서울대학교
공과대학
조선해양공학과

2009년 2학기
전산 선박 설계 (Computer Aided Ship Design)
Programming Assignment #7

담당 교수명	이규열
성명	이재훈
학과	조선해양공학과
학번	2007-11949
제출일	2009.11.29
점수	

목차

1. Prologue

- 이번 project에서는 6차 project에서 구현한 B-spline curve를 이용해서 B-spline surface를 생성하는 것을 목표로 한다. 따라서 surface 위에 존재하는 point(point on surface)들을 입력 받아야 한다. 그런데 입력 받는 point의 형식은 curve의 경우와 다르다. 이는 curve는 1차원 형상이지만, surface는 2차원 형상이기 때문이다. 따라서 curve와는 달리, surface는 Cox-de Boor recurrence formula를 적용할 때 사용되는 parameter가 2개가 필요하다. curve에서는 parameter u만을 사용했지만, surface에서는 parameter u, v를 사용한다. 이 때, u 방향을 row라 하고, v 방향을 column이라 명한다. 따라서 입력 받는 point는 row와 column을 모두 고려한 2차원 형식의 point $\mathbf{p}_{ij}$의 형식을 갖고 있어야 한다. 여기서 i는 row에 대한 index이며, j는 column에 대한 index이다. 이제 입력 받은 $\mathbf{p}_{ij}$를 통해서 B-spline surface를 구현할 수 있다. 이에 대한 과정을 정리한 것은 다음과 같다.

1) text file을 통해서 row의 크기, column의 크기, 그리고 surface 위의 point $\mathbf{p}_{ij}$(point on curve) 의 정보를 입력 받는다. 여기서 row의 크기란 u 방향으로 몇 개의 point가 존재하는지를 의미하며, 마찬가지로 column의 크기란 v 방향으로 몇 개의 point가 존재하는지를 의미한다.

2) 입력된 u 방향의 surface 위의 point들의 정보를 통해서 u 방향의 B-spline curve의 control point를 계산한다.

3) 위에서 계산된 u 방향의 control point들을 v 방향으로 연결하는 B-spline curve의 control point를 게산한다. 이 control point가 B-spline surface의 control point가 되는 것이다.

4) 위에서 계산된 B-spline surface의 control point를 Cox-de Boor recurrence formula에 대입하여 parameter u, v에 대한 surface 위의 point(design point)를 계산한다.

5) 위에서 계산된 각각의 u, v에 대한 design point들을 연결하여 B-spline surface를 가시화한다.

이번 project에서도 surface를 가시화하는 것이 최종 목표이므로 이를 위해서 ASDAL 연구실에서 제공하는 OpenGL 관련 tool인 CASDMFC를 사용한다. 6차 project에서와 마찬가지로 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 것이다. 따라서 이번 과

제에서는 B-spline surface를 생성할 때 필요한 row의 크기, column의 크기, 그리고 surface 위의 point들을 이 class에서 입력 받는다. 또한 입력 받은 point의 정보를 이에 대하여 계산을 수행하는 class(본 과제에서는 Bspline class와 BsplineSurface class이다.)로 할당함으로써 계산을 수행하게 하고, 계산된 결과를 다시 이 class에 저장한다. 마지막으로 가시화를 수행하는 class(본 과제에서는 OpenGLVeiw class이다.)에 계산 결과를 입력하는 역할을 최종적으로 수행한다. 따라서 가시화된 형상에 대해 변화를 주고자 하는 메뉴들(control point와 surface 위의 point의 이동, 그리고 그에 따라 변화하는 surface의 형상 계산 등)을 수행하기 위해서는 이에 대해서 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 project를 수행하기 위해서는 위에서 언급한 입력 받은 point의 정보를 토대로 계산을 수행하는 class가 필요하다. 본 과제에서는 그러한 class로서 Bspline class와 BsplineSurface class를 구현하였다. 지금까지 언급한 4개의 class를 기반으로 이번 project는 구현되며 이에 대한 연관관계를 다음의 diagram으로 표시하였다.

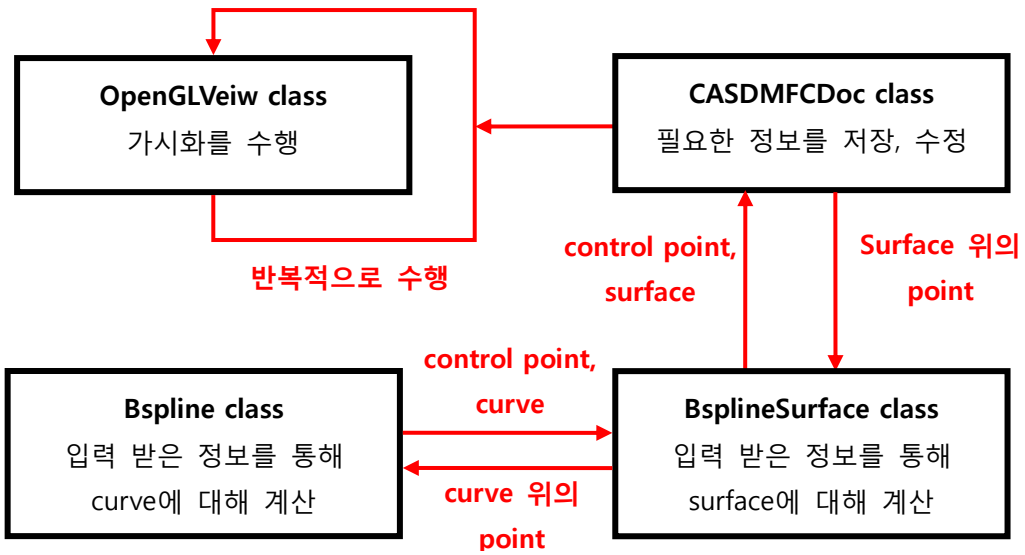


그림 1 - class간의 관계에 대한 diagram

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

1) 국부적 가시화

- 이번 과제에서는 가시화하는 항목은 3개로 나눌 수 있다. Point on curve, control point, B-spline surface가 바로 그것이다. 초기에 surface 위의 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) surface 위의 point 또는 control point의 위치 변경 – 그에 따른 surface의 변화를 가시화

- 초기에 입력된 point들을 통해 계산을 하고 이를 가시화한 후에, 만약 surface 위의 point 또는 control point의 위치가 변화한다면 curve의 형상이 어떻게 변화하는지를 볼 수 있도록 다시 계산을 수행하는 메뉴를 추가한다.

4) Picking 기능 – 그에 따른 point의 좌표 계산

- surface 위의 point들과 control point를 가시화한 후에, surface의 형상을 제대로 이해하기 위해서는 각각의 point들이 u 방향으로, 그리고 v 방향으로 몇 번째 point이며, 어느 좌표에 위치하고 있는지를 알 필요가 있다. 따라서 각각의 point를 mouse의 wheel button을 통해 선택할 수 있고(선택 시, 색깔을 빨간색으로 변화시킨다.) 이 선택된 point에 대해서 이 point가 몇 번째 point이고, 그 point의 좌표가 어디인지를 알 수 있도록 하는 메뉴를 추가한다.

5) Size 확대, 또는 축소 기능

- 가시화한 형상이 너무 크거나, 너무 작게 되면 형상을 제대로 관찰할 수 없다. 따라서 scale ratio를 입력 받아 초기에 입력 받은 surface 위의 point의 좌표에 scale ratio를 곱하여 다시 계산을 수행한 후, 이를 가시화하여 형상을 확대, 또는 축소할 수 있도록 하는 메뉴를 추가한다.

6) 가시화의 격자 간격 delta 조절

- 이번에 surface를 가시화할 때에는 u, v를 0에서부터 1까지 dalta만큼 증가시켜 가면서, 각각의 u, v에 대한 surface 위의 point(design point)를 계산하고 이 point들을 사각형으로 연결함으로써 surface를 구현한다. 즉 수많은 미소한 사각형의 연결을 통해 surface를 구현하는 것이다. 따라서 delta는 surface를 이루는 사각형, 즉 격자간의 간격이라고 볼 수 있는 것이다. 이 delta를 크게 할수록 surface가 부드럽게 연결되어 얻고자 하는 surface의 형상에 근사하는 정도를 높일 수 있다. 그러나 delta가 너무 커지게 되면 program의 계산 시간이 오래 걸리게 되므로 적절한 delta를 입력해야 한다. 따라서 적절한 delta를 찾기 위해 이 delta를 조절할 수 있는 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. Bspline class

2.1. 알고리즘 분석

- 기본적인 알고리즘은 6차 project에서 구현한 B-spline curve에 대한 알고리즘과 같다. 단 이번 project에서는 tangent vector가 주어지지 않는 경우이므로 Bessel end condition을 무조건 사용해야 한다. Bessel end condition을 다시 설명하면 다음과 같다.

