GOVP1199800092

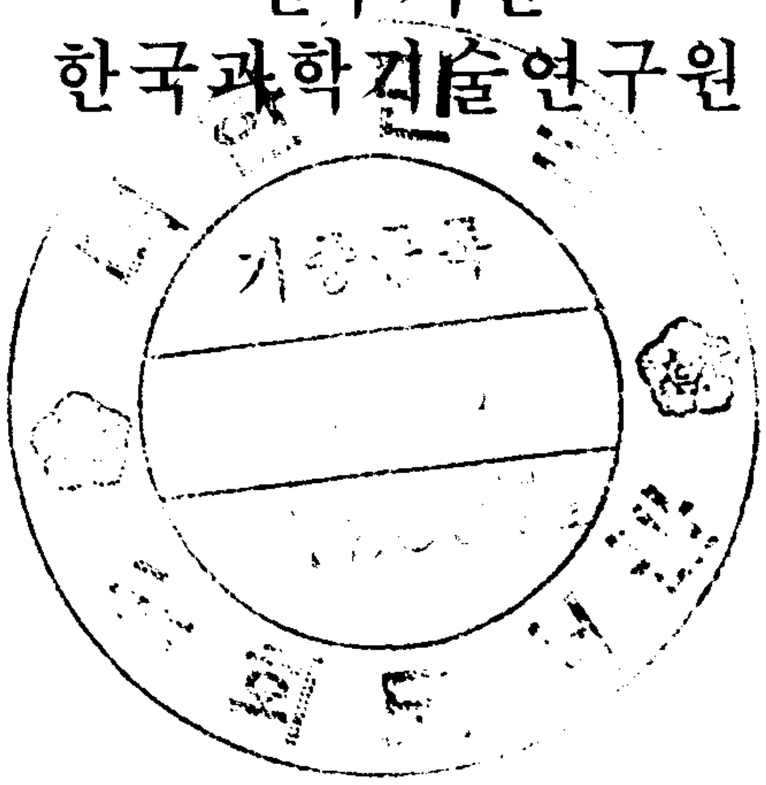
3차원 금형 제작 단축 자동화 시스템 개발

Development of a Automation System for Reducing the Production Time of a 3-dimensional Mold

3차원 Surface Modeling 및 최적가공 S/W 개발

Development of a Software for 3-dimensional Surface Modeling and Optimum Machining

연구기관
한국과학기술연구원



과 학 기 술 처

제 출 문

과 학 기 술 처 장 관 귀하

본 보고서를 "3 차원 금형 제작 단축 자동화 시스템 개발" 과제의 세부과제 "3차원 Surface Modeling 및 최적가공 S/W 개발"의 최종 보고서로 제출합니다.

1997. 7. 31

주관연구기관명 : 한국과학기술연구원

총괄연구책임자 : 박 세 형

박 면 웅

연 구 원 : 손 영 태

전 용 태

이 윤 영

이 희 관

서 지 한

구 미 정

이 유 미

위탁연구기관명 : 경상대학교

위탁연구책임자 : 전 차 수

여 백

요 약 문

Ⅰ. 제목

3차원 Surface Modeling 및 최적가공 S/W 개발

Ⅱ. 연구개발의 목적 및 중요성

90년대 들어서 소비자의 욕구가 다양해지고 제품의 Life Cycle 이 짧아지면서, 어떻게 하면 시장에 빨리 제품을 내어 놓느냐가 생산의 최대 목표가 되었다. 제품개발기간을 단축할 수 있는 기술로 동시공학 (Concurrent Engineering) 의 개념이 도입되고 있다. 동시공학이란 개념설계단계에서 생산의 모든 단계를 고려하여 설계가 이루어 지고, 설계가 끝이나면 모든 단계의 준비가 동시에 이루어 지는 기술로서, 생산준비기간 및 비용을 획기적으로 단축 시킬 수 있는 기술이다. 동시공학을 구현하기 위한 요소기술로는 Engineering Database (Product Model), 형상정보의 표준화 (IGES, STEP), Knowledge-based CAD (Intelligent CAD), CAE, Virtual Manufacturing, Reverse Engineering, Rapid Prototyping, Rapid Tooling 등이 있다. 본 연구는 금형 제작에 있어서 동시공학의 기법을 활용하여 금형 개발기간을 단축하기 위한 과제의 세부과제로 추진된 연구이다. 본 연구에서는 3차원 모형을 Scanning하여 얻어진 Point 정보로부터 곡면을 형성하고, 이에 대한 유동해석을 실시한 후 금형을 설계하고 가공하는 일련의 연구 중 일부이다. 본 연구에서는 이 중에서 Reverse Engineering에 필요한 기술을 개발하기 위한 것이다. 즉, 측정한 데이터로부터 곡면을 형성하고, 이를 가공할 수 있는 데이터를 만들기 위한 것이다.

III. 연구개발 내용 및 범위

본 연구는 3차원금형제작 단축 자동화 시스템 개발의 세부과제로서, 협동과제로 개발되는 Digitizing Machine으로 측정된 데이터를 이용하여 자유곡면을 포함하는 금형의 Core 및 Cavity 면을 신속하게 Modeling 하고 가공할 수 있는 소프트웨어를 개발하기 위한 것이다. Modeling 기능으로는 측정데이터의 Filtering, Reduction 기능, 타 CAD와 시스템과의 Interface 기능, 불규칙적인 측정 점의 보간 기능, Skinning, Sweeping, Blending, Trimming 기능 등이 주로 개발 되었다. 가공기능은 Parametric Cut, Cartesian Cut, 간섭방지, 공구경로 및 가공조건 최적화, 자동황삭 및 잔삭처리 기능이 주로 개발되었다.

IV. 연구개발결과 및 활용에 대한 건의

PC 환경에서 측정된 데이터를 이용하여 곡면을 모델링 하고 가공정보를 생성하는 소프트웨어를 개발하였다. WINDOWS 환경에서 OpenGL Graphics Library와 Visual C++ 프로그래밍 언어를 사용하여, 범용성을 가지는 소프트웨어를 개발하였다. 주요 기능으로는 측정데이터의 입력 및 처리, 곡선, 곡면 모델링, NC 기능이 포함되어 있다. 자료구조는 타 시스템과의 Interface 및 향후 Solid Modeling으로의 확장을 고려하여 NURBS곡면을 형상정의의 기본으로 하고 Topology는 Non-manifold 구조를 갖도록 개발하였다. IGES, DXF, ZES 등을 통하여 타 시스템과의 형상정보 교환도 가능토록 하였다.

개발된 소프트웨어는 향 후 금형업체를 위한 NC전용 소프트웨어, 측정데이터 처리 소프트웨어, NC Controller의 Shop-floor Programming 소프트웨어, 특정제품의 모델링 및 가공 전용 소프트웨어 개발의 기반 소프트웨어로 쓰

일 수 있도록 할 예정이다. 이를 위하여, 프로그램의 Documentation을 더욱 철저히 하고, 데이터 처리 및 계산의 효율을 향상시키며, 프로그램의 Error를 완전히 없애는 작업을 계속 수행할 것이다. 또한 본 소프트웨어를 대학의 연구팀에 나누어 주어 형설설계 관련 연구의 기반이 되도록 하고자 한다. 국내 CAD/CAM 수준에서 범용의 소프트웨어를 개발하여 판매하는 것은 경쟁력이 없으나 전용화되고 차별화된 소프트웨어를 개발하는 경우 국제 시장에서도 경쟁을 할 수 있을 것이다.

여 백

SUMMARY

As the demand of the market tends to be complicated, new tools for reducing the development cycle of products are required. CE (Concurrent Engineering) is a new technology that is thought to be one of the promising candidate for significantly improving the product development process. Among many enabling technologies of CE, reverse engineering is a technology proven to be successful for products with sculptured surfaces. The goal of this project is to develop a surface modeling and machining software that will be used with digitizing machine for reducing the process time of reverse engineering of dies and molds.

The most advanced environment on PC has been built to develop the software. Windows-NT (Window 95) operating system is used with C++ language and OpenGL graphics library. A non-manifold data structure with NURBS geometric representation is used to facilitate the interface with other CAD/CAM softwares and to accomplish the various modeling goals. The main modeling functions developed include data reduction and filtering of digitized points, skinning, sweeping, blending and trimmed surface generation. NC functions include automatic rough-cut planning, interference-free tool-path generation, clearing cut, etc.

It is expected that the system will contribute to increase the productivity and the quality of the die-and-mold products.

여 백

CONTENTS

여 백

목 차

여 백

제 1 장 서론

제 1 절 연구의 배경

60년대 이후의 생산의 목표가 변천해 온 것을 보면, 물자가 부족했던 60년대에는 어떻게 하면 많이 만들 것인가가 목표였으며, 70년대에는 어떻게 싸게 만들 것인가가 목표였었다. 80년대 들어와서 생활이 윤택해지면서 좋은 물건을 찾게 되고 따라서 어떻게 좋은 물건을 만들 것인가가 생산의 목표가 되었다. 90년대에는 소비자의 욕구가 다양해지고 제품의 Life Cycle이 짧아지면서, 어떻게 하면 시장에 빨리 제품을 내어 놓느냐가 생산의 최대 목표가 되었다. 경영자문회사인 McKinsey 의 보고서에 의하면, 주어진 예산 안에서 6개월 늦게 출하된 제품은 5년간에 걸쳐서 33% 의 이윤 감소를 초래한 반면, 50% 의 예산을 초과하여 예정 시기에 출하된 제품은 5년간에 걸쳐서 4%의 이윤 감소를 초래한 예가 있다. 이를 보더라도 생산에 있어서 적기에 제품을 내어 놓는 일이 얼마나 중요한가를 알 수 있다.

제품 개발기간을 단축할 수 있는 기술로 동시공학 (Concurrent Engineering) 이 각광을 받게 된 것도 이 때문이다. 동시공학이란 개념설계단계에서 생산의 모든 단계를 고려하여 설계가 이루어 지고, 설계가 끝이 나면 모든 단계의 준비가 동시에 이루어 지는 기술로서, 생산 준비기간 및 비용을 획기적으로 단축 시킴으로써 제조업의 경쟁력을 증대 시킬 수 있는 기술이다. 동시공학을 구현하기 위한 요소 기술로는 Engineering Database (Product Model), 형상정보의 표준화 (IGES, STEP), Knowledge-based CAD (Intelligent CAD), CAE, Virtual Manufacturing, Reverse Engineering, Rapid Prototyping, Rapid Tooling 등이 있다. 본 연구는 금형 제작에 있어서 동시공학의 기법을

활용하여 금형 개발기간을 단축하기 위한 과제의 세부과제로 추진된 연구이다. 국내의 금형업체들은 대기업으로부터 금형제작을 의뢰 받을 때, 대개 2차원의 도면과 3차원 모형을 받는 경우가 많다. 2차원의 도면은 에러가 많고 3차원의 모형과 형상에 차이가 나는 수가 많다. 따라서 제품을 모델링하여, NC 데이터를 만들고 이를 가공하여 시사출 후 재가공과 시사출을 반복하는 경우가 많다. 본 연구에서는 3차원 모형을 Scanning하여 얻어진 Point 정보로부터 곡면을 형성하고, 이에 대한 유동해석을 실시한 후 금형을 설계하고 가공하는 일련의 연구 중 일부이다. 본 연구에서는 이 중에서 Reverse Engineering에 필요한 기술을 개발하기 위한 것이다. 즉, 측정한 데이터로부터 곡면을 형성하고, 이를 가공할 수 있는 데이터를 만들기 위한 것이다.

제 2절 연구의 내용

본 연구는 3차원금형제작 단축 자동화 시스템 개발의 세부과제로서, 협동과제로 개발되는 Digitizing Machine으로 측정된 데이터를 이용하여 자유곡면을 포함하는 금형의 Core 및 Cavity 면을 신속하게 Modeling 하고 가공할 수 있는 소프트웨어를 개발하기 위한 것이다. Modeling 기능으로는 측정데이터의 Filtering, Reduction 기능, 타 CAD 시스템과의 Interface 기능, 불규칙적인 측정 점의 보간 기능, Skinning, Sweeping, Blending, Trimming 기능 등이 주로 개발 되었다. 가공기능은 Parametric Cut, Cartesian Cut, 간섭방지, 공구경로 및 가공조건 최적화, 자동황삭 및 잔삭처리 기능이 주로 개발되었다.

일반적으로 CAD/CAM 관련 Software 는 상당히 방대하고 여러 사람이 나누어서 개발하게 되므로, Program Structure, Data Structure,

Documentation 등에 대한 체계적인 준비 후 추진하여야만 개발 및 Maintenance 에 문제가 없게 된다. 또한, 요즘 급격히 변하는 Hardware 및 Software 환경에 대한 충분한 검토를 하여 개발환경을 선택하는 일도 중요하다. 본 연구에서는 소프트웨어의 기능 및 구조를 설계하고, GUI, DB 등 기본기능을 완성한 후 곡면모델링 및 NC기능을 개발하였다. PC의 기능이 향상되면서, 많은 CAD/CAM 소프트웨어들이 PC에서 개발되고 있다. 본 연구에서는 이러한 추세에 맞추어 PC의 표준시스템인 WINDOWS-NT (WINDOWS 95), VisualC++, OpenGL 을 이용하여 개발 환경을 구축 하였다. 타 CAD와의 Interface를 고려하여 Non-manifold 데이터구조를 갖고 NURBS 를 기본표현식으로 사용하는 데이터 구조를 구축 하였으며, IGES, DXF, ZES 등의 Format을 일어들이거나 쓰는 기능을 개발하였다.

제 3 절 본 보고서의 내용

본 보고서는 Prototype 단계의 소프트웨어에 대한 Document로써, 사용자의 측면보다는 개발에 참여 하고 있는 사람이나, 앞으로 소프트웨어의 상품화 단계에서 참여할 사람들을 위한 참고문헌이 될 수 있도록 구성 하였다. 따라서, 연구과정에서 수집한 자료나, 소프트웨어 개발에 사용한 각종 Tool 들에 대한 자료를 모두 수록 하려고 노력 하였다.

서론에 이어서 제 2 장에서는 시스템 구조와 구현환경, GUI 구현에 대하여 설명 한다. 제 3 장에서는 시스템의 기본 기능을 설명 하는데, 수치계산용 객체의 정의, 객체 관리 기능, 기본 곡선·곡면 생성 기능, 형상 요소의 변환 기능, 형상 display 기능에 대하여 설명한다. 제 4 장은 데이터구조에 대한 장으로, non-manifold, object-orinted, NURBS를 고려하여 본 연구에서 사

용하는 데이터 구조에 대하여 설명한다. 제 5 장은 측정 데이터의 filtering, fairing을 설명한다. 제 6 장은 응용 곡선·곡면기능에 대한 설명이다. 제 7장은 타 시스템과 파일을 읽고 쓰는 기능에 대한 설명으로 IGES, ZES 파일포맷에 대한 것이다. 제 8 장은 NC 기능에 대한 설명이다. 제 9 장에서는 예제를 중심으로 개발된 기능을 소개하고, 마지막으로 제 10 장 결론부에서는 연구결과의 요약과 향후 연구계획에 대하여 설명한다.

제 2 장 시스템 설계 및 구현

제 1 절 개 요

CAD/CAM Software 개발의 첫 단계는 대상기계, 대상업종, 기술현황 등을 고려하여 프로그램의 기능을 설계하고 개발환경을 결정하는 것이다. 이를 위해서는 CAD/CAM Software 의 시장현황, 기술현황에 대한 정보가 필요하다. 프로그램의 기능 및 개발환경이 결정되면, 프로그램의 근간의 되는 기본기능을 개발하게 되고, 이 기본기능을 이용하여 응용기능을 개발하게 된다. 기본기능으로는 사용자와 프로그램 간의 interaction 을 위한 GUI (Graphic User Interface), 형상정보, NC 정보등을 외부 파일로부터 읽어들이고, 저장하고 수정하는 데이터베이스 관리 기능 및 intersection, interpolation, 등 형상 및 NC 정보 생성과 관련된 mathematical routines 의 library 가 있다. 응용기능은 점, 선분, 곡선, 곡면 등 형상을 생성하기 위한 기능, NC 기능, 측정데이터 관리 기능 등 필요에 따라, 기본기능을 이용하여 구축할 수 있다. 본 연구에서 개발한 CAD/CAM software 의 명칭을 SurfART 라 한다. 본 장의 제 2 절에서는 SurfART 의 화면 구조 및 메뉴 구조와 프로그램 구조에 대해 서술한다. 제 3 절에서는 SurfART 의 기본기능과 응용기능에 대해 설명한다. 제 4 절에서는 SurfART 의 구현환경에 대해 언급하고, 제 5 절에서는 GUI 구현으로서 화면생성과 메뉴생성에 대해 서술한다.

제 2 절 시스템의 구조

1. 화면 구조

SurfART 의 화면 구조는 graphic 영역, menu 영역, coordinate display 영역, message 영역, tool bar 영역, icon menu 영역으로 나누어진다. 그림 2.1 은 SurfART 의 화면 구조를 나타낸 것이다.

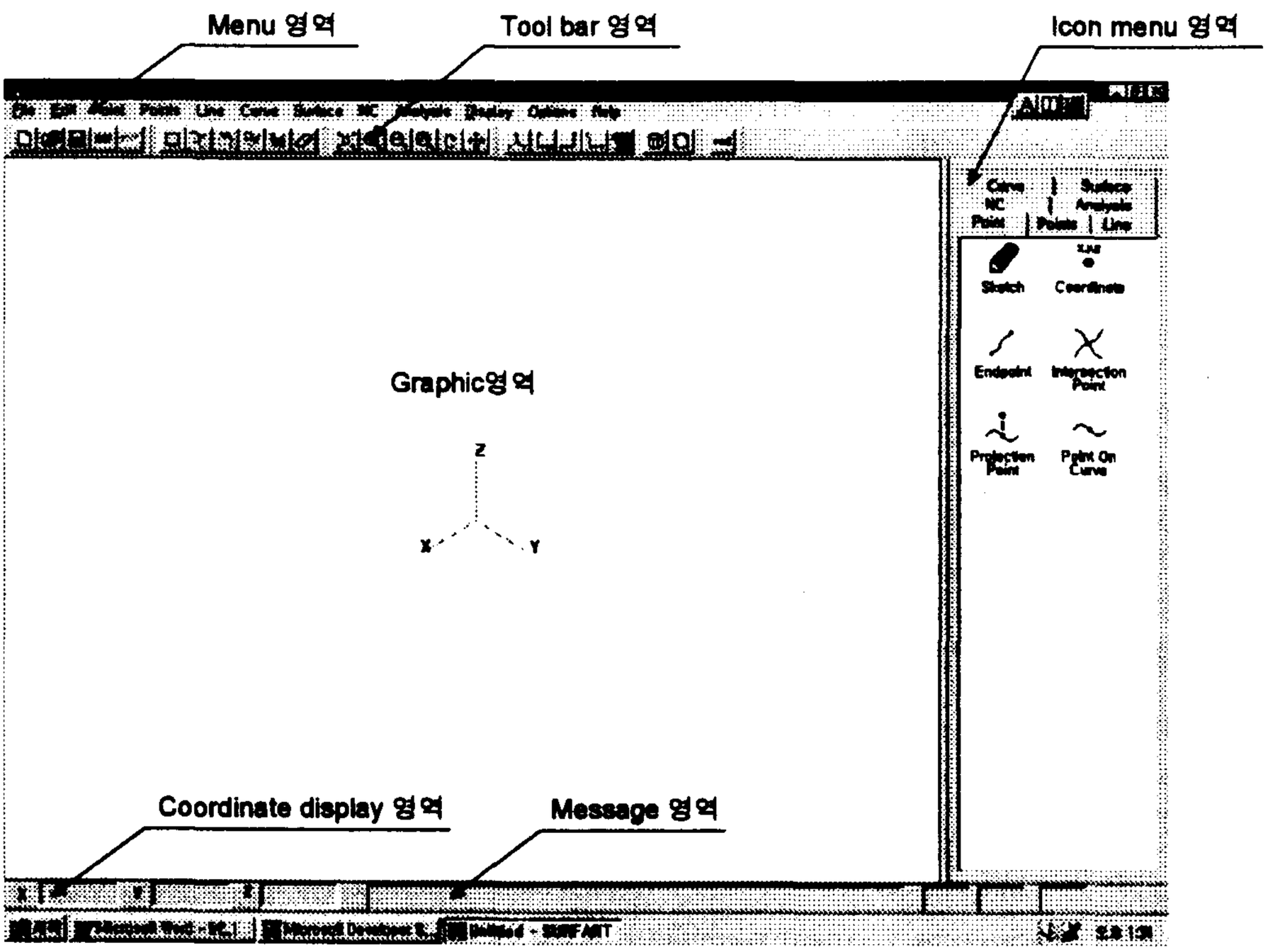


그림 2.1 SurfART 의 화면구조

① Graphic 영역

SurfART 에서 모델을 화면에 표시하여 준다.

② Menu 영역

Main menu 와 sub menu 로 표시한다. 작업자가 메뉴를 선택하면 그에 해당하는 작업방법 메시지가 message 영역에 표시되거나, 다이얼로그 박스가 띄워진다.

③ Status bar

ⓐ Coordinate display 영역

선택된 점(point)의 좌표값을 보여준다.

ⓑ Message 영역

메뉴를 선택하면 필요한 작업방법을 message 부분에 나타난다.

④ Tool bar 영역

메뉴에서 많이 사용되는 기능은 tool bar 로 표시하여 항상 쉽게 Access 할 수 있도록 한다.

⑤ Icon menu 영역

Main menu 중에서 많이 사용되는 menu 는 icon menu 영역에 표시하고, 메뉴의 기능을 아이콘으로 표시하여 사용자가 그 기능을 쉽게 이해할 수 있도록 한다.

2. 메뉴 구조

개발한 SurfART 의 메뉴구조는 주 메뉴(main menu)와 부 메뉴(sub menu)로 구분할 수 있다. 주 메뉴는 시스템의 기능별로 구성하였으며 부 메뉴는 사용자가 주 메뉴의 기능을 수행하는 방법에 따라 구분하였다. 메뉴는 크게 pull-down 메뉴와 icon 메뉴로 구현하였으며 사용자의 편의를 위해서 pull-down 메뉴는 화면의 상단에, icon 메뉴는 화면의 우측에 배치하였다. 또한, 시스템 수행시 사용 빈도수가 많은 file 기능, edit 기능, display 기능은 icon 메뉴화하여 tool bar 영역에 배치함으로써 편리하게 사용할 수 있도록 하였다. 그림 2.2 는 SurfART 의 메뉴구조는 나타낸 것으로 point, points, line, curve, surface, NC, analysis 등은 icon 메뉴영역에 배치하였다.

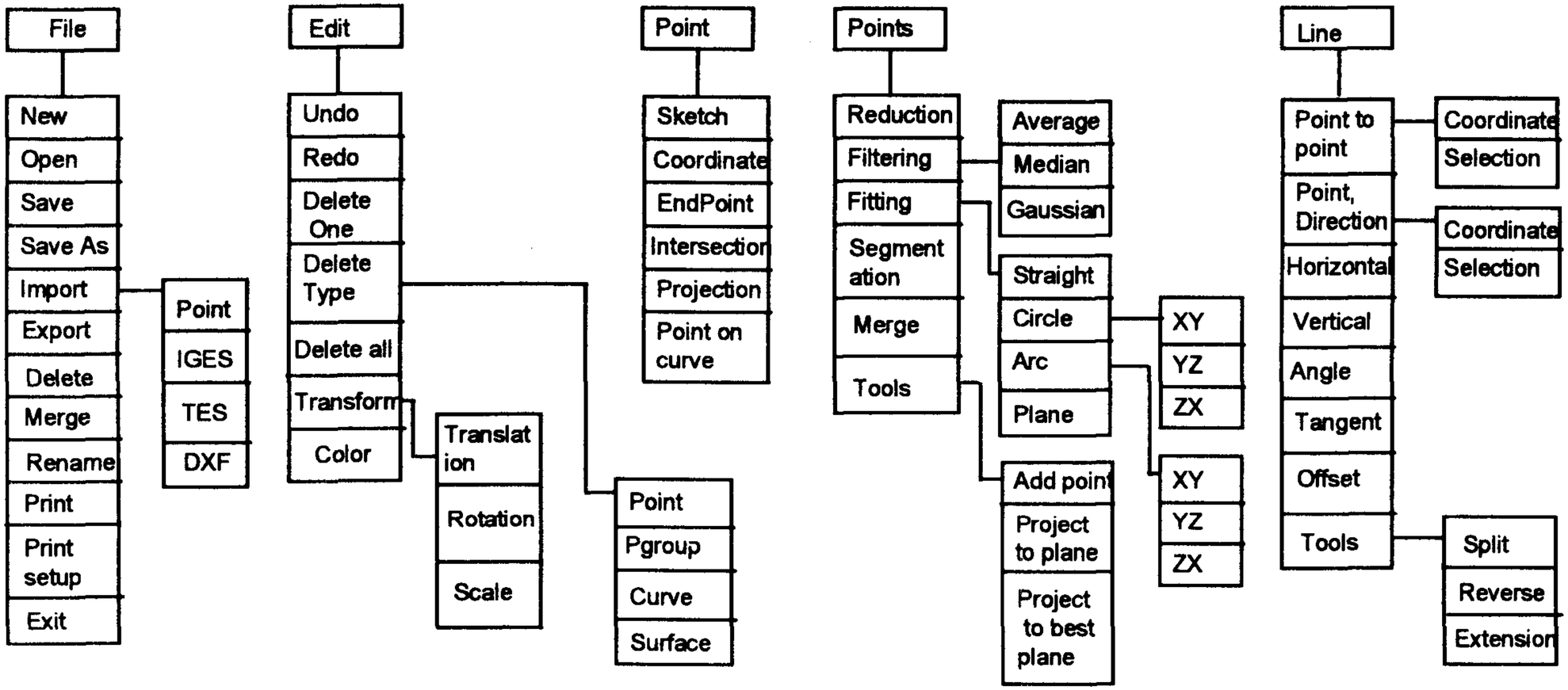


그림 2.2 SurfART 의 메뉴구조

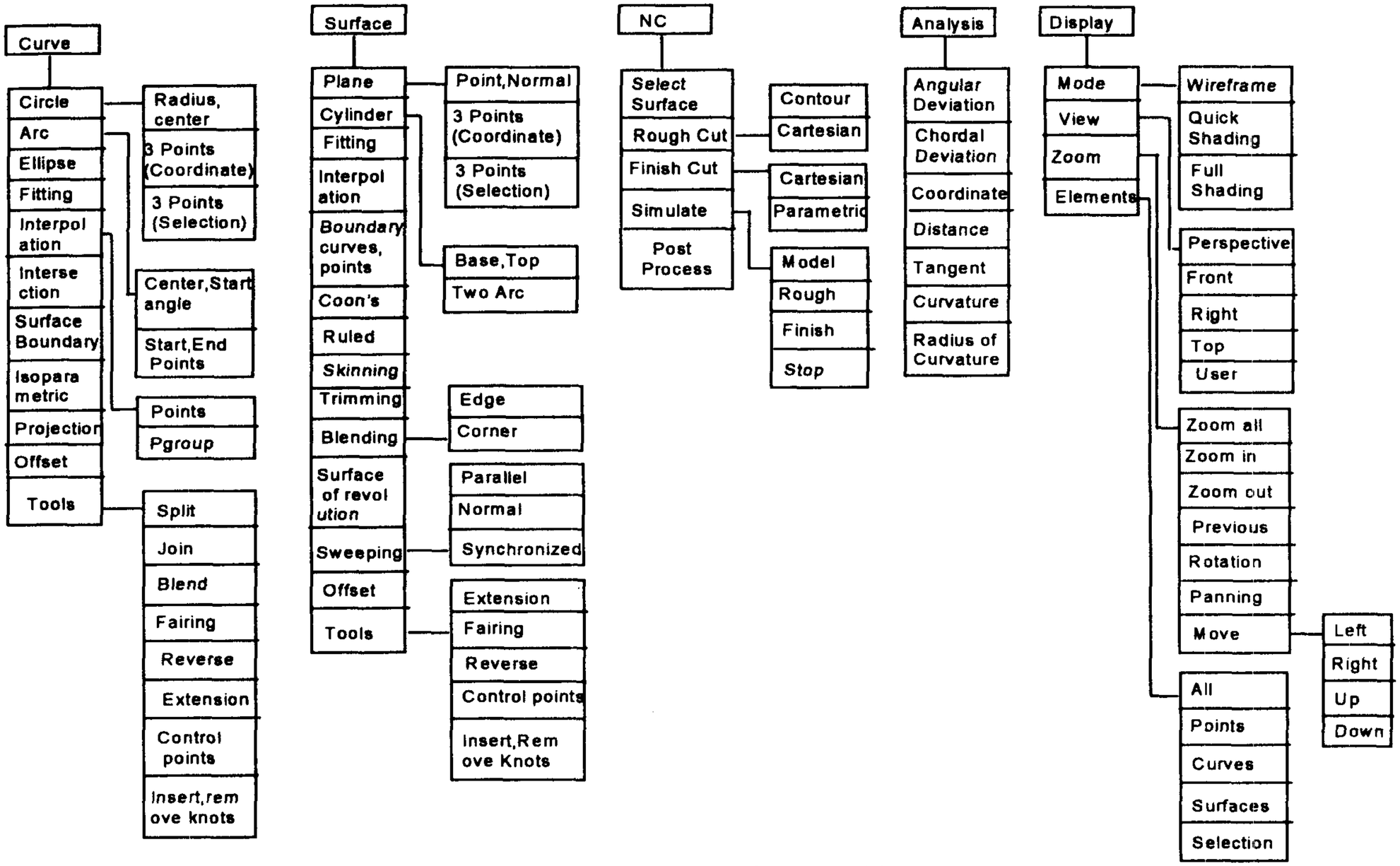


그림 2.2 SurfART 의 메뉴구조(계속)

3. 프로그램 구조

개발한 SurfART 의 프로그램 구조는 그림 2.3 과 같이 나타낼 수 있다. 개발된 소프트웨어는 다른 개발환경에서도 이용할 수 있도록 kernel module 을 독립적으로 구성하였다. 즉 GUI module 만 하드웨어 환경에 맞도록 수정하면, kernel module 을 수정하지 않고 사용할 수 있다. Kernel module 은 application 과 프로그램을 연결하는 API module, program data structure 와 그것을 관리하는 DB module, 도형간의 처리를 담당하는 geometric library, 수학적 계산을 담당하는 math library, 가공데이터를 생성하는 NC module, 외부 모델러와 데이터 교환을 위한 external modeler interface module 로 구성된다.

