서울대학교
공과대학
조선해양공학과

2009년 2학기

전산 선박 설계 (Computer Aided Ship Design)

Programming Assignment #6

담당 교수명	이규열
성명	이재훈
학과	조선해양공학과
학번	2007-11949
제출일	2009.11.22
점수	

목차

1. Prologue **3**

2. Bspline classs 5

2.1. 알고리즘 분석 5

2.2. Code 분석 9

3. CASDMFCDoc class 11

4. OpenGLVeiw class **13**

5. 실행 화면 16

5.1. Case 1 16

5.2. Case 2 17

5.3. Point 의 이동 기능 18

5.4. 국부적 가시화 19

5.5. 색깔 변경 20

5.6. Point 의 순서와 좌표를 계산 20

6. Discussion 21

7. 후기 22

1. Prologue

- 이번 과제에서는 초기에 curve 위를 지나는 point들과 curve의 시작점과 마지막 점에서의 tangent vector를 입력 받아 이 점들을 지나는 B-spline curve를 구현하는 것을 목표로 한다. 따라서 이를 구현하기 위해 program을 coding할 때에는 다음의 단계를 순차적으로 수행한다.

1) text file을 통해서 curve 위의 point들과 tangent vector의 정보를 입력 받는다.
2) 입력된 점들의 정보를 통해서 B-spline curve를 구현하기 위한 control point를 계산한다.
3) 계산된 control point를 Cox-de Boor recurrence formula에 대입하여 curve를 생성한다.
4) 생성된 curve를 가시화한다.

이번 project에서는 curve를 가시화하는 것이 최종 목표이므로 이를 위해서 ASDAL 연구실에서 제공하는 OpenGL 관련 tool인 CASDMFC를 사용한다. CASDMFC program을 사용할 때에는 사용자는 CASDMFCDoc class와 OpenGLVeiw class를 사용한다. 각각의 class에 대한 역할은 다음과 같다.

1) CASDMFCDoc class

- 이 class의 역할은 필요한 정보를 입력 받고 이를 저장, 수정하는 기능을 수행하는 것이다. 따라서 이번 과제에서는 B-spline curve를 생성할 때 필요한 curve 위의 point들과 tangent vector를 이 class에서 입력 받는다. 또한 입력 받은 점의 정보에 대하여 계산을 수행하는 class(본 과제에서는 Bspline class이다.)로 할당함으로써 계산을 수행하게 하고, 계산된 결과를 다시 이 class에 저장한다. 마지막으로 가시화를 수행하는 class에 계산 결과를 입력하는 역할을 최종적으로 수행한다. 따라서 가시화된 형상에 대해 변화를 주고자 하는 메뉴들(control point와 curve 위의 point의 이동, 그리고 그에 따라 변화하는 curve 형상 계산 등)을 수행하기 위해서는 계산을 수행하는 class를 다시 호출하여야 하기 때문에 이 class에서 위의 메뉴를 수행해야 할 것이다.

2) OpenGLVeiw class

- 이 class는 위의 CASDMFCDoc class에서 계산된 결과를 입력 받아 가시화를 수행하는 역할을 한다. 가시화의 방법에 대한 code는 이미 CASDMFC program에 구현되어 있으므로 사용자는 이를 적절히 사용하여 원하는 형상을 가시화할 수 있다. 따라서 가시화와 관련된 메뉴들(국부적 가시화, 형상의 색깔 변경, 가시화된 형상에 대한 계산 등)은 이 class에서 수행하여야 할 것이다.

이번 과제를 수행하기 위해서는 위에서 언급한 입력 받은 점의 정보와 tangent vector를

토대로 계산을 수행하는 class가 필요하다. 본 과제에서는 그 class를 B-spline class라 명명하고 이를 coding하여 계산을 수행하도록 하였다. 지금까지 언급한 3개의 class를 기반으로 이번 project는 구현되며 이에 대한 연관관계를 다음의 diagram으로 표시하였다.

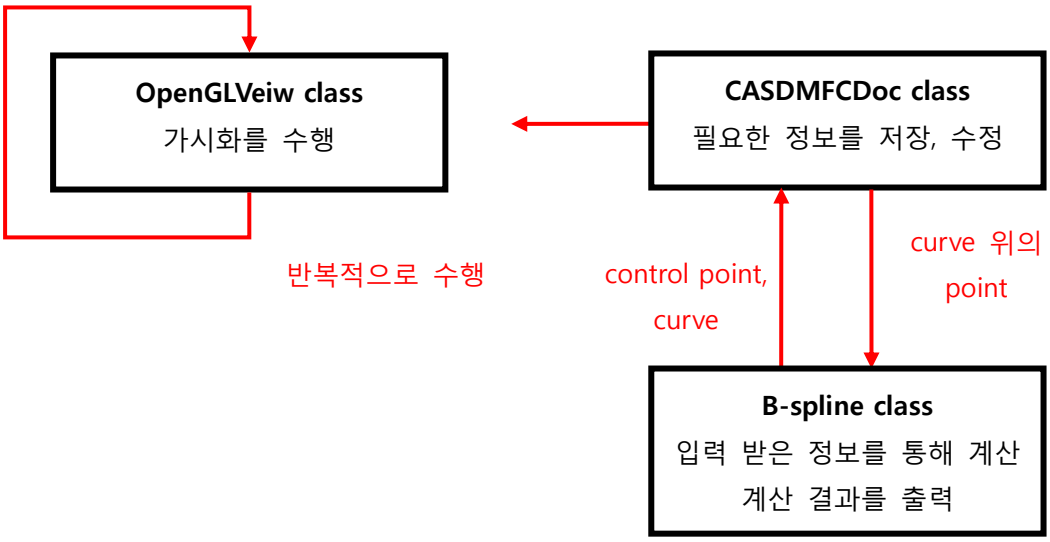


그림 1 - class간의 관계에 대한 diagram

이번 과제에서는 B-spline curve를 가시화할 뿐만 아니라 다음의 메뉴를 추가하여 다양한 기능을 구현할 수 있도록 한다.

1) 국부적 가시화

- 이번 과제에서는 가시화하는 항목은 4개로 나눌 수 있다. Control point, control polygon(control point를 연결한 선), curve 위의 point, B-spline curve가 바로 그것이다. 초기에 curve를 지나는 point에 대한 정보를 입력했을 때는 위의 사항이 모두 가시화되어야 하겠지만, 형상을 파악할 때 용이하도록 특정 항목만 가시화되게 하는 메뉴를 추가한다.

2) 색깔 변경

- 위에서 언급한 항목들에 대하여 가시화되는 색깔을 변경할 수 있도록 메뉴를 추가한다.

3) Curve 위의 point 또는 control point의 위치 변경 – 그에 따른 curve의 변화를 가시화

- 초기에 입력된 point들을 통해 계산을 하고 이를 가시화한 후에, 만약 curve 위의 point 또는 control point의 위치가 변화한다면 curve의 형상이 어떻게 변화하는지를 볼 수 있

도록 하는 메뉴를 추가한다.

4) Picking 기능 – 그에 따른 점의 좌표 계산

- Curve 위의 point들과 control point를 가시화한 후에, curve의 형상을 제대로 이해하기 위해서는 각각의 point들이 몇 번째 점이며, 어느 좌표에 위치하고 있는지를 알 필요가 있다. 따라서 각각의 point를 mouse의 wheel key를 통해 선택할 수 있고(선택 시, 색깔을 빨간색으로 변화시킨다.) 이 선택된 점에 대해서 이 점이 몇 번째 점이고, 그의 좌표가 어디인지를 알 수 있도록 하는 메뉴를 추가한다.

  지금까지 대략적으로 작성한 program에 대해서 어떠한 구성요소를 이루어지고 있고, 어떠한 기능을 갖는지, 그리고 이러한 기능을 수행하기 위해서 어떠한 단계로 진행되고 있는지 설명하였다. 이제 각각의 class에 대하여 구체적으로 살펴, 어떠한 알고리즘과 code로 구성되어 있는지 파악해 보자.

