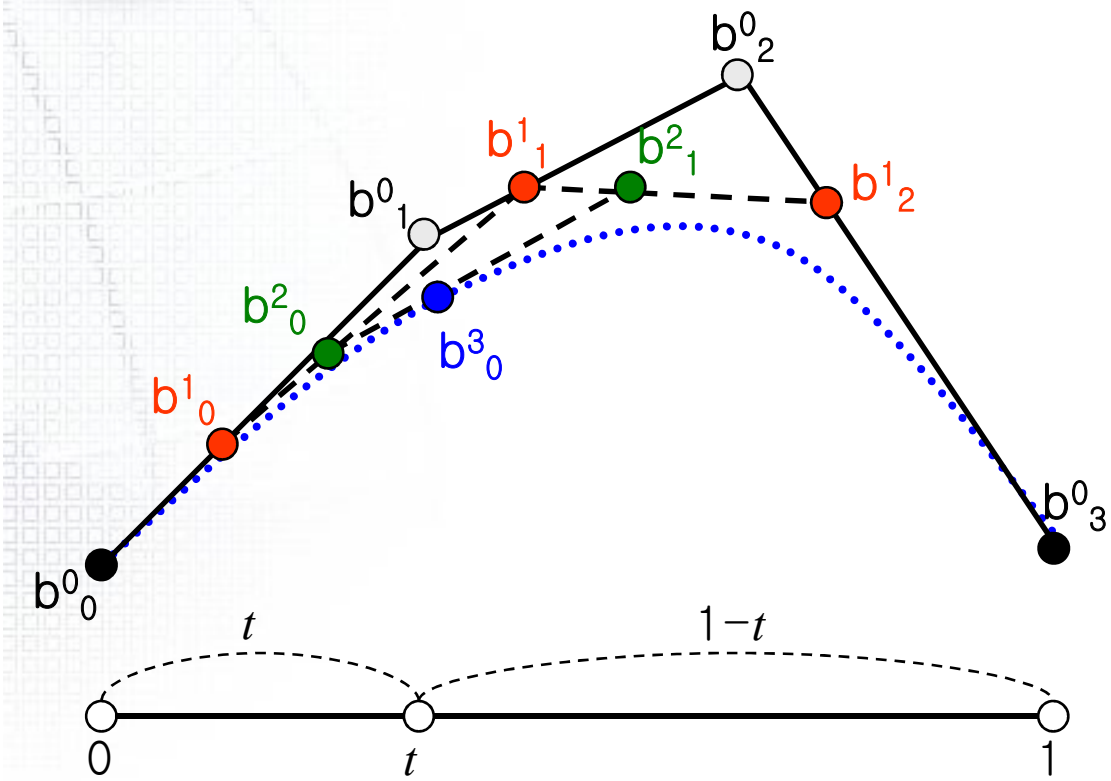


Linear interpolation

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

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

2.2.3.1 de Casteljau algorithm & Bezier curves (3)



Linear interpolation

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

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

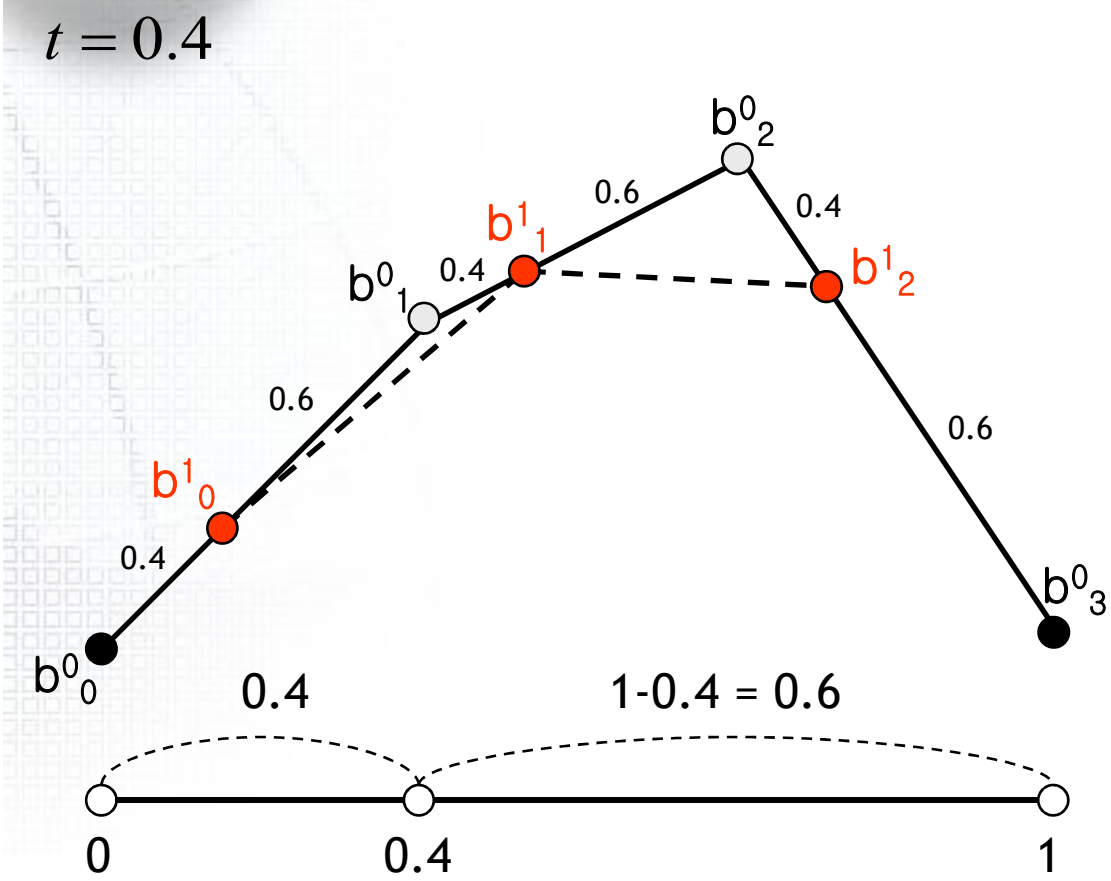
$$\mathbf{b}_0^3(t) = (1-t)\mathbf{b}_0^2 + t\mathbf{b}_1^2$$

3차 Bezier curves 와 동일한 함수식 !!!

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

2.2.3.1 Example of de Casteljau algorithm (1)

- de Casteljau algorithm at t = 0.4



Linear interpolation

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

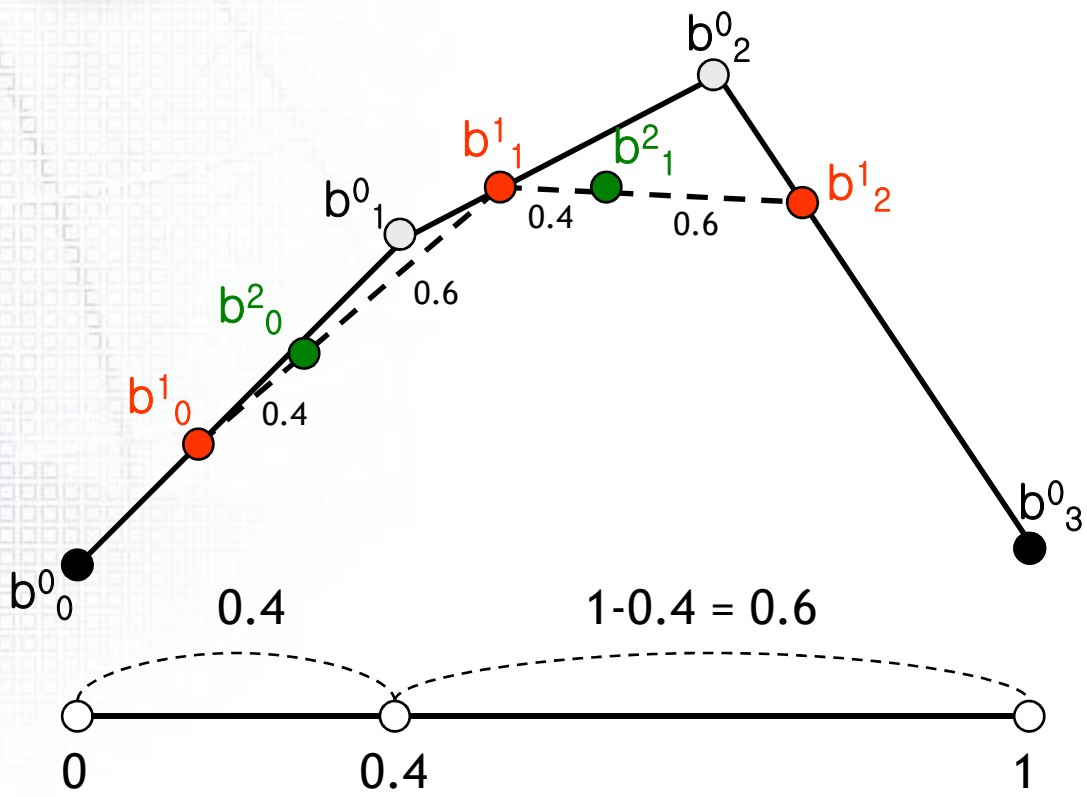
$$\mathbf{b}_1^1(0.4) = (1-0.4)\mathbf{b}_1^0 + 0.4\mathbf{b}_2^0$$