1) Bessel end condition

- Bessel end condtion이란 위의 curve에 대한 계산을 수행할 때 양 끝 point에서의 tangent vector $\mathbf{t_0}$, $\mathbf{t_1}$가 주어지지 않았을 때($\mathbf{t_0}$가 curve의 시작 point에서의 tangent vector이고, $\mathbf{t_1}$가 curve의 마지막 point에서의 tangent vector이다.), curve의 양 끝의 연속된 세 point로 터 2차 curve를 생성하고 이 curve의 1차 미분 값을 통해 tangent vector를 근사하는 방법이다. 이에 대한 수식은 다음과 같다.

$$\mathbf{t_0} = -\frac{2\Delta_2+\Delta_3}{\Delta_2(\Delta_2+\Delta_3)}\mathbf{p_0} + \frac{(\Delta_2+\Delta_3)}{\Delta_2\Delta_3}\mathbf{p_1} + \frac{\Delta_2}{\Delta_3(\Delta_2+\Delta_3)}\mathbf{p_2}$$
$$\mathbf{t_1} = -\frac{\Delta_{K-4}}{\Delta_{K-5}(\Delta_{K-5}+\Delta_{K-4})}\mathbf{p_{m-3}} - \frac{(\Delta_{K-5}+\Delta_{K-4})}{\Delta_{K-5}\Delta_{K-4}}\mathbf{p_{m-2}} + \frac{2\Delta_{K-4}+\Delta_{K-5}}{\Delta_{K-4}(\Delta_{K-5}+\Delta_{K-4})}\mathbf{p_{m-1}} \quad (2.1)$$

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

$$\mathbf{A} = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 & \cdots & 0 & 0 \\ 0 & \alpha_1 & \beta_1 & \gamma_1 & 0 & \cdots & 0 \\ & & & \ddots & & & \\ 0 & \cdots & 0 & \alpha_{n-2} & \beta_{n-2} & \gamma_{n-2} & 0 \\ 0 & 0 & \cdots & 0 & 0 & -3 & 3 \\ 0 & 0 & 0 & \cdots & 0 & 0 & 1 \end{pmatrix} \qquad (2.2)$$

단, 유의해야 할 점이 있다. 위의 식 (2.1)에서 tangent vector를 계산할 때, 각 항은 Δ_i의 -1차 항인 것을 알 수 있다. 6차 과제에서는 Δ_i를 knot간의 간격으로 설정하였다. 여기서 knot간의 간격이란 것은 curve 위의 point들간의 간격을 계산하고 이를 point들간의 간격을 전부 더한 것으로 나눈 것으로부터 계산된다. 즉 knot간격은 0에서 1사이의 값으로 scaling된 것이다. 하지만 위에서 언급했듯이, tangent vector를 계산할 때에 각 항이 Δ_i의 -1차 항이므로 만약에 curve 위의 point의 수가 많아지면 Δ_i의 값이 너무 작아지게 되고 (보통 0에서 1사이의 값이 된다.), 따라서 tangent vector의 크기가 너무 커지게 된다. 이번 7차 과제의 경우, 선체의 형상을 가시화하게 되므로 curve 위의 point의 수가 많아지는 경우가 발생한다. 따라서 만약에 Δ_i를 6차 과제에서와 같이 knot 간격으로 설정한다면, tangent vector가 너무 커져서 제대로 된 형상을 가시화할 수 없다. 이를 방지하기 위해서 Δ_i를 curve 위의 point간의 간격으로 설정한다. 이렇게 하면 Δ_i가 보통 1보다 큰 값이 되고, tangent vector의 각 항이 Δ_i 의 -1차 항으로 이루어져 있기 때문에 오히려 tangent vector가 작아지게 된다. 이러한 경우, curve 위의 point들을 통해서 얻고자 하는 형상을 제대로 가시화할 수 있게 되는 것이다.

2.2. Code 분석

- Code는 아래의 calc_c() 함수를 제외하고는 6차 project와 동일하다.

1) calc_c() 함수

```
void Bspline::calc_c(Vector *InputPoint)
{
		int i=0;

		double sum = 0.0;

		c[0] = 0.0;
		c[1] = 0.0;

		for(i=2; i<m_nNumofControl-1; i++)
		{
			c[i] = sqrt((InputPoint[i-2].x - InputPoint[i-1].x)*(InputPoint[i-2].x - InputPoint[i-1].x) + (InputPoint[i-
2].y - InputPoint[i-1].y)*(InputPoint[i-2].y - InputPoint[i-1].y) + (InputPoint[i-2].z - InputPoint[i-1].z)*(InputPoint[i-2].z -
InputPoint[i-1].z));
			sum += c[i];
		}

		c[m_nNumofControl-1] = 0.0;
```

```
		STV.x = -((2.0*c[2]+c[3])*InputPoint[0].x/(c[2]*(c[2]+c[3]))) + (c[2]+c[3])*InputPoint[1].x/(c[2]*c[3]) -
(c[2])*InputPoint[2].x/(c[3]*(c[2]+c[3]));
		STV.y = -((2.0*c[2]+c[3])*InputPoint[0].y/(c[2]*(c[2]+c[3]))) + (c[2]+c[3])*InputPoint[1].y/(c[2]*c[3]) -
(c[2])*InputPoint[2].y/(c[3]*(c[2]+c[3]));
		STV.z = -((2.0*c[2]+c[3])*InputPoint[0].z/(c[2]*(c[2]+c[3]))) + (c[2]+c[3])*InputPoint[1].z/(c[2]*c[3]) -
(c[2])*InputPoint[2].z/(c[3]*(c[2]+c[3]));

		ETV.x = (c[m_nNumofControl-2])*InputPoint[m_nNumofControl-5].x/(c[m_nNumofControl-3]*(c[m_nNumofControl-
3]+c[m_nNumofControl-2])) - (c[m_nNumofControl-3]+c[m_nNumofControl-2])*InputPoint[m_nNumofControl-4].x/(c[m_nNumofControl-
3]*c[m_nNumofControl-2]) + (2.0*c[m_nNumofControl-2]+c[m_nNumofControl-3])*InputPoint[m_nNumofControl-3].x/((c[m_nNumofControl-
3]+c[m_nNumofControl-2])*c[m_nNumofControl-2]);
		ETV.y = (c[m_nNumofControl-2])*InputPoint[m_nNumofControl-5].y/(c[m_nNumofControl-3]*(c[m_nNumofControl-
3]+c[m_nNumofControl-2])) - (c[m_nNumofControl-3]+c[m_nNumofControl-2])*InputPoint[m_nNumofControl-4].y/(c[m_nNumofControl-
3]*c[m_nNumofControl-2]) + (2.0*c[m_nNumofControl-2]+c[m_nNumofControl-3])*InputPoint[m_nNumofControl-3].y/((c[m_nNumofControl-
3]+c[m_nNumofControl-2])*c[m_nNumofControl-2]);
		ETV.z = (c[m_nNumofControl-2])*InputPoint[m_nNumofControl-5].z/(c[m_nNumofControl-3]*(c[m_nNumofControl-
3]+c[m_nNumofControl-2])) - (c[m_nNumofControl-3]+c[m_nNumofControl-2])*InputPoint[m_nNumofControl-4].z/(c[m_nNumofControl-
3]*c[m_nNumofControl-2]) + (2.0*c[m_nNumofControl-2]+c[m_nNumofControl-3])*InputPoint[m_nNumofControl-3].z/((c[m_nNumofControl-
3]+c[m_nNumofControl-2])*c[m_nNumofControl-2]);

		for(i=0; i<m_nNumofControl; i++)
		{
			c[i] = c[i]/sum;
		}
}
```

- 위의 code는 6차 과제의 Bspline class에서는 없는 calc_c() 함수이다. 이 함수는 매개변수로 curve 위의 point의 정보를 갖고 있는 변수 InputPoint를 갖는다. 따라서 이 변수를 통해서 knot 간격을 뜻하는 변수 c를 계산한다. 위의 code에서 변수 STV는 curve의 시작 point에서의 tangent vector를 뜻하며, 변수 ETV는 curve의 마지막 point에서의 tangent vector를 뜻한다. 계산할 때 변수 c를 사용하는데, 여기서 c는 알고리즘에서도 설명했듯이 knot 간격이 아닌 curve 위의 point들간의 간격이어야 한다. 위의 code를 보면 변수 sum은 point들간의 간격을 전부 더한 것이며, 따라서 tangent vector를 계산할 때에는 sum으로 c를 나누기 전에, 즉 c를 0에서 1사이의 값으로 scaling하기 전에, tangent vector를 계산하는 것을 볼 수 있다. 이후에 c를 sum으로 나누어줌으로써 이제서야 c가 knot 간격이 되는 것이다. 후에 B-spline curve에 대한 계산을 할 때에는 이 c, 즉 knot 간격을 사용한다.

2) Solution() 함수

```
Vector Bspline::Solution(double u, Vector *ControlPoint)
{
		int i=0;

		Vector DesignPoint;
		DesignPoint.x = 0.0;
		DesignPoint.y = 0.0;
		DesignPoint.z = 0.0;

		if(fabs(u - 1.0) < 10e-9)
		{
			DesignPoint = ControlPoint[m_nNumofControl-1];
		}
		else
		{
			for(i=0; i<m_nNumofControl; i++)
			{
				DesignPoint.x += ControlPoint[i].x * cox_deboor_recursion(i, m_nk, u);
				DesignPoint.y += ControlPoint[i].y * cox_deboor_recursion(i, m_nk, u);
				DesignPoint.z += ControlPoint[i].z * cox_deboor_recursion(i, m_nk, u);
			}
		}

		return DesignPoint;
}
```

- 이 함수는 특정 u에 대해서 design point를 계산하는 함수로써 6차 project와 대부분의 구성은 같다. 하지만 수정한 부분이 존재한다. Cox-de Boor recurrence formula는 parameter u가 1일 때에는 계산을 수행하지 못한다. 그런데 u=1일 때에 design point는 curve의 마지막 control point와 같다. 따라서 만약 u가 1이라면(code에서는 1과의 차가 10^{-9}보다 작게 되면) Cox-de Boor recurrence formula에 대한 계산을 수행하지 않고 마지막 control point를 design point로 하도록 하였다. 즉 마지막 control point를 뜻하는 변수 ControlPoint[m_nNumofControl-1]을 design point를 뜻하는 변수 DesignPoint에 할당한다.

3. BsplineSurfave class

3.1. 알고리즘 분석

- 이제 Bspline class를 통해서 B-spline surface를 구현하는 BsplineSurface class에 대하여 알아 보자. 이는 다음의 알고리즘의 순서를 따른다.