2. B-spline class

2.1. 알고리즘 분석

1) Control point 계산

- 이 class에서는 계산에 필요한 정보(curve 위의 point와 curve 위의 시작점과 마지막 점의 tangent vector)를 CASDMFCDoc class로부터 입력 받아 B-spline curve를 생성하기 위한 control point를 계산하고, 이를 통해서 Cox-de Boor recurrence formula를 구현하는 기능을 수행한다. 이를 위해서는 먼저 knot간의 간격 Δ_i를 알아야 한다. 이는 curve 위의 point 간의 간격을 통해서 구할 수 있다. n개의 curve 위의 point $\mathbf{p}_i$(i=0, … ,(n-1))에 대하여 이를 수식으로 쓰면 다음과 같다.

$$\Delta_0 = 0, \Delta_1 = 0, \Delta_{n+1} = 0$$
$$\Delta_i = \overline{\mathbf{p}_{i-2}\mathbf{p}_{i-1}}/\mathrm{SUM} \qquad (\mathrm{i} = 2, \cdots, \mathrm{n}) \qquad (2.1)$$
$$SUM = \sum_{i=2}^{n} \overline{\mathbf{p}_{i-2}\mathbf{p}_{i-1}}$$

  위의 식을 보면 알 수 있지만 Knot 간격은 $\mathbf{p}_i$의 개수보다 2개가 많은 (n+2)의 개수를 갖지만, 값이 들어가는 값은 $\mathbf{p}_i$의 개수보다 1개가 적은 (n-1)의 개수를 갖는다. knot 간격을 통해서 knot u_i의 값들을 계산할 수 있는데, 이 때 u_i의 범위를 0 이상 1이하의 값으로 하도록 하기 위해서 위의 식에서 $\mathbf{p}_i$간의 거리의 총합 SUM으로 knot 간격을 나눈 것이다. 이제 knot u_i를 계산하면 다음과 같다.

$$
\begin{aligned}
&u_{-1}=0, u_0=0, u_1=0, u_2=0 \\
&u_i = u_{i-1} + \Delta_{i-1} \qquad (\mathrm{i}=3,\cdots,\mathrm{n}+2) \\
&u_{n+3}=1, u_{n+4}=1, u_{n+5}=1
\end{aligned}
\tag{2.2}
$$

위의 식을 보면 알 수 있지만, u_i의 수는 $\mathbf{p_i}$의 수보다 6개 많으며 처음 4개는 0의 값을 마지막 4개는 1의 값을 갖는 것을 알 수 있다.
이제 control point $\mathbf{d_i}$와 $\mathbf{p_i}$와 시작점의 tangent vector $\mathbf{t_0}$과 마지막 점의 tangent vector $\mathbf{t_1}$ 간의 관계를 알아보자. 이는 다음의 행렬로 표현된다.

$$
\begin{pmatrix} \mathbf{p_0} \\ \mathbf{t_0} \\ \mathbf{p_1} \\ \vdots \\ \mathbf{p_{n-2}} \\ \mathbf{t_1} \\ \mathbf{p_{n-1}} \end{pmatrix}
=
\begin{pmatrix}
1 & 0 & 0 & \cdots & 0 & 0 & 0 \\
\frac{-3}{\Delta_2} & \frac{3}{\Delta_2} & 0 & 0 & \cdots & 0 & 0 \\
0 & \alpha_1 & \beta_1 & \gamma_1 & 0 & \cdots & 0 \\
 & & & \ddots & & & \\
0 & \cdots & 0 & \alpha_{n-2} & \beta_{n-2} & \gamma_{n-2} & 0 \\
0 & 0 & \cdots & 0 & 0 & \frac{-3}{\Delta_n} & \frac{3}{\Delta_n} \\
0 & 0 & 0 & \cdots & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix} \mathbf{d_0} \\ \mathbf{d_1} \\ \mathbf{d_2} \\ \vdots \\ \mathbf{d_{n-1}} \\ \mathbf{d_n} \\ \mathbf{d_{n+1}} \end{pmatrix}
\tag{2.3}
$$

위의 식을 보면 알 수 있지만 control point는 p_i의 개수보다 2개 많은 (n+2)개를 개수로 갖는다. 위의 식에서 α_i, β_i, γ_i에 대한 식은 다음과 같다.

$$
\begin{aligned}
\alpha_i &= \frac{(\Delta_{i+2})^2}{(\Delta_i+\Delta_{i+1}+\Delta_{i+2})(\Delta_{i+1}\Delta_{i+2})} \\
\beta_i &= \{\frac{\Delta_{i+2}(\Delta_i+\Delta_{i+1})}{(\Delta_i+\Delta_{i+1}+\Delta_{i+2})} + \frac{\Delta_{i+1}(\Delta_{i+2}+\Delta_{i+3})}{(\Delta_{i+1}+\Delta_{i+2}+\Delta_{i+3})}\}/(\Delta_{i+1}+\Delta_{i+2}) \qquad (\mathrm{i}=2,\cdots,\mathrm{n}-2) \\
\gamma_i &= \frac{(\Delta_{i+1})^2}{(\Delta_{i+1}+\Delta_{i+2}+\Delta_{i+3})(\Delta_{i+1}\Delta_{i+2})}
\end{aligned}
\tag{2.4}
$$

이제 우리는 control point $\mathbf{d_i}$를 계산할 수 있다 위의 식 (2.3)은 다음과 같이 간단한 행렬 식으로 볼 수 있다.

$$
\therefore \mathbf{p} = \mathbf{A}\mathbf{d} \tag{2.5}
$$

우리가 구하고자 하는 것은 행렬 **d**이다. 따라서 A의 역 행렬을 양변에 곱해줌으로써 **d**를 구할 수 있다. 하지만 과제에서는 curve의 형상이 복잡해질수록 위의 행렬 식은 크기가 커지게 되므로 선형적으로 문제를 간단히 풀이할 수 있는 LU 분해 법을 사용한다. 이에 대해서는 행렬 A가 tridiagonal 행렬(주 대각선과 그 위, 아래의 성분만이 값을 갖는

matrix)라는 것을 전제로 한다. 식 (2.3)에서 알 수 있듯이 행렬 A는 위의 전제를 만족하며 LU 분해 법을 적용할 수 있다. 분해 법의 적용은 다음과 같다. 먼저 A를 다음의 행렬로 표현한다.

$$\mathbf{A}=\begin{pmatrix} b_0 & c_0 & 0 & \cdots & 0 & 0 & 0 \\ a_1 & b_1 & c_1 & 0 & \cdots & 0 & 0 \\ 0 & a_2 & b_2 & c_2 & 0 & \cdots & 0 \\ & & & \ddots & & & \\ 0 & \cdots & 0 & a_{n-2} & b_{n-2} & c_{n-2} & 0 \\ 0 & 0 & \cdots & 0 & a_{n-2} & b_{n-1} & \gamma_{n-1} \\ 0 & 0 & 0 & \cdots & 0 & a_n & b_n \end{pmatrix} \tag{2.6}$$

이 행렬 A를 다음과 같이 행렬 L과 U로 분해한다.

$$\mathbf{A}=\mathbf{LU}=\begin{pmatrix} \beta_0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ \alpha_1 & \beta_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & \alpha_2 & \beta_2 & 0 & 0 & \cdots & 0 \\ & & & \ddots & & & \\ 0 & \cdots & 0 & \alpha_{n-2} & \beta_{n-2} & 0 & 0 \\ 0 & 0 & \cdots & 0 & \alpha_{n-2} & \beta_{n-1} & 0 \\ 0 & 0 & 0 & \cdots & 0 & \alpha_n & \beta_n \end{pmatrix}\begin{pmatrix} 1 & \gamma_1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1 & \gamma_2 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \gamma_3 & 0 & \cdots & 0 \\ & & & \ddots & & & \\ 0 & \cdots & 0 & 0 & 1 & \gamma_{n-1} & 0 \\ 0 & 0 & \cdots & 0 & 0 & 1 & \gamma_n \\ 0 & 0 & 0 & \cdots & 0 & 0 & 1 \end{pmatrix} \tag{2.7}$$

이때 식 (2.6)과 식(2.7) 간의 관계는 다음과 같다.

$$\begin{aligned} &\alpha_i = a_i && (\mathrm{i}=1,\cdots,\mathrm{n}) \\ &\gamma_{i+1} = \frac{c_i}{\beta_i} && (\mathrm{i}=0,\cdots,\mathrm{n}\text{-}1) \\ &\beta_0 = b_0 \\ &\beta_{i+1} = b_{i+1} - \alpha_{i+1}\gamma_{i+1} && (\mathrm{i}=0,\cdots,\mathrm{n}\text{-}1) \end{aligned} \tag{2.9}$$