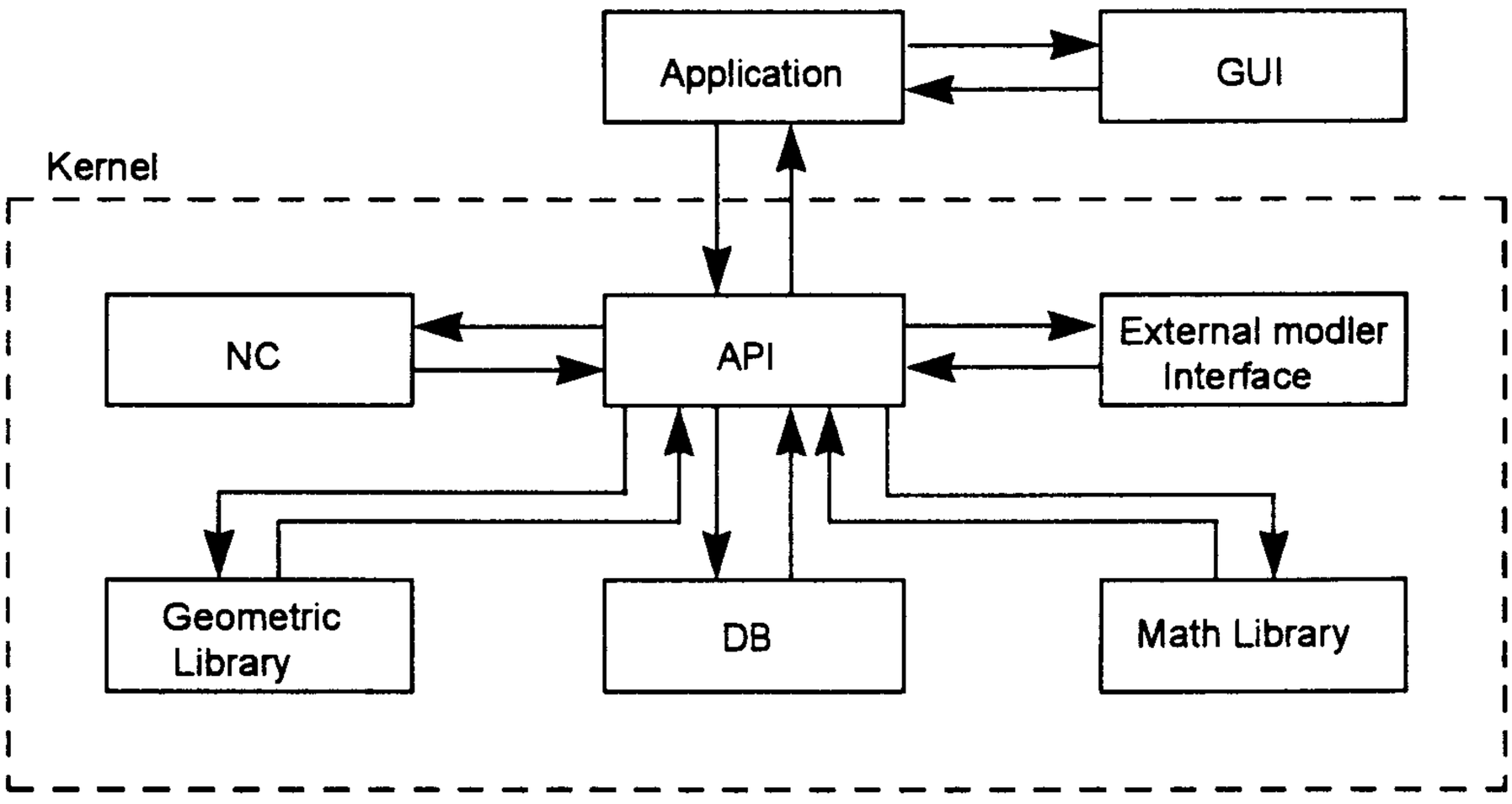


그림 2.3 SurfART 의 프로그램 구조

제 3 절 시스템의 기능

본 시스템은 측정기로부터 입력된 점 데이터를 이용하여 곡선, 곡면등을 생성하여 NC 가공 정보를 생성하는 곡면 모델링 시스템이므로 곡선을 생성하는 기능, 곡면을 생성하는 기능, NC 가공 정보를 생성하는 기능 등이 주요 기능이다. 또한, 보조적인 기능으로 형상 요소(geometric entity)를 변환(transformation)하는 기능, 삭제하는 기능, 파일로 저장 및 복원하는 기능, 사용자 인터페이스를 위하여 graphic 영역에 display하는 기능 등이 사용된다.

1. File 기능

File 기능에는 point 데이터나 타 CAD 시스템에서 만들어진 곡면 정보를 읽어 들이고, 만들어진 모델을 저장, 복원하는 기능등이 있다. 또한 생성된 모델을 타 CAD 시스템에서 읽어 들일 수 있도록 중간 파일(neutral file)을 생성하는 기능등이 있으며 다음과 같은 메뉴가 file 기능을 수행한다.

① New

새로운 모델을 생성할 수 있도록 파일의 환경을 초기화한다.

② Open

기존에 생성된 모델 파일을 읽는다.

③ Save

현재의 모델을 파일에 저장한다.

④ Save as

현재의 모델을 현재의 파일이름이 아닌 다른 이름으로 저장한다.

⑤ Import Point, IGES, TES, DXF file

Point, IGES(Initial Graphics Exchange Specification), TES(Topology ZES),

DXF(Drawing Interchange File)파일을 읽는다.

⑥ Export IGES file

현재의 모델에서 IGES 파일을 생성한다.

2. Edit 기능

Edit 기능에는 형상 요소를 삭제하는 기능, 변환하는 기능, 색깔을 변화시키는 기능등이 있으며 다음과 같은 메뉴가 edit 기능을 수행한다.

① Delete one

마우스로 선택된 형상 요소를 삭제한다.

② Delete type

점(Point), 점열(point group), 곡선(curve), 곡면(surface) 각각의 타입에 따라 삭제한다.

③ Delete all

모든 형상 요소를 삭제한다.

④ Transform(translation, rotation, scale)

형상 요소를 이동하고, 회전하고, 크기변화를 준다.

3. Point 기능

Point 기능으로는 좌표값을 입력하여 점을 생성하는 기능, 곡선상의 양 끝단에 점을 생성하는 기능, 곡선상의 점을 생성하는 기능등이 있으며 다음과 같은 메뉴가 point 기능을 수행한다.

① Coordinate

다이얼로그 박스에서 x, y, z 좌표값을 입력하여 점을 생성한다.

② EndPoint

곡선상의 양 끝단에 점을 생성한다.

③ Point on curve

곡선상에서 파라미터를 입력하여 점을 생성한다.

4. Points 기능

측정된 점 데이터의 양이 대단히 많으므로 이것을 처리하는 다양한 기능이 요구된다. 우선 Data reduction 기능이 필요하다. 측정되는 정보는 곡면을 만드는데 필요한 갯수보다 훨씬 많을 수가 있다. 특히 단층촬영 등을 이용하여 얻어진 데이터는 그 갯수가 필요한 양보다 훨씬 많다. 따라서, 데이터의 특성을 유지시키면서, Point의 갯수를 줄이는 기능이 필요하다. data reduction은 일정한 간격으로, 예를 들어서 매 5 Point 간격으로, 추출될 수 있다. 또는, 데이터 특성(chordal deviation)이 크게 변하는 점들을 기준으로 데이터를 추출할 수 있다. 측정데이터 중 오차가 특히 큰 데이터를 보정해 주거나 제거 시켜주는 Filtering 기능등이 있으며 다음과 같은 메뉴가 points 기능을 수행한다.

① Reduction

마우스로 점열을 선택하여 점의 수를 최소화하여 형상을 표현한다.

② Average, Median, Gaussian filtering

마우스로 점열을 선택하여 점의 통계적 평균값, median value, gaussian theory을 이용하여 에러를 줄인다.

③ Straight, Circle, Arc, Plane fitting

마우스로 점을 선택하여, 선택된 점들을 이용하여 직선, 원, 원호, 평면으로 구현한다.

5. Line 기능

Line 기능에는 두 점을 입력으로 하여 직선을 생성하는 기능, 시작점과 방향벡터를 입력하여 직선을 생성하는 기능, offset을 이용하여 직선을 생성하는 기능등이 있으며 다음과 같은 메뉴가 line기능을 수행한다.

① **Point to point**

두 개의 점을 입력 또는 선택하여 직선을 생성한다.

② **Point, direction**

시작점과 방향벡터를 입력 또는 선택하여 직선을 생성한다.

③ **Offset**

Offset할 직선을 선택하고, offset vector를 입력하여 직선을 생성한다.

6. Curve 기능

곡면모델러에서 최종적으로 얻는 형상 요소는 곡면이나, 점으로부터 바로 곡면을 얻을 수 있는 경우는 입력데이터가 잘 정열되어 있는 단순한 경우이며, 대부분의 경우는 먼저 곡선을 구한 후 이를 이용하여 곡면을 얻게 된다. 따라서 다양한 curve 기능이 요구된다. 측정한 점을 이용하여 곡선을 interpolation하는 기능, 원, 원호등 analytical한 곡선을 대화식으로 생성하는 기능등이 있으며 다음과 같은 메뉴가 curve 기능을 수행한다.

① **Circle**

Reference plane, 중심, 반지름을 입력하는 방법과 세 점을 입력 또는 선택하여 원을 생성한다.

② **Arc**

Reference plane, 중심, 시작점, 반지름, 각도를 입력하는 방법과 reference

plane, 중심, 시작점, 끝점, 반지름을 입력하여 원호를 만든다.

③ Ellipse

Reference plane, 중심, major radius, minor radius를 입력하여 타원을 만든다.

④ Interpolation

점 또는 점열을 선택하여 NUBS(Non-Uniform B-Splines) 곡선으로 보간한다.

⑤ Intersection

두 개의 곡면을 선택하여 교차하는 곡선을 생성한다.

⑥ Projection

투영할 곡선과 투영될 곡면을 선택하고 투영방향벡터를 입력하여 투영된 곡선을 생성한다.

⑦ Split

동일 곡면상에서 분할될 곡선과 knife 곡선을 마우스로 선택하여 곡선을 분할한다.

7. Surface 기능

곡면을 만드는 기능은 측정데이터를 그대로 interpolation하는 기능과, 측정데이터를 이용하여 생성된 곡선들로부터 곡면을 만드는 기능등이 있다. 곡선을 이용하여 곡면을 만드는 주요기능으로는 두개의 마주보는 곡선을 linear하게 연결 시켜주는 ruled 곡면, 단면 곡선들을 주어진 곡선을 따라 가면서 변화시켜서 곡면을 만드는 Sweep 곡면 등 다양하다. 대부분의 복합곡면의 경우 곡면과 곡면 사이에 일정한 반경으로 rounding 되거나 filleting 되게 되는데, 이를 위한 blending 곡면 기능은 곡면모델러에서 가장 중요한 기능 중 하나이다. blending 되거나 필요없는 부분은 trimming하는 기능등이 있으며 다음과 같은

메뉴가 surface 기능을 수행한다.

① Plane

중심점, normal vector, u,v direction size를 입력하는 방법과 세 점을 입력 또는 선택하는 방법으로 평면을 생성한다.

② Cylinder

Base plane, top plane, 중심점, 반지름을 다이얼로그 박스에서 입력하는 방법과 base, top의 경계로 사용될 곡선을 선택하는 방법으로 실린더를 생성한다.

③ Interpolation

점열을 선택하여 NUBS(Non-Uniform B-Splines) 곡면으로 보간한다.

④ Ruled

두 개의 곡선을 선택하여 두 곡선을 linear하게 연결하는 ruled 곡면을 생성한다.

⑤ Skinning

두 개 이상의 곡선을 선택하여 skinning 곡면을 생성한다.

⑥ Trimming

곡면상의 곡선을 선택하여 edge closure를 수행하고, 선택하는 방향에 따라 트림 곡면(trimmed surface)을 생성한다.

⑦ Blending

Blending할 곡면을 선택하고, extension option을 선택하지 않는 경우는 blending radius만 입력하고, extension option이 선택한 경우는 trim 또는 split option을 선택하고 blending radius를 입력하여 blending 곡면을 생성한다.

⑧ **Sweeping**

2차원 단면곡선이 어떤 규칙을 갖고 안내곡선 또는 경계곡선을 따라 이동하면서 곡면을 생성한다.

8. NC 기능

NC 기능은 가공해야 할 곡면을 선택하고, 경계를 넘으면 안되는 곡면이나 곡선을 지정해주는 일이 NC programming의 첫 단계이다. 공구경로는 isoparametric path 또는 cartesian path가 일반적이며, 복합곡면인 경우는 대개 cartesian path가 사용된다. 가공할 곡면이 선택되고 공구경로 생성방법이 결정되면, 공구 및 절삭조건에 대한 정보를 대화식으로 결정해 주게된다. 그런 후 자동으로 공구경로가 생성 되는데, 공구경로를 display 하여 과절삭, 잔삭 등을 확인해 볼 수 있는 기능등이 있으며 다음과 같은 메뉴가 NC 기능을 수행한다.

① **Select surface**

NC가공할 곡면을 마우스로 선택한다.

② **Cartesian cut**

Cartesian domain상에서 공구 경로 정보를 다이얼로그 박스에서 입력하여 공구경로를 얻는다.

③ **Parametric cut**

Parametric domain상에서 공구 경로 정보를 다이얼로그 박스에서 입력하여 공구경로를 얻는다.

④ **Simulate**

CL data에 따라 공구를 이동시키면서 공구의 간섭을 파악할 수 있다.

9. Display 기능

Display 기능은 일반적인 CAD/CAM 시스템이 갖는 기능들 외에 측정한 데이터로부터 최종곡면의 형상을 빠르게 Display 해 볼 수 있는 기능이 필요하다. 측정데이터를 Point로 Display한다. 일반적으로 CAD/CAM Software가 갖는 rotation, translation, zooming 및 shading 기능도 필수적이다. 본 연구에서는 PC에서 Software를 개발하므로 Graphic 처리속도가 늦다는 특성을 감안하여 Shading 처리시 옵션을 선택할 수 있게 했으며, 이동시 Shading 된 Image가 Wireframe으로 바뀌어 이동할 수 기능등이 있으며 다음과 같은 메뉴가 display 기능을 수행한다.

① Mode

Wireframe mode과 shading mode로 나누어 지고, shading mode는 quick shading과 full shading로 구분한다.

② View

View는 isometric, front, right, top, user view로 나누어진다.

③ Zoom

ⓐ Zoom all: 물체의 위치가 한쪽 방향으로 기울어지거나 전체 형상이 완전히 보이지 않는 경우에 물체를 스크린 크기에 맞추어 준다.

ⓑ Zoom in, out: 물체를 확대 또는 축소하기 위해 마우스로 선택한 영역을 보여준다.

ⓒ Previous: Zoom in, zoom out, rotation, panning, move를 수행한 경우에는 이전 단계로 돌아갈 수 있다.

ⓓ Rotation: 마우스로 이용하여 눈의 보는 방향을 조절한다.

ⓔ **Panning**: 마우스를 드래깅하여 화면을 이동시킨다.

ⓕ **Move**: 상하좌우로 일정한 간격만큼 이동한다.

④ **Elements**

모든 형상 요소 또는 점, 곡선, 곡면 타입 또는 선택한 점 타입을 화면상에 디스플레이한다.

제 4 절 시스템 구현 환경

1. 운영체제

DOS는 일반 사용자들에게 가장 널리 알려진 운영체제로 10년간 운영체제 시장을 제패하고 있다. Windows95는 현재 일반 사용자에게 가장 널리 알려진 윈도우 형태의 운영체제이다. 기본으로 플러그앤플레이, PCMCIA지원, OOP적 인터페이스 구성, 네트워크 기능등이 지원되고 있다. 팬 윈도우는 컴팩, HP와 공동으로 개발한 것으로 PDA에 사용된다. 모듈러 윈도우는 TV, VTR등과 같은 가전기기에 들어가는 운영체제 환경이고 Windows NT는 광범위 기업형 네트워크와 클라이언트 서버 네트워크 시장을 공략하여 유닉스와 대응하기 위한 워크스테이션급 32비트 운영체제이다.

데이토나는 Windows NT 3.5로서 예전의 NT보다 확장된 기능을 가졌다. 노벨 네트웨어 서버와 바로 연결할 수 있고, 인터네트를 지원하는 TCP/IP 드라이버가 크게 향상되어 기존 유닉스 시스템에 있는 프린터를 그대로 사용하고 32비트 OLE2.1지원, OpenGL 3D그래픽스 라이브러리를 제공한다. 도스나 윈도우즈에 비해 고가이다. 카이로는 NT 4.0으로 NT에 시카고의 새로운 기능들이 추가된 것으로 분산형 운영체제를 지향하고 있다. OLE 2.0을 기반으로 분산형 파일 시스템, 디렉토리 서비스등을 결합한 객체 지향적인 구조를 가지될 것이다.

OS/2 2.x는 NT를 능가하는 DOS, 윈도우즈 프로그램 에뮬레이션, 멀티테스킹, HPFS(Hogh Performance File System), 개체 지향적 워크플레이스 쉘, REXX 배치언어, 거의 모든 네트워크 환경을 지원하는 32비트 운영체제로서, 8MB와 80MB의 하드웨어 요구한다. 그러나 일반 사용자용 응용프로그램이

다소 부족하고, 다른 시스템으로의 이식성이 부족하다. 넥스트 스텝은 GUI 유닉스의 선두 주자로서 Display PostScript, 객체지향 API를 지원하고 강력한 네트워크, 메킨토시와 같은 GUI를 지원한다. 설치에 필요한 시스템 요구사항이 까다롭다.

솔라리스 for x86은 BSD계열 유닉스, SCO유닉스, 제닉스의 장점을 통합한 SVR4를 기반으로한 워크스테이션용 솔라리스의 PC급 32비트 버젼으로 서버와 디스크리스(Diskless)클라이언트, 데이터리스(Dataless)클라이언트로서 네트워크상의 클라이언트/서버, NFS를 기반으로한 클라이언트/서버, 오라클, 인포믹스, 사이베이스 등 수많은 데이타베이스 등이 운용되는 클라이언트/서버를 지원한다. TCP/IP, X.25, SNA, OSI등의 프로토콜 지원하고, PC, IBM, DEC간의 접속솔루션 제공한다. 기업, 은행, 보험회사, 정부기관 등에서 PC를 네트워크로 연결해 편리한 환경을 만들때 용이하다.

유닉스웨어는 32비트 멀티유저 운영체제이다. TCP/IP와, IPX/SPX를 기본 장착으로 유닉스와 네트웨어를 통합한 환경을 제공한다. WAN의 연결을 위해 PPP(Point to Point Protocol)및 SLIP(Serial Line Internet Protocol), 네트워크 관리기능을 지원하고 X윈도우 버젼 II릴리즈 5를 기본으로 제공하며 모티프, 오픈룩을 사용해서 윈도우와 같은 GUI환경을 지원한다. 기존 네트웨어 환경에서 32비트 운영체제의 성능을 필요로 하는 기업체에 적합하다.

가. 32비트 운영체제의 특성

- 미국 시장의 40% 점유하고 있다.

- 한글 OS/2 2.1(IBM), 한글 넥스트스텝(넥스트사), 한글 솔라리스 for x86(SUN)등의 한글화된 운영체제가 있다.

- 유닉스웨어(노벨), 오픈데스크탑(SCO), 윈도우즈 NT(MS사), 리눅스 32비트가 한글화 중이다.

- 완전한 32비트 보호모드 오퍼레이션지원 : 하드웨어 보호 스킴을 이용하여, 중요한 시스템 요소에 사용자 레벨의 소프트웨어가 끼어들지 못하도록 일련의 보호규칙을 적용한 것으로 만일 어플리케이션이 에러를 일으켜도 운영체제와 그외에 동작중인 어플리케이션에 영향을 미치지 않게 한다.

- 무한에 가까운 메모리 : 플랫 메모리 모델로 하나로 통합된 4GB의 메모리 어드레싱 능력을 갖는다.

- 멀티 테스킹 : 선점형 멀티테스킹으로 윈도우즈3.1의 협조형 멀티테스킹과는 달리 각 프로세서간의 CPU사용 시간을 분할 관리하는 스케줄을 제공한다.

- 멀티 쓰레딩 : 어플리케이션이 쓰레드라고 하는 프로그램 유닛으로 등록되어 쓰레드 레벨에서의 멀티테스킹을 지원한다. 예를 들면 스프레드쉬트에서 자료를 계산하고 출력하면서 동시에 자료를 검색할 수 있다.

- 대칭형 멀티프로세싱 : 한대의 컴퓨터에서 하나 이상의 CPU를 사용할 수 있게하는 기능으로 시스템 성능을 향상시키기 위해서는 CPU를 하나 더 달면 된다.

- CPU 이식성 : 대부분의 32비트 운영체제들이 서로 다른 플랫폼에서 사용할 수 있다. 즉, 다른 아키텍쳐의 프로세서를 모두 지원한다.

- 도스/윈도우즈 어플리케이션 에뮬레이션 기능 제공한다. (OS/2, NT)

- LAN에 적합한 네트워크 기능을 갖추고 있다.

- 객체지향 개발환경을 지원한다.

32비트 운영체제의 가장 큰 이익은 메모리와 처리 속도이다. 32비트

플랫 메모리는 화일스와핑이나 세그먼트 옵셋에 대한 계산이 필요없어 속도가 개선된다. PC에서 메모리를 많이 차지하는 이미지 편집이나 캐드, 멀티미디어 어플리케이션 분야에 적합하다.

나. 마이크로소프트의 WIN32 개념

마이크로소프트는 두가지 방향에서 윈도우즈 패밀리를 발표해 나가고 있다. Windows 3.1과 Windows 95(Chicago)로 이어지는 대중형 윈도우 패밀리와 NT, NT 3.5(데이토나), NT 4.0(카이로)로 이어지는 워크스테이션급의 산업형 윈도우 패밀리가 있다. 인터페이스면에서는 Windows 3.1과 NT, NT 3.5가 유사한 구조를 가지고 Chicago와 Cairo가 유사한 구조를 가진다. 이런 각기 다른 운영체제에서 프로그래머를 위해 운영체제와 비의존적인 32비트 프로그램을 할 수 있도록 하는 것이 WIN32 API이다. WIN32 API는 WIN32 API로 생성된 코드는 NT든 Chicago든 Windows 3.1이든 코드의 수정없이 실행가능하게 하는 해결책을 제시함으로써 앞으로 발표될 윈도우즈 패밀리에서도 호환성을 가질 수 있도록 해준다. 각 운영체제에 대한 WIN32가 모두 동일한 것은 아니다. WIN32가 원래 NT의 32비트 프로그램을 위해 개발되었으므로 Windows 3.1이나 Chicago에서는 WIN32s(Subset)와 WIN32c(Chicago)가 사용되고 각각의 운영체제의 능력에 따라 기능이 제한된다.

WIN32s < WIN32c < WIN32

다. WIN32의 기능들

32비트 메모리 관리는 더이상 세그먼트와 옵셋에 관련한 메모리 관리가 필요 없다. near, far pointer, GlobalXX, LocalXX등은 쓰지 않아도 된다.

WIN32는 32비트라는 한가지 모델로 동작하며 메모리 할당도 VirtalAlloc, VirtualFree, VirtualLock만 사용한다. 각 프로세스마다 4GB 정도의 메모리가 할당된다.

메모리 맵드 화일은 WIN16에서는 파일을 다룰 때 파일 열기, 읽을 위치 찾기, 화일 닫기 등의 조작이 필요하다. WIN32에서는 디스크에 위치하는 화일을 시스템 메모리 공간과 연결하여 매핑시켜주어 화일 엑세스를 메모리 에세스처럼 할 수 있게 된다.

① PE(Portablr Executable) 화일포맷을 갖는 EXE와 DLL의 실행

② 프로그램에서 데타화일을 다룰 때 메모리에 있는 것처럼 엑세스

③ 다중 프로세스 간의 동일 데이타를 공유할 때

유니코드의 경우는 미국의 유수 소프트웨어 업체들이 모여 컨소시엄을 구성했고 이 그룹에 의해 유니코드 표준안이 마련되었다. 서로 다른 언어간의 데이타 교환이 쉬워지고 하나의 EXE나 DLL로 모든 언어를 지원할 수 있다. 외국의 소프트웨어가 한글화되지 않고도 국내환경에서 쉽게 사용된다.

향상된 GDI함수로 Bezier곡선이나 Path등을 2D용 그래픽 함수 제공한다. 유용한 그래픽 라이브러리인 OpenGL 라이브러리 제공한다.

만일 윈도우즈의 현재와 미래에 모든 버젼에서 실행되는 코드를 원한다면 WIN32s에서 프로그래밍해야 하고 NT의 향상된 기능만 원한다면 WIN32만을 사용해야 한다.