1) 주어진 surface 위의 point들을 u 방향으로 연결한 B-spline curve의 control point 계산

- CASDMFCDoc class를 통해서 row의 크기, column의 크기, 그리고 surface 위의 point $\mathbf{p}_{ij}$를 입력 받는다. 이 때, row의 크기를 m, column의 크기를 n이라 하자. 이제 u방향으로 B-spline curve의 control point를 계산한다. u 방향으로는 m개의 point가 존재하므로 curve 위의 point가 m개인 curve에 대하여 control point를 계산하는 것이다. 이는 각각의 column에 대해서 수행하는 것이다. 따라서 이러한 curve의 개수는 n개가 될 것이다. 정리하자면, m개의 curve 위의 point를 갖는 B-spline curve n개에 대해서 control point를 계산하는 것이다. 이 과정을 간략히 표현하기 위해서 row의 크기가 3, column의 크기가 3인 경우를 도시한 것이 아래의 그림이다.

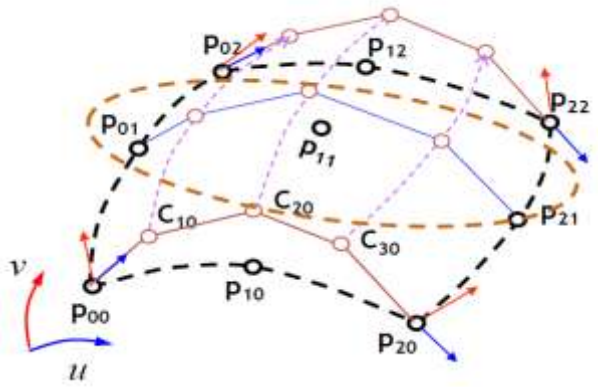


그림 2 - u 방향에 대하여 control point 계산

위의 그림을 보면 알 수 있듯이, $\mathbf{c}_{10}$, $\mathbf{c}_{20}$, $\mathbf{c}_{30}$는 u 방향의 B-spline curve 중 첫 번째 curve의 control point가 되는 것이다. 각 curve 당 control point가 (m+2)개 만큼 계산되는 것을 알 수 있다. 단, 유의해야 할 점이 있다. Control point를 계산할 때에는 knot 간격이 필요하다. Knot 간격은 curve 위의 point들간의 간격에 의해 결정되는데, curve마다 curve 위의 point들간의 간격이 다르므로 각 curve의 knot 간격은 다르게 된다. Surface에 대하여 가시화를 할 때에는 이에 대한 부분을 통합해주어야 한다. 이유는 Cox-de Boor recurrence formula의 적용에 따른 것인데 이에 대한 구체적인 설명은 후에 한다. 따라서 knot 간격을 평균을 하고 평균한 knot 간격을 통해 각 curve의 control point를 계산한다.

2) u 방향의 control point들을 v 방향으로 연결한 B-spline curve의 control point 계산

- 이제 위에서 계산된 u 방향의 control point를 v 방향으로 연결한 B-spline curve의 control point를 계산해보자. 위에서 언급했듯이, u 방향의 control point $\mathbf{c}_{ij}$는 각 curve 당 (m+2)개가 계산된다. 따라서 $\mathbf{c}_{ij}$를 v 방향으로 연결한다면 이 때 curve 위의 point의 개수는 n개가 되며, 이러한 curve의 개수는 (m+2)개가 되는 것이다. 이에 대해서도 m=3, n=3인 간단한 예제를 통해서 구체적으로 살펴보자. 이에 대한 그림은 아래의 그림과 같다.

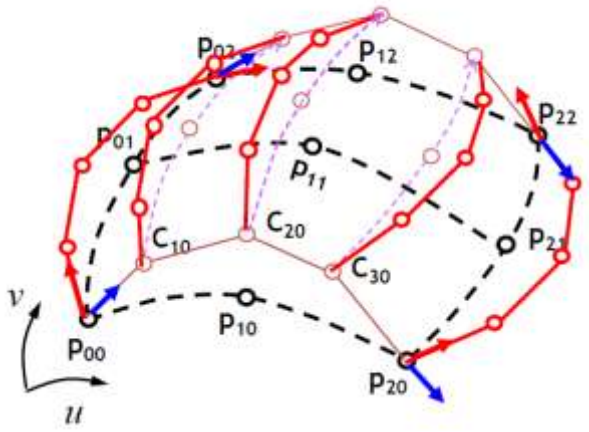


그림 3 - u 방향의 control point를 v 방향으로 연결한 B-spline curve의 control point 계산

위의 그림을 살펴 보면 u 방향의 B-spline curve의 control point는 $\mathbf{c}_{ij}$이다. 이를 v 방향으로 연결한다면 다음과 같다. 우선 v 방향으로 연결한 첫 번째 B-spline curve의 curve 위의 point는 $\mathbf{c}_{00}$, $\mathbf{c}_{01}$, $\mathbf{c}_{02}$이 된다. 이를 통해서 이 curve에 대한 control point를 계산한다. 마찬가지로 v 방향으로 연결한 두 번째 B-spline curve의 curve 위의 point는 $\mathbf{c}_{10}$, $\mathbf{c}_{11}$, $\mathbf{c}_{12}$가 되며, 이를 통해서 이 curve에 대한 control point를 계산한다. 세 번째, 네 번째, 다섯 번째 curve에 대해서도 같은 방법을 적용한다. 위에서 언급했듯이 row의 크기에 2를 더한 만큼의 curve에 대해서 계산을 수행해야 한다. 이렇게 계산된 control point가 위의 그림에서 빨간색 point가 되며, 이 point가 전체 B-spline surface에 대한 control point $\mathbf{d}_{ij}$가 되는 것이다. 물론 control point를 계산할 때에 curve 마다 knot 간격이 다르기 때문에

이를 각 curve에 대하여 knot 간격을 평균을 하고 평균한 knot 간격으로 control point를 계산해야 한다.

3) Cox-de Boor recurrence formula의 적용

- 이제 위에서 계산된 control point $\mathbf{d}_{ij}$를 Cox-de Boor recurrence formula에 적용하여 특정 u, v에 대한 design point **r**(u,v)를 계산해보자. 먼저 우리는 Cox-de Boor recurrence formula의 basis 함수 $N_i^n(u)$의 n을 3으로 한다. 즉 surface를 구현하기 위해서 생성하는 curve의 차수를 3차로 한다. 아래의 그림을 살펴 보자.

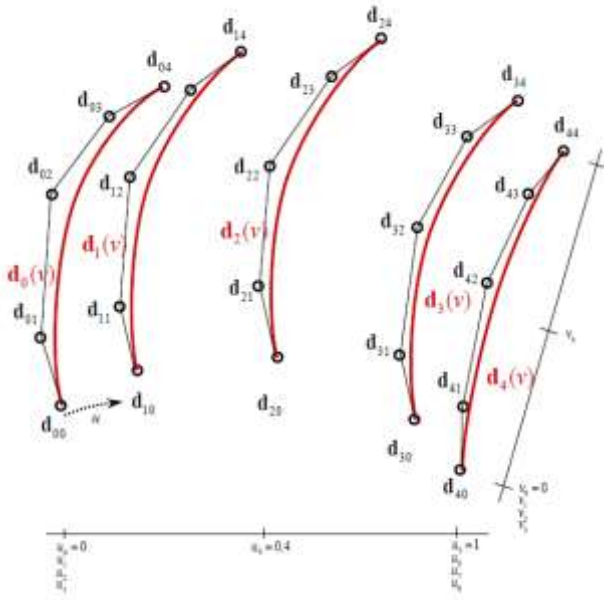


그림 4 - 특정 v에 대한 u 방향의 B-spline curve의 control point

먼저 **r**(u,v)를 구하기 위해서는 특정 v에 대해서 u 방향의 B-spline curve의 control point $\mathbf{d}_i(v)$를 계산해야 한다. 위의 그림은 m=3, n=3인 경우에 대해서 이를 표현한 것이다. 이에 대한 계산은 v와 v 방향으로 연결한 B-spline curve의 control point $\mathbf{d}_{ij}$를 Cox-de Boor recurrence formula에 대입함으로써 계산할 수 있으며, 이는 아래의 식과 같다.

$$\mathbf{d}_i(v) = \sum_{j=0}^{(n+1)} \mathbf{d}_{ij} N_j^3(v)$$

$$\therefore \begin{pmatrix} \mathbf{d}_0(v) \\ \vdots \\ \mathbf{d}_{m+1}(v) \end{pmatrix} = \begin{pmatrix} \mathbf{d}_{00} & \cdots & \mathbf{d}_{0(n+1)} \\ \vdots & \ddots & \vdots \\ \mathbf{d}_{(m+1)} & \cdots & \mathbf{d}_{(m+1)(n+1)} \end{pmatrix} \begin{pmatrix} N_0^3(v) \\ \vdots \\ N_{(n+1)}^3(v) \end{pmatrix} \qquad (2.3)$$

위의 식을 통해서 특정 v에 대한 u 방향의 B-spline curve의 control point $\mathbf{d}_i(v)$를 계산할

수 있고, 이제 특정 u와 $\mathbf{d}_i(v)$를 Cox-de Boor recurrence formula에 대입함으로써 특정 u, v에 대한 design point **r**(u,v)를 계산할 수 있다. 이에 대해서 m=3, n=3인 경우를 표현한 것이 아래의 그림이다.