이렇게 행렬로 나누게 되면 다음의 수식에 따라 선형적으로 계산을 수행할 수 있다.

$$\begin{aligned}&\mathbf{A}=\mathbf{LU}\\&\mathbf{p}=\mathbf{Ad}=\mathbf{LUd}\\&\mathbf{Ly}=\mathbf{p}\\&(\because \mathbf{y}=\mathbf{Ud})\end{aligned} \qquad (2.9)$$

p의 성분을 $\mathbf{p_i}$, **y**의 성분을 $\mathbf{y_i}$라 한다면

$$\begin{aligned}&\mathbf{y_0}=\frac{\mathbf{d_0}}{\beta_0}\\&\mathbf{y_i}=\frac{\mathbf{d_i}-\alpha_i\mathbf{y_{i-1}}}{\beta_i}\qquad (\mathrm{i}=1,\cdots,\mathrm{n})\end{aligned} \qquad (2.10)$$

이제 계산된 y_i를 통해서 **d**의 성분 $\mathbf{d_i}$를 계산할 결과는 다음과 같다.

$$\begin{aligned}&\mathbf{d_n}=\mathbf{y_n}\\&\mathbf{d_i}=\mathbf{y_i}-\gamma_i\mathbf{d_{i-1}}\qquad (\mathrm{i}=n-1,\cdots,0)\end{aligned} \qquad (2.11)$$

2) Cox –de Boor recurrence formula

- $\mathbf{d_i}$는 control point에 대한 정보로 이제 control point의 좌표의 계산을 완료한 것이다. 이제 계산된 control point $\mathbf{d_i}$(개수가 N($\mathbf{p_i}$의 개수보다 2개 많은))를 Cox-de Boor recurrence formula에 대입하여 parameter u(0 이상 1이하의 값)에 대한 curve의 point **r**(u)를 계산한다. 이에 대한 수식은 다음과 같다.

$$\begin{aligned}&\mathbf{r}(u)=\sum_{i=0}^{N-1}\mathbf{d_i}N_i^3(u)\\&N_i^n(u)=\frac{u-u_{i-1}}{u_{i+n-1}-u_{i-1}}N_i^{n-1}(u)+\frac{u_{i+n}-u_i}{u_{i+n}-u_i}N_{i+1}^{n-1}(u)\\&N_i^0(u)=\begin{cases}1 & \text{if } \mathrm{u_{i-1}}\le \mathrm{u}<\mathrm{u_i}\\0 & \text{else}\end{cases}\end{aligned} \qquad (2.12)$$

위의 식에서 $N_i^n(u)$는 B-spline curve의 basis function으로서 n은 curve의 차수를 의미한다. 이번 과제에서는 3차의 basis function을 사용한다(n=3).

3) Bessel end condition

- Bessel end condtion이란 위의 알고리즘을 수행할 때 양 끝 점에서의 tangent vector $\mathbf{t_0}$, $\mathbf{t_1}$

가 주어지지 않았을 때 curve의 양 끝의 연속된 세 point로 터 2차 curve를 생성하고 이 curve의 1차 미분 값을 통해 tangent vector를 근사 하는 방법이다. 이에 대한 수식은 다음과 같다.

$$\mathbf{t_0} = -\frac{2\Delta_2+\Delta_3}{\Delta_2(\Delta_2+\Delta_3)}\mathbf{p_0} + \frac{(\Delta_2+\Delta_3)}{\Delta_2\Delta_3}\mathbf{p_1} + \frac{\Delta_2}{\Delta_3(\Delta_2+\Delta_3)}\mathbf{p_2}$$
$$\mathbf{t_1} = -\frac{\Delta_{K-4}}{\Delta_{K-5}(\Delta_{K-5}+\Delta_{K-4})}\mathbf{p_{m-3}} - \frac{(\Delta_{K-5}+\Delta_{K-4})}{\Delta_{K-5}\Delta_{K-4}}\mathbf{p_{m-2}} + \frac{2\Delta_{K-4}+\Delta_{K-5}}{\Delta_{K-4}(\Delta_{K-5}+\Delta_{K-4})}\mathbf{p_{m-1}} \quad (2.13)$$

위의 식에서 K는 knot의 개수, m은 주어지는 curve의 point들의 개수이다. 이제 이렇게 계산된 tangent vector를 이용하여 control point를 계산할 수 있다. 단 주의해야 할 점은 위에서 control point를 계산할 때 사용되는 행렬 A가 다음과 같이 같다는 것이다.

$$\mathbf{A} = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 & \cdots & 0 & 0 \\ 0 & \alpha_1 & \beta_1 & \gamma_1 & 0 & \cdots & 0 \\ & & & \ddots & & & \\ 0 & \cdots & 0 & \alpha_{n-2} & \beta_{n-2} & \gamma_{n-2} & 0 \\ 0 & 0 & \cdots & 0 & 0 & -3 & 3 \\ 0 & 0 & 0 & \cdots & 0 & 0 & 1 \end{pmatrix} \quad (2.14)$$

2.2. Code 분석

- Code를 통해서 계산을 수행할 때에 각각의 변수들은 배열 변수를 사용하여 구현한다. 그런데 배열 변수는 항상 index가 0부터 시작하므로 이에 유의해 code를 작성해야 한다. B-spline class는 다음과 같은 멤버 함수로 이루어져 있다.

1) `Bspline(int degree, int numofdata);` 함수

- 이 함수는 Bspline class를 정의할 때 사용하는 생성자 함수로서 입력 받은 degree 변수를 이용하여 basis 함수의 차수를 결정하고 입력 받은 numofdata 변수를 통해서 입력 받은 curve 위의 point 및 다른 변수들의 개수를 결정하는 데 사용한다.

2) `Interpolation(Vector *InputPoint, Vector STV, Vector ETV)` 함수

- 이 함수는 control point 계산에 필요한 함수들을 종합하는 기능을 한다. 이 함수에서는 초기에 입력 받는 정보인 curve 위의 point들과 양 끝 점에서의 tangent vector를 입력 받아 계산을 수행하는 함수에 입력시켜준다.

3) set_up_system(double *alpha, double *beta, double *gamma) 함수

- 이 함수는 알고리즘에서 언급한 행렬 A를 계산하고 계산된 행렬 A를 LU system을 적용하여 행렬 L과 U로 분리시키는 과정을 수행한다. 이 때 행렬 A의 성분을 매개 변수로 입력받는데 이 때 pointer 변수로 입력 받음으로써 계산된 결과를 이 변수에 다시 출력하게 된다. 주의해야 할 점은 Bessel end condition을 적용해야 하는 경우에 if문을 통해 case를 구별하여 A를 계산해야 한다는 것이다.

4) solve_system(double *alpha, double *beta, double *gamma, Vector *InputPoint, Vector STV, Vector ETV) 함수

- 이 함수는 위에서 LU system을 통해 control point의 좌표를 계산하는 과정을 수행한다. 따라서 행렬의 성분을 뜻하는 변수들을 매겨 변수로 입력 받아 계산을 수행한다.

5) Vector Solution(double u, Vector *ControlPoint) **함수**

- 이 함수는 위에서 계산된 control point와 parameter u를 입력 받아 Cox-de Boor recurrence formula에 대입함으로써 u에 대한 curve 위의 point를 계산하는 함수이다. 이 함수는 Vector type의 함수로서 계산된 point의 좌표를 Vector type의 변수로 return한다.