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

2.2.3.1 Example of de Casteljau algorithm (2)

- de Casteljau algorithm at t = 0.4

$t = 0.4$



Linear interpolation

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

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

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

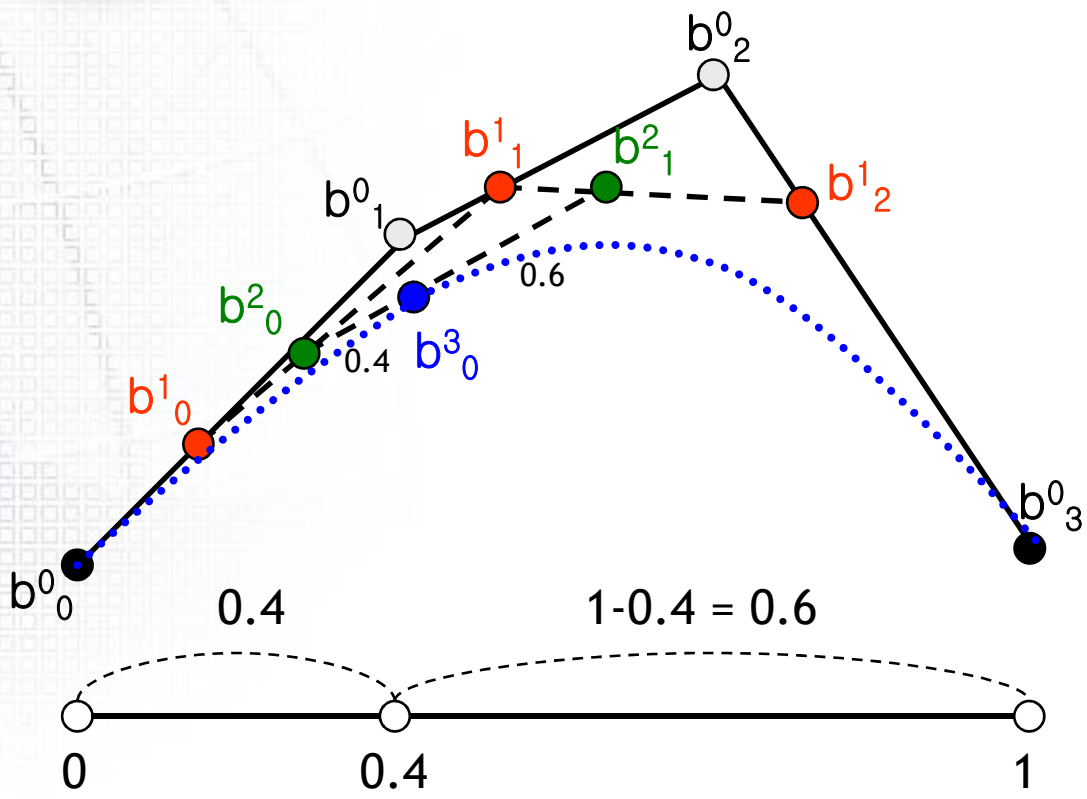
$$\mathbf{b}_0^2(0.4) = (1-0.4)\,\mathbf{b}_0^1 + 0.4\mathbf{b}_1^1$$

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

2.2.3.1 Example of de Casteljau algorithm (3)

- de Casteljau algorithm at t = 0.4

$t = 0.4$



Linear interpolation

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

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

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

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

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

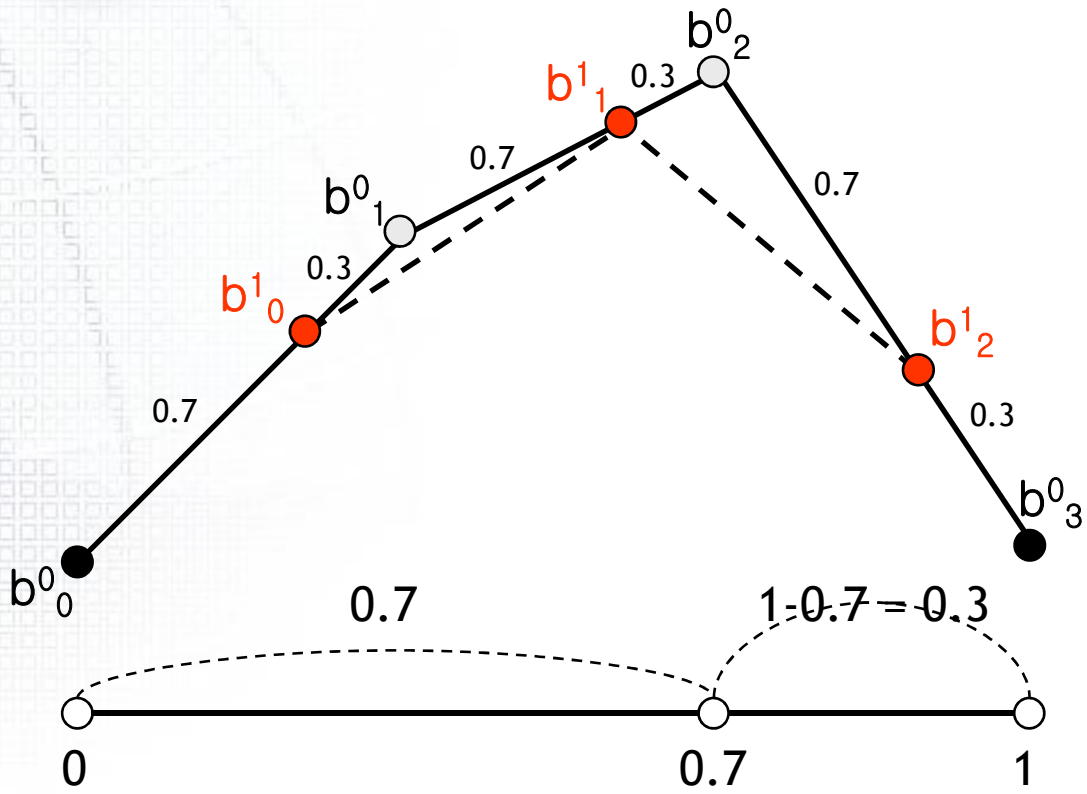
$$\mathbf{b}_0^3(0.4) = (1-0.4)\,\mathbf{b}_0^2 + 0.4\mathbf{b}_1^2$$

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

2.2.3.1 Example of de Casteljau algorithm (1)

- de Casteljau algorithm at t = 0.7

$t = 0.7$



Linear interpolation

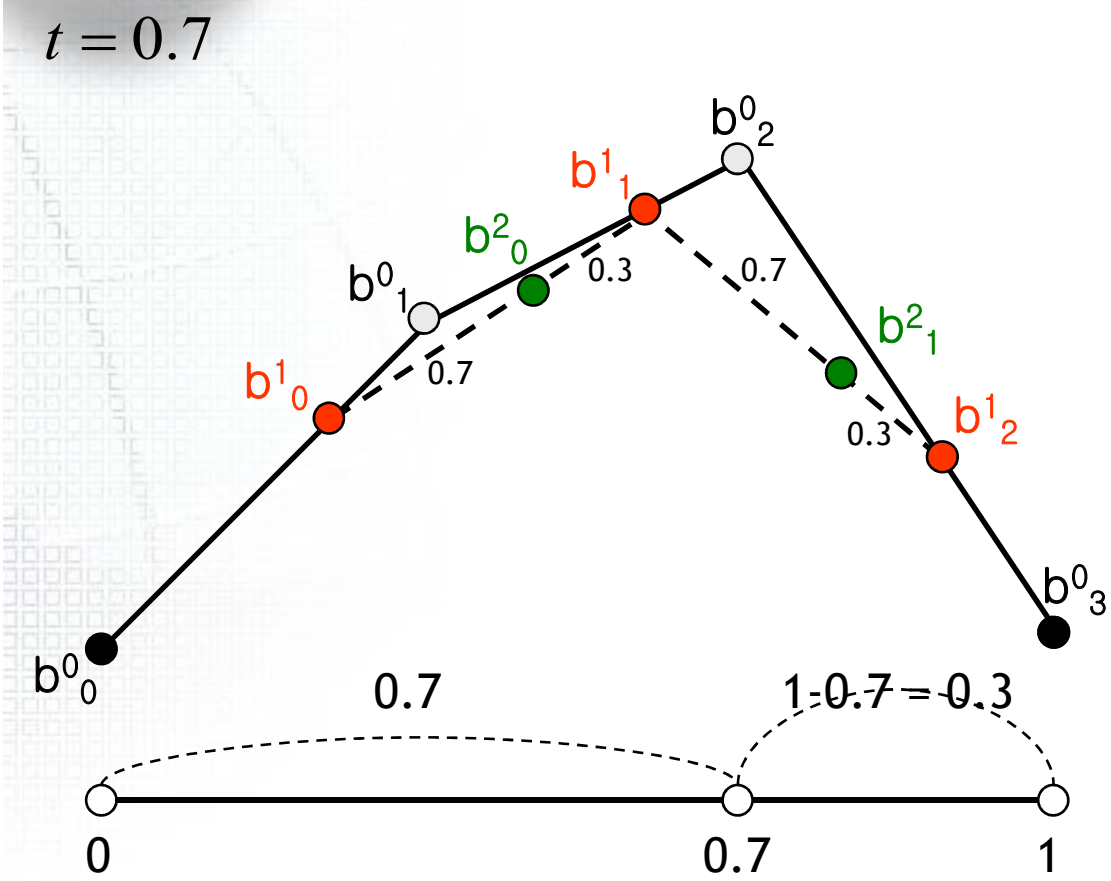
$$\mathbf{b}_0^1(0.7) = (1-0.7)\mathbf{b}_0^0 + 0.7\mathbf{b}_1^0$$