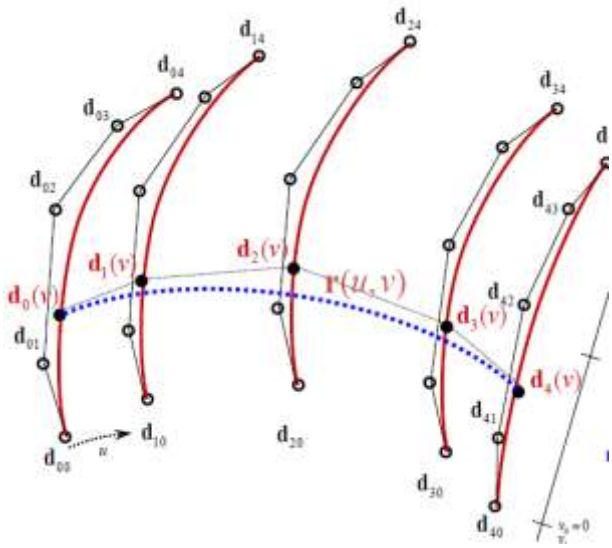


그림 5 - 특정 u와 control point $\mathbf{d}_i(v)$를 통한 design point r(u,v) 계산

이 과정을 수식으로 표현하면 다음과 같다.

$$\mathbf{r}(u,v) = \sum_{i=0}^{(m+1)} N_i^3(u)\mathbf{d}_i(v)$$

$$\therefore \mathbf{r}(u,v) = \begin{pmatrix} N_0^3(u) & \cdots & N_{(m+1)}^3(u) \end{pmatrix} \begin{pmatrix} \mathbf{d}_0(v) \\ \vdots \\ \mathbf{d}_{m+1}(v) \end{pmatrix} \tag{2.4}$$

이제 식 (2.3)과 식 (2.4)를 합치면 다음과 같다.

$$\mathbf{r}(u,v) = \sum_{i=0}^{(m+1)}\sum_{j=0}^{(n+1)} N_i^3(u)\mathbf{d}_{ij}N_j^3(v)$$

$$\therefore \mathbf{r}(u,v) = \begin{pmatrix} N_0^3(u) & \cdots & N_{(m+1)}^3(u) \end{pmatrix} \begin{pmatrix} \mathbf{d_{00}} & \cdots & \mathbf{d_{0(n+1)}} \\ \vdots & \ddots & \vdots \\ \mathbf{d_{(m+1)}} & \cdots & \mathbf{d_{(m+1)(n+1)}} \end{pmatrix} \begin{pmatrix} N_0^3(v) \\ \vdots \\ N_{(n+1)}^3(v) \end{pmatrix} \tag{2.5}$$

이제 식 (2.5)를 통해서 surface의 control point $\mathbf{d}_{ij}$와 특정 u, v를 Cox-de Boor recurrence formula에 대입함에 따라 design point **r**(u,v)를 계산할 수 있다. 그런데 유의할 점이 있다. 위의 알고리즘은 모든 v 방향의 B-spline curve에 대해서 같은 $N_i^3(v)$ 를 적용한다. 마찬

가지로 모든 u 방향의 B-spline curve에 대해서도 같은 $N_i^3(u)$ 을 적용한다. 하지만 basis function $N_i^n(u)$ 에 대한 식은 다음과 같다.

$$N_i^n(u) = \frac{u - u_{i-1}}{u_{i+n-1} - u_{i-1}} N_i^{n-1}(u) + \frac{u_{i+n} - u_i}{u_{i+n} - u_i} N_{i+1}^{n-1}(u)$$

$$N_i^0(u) = \begin{cases} 1 & \text{if } u_{i-1} \le u < u_i \\ 0 & \text{else} \end{cases} \qquad (2.6)$$

위의 식에서 u_i는 curve의 knot이다. 이 knot은 curve 위의 point들간의 간격으로 결정되는데, 각 curve의 curve 위의 point들간의 간격이 다 다르기 때문에 $N_i^n(u)$ 는 curve마다 다른 식이 된다. 이를 보정해주기 위해서 v 방향의 각 curve의 v_i를 평균을 한 후에 이 평균한 v_i를 통해서 $N_i^n(v)$ 를 구현한다. 마찬가지로 u 방향의 curve들에 대해서도 각 curve의 u_i를 평균을 한 후에 이 평균한 u_i를 통해서 $N_i^n(u)$ 를 구현한다.

3.2. Code 분석

- BsplineSurface class는 다음의 멤버 함수로 이루어져 있다.

1) calc_m_dcofRow() 함수

- 이 함수는 변수 m_dcofRow를 계산하는 함수이다. 이 함수는 CASDMFCDoc class에서 입력 받은 surface 위의 point의 정보를 갖고 있는 변수 InputPoint를 매개변수로 입력받는다. 이를 통해서 계산을 수행하여 m_dcofRow 변수에u 방향으로 연결한 B-spline curve들의 knot 간격을 평균한 값이 입력한다. Class의 멤버 변수인 m_bsRowCurve는 u 방향으로 연결한 curve에 대한 정보가 들어 있는 Bspline class 변수이다. 따라서 이 변수에 대해서 Bspline class의 멤버 함수인 calc_c()함수를 호출함으로써 각각의 u 방향으로 연결된 curve들의 knot간격을 계산한다. 위에서 언급했듯이 knot 간격은 Bspline class의 멤버 변수인 c에 계산된다. 이렇게 계산된 각각의 curve의 c를 평균을 내어 m_dcofRow 변수에 저장하는 것이다. 저장한 후에는 다시 각 curve의 변수 c에 이 변수를 저장함으로써 모든 curve에 대하여 평균을 한 동일한 knot 간격이 존재하도록 한다.

2) Interpolation() 함수

```
void BsplineSurface::Interpolation(Vector **InputPoint)
{
            int i=0;
            int j=0;


            calc_m_dcofRow(InputPoint);

            for(i=0; i<m_nNumofCol; i++)
            {
```

```
                m_bsRowCurve[i].Interpolation(temp_Point[i]);
        }

        for(i=0; i<m_nNumofRowControl; i++)
        {
                for(j=0; j<m_nNumofCol; j++)
                {
                        temp_ControlPoint[i][j] = m_bsRowCurve[j].m_vControlPoint[i];
                }
        }

        for(i=0; i<m_nNumofColControl; i++)
        {
                m_dcofCol[i] = 0.0;
        }

        for(i=0; i<m_nNumofRowControl; i++)
        {
                m_bsColCurve[i].calc_c(temp_ControlPoint[i]);
                for(j=0; j<m_nNumofColControl; j++)
                {
                        m_dcofCol[j] += m_bsColCurve[i].c[j];
                }
        }

        for(i=0; i<m_nNumofColControl; i++)
        {
                m_dcofCol[i] = m_dcofCol[i]/(double)m_nNumofRowControl;
        }

         for(i=0; i<m_nNumofRowControl; i++)
        {
                for(j=0; j<m_nNumofColControl; j++)
                {
                        m_bsColCurve[i].c[j] = m_dcofCol[j];
                }
        }

        for(i=0; i<m_nNumofRowControl; i++)
        {
                m_bsColCurve[i].Interpolation(temp_ControlPoint[i]);
        }

        for(i=0; i<m_nNumofRowControl; i++)
        {
                for(j=0; j<m_nNumofColControl; j++)
                {
                        m_vControlPoint[i][j] = m_bsColCurve[i].m_vControlPoint[j];
                }
        }
}
```

- 이 함수는 CASDMFCDoc class에서 입력 받은 surface 위의 point의 정보를 갖고 있는 변수 InputPoint를 통해서 surface의 control point에 대한 정보를 갖는 m_vControlPoint 변수를 계산하는 과정을 수행한다. 먼저 u 방향으로 연결한 B-spline curve의 정보를 갖고 있는 변수 m_bsRowCurve에 대해서 Bspline class의 멤버 함수인 Interpolation() 함수를 호출함으로써, u 방향으로 연결된 curve의 control point를 계산한다. 이 때, temp_Point 변수를 이 함수에 입력한다. 이 변수는 우리가 Bspline class에 point를 입력할 때, row 방향의 point들을 입력해야 하므로, 이를 용이하게 입력할 수 있도록 하게 해준다. 따라서 InputPoint 변수의 row와 column을 바꿔서 temp_Point 변수에 대입한다. 이렇게 계산된 u 방향의 curve들의 control point에 대한 정보는 Bspline class의 m_vControlPoint 변수에 계산되며, 이를 변수 temp_ControlPoint에 대입한다. 그 다음에 v 방향으로 연결한 B-spline curve의 knot 간격들을 평균을 하여 m_dcofCol 변수에 할당하고 이를 다시 각 curve의 변수 c에 할당한다. 과정은 위의 calc_m_dcofRow() 함수와 같고 v 방향으로 연결한 B-spline curve에 대한 정보는 BsplineSurface class의 멤버 변수이자, Bspline class 변수인 m_bsColCurve에 저장되어 있다. 따라서 이후에 m_bsColCurve의 멤버 함수인

Interpolation() 함수에 temp_ControlPoint를 입력함에 따라 최종적인 surface의 control point를 계산하고 계산된 정보를 BsplineSurface의 멤버 변수인 m_vControlPoint에 할당한다.