6) cox_deboor_recursion(int i, int k, double u) **함수**

- 이 함수는 Cox-de Boor recurrence formular의 basis 함수를 정의하는 함수이다. 이에 대해서는 code를 살펴 보자.

```
double Bspline::cox_deboor_recursion(int i, int k, double u)
{
		if(k == 0)
		{
			if(m_dKnot[i] <= u && m_dKnot[i+1] > u)
			{
				return 1.0;
			}
			else
			{
				return 0.0;
			}
		}
		else
		{
			if(fabs(cox_deboor_recursion(i, k-1, u)) < 10e-6 && fabs(cox_deboor_recursion(i+1, k-1, u)) < 10e-6)
			{
				return 0.0;
			}
			else if(fabs(cox_deboor_recursion(i, k-1, u)) < 10e-6)
			{
				return (((m_dKnot[i+k+1]-u)/(m_dKnot[i+k+1]-m_dKnot[i+1]))*cox_deboor_recursion(i+1, k-1, u));
			}
			else if(fabs(cox_deboor_recursion(i+1, k-1, u)) < 10e-6)
			{
				return (((u-m_dKnot[i])/(m_dKnot[i+k]-m_dKnot[i]))*cox_deboor_recursion(i, k-1, u));
			}
			else
			{
				return (((u-m_dKnot[i])/(m_dKnot[i+k]-m_dKnot[i]))*cox_deboor_recursion(i, k-1, u) +
((m_dKnot[i+k+1]-u)/(m_dKnot[i+k+1]-m_dKnot[i+1]))*cox_deboor_recursion(i+1, k-1, u));
```

```
                }
            }
}
```

이 함수를 작성할 때에는 위에서 언급한 알고리즘을 그대로 따라가면 된다. 하지만 주의해야 할 점이 있다.

$$N_i^n(u) = \frac{u - u_{i-1}}{u_{i+n-1} - u_{i-1}} N_i^{n-1}(u) + \frac{u_{i+n} - u_i}{u_{i+n} - u_i} N_{i+1}^{n-1}(u)$$
$$N_i^0(u) = \begin{cases} 1 & \text{if } u_{i-1} \le u < u_i \\ 0 & \text{else} \end{cases} \qquad (2.12)$$

위의 식은 basis 함수 $N_i^n(u)$에 대한 정의이다. 위 식에서 n=3, i=0일 때를 보면 위의 식의 첫 번째 항의 분모는 0이 되는 것을 알 수 있다(u_{-1}=0, u_2=0). 그러나 이 때에 이 항의 값은 0이 된다. 그 이유는 곱해지는 $N_0^2(u)$이 0의 값을 갖기 때문이다. 하지만 위의 식을 그대로 code를 작성하면 분모가 0이 되어 무한대의 값(정의되지 않는 값)을 갖게 되므로 올바른 계산을 할 수 없다. 따라서 code를 작성할 때 곱해지는 basis function의 값을 fabs() 함수에 입력하여 절대값을 구한 후 이 값이 아주 작은 값 10^{-6}보다 작을 때는 앞의 분수를 계산하지 않도록 if문을 통해 case를 나누어 계산을 수행하였다. 또한 아래의 식을 살펴 보면 u_{i-1}과 u_i, 그리고 u가 모두 1인 경우에는 $N_i^0(u)$는 0의 값을 갖게 된다. 그러나 사실 알고리즘 상으로는 1의 값을 갖고 있어야 한다. 따라서 후에 u=1, 즉 curve의 마지막 점은 Cox-de Boor recurrence formula로 계산하기에 무리가 있다는 것을 유념해야 한다.

3. CASDMFCDoc class

- 이 class는 초기에 입력 받는 curve 위의 점들과 양 끝 점의 tangent vector에 대한 정보를 저장하고 B-spline class를 통해서 계산을 수행한 다음, 가시화를 수행하는 OpenGLVeiw class로 계산된 정보를 입력하는 기능을 수행하고 있다. 이 class는 다음의 멤버 함수로 이루어져 있다.

1) CCASDMFCDoc::OnFileOpen() **함수**

- 이 함수는 초기에 정보를 입력 받아 저장하는 기능을 수행한다. 이 함수는 Open 메뉴와 연결되어 있어 Open 메뉴를 실행하면 이 함수가 실행한다. 이 함수는 text file로 된 정보를 입력 받도록 하기 위해 CFileDialog class를 사용한다. 이 class를 실행하면 '열기' 대화상자가 실행되어 우리는 원하는 text file에 있는 정보를 입력 받을 수 있다. 주의해야 할 점은 양 끝점의 tangent vector가 주어진 경우에는 최소 2개의 점 정보를 입력 받아야 하고, tangent vector가 주어지지 않은 경우에는 최소 3개의 점 정보를 입력 받아야 한다. 만약 text file에서 이를 만족하는 경우와 만족하지 않는 경우를 if문을 통해 case를 나누

어 경고 메시지를 출력하도록 해야 한다. 입력되는 text file은 다음의 형식을 갖고 있어야 한다.