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

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

2.2.3.1 Example of de Casteljau algorithm (2)

- de Casteljau algorithm at t = 0.7



Linear interpolation

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

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

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

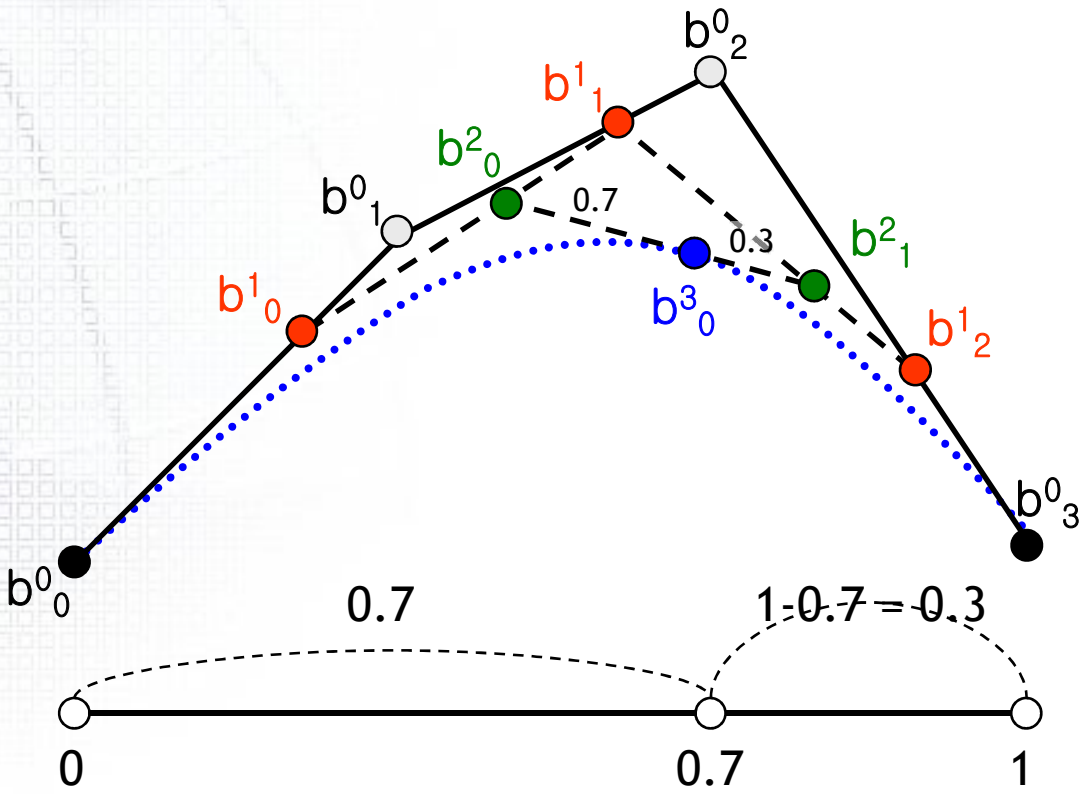
$$\mathbf{b}_0^2(0.7) = (1-0.7)\,\mathbf{b}_0^1 + 0.7\mathbf{b}_1^1$$

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

2.2.3.1 Example of de Casteljau algorithm (3)

- de Casteljau algorithm at t = 0.7

$t = 0.7$



Linear interpolation

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

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

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

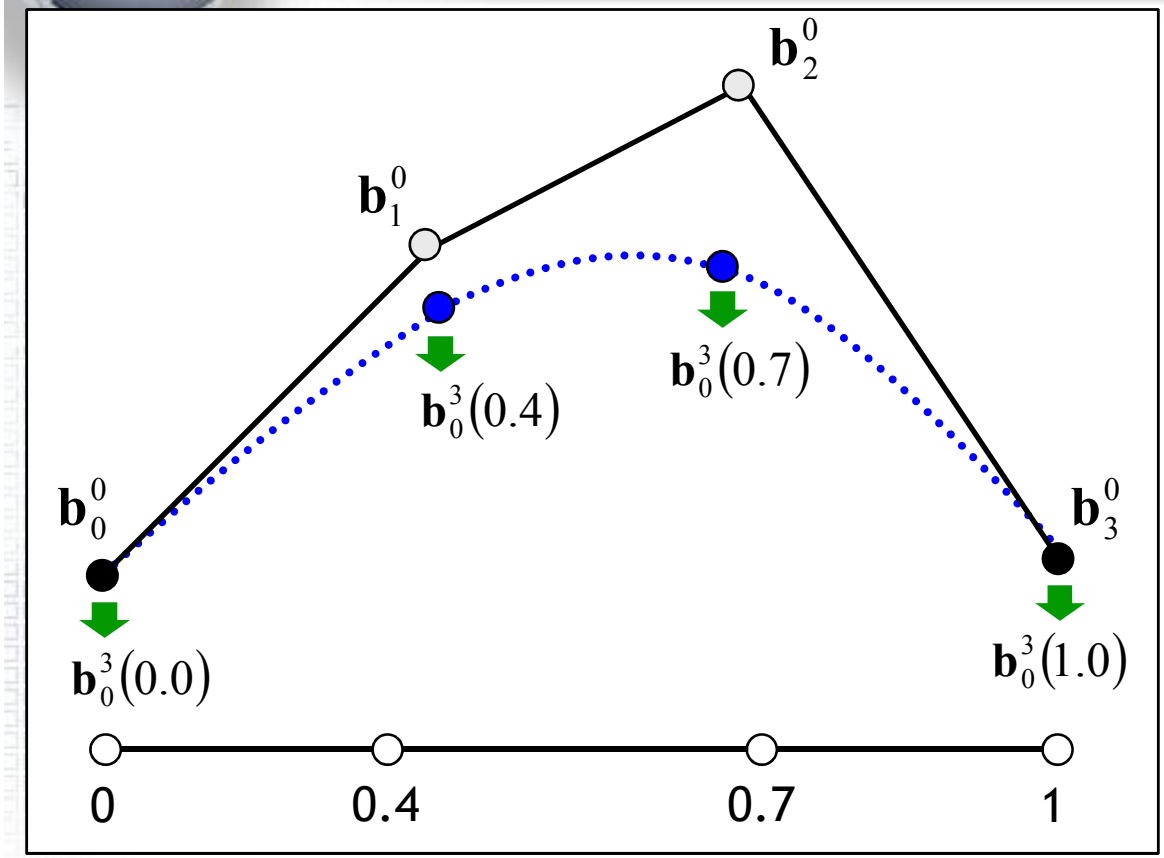
$$\mathbf{b}_0^2(0.7) = (1-0.7)\,\mathbf{b}_0^1 + 0.7\mathbf{b}_1^1$$

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

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

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

2.2.3.1 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

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

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

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

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

2.2.3.1 de Casteljau algorithm & Bezier curves (4)

- de Casteljau **알고리즘**: "Constructive Approach"

  Input: $\mathbf{b}_i$ (Bezier control points)

  Processor: n**번 순차적** 'linear interpolation'

  Output : n**차 곡선상의 점**

  → Bernstein basis function(polynomial) **형태로 표현 됨**

- Bezier **곡선식**: "Bernstein Function evaluation Approach"

  Input: $\mathbf{b}_i$ (Bezier control points)

  Processor: **공간 상의 점** $\mathbf{b}_i$ **와** Bernstein Basis function**을 "blending"하여 함수를 값을 계산하면 곡선상의 점을 구할 수 있음**

  Output: Bernstein basis function(polynomial) **과** $\mathbf{b}_i$ **의 혼합 함수 형태로 표현**

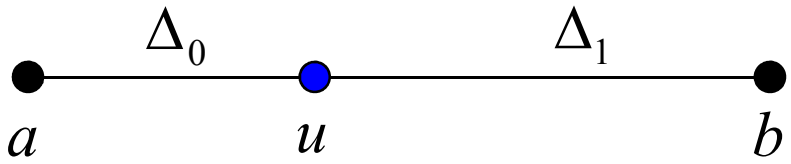
2.2.3.2 Parameter Transformation

- Parameter Transformation
- the affine map for the interval of $t \in [0,1] \rightarrow u \in [a,b]$,
- We get

$$t = \frac{u-a}{b-a}. \quad \text{(or)} \quad 1-t = \frac{b-u}{b-a}.$$

- u → global parameter, t → local parameter
- the process of changing interval is called parameter transformation.