3) Evaluation() 함수

```
Vector BsplineSurface::Evaluation(double u, double v, Vector **ControlPoint)
{
		int i=0;
		int j=0;

		for(i=0; i<m_nNumofCol; i++)
		{
			for(j=0; j<m_nNumofRowControl; j++)
			{
				m_bsRowCurve[i].c[j] = m_dcofRow[j];
			}
		}

		for(i=0; i<m_nNumofRowControl; i++)
		{
			for(j=0; j<m_nNumofColControl; j++)
			{
				m_bsColCurve[i].c[j] = m_dcofCol[j];
			}
		}

		Vector DesignPoint;
		DesignPoint.x = 0.0;
		DesignPoint.y = 0.0;
		DesignPoint.z = 0.0;
		Vector *temp_RowVector;
		temp_RowVector = new Vector[m_nNumofRowControl];

		for(i=0; i<m_nNumofRowControl; i++)
		{
			temp_RowVector[i] = m_bsColCurve[0].Solution(v, ControlPoint[i]);
		}

		DesignPoint = m_bsRowCurve[0].Solution(u, temp_RowVector);

		if(temp_RowVector)
		{
			delete []temp_RowVector;
			temp_RowVector = NULL;
		}

		return DesignPoint;
}
```

- 이 함수는 특정 u, v에 대한 design point를 계산하는 기능을 수행하는 함수이다. 따라서 매개변수로는 u에 대한 u, v에 대한 변수 v, 그리고 계산된 control point에 대한 정보를 갖고 있는 변수 ControlPoint가 있다. 제일 먼저 작성해야 할 code는 알고리즘에서 언급했듯이 knot 간격을 평균하여 구현한 knot을 통해 Cox-do Boor recurrence formula를 구현하는 것이다. 이를 위해서 위에서 평균한 knot 간격에 대한 정보를 가지고 있는 변수인 m_dcofRow, m_dcofCol을 각각의 curve의 knot 간격을 뜻하는 변수, 즉 Bspline class의 멤버 변수인 c에 대입한다. 이제 u 방향으로 연결된 curve에 대한 정보를 가지고 있는 m_bsRowCurve 변수의 knot 간격은 평균한 값이 되며, 이는 v 방향에 대해서도 마찬가지이다. 이제 알고리즘에 따라서 design point를 계산하여 변수 DesignPoint에 할당한다. 이 함수는 Vector type의 함수로서, 계산된 DesignPoint를 return하도록 한다. DesignPoint를 계산할 때에는 먼저 특정 v에 대한 u 방향으로 연결된 B_spline curve의 control point를 계산해야 하는데, 이 때, curve에 대한 design point를 계산하는 Bspline class의 멤버 함수

인 Solution() 함수에 v와 surface의 v 방향의 control point를 대입함으로써 계산을 수행한다. Solution() 함수를 호출할 때 사용되는 class 변수는 m_bsRowCurve[0]을 사용한다. 사실 모든 m_bsRowCurve의 변수들에 평균한 knot 간격이 대입되어 있으므로, 어느 것을 사용해도 상관이 없다. 계산된 결과, 즉 특정 v에 대한 u 방향의 curve의 control point를 temp_RowVector 변수에 저장한다. 다시 이 변수와 u를 m_bsColCurve[0]에 대하여 호출한 Solution() 함수에 대입함으로써 최종적인, Surface에 대한 design point를 계산하고, 이를 변수 DesignPoint에 대입한다.

4. CASDMFCDoc class

4.1. Code 분석

- CASDMFCDoc class는 다음의 멤버 함수로 이루어져 있다.

1) OnFileOpen() 함수

- 이 함수는 text file을 통해서 row의 크기와 column의 크기, 그리고 surface 위의 point에 대한 정보를 입력 받아 각각 이 class의 멤버 변수인 nNumofRow, nNumofCol, PointonCurve에 할당하는 과정을 수행한다. 일반적인 format은 6차 project와 동일하다. 따라서 여기서도 이 class의 멤버 변수인 정보가 입력되면 INDEX를 0에서 1로 바꿔 주어 후에 정보가 입력되었을 때만 계산을 수행하도록 하게 한다. 단 유의해야 할 것은 row의 크기와 colume의 크기가 3 이상이어야 하므로 만약 이 조건을 충족하지 않으면 경고 메시지를 출력하며 정보가 입력되지 않도록 한다. 마지막으로 정보가 입력되면 계산을 수행하는 함수인 Calculatio() 함수를 호출함으로써 계산을 수행하게 한다.

2) Calculation() 함수

- 이 함수는 입력 받은 surface 위의 point에 대한 정보를 통해 control point를 계산하고 u, v에 대한 design point를 계산하는 기능을 수행한다. 따라서 BsplineSurface class 변수 test를 선언하고 이 변수에 대하여 BsplineSurface class의 Interpolation() 함수를 이 함수에 PointonCurve 변수를 입력하여 호출함으로써 control point를 계산하여 CASDMFCDoc class의 멤버 변수인 ControlPoint에 할당한다. 그 다음 test 변수에 대하여 BsplineSurface class의 Evaluation() 함수를 이 함수에 ControlPoint를 입력하여 호출함으로써 design point를 계산하여 CASDMFCDoc class의 멤버 변수인 DesignPoint에 할당한다. 그런데 중요한 것은 Evaluation() 함수에는 특정 u, v에 대한 변수들을 역시 입력해야 하는데, 이 변수들을 0에서 1까지 미소하게 변화시키면서 입력한 후, 각각의 경우에 대하여 design point를 계산해야 한다. 이 때, 얼마나 미소하냐를 결정하는 변수들이 바로 CASDMFCDoc class의 멤버 변수들인 nRowDelta, nColDelta이다. 각각 u 방향, v 방향에 대한 것이며, int

type의 변수이다. 만약 nRowDelta 변수가 100이라면 u를 미소하게 변화시키는 정도는 이 값의 역수인 0.01이 된다. 따라서 DesignPoint 변수는 nRowDelta보다 1이 큰 값으로 동적 할당하고, 이를 다시 nColDelta보다 1 큰 값으로 동적 할당하여 (nRowDelta+1)X(nColDelata+1)의 크기를 갖는 변수로 선언해야 한다. 각각의 경우에 대하여 1을 더해주는 이유는 간격이 100이라면 point는 101개이기 때문이다. Design point를 계산할 때에는 이중 for문을 각각 (nRowDelta+1), (nColDelta+1)만큼 수행하여 각각의 u, v에 대한 design point를 계산하고 이를 변수 DesignPoint에 할당해준다. 초기에 nRow Delta와 nColDelta는 생성자에서 각각 50의 값을 갖도록 하였다.

3) OnMovingpointPointoncurve() 함수, OnMovingpointControlpoint() 함수

- 전자는 dialog 창을 통해 초기에 입력 받은 surface 위의 point들에 대한 정보를 가지고 있는 변수 PointonCurve 변수를 변경함으로써 surface 위의 point들을 변경하였을 때, 어떻게 surface가 변화하는지를 알아보는 함수이다. dialog 창에 대한 class는 MovingPointonCurve class이며, 이 class는 멤버 변수로 변경하고자 하는 point의 row에 대한 index에 대한 정보를 가지고 있는 변수 m_nIndexofRow, colume에 대한 index에 대한 정보를 가지고 있는 변수 m_nIndexofCol, 그리고 변경하고자 하는 point의 좌표를 의미하는 변수들 m_dPositionofx, m_dPositionofy, m_dPositioofz이 있다. 이 변수들에 dialog 창을 통해 원하는 값을 입력하고 이를 통해서 다시 PointonCurve 변수를 변경한 후, Calculation() 함수를 호출하여 변화된 형상에 대하여 계산을 수행한다.
  후자는 control point의 정보를 가지고 있는 변수 ControlPoint를 변경하여 surface가 어떻게 변화하는지를 알아보는 함수이다. ControlPoint에 변경된 값을 입력 하는 과정은 전자의 경우와 일치한다. 하지만 변경한 후, Calcuation() 함수를 호출해서는 안될 것이다. Calculation() 함수는 PointonCurve로부터 계산을 시작하기 때문이다. 따라서 test2라는 새로운 BsplinSurface class 변수를 선언하고 이에 대해서 InterPolation() 함수를 이 함수에 PointonCurve를 입력하여 호출하여 원하는 knot 간격과 초기 control point를 계산하고 d 여기에 다시 변경된 control point를 ControlPoint 변수에 대입한 후, test2 변수에 대해서 Evalauation() 함수를 이 변경된 ControlPoint 변수를 입력하여 호출함으로써 control point의 변화에 따른 surface의 변화를 계산을 수행할 수 있도록 한다.

4) OnChangesizeInputscaleratio

```
void CCASDMFCDoc::OnChangesizeInputscaleratio()
{
	int i=0;
	int j=0;

	InputScaleRatio dlg;

	if(INDEX == 1)
	{
		if(dlg.DoModal() == IDOK)
		{
			if(dlg.m_dScaleRatio > 0)
			{
				for(i=0; i<nNumofRow; i++)
				{
```

```
                                                for(j=0; j<nNumofCol; j++)
                                                {
                                                        PointonCurve[i][j] = dlg.m_dScaleRatio * PointonCurve[i][j];
                                                }
                                        }

                                        Calculation();
                                }
                                else
                                {
                                        AfxMessageBox("Scale Ratio가올바르지않습니다.");
                                }
                        }
                }
                else
                {
                        AfxMessageBox("아직Point on Curve가입력되지않았습니다.");
                }

                // TODO: 여기에명령처리기코드를추가합니다.
}
```

- 이 함수는 dialog 창을 통해서 scale ratio를 입력 받아 이 입력된 값을 surface 위의 point에 대한 정보를 가지고 있는 변수 PointonCurve에 곱해줌으로써, 가시화한 형상을 확대, 축소하도록 하는 기능을 수행한다. Dialog 창에 대한 class는 InputScaleRatio class이며, 이 함수는 scale ratio에 대한 변수 m_dScaleRatio를 멤버 변수로 갖는다. Dialog 창을 통해 이 변수를 입력 받아서 PointonCurve에 곱해주고 이 변경된 PointonCurve에 대해서 다시 Calculation() 함수를 호출함으로써 형상을 확대, 또는 축소하도록 한다.