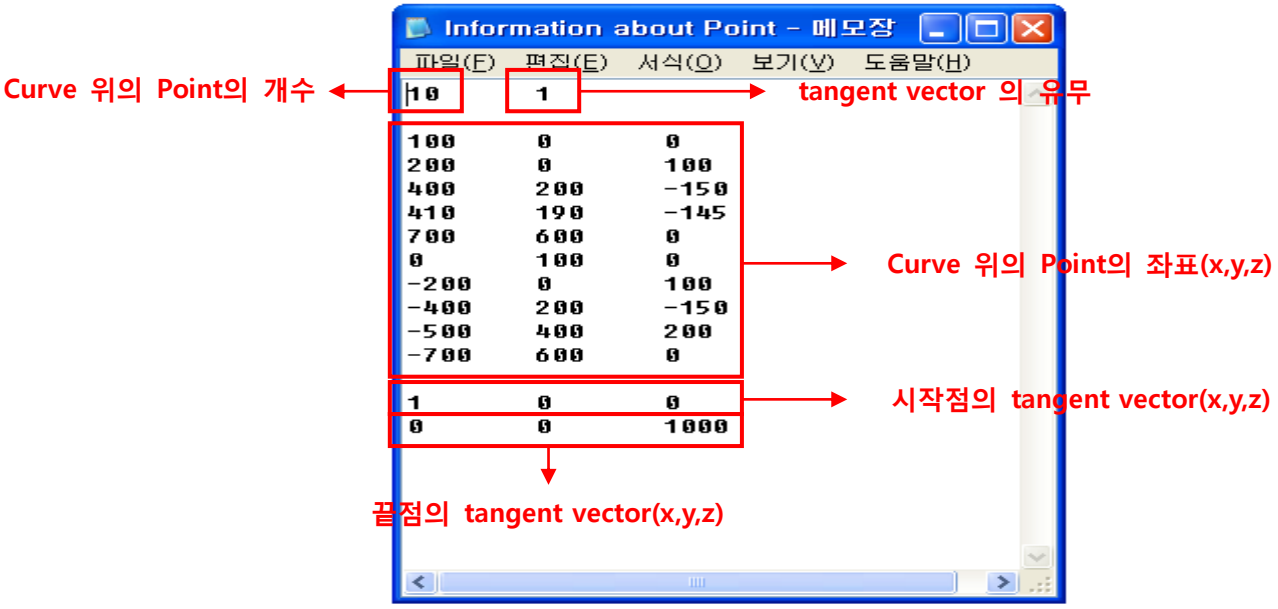


그림 2 - 입력하는 text file의 형식

2) CCASDMFCDoc::Calculation(void) **함수**

```
if(CheckTangentVector == 0)
		{
			double *c;
			c = new double[NumofPoint-1];
			double sum = 0;

			for(i=0; i<NumofPoint-1; i++)
			{
				c[i] = sqrt((InitPoint[i].x-InitPoint[i+1].x)*(InitPoint[i].x-InitPoint[i+1].x) +
(InitPoint[i].y-InitPoint[i+1].y)*(InitPoint[i].y-InitPoint[i+1].y) + (InitPoint[i].z-InitPoint[i+1].z)*(InitPoint[i].z-
InitPoint[i+1].z));
				sum += c[i];
			}
			for(i=0; i<NumofPoint-1; i++)
			{
				c[i] = c[i]/sum;
			}

			StartTangentVector.x = -(2*c[0]+c[1])*InitPoint[0].x/(c[0]*(c[0]+c[1])) +
(c[0]+c[1])*InitPoint[1].x/(c[0]*c[1]) - (c[0])*InitPoint[2].x/(c[1]*(c[0]+c[1]));
			StartTangentVector.y = -(2*c[0]+c[1])*InitPoint[0].y/(c[0]*(c[0]+c[1])) +
(c[0]+c[1])*InitPoint[1].y/(c[0]*c[1]) - (c[0])*InitPoint[2].y/(c[1]*(c[0]+c[1]));
			StartTangentVector.z = -(2*c[0]+c[1])*InitPoint[0].z/(c[0]*(c[0]+c[1])) +
(c[0]+c[1])*InitPoint[1].z/(c[0]*c[1]) - (c[0])*InitPoint[2].z/(c[1]*(c[0]+c[1]));

			EndTangentVector.x = (c[NumofPoint-2])*InitPoint[NumofPoint-3].x/(c[NumofPoint-3]*(c[NumofPoint-
3]+c[NumofPoint-2])) - (c[NumofPoint-3]+c[NumofPoint-2])*InitPoint[NumofPoint-2].x/(c[NumofPoint-3]*c[NumofPoint-2]) + (2*c[NumofPoint-
2]+c[NumofPoint-3])*InitPoint[NumofPoint-1].x/(((c[NumofPoint-3]+c[NumofPoint-2])*c[NumofPoint-2]);
			EndTangentVector.y = (c[NumofPoint-2])*InitPoint[NumofPoint-3].y/(c[NumofPoint-3]*(c[NumofPoint-
3]+c[NumofPoint-2])) - (c[NumofPoint-3]+c[NumofPoint-2])*InitPoint[NumofPoint-2].y/(c[NumofPoint-3]*c[NumofPoint-2]) + (2*c[NumofPoint-
2]+c[NumofPoint-3])*InitPoint[NumofPoint-1].y/(((c[NumofPoint-3]+c[NumofPoint-2])*c[NumofPoint-2]);
			EndTangentVector.z = (c[NumofPoint-2])*InitPoint[NumofPoint-3].z/(c[NumofPoint-3]*(c[NumofPoint-
3]+c[NumofPoint-2])) - (c[NumofPoint-3]+c[NumofPoint-2])*InitPoint[NumofPoint-2].z/(c[NumofPoint-3]*c[NumofPoint-2]) + (2*c[NumofPoint-
2]+c[NumofPoint-3])*InitPoint[NumofPoint-1].z/(((c[NumofPoint-3]+c[NumofPoint-2])*c[NumofPoint-2]);
		}
```

- 이 함수는 입력 받은 정보에 대하여 B-spline class를 호출함으로써, 계산을 수행하는 과정이다. 이 때, tangent vector의 유무를 통해서 만약 없다면 Bessel end condition을 적용하여 tangent vector를 계산하여야 한다. 위의 code는 바로 이 내용이다. 또한 Cox-de Boor recurrence formula를 통해서 계산을 수행할 때에는 parameter u는 0과 0.999사이에서 0.001씩 증가시키면서 각각의 u에 대한 Curve 위의 점을 계산한다. 단 위에서 언급했듯이 u가 1인 경우에는 올바른 계산을 하지 못하므로 계산을 수행하지 않는다.

3) CCASDMFCDoc::OnMovingpointInitialpoint() **함수**

- 이 함수는 Curve 위의 점을 이동했을 때에 대하여 다시 계산을 수행하는 함수이다. 이 함수는 창의 점을 이동시키는 메뉴와 연결되어 있으며 이를 실행하면 이에 대한 dialog 창이 실행된다. 이 dialog 창은 CMovingInitialPoint class와 연결되어 있으며 이 class 멤버 변수들을 입력 받고, 이 변수들을 Curve 위의 point를 뜻하는 변수들에 대입함으로써 Curve 위의 점을 이동했을 시킬 수 있다. 이 때 어떤 point인지를 알 기 위해서는 해당 point의 배열에서의 index를 알아야 한다. 만약 이 때, 이 index가 잘못 입력된다면, 경고 메시지를 출력할 수 있어야 한다. 이동시킨 후에는 다시 Calculation() 함수를 호출함으로써 다시 계산을 수행한다.

4) CCASDMFCDoc::OnMovingpointControlpoint() **함수**

- 이 함수는 control point를 이동했을 때의 계산을 다시 수행하는 함수이다. 이 함수의 일련의 알고리즘은 위의 OnMovinginitialpoint() 함수와 같다. 다른 점이 있다면 control point를 변화 시킨 후 Calculation() 함수를 호출 할 필요 없이 변경된 control point에 대하여 바로 Cox-de Boor recurrence formula를 적용하여 Curve를 계산한다. 하지만 Calculation() 함수를 호출하지 않으면 tangent vector가 초기에 입력 되지 않은 경우에 대하여 tangent vector를 계산할 수 없으므로 Bessel end condition에 따라 tangent vector를 계산하는 과정을 추가해준다.

4. OpenGLVeiw class

- 이 class에서 가시화를 담당하는 함수는 RenderScene() 함수이다. 따라서 가시화하고 싶은 형상을 이 함수에서 호출함으로써 가시화를 수행할 수 있다. 가시화할 항목은 총 4개이다. 이는 초기에 입력 받은 curve 위의 point들, 계산된 control point들, 그리고 이를 연결한 control polygon, 마지막으로 parameter u를 미소하게 변화시키면서 계산한 curve의 형상이 있다. 이에 대한 계산을 CASDMFCDoc class에서 수행하였으므로 이 class에서 계산된 값들을 OpenGLVeiw class로 불러 올 수 있어야 한다. 이를 위해 OpenGLVeiw class에서는

CASDMFCDoc class에서 GetDocument() 라는 이름의 class 변수로 선언해놓았으므로 이를 통해서 계산 결과를 호출할 수 있다. 가시화를 수행할 때 유의해야 할 점은 CASDMFCDoc class에서 text file에 따라 정보를 input 받은 다음에 가시화를 수행해야 하므로 이를 나타내는 index인 index 변수에 따라 case를 나누어 가시화를 수행해야 한다. 가시화를 수행하는 함수들은 다음의 4개 이다.

1) DrawControlPoint(void) - control point 가시화를 담당
2) DrawControlPolygon(void) – control polygon 가시화를 담당
3) DrawPointOnCurve(void) – 입력 받은 curve 위의 point를 가시화
4) DrawCurve(void) – 계산된 B-spline curve를 형상화

```
void COpenGLView::DrawPointOnCurve(void)
{
	int i=0;

	if(GetDocument()->INDEX == 1)
	{
		glColor3f(red3,green3,blue3);
		glInitNames();
		glMatrixMode(GL_MODELVIEW);
		for(i=0; i<GetDocument()->NumofPoint; i++)
		{
			glPushMatrix();
			glPushName(200 + (i+1));
			if(m_uClicked == 200 + (i+1))
			{
				glColor3f(1.0,0.0,0.0);
			}
			else
			{
				glColor3f(red3,green3,blue3);
			}
			glTranslatef(GetDocument()->InitPoint[i].x, GetDocument()->InitPoint[i].y, GetDocument()-
>InitPoint[i].z);
			glutSolidSphere(0.1,10,10);
			glPopName();
			glPopMatrix();
		}
	}
}
```

- 이 함수는 위에서 언급한 입력 받은 curve 위의 point를 가시화하는 함수이다. Point를 가시화를 할 때에는 glutSolidSphere() 함수를 사용하여 구로서 가시화를 한다. 이 때 유의해서 봐야 할 점은 PushName()~PopName() 함수를 사용하여 각각의 point들에 대하여 이름을 부여 했다는 것이다. 이 때 이름을 차례대로 201부터 1씩 증가된 숫자로 설정하였다. 이는 picking 기능을 수행하기 위한 것으로 만약 mouse의 wheel로 click하면 해당 point의 이름이 m_uClicked 변수에 할당된다. 이를 위해서는 PickProcess() 함수에서 가시화하는 함수를 호출하여야 하고, 또한 RenderScene() 함수에서 PickProcess() 함수를 호출해야 한다. m_uClicked 값을 비교함에 따라 선택 시, 해당 point의 색은 빨간색을 변하게 되고 따라서 어떠한 point를 선택했는지 알 수 있는 것이다. Control point를 가시화할 때에도 이와 같은 방식을 사용한다.

```
void COpenGLView::DrawCurve(void)
{
	int i=0;

	if(GetDocument()->INDEX == 1)
```

```
            {
                        glColor3f(red4,green4,blue4);
                        glPushMatrix();
                        glBegin(GL_LINE_STRIP);
                        for(i=0; i<1000; i++)
                        {
                                    glVertex3f(GetDocument()->PointonCurve[i].x, GetDocument()->PointonCurve[i].y, GetDocument()-
>PointonCurve[i].z);
                        }
                        glVertex3f(GetDocument()->InitPoint[GetDocument()->NumofPoint-1].x, GetDocument()->InitPoint[GetDocument()-
>NumofPoint-1].y, GetDocument()->InitPoint[GetDocument()->NumofPoint-1].z);
                        glEnd();
                        glPopMatrix();
            }
}
```

- 위의 함수는 위에서 언급한 curve를 생성하는 함수이다. curve를 생성하기 위해서 parameter u를 0.001씩 증가시키면서 각 u에 대한 curve 위의 point의 좌표를 계산하고 이를 직선으로 근사하여 연결하는 방법을 사용하였다. 그런데 유의해야 할 점은 우리는 u=1일 때에 대해서는 올바른 계산을 하지 못하므로 point들을 연결할 때 curve의 마지막 Point를 연결할 수 가 없다. 하지만 마지막 point는 초기에 입력 받은 curve 위의 point와 같으므로 curve를 끝까지 연결한다.