2.2.3.3 Linear Interpolation on [a, b]



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

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

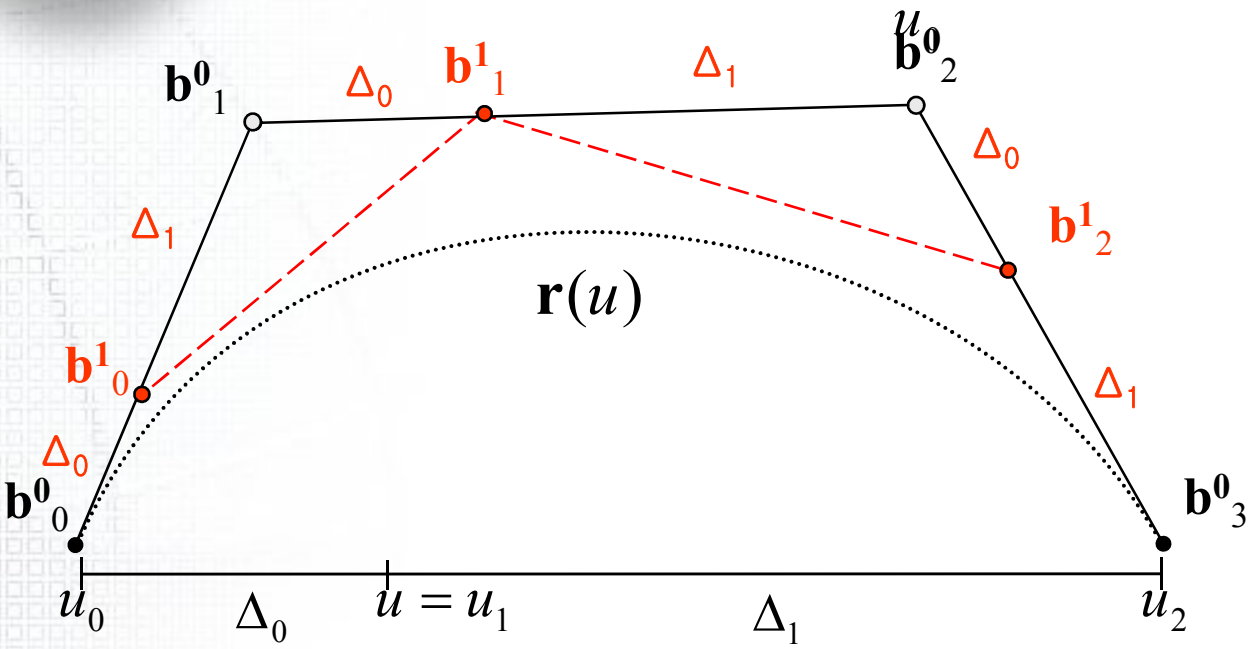
$$(\Delta_0+\Delta_1)u=\Delta_1 a+\Delta_0 b$$

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

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

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

2.2.3.4 매개변수 구간이 [u_0 , u_2]인 경우, $u = u_1$ 에서의 곡선상의 점 구하기: de Casteljau Algorithm



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

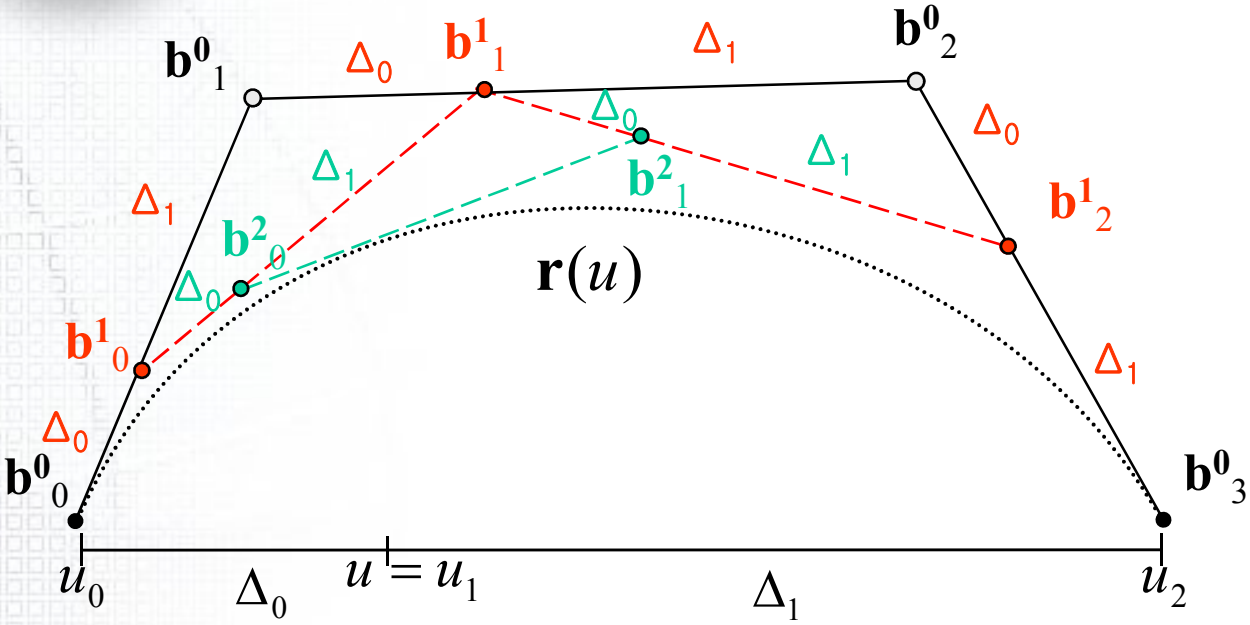
$$\mathbf{b}^1_1(u) = \frac{\Delta_1}{\Delta}\mathbf{b}^0_1 + \frac{\Delta_0}{\Delta}\mathbf{b}^0_2$$

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

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

$$\text{ratio}(b^2_0, b^3_0, b^2_1) = \frac{u - u_0}{u_2 - u} = \frac{\Delta_0}{\Delta_1}$$

2.2.3.4 매개변수 구간이 [u_0 , u_2]인 경우, $u = u_1$ 에서의 곡선상의 점 구하기: de Casteljau Algorithm



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

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

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

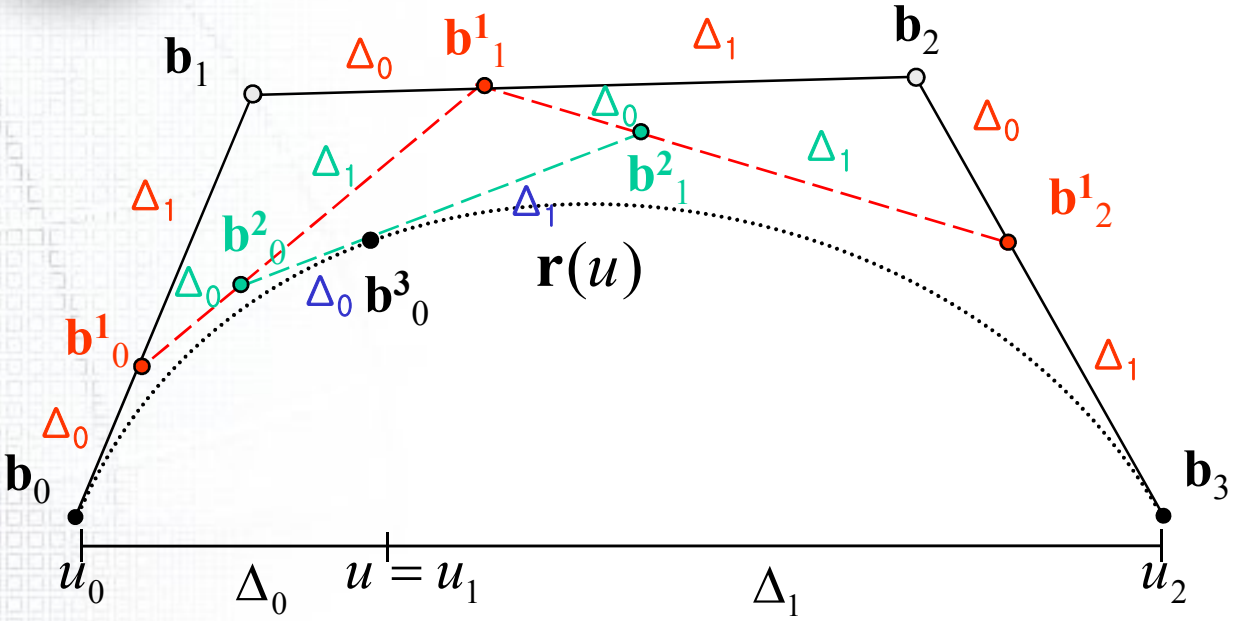
$$\mathbf{b}_1^1(u) = \frac{\Delta_1}{\Delta}\mathbf{b}_1^0 + \frac{\Delta_0}{\Delta}\mathbf{b}_2^0$$