5) OnChangedeltaInput() 함수

```
void CCASDMFCDoc::OnChangedeltaInput()
{
        InputDelta dlg;

        if(INDEX == 1)
        {
                if(dlg.DoModal() == IDOK)
                {
                        if(dlg.m_nRowDelta > 0 && dlg.m_nColDelta > 0)
                        {
                                nRowDelta = dlg.m_nRowDelta;
                                nColDelta = dlg.m_nColDelta;

                                Calculation();
                        }
                        else
                        {
                                AfxMessageBox("Row Delta 또는Column Delta가올바르지않습니다.");
                        }
                }
        }
        else
        {
                AfxMessageBox("아직Point on Curve가입력되지않았습니다.");
        }

        // TODO: 여기에명령처리기코드를추가합니다.
}
```

- 이 함수는 dialog 창을 통해서 위에서 언급한 u방향으로, 또는 v 방향으로 격자 간격을 결정하는 변수들 nRowDelta, nColDelta를 변경하는 기능을 수행하는 함수이다. Dialong 창에 대한 class는 InputDelta class이며, 이 class는 변경하고자 하는 격자 간격을 결정하는 변수들인 m_nRowDelta, m_nColDelta를 멤버 변수로 갖고 있으며, 이를 dialog 창을

통해 입력 받고 이를 다시 nRowDelta, nColDelta에 할당한 다음, Calculation() 함수를 호출하여 변경된 격자 간격에 대하여 계산을 수행하도록 한다.

5. OpenGLVeiw class

5.1. Code 분석

- 이 class의 멤버 함수들은 6차 project와 같은 구성으로 되어 있다. 몇 가지 다른 점이 있는데, 그 중 한 가지는 PushName()~PopName() 함수를 통한 point들에 대한 이름 부여 방식이다. 먼저 surface 위의 point들에 대해서는 만약 row0column와 같은 숫자로 이름을 부여하였다. 즉 만약 row 값이 1, column 값이 1이라면 그 point의 이름은 101이 되는 것이다. Control point는 row00column와 같은 숫자로 이름을 부여하였다. Row=1, column=1인 control point의 이름은 1001이 되는 것이다. Rowcolumn와 같은 방식의 이름을 부여하지 않은 이유는 point의 개수가 각 방향으로 10개가 넘는 경우가 존재하기 때문에 정확히 이름을 표현할 수 없기 때문이다. 이번 project에서도 point에 대해 이름을 부여한 후, 이에 대해서 picking 기능(선택 시, point의 색깔이 빨간색으로 변화)을 구현하였다(방식은 6차 project와 마찬가지로 picking 시, m_uClicked 변수에 point의 이름이 할당되는 사실을 이용하였다.). 또한 6차 project에서와 마찬가지로 picking 시, picking된 point의 이름이 m_uClicked 변수에 할당되어 있다는 사실을 이용하여, 선택한 point의 raw와 column의 값을 알아낸 다음, 이를 통해서 이 point의 좌표를 dialog창을 통해서 출력하는 기능 역시 구현하였다.
  두 번째로 6차와 다른 점은 6차에서와 달리 surface를 가시화한다는 점이다. 이에 대해서는 구체적으로 살펴 보자.

  1) DrawSurface() 함수

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

		Vector vec1;
		Vector vec2;
		Vector Normal;

		if(GetDocument()->INDEX == 1)
		{
			glColor3f(red3,green3,blue3);
			glPushMatrix();
			glBegin(GL_QUADS);
			for(i=0; i<GetDocument()->nRowDelta; i++)
			{
				for(j=0; j<GetDocument()->nColDelta; j++)
				{
					vec1 = GetDocument()->DesignPoint[i+1][j] - GetDocument()->DesignPoint[i][j];
					vec2 = GetDocument()->DesignPoint[i][j+1] - GetDocument()->DesignPoint[i][j];

					Normal = vec1 * vec2;
					Normal.Normalize();

					glNormal3f(Normal.x, Normal.y, Normal.z);

					glVertex3f(GetDocument()->DesignPoint[i][j].x, GetDocument()->DesignPoint[i][j].y,
GetDocument()->DesignPoint[i][j].z);
```