2. 프로그래밍 언어

가. Visual Basic과 Visual C++, SDK의 비교

(1) MS Visual Basic

Visual Basic은 MS Windows에서 가장 빠르고 쉽게 어플리케이션을 생성하는 프로그래밍 툴이다. 만약 프로그래머가 MS SDK를 사용하여 Window용 응용프로그램을 작성해 본 일이 있다면 그 작업이 얼마나 많은 노력을 필요로 하는지를 알 것이다. 특히 전문적인 시스템 프로그래머가 아닌 일반 업무용 응용프로그램 작성자들에게는 Visual Basic이 획기적인 대안이 된다. 흔히 업무용 응용프로그램들은 MS Windows 시스템에 대하여 아주 미세한 부분까지 제어할 필요는 없다. 업무용 응용프로그램들은 대부분 메뉴구조와 입력필드, 출력필드가 필요하고 용도에 따라 데이타 관리를 위한 데이타베이스와 그래픽 라이브러리, 마우스 처리 루틴등이 공통적으로 필요하다. 물론 MS Visual Basic은 이러한 기능을 모두 지원하는 MS Windows용 GUI툴이다. MS Visual Basic의 모든 오브젝트는 컨트롤을 이용하여 생성되는데 전문가판에서 제공하는 컨트롤은 44개 정도로 특수용도가 아닌 일반 응용프로그램에서는 모자람을 느끼지 않는다.

Visual Basic에서 Graphic을 구현하기 위해서는 MS SDK에서 제공하는 Device-Context Function을 이용해야 한다. Device Context 함수는 Windows 시스템과 응용프로그램을 연결해주는 장치들로 Color-Palette Functon, Drawing-Attribute Functon, Mapping Functon, Coordinate Functon, Region Functon, Clipping Functon, Line-Output Functon, Ellipse Functon, Polgon Functon, Bitmap Functon, Text Functon, Font Functon등이 지원된다. Visual Basic에서 위의 유용한 함수들을 쓰기 위해서는 Visual Basic 코드에 C 코드를 접속시키는 약간의 방법이 필요하며 windows의 프로그램과 구조에 대한 전문적인 지식

이 필요하다.

다른 방법으로 Visual Basic에서 쓰이는 언어인 베이직에서 제공하는 그래픽 루틴을 쓰는 방법이 있다. 기본좌표시스템은 트윕(TWIP)으로 1/1440 inch이다. 그러나 사용자의 편의에 따라 화면의 기본 단위인 픽셀, 센티미터, 포인터 등으로 바꿔 쓸 수 있고 좌표축 또한 바꿀 수 있다. 기본기능은 RGB colored Point, Line, Rectangle, Circle, Ellipse, Arc가 있다.

(2) MS Visual C++ with MFC Library

MS Visual Basic등의 대화식 개발툴을 제외한 경우는 MS Visual C++에 포함되어 있는 Microsoft Foudation Class Library 응용 프레임워크를 사용하는 것이다. MFC는 MS Windows API로서 클래스 라이브러리 응용프레임워크에는 자신의 응용구조가 있어 윈도우 프로그램을 안전하고 자유롭게 코드를 작성할 수 있다. 클래스 구조를 사용해도 SDK로 할 수 있는 모든 프로그램을 할 수 있다. 클래스 라이브러리로 작성한 프로그램은 거의 SDK 프로그램만큼 작다. Visual C++의 App Stdio, AppWizard, ClassWizard등은 응용프로그램마다의 프로그램 구조를 새로 개발하는 부담을 덜어준다. 실제로 클래스 라이브러리의 사용법을 모두 배우려면 최소한 6개월 정도가 걸리며 이 기간은 SDK를 배우는 시간과 거의 비슷하다. 그러나 클래스 라이브러리는 SDK와 달리 세세한 프로그래밍을 피할 수 있어 프로그래밍을 시작하기 쉽고 또, 일단 익숙해지고 나면 생산성이 매우 향상된다.

(3) MFC에서의 Graphic

MFC에서 제공되는 CDC 클래스로서 SDK의 Device Context를 위한 base class이며 디스플레이에 대한 모든 기능이 들어있다[2-1]. MFC 2.0은 SDK의 Device Context 함수와 기능이 동일하고 MFC 3.0은 많은 Drawing Routine이 추

가되어 3D 그래픽에 필요한 기본 루틴을 작성하는데 용이하다. MFC 3.0은 Visual C++ 2.0(for NT)에서 사용가능하다.

(4) MS SDK

MS SDK는 MS Windows의 기본 API의 Windows 시스템 프로그램을 위한 수많은 기본 API로 구성되어 있다. Windows 시스템에 대한 전문적인 지식을 필요로 한다.

나. 윈도우즈 프로그래밍 모델

(1) Message Driven

윈도우즈 프로그램과 MS-DOS 프로그램의 기본적인 차이점은 MS-DOS는 user input을 받기 위해 운영체제를 호출하지만 윈도우즈 프로그램은 운영체제로부터의 메시지를 통해서 user input을 처리한다.

(2) GDI - Graphic Device Interface

모든 디스플레이 카드와 프린터 모델에 똑같이 적용되는 편리한 인터페이스이다.

(3) Resouce Binding

자원으로는 bitmap, icon, menu, dialog Box, string등이 있고 이들을 편집하기 위한 interactive tool이 있다.

(4) Memory Management - 16bit(Windows 3.5), 32bit (Windows NT)

Extened/Expaned 메모리 사용에 특별한 제한이나 기술이 필요없다.

(5) DLL (Dynamic Linking Library)

실행되는 동안 동적으로 프로그램에 링크되므로 메모리를 절약할 수 있다.

(6) OLE(Object Linking and Embedding) anf TrueType Font

폰트의 크기를 임의로 조절할 수 있고 임의의 프린터에서 작업할 수 있게 해준다.

다. MFC 소개

MFC 라이브러리 2.0 (Microsoft Foundation Classes Library 2.0)은 100개 이상의 클래스와 60000개의 함수로 이루어져 있고 Visual C++에서 윈도우즈 API(SDK)를 완전히 엑세스할 수 있다.

(1) CObject 클래스

MFC의 대부분의 클래스는 CObject클래스에서 파생된다[2-2]. 모든 MFC 클래스 이름과 MFC 클래스로부터 상속된 클래스의 이름은 'C'로 시작한다. CObject클래스를 쓰는 잇점으로 연속화로 오브젝트의 쓰기와 읽기를 처리할 수 있다는 것과 오브젝트의 동적 생성이 가능하고 실시간 클래스 정보를 사용할 수 있다. 또한 타당성 검사로 특정 오브젝트의 포인터가 타당한지 검사한다. 그 외에도 TRACE ( ), Dump ( ), 예외처리 루틴이 제공된다.

(2) 다른 MFC 클래스들

CArchive 클래스는 디스크 파일에서 데이타를 읽거나 쓰는 것에대한 지원을 한다. CFile클래스는 화일 읽기모드와 쓰기모드를 관리하고 운영체제 화일에 대한 C++ 프로그래밍 인터페이스를 지원한다. CMemFile클래스는 메모리에 저장된 데이타를 화일에 저장된 데이타처럼 사용하도록 지원한다. 배열 클래스로는 BYTE, WORD, DWORD, Object, Object Point, CString 의 배열을 생성할 수 있다. 리스트 클래스로 Doubly Linked List를지원하고 맵(Maps)을 이용하여 Object를 관리하는 사전을 제공한다. CString 클래스는 문자열을 관리하는 가장 강력한 클래스이다. 시간에 관련된 클래스는 CTime과 CTimeSpan이 있다.

라. Visual Workbench 도구들

(1) 프로젝트

일반적인 Workbench에서 사용하는 프로젝트처럼 관련된 모든 소스화일, 리소스화일, Compile, Link Environment에 대한 정보를 가지고 있다.

(2) C, C++ Compiler

소스코드 파일명의 확장자를 살펴서 C와 C++ 모두를 컴파일할 수 있다.

(3) Linker

.OBJ 화일을 여러 라이브러리와 링크하여 EXE를 생성한다.

(4) Resource Compiler

Resource화일 (.RC)를 Binary화일(.RES)로 컴파일하고 Binary화일을 EXE화일에 합병한다.

(5) Debugger

브레이크 포인트에 의한 디버깅, CodeView 디버거가 지원된다.

(6) AppWizard

AppWizard 대화상자를 통해 지정한 특성들, 클래스명, 소스코드화일명을 가진 윈도우즈 응용프로그램의 골격을 만들어 주는 코드 생성기이다. Blue Sky WindowsMAKER와 같은 기본코드 생성기와는 다르다. AppWizard 코드는 새로운 프로그램을 빨리 시작하도록 해주는 응용프레임워크 기본 클래스를 제공한다.

(7) ClassWizard

Visual C++ 클래스와 메시지 코드를 쉽게 유지하게 해준다.

(8) Source Browser

화일의 관점대신 클래스나 함수의 관점에서 응용프로그램을 테스트하고 편집하며 검색할 수 있도록 해준다.

(9) 온라인 도움말

MS Windows SDK Reference Manual, MFC Library Reference Manual의 전체 내용은 Visual C++ 온라인 도움말에 모두 포함되어 있다.

3. 그래픽 라이브러리

OpenGL은 산업표준으로 가장 널리 쓰이는 Graphic Library이다. OpenGL을 쓰는 경우는 Picking, Transformation, Shading등을 손쉽고 안정되게 구현할 수 있다[2-3]. 특히, NURBS 곡면등 다양한 Display 기능을 제공하여 준다.

가. OpenGL구성

(1) OpenGL 명령어 - 115개 함수

* Object Shape Description
* Matrix Transformation
* Lighting
* Coloring
* Texture
* Clipping
* Bitmaps
* Fog
* Anti-alising

(2) OpenGL Utility (GLU) - 43개 함수

* Texture Support
* Coordinate Transformation
* Polygon Tesselation
* Rendering Spheres

* Cylinders & Disks

* NURBS(Non-Uniform Rational B-Spline) Curves & Surface

* Error Handling

(3) OpenGL Programming Guide Auxiliary Library - 31개 함수

(4) WGL APIs - 6개 함수

* OpenGL과 Windows NT windowing system을 연결해 준다

* Rendering Context

* Font Bitmaps

* X-Window windowing system의 GLX와 비슷함

(5) new Win32 APIs - 5개 함수

* Pixel Format

* Double Buffering

나. 구현상의 특성

Windows NT 3.5가 지원하는 모든 하드웨어에서 사용 가능하다. VGA, Super VGA 16 color mode에서도 사용 가능하다. 그러므로 모든 CPU 플랫폼에서 사용가능하다.

다. OpenGL을 Windows NT에서 사용하기 위해 필요한 개념

(1) Rendering Context

모든 OpenGL command는 OpenGL Rendering Context를 통해서 수행된다. 즉 모든 OpenGL call은 current Rendering Context를 통해서 가능하다는 것이다. 그러므로 Rendering Context는 OpenGL과 Windows NT windowing system을 연결해 준다.

모든 Rendering Context는 Windows Device Context를 가지는데 Rendering Context는 Device Context와 같은 Pixel Format를 갖는다. 그러나 Rendering Context와 Device Context는 같지 않다. Device Context는 window의 graphic component인 GDI의 정보를 갖고 있고 Rendering Context는 OpenGL의 정보를 갖고 있다. 응용프로그램은 Rendering Context를 생성하기 전에 반드시 Device Context를 생성해야 한다.

OpenGL call을 하는 thread는 반드시 current Rendering Context를 가져야 한다. (current Rendering Context를 갖지 않는 thread를 통해 OpenGL call을 하면 아무런 결과를 얻을 수 없다) 그러므로 응용프로그램은

- Rendering Context를 생성하고
- thread의 current Rendering Context로 지정하고
- OpenGL함수를 call해야한다.

함수의 call을 끝냈을 때는

- Rendering Context를 thread로 부터 분리하고
- Rendering Context를 제거한다.

응용프로그램은 하나의 window에 multiple Rendering Context를 갖을 수 있지만 하나의 thread는 하나의 current(active)Rendering Context만을 가질 수 있다.

Current Rendering Context는 관련된(지정한) Device Context를 가지고 있다. 이 Device Context는 Rendering Context를 생성할 때의 바로 그 Device Context가 아니어도 된다. 다만 반드시 같은 Device여야 하고 같은 Pixel Format을 가져야 한다.

(2) Rendering Context Function

- wglCreateContext( ): 새로운 Rendering Context를 생성한다
- wglMakeCurrent( ): thread의 current Rendering Context를 지정
- wglGetCurrentContext( ) : thread의 current Rendering Context의 핸들을 얻음
- wglGetCurrnetDC( ): thread의 current Rendering Context와 연결된 Device Context의 핸들을 얻음
- wglDeleteContext( ): Rendering Context를 제거한다

Rendering Context를 생성한 후에는 Device Context는 release되거나 dispose할 수 있다. Rendering Context에 전달되는 Device Context는 반드시 Display Device Context이어야 한다. 응용프로그램은 current Rendering Context와 연결된 Device Context를 release하거나 dispose할 수 없다. Current Rendering Context가 현재 thread에 not current하게 되면 dispose 할 수 있다. 응용프로그램은 thread의 current Rendering Context를 얻을 수 있다. 또한 current Rendering Context와 연결된 Device context도 얻을 수 있다. 응용프로그램의 current Rendering Context와의 관계를 not current로 하는 방법은

- NULL Rendering Context와 wglMakeCurrnet( )하거나
- 다른 Rendering Context와 wglMakeCurrnet( )하면된다.

(3) Pixel Format

OpenGL은 drawing Surfart에 몇가지 성질을 규정한다.

- pixel buffer - single / double-buffered
- pixel data - RGBA / color-indexed form
- color data를 저장하는데 쓰이는 bit 수
- depth(z-aixs) buffer를 저장하는데 쓰이는 bit 수
- stencil buffer를 저장하는데 쓰이는 bit 수
- 다양한 visibility mask

Window NT는 OpenGL을 구현하는데 PIXELFORMATDESCRIPTOR라는 structure를 사용해서 pixel format data를 저장하고 있다. Device Context는 여러가지 pixel format을 제공하는데 실행시는 보통 하나의 current pixel format를 가진다. 응용프로그램은 Device Context의 pixel format을 지정한 뒤 Rendering Context를 그에 맞게 상속해서 쓴다.

(4) Pixel Format 함수

- ChoosePixelFormat( ) : 지정한 pixel format에 가장 가까운 것으로 Device Context의 pixel format을 가져온다.

- SetPixelFormat( ) : Device Context의 current pixel format을 주어진 pixel format index에 지정한다.

- GetPixelFormat( ) : Device Context의 current pixel format의 index를 얻는다.

- DescribePixelFormat( ) : 주어진 Device Context와 pixel format index로 PIXELFORMATDESCRIPTOR의 member를 채워준다.

한 window의 Device Context에 SetPixelFormat( )으로 설정하면 Device Context가 참조하는 window의 pixel format도 같이 바뀐다. 그러므로 한번 이상 SetPixelFormat을 하는 것은 에러를 유발한다. 즉 한번 window의 pixel format이 지정되면 바꿀 수 없다.

(5) Front, Back and Other Buffers

OpenGL은 pixel data를 frame buffer에다 저장하고 사용한다. frame buffer는 logical buffer로 이루어져 있는데 color buffer, depth buffer, accumulation buffer, stencil buffer등이다. Windows NT에서 구현된 OpenGL은 image의 double-buffering을 지원한다. Off-screen buffer에 필요한 내용을 모두 드로잉한 후 on-screen buffer에다가 copy하여 animation과 같은 자연스러운 image changing을 할 때 쓰인다. Color buffer는 front buffer와 back buffer를 갖는다. Front buffer가 screen에 display되어 있는 동안 응용프로그램은 back buffer에다 drawing command로 그리라는 명령을 한다. 모든 그림이 그려졌을 때 SwapBuffers( )를 call하면 NT는 off-screen buffer를 on-screen buffer에다가 복사한다. Double-buffering을 할려면 PIXELFORMATDESCRIPTOR에 PFD_DOUBLEBUFFER flag를 지정해서 ChoosePixelFormat을 불러준다.

(6) Buffer 함수

- SwapBuffers( ) : off-screen buffer와 on-screen buffer를 swap하는 것이 아니라 off-screen buffer를 on-screen buffer에 복사하고 off-screen buffer를 undefined로 만든다.

(7) Font & Text

Windows NT에서 구현된 OpenGL은 single-buffered OpenGL window에만 GDI graphic을 지원하였다. 그러므로 GDI font와 text함수는 double_buffered OpenGL window에는 쓸 수 없다. double_buffered OpenGL window에서 text를 쓰기 위해서는 아래와 같은 3단계가 필요하다.

- Device Context를 위한 font를 선택한다.
- bitmaps display list생성한다.
- display lists를 이용하여 string에 글씨를 쓴다.

(8) Font & Text Function

- WglUseBitMaps( ) - character bitmap display lists set를 생성한다.

- WglUseFontBitmaps( )는 Device Context의 핸들을 통해 Device Context의 font를 지정하고 난 후에 wglUseFontBitMaps를 불러야 한다. 문자열을 쓸 때는 wglUseFontBitMaps한 후에 glCallLists( )로 쓴다. 문자열 쓰기가 끝난 후에는 glDeleteLists( )로 display lists를 삭제시켜야 한다.

제 5 절 GUI 구현

1. 화면 생성(응용프레임워크)

가. 응용프레임워크의 구조

AppWizard를 구동시켜서 "SurfART"의 응용프레임워크를 생성한다. "SurfART" 응용프레임워크에 적용된 속성은 다음과 같다.

표 2.1 MFC AppWizard에서 선택된 속성들

항 목	선 택 속 성
Project Type	MFC AppWizard(exe)
Application Type	Single Document
Database Support	None
OLE compound document	None
OLE Spport	No, please
OLE Automation	No, please
Application Feature	Dockable Toolbar Initial Status Bar Printing and Print Preview Use 3D Controls
MRU list 갯수	4
Source File Comments	Yes, please

Makefile Type	Visual C++ makefile
MFC Library	MFC in a shared dll

나. 결과화일과 기능

AppWizard를 사용해서 생성한 응용프레임워크는 아래와 같은 결과화일을 출력한다. 각각의 클래스마다 하나의 헤더와 실행화일을 가지며 기타 리소스화일이 있다.

표 2.2 AppWizard에 의해 생성되는 결과화일

헤더 화일	시행 화일	클 래 스	베이스 클래스	기 능
mainfrm.h	mainfrm.cpp	CMainFrame	CFrameWnd	툴바 생성, 서브툴바 생성과 관리, 상태바 생성
Surfart.h	Surfart.cpp	CSurfartApp	CWinApp	응용프로그램 초기화, 화일오픈
surfadoc.h	surfadoc.cpp	CSurfartDoc	CDocument	데이타 로드, 저장
surfavw.h	surfavw.cpp	CSurfartView	CView	데이타 전시, 사용자 입력 처리

stdafx.h	stdafx.cpp	-	-	응용프레임워크용 헤더를 포함
resource.h	-	-	-	프로그램의 자원(resource)에 대한 상수값 정의
Surfart.rc	-	-	-	프로그램의 자원(resource)에 대한 정의 - 메뉴, 다이얼로그 박스, 문자열, 키보드 엑셀러레이터, 아이콘 정의
Surfart.mak	-	-	-	프로젝트 화일
Surfart.clw	-	-	-	ClassWizard에서 사용되는 정보를 저장
resWSurfart.ico	-	-	-	표준 아이콘 화일
resWSurfart.rc2	-	-	-	AppStudio에서 정의되지 않는 자원 정의
resWtoolbar.bmp	-	-	-	주화면의 툴바에 쓰이는 비트맵

다. 추가되는 화일과 기능

"SurfART"에는 OpenGL의 사용과 다이얼로그바의 생성, Viewing Transformation 관리, OpenGL을 사용한 그림 그리기를 위해 다음과 같은 화일들이 추가된다.

표 2.3 추가되는 화일과 기능

헤더 화일	시행 화일	클 래 스	베이스 클래스	기 능
copengl.h	copengl.cpp	CopenGL	CObject	OpenGL과 윈도우즈 NT를 연결하기위한 클래스
mydialogbar.h	mydialogbar.cpp	CMyDialogBar	CDialogBar	다이얼로그바를 만들고 운영하기 위한 클래스
gldrawer.h	gldrawer.cpp	-	-	전체 Model의 Point 데이타를 전시하고 Picking을 관리하는 루틴

2. 프로그램의 구조

가. 기본클래스 - CSurfartDoc, CSurfartView, CSurfartApp, CMainFrame

응용프레임워크에서 CSurfartDoc 클래스는 프로그램 데이타를 디스크 화일로 부터 읽고 로딩할 뿐만 아니라 이 프로그램 데이타를 저장하는 일을 맡고 있다. CSurfartView 클래스는 view window를 관리한다. 이 클래스는 사용자 입력을 처리하고 문서를 view window내에서 그리고 다른 장치에 전시(display)하는 일을 한다. CMainFrame 클래스는 주 프로그램 윈도우를 관리한다. 이 윈도우는 메뉴, 경계, 타이틀바와 같은 사용자 인터페이스 오브젝트를 전시하고 버튼과 툴바, 상태바를 최소화하거나 최대화한다. View window는 메인 프로그램 윈도우에서 위와같은 오브젝트가 차지하는 영역을 제외한 나머지 부분에 위치한다. CSurfartApp 클래스는 전체적인 응용프로그램을 관리하는데 그 응용프로그램을 초기화하고 마지막에 삭제하는 것까지 일반적인 작업을 실행한다. MFC 응용프레임워크가 제공하는 기본클래스의 기능은 위와 같으나 "SurfART"에서는 위의 기능을 모두 사용하지 않는다. 특히 CSurfartDoc의 화일 로드와 저장 기능, CSurfartView의 display 기능은 "SurfART"만의 코드로 대치된다.

나. 프로그램 제어의 흐름

① CSurfartApp 클래스의 생성자인 CSurfartApp::CSurfartApp( )가 수행된다.

② MFC의 WinMain( )이 수행된다.

③ WinMain( )이 CSurfartApp::InitInstance( )를 호출한다.

④ WinMain()이 메세지 루프를 실행해서 각종 메세지를 처리한다.

⑤ WinMain()을 빠져나온다.

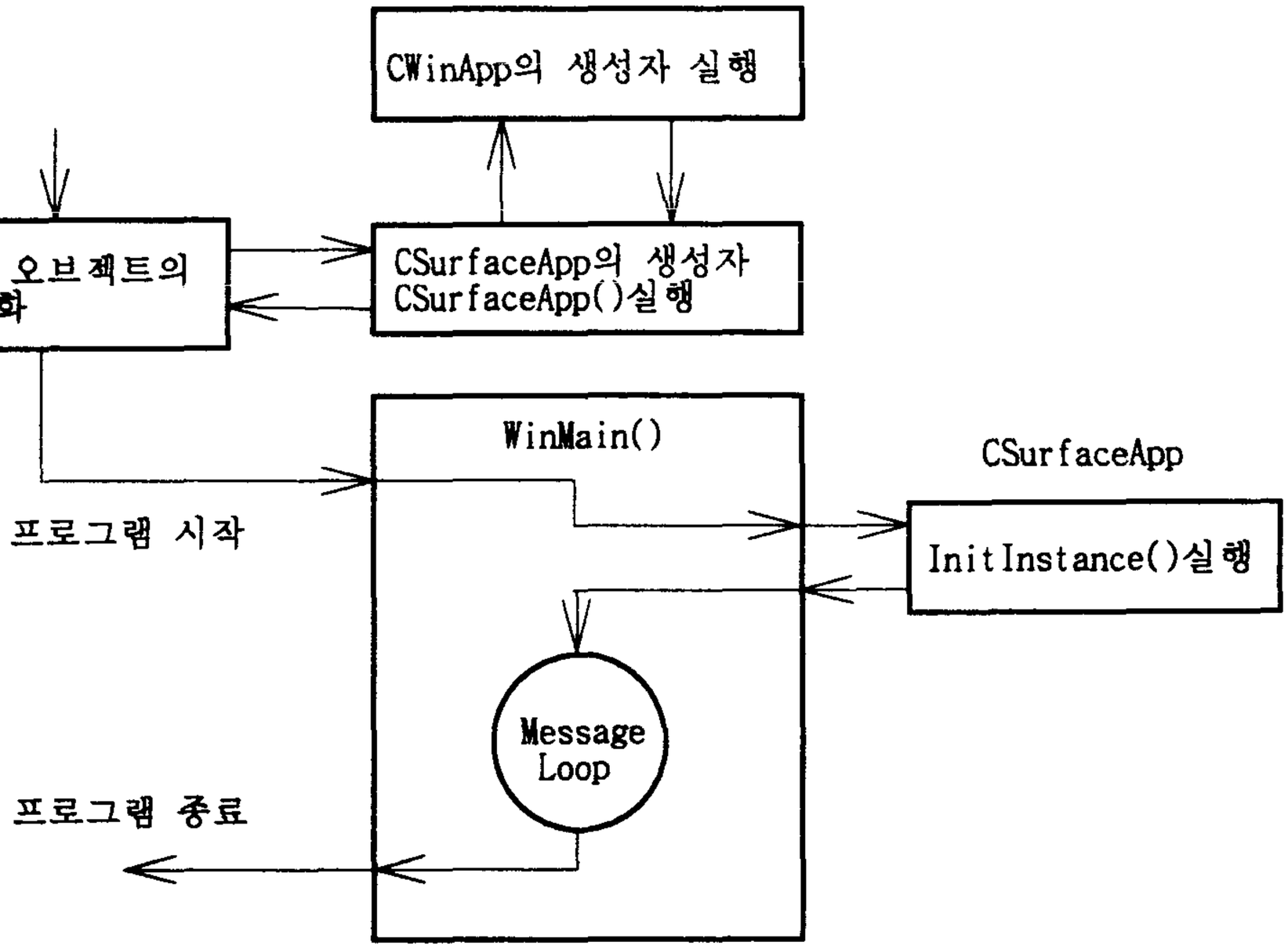


그림 2.4 "SurfART"의 프로그램 제어의 흐름

다. 메세지 관리

각각의 기본클래스(CSurfartDoc, CSurfartView, CSurfartApp, CMainFrame)와 Dialog 클래스는 윈도우 시스템으로부터 메시지를 받는다. 각 클래스는 보내지는 메시지를 처리하기 위한 핸들러를 만들 수 있는데 메시지와 핸들러간의 연결은 메시지 맵이 담당한다.