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

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

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

2.2.3.4 매개변수 구간이 [u_0 , u_2]인 경우, $u = u_1$ 에서의 곡선상의 점 구하기: de Casteljau Algorithm



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

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

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

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

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

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

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

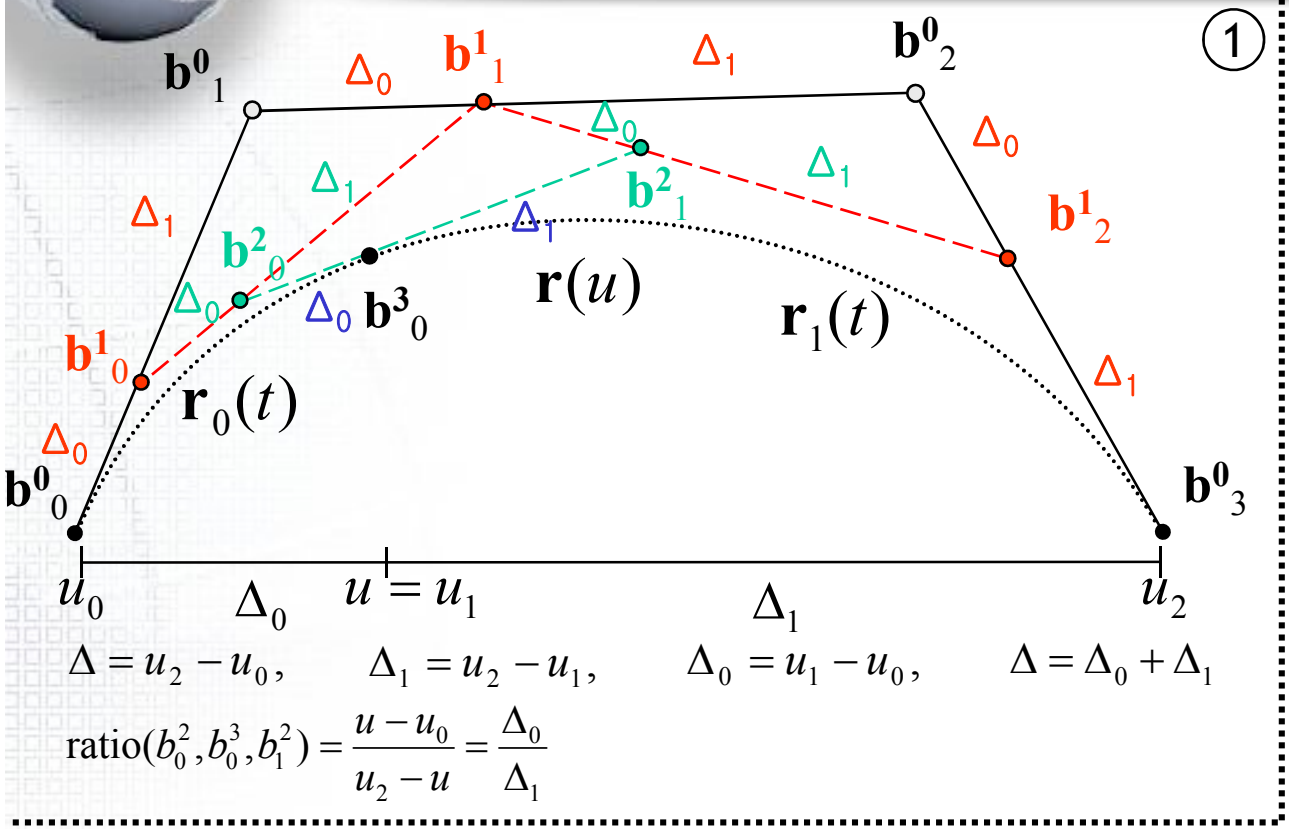
$$\mathbf{b}_0^3(u) = \frac{\Delta_1}{\Delta}\mathbf{b}_0^2 + \frac{\Delta_0}{\Delta}\mathbf{b}_1^2$$

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

3차 Bezier curves 와 동일한 함수식 !!!

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

2.2.3.5 de Casteljau Algorithm의 특성 (1)



1. Bezier 곡선의 $u = u_1$에서의 점 계산
2. 곡선 분할:
   $u = u_1$에서 2개의 Bezier 곡선으로 분할됨.

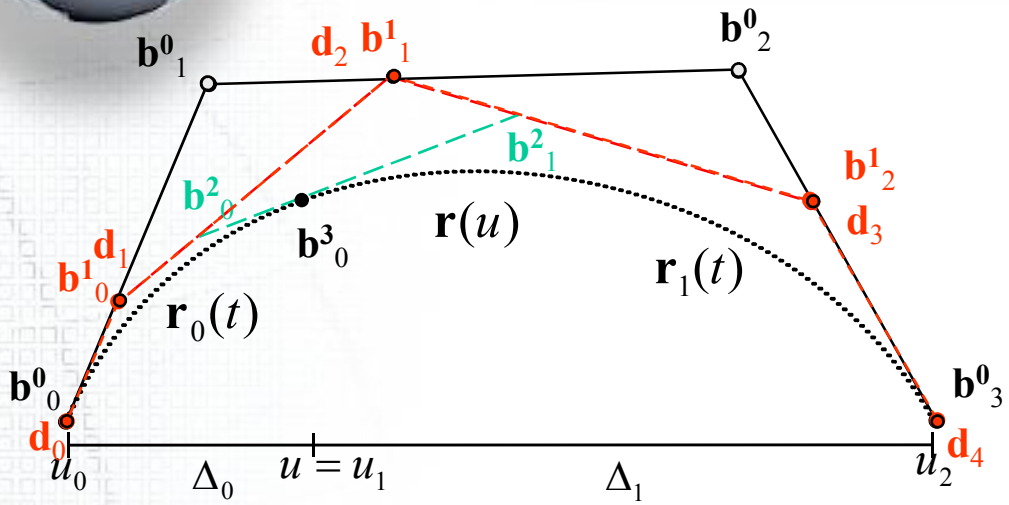
   즉, 왼쪽 곡선 $\mathbf{r}_0(t)$은 Bezier points $\mathbf{b}^0{}_0$, $\mathbf{b}^1{}_0$, $\mathbf{b}^2{}_0$, $\mathbf{b}^3{}_0$으로 이루어지고,
   오른쪽 곡선 $\mathbf{r}_1(t)$은 Bezier points $\mathbf{b}^3{}_0$, $\mathbf{b}^2{}_1$, $\mathbf{b}^1{}_2$, $\mathbf{b}^0{}_3$으로 이루어짐

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

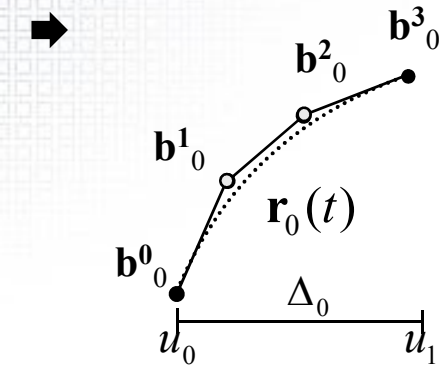
$\mathbf{r}_0(t) = (1-t)^3\mathbf{b}_0^0 + 3(1-t)^2 t\mathbf{b}_0^1 + 3(1-t)t^2\mathbf{b}_0^2 + t^3\mathbf{b}_0^3, \; t = \dfrac{u-u_0}{u_1-u_0}$

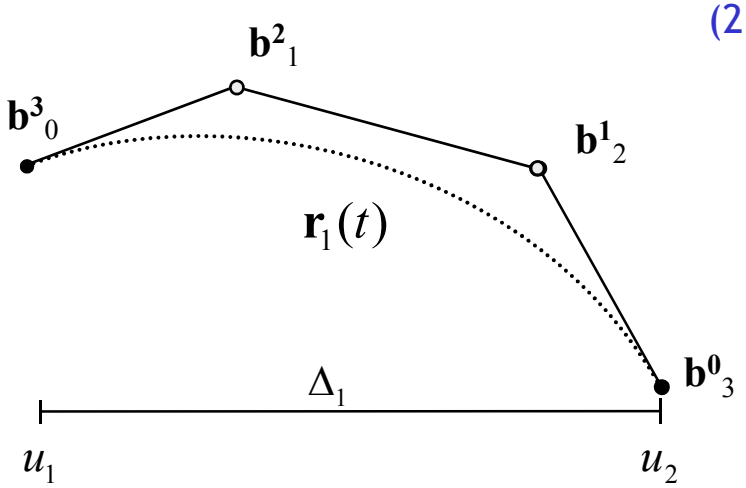
$\mathbf{r}_1(t) = (1-t)^3\mathbf{b}_0^3 + 3(1-t)^2 t\mathbf{b}_1^2 + 3(1-t)t^2\mathbf{b}_2^1 + t^3\mathbf{b}_3^0, \; t = \dfrac{u-u_1}{u_2-u_1}$