- OpenGLVeiw class에서 추가적인 기능을 수행하는 함수들로는 다음과 같은 함수가 있다.

1) 국부적 가시화
- OnUpdateViewControlpoint(CCmdUI *pCmdUI) **함수**
- OnViewControlpoint() **함수**
- OnUpdateViewCurve(CCmdUI *pCmdUI) **함수**
- OnViewCurve() **함수**

- 이 함수들은 control point와 control polygon에 대하여 가시화를 할지 말지 결정하는 함수이다. 가시화를 할 때 RenderScene() 함수에서 index에 따라 if문으로 case를 나누어 가시화를 할지 말지 결정하도록 하였다. 이 함수들은 메뉴의 Veiw 메뉴에서 checking을 통해서 실행되어 index를 조정한다. OnViewControlpoint() 함수는 실행 시 index를 변화시키고, OnUpdateViewControlpoint() 함수는 매개 변수 pCmdUL의 멤버 함수인 SetCheck() 함수를 통해 현재 가시화를 하고 있는지 아닌지를 판단한다. 이러한 알고리즘은 curve 위의 점들과 curve를 가시화 할 때에도 적용한다.

2) 색깔 변경
- OnChangecolorControlPoint() **함수**
- OnChangecolorControlPolygon() **함수**
- OnChangecolorPointonCurve() **함수**
- OnChangecolorCurve() **함수**

- 이 함수들은 가시화하는 각 항목에 대하여 색깔을 변경하는 함수이다. 이를 위해서는 CColorDialog class를 이용해야 한다. 이 class에서 DoModal() 함수를 실행하면 '색상 지정'

대화상자가 실행되고 여기서 우리가 선택한 색상은 이 class의 멤버 함수인 GetRValue() 함수, GetGValue() 함수, GetBValue() 함수를 통해서 알 수 있다. 이 함수는 빨강, 초록, 파랑의 비율을 출력하고 이 값을 255.0으로 나누면 OpenGLVeiw class에서 제공하는 색상으로 변화시킬 수 있다.

3) Picking 기능과 그에 따른 해당 point의 순서와 좌표를 계산

- OnCalculateControlPoint();
- OnCalculatePointoncurve();

- 위에서 언급했듯이 curve 위의 point들과 control point를 mouse로 선택하면 그 점이 몇 번째 점인지를 point들에 할당된 이름이 m_uClicked 변수에 입력되므로 이를 통해 알 수 있었다. 위의 함수들은 point를 선택한 후에 실행하는 것으로 실행 시 dialog 창을 통해 이 점이 curve 위의 점인지, control point인지를 알려주고 각 point가 몇 번째 점이며 좌표는 어떻게 되는지 알려 준다.

5. 실행 화면

5.1. Case 1 - 다음의 text file을 입력 했을 때

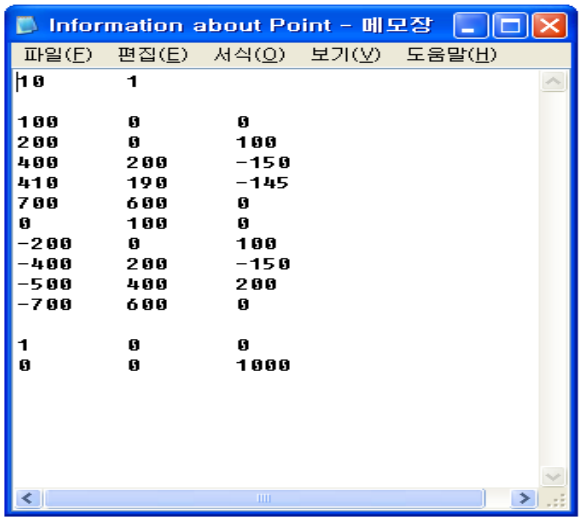


그림 3 -case 1의 text file

<계산된 control point>

Control point d_i	x 좌표	y 좌표	z 좌표
d_0	100.000	0.000	0.000
d_1	100.014	0.000	0.000

d_2	311.253	-52.242	286.289
d_3	318.680	288.470	-195.550
d_4	523.689	66.380	-82.033
d_5	971.239	1114.085	114.256
d_6	147.852	182.667	-208.100
d_7	-240.350	-69.169	228.901
d_8	-451.268	256.867	-432.385
d_9	-406.599	329.729	478.691
d_{10}	-700.000	600.000	-34.980
d_{11}	-700.000	600.000.	0.000

<가시화한 형상>

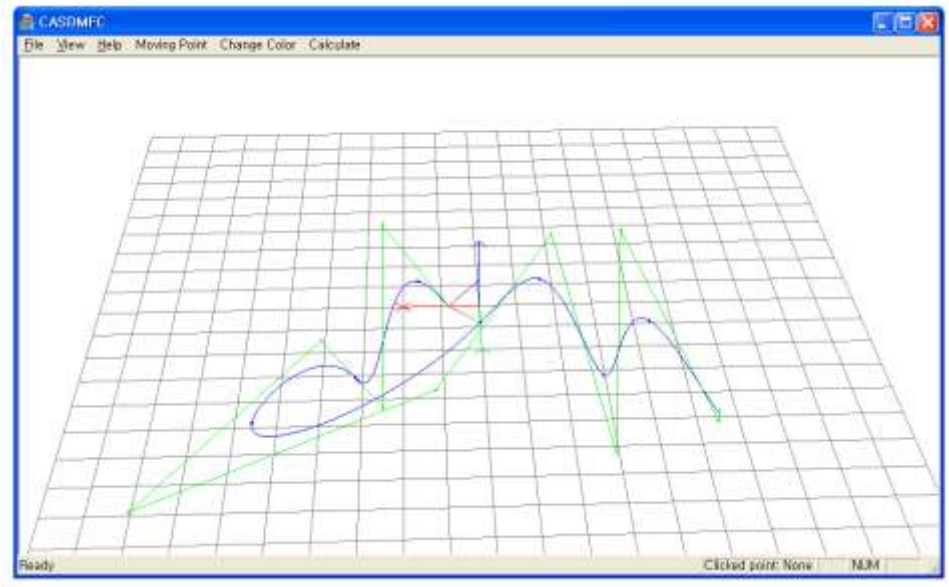


그림 4 - case 1에 대한 가시화 형상(축척비 1/50)