```
                                        glVertex3f(GetDocument()->DesignPoint[i+1][j].x, GetDocument()->DesignPoint[i+1][j].y,
GetDocument()->DesignPoint[i+1][j].z);
                                        glVertex3f(GetDocument()->DesignPoint[i+1][j+1].x, GetDocument()->DesignPoint[i+1][j+1].y,
GetDocument()->DesignPoint[i+1][j+1].z);
                                        glVertex3f(GetDocument()->DesignPoint[i][j+1].x, GetDocument()->DesignPoint[i][j+1].y,
GetDocument()->DesignPoint[i][j+1].z);
                                    }
                            }
                            glEnd();
                            glPopMatrix();
                    }
            }
```

- 위의 함수는 surface를 가시화하는 함수이다. 먼저 u, v를 0부터 1까지 미소하게 변화시켜 가면서 계산한 design point의 정보를 가지고 있는 변수를 CASDMFCDoc class 변수 GetDocument()를 통해서 이 class의 멤버 변수인 DesignPoint로 사용한다. 이 design point들을 사각형으로 연결하기 위해서는 glBegin(GL_QUADS)~glEnd() 함수를 사용한다. 이 함수 사이에 point의 좌표를 입력하면 4개씩 끊어서 사각형을 생성한다. 유의해야 할 점은 생성한 사각형이 어느 방향을 뜻하는지를 나타낼 수 있도록 사각형의 정확한 normal vector를 설정하는 것이다. 아래의 그림을 살펴보자.

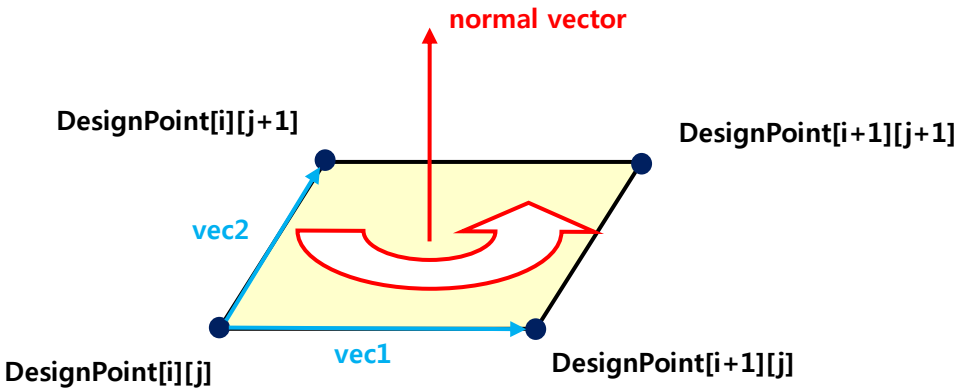


그림 6 - 사각형 단면의 noraml vector 설정

위의 그림은 사각형 단면의 normal vector를 설정하는 방법을 표현한 것이다. 먼저 사각형이 위 쪽을 향하기 위해서는 normal vector 역시 위로 향해야 한다. 따라서 normal vector를 계산하기 위해서는 vec1 X vec2와 같이 vec1과 vec2를 외적함으로써 구할 수 있다. vec1은 DesignPoint[i+1][j]에서 DesignPoint[i][j]를 빼서 구할 수 있고, vec2는 DesingPoint[i][j+1]에서 DesingPoint[i][j]를 빼서 구할 수 있다. 이렇게 구한 normal vector를 변수 Normal에 저장한 후, 크기를 1로 하기 위해서 Normalize() 함수를 호출한다. 이제 이 변수를 glNormal3f() 함수에 입력하여 해당 사각형의 normal vector를 구현한다. 또 한 가지 유의해야 할 사실은 뒤에 사각형을 이루는 point 4개를 입력할 때 normal vector에 대해서 오른손 법칙을 적용한 순서대로 point를 입력해야 한다는 사실이다. 위의 그림에서는 반 시계 방향을 의미한다.

이제 이러한 방식으로 이중 for문을 통해 미소한 사각형을 차례대로 가시화한다. for문을 반복하는 횟수는 위에서 언급한 변수들인 GetDocument()의 멤버 변수들 nRowDelta, nColDeta이다.

6. 실행 화면

6.1. "곡면생성예제" file을 입력하였을 때

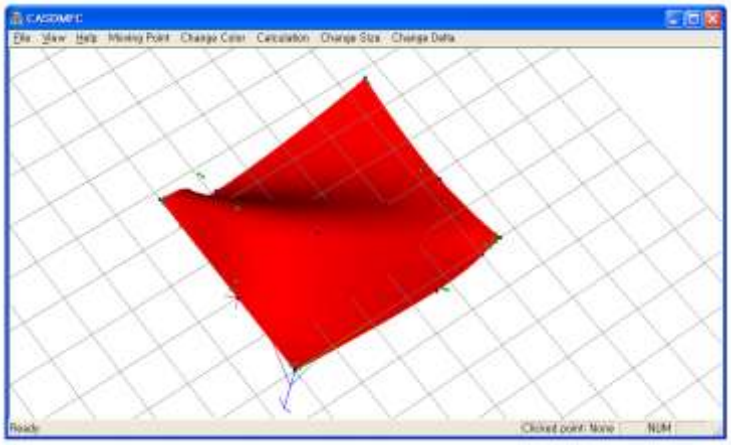


그림 7 - "곡면생성예제"에 대한 가시화 형상

6.2. 300K VLCC의 "전체선형(트랜섬부분포함)" file을 입력하였을 때

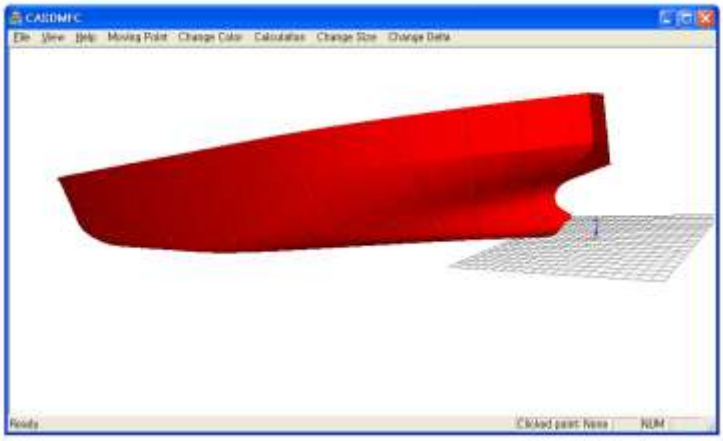


그림 8 - 300k VLCC의 "전체선형(트랜섬부분포함)"에 대한 가시화 형상

6.3. "선미부(트랜섬부분제외)" file을 입력하였을 때

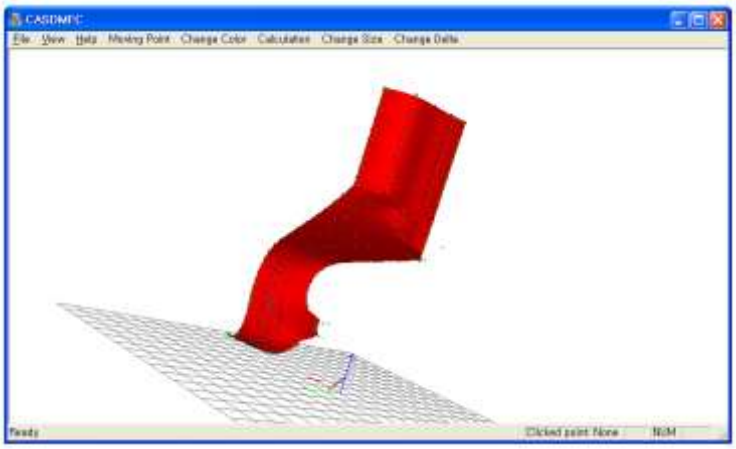


그림 9 - "선미부(트랜섬부분제외)"에 대한 가시화 형상

6.4. 국부적 가시화 – “곡면생성예제”에 대해서

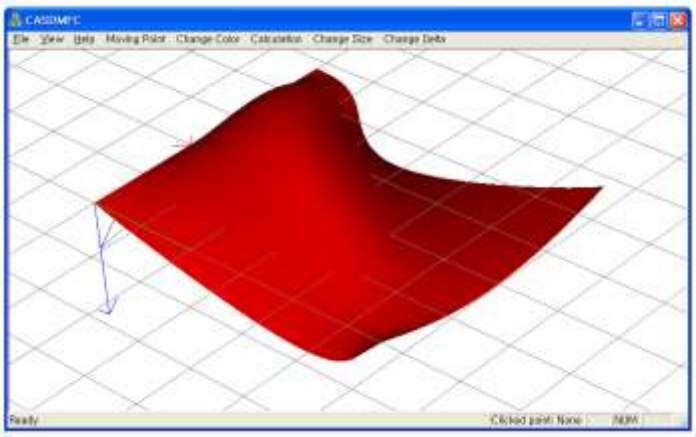


그림 10 - surface만 가시화

6.5. Point의 이동 – “곡면생성예제에 대해서”

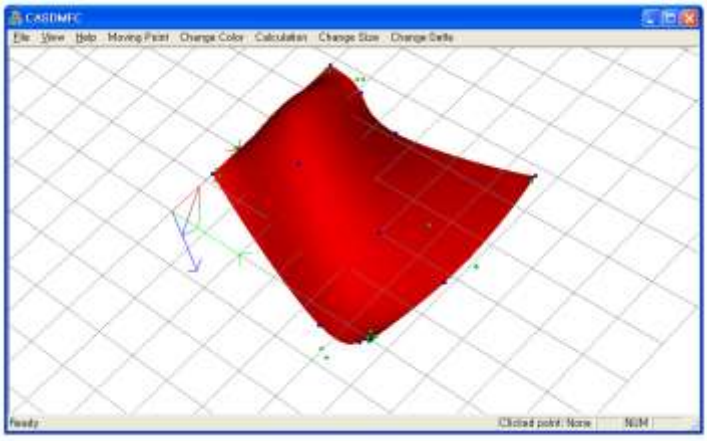


그림 11 - row=1, column=1인 point on curve의 좌표를 (3,0,0)으로 이동한 경우

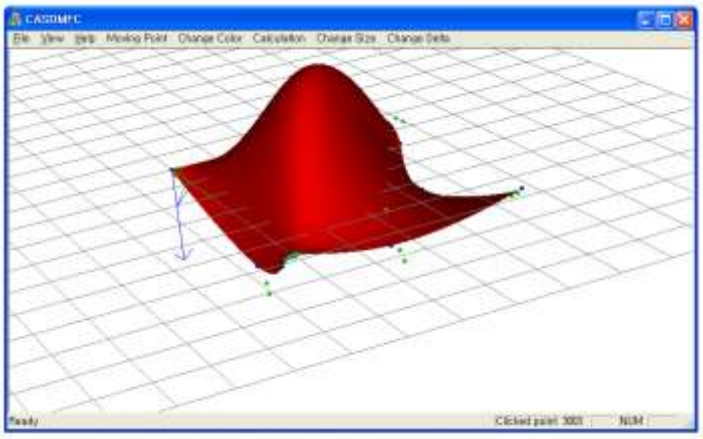


그림 12 - row=3, column=3인 control point의 좌표를 (5,5,-20)으로 이동한 경우

6.6. 색깔 변경 – "곡면생성예제"에 대해서

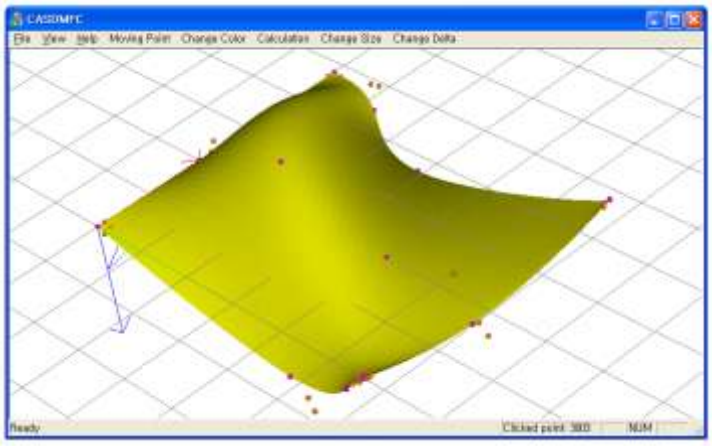
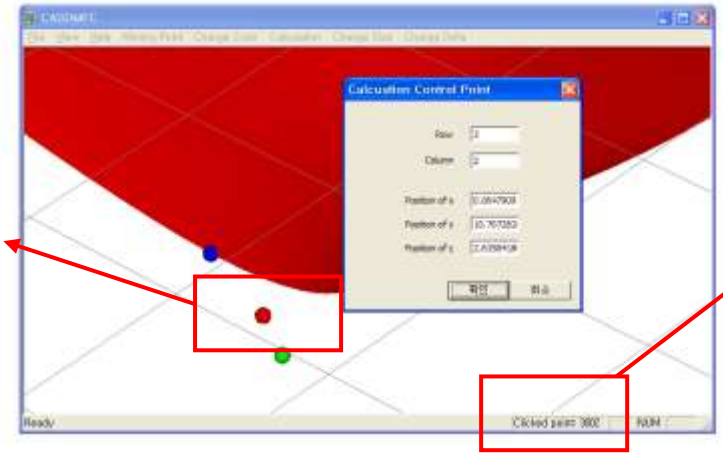


그림 13 - 색깔 변경한 경우

6.7. Point에 대한 정보 계산 – "곡면생성예제"에 대해서

control point가 빨간색으로 변화
–
picking되었음을 의미



3002
–
control point의 이름으로 row=3, column=2임을 의미

그림 14 - row=3, column=2인 control point의 정보 계산

6.8. 가시화 형상의 size 변경 – "곡면생성예제"에 대해서

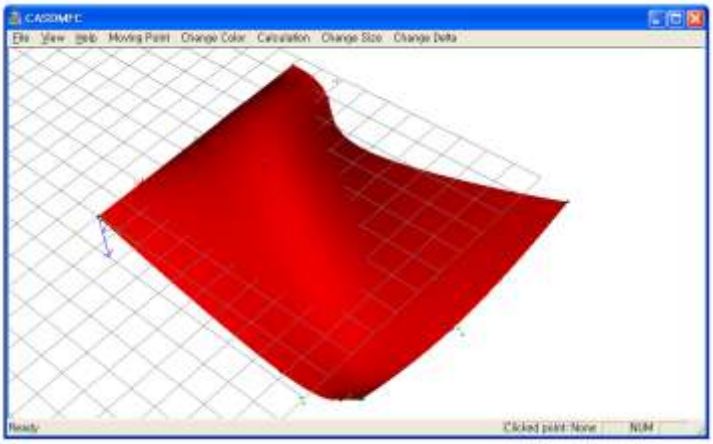


그림 15 - 형상의 size를 2배로 한 경우

6.9. 격자의 개수를 바꾼 경우 – "곡면생성예제"에 대해서

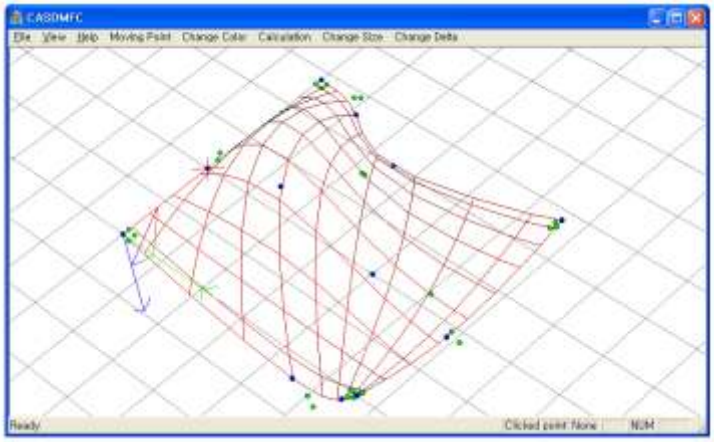


그림 16 - 격자를 10X10으로 한 경우

7. Discussion

7.1. Tangent vector의 계산