2.2.3.5 de Casteljau Algorithm의 특성 (2)



(1) 곡선 $\mathbf{r}(u)$은 떨어져 있는 두 개의 Bezier 곡선 $\mathbf{r}_0(t)$, $\mathbf{r}_1(t)$를 $u = u_1$에서 de Casteljau algorithm을 만족하도록 연결한 것으로 생각할 수 있음. 이 때 u_1은 곡선을 하나로 묶는 매듭이란 의미로 knot 라고 부름





(2) 다른 의미로 보면, 곡선 $\mathbf{r}(u)$은 떨어져 있는 두 개의 Bezier 곡선 $\mathbf{r}_0(t)$, $\mathbf{r}_1(t)$를 $u = u_1$에서 C^0, C^1, C^2 연속조건을 만족하도록 연결한 것으로 생각할 수 있다

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

$$t = \frac{u-u_0}{u_1-u_0}$$

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

$$t = \frac{u-u_1}{u_2-u_1}$$

2.2.3.6 Sample code of de Casteljau algorithm (1)

```
#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);                    // CalcPoint by de Casteljau algorithm
    double B (int i, double t);
};
#endif
```

2.2.3.6 Sample code of de Casteljau algorithm (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;
}
```

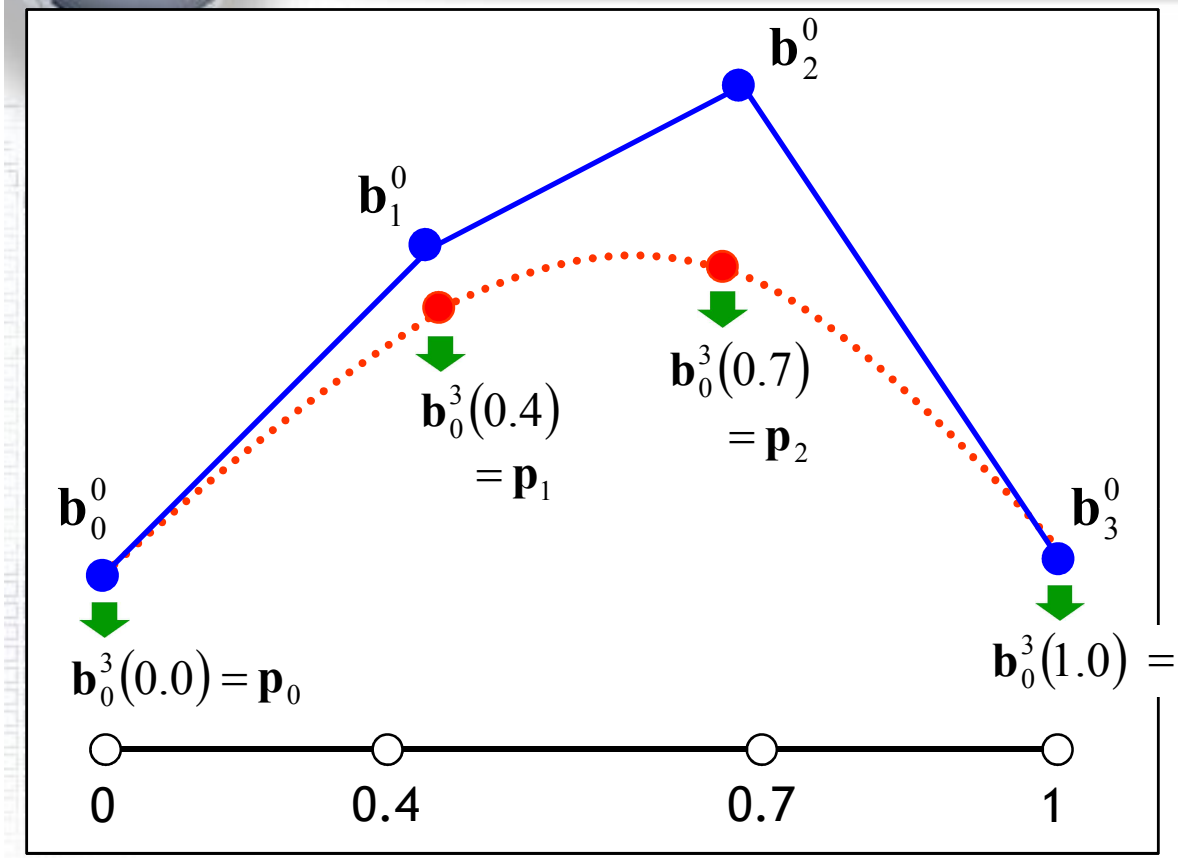
$$\mathbf{b}_0^0 \quad \mathbf{b}_0^1 \quad \mathbf{b}_0^2 \quad \mathbf{b}_0^3$$
$$\mathbf{b}_1^0 \quad \mathbf{b}_1^1 \quad \mathbf{b}_1^2$$
$$\mathbf{b}_2^0 \quad \mathbf{b}_2^1$$
$$\mathbf{b}_3^0$$

2.2.4 Bezier Curve Interpolation / Approximation

2.2.4.1 Introduction to Curve Interpolation
2.2.4.2 Cubic Bezier curve Interpolation
2.2.4.3 Bezier curve Interpolation beyond Cubics
2.2.4.4 Bezier curve Approximation
2.2.4.5 Finding the right parameters
2.2.4.6 Sample code of Bezier curve Interpolation

Advanced
Ship
Design
Automation
Laboratory

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

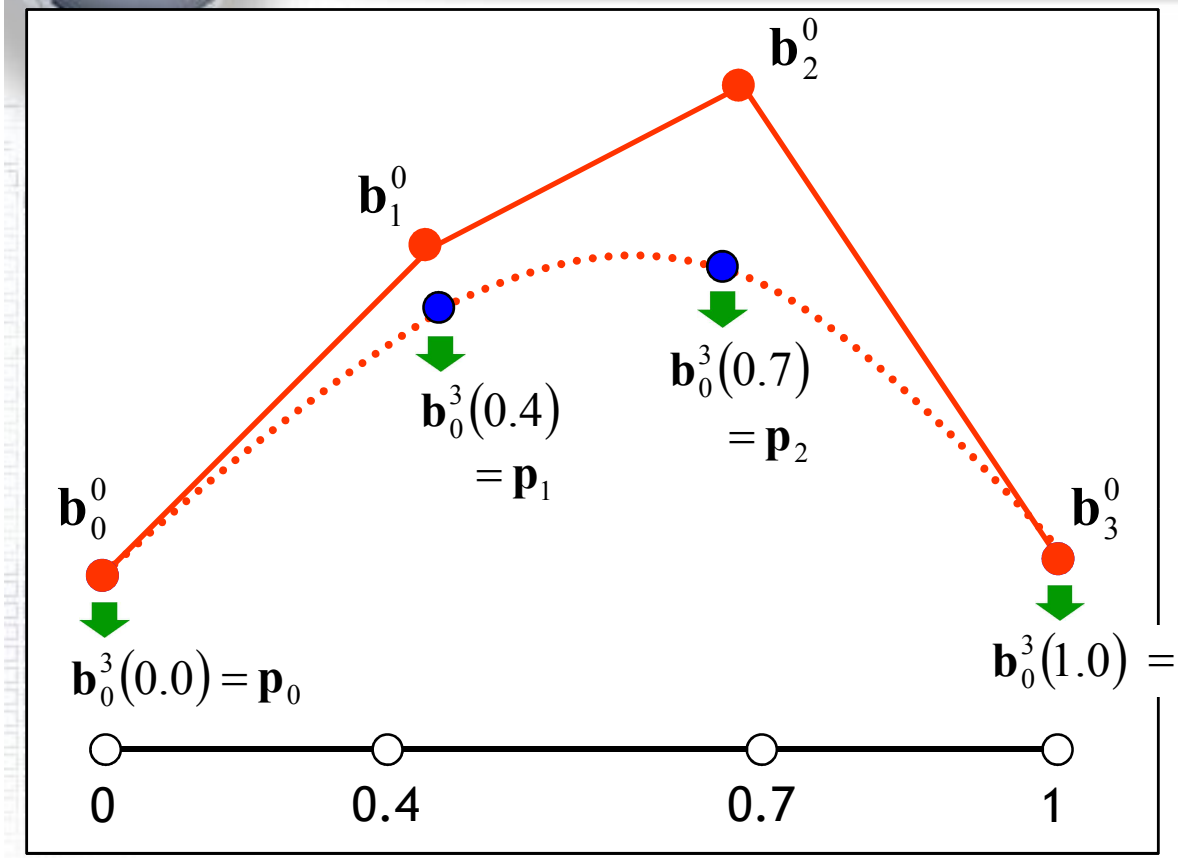
$$\mathbf{b}_0^3(0.0) = (1-0.0)^3\mathbf{b}_0^0 + 3\cdot 0.0(1-0.0)^2\mathbf{b}_1^0 + 3\cdot 0.0^2(1-0.0)\mathbf{b}_2^0 + 0.0^3\mathbf{b}_3^0 = \mathbf{b}_0^0$$

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

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

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

2.2.4.1 Introduction to Curve Interpolation (1)



Given

Points on 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

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

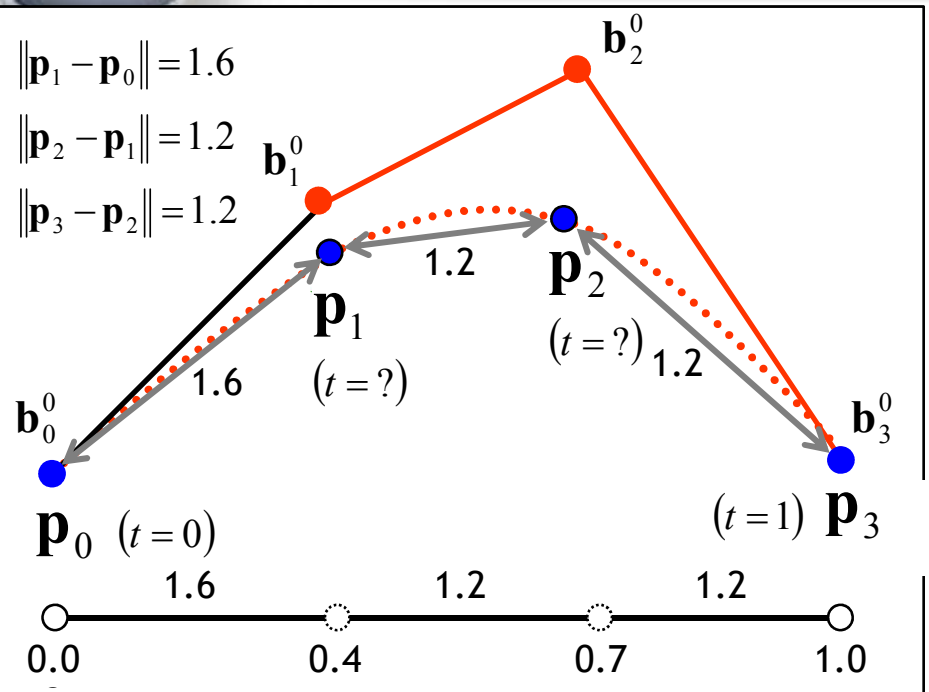
- If we are given fitting points $\mathbf{P}_i$ and we wish to pass a curve through them. There, the points are 2D, but the curve might as well be 3D. This is called "curve interpolation".

- We may choose among many kinds of curves; for right now, we will use a cubic Bezier curve.

→ "cubic Bezier curve interpolation"

$$\mathbf{b}_0^3(0.0) = (1-0.0)^3\mathbf{b}_0^0 + 3\cdot 0.0(1-0.0)^2\mathbf{b}_1^0 + 3\cdot 0.0^2(1-0.0)\mathbf{b}_2^0 + 0.0^3\mathbf{b}_3^0 = \mathbf{b}_0^0$$
$$\mathbf{b}_0^3(0.4) = (1-0.4)^3\mathbf{b}_0^0 + 3\cdot 0.4(1-0.4)^2\mathbf{b}_1^0 + 3\cdot 0.4^2(1-0.4)\mathbf{b}_2^0 + 0.4^3\mathbf{b}_3^0$$
$$\mathbf{b}_0^3(0.7) = (1-0.7)^3\mathbf{b}_0^0 + 3\cdot 0.7(1-0.7)^2\mathbf{b}_1^0 + 3\cdot 0.7^2(1-0.7)\mathbf{b}_2^0 + 0.7^3\mathbf{b}_3^0$$