각 클래스의 메시지 맵은 다음과 같다. 예는 CSurfartView 클래스의 경우이다.

surfavw.h화일

```
class CSurfartView : public CView
{
```

```
// Generated message map functions
protected:
    //{{AFX_MSG(CSurfartView)
    .
    .
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    afx_msg void OnDisplayZoomZoomall();
    afx_msg void OnDisplayZoomZoomin();
    afx_msg void OnDisplayZoomZoomout();
    afx_msg void OnDisplayZoomPrevious();
    afx_msg void OnDisplayZoomRotation();
    afx_msg void OnDisplayZoomPanning();
    afx_msg void OnDisplayZoomMoveLeft();
    afx_msg void OnDisplayZoomMoveRight();
    afx_msg void OnDisplayZoomMoveUp();
    afx_msg void OnDisplayZoomMoveDown();
    .
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};
```

surfavw.cpp화일

```
IMPLEMENT_DYNCREATE(CSurfartView, CView)
BEGIN_MESSAGE_MAP(CSurfartView, CView)
//{{AFX_MSG_MAP(CSurfartView)
ON_WM_CREATE()
ON_COMMAND(ID_DISPLAY_ZOOM_ZOOMALL, OnDisplayZoomZoomall)
ON_COMMAND(ID_DISPLAY_ZOOM_ZOOMIN, OnDisplayZoomZoomin)
ON_COMMAND(ID_DISPLAY_ZOOM_ZOOMOUT, OnDisplayZoomZoomout)
ON_COMMAND(ID_DISPLAY_ZOOM_PREVIOUS, OnDisplayZoomPrevious)
ON_COMMAND(ID_DISPLAY_ZOOM_ROTATION, OnDisplayZoomRotation)
ON_COMMAND(ID_DISPLAY_ZOOM_PANNING, OnDisplayZoomPanning)
ON_COMMAND(ID_DISPLAY_ZOOM_MOVE_LEFT, OnDisplayZoomMoveLeft)
ON_COMMAND(ID_DISPLAY_ZOOM_MOVE_RIGHT, OnDisplayZoomMoveRight)
ON_COMMAND(ID_DISPLAY_ZOOM_MOVE_UP, OnDisplayZoomMoveUp)
ON_COMMAND(ID_DISPLAY_ZOOM_MOVE_DOWN, OnDisplayZoomMoveDown)
//}}AFX_MSG_MAP
// Standard printing commands
    ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview)
END_MESSAGE_MAP()
```

Surfavw.h 화일에서는 메시지에 응답할 핸들러 함수를 선언하고

surfavw.cpp에서는 메시지와 핸들러함수를 연결해준다. "ON_WM_CREATE()"는 WM_CREATE메시지에 대응하는 핸들러임을 의미하고 "ON_COMMAND(ID_DISPLAY_ZOOM_ROTATION, OnDisplayZoomRotation)"는 메뉴 메시지 ID_DISPLAY_ZOOM_ROTATION에 대한 핸들러 함수 OnDispalyZoomRotation()를 지정한다. 이렇게 각 클래스에서 처리될 메지시는 ClassWizard를 통해서 선택되고 핸들러함수도 생성된다.

만약 같은 메시지에 대해 위의 여러 기본클래스가 핸들러를 중복해서 가지고 있다면 아래와 같은 순서로 먼저 선언된 핸들러가 제어를 갖는다.

CSurfartDoc -> CSurfartView -> CSurfartApp -> CMainFrame

3. 메뉴 생성 및 관리

가. 메뉴의 정의

메뉴는 Surfart.rc화일에 Menu 항목에 정의되어 있다. "SurfART"의 주메뉴는 IDR_MAINFRAME이고 메뉴의 ID와 Text, Value의 예는 다음과 같다.

ID : ID_POINT_SKETCH

Text : Sketch

Value : 32790

단, 메뉴의 ID와 Text, Value의 정의는 ClassWizard의 Menu Design tool로 해야 하며 메뉴 ID의 값은 자동으로 정의된다.

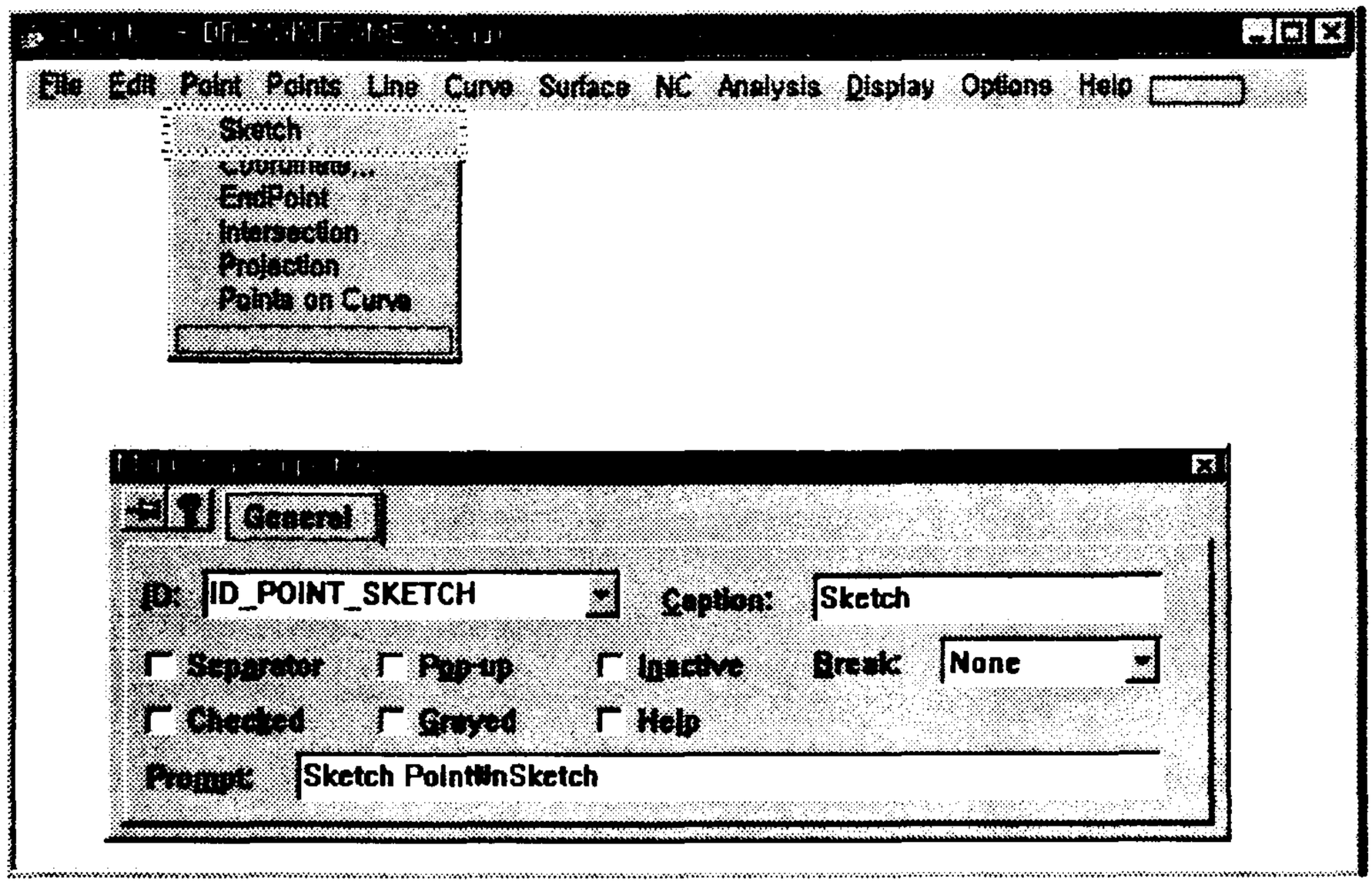


그림 2.5 메뉴 생성 도구

나. 메뉴 메세지 핸들러 생성

메뉴가 정의되면 ClassWizard의 Message Maps를 이용하여 메뉴 메시지(Command)를 받을 클래스와 메뉴 메시지(Command) 핸들러 함수를 생성한다. 메뉴 ID와 핸들러 함수의 리스트는 Appendix A에 있다.

4. 툴바 생성 및 관리

가. 툴바 비트맵 생성

툴바는 Surfart.rc화일에 Bitmap 항목에 정의되어 있다. "SurfART"에서 쓰는 툴바 버튼으로 정의된 bitmap 예는 다음과 같다. 각각의 CToolBar 오

브젝트 변수는 CMainFrame 클래스에 선언되어 있다.

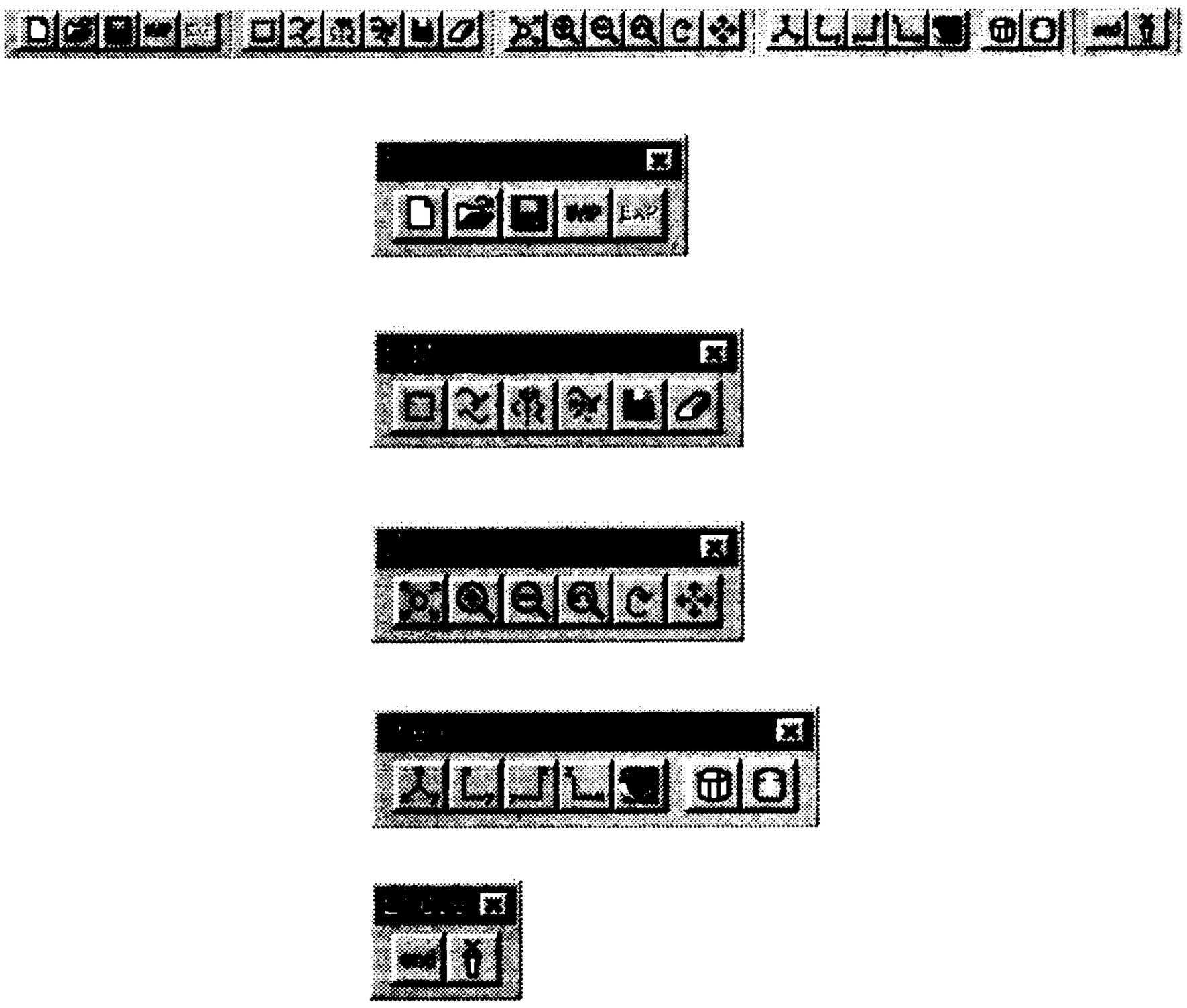


그림 2.6 "surfART"의 툴바와 기능

표 2.4 툴바 비트맵 ID와 CToolBar 오브젝트

Bitmap ID	CToolBar 오브젝트
IDR_FILE	m_wndFileTBar
IDR_EDIT	m_wndEditTBar
IDR_ZOOM	m_wndZoomTBar
IDR_VIEW	m_wndViewTBar

IDR_ENDSEL	m_wndEndselTBar

나. 툴바와 메뉴의 연결

툴바 비트맵의 각 버튼과 메뉴 ID를 연결한다. CMainFrame 클래스 CMainFrame::OnCreate() 함수에서 실행한다.

m_wndFileTBar.Create( ) - 툴바를 생성한다.

m_wndFileTBar.LoadBitmap( ) - 툴바 Bitmap을 각 버튼에 연결한다.

m_wndFileT.SetButtons( ) _ 메뉴 ID와 툴바 버튼을 연결한다.

m_wndFileT.EnableDocking( ) - 툴바가 docking할 수 있도록 한다.

ShowControlBar(&m_wndFileTBar, FALSE, FALSE) - 생성된 툴바를 display되지 않게 한다.

5. Icon Menu의 생성 및 관리

Icon Menu는 메인 윈도우 영역 왼쪽에 Dialog Bar위에 Window95 Common Control인 ListCtrl과 TabCtrl를 사용하여 아이콘으로 보다 편리하게 기능을 알수 있도록 표현한 Menu이다. 이 메뉴는 Top 메뉴보다 선택하기가 용이하고 아이콘으로 되어 있기 때문에 그 기능을 쉽게 알 수 있도록 구성되어 있다.

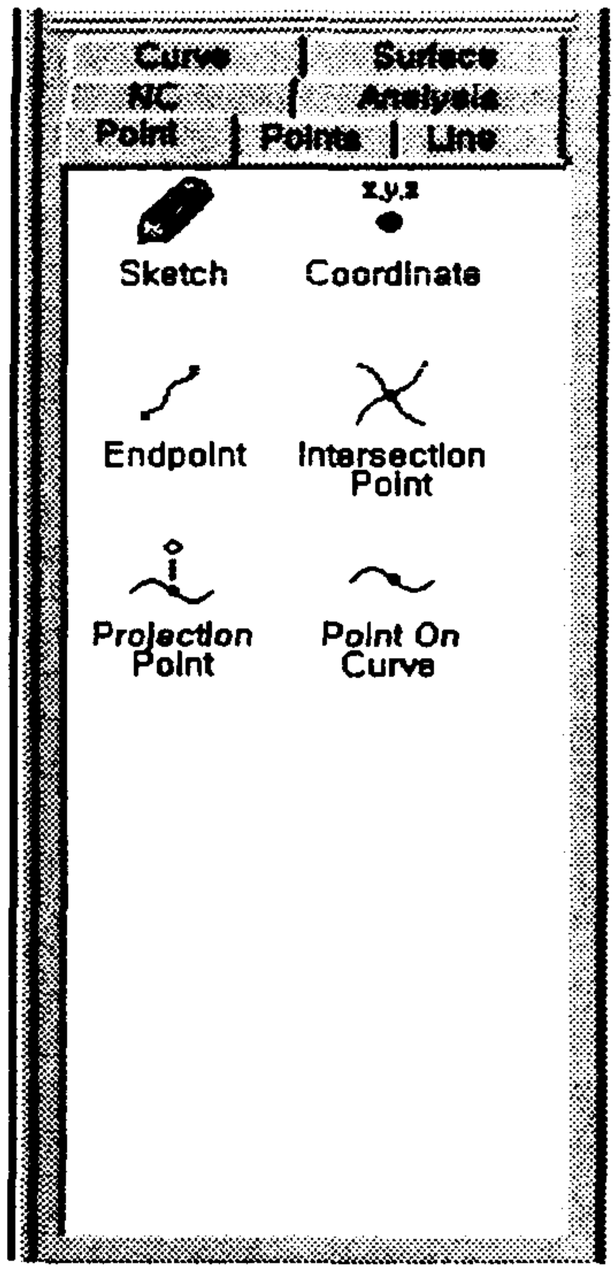


그림 2.7 icon 메뉴

가. Dialog Bar 생성

Dialog Bar는 메인 윈도우의 좌측에 위치해 있는 Bar형식의 다이얼로그 박스로서 이미지는 Surfart.rc의 Dialog Box Design에서 생성하고 Bar의 생성은 CMainFrame 클래스 CMainFrame::OnCreate()함수에서 실행한다.

m_wndMyDlgBar.Create( ) - Dialog Bar를 생성하고 Display한다

m_wndMyDlgBar.EnableDocking( ) - 다이얼로그바가 docking할 수 있도록 한다.

나. ListCtrl 생성

리스트 콘트롤은 window95의 common control로서 리스크 박스 콘트롤과 같이, 사용자가 선택할 수 있는 리스트 항목을 제시한다. 그러나 텍스트만을 나타내는 리스트 박스 콘트롤과는 달리, 리스트 콘트롤은 텍스트와 함

께 아이콘과 컬럼 헤더(column header)를 함께 보여준다.

MFC 라이브러리에서는 리스트 콘트롤에 대하여 CListCtrl 클래스를 제공한다[2-4]. 다이얼로그 바에서 리스트 콘트롤을 사용하기 위해서는 다이얼로그 리소스에 리스트 콘트롤을 추가한 후, CMainFrame 클래스 선언부에 CListCtrl 또는 CListCtrl의 파생 클래스 오브젝트를 정의한다.

(1) ListCtrl의 아이콘 생성 및 관리

아이콘은 Surfart.rc화일에 Icon 항목에 정의되어 있다. "SurfART"에서 쓰는 아이콘의 예는 Appendix A에 있다.

아이콘들을 효율적으로 관리하기 위해 이미지 리스트 콘트롤을 생성하여 관리한다. 이미지 리스트 콘트롤(CImageList)은 같은 크기의 비트맵 또는 아이콘 이미지의 집합을 관리하며, 일반적으로 리스트 콘트롤(CListCtrl) 또는 트리 콘트롤(CTreeCtrl)에서 사용하는 이미지를 저장, 관리한다.

리스트 콘트롤에서는 32x32 크기의 큰 아이콘과 16x16 크기의 작은 아이콘 등 2개의 이미지 리스트를 사용하며, 트리 콘트롤에서는 16x16 크기의 아이콘 이미지 리스트만을 사용한다.

(2) 이미지 리스트 콘트롤과 ListCtrl의 연결

이미지 리스트 콘트롤을 생성하고 ListCtrl과 연결하기 위해서는 다음 과정을 수행한다.

① 이미지 리스트 콘트롤을 필요로 하는 클래스 선언부에 CImageList 오브젝트를 정의한다.

② 해당 클래스의 적절한 멤버 함수에서 CImageList::Create 멤버 함수를 사용하여 이미지 리스트 콘트롤을 생성한다.

③ CImageList::Add 멤버 함수를 사용해 필요한 이미지 정보를 이미지 리스트 콘트롤에 추가한다.

④ 이미지 리스트를 필요로 하는 콘트롤(CListCtrl 또는 CTreeCtrl)의 SetImageList 멤버 함수를 사용하여 해당 콘트롤에 이미지 리스트를 지정한다.

다. TabCtrl 생성

카드위에 약간 튀어나온 윈도우인 카드헤더를 클릭하면 그 것에 해당하는 카드 리스트를 보여주는 컨트롤이다. "SurfART"에서는 7개의 ListCtrl을 이용하여 메뉴를 대신 아이콘으로 메세지 처리를 관리한다. 따라서 각 ListCtrl을 다이얼로그바에서 표현하기 위하여 TabCtrl을 이용한다. TabCtrl을 선택하면 해당하는 ListCtrl을 화면에 표현해 준다.

라. ListCtrl의 아이콘과 메뉴의 연결

(1) 윈도우 공통 콘트롤의 메시지 처리

윈도우 3.1에서 콘트롤(child Window)에 대하여 마우스 클릭 등의 이벤트가 발생할 때 상위 윈도우(Parent Window)에는 BN_CLICK 등 통지(notification) 코드와 함께 WM_COMMAND 메시지가 전달된다.

윈도우 공통 콘트롤에 대해서는 표준 콘트롤보다 많은 정보를 필요로 한다. 따라서, Win32에서는 WM_NOTIFY 메시지를 추가하여 이 메세지를 통하여 이들 추가된 정보를 넘겨준다. WM_NOTIFY 메시지의 wParam에는 메시지를 보내

는 콘트롤의 ID를 넘겨주고, lParam에는 NMHDR 구조체 또는 NMHDR 구조체를 멤버로 포함하는 구조체에 대한 포인터를 넘겨준다.

모든 윈도우 공통 콘트롤은 다음과 같은 통지(notification) 코드를 NMHDR 구조체에 넘겨준다.

표 2.5 통지 코드의 종류

통지 코드	발생 원인
NM_CLICK	사용자가 콘트롤에 왼쪽 마우스 버튼을 클릭할 때
NM_DBCLICK	사용자가 콘트롤에 왼쪽 마우스 버튼을 더블 클릭할 때
NM_RCLICK	사용자가 콘트롤에 오른쪽 마우스 버튼을 클릭할 때
NM_RDBLCLK	사용자가 콘트롤에 오른쪽 마우스 버튼을 더블 클릭할 때
NM_RETURN	사용자가 콘트롤에서 ENTER 키를 누를 때
NM_SETFOCUS	콘트롤이 포커스를 받을 때
NM_KILLFOCUS	콘트롤이 포커스를 읽을 때

(2) 아이콘의 메시지와 메뉴 메시지의 연결

ListCtrl의 각 아이콘과 메뉴와 연결한다. ON_NOTIFY(NM_DBLCLK, IDS_POINT, OnDblclkItems)와 같은 notify함수를 이용하여 연결한다. 해당 아이콘을 더블 클릭할 때의 발생하는 함수 OnDblclkItems을 살펴보면 다음과 같다. 이 함수에서는 선택된 아이콘의 text 이름이 해당 메뉴의 이름과 같으면 그 메뉴에 해당하는 함수를 call하여 실행하도록 하여 아이콘 메시지 처리를 메뉴 메시지처리와 연결하는 역할을 한다.

```
void CSurfartView::OnDblclkItems(NMHDR* pNMHDR, LRESULT* pResult)
{
        // double-clicking is just a different way to press the button
```

```
 CListCtrl* pList;

CMainFrame *pMain = (CMainFrame *)AfxGetMainWnd();
pList = (CListCtrl*) pMain->m_wndMyDialogBar.GetDlgItem(IDS_POINT);
CWnd* pOne = pMain->m_wndMyDialogBar.GetDlgItem(IDC_BUTTON1);
ASSERT(pList != NULL);

    LV_ITEM itemSelected;
    int nSelected = pList->GetNextItem(-1, LVNI_SELECTED);
    if (nSelected == -1)
            return;

    TCHAR szName[80];

    itemSelected.iItem = nSelected;
    itemSelected.iSubItem = 0;
    itemSelected.mask = LVIF_PARAM | LVIF_TEXT;
    itemSelected.pszText = szName;
    itemSelected.cchTextMax = 30; //ELEMENTS(szName);
    pList->GetItem(&itemSelected);

    // and pop up a message box
    // Point function
    if(strcmp(szName, "Coordinate") == 0) {
            OnPointKeyboard();
    // Points function
    } else if(strcmp(szName, "Reduction") == 0) {
            OnPointsDatareduction();
    } else if(strcmp(szName, "Average Filtering") == 0) {
            OnPointsAveragefiltering();
    }
    -
    -
    -
    } else {
            MessageBox("아직 지원되지 않는 기능입니다.");
}
    *pResult = 0;
}
```

6. 다이얼로그 박스 작성

가. 다이얼로그 박스 이미지 생성

다이얼로그 박스는 surfart.rc화일에 Dialog 항목에 정의되어 있다. "SurfART"에서 쓰는 다이얼로그 박스 ID와 사용 용도는 Appendix A에 있다.

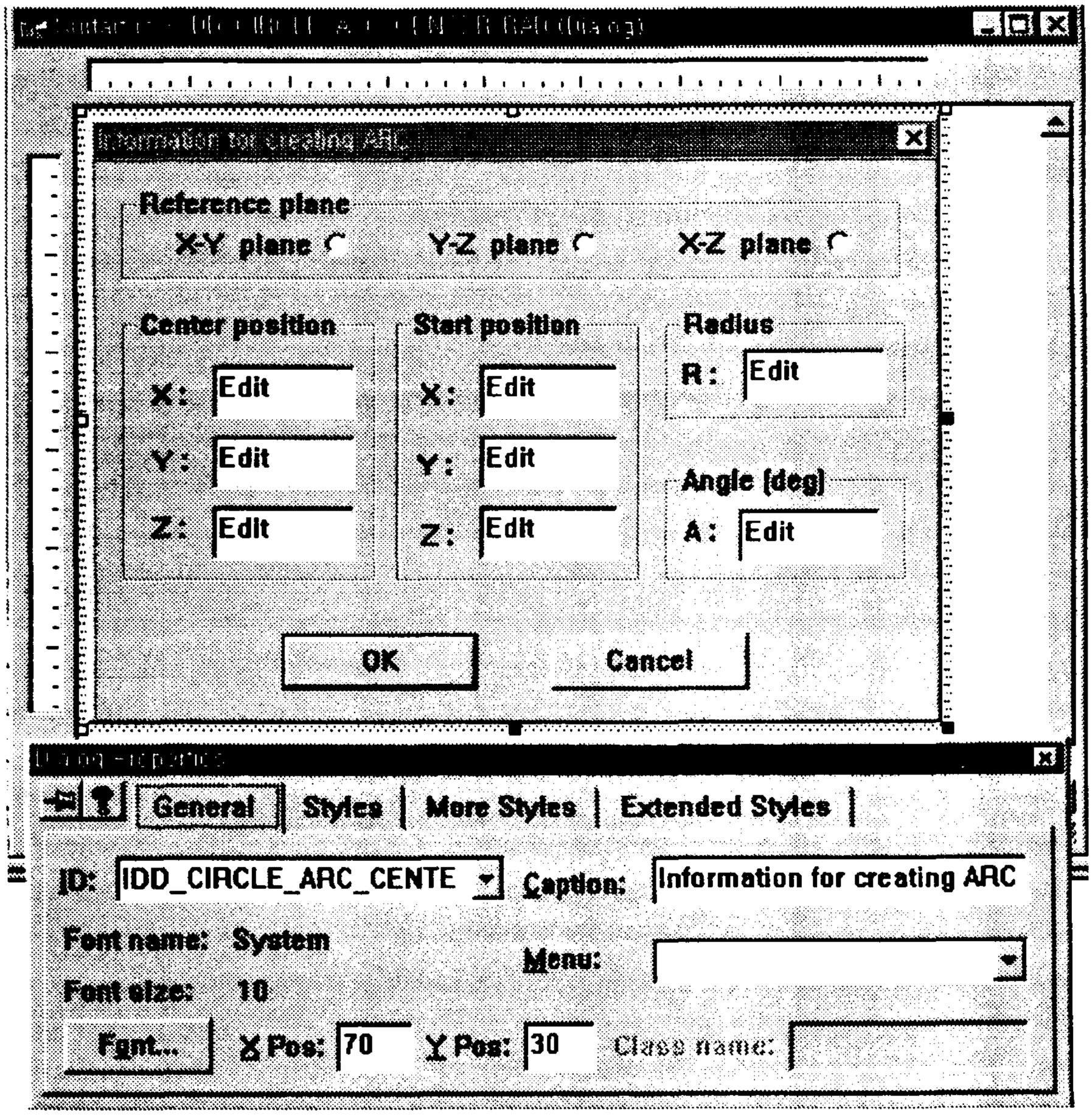


그림 2.8 다이얼로그 생성 도구

나. 다이얼로그 박스 경영을 위한 클래스 생성

ClassWizard의 Add Class를 실행해서 베이스 클래스로 CDialog 클래스를 선택한다. "SurfART"에서 쓰이는 다이얼로그 박스를 위한 클래스 화일명은 Appendix A에 있다.

7. 메시지 관리

가. 상태바 생성

보통 응용워크프레임을 사용하여 프로그램을 생성하면 기본적으로 MFC 라이브러리의 CStatusBar 클래스를 통하여 상태바를 구현하게 된다. 그러나 본 시스템에서는 상태바에 x, y, z의 좌표 표현과 아이콘 메세지 처리를 용이하게 표현하기 위해 새롭게 상태바를 생성하였다. 즉 다이얼로그바의 다른 형태로 상태바를 표현하였다. 따라서, 이미지는 surfart.rc의 Dialog Box Design에서 생성하고 Bar의 생성은 CMainFrame 클래스 CMainFrame::OnCreate() 함수에서 실행한다.