$$\mathbf{t_0} = -\frac{2\Delta_2 + \Delta_3}{\Delta_2(\Delta_2 + \Delta_3)}\mathbf{p_0} + \frac{(\Delta_2 + \Delta_3)}{\Delta_2\Delta_3}\mathbf{p_1} + \frac{\Delta_2}{\Delta_3(\Delta_2 + \Delta_3)}\mathbf{p_2}$$

$$\mathbf{t_1} = -\frac{\Delta_{K-4}}{\Delta_{K-5}(\Delta_{K-5} + \Delta_{K-4})}\mathbf{p_{m-3}} - \frac{(\Delta_{K-5} + \Delta_{K-4})}{\Delta_{K-5}\Delta_{K-4}}\mathbf{p_{m-2}} + \frac{2\Delta_{K-4} + \Delta_{K-5}}{\Delta_{K-4}(\Delta_{K-5} + \Delta_{K-4})}\mathbf{p_{m-1}} \quad (2.1)$$

- 위의 식은 Bessel end condition에 의해 curve의 양 끝 point에서 control point를 계산하는 과정이다. 위에서도 언급했듯이 Δ_i는 knot 간격이 아닌 curve 위의 point들간의 간격이다. 다시 말해서 curve 위의 point들간의 간격을 point들간의 간격을 전부 더한 것으로 나누어 0과 1사이의 값으로 scaling하기 전의 값이라는 것이다. Scaling을 하지 않은 이유는 만약에 curve 위의 point의 개수가 너무 많아지면, Δ_i는 매우 작은 0과 1사이의 값을 갖게 되는데, 위의 식 (2.1)의 각 항들은 Δ_i의 -1차 항이므로 tangent vector는 매우 큰 값으로 계산된다. 매우 큰 tangent vector는 surface의 형상을 왜곡시킨다. Δ_i를 curve 위의 point들간의 간격으로 한다면, 보통 1보다 큰 값이 되어 tangent vector가 작아지게 되고, 이에 따라 원하는 surfave의 형상을 계산할 수 있게 된다. 하지만 만약 curve 위의 point들간의 간격이 1보다 작은 경우에는 어떻게 될까? 아래의 그림을 살펴보자.

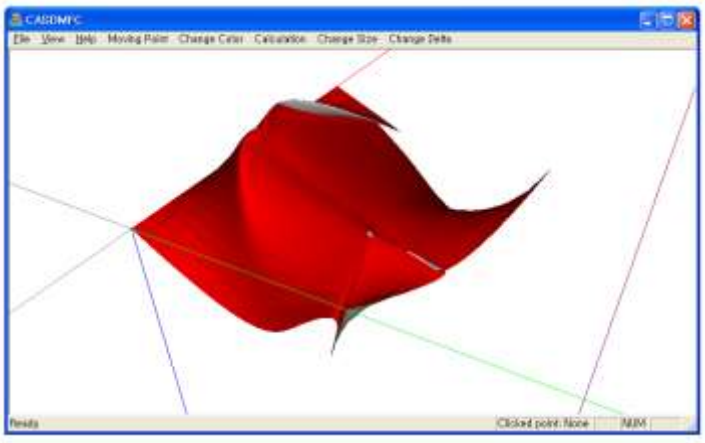


그림 17 - "곡면생성예제"에 대한 가시화 형상을 0.05배로 축소한 것

위의 그림은 "곡면생성예제"에 대하여 가시화한 형상을 0.05배로 축소한 경우이다. 0.05배로 축소하면 surface 위의 point들간의 간격이 매우 줄어들게 되며, 따라서 Δ_i는 1보다 작은 매우 작은 값이 되게 된다. 이런 경우 tangent vector가 위에서 말했듯이 너무 크게 계산되고, 이에 따라 위의 그림과 같이 surface가 찌그러진 형상으로 가시화되게 되는 거이다. 따라서 Bessel end condition을 통해서 가시화를 수행할 때에는 surface 위의 point

간의 간격이 1보다 크도록 point를 측정하는 것이 중요할 것이다.

7.2. Knot 간격을 평균해주는 것에 대한 고찰

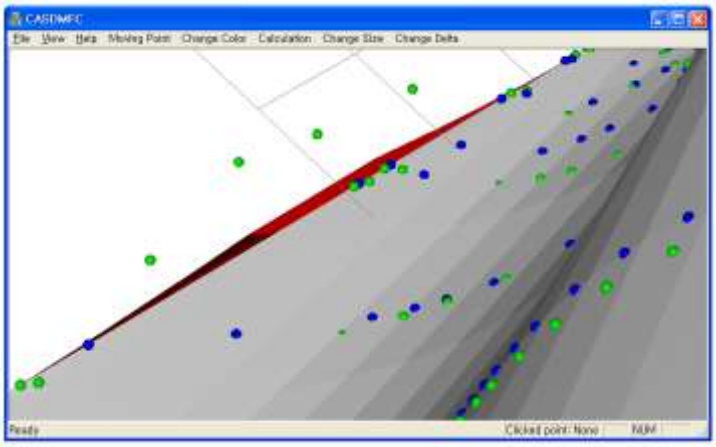


그림 18 - 300K VLCC의 "선형전체(트랜섬부분포함)"에 대한 가시화 형상의 선미부

- 위의 그림은 300K VLCC의 "선형전체(트랜섬부분포함)"의 선미부의 가시화 형상이다. 위의 그림을 잘 살펴 보면 가시화 형상에서 control point가 약간 위쪽 방향(실제로는 –y 방향)으로 치우쳐 계산되어 곡면의 형상이 약간 뒤집히는 것을 볼 수 있다. 특이한 점은 surface의 형상이 surface 위의 point를 모두 지난다는 것이다. 따라서 계산에는 오류가 없다. 단지 우리가 원하는 surface의 형상이 아닌 약간 뒤집힌 형상이 가시화된 것이다. 왜 이러한 상황이 발생할까? 이유는 간단하다. 우리는 curve의 control point를 계산할 때 knot 간격을 평균을 내어 control point를 계산하였기 때문이다. 하지만 이렇게 할 수 있는 데에는 '각 curve위 point들간의 간격이 비슷하다'라는 전제가 필요하다. 위의 뒤집힌 형상에서는 knot 간격이 curve마다 비슷하지 않고 차이가 있어서 surface의 왜곡이 발생한 것이다. 실제로 knot 간격은 curve 위의 point들간의 간격에 의해 결정되며, 이는 curve의 형상에 관여하는 바가 크다. 만약 knot 간격을 평균을 함에 따라 knot 간격이 변화가 심하게 일어난다면 이 평균을 한 knot 간격은 해당 curve의 형상을 제대로 반영하지 못하게 되는 것이며, 이에 따라 surface의 왜곡이 일어나게 되는 것이다. 따라서 surface 위의 point들을 측정할 때에는 최대한 각 curve 위의 point들간의 간격이 비슷하도록 측정하는 것이 중요할 것이다.

8. 후기

- 이번 project는 6차 project를 기반으로 한 것이므로 알고리즘 상으로는 크게 어렵지 않았다. 하지만 Bessel end condition이라든지, knot 간격을 평균을 내주는 과정에 있어서는 심도 있게 그 과정을 검토할 필요가 있었다. 계산이 올바르게 되었음에도 불구하고, surface를 제대로 가시화하지 못하는 경우가 존재했기 때문이다. 사실 6차, 7차 project는 control point로부터 curve 또는 surface 위의 point들을 계산하는 것이 아닌 이의 역 과정의 문

제였기 때문에, 필요한 경우에 원하는 값을 근사하는 방식을 취하였다. 근사를 하는 경우로는 tangent vector를 계산할 때, 또는 knot 간격을 구할 때가 있었다. 이 때에 iteration을 이용하여 계산을 보정하는 과정이 있었다면 조금 더 얻고자 하는 형상에 가깝게 계산을 수행할 수 있었을 것이라고 추측된다. 이에 대한 추가적인 검토와 노력이 필요할 것이다.

이번 project를 통해서 B-spline 알고리즘이 얼마나 체계적이며, 간단한 방법인지를 알 수 있었다. 기존에는 curve에 대한 알고리즘밖에 알 수 없었지만 이번 기회를 통해서 surface에 대해서도 충분한 이해를 할 수 있어서 좋은 기회가 되었던 것 같다. 또한 MFC의 사용, OpenGL의 사용에서도 많은 기능을 다루면서, 이에 대한 활용도에 대한 이해를 높일 수 있었다. 덕택에 앞으로의 MFC에 대한 사용에 있어서 자신감을 키울 수 있었다.

B-spline에 대해서 지금까지 난해하지 않으면서도 구체적으로, 즉 심도 있게 이해할 수 있도록 가르쳐 주신 교수님께 감사드린다. 또한 code에 대해서 체계적으로 설명해주시고, 항상 질문에 친절하게 답변해주시는 조교님께도 감사드린다.