$$\mathbf{b}_0^3(0.0) = (1-1.0)^3\mathbf{b}_0^0 + 3\cdot 1.0(1-1.0)^2\mathbf{b}_1^0 + 3\cdot 1.0^2(1-1.0)\mathbf{b}_2^0 + 1.0^3\mathbf{b}_3^0 = \mathbf{b}_3^0$$

2.2.4.1 Introduction to Curve Interpolation (2)



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

Given

Points on 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 points 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 $\mathbf{P}_i$.

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

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

$$t_0=0.0,\ t_3=1.0$$
$$t_1=\frac{1.6}{1.6+1.2+1.2}=0.4$$
$$t_2=\frac{1.6+1.2}{1.4+1.0+1.6}=0.7$$

Set parameter t using chord length

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

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

2.2.4.2 Cubic Bezier curve interpolation (1)

- The cubic Bezier curve of the form:

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

- All interpolation conditions are:

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

4 Unknown Vectors, 4 Vector Equations

2.2.4.2 Cubic Bezier curve interpolation (2)

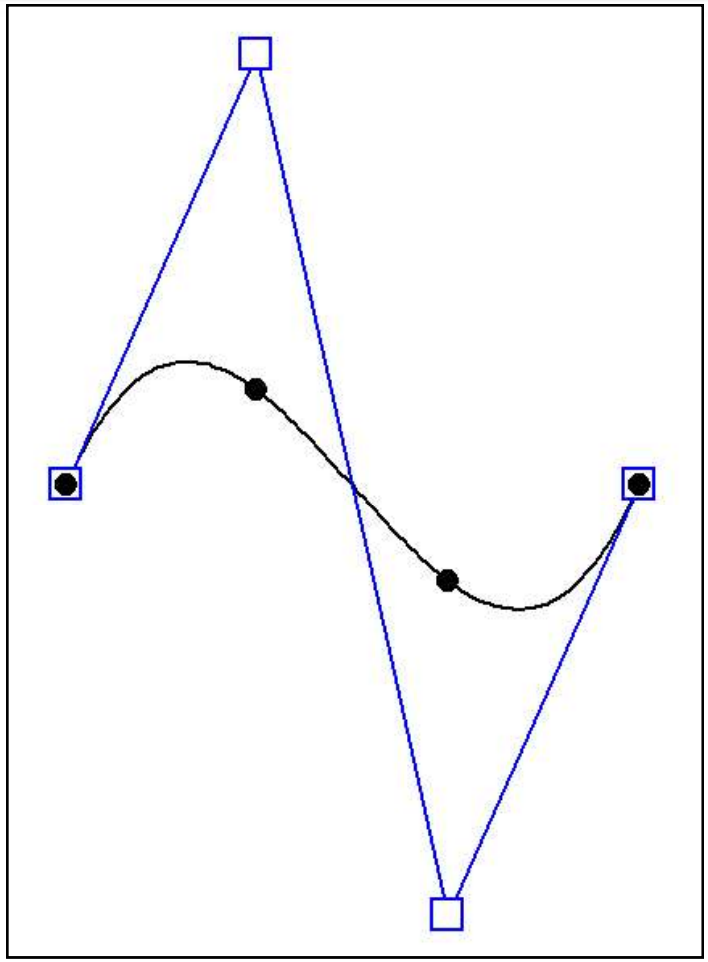
- To find the solution of these four equations for four unknowns, we can write in matrix form:

$$\begin{bmatrix} \mathbf{p_0} \\ \mathbf{p}_1 \\ \mathbf{p_2} \\ \mathbf{p_3} \end{bmatrix} = \begin{bmatrix} B_0^3(t_0) & B_1^3(t_0) & B_2^3(t_0) & B_3^3(t_0) \\ B_0^3(t_1) & B_1^3(t_1) & B_2^3(t_1) & B_3^3(t_1) \\ B_0^3(t_2) & B_1^3(t_2) & B_2^3(t_2) & B_3^3(t_2) \\ B_0^3(t_3) & B_1^3(t_3) & B_2^3(t_3) & B_3^3(t_3) \end{bmatrix} \begin{bmatrix} \mathbf{b_0} \\ \mathbf{b_1} \\ \mathbf{b}_2 \\ \mathbf{b}_3 \end{bmatrix}.$$

- To abbreviate the above form as: $\mathbf{P} = \mathbf{MB}$.
- 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 $\mathbf{p}_i$.

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

2.2.4.2 Cubic Bezier curve interpolation (3)



Cubic Bezier interpolation.

2.2.4.3 Bezier curve interpolation beyond Cubics (1)

- Polynomial interpolation can also works for more than four data points.
- Given: points $\mathbf{p}_0, \cdots\cdots \mathbf{p}_m$ 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"
- $n < m$: overdetermined system → "Approximation problem"

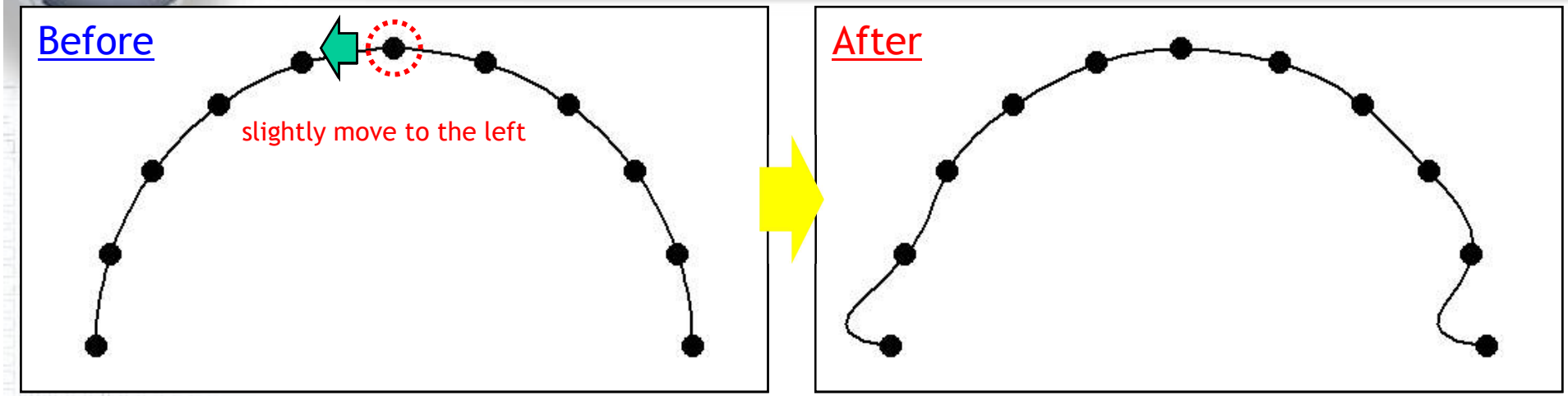
2.2.4.3 Bezier curve interpolation beyond Cubics (2)

- 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: $\mathbf{P} = \mathbf{MB}.$
- $\mathbf{M}$ 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.

2.2.4.3 Bezier curve interpolation beyond Cubics (3)



Top: Data from a circle; Bottom: one point 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.

2.2.4.4 Bezier curve approximation (1)

- One is given more data points than should be interpolated by a polynomial curve (i.e. number of data points more than 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'.

2.2.4.4 Bezier curve approximation (2)

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