그림 2.9 "surfART"의 상태바

나. 상태바의 메시지 처리방법

시스템의 message는 사용자가 시스템을 안정적이고 효율적으로 사용

할 수 있도록 하는 중요한 정보이다. 본 시스템은 두 종류의 message를 사용자에게 제공한다. 첫째는 시스템 수행시 발생된 오류를 알려주는 message로 MFC에서 지원하는 message box를 통하여 화면의 중앙에 display된다. 둘째는 사용자에게 시스템의 사용방법을 안내하는 message로 화면 하단에 있는 상태바에 display된다. 일반적으로 상태바에 message를 display하기 위해서는 CMainFrame 클래스에 있는 상태바 멤버를 access하고 상태바의 member함수를 이용해야 한다. 그러나 상태바는 protected 멤버이므로 응용프로그램에서는 직접적으로 access 할 수 없게 된다. 그러므로 본 시스템에서는 상태바 멤버를 access할 수 있는 멤버함수 StatusMessageOut()과 StatusMessageClear()를 public하게 정의하여 응용프로그램에서도 이 함수들을 사용하여, 편리하게 message를 사용자에게 제공하거나 제거할 수 있도록 하였다.

StatusMessageOut() 함수는 display 할 위치를 지정하는 에디트 콘트롤(edit control)과 message의 내용을 매개변수로 전달받아 해당 위치에 message를 display 하는 기능을 수행한다. 이런 과정을 거쳐 display 된 message는 다른 message가 display될 때까지 계속 유지된다. 그러나 StatusMessageClear()함수는 지우고자 하는 에디트 콘트롤을 매개변수로 받아 해당 에디트 콘트롤 영역을 빈 여백으로 바꾸는 기능을 한다.

루틴명: void CMainFrame::StatusMessageOut(int nID, LPCTSTR lpszString)

nID : 메시지를 출력될 해당 에디트 콘트롤의 아이디

lpszString : 출력될 메세지 내용

루틴명: void CMainFrame::StatusMessageClear(int nID)

nID : 메시지를 지워야 할 해당 에디트 콘트롤의 아이디

여 백

제 3 장 기본 기능 모듈

제 1 절 수치계산용 객체의 정의

곡면모델러를 C++ 언어로 개발하기 위해서는 이용되는 모든 데이터들을 하나의 객체로 간주하여 클래스로 정의하여 사용하는 것이 편리하다. 클래스란 객체를 표현하는 방법으로 객체를 유일하게 정의하는 데이터를 갖고 있는 데이터 멤버(data member)와 데이터 멤버를 조작할 수 있는 멤버함수(member function, or method)로 구성된다. 또한 클래스에는 객체를 조작할 수 있는 연산자(operator)를 정의하여 사용할 수 있다. 그러나 공통된 데이터 멤버를 갖고 있는 객체들을 독립된 클래스로 정의하면 데이터의 중복이 발생하므로 이러한 중복을 피하기 위해 파생 클래스(derived class)를 이용한다. 공통된 데이터 만을 데이터 멤버로 기반 클래스(base class)를 정의하고 그렇지 않은 데이터만으로 파생 클래스를 정의하는 것이다. 파생 클래스는 기반 클래스의 모든 데이터 멤버와 멤버함수를 계승받아 사용할 수 있다. 일반적으로 곡면 모델링 알고리즘 계산에 이용되는 수치계산용 클래스(mathematical class)는 점(position), 벡터(vector), 단위벡터(unit vector), 매개변수(parameter), 매개변수 공간의 점(parametric position), 매개변수 공간의 벡터(parametric vector), 매개변수 공간의 단위벡터(parametric direction), 변환행렬(transformation matrix), interval, box등 여러가지가 있다. 수치계산용 클래스의 헤더 파일은 APPENDIX B에 수록하였다. 그림 3.1은 본 연구에서 정의한 수치계산용 클래스의 구조를 나타낸 것이다.

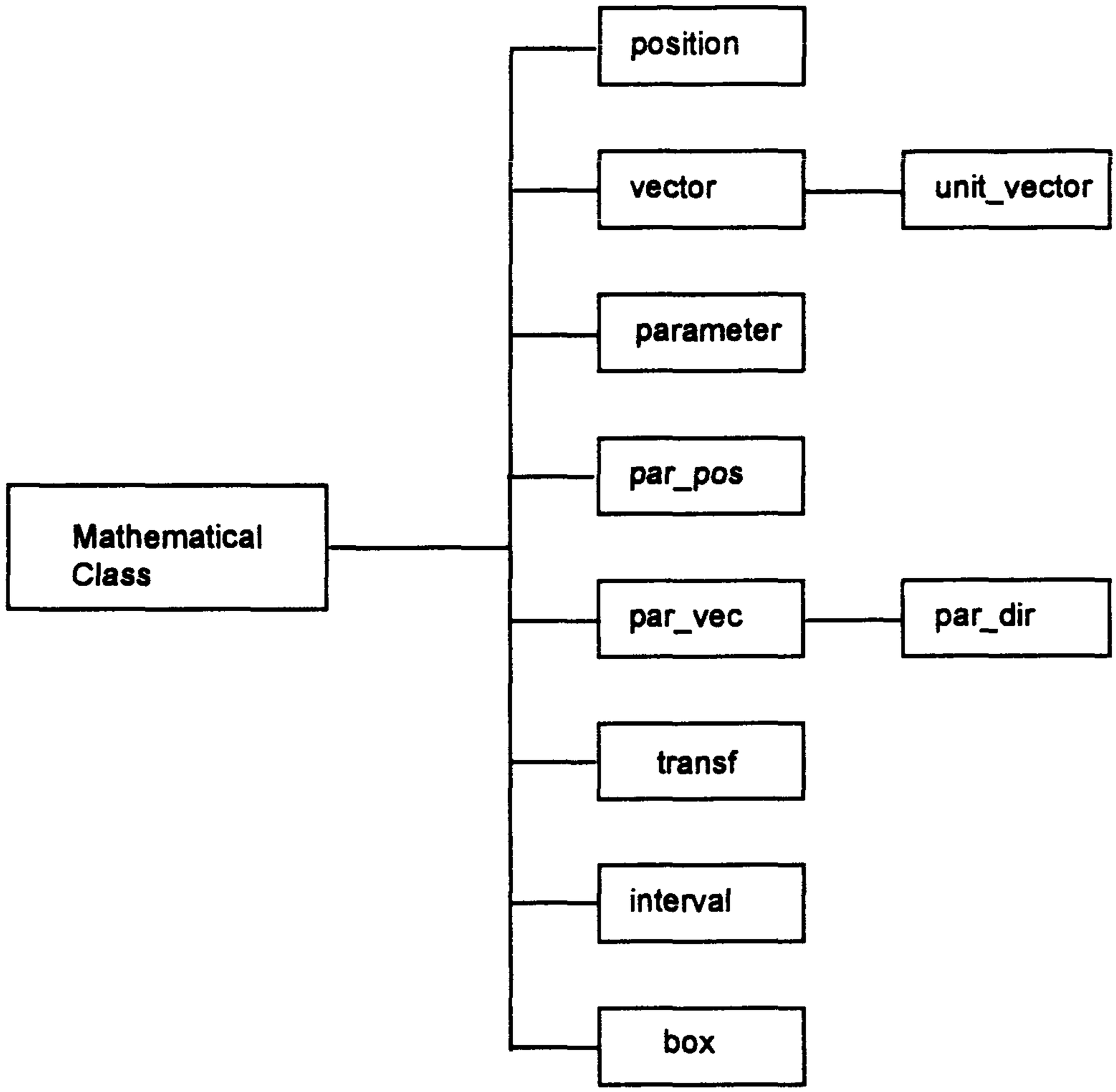


그림 3.1 수치계산용 클래스의 구조

1. Position

position 클래스는 3 차원 cartesian 공간에서 어떤 한 점의 위치를 position 으로 정의한다. position 클래스의 멤버는 double 형의 x, y, z 좌표값이며, 생성자(constructor)는 double 형의 x, y, z 좌표를 인자로 갖는다. 이 클래스의 멤버 함수는 position 의 x, y, z 값을 읽어오는 함수와 주어진 값을 position 의 x, y, z 값을 지정하는 함수가 있다. 그 밖의 멤버 함수는 표 3.1 과 같이 나타낸다.

표 3.1 position 클래스의 멤버 함수

멤버 함수	기 능
position operator*(position const &p1, double d1)	position p1 의 멤버 각각에 d1 을 곱한다.
vector operator-(position const &p1, position const &p2)	position p1 의 멤버에서 position p2 의 멤버를 빼어서 vector 를 생성한다.
Position operator+(position const &p1, vector const &v1)	posiiotn p1 의 멤버와 vector v1 의 멤버를 더해서 position 을 생성한다.
Position operator+(vector const &v1, position const &p1)	vector v1 의 멤버와 position p1 의 멤버를 더해서 position 을 생성한다.
Position operator-(position const &p1, vector const &v1)	position p1 의 멤버에서 vector v1 의 멤버를 빼어서 position 을 생성한다.
double operator%(position const &p1, vector const &v1)	position p1 의 멤버와 vector v1 의 멤버를 곱해준다.
double operator%(vector const &v1, position const &p1)	vector v1 의 멤버와 position p1 의 멤버를 곱해준다.
Position operator*(position const &p1, unit_vector const &uv1)	position p1 과 unit vector uv1 과 cross product 한다.
logical operator==(position const& p1, position const& p2)	position p1 과 p2 를 비교하여 같으면 true, 아니면 false 를 리턴한다.

2. Vector, unit vector

vector 클래스는 3 차원 cartesian 공간상의 좌표(x, y, z)에서 시작점을 좌표계의 원점으로 생각했을 때의 vector 로 정의한다. vector 클래스의 멤버는 double 형의 x, y, z 값이다. unit_vector 클래스는 3 차원 cartesian 공간상에서 크기가 1 인 벡터로 정의하고, vector 클래스를 기반 클래스로 갖는 파생 클래스이다. vector 클래스 멤버 함수는 표 3.2 와 같이 나타내고, unit_vector 클래스의 멤버

함수는 표 3.3 과 같이 나타낸다.

표 3.2 vector 클래스의 멤버 함수

멤버 함수	기 능
vector operator+(vector const &v1, vector const &v2)	vector v1 의 멤버와 vector v2 의 멤버를 더한다.
vector operator-(vector const &v1, vector const &v2)	vector v1 의 멤버와 vector v2 의 멤버를 빼준다.
double operator%(vector const &v1, vector const &v2)	vector v1 의 멤버와 vector v2 의 멤버를 곱한다.
vector operator*(vector const &v1, vector const &v2)	vector v1 과 vector v2 를 cross product 한다.
vector operator*(double d1, vector const &v1)	vector v1 의 멤버 각각에 d1 을 곱한다.
vector operator/(vector const &v1, double d1)	vector v1 의 멤버 각각을 d1 으로 나눈다.
double vector::len()const	vector 의 크기를 구한다.
unit_vector normalise(vector const &v1)	vector v1 을 normalize 한다.

표 3.3 unit_vector 클래스의 멤버 함수

멤버 함수	기 능
unit_vector operator-(unit_vector const &uv1)	unit_vector uv1 의 멤버에 -1 을 곱한다.
Double operator%(position const &p1, unit_vector const &uv1)	position p1 과 unit_vector uv1 을 dot product 한다.
double operator%(unit_vector const &uv1, position const &p1)	unit_vector uv1 과 position p1 을 dot product 한다.
Position operator*(unit_vector const &uv1, position const &p1)	unit_vector uv1 과 position p1 을 cross product 한다.
logical operator==(unit_vector const& uv1, unit_vector const& uv2)	unit_vector uv1 과 uv2 를 비교하여 같으면 true, 아니면 false 를 리턴한다.

3. Parameter, parametric position, parametric vector, parametric direction

parameter 클래스는 곡선 파라미터 값으로 정의된다. parameter 클래스는 double 형의 값을 갖는다. par_pos 클래스는 곡면상의 점을 나타내려 할 때 u, v 파라미터 공간에서 정의되는 2 차원 파라미터로 나타낸다. par_vec 클래스는 2 차원 파라미터 공간상의 벡터로 정의한다. par_dir 클래스는 파라미터 공간상의 방향을 나타내는 유니트 벡터이고, par_vec 클래스를 기반 클래스로 갖는 파생 클래스이다. 표 3.4 는 parameter 클래스의 멤버 함수를 나타낸 것이다.

표 3.4 parameter 클래스의 멤버 함수

멤버 함수	기능
double value()	parameter 의 멤버를 리턴한다.
void set_value(double pval)	pval 을 parameter 의 멤버에 지정한다.
operator double() const	parameter 의 멤버를 리턴한다.
parameter operator-() const	parameter 의 멤버에 -1 을 곱한다.
double operator+(parameter const &p1, parameter const &p2)	parameter p1 과 parameter p2 를 더한다.
double operator+(parameter const &p, double d)	parameter p 과 double 형의 d 를 더한다.
double operator+(double d, parameter const &p)	double 형의 d 와 parameter p 를 더한다.
double operator-(parameter const &p1, parameter const &p2)	parameter p1 에서 parameter p2 을 빼준다.
double operator-(parameter const &p, double d)	parameter p 에서 double 형의 d 를 빼준다.
double operator*(parameter const &p1, parameter const &p2)	parameter p1 과 parameter p2 를 곱한다.
double operator*(parameter const &p, double d)	parameter p 와 double 형의 d 를 곱한다.
double operator*(double d, parameter const &p)	double 형의 d 에 parameter p 를 곱한다.
double operator/(parameter const &p1, parameter const &p2)	parameter p1 을 parameter p2 로 나눈다.
double operator/(parameter const &p, double d)	parameter p 를 double 형의 d 로 나눈다.
double operator/(double d, parameter const &p)	double 형의 d 를 parameter p 로 나눈다

4. Transformation

형상 요소의 점을 좌표계를 기준으로 주어진 양만큼 이동하고 회전하였을 때의 새로운 좌표값을 계산하여야 하는데 이 과정에 해당하는 변환행렬(transformation matrix)을 원래의 좌표값에 곱함으로써 수행된다. 여기에서 호모지어스 좌표(homogeneous coordinates)를 다루기 위하여 4 × 4 호모지니어스 변환행렬을 멤버로 갖는 transf 클래스를 정의하였다. 표 3.5 는 transf 클래스의 멤버 함수를 나타낸 것이다.

표 3.5 transf 클래스의 멤버 함수

멤버 함수	기 능
transf scale_transf(double sf)	global scaling factor sf 를 이용하여 크기 변환한다.
transf scale_transf(double sfx, double sfy, double sfz)	local scaling factor sfx, sfy, sfz 를 이용하여 크기변환한다.
transf rotate_transf(double angle, vector const &vec)	회전각 angle, 회전축 vec 를 이용하여 회전변환한다.
transf translate_transf(vector const &vec)	이동변환벡터 vec 를 이용하여 이동변환한다.
transf operator*(transf const &tfm1, transf const &tfm2)	transf tfm1 과 transf tfm2 를 곱한다.
vector operator*(vector const &vec, transf const &tfm)	vector vec 에 변환행렬 tfm 를 곱하여 변환한다.
unit_vector operator*(unit_vector const &unit_vec, transf const &tfm)	unit_vector unit_vec 에 변환행렬 tfm 를 곱하여 변환한다.
Position operator*(position const &ps, transf const &tfm)	position ps 에 변환행렬 tfm 를 곱하여 변환한다.

5. Interval, box

interval 클래스는 1차원 적인 구간의 크기를 양 끝의 수치로 정의하는 클래스로 작은 수치을 low, 큰 수치를 high 멤버로 저장한다. interval 클래스의 기본 생성자는 low가 high보다 크게 지정하며, 정의된 interval 객체가 사용되는 객체인지를 검사하는 멤버 함수를 갖고 있다. 그 외에 멤버 데이터에 접근할 수 있는 함수와 interval 객체간의 포함 관계를 조회하는 연산자를 갖고 있다. box 클래스는 세 개의 interval 객체를 멤버 데이터로 하여 3차원 공간상에 정의되는 영역을 표현하는 클래스이다. box 클래스는 임의의 위상 요소나 형상 요소의 크기를 표현하는데 사용되며, 기본적인 멤버 함수 이외에 표 3.6과 같이 박스의 포함 관계나 교차 영역을 구하는 함수를 갖고 있다.

표 3.6 box 클래스의 멤버 함수와 연산자

멤버 함수	기능
position low() const; position high() const;	박스 영역의 최소, 최대 위치를 계산한다.
box operator|(box const &, box const &);	두 박스를 결합하여 새로운 box를 생성한다.
box operator&(box const &, box const &);	두 박스의 교차 영역으로 이루어진 box를 생성한다.
logical operator&&(box const &, box const &);	두 박스의 교차 여부를 판단한다.
logical operator<<(position const &) const;	임의의 점이 박스에 포함되는지를 판단한다.
logical operator>>(box const &) const;	박스가 주어진 박스에 에 포함되는지를 판단한다.
logical operator==(box const &) const;	박스가 주어진 박스와 동일한지를 판단한다.

제 2 절 객체 관리 기능

데이터 구조를 효율적으로 관리하고, 필요한 데이터를 빠르고 편리하게 얻기 위해서는 데이터를 개별 단위로 관리하는 것보다 집합체로 관리하는 것이 편리하다. C++에서는 이러한 방법으로 배열(array)를 제공하고 있으나 배열은 정의(definition) 단계에서 크기(size)가 결정되어야 하며, 메모리 관리가 불편한 단점을 갖고 있다. 또한, MFC 에서도 이러한 기능을 할 수 있는 클래스를 제공하나 GUI 와 독립된 kernel 을 개발하기 위해서는 C++에서 제공하는 객체만 사용해야 한다. 그러므로 본 시스템의 구현에는 배열과 같이 데이터를 묶음으로 관리할 수 있고, 배열의 단점을 제거한 객체 클래스 PTRLIST 를 정의하여 사용하였다. PTRLIST 는 C++ 언어의 템플레이트(template) 기능을 이용하여 구현하였으므로, 시스템에서 사용하는 모든 객체와 데이터를 집합체로 관리할 수 있으며, 데이터 구조를 관리하는데 많이 사용되고 있다. 특히, 이 클래스는 데이터 구조에 저장된 형상 요소의 포인터를 리스트로 관리하는 기능에 사용되어, 데이터 구조를 조회하지 않고 형상 요소를 빠르게 화면에 디스플레이하거나, 선택(picking, selection)하는데 중요하게 이용된다. PTRLIST 클래스는 일정한 수의 객체 배열을 갖는 객체 클래스 PTRLUMP 를 doubly-linked list 로 연결하여 객체를 묶음으로 관리하며, 새로운 객체의 추가는 멤버 함수를 이용한다.

1. PTRLUMP class

PTRLUMP 는 PTRLIST 를 구성하는 기본 단위인 객체의 집합체를 구현한 클래스이다. PTRLUMP 는 객체를 저장하는데 배열을 이용함으로 생성될 때 저장되는 객체의 갯수를 지정해야 한다. 지정된 크기를 가진 PTRLUMP 가

모여 하나의 PTRLIST 를 구성함으로 관리되는 객체의 갯수나 객체를 조회하는 속도 등을 고려하여 결정해야 한다. 아래 코드는 PTRLUMP 를 정의하는 것으로, 멤버로는 doubly-linked list 를 구현하는데 사용할 PTRLUMP 클래스의 포인터, 객체를 저장하는 공간, 저장되어 있는 객체의 수 등으로 되었다. PRTLUMP_SIZE 는 하나의 PTRLUMP 에 최대로 저장할 수 있는 객체의 수이며, 구현된 시스템에서는 10 개로 지정되어 있다.

```
// Definition of class PTRLUMP

template <class T>
class PTRLUMP {
                    PTRLUMP *pre_ptrlump;       // The pointer to previous PTRLUMP
                    PTRLUMP *nxt_ptrlump;       // The pointer to next PTRLUMP

                    long no_items;              // The number of object-pointers
                    T items[PTRLUMP_SIZE];      // The pointers to objects

                    friend class PTRLIST<T>;

          public:
                    // Default constructor
                     PTRLUMP() { no_items = 0; pre_ptrlump = NULL; nxt_ptrlump = NULL; }

                    // Default destructor
                     ~PTRLUMP() { ;}
};
```

2. PTRLIST class

PTRLIST 클래스는 그림 3.2 와 같이 PTRLUMP 를 doubly-linked list 로 연결하여 많은 객체를 하나의 변수로 관리하는 클래스이다. 다음의 코드는 PTRLIST 클래스를 정의하는 헤더 파일로, 멤버 데이터는 관리되는 객체의 총 갯수, 첫번째 PTRLUMP 의 포인터, 마지막 PTRLIST 의 포인터 등을 있다. 멤버 함수는 객체를 추가, 삭제하는 함수, 인덱스를 기준으로 조회하는 함수, 임의의 객체가 리스트에서 관리되고 있는지를 조회하는 함수 등과 같이 다양하게 리

스트를 관리하는 함수들이 정의되어 있다. 특히, 인덱스를 기준으로 조회하는 경우에는 인덱스가 총 갯수의 절반보다 크면 뒤에서부터 추적을 하여 가능한 빠르게 조회할 수 있도록 하였다.

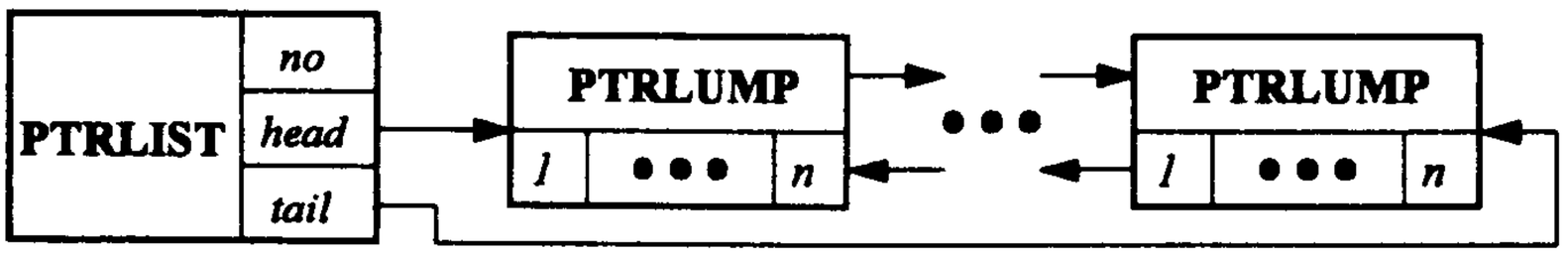


그림 3.2 Structure of PTRLIST class

```
// Declaration of class PTRLIST

template <class T>
class PTRLIST {
        protected:
                long no_obj_ptrs;     // The number of object-pointers in OBJLIST
                PTRLUMP<T> *head_ptrlump;      // The pointer to first PTRLUMP
                                                  in PTRLIST
                PTRLUMP<T> *tail_ptrlump;       // The pointer to last PTRLUMP
                                                  in PTRLIST
        public:
                // Constructors
                PTRLIST();
                PTRLIST( const PTRLIST<T> & );

                // Destructor
                ~PTRLIST();

                // Retrieving of the pointer to object   ======================
                // Get head ptrlump of list
                PTRLUMP<T> *head_lump();
                // Get first item in list
                T head();
                // Get last item in list
                T tail();
                // Get item at a given position( zero based index )
                T operator[](long) const;
                // Get next item
                T getnext( PTRLUMP<T> *&, long & ) const;
                // Get all items in list
                void getall( T *&, long & );
```

```
// Get status of list ==========================================
// Get the number of items in list
long count() const { return no_obj_ptrs; }
// Check whether list is empty or not
logical isempty() const
{ return (no_obj_ptrs == 0 ? TRUE : FALSE); }
// Find item which has the pointer to given object in list
long find( const T &) const;
// Check whether the item has the pointer to given object exists
logical exist( const T & ) const;

// Modifing of list ============================================
// Set a given object-pointer as last item in list
void addtail( const T & );
// Make a same list with a given list
PTRLIST<T> &operator=( const PTRLIST<T> & );
// Remove item at a given position
void removeat(long);
// Remove item which has the pointer to given object from list
void remove( const T & );
// Replace item on list as a given object-pointer at a given position
T replace( int, const T & );
// Remove all items from list
void removeall();
// Unite two PTRLISTs
void unite(const PTRLIST<T> &);
// Intersect two PTRLISTs
void intersect(const PTRLIST<T> &);
// Subtract PTRLISTs
void subtract(const PTRLIST<T> &);
// Remove all items with objects
void removeall_with_objects();
```

제 3 절 기본 곡선, 곡면 생성 기능

기본적인 곡선 기능에는 직선, 원, 원호, 자유곡선을 생성하는 기능과 평면, 실린더, 자유곡면을 생성하는 기능이 있다. 다음은 곡선과 곡면을 나누어 기능을 설명한 것이다.

1. 곡선(curve)

SurfART에서 곡선을 나타내기 위해서 그림3.3과 같이 곡선 클래스 구조를 정의하여 나타낸다. curve 클래스는 형상요소의 종류에 따라 직선을 나타내는 straight 클래스, 타원과 원을 나타내는 ellipse 클래스, 자유곡선을 나타내는 intcurve 클래스로 분류되고, 각 클래스는 curve class로부터 유도된 클래스이고, nurbcrv class를 멤버로 갖는다. 클래스의 헤더 파일은 APPENDIX B에 수록하였다.

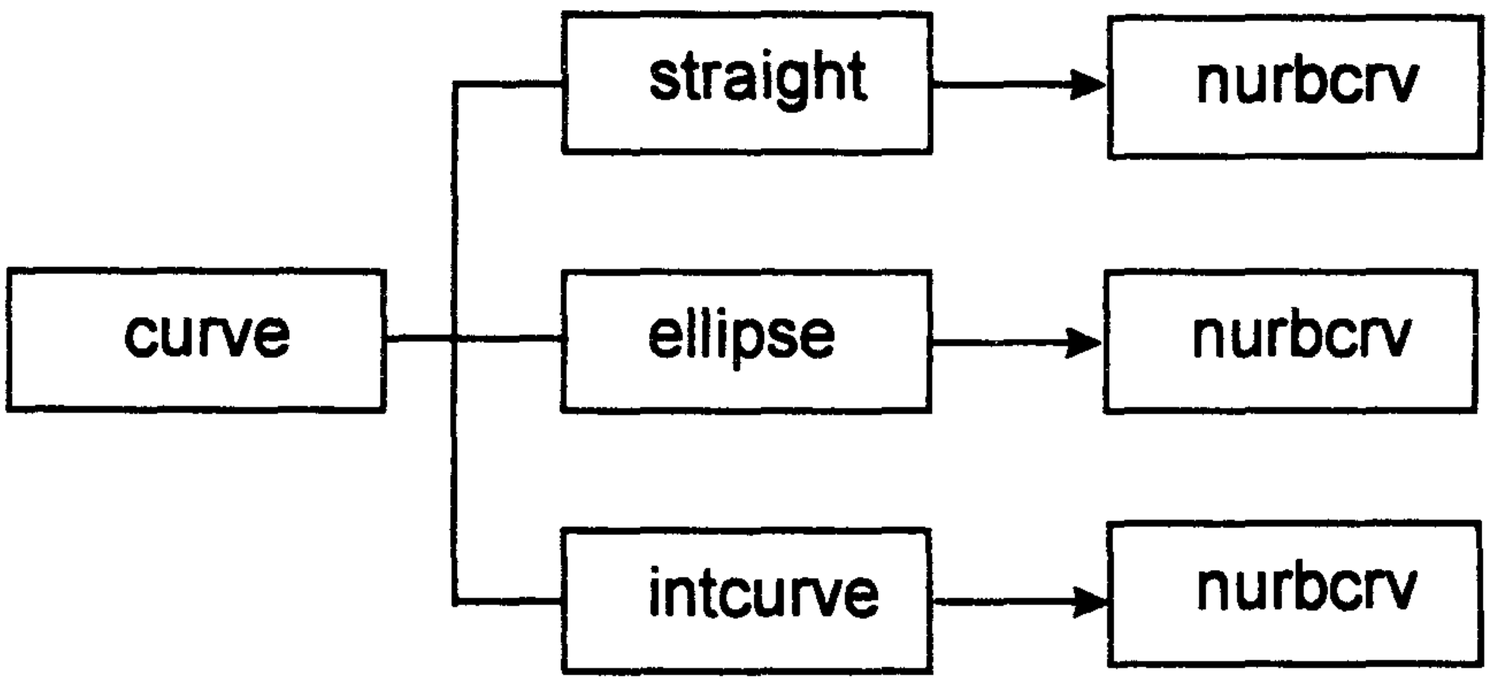


그림 3.3 곡선 클래스의 구조

가. 직선(Line)

SurfART 에서 직선은 straight class 로 정의된다. straight class 는 직선을 NURBS 로 표현하는 nurbcrv class, 시작점, 방향벡터 등을 member 로 갖고 있다. 따라서 straight 를 생성하기 위해서는 먼저 nurbcrv 를 생성해야 한다. 본 시스템에서 직선은 차수가 1, 조정점은 직선의 시작점과 끝점, knot vector 는 0, 0, 1, 1, weight 는 각 조정점에 대하여 1.0 인 NURBS 로 정의하여 nurbcrv 를 생성한다. 생성된 nurbcrv 와 nurbcrv 의 member 데이터로부터 구한 시작점, 방향벡터 등으로 straight 를 생성하여 리턴한다. SurfART 에서 사용자가 직선을 생성하는 방법은 직선의 시작점과 끝점의 좌표값을 입력하는 방법, 정의된 점들을 선택하는 방법, 시작점과 방향 벡터를 입력하는 방법, 정의된 직선을 선택하고 옵셋 벡터(offset vector)를 입력하여 옵셋 직선을 생성하는 방법 등이 있으며 각각은 APPENDIX B 와 같이 API 함수로 구현되어 있다.