5.2. Case 2 – text file을 입력 했을 때

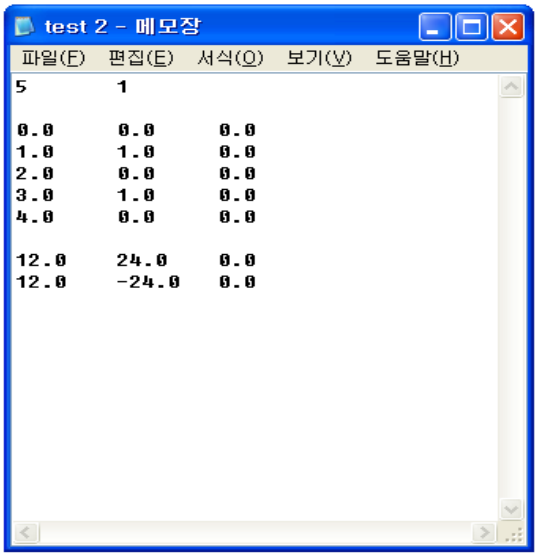


그림 5 - case 2의 text file

<가시화한 형상>

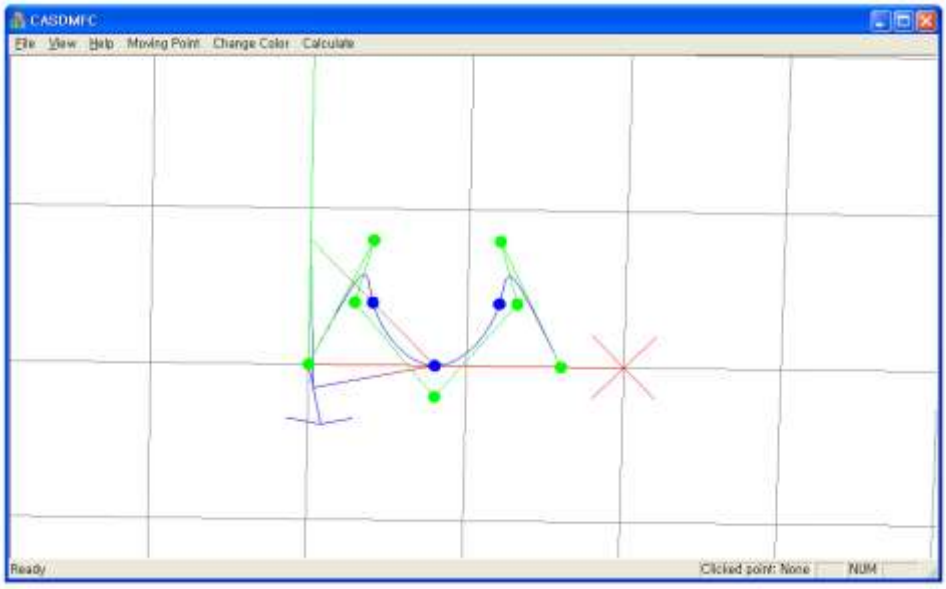


그림 6 - case 2에 대하여 가시화한 형상

5.3. Point의 이동 기능 – Case 2에 대해서

i) Point on curve의 이동

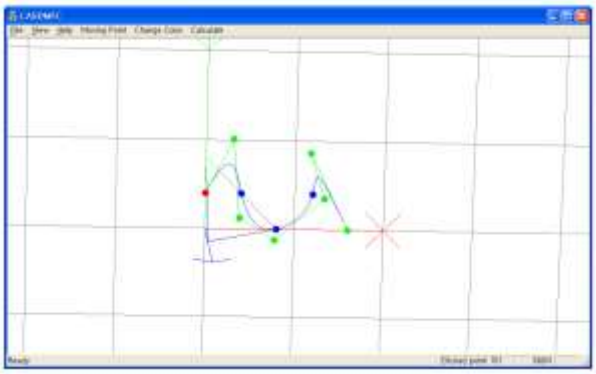


그림 7 - Point on curve의 이동(첫 번째 점을 (0,1,0)로 이동시킨 결과)

ii) Control point의 이동

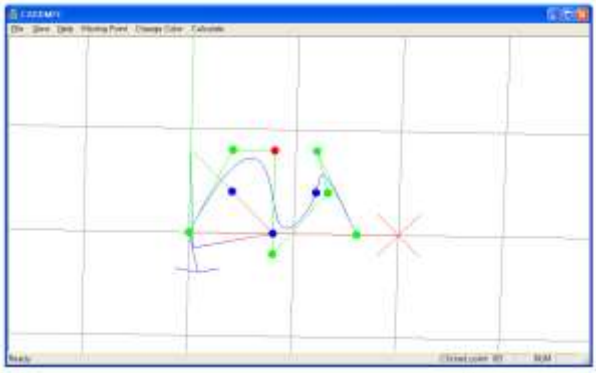


그림 8 - Control point의 이동(3번째 점을 (2,2,0)로 이동시킨 결과)

5.4. 국부적 가시화 – Case 2에 대해서

i) Control point와 control polygon만 가시화

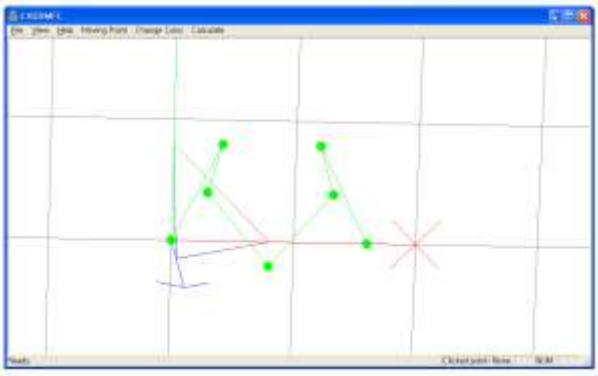


그림 9 - control point와 control polygon만 가시화

ii) Curve 위의 point와 curve만 가시화

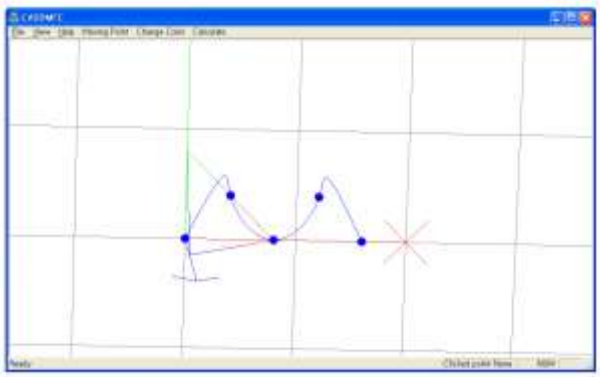


그림 10 - curve 위의 point와 curve만 가시화

5.5. 색깔 변경 – Case 2에 대해서

i) 색상 지정 대화 상자

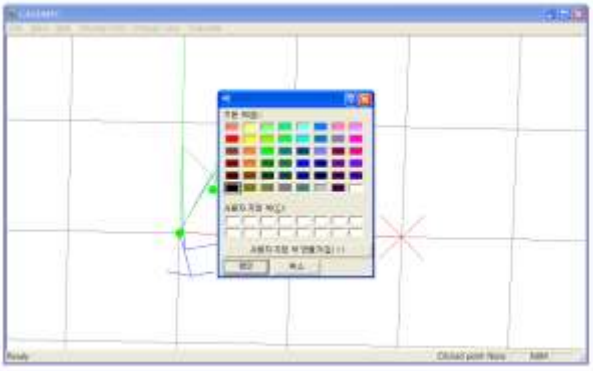


그림 11 - 색상 지정 대화 상자

ii) 색깔이 변경된 후의 가시화 모습

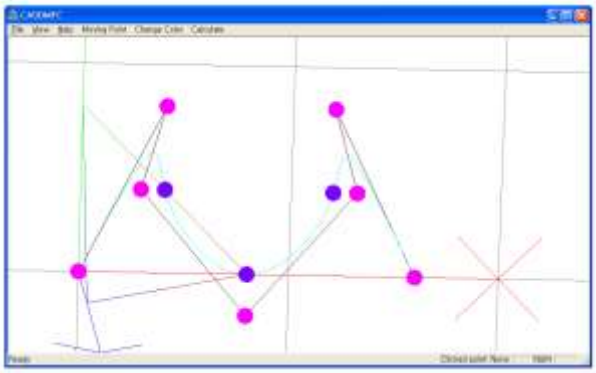
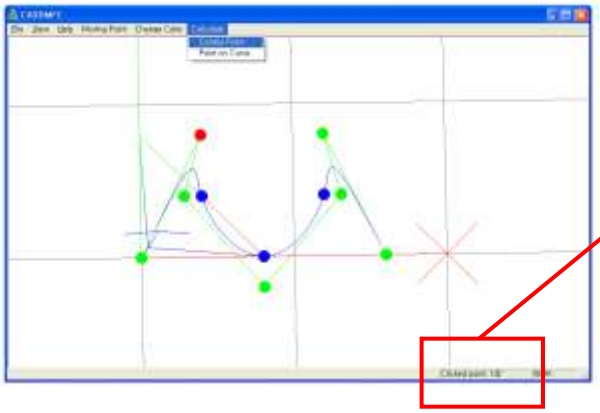


그림 12 - 색깔이 변경된 후의 가시화 모습

5.6. Point의 순서와 좌표를 계산 – case 2에 대해서

i) 2번째 control point를 picking



102는 이 control point의 이름으로 일의 자리 숫자 2는 control point의 순서를 의미한다. 만약 백의 자리가 2라면 이는 curve 위의 point를 의미하는 것이다.