- We wish to find a polynomial curve $\boldsymbol{r}(t)$ of a given degree $n\ (< m)$ such that

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

- Polynomial curve is of the Bezier form:

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

2.2.4.4 Bezier curve approximation (3)

- We would like the following to hold:

$$\begin{aligned}
\mathbf{p}_0 &= \mathbf{b}_0 B_0^n(t_0) + \ldots\ldots\ldots + \mathbf{b}_n B_n^n(t_0) \\
\mathbf{p}_1 &= \mathbf{b}_0 B_0^n(t_1) + \ldots\ldots\ldots + \mathbf{b}_n B_n^n(t_1) \\
&\vdots \qquad\qquad\quad \vdots \qquad\qquad\quad \vdots \\
\mathbf{p}_m &= \mathbf{b}_0 B_0^n(t_m) + \ldots\ldots\ldots + \mathbf{b}_n B_n^n(t_m)
\end{aligned}
\quad \Rightarrow \quad
\begin{bmatrix} B_0^n(t_0) & \cdots & B_n^n(t_0) \\ \vdots & & \vdots \\ B_0^n(t_m) & & B_n^n(t_m) \end{bmatrix}
\begin{bmatrix} \mathbf{b}_0 \\ \vdots \\ \mathbf{b}_n \end{bmatrix}
= \begin{bmatrix} \mathbf{p}_0 \\ \vdots \\ \mathbf{p}_m \end{bmatrix}$$

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

(m+1)*(2 or 3) Unknowns < (n+1)*(2 or 3) Equations

2.2.4.4 Bezier curve approximation (4)

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

$$\mathbf{M}^T\mathbf{M}\mathbf{B} = \mathbf{M}^T\mathbf{P}. \qquad \leftarrow \text{Normal equation}$$

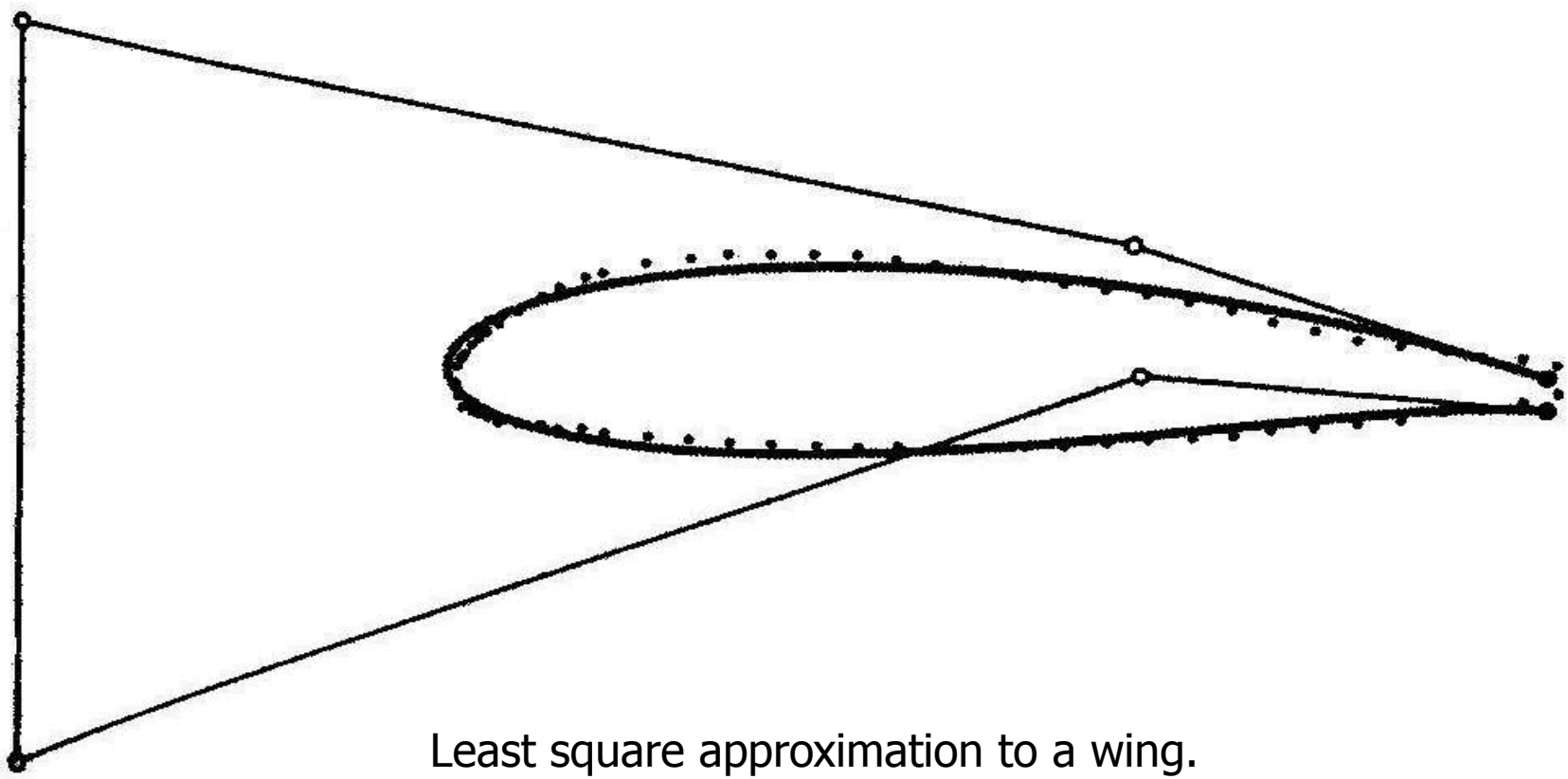
where $\mathbf{M}^T\mathbf{M}$ is a square and symmetric matrix, which is always invertible.

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

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

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

2.2.4.4 Bezier curve approximation (5)



Least square approximation to a wing.
A quintic Bezier curve with chord length parameters assigned to the data.

2.2.4.5 Finding the right parameters (1)

- In both interpolation & 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 $\mathbf{p}_i$,
- 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.

$$t_0 = 0$$

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

$$\vdots$$

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

2.2.4.5 Finding the right parameters (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, chord length parameterization method is superior to the uniform method, because it takes into account the geometry of the data.

2.2.4.6 Sample code of Interpolation/Approximation (1)

```
#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);
};
```

2.2.4.6 Sample code of Interpolation/Approximation (2)

```
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
    }
}
```

2.2.4.6 Sample code of Interpolation/Approximation (3)

```
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 normal equation
    ….
   delete[] t;
}
```

2.2.4.6 Sample code of Interpolation/Approximation (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);
   ….

}
```

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

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

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