나. 타원(ellipse)

SurfART 에서 원(circle)과 타원은 ellipse class 로 정의된다. 원과 타원은 동일한 알고리즘으로 구현했다. ellipse class 는 타원을 NURBS 로 표현하는 nurbcrv class, 타원의 중심점, 타원이 위치하는 평면의 법선벡터, major axis vector, minor axis vector, major axis 에 대한 minor axis 의 비를 나타내는 반지름 비를 member 로 갖고 있다. 따라서 ellipse 를 생성하기 위해서는 먼저 nurncrv 를 생성해야 한다. 본 시스템에서 타원은 차수가 2, 조정점은 식(3.1)과 같은 방법으로, knot vector 는 0, 0, 0, 0.25, 0.25, 0.5, 0.5, 0.75, 0.75, 1, 1, 1, weight 는 각 조정점에 대해 1.0, 0.701, 1.0, 0.701, 1.0, 0.701, 1.0, 0.701, 1.0 인 NURBS 로 정의하여 nurbcrv 를 생성한다. 생성된 nurbcrv 와 nurbcrv 의 member 데이터로부터 구한 타원의

중심점, major axis vector, minor axis vector, 반지름비로 타원을 생성하여 리턴한다. 원의 경우도 타원과 동일한 방법으로 구하며 다른 점은 반지름비가 1.0 이라는 것이다. 타원의 조정점은 $C_0, C_1,...C_8$로 나타내고, Q는 타원의 중심, r_1은 major radius, r_2는 minor radius, U는 major axis vector, V는 minor axis vector를 각각 나타낸다. 그림 3.4는 타원의 조정점을 나타내었고, 식(3.1)은 타원의 조정점을 계산하는 식이다[3-1].

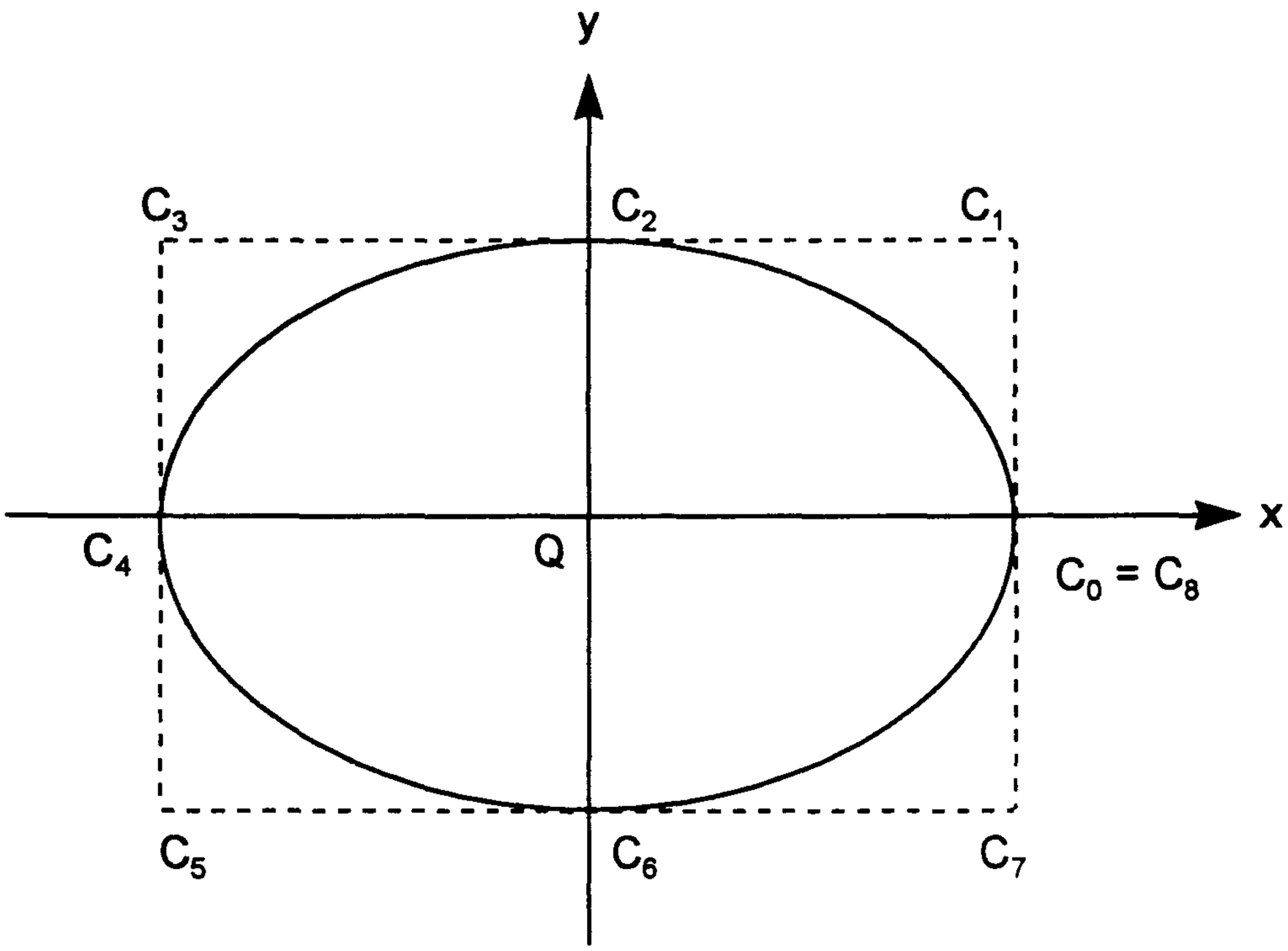


그림 3.4 타원의 조정점

$$C_0 = Q + r_1 * U$$

$$C_1 = C0 + r_2 * V$$

$$C_2 = Q + r_2 * V$$

$$C_3 = C_2 - r_1 * U$$

$$C_4 = Q - r_1 * U$$

$$C_5 = C_4 - r_2 * V$$

$$C_6 = Q - r_2 * V$$

$$C_7 = C_6 + r_1 * U$$

$$C_8 = Q + r_1 * U \qquad (3.1)$$

SurfART 에서 사용자가 원을 생성하는 방법은 원의 위치할 평면을 결정하고, 원의 중심과 반지름을 입력하는 방법, 세 점을 입력하는 방법, 정의된 세 점을 선택하는 방법등이 있으며 각각은 APPENDIX B 와 같이 API 함수로 구현되어 있다.

다. 원호(arc)

SurfART 에서 원호는 ellipse class 로 정의된다. ellipse class 는 원을 NURBS 로 표현하는 nurbcrv class, 원의 중심점, 원의 위치하는 평면의 법선벡터, major axis vector, minor axis vector, major axis 에 대한 minor axis 의 비를 나타내는 반지름 비를 member 로 갖고 있다. 따라서 ellipse 를 생성하기 위해서는 먼저 nurncrv 를 생성해야 한다. 본 시스템에서 원호는 차수가 2, 조정점, knot vector, weight 는 원호의 각도에 따라 계산되어 NURBS 로 정의되어 nurbcrv 를 생성한다. 생성된 nurbcrv 와 member 데이터로부터 구한 원호의 중심점, major axis

vector, minor axis vector, 반지름비로 ellipse를 생성하여 리턴한다. SurfART에서 사용자가 원호를 생성하는 방법은 원호가 위치할 평면, 원호의 중심, 시작점, 반지름, 각도를 다이얼로그 박스에서 입력하는 방법, 원호가 위치할 평면, 중심, 시작점, 끝점, 반지름을 다이얼로그 박스에서 입력하는 방법등이 있으며 각각은 APPENDIX B와 같이 API 함수로 구현되어 있다.

라. 자유곡선

자유곡선을 NURBS로 보간하는 알고리즘은 자유곡면에서 언급되므로 여기에서는 생략한다.

2. 곡면(surface)

SurfART에서 곡면을 나타내기 위해서 그림3.5와 같이 곡면 클래스 구조를 정의하여 나타낸다. surface class는 형상요소의 종류에 따라 plane, cone, sphere, torus, spline class로 분류되고, 각 클래스는 surface class로부터 유도된 파생 클래스이고, nurbsuf class를 멤버로 갖는다. 클래스의 헤더 파일은 APPENDIX B에 수록하였다.

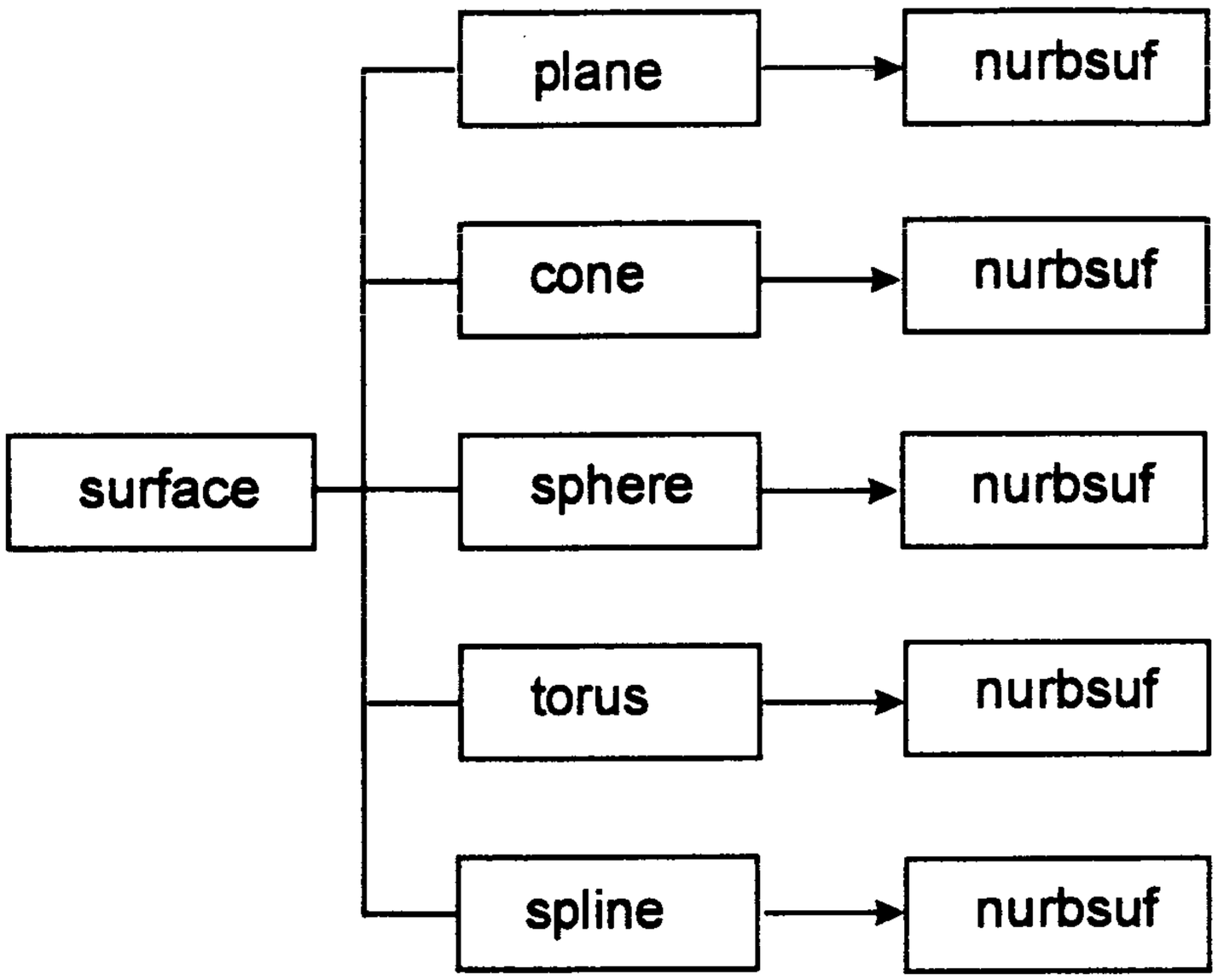


그림 3.5 곡면 클래스 구조

가. 평면(planar surface)

SurfART 에서 평면은 plane class 로 정의된다. plane class 는 평면을 NURBS 로 표현하는 nurbsuf class, 평면상의 한 점, 평면의 법선벡터등을 member 로 갖고 있다. 따라서 plane class 를 생성하기 위해서는 먼저 nurbsuf 를 생성해야 한다. 본 시스템에서 평면은 u,v 방향 차수는 1 이고, 조정점은 평면의 경계를 이루는 점이고, u 방향 knot vector 는 0, 0, 1, 1, v 방향 knot vector 는 0, 0, 1, 1, weight 는 각 조정점에 대하여 1.0 인 NURBS 로 정의하여 nurbsuf 를 생성한다. 생성된 nurbsuf 와 nurbsuf 의 member 데이터로부터 구한 평면상의 한 점, 평면의 법선벡터등으로 plane 을 생성하여 리턴한다. SurfART 에서 사용자가 평면을 생성하는 방법은 평면의 중심점, 법선벡터(normal vector), 평면의 u,v 크기를 다이얼로그 박스에서 입력하는 방법, 세 점을 다이얼로그 박스에서 입력하는 방법, 정의된 세 점을 선택하는 방법등이 있으며 각각은 APPENDIX B 와 같이

API 함수로 구현되어 있다. 그림 3.6 에서 평면의 중심점(cen), 법선 벡터(norm), 평면의 u 방향크기(u_size), v 방향크기(v_size)를 입력으로 하여 평면을 정의하였고, 평면의 조정점(C0, C1, C2, C3)을 구하는 알고리즘은 그림 3.7 과 같이 나타낸다.

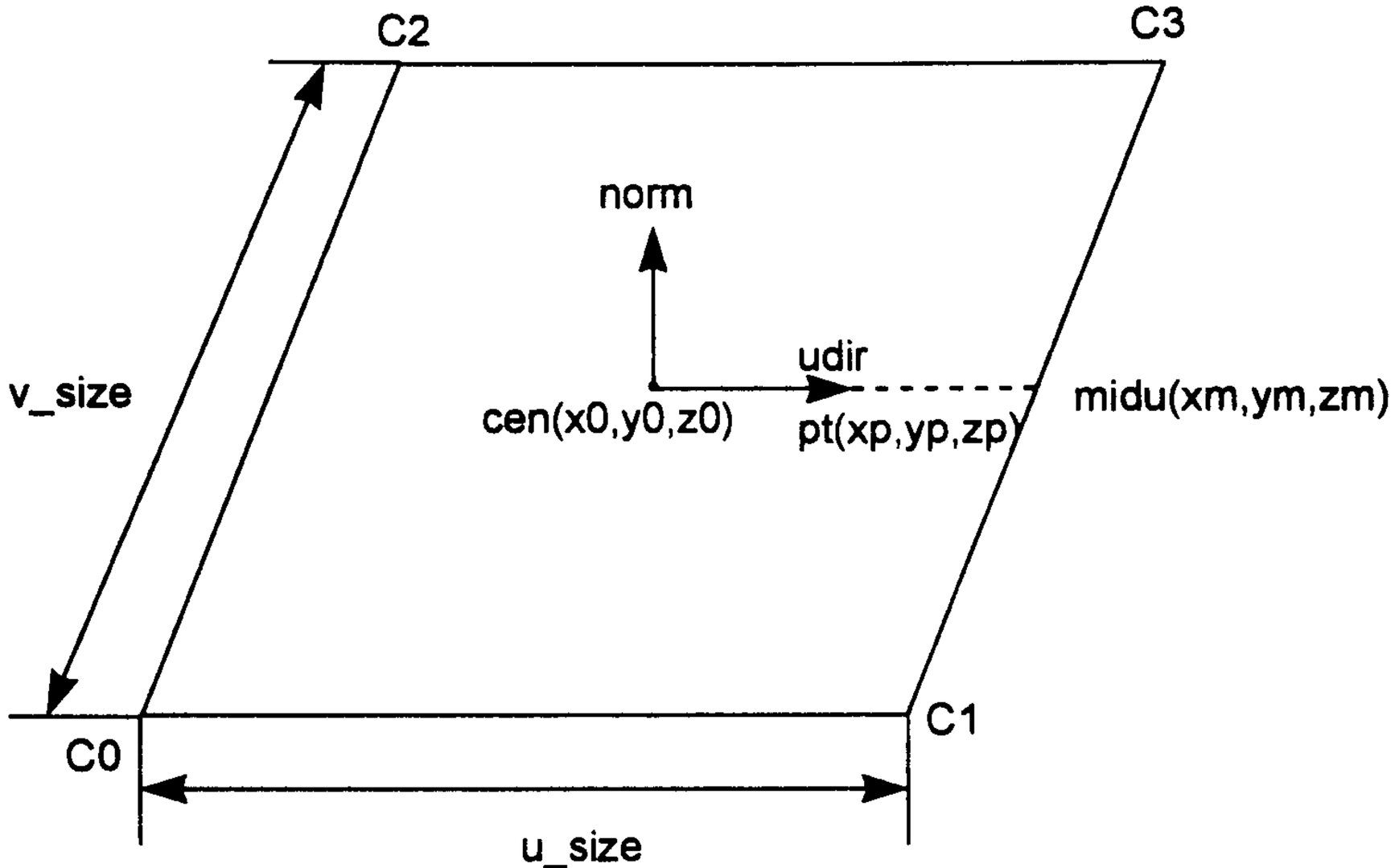


그림 3.6 평면의 조정점

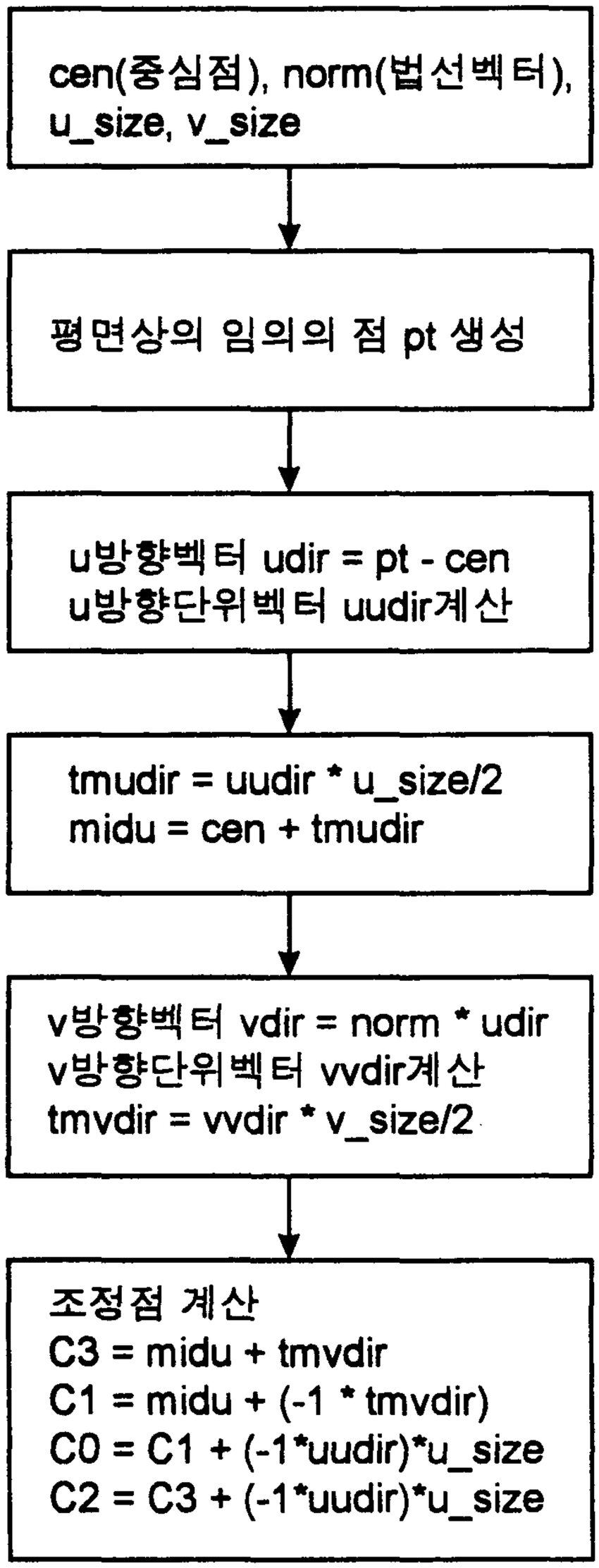


그림 3.7 평면을 조정점을 구하는 알고리즘

정의된 세 점을 입력 또는 선택하여 평면을 생성하는 알고리즘은 다음과 같다. 그림 3.6 에서 pos1, pos2, pos3 는 입력 또는 선택되는 세 점이고, udir, vdir 은 u, v 방향벡터이고, norm 은 평면의 법선벡터를 나타낸다. 그림 3.8 의 세 점을 이용한 알고리즘에 의해 평면의 중심점(cen), 평면의 법선벡터(norm), u, v 방향의 크기(u_size, v_size)가 구해지므로 그 이후의 과정은 그림 3.7 의 평면의 조정점을 구하는 알고리즘을 따른다.

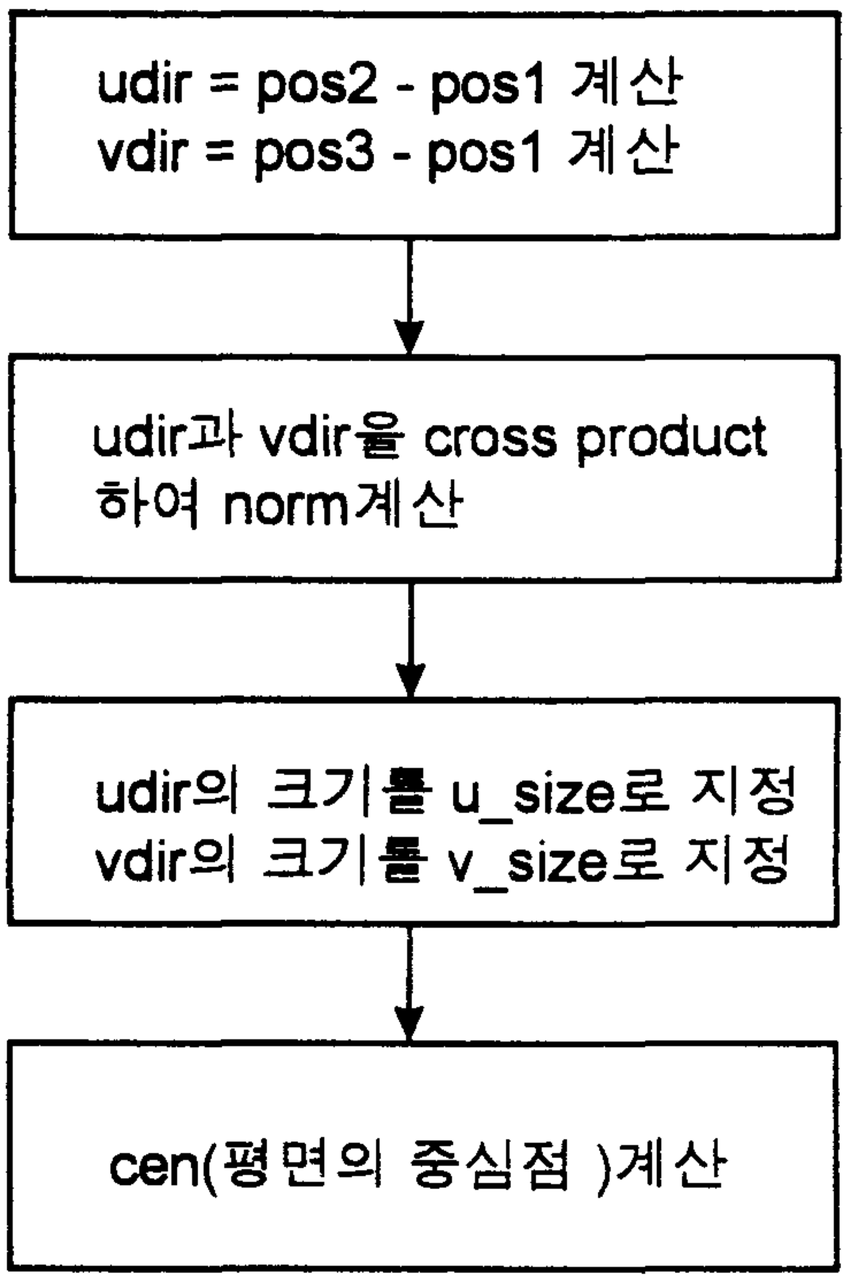


그림 3.8 세 점을 이용한 평면의 중심점, 법선, u v 크기 계산

나. 원통면(cylindrical surface)

SurfART에서 원통면은 cone class로 정의된다. cone class는 NURBS로 표현하는 nurbsuf class, base면의 ellipse class, top면의 ellipse class등을 member로

갖고 있다. 따라서 cone class를 생성하기 위해서는 먼저 nurbsuf를 생성해야 한다. 본 시스템에서 원통면은 u방향 차수는 2, v방향 차수는 1, 조정점은 base, top면의 ellipse의 조정점이고 , knot vector는 0.0, 0.0, 0.0, 0.25, 0.25, 0.5, 0.5, 0.75, 0.75, 1.0, 1.0, 1.0, weight는 각 조정점에 대하여 1.0, 0.701, 1.0, 0.701, 1.0, 0.701, 1.0, 0.701, 1.0, 1.0, 0.701, 1.0, 0.701, 1.0, 0.701, 1.0, 0.701, 1.0인 NURBS로 정의하여 nurbsuf를 생성한다. 생성된 nurbsuf와 원통면의 base와 top의 ellipse등으로 생성하여 리턴한다. SurfART에서 사용자가 원통면을 생성하는 방법은 원통면의 base와 top의 경계를 정의하기 위한 데이터를 다이얼로그 박스에서 입력하는 방법, base와 top의 경계를 이루는 타원을 선택하는 방법등이 있으며 각각은 APPENDIX B와 같이 API함수로 구현되어 있다.