ii) 계산 결과를 출력하는 dialog 창

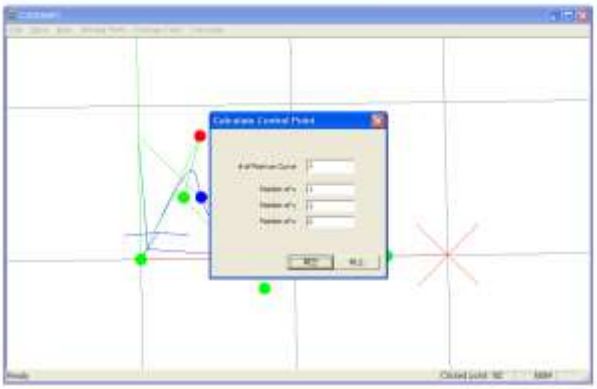


그림 13 – 계산결과를 출력하는 dialog 창

6. Discussion

- 위의 case 2에 대해서 curve 위의 첫 번째 점을 (1,1,0)로 이동시켜보자. 이 때 특별한 상황이 발생한다. Curve 위의 두 번째 점의 좌표가 (1,1,0)이므로 첫 번째 점과 두 번째 점이 포개지는 상황이 발생한다. 이러한 경우에는 첫 번째 point와 두 번째 point간의 간격이 0이 되므로 Δ_2=0이 된다. 이 때 우리가 작성한 code로는 올바른 계산을 수행하지 못한다. 따라서 이러한 경우, 즉 점을 이동시켰는데 point가 겹치는 경우에 점의 개수를 줄이고 줄여진 점에 대해서 가시화를 하도록 code를 CASDMFCDoc class의 Calculation() 함수에서 구현하였다. 아래의 code는 바로 이 내용이다.

```
int tempNumofPoint;
tempNumofPoint = NumofPoint;
Vector *tempInitPoint;
tempInitPoint = new Vector[tempNumofPoint];
for(i=0; i<tempNumofPoint; i++)
{
        tempInitPoint[i] = InitPoint[i];
}

for(i=0; i<NumofPoint-1; i++)
```

```
		{
			if(InitPoint[i] == InitPoint[i+1])
			{
				tempNumofPoint = tempNumofPoint - 1;
				InitPoint = new Vector[tempNumofPoint];
				for(int j=0; j<i+1; j++)
				{
					InitPoint[j] = tempInitPoint[j];
				}
				for(int j=i+1; j<tempNumofPoint; j++)
				{
					InitPoint[j] = tempInitPoint[j+1];
				}

				AfxMessageBox("Curve 위의점이겹치도록이동되었습니다.
이후의이동에대한계산에차질이있을수있습니다.");
			}
		}

		delete []tempInitPoint;

		NumofPoint = tempNumofPoint;
```

하지만 이 code도 문제점이 존재한다. 만약 point가 겹치도록 한 이후에 다시 한번 겹쳐진 point를 원래 위치로 이동시키면 점의 개수가 이미 줄어든 상태이므로 원래의 상태로 원상 복구를 할 수가 없다. 이에 대한 code의 점검은 추가적으로 고찰해야 할 것이다. 아래의 그림은 위에서 언급했듯이, case 2에 대해서 curve 위의 첫 번째 점을 (1,1,0)으로 이동시킨 후 가시화한 결과이다.

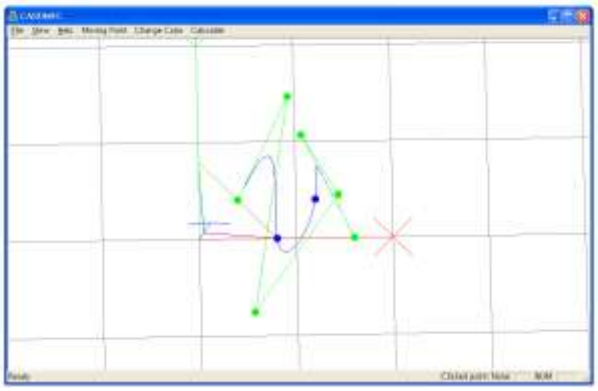


그림 14 - case 2에서 curve 위의 첫 번째 점을 (1,1,0)으로 이동한 경우

- 위의 그림을 보면 알 수 있지만 curve 위의 point는 원래 5개였지만 4개로 줄어들었고 control point의 좌표도 7개에서 6개로 줄어든 것을 확인할 수 있다.

7. 후기

- 이번 과제의 기초적인 내용은 3학년 1학기 과목인 그래픽스모델링에서 이미 배운 바 있다. 하지만 그 때는 Bezier curve에 대하여 초점이 맞추어져 있었으며, Bezier curve의 연결로 이루어진 B-spline curve에 대해서는 충분히 숙지하지 못하였었다. 이번 기회를 통해 Bezier curve가 어떻게 연결되어 B-spline curve가 생성되는지를 이해할 수 있었고, 이를 이해함에 따라 주어진 점을 지나는 B-spline curve를 구해내는 역 과정의 문제 역시 완벽하게 이해할 수 있었다. 지금까지 B-spline curve와 Bezier curve가 큰 상관관계가 없다고

생각했던 나에게 이번 과제는 매우 유익했다. 이제 생성한 curve를 연결하여 surface를 생성하는 과제를 수행해야 하는데 이번 과제를 열심히 노력하여 이해한 덕에 큰 무리없이 할 수 있을 것 같다. Curve에 대한 내용은 얼핏 살펴보면 매우 어려운 내용처럼 느껴지는데 이를 아주 간단 명료하면서 심층적으로 이해할 수 있도록 지도해주신 교수님께 감사의 말씀을 드린다. 또한 이번 과제를 수행하기 위해서는 OpenGL을 사용할 수 있도록 해야 하는데 새로운 tool을 적용함에 있어 무리가 없도록 지도해주신 조교님께도 감사의 말씀을 드린다. 이번 과제로 인해 curve에 대한 이해뿐만 아니라 MFC에 대한 전반적인 이해도도 키울 수 있었다.

Cf) Bessel end condition을 적용한 경우 – 다음의 text file을 입력한다.

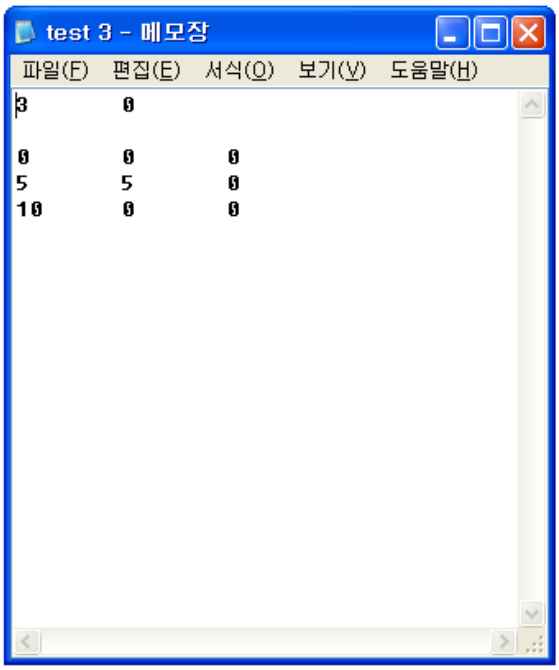


그림 15 - Bessel end condition을 사용하기 위한 text file

<가시화한 형상>

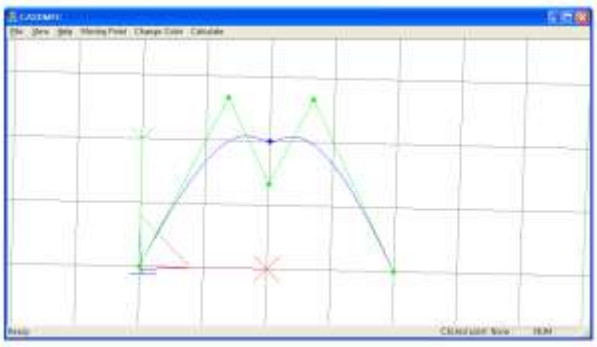


그림 16 - Bessel end condition을 적용한 예