다. 자유 곡면

일반적으로 그림 3.9와 같이 가로, 세로 배열을 이룬 점 군 데이타를 B-spline 곡면으로 보간하는 알고리즘은 가로방향(u방향)의 점 데이타들을 지나는 B-spline 곡선을 생성하여 조정점을 구하고, 얻어진 조정점들을 세로 방향(v방향)으로 지나는 B-spline곡선을 생성하여 조정점을 구하는 방법을 사용한다. B-spline 곡면을 보간하는 알고리즘[3-2]을 요약하면 다음과 같으며 그림 3.10은 이들을 구현한 흐름도이다.

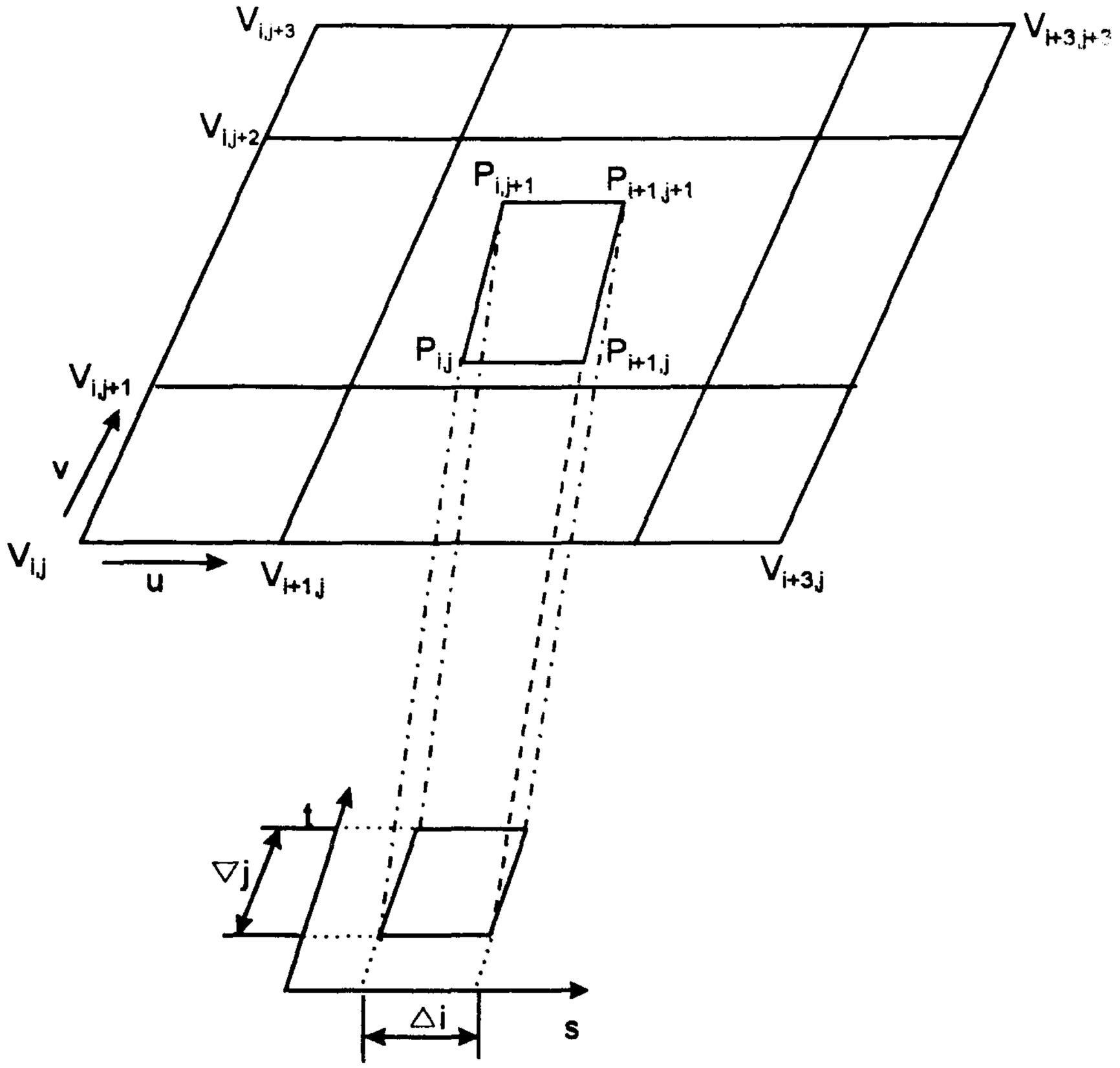


그림 3.9 3차 B-spline곡면의 정의

(1) 가로, 세로 방향의 knot span 결정

B-spline 곡면을 보간하기 위해서는 가로 (u),세로 (v) 방향의 knot span 또는 knot 벡터가 필요하다. 일반적으로 가로방향의 knot span을 { △ij, ; i=0,1,...,m+1, j=0,1,...,n},세로 방향의 knot span 벡터를 {▽ij ; i=0,1,...,m, j=0,1,...,n+1}이라 하며, 각각은 다음과 같이 점 데이타 간의 거리 (chord-length)에 비례하도록 하며, extended knot span 은 0으로 하여 knot의 중첩도가 차수와

같게 한다.

$$\Delta_{ij} = \left| P_{i+1j} - P_{ij} \right| \qquad i = 0,1,\ldots,m-1, \quad j = 0,1,\ldots,n$$

$$\nabla_{ij} = \left| P_{ij+1} - P_{ij} \right| \qquad i = 0,1,\ldots,m, \quad j = 0,1,\ldots,n-1$$

(2) 곡면의 end 접선벡터 추정

곡면의 end 접선벡터 $\{\hat{S}_{0j}, \hat{S}_{mj}, \hat{t}_{i0}, \hat{t}_{in} \; ; i = 0,1,\ldots,m, j = 0,1,\ldots,n\}$가 입력되지 않았으면, circular end condition, polynomial end condition, free end condition 등의 3가지 방법으로 접선벡터를 추정할 수 있으며, 일반적으로 곡면 보간에서는 free end condition을 많이 사용하지만 개발된 알고리즘에서는 사용자가 선택할 수 있도록 하였다.

(3) 가로방향의 조정점 결정

B-spline 곡면 보간시 가로방향 곡선의 조정점을 세로방향으로 보간할 때 이용하므로 이를 intermediate 조정점(intermediate control points) {Cij ; i=0,1,...,m+2, j=0,1,...n}이라 하고 B-spline 곡선을 보간하는 식과 접선벡터 $\{\hat{S}_{0j}, \hat{S}_{mj} \; j = 0,1,\ldots,n\}$을 이용하면 구할 수 있다.

(4) boundary 벡터의 추정

(3)에서 구한 조정점을 세로 (v) 방향으로 보간하기 위해서는 조정점 상의 시작점 접선벡터 d_i와 끝점의 접선 벡터 e_i가 필요하며 (2)에서 구한 접선벡터 $\{\hat{t}_{i0}, \hat{t}_{in} \; ; i = 0,1,\ldots,m\}$를 구할 수 있다.

(5) B-spline 곡면의 조정점 결정

(4)에서 구한 intermediate 조정점을 세로(v) 방향으로 지나는 B-spline 곡선의 end 접선벡터를 boundary 벡터 d_i, e_i로 대치하여 보간하면 (m+3)×(n+3) 개의 조정점을 구할 수 있다. 이 조정점들이 (m+1)×(n+1) 개의 점 데이타를 지나는 B-spline 곡면의 조정 V_{ij}가 된다.

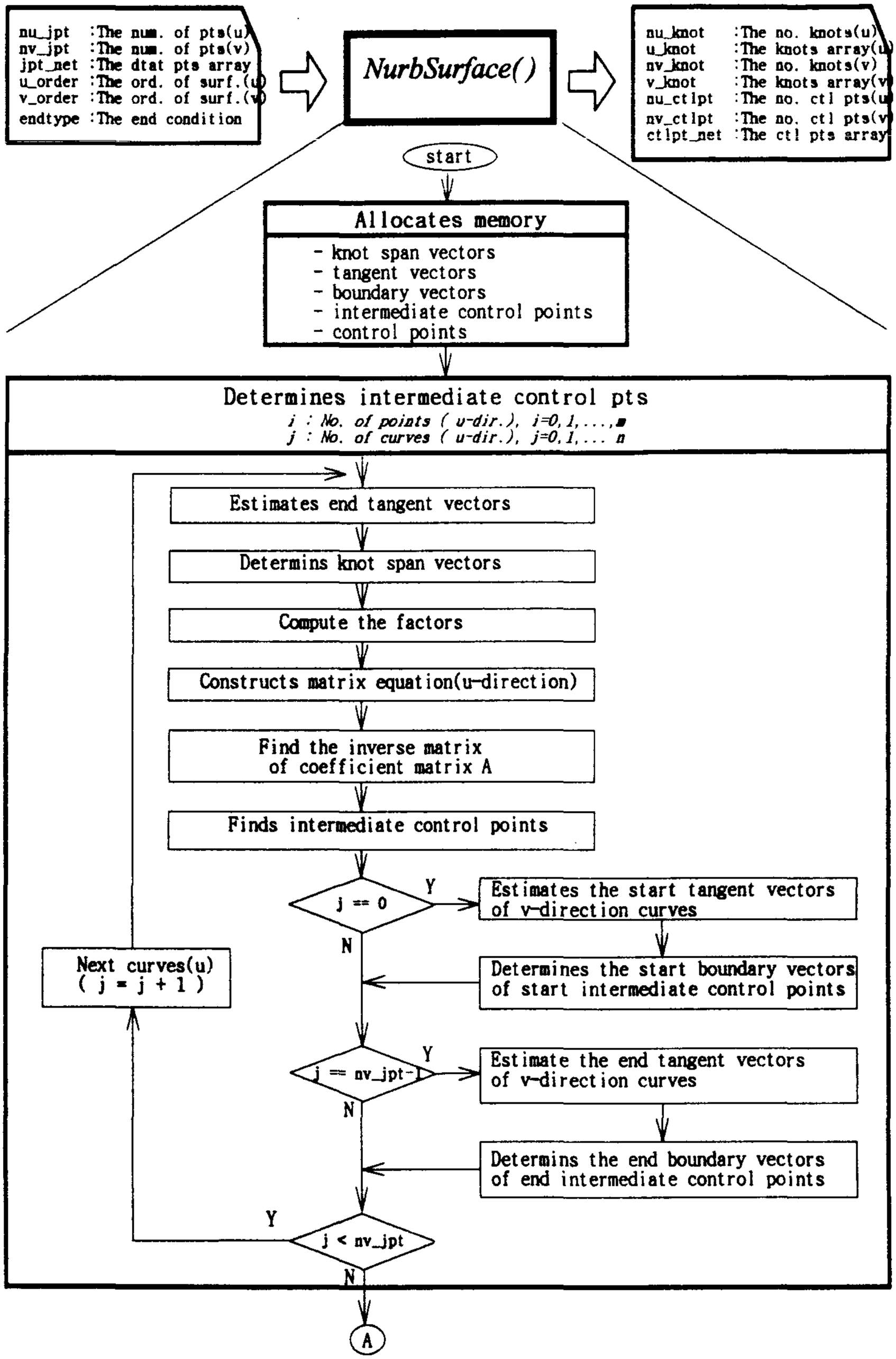


그림 3.10 B-spline 곡면 보간 알고리즘

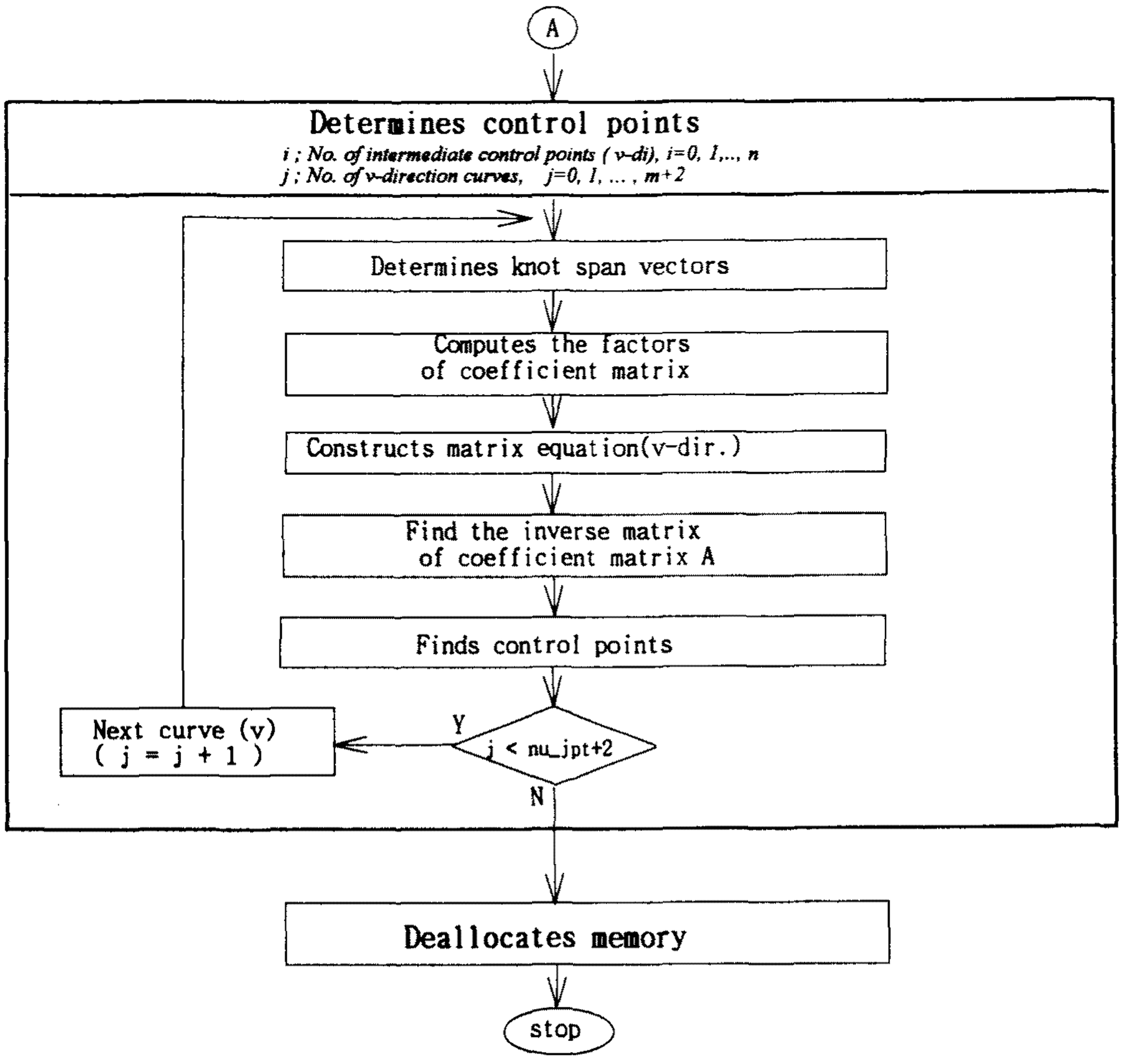


그림 3.10 B-spline 곡면 보간 알고리즘(계속)

제 4 절 형상요소의 변환 기능

형상요소를 변환하는 기능으로는 형상요소를 직선으로 움직이는 이동변환(translation), 형상요소를 회전시키는 회전변환(rotation), 형상요소의 크기를 변화시키는 크기변환(scaling)이 있다. 각각의 변환은 형상요소를 복사(copy)하는 방법과 이동(move)하는 방법으로 나누어 진다. 다음은 각각의 변환을 설명한 것이다.

1. 이동변환(translation)

형상요소를 이동변환 하는 방법은 형상요소(점, 점열, 곡선, 곡면)를 선택하고 다이얼로그 박스에서 이동변환 벡터를 입력하여 변환시킨다. 식(3.2)에서 이동변환 한 점(x_t, y_t, z_t, 1)은 원래의 점(x_o, y_o, z_o, 1)과 이동변환행렬을 곱하여 구해진다. 식 3.2 에서 Tx, Ty, Tz 는 각 방향의 이동량을 나타낸다.

$$(x_t \quad y_t \quad z_t \quad 1) = (x_0 \quad y_o \quad z_o \quad 1) \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ Tx & Ty & Tz & 1 \end{bmatrix} \quad (3.2)$$

이동변환하는 형상요소에는 점, 점열, 곡선, 곡면이 있다. 각각의 이동변환방법을 살펴보면 다음과 같다. 점의 이동변환은 위의 방법에 의해 수행되고, 점열의 이동변환은 점열이 많은 점의 집합이므로 각각의 점을 이동변환하면 된다. 곡선과 곡면의 이동변환은 곡선과 곡면이 NURBS 로 표현되어 있으므로 곡선과 곡면의 조정점을 이동변환하면 곡선과 곡면이 이동변환된다. 회전변환과 크기변환도 이동변환과 같은 방법으로 구현하였다.

2. 회전변환(rotation)

형상요소를 회전변환 하는 방법은 형상요소(점, 점열, 직선, 곡면)를 선택하고 다이얼로그 박스에서 회전변환할 형상요소를 회전축과 회전각도를 입력하여 변환시킨다. 식(3.3)에서 회전변환 한 점(x_t, y_t, z_t, 1)은 원래의 점(x_o, y_o, z_o, 1)과 회전변환행렬을 곱하여 구해진다. 식(3.4)는 회전변환행렬(Rot)[3-3]을 나타낸 것이다.

$$(x_t \quad y_t \quad z_t \quad 1) = (x_0 \quad y_o \quad z_o \quad 1)\ \mathbf{Rot} \tag{3.3}$$

$$\mathbf{Rot}=\begin{bmatrix} n^2_x+(1-n^2_x)\cos\theta & n_xn_y(1-\cos\theta)+n_z\sin\theta & n_xn_z(1-\cos\theta)-n_y\sin\theta & 0 \\ n_xn_y(1-\cos\theta)-n_z\sin\theta & n^2_y+(1-n^2_y)\cos\theta & n_yn_z(1-\cos\theta)+n_x\sin\theta & 0 \\ n_xn_z(1-\cos\theta)+n_y\sin\theta & n_yn_z(1-\cos\theta)-n_x\sin\theta & n^2_z+(1-n^2_z)\cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.4}$$

3. 크기변환(scaling)

형상요소를 크기변환 하는 방법은 형상요소(점, 점열, 직선, 곡면)를 선택하고, x, y, z factor 각각을 입력하는 local scaling 과 전체적인 factor 를 입력하는 overall scaling 방법에 의해 변환시킨다. 식(3.5)와 식(3.6)에서 크기변환 한 점(x_t, y_t, z_t, 1)은 원래의 점(x_o, y_o, z_o, 1)과 크기변환행렬을 곱하여 구해진다. 식(3.5)는 local scaling 을 나타내고 Sx, Sy, Sz 는 scaling factor 이다. 식(3.6)은 overall scaling 을 나타내고 So 는 scaling factor 이다.

$$(x_t \quad y_t \quad z_t \quad 1) = (x_0 \quad y_o \quad z_o \quad 1) \begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.5}$$

$$(x_t \quad y_t \quad z_t \quad 1) = (x_0 \quad y_o \quad z_o \quad 1) \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & So \end{bmatrix} \tag{3.6}$$

제 5 절 형상 display 기능

1. Window NT와 OpenGL의 interface 방법

윈도즈 NT에서는 운영체제 차원에서 OpenGL의 엔진이 기본으로 탑재되어 있다. 이러한 기능은 윈도즈 NT가 웍스테이션 시장을 잠식하는데 필수적이라는 생각에 기인한 것이다. 또한 이것을 바탕으로 마이크로소프트사 측에서는 3차원 그래픽 소프트웨어의 제작에도 심혈을 기울이고 있는 것으로 알고 있다. 그러면 윈도즈 NT에서 OpenGL은 어떻게 구성되었는지 살펴보면 다음 그림과 같다.

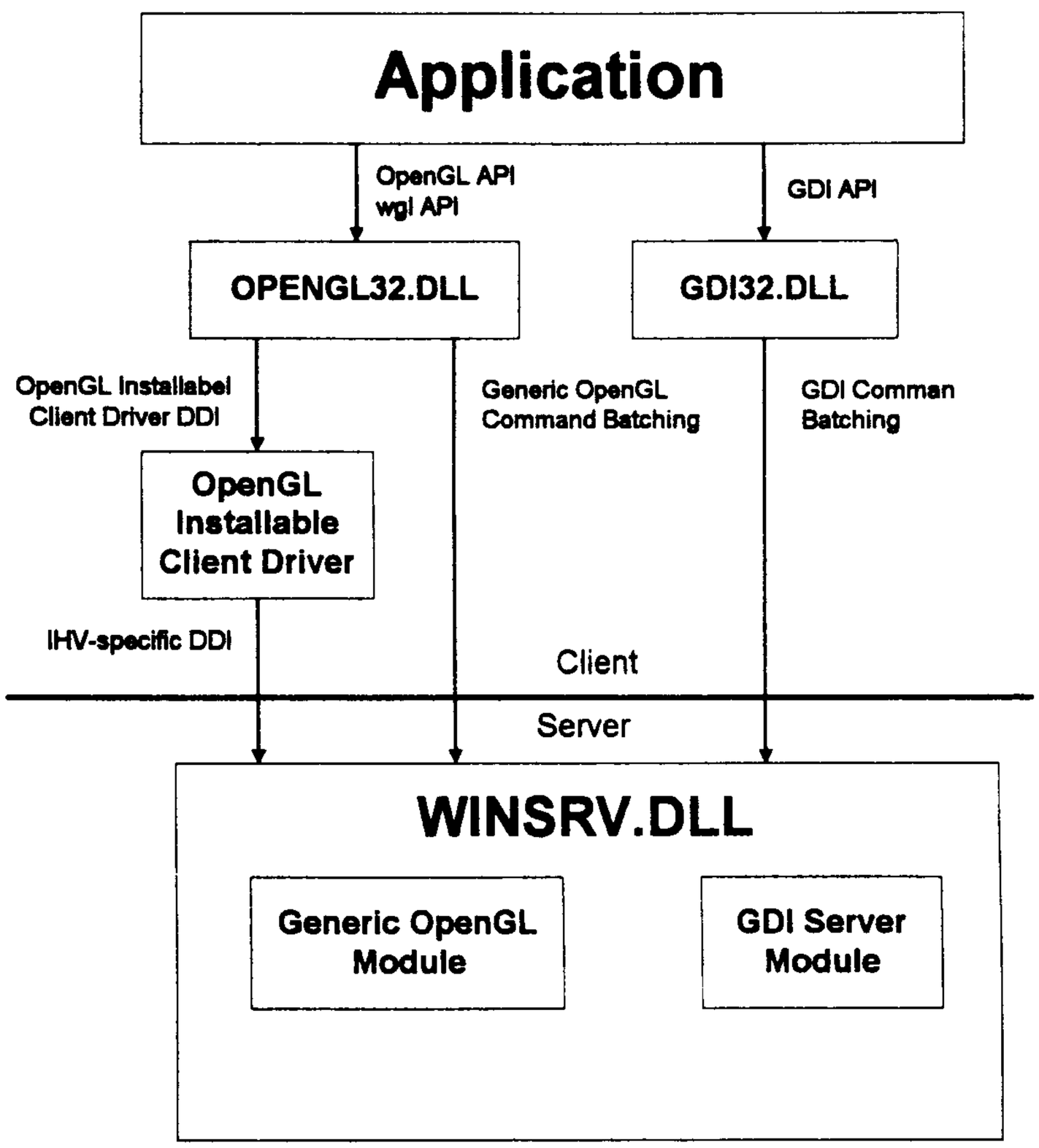


그림 3.11 WindowNT에서의 OpenGL 구조

이것을 보면 GDI32와 OPENGL32모듈로 나누어져 있는 것을 알 수 있다. 그래서 DoubleBuffer모드를 사용하는 OpenGL의 경우 BackBuffer와 FrontBuffer 모두 사용할 수 있지만 GDI의 경우 FrontBuffer만 사용할 수 있는 등 몇 가지 제한 사항들을 가지고 있다.

Window NT에서 OpenGL을 사용할 수 있도록 interface를 지정하는 과정을 살펴보면 다음과 같다.

① PIXELFORMATEDESCRIPTOR 구조의 값들을 지정

PIXELFORMATDESCRIPTOR pixelDesc;

② ChoosePixelFormat을 이용하여 가장 비슷한 픽셀 포맷에 관한 인덱스를 선택

③ SetPixelFormat을 이용하여 픽셀 포맷 결정

④ wglCreateContext를 이용하여 HGLRC를 만든다.

⑤ wglMakeCurrent를 이용하여 현재 컨텍스트를 지정한다.

"SurfART"에서는 OpenGL을 사용하기 위해서 COpenGL 클래스를 사용한다. COpenGL 클래스는 OpenGL과 Windows NT 시스템을 잘 연결하여 그래픽 이미지가 NT의 윈도우에 잘 display되도록 해준다. copengl.h와 copengl.cpp화일에 정의되어 있다. 다음은 surfavw.cpp화일에 정의된 COpenGL 클래스의 사용 예이다.

```
m_glptr = new COpenGL(hdc);
int nNewIdx;

// Choose Pixel Format & Create OpenGL Rendering Context
memset(&m_glptr->m_PixFmtDesc, 0, sizeof(PIXELFORMATDESCRIPTOR));

// size of this pfd
m_glptr->m_PixFmtDesc.nSize = sizeof(PIXELFORMATDESCRIPTOR);
m_glptr->m_PixFmtDesc.nVersion = 1;
```

```
// version
numberm_glptr->m_PixFmtDesc.dwFlags = PFD_DRAW_TO_WINDOW |
// support window
PFD_SUPPORT_OPENGL |        // support OpenGL
PFD_DOUBLEBUFFER;           // double buffered
m_glptr->m_PixFmtDesc.iPixelType = PFD_TYPE_RGBA;      // RGBA type
m_glptr->m_PixFmtDesc.cColorBits = 24;                  // 24-bit color
depthm_glptr->m_PixFmtDesc.cRedBits = 0;              // color bits ignored
m_glptr->m_PixFmtDesc.cRedShift = 0;             // color bits ignored
m_glptr->m_PixFmtDesc.cGreenBits = 0;      // color bits ignored
m_glptr->m_PixFmtDesc.cGreenShift = 0;     // color bits ignored
m_glptr->m_PixFmtDesc.cBlueBits = 0;       // color bits ignored
m_glptr->m_PixFmtDesc.cBlueShift = 0;      // color bits ignored
m_glptr->m_PixFmtDesc.cAlphaBits = 0;      // no alpha buffer
m_glptr->m_PixFmtDesc.cAlphaShift = 0;     // shift bit ignored
m_glptr->m_PixFmtDesc.cAccumBits = 0;      // no accumulation buffer
m_glptr->m_PixFmtDesc.cAccumRedBits = 0;   // accum bits ignored
m_glptr->m_PixFmtDesc.cAccumGreenBits = 0;// accum bits ignored
m_glptr->m_PixFmtDesc.cAccumBlueBits = 0; // accum bits ignored
m_glptr->m_PixFmtDesc.cAccumAlphaBits = 0;// accum bits ignored
m_glptr->m_PixFmtDesc.cDepthBits = 32;     // 32-bit z-buffer
m_glptr->m_PixFmtDesc.cStencilBits = 0;    // no stencil buffer
m_glptr->m_PixFmtDesc.cAuxBuffers = 0;     // no auxiliary buffer
m_glptr->m_PixFmtDesc.iLayerType = PFD_MAIN_PLANE; // main layer
m_glptr->m_PixFmtDesc.bReserved = 0;       // reserved
m_glptr->m_PixFmtDesc.dwLayerMask = 0;     // layer masks ignored
m_glptr->m_PixFmtDesc.dwVisibleMask = 0;   // layer masks ignored
m_glptr->m_PixFmtDesc.dwDamageMask = 0;    // layer masks ignored

nNewIdx=m_glptr->ChoosePixelFormat(m_glptr->m_hdc,                &m_glptr-
>m_PixFmtDesc);
if((m_glptr->IsNativeIndex(m_glptr->m_hdc,nNewIdx))||(m_glptr->
IsDeviceIndex (m_glptr->m_hdc, nNewIdx)))
m_glptr->GetGLRC(m_glptr->m_hdc);
```

m_glptr은 COpenGL클래스 오브젝트로서 view 클래스에 정의되고 CSurfartView::OnCreate() 함수에서 생성된다. OnCreate() 함수는 view 윈도우가 생성될 때 불려지는 함수로 view 윈도우가 생성될 때 view 윈도우와 OpenGL 윈도우의 Pixel Format을 결정해 주어야 한다. m_glptr->m_PixFmtDesc는 Pixel Format에 대한 정보를 저장하고 있는 구조로서 ChoosePixelFormat()에 의해 적

절한 Pixel Format이 선택되며 GetGLRC()에 의해 view 윈도우와 OpenGL 윈도우의 데이타 교환을 위한 작업이 마무리된다.

2. 형상 DISPLAY

정의된 형상을 화면에 display하기 위해서는 시스템 화면의 그래픽 영역을 초기화 해야 한다. 초기화 작업은 하나의 색으로 그래픽 영역을 칠하는 것이다. 윈도우의 그래픽 영역을 칠하는 기능은 glClear()라는 함수에서 담당하고 있다. 이 함수는 특정한 버퍼를 특정 내용으로 초기화 시켜주는 함수이다. 이 과정에서 먼저 알아두어야 할 사항은 OpenGL에서는 여러가지의 버퍼를 가지고 있다는 것이다. OpenGL이 가지는 있는 중요한 두 가지 버퍼는 다음과 같다.

① 칼라 버퍼 : 픽셀의 색을 저장하는 버퍼로서, 8비트에서 32비트까지 다양하다.

② 깊이 버퍼 : 픽셀의 깊이를 나타내는 것으로 숨은 선이나 면의 제거에 필수적인 버퍼로 화면에 Light 효과를 주는 경우 반드시 있어야 한다.

화면을 지우고자 하는 경우는 칼라 버퍼를 지워야 한다. 이 경우 화면 전체를 지우는 필요한 배경 색상은 glClearColor()를 이용해 지정한다.

이제 화면을 지우고 나면 화면에 그릴 그림을 위하여 색을 지정하여야 한다. 색을 지정하는 것은 glColor()를 이용하게 된다. 이 함수의 인수는 인수의 데이타형에 따라서 정수인 경우는 0에서 255까지, 실수인 경우는 0.0에서 1.0까지의 값을 사용하게 된다.

화면에 그림을 그리는 기능을 살펴보면 먼저 그리고자 하는 도형의

종류를 지정하고 그 도형을 그리는데 필요한 꼭지점을 넘겨줌으로써 그림을 그리는 형식으로 되어 있다.

OpenGL에서 point를 display할 때는 아래와 같은 함수를 쓴다.

```
glBegin(GL_POINTS);
// 꼭지점 지정이 시작됨을 나타내는 것으로 꼭지점들이 만들어야 할 도형의 종류를 인수로 넘겨 준다.
  glVertex3f(0.0, 0.0, 0.0);
  glVertex3f(1.0, 0.0, 0.0);
  glVertex3f(0.0, 2.0, 3.0);
glEnd();  // 도형의 정의가 끝이 났음을 알린다.
```

OpenGL에서는 glBegin()과 glEnd()로 여러가지 Geometric Primitive를 그릴 수 있다. Geometric Primitive의 종류와 타입은 다음과 같다.

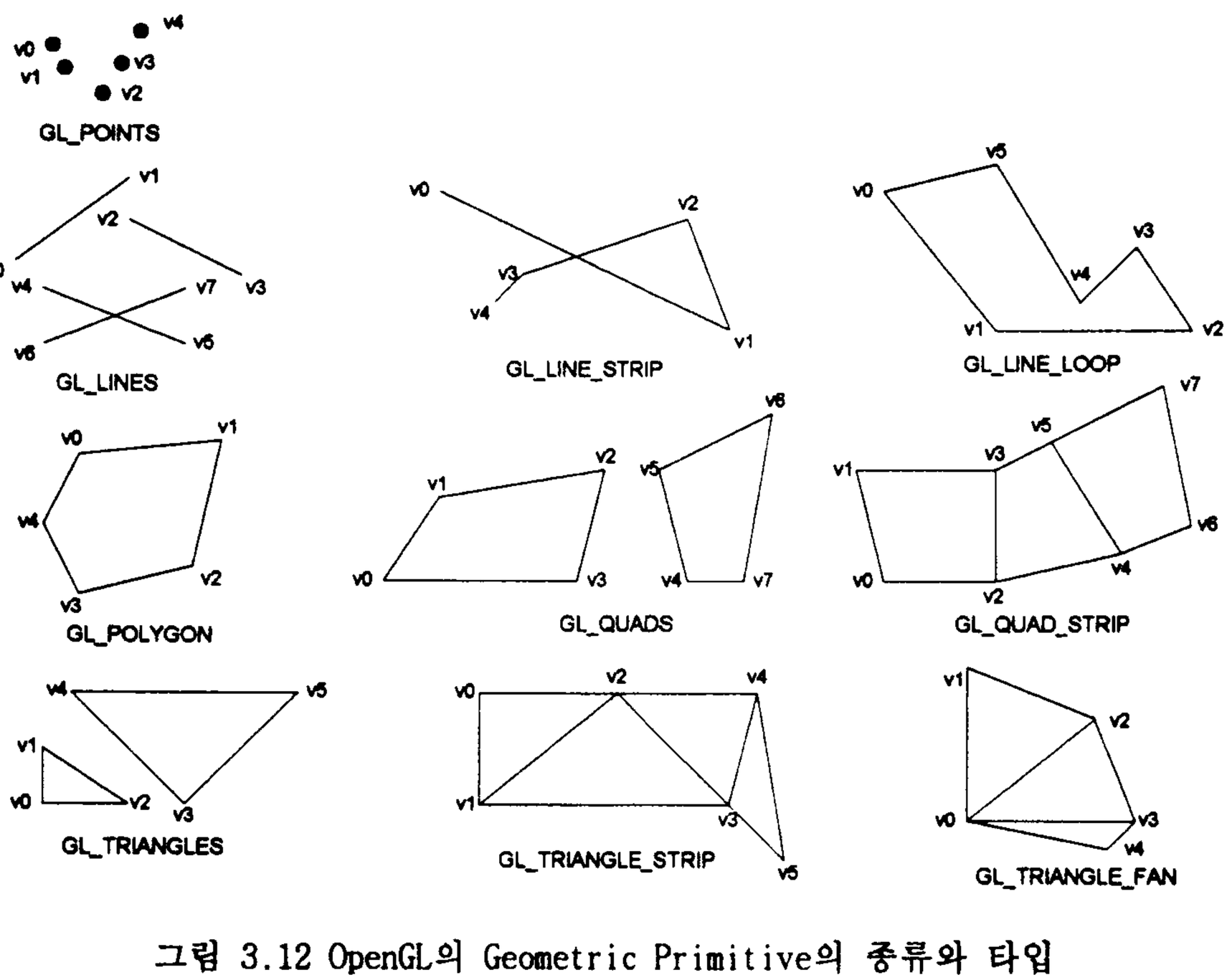


그림 3.12 OpenGL의 Geometric Primitive의 종류와 타입

Curve, Surface를 그리기 위해서는 짧은 선이나 작은 폴리곤 영역들의 연속적인 모임으로 표현이 가능하다. 그러나 실제적인 작업을 할 때에는 선이나 폴리곤의 세분화가 스크린 픽셀크기보다 더 작아야 하므로 위와 같은 방법으로는 표현이 힘들다. OpenGL에서 이러한 문제점을 해결하고 보다 간편하게 Curve, Surface를 표현할 수 있도록 GLU NURBS(Non-Uniform Rational B-Spline) interface를 제공하고 있다. 이를 사용하면 NURBS로 정의된 곡선, 곡면을 쉽게 그릴 수 있다. 또한 그 외에 간단한 NURBS로 정의되지 않은 해석곡면(spheres, cylinders, disks 등)은 gluCylinder, gluDisk, gluPartialDisk, gluSphere 함수[3-4]를 이용하면 된다. 위와 같은 quadric object등을 관리하도록 gluNewQuadric, gluDeleteQuadric, gluQuadricCallback 함수등을 제공하고 있다.

"SurfART" 시스템에서 정의된 모델을 화면에 디스플레이 하는 기능은 위상 요소를 이용하여 디스플레이 하는 방법과 형상요소를 이용하여 디스플레이하는 방법이 있다. 위상요소를 이용하여 디스플레이 하는 함수는 형상요소를 이용하여 디스플레이하는 함수를 호출하여 최종적으로 형상요소를 화면에 그려 사용자가 확인할 수 있도록 한다. 형상요소를 디스플레이 하는 함수는 OpenGL을 이용하여 점, 곡선, 곡면의 모양을 화면에 그리는 기능을 수행하며 DrawPoint(), DrawCurve(), DrawSurface() 등이 있다. 그러나 DrawSurface로는 트림 곡면(trimmed Surface)을 그릴 수 없으므로, OpenGL의 트림 기능을 이용하여 트림된 곡면을 그리는 DrawFace()를 구현하였다.

가. DrawPoint

DrawPoint() 함수는 GPOINT에 대한 포인터와 point 색깔을 지정해 주는 변수를 매개변수로 받아 point를 그려주는 기능을 한다. 앞서 설명한 OpenGL을 이용한 도형을 그리는 방법을 이용하여 그려주면 된다. 즉,

glBegin(), glEnd() 사이에 표현하고자 하는 점의 x, y, z값만을 glVertex()의 매개변수로 전달해 주면 된다.

나. DrawCurve

DrawCurve() 함수는 CURVE에 대한 포인터와 curve 색깔을 지정해 주는 변수를 매개변수로 받아 NURBS curve를 그려주는 기능을 한다.

이 함수의 과정을 살펴보면, 먼저 CURVE 포인터로부터 NURBS curve 포인터를 가져온다. 그리고, NURBS curve의 order, control point 갯수, knot 갯수, control point, knot를 가져온다. 이러한 정보를 가지고, GLU NURBS Interface를 이용하여 NURBS curve를 그릴 수 있다. OpenGL을 이용하여 NURBS curve를 그리는 방법은 아래와 같다.

① 새로 만들어진 NURBS curve나 surface를 참조하기 위한 NURBS object의 pointer를 생성하기 위해 gluNewNurbsRenderer()을 사용한다.

② rendering value값(선이나 폴리곤의 최대 크기)을 선택하기 위해서는 gluNurbsProperty() 함수를 호출한다.

void GluNurbsProperty(GLUnurbsObj *nobj, GLenum property, GLfloat value);

property : GLU_SAMPLING_TOLERANCE, GLU_DISPLAY_MODE, GLU_CULLING, GLU_AUTO_LOAD_MATRIX

③ 만약 에러를 만났을 때 그 것을 알려주길 원하면 gluNurbsCallback() 함수를 호출한다.(에러 체크는 조금씩 성능을 감소시킨다.)

④ gluBeginCurve() 이나 gluBeginSurface()을 호출함으로써 curve, surface를 만들기 시작한다.

⑤ 실제로 curve나 surface를 만드는 방법은 적어도 하나의 control point와

knot sequence 그리고 NURBS를 위한 polynomial basis function을 매개변수로 하는 gluNurbsCurve(), gluNurbsSurface() 함수를 호출하는 것이다.

⑥ gluEndCurve(), gluEndSurface()을 호출함으로써 curve나 surface를 완성한다.

```
BOOL DrawCurve( CURVE *crv, UINT nFoColFlag)
{
    // Get NURB curve ==========================================
    nurbcrv *nurb_crv;
    SApi_GetNurbOfCrv(crv->geo_curve(), nurb_crv);

    // Get informations of NURB curve ==========================
    long order = nurb_crv->get_order();
    long no_ctlpt = nurb_crv->get_no_ctlpt();
    long no_knot = nurb_crv->get_no_knot();
.
.
.
    // Draw curve with OpenGL function =========================
    GLUnurbsObj *NurbCurve;

    NurbCurve = gluNewNurbsRenderer();
    gluNurbsProperty(NurbCurve, GLU_SAMPLING_TOLERANCE, 25.0);

    gluBeginCurve(NurbCurve);
      gluNurbsCurve(NurbCurve,  no_knot,  knot,  4,  ctlpt,  order,
GL_MAP1_VERTEX_4);
    gluEndCurve(NurbCurve);

    return TRUE;
}
```

다. DrawSurface

DrawSurface() 함수는 SURFACE에 대한 포인터와 quick shading으로 할 건지

full shading으로 할 건지를 결정하는 ShadeMode값과 surface 색깔을 지정해 주는 nFocolFlag 변수를 매개변수로 받아 NURBS surface를 그려주는 기능을 한다. 이 함수의 과정을 살펴보면, 먼저 SURFACE 포인터로 부터 NURBS surface 포인터를 가져온다. 그리고, NURBS suriace의 u_order, v_order, u_control point 갯수, v_control point 갯수, u_knot 갯수, v_knot 갯수, control point, u_knot, v_knot를 가져온다. 이러한 정보를 가지고, GLU NURBS Interface를 이용하여 NURBS surface를 그릴 수 있다. OpenGL을 이용하여 NURBS surface를 그리는 방법은 위에서 설명한 NURBS curve 방법과 같다. 단지, NURBS surface와 함께 빛(lighting)을 사용하려면, 자동적으로 surface normal을 구하도록 glEnable(GL_AUTO_NORMAL)을 해주어야 한다는 점과 gluNurbsProperty() 함수중 두번째 변수인 property를 GLU_SAMPLING_TOLERANCE 지정해 주었다면, shading 시 quick shading인지 full shading인지의 여부에 따라 스크린 픽셀에서의 선이나 폴리곤을 세분화하는 최대 길이를 지정해 주면 된다. 기본 값은 50.0으로 되어 있다. 따라서, full shading을 원할 때는 최대길이를 5.0정도로 짧게 주어서 보다 촘촘하고 정확한 shading으로 표현되도록 한다.

```
BOOL DrawSurface( SURFACE* surf, UINT ShadeMode, UINT nFoColFlag)
{
    long i,j;

    if( surf == NULL ) return TRUE;

    // Set display color ==================================
    if( StateShaded == TRUE )
            SetShadingColor( (ENTITY*)surf, nFoColFlag );
    else
            SetDisplayColor( (ENTITY*)surf, nFoColFlag );

    // Get NURB data of surface ===========================
    nurbsuf *nurb_suf;
```

```
    SApi_GetNurbOfSurf(surf->geo_surface(), nurb_suf);

    long u_order = nurb_suf->get_u_order();
    long v_order = nurb_suf->get_v_order();
    long nu_ctlpt = nurb_suf->get_nu_ctlpt();
    long nv_ctlpt = nurb_suf->get_nv_ctlpt();
    long nu_knot = nurb_suf->get_nu_knot();
    long nv_knot = nurb_suf->get_nv_knot();
.
.
.
    // Set sampling tolerance for shading ========================
    double smptol;
    if( ShadeMode == 0 ) smptol = 50.0;
    else if( ShadeMode == 1 ) smptol = 5.0;
    else smptol = 30.0;

    // Set parameter for display surface =====================

    GLUnurbsObj *NurbSurface;


    NurbSurface = gluNewNurbsRenderer();
    gluNurbsProperty(NurbSurface, GLU_SAMPLING_TOLERANCE, smptol);
        if( StateShaded == FALSE )
gluNurbsProperty(NurbSurface, GLU_DISPLAY_MODE, GLU_OUTLINE_PATCH);

    // Draw surface ==========================================
    gluBeginSurface(NurbSurface);
    gluNurbsSurface(NurbSurface,
                    nu_knot, u_knot,
                    nv_knot, v_knot,
                    4, nu_ctlpt*4, ctlpt,
                    u_order, v_order,
                    GL_MAP2_VERTEX_4);
    gluEndSurface(NurbSurface);
    return TRUE;
}
```

3. 형상 PICKING

고급 그래픽 라이브러리에서 가장 편리한 기능은 Picking기능이라고 할 수 있다. 이 기능은 화면에 그림을 그리고, 사용자로부터 그려진 그림 중에서 특정한 물체를 선택할 수 있도록 해주는 기능이다. 이 기능을 이용하면, 아무리 복잡한 물체라도 어렵지 않게 사용자의 물체 선택 입력을 받을 수 있다.

화면의 물체를 선택하는데는 두 가지 방법을 사용할 수 있는데, 첫째 Selection기법을 이용한 것이고, 둘째는 Picking을 이용하는 것이다. Picking 기법은 Selection기법의 일종으로 OpenGL의 유틸리티 함수를 사용하여 Selection을 보다 쉽게 만들어 놓은 것이다.

Selection을 이용하려면 몇 가지 사항을 알아두어야 할 필요가 있다. 첫째는 glRenderMode( )함수를 이용하여 현재 Selection 모드인가 아니면 Render모드인가를 지정하는 것이다. 그래픽 라이브러리에서 Selection은 화면의 그림을 모두 그린 후에 Selection을 할 영역을 잡고 그림을 메모리 상에 다시 그리면 OpenGL은 그림 중에 어떠한 부분이 Selection영역에 포함되어있는가를 검사하여 그 값을 돌려준다.

여기에서 Selection영역을 잡아주는 방법은 Projection행렬을 이용한다. 즉, Projection행렬을 원하는 영역으로 지정함으로써 영역에 물체가 포함되는가를 검사하게 된다.

그리고 또 한가지 중요한 사항이 있는데, 이는 Name이라는 것이다. 이것은 특정한 물체를 구분하는데 사용하는 것으로 glLoadName( )을 사용하여 이름이 지정이 되면 이후에 그려지는 모든 그림은 이 함수가 다시 나타날때까지 이 이름을 갖게 된다. 해당 그림이 선택되는 경우 이 이름으로 그 결과가 돌려지는 것이다.

이러한 이름도 역시 스택을 가질 수 있으며 glPushName( )과 glPopName( ) 함

수가 제공된다. 이를 이용하면 계층적으로 이름을 지정할 수 있으며, 그 결과도 계층적으로 돌려 받게 되므로 원하는 그림을 훨씬 쉽게 선택할 수 있다.

Picking이라 함은 Selection의 일종으로 gluPickMatrix( )함수를 이용하여 Selection 영역을 지정하는 것이 다르다. 이 함수는 화면좌표계상의 한점과 폭, 높이를 입력받아 지정한 점 주위에 지정한 크기로 Selection 영역을 자동적으로 잡아준다. 이를 이용하면 마우스를 이용한 그림의 선택과정이 매우 간단해 진다.

그러면 Picking을 구현하기 위한 과정을 설명하면 다음과 같다.

① glSelectBuffer( )를 이용하여 결과를 돌려받을 자료구조(정수의 일차원 배열로 구성된다)를 지정한다.

② glRenderMode( )를 이용하여 Select 모드로 바꾼다.

③ glInitName( )과 glPushName( )을 이용하여 이름스택을 초기화한다. 언제나 glPushName( )을 glInitName( ) 다음에 꼭 실행하여야 한다.

④ Selection에 사용될 Projection 행렬을 지정한다.

⑤ 선택되기 원하는 그림을 그린다. 이 그림을 그리는 과정에 glLoadName( ), glPushName( ), 그리고 glPopName( )을 이용하여 이름을 지정한다.

⑥ glRenderMode( )를 이용하여 Render모드로 다시 환원한다.

⑦ glSelectBuffer( )에서 지정한 버퍼를 이용하여 선택된 그림을 처리한다.

"SurfART"에서의 picking 구현을 살펴보면 다음과 같다.

```
BOOL    PickEntity(    CPoint    point,    PTRLIST<ENTITY*>    *EntList,
```

```
PTRLIST<ENTITY*> &CPickedEntList )
{
        if( EntList->count() <= 0) return FALSE;

        GLuint selectBuf[PICK_BUFSIZE];
        GLint hits;
        GLint viewport[4];
        int x, y;
        UINT nFoColFlag;

        // Get picked position
        x = point.x;
        y = point.y;

        // Start picking mode
        glGetIntegerv(GL_VIEWPORT, viewport);

        // hit record에 저장할 array 지정
        glSelectBuffer(PICK_BUFSIZE, selectBuf);
        // selection mode로 선택
        (void) glRenderMode(GL_SELECT);

         // name stack을 초기화한다.
        glInitNames();
          glMatrixMode(GL_PROJECTION);

      // Save Current Projection Matrix
      // 원래의 scene을 그리는 viewing volume과 selection 사용 시의  viewing
volume과 다르기 때문에 glPushMatrix()와 glPopMatrix()로 현재상태를 저장한
다.
        glPushMatrix();
        glLoadIdentity();

       // Create 5x5 pixel picking region near cursor location
        gluPickMatrix((GLdouble)x, (GLdouble)(viewport[3] - y), 5.0, 5.0,
viewport);
       glOrtho(Ortho_left, Ortho_right, Ortho_bottom, Ortho_top,
                    Ortho_near, Ortho_far);

        // Redraw picking list with select mode
        DisplayEntity( EntList, GL_SELECT );

        // Reload Matrix
        glPopMatrix();

        // Exit Picking Mode
```

```
 hits = glRenderMode( GL_RENDER );

// if there is no picked entity
 if (hits < 1) return FALSE;

 // Set picked entity to list of picked entities
  ENTITY *Ent;
  GLuint EntName, *ptr;

// hit record 구성 요소
// ① hit 발생했을 때의 name stack에 저장된 name의 개수
// ② 기록된 hit중 최소 최대 큰 값 [0, 1]
// ③ hit 발생했을 때의 name stack의 내용, 가장 밑의 내용이 먼저 나온다.

// 실제 내용은 array중 4번째 요소부터
    ptr = (GLuint *) selectBuf;
        ptr++;ptr++;ptr++;
        EntName = *ptr;

        Ent = (*EntList)[EntName];

        if( CPickedEntList.find(Ent) < 0 )
                CPickedEntList.addtail( Ent );
        return TRUE;
}
```

4. View Transformation

특정한 3차원 세계의 도형을 화면에 나타내기 위해서는 여러 과정을 거쳐야 하는데, 이를 그래픽 파이프라인(graphics pipeline)이라고 한다. OpenGL 역시 일반적으로 3차원 그래픽 라이브러리에서 통상적으로 사용되는 방식과 비슷하다. 이런 과정을 거치면 특정한 모델 좌표계의 점이 화면 좌표계의 점으로 변환된다. 이 과정에서 보면 사용자 프로그램에서 입력으로 들어오는 꼭지점은 모델 좌표계라는 각각의 형상마다 정의된 좌표계 상에서 정의된다는

것을 알 수 있다. 이 점들은 모델 뷰 행렬(modelview matrix)에 의해 eye 좌표계의 값으로 변환되고, 다시 프로젝션 변환 행렬(projection matrix)을 이용해 화면에 나타낼 물체의 크기가 조정되며 다시 클리핑에 관련된 투시 디비전(perspective division)과정을 거쳐 정규화된 좌표계로 변화된 후 최종적으로 뷰포트 변환(viewport transformation)을 통해 화면에 투영된다.

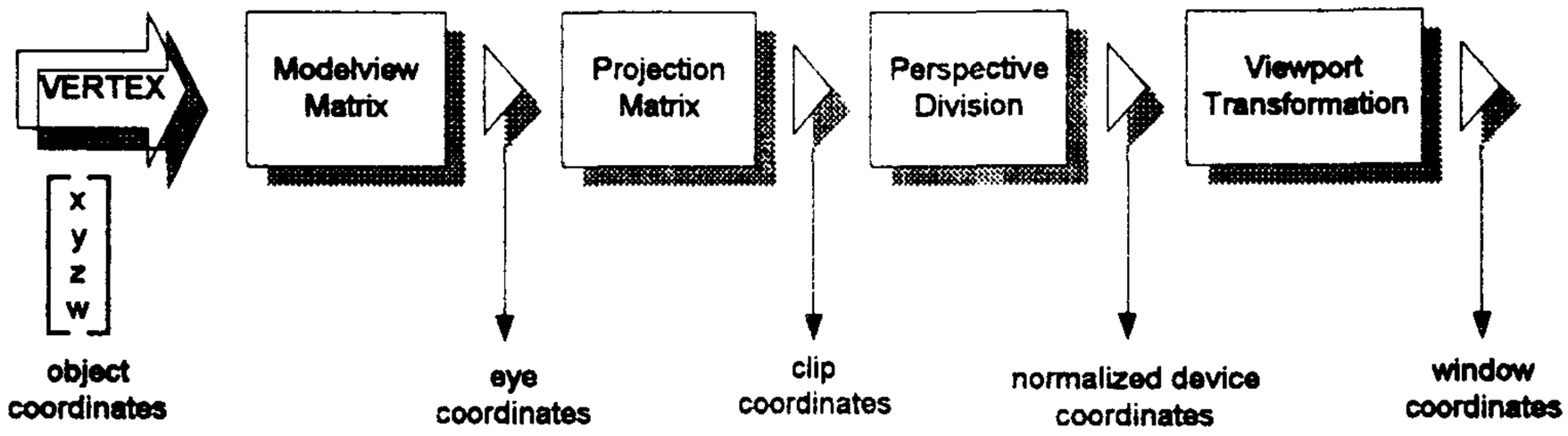


그림 3.13 OpenGL에서의 graphics pipleline

일반적으로 모델 뷰 행렬은 모델 좌표계 상의 특정 점을 월드(world) 좌표계 상의 점으로 이동시키는 역할을 한다. 통상 다른 라이브러리의 경우 이 행렬의 역할을 담당하는 행렬이 모델 행렬과 뷰 행렬의 두 가지로 나누어져 있지만, OpenGL의 경우는 하나로 해결하고 있다. 사실 뷰 행렬 자체가 모델 변환 행렬의 조합으로 만들어지므로 이런 접근 방식이 더 효율적일 수도 있다. 그러면 그래픽 파이프라인을 따라서 각각의 행렬에 대해 구체적으로 알아보도록 하자.

모델 뷰 행렬은 회전, 이동, 크기 변환 등의 기능을 제공한다. 물론 뷰를 바꾸고자 하는 경우에도 사용될 수 있으며, 이들 함수는 OpenGL 유틸리티 함수를 통해 제공된다. OpenGL에서 뷰의 설정은 카메라를 연상하면 된다. 즉 한 장의 사진을 만들기 위해서는 먼저 카메라를 원하는 위치에 놓고, 피사체를 결정한 후 그것에 초점을 맞추듯이 OpenGL에서도 눈의 위치와 바라보는 위치를 지정한

다. 물론 여기에 눈의 위쪽이 향한 방향(upvector)도 입력해야 한다. 이런 사항을 지정하기 위해 gluLookAt이라는 함수가 제공된다. 이 함수는 두 번의 회전 모델 행렬을 이용해 만들 수 있다. 이제 3차원 그래픽에서 가장 중요한 요소로 원근감을 나타내는 프로젝션 변환 행렬에는 일반적으로 정사영 프로젝션(orthographic projection)과 투시 프로젝션(perspective projection)의 두 가지 방법이 있는데, 전자의 경우는 거리에 관계없이 화면 상에는 항상 똑같은 크기로 표현되는 것을 말하고, 후자는 먼 거리의 물체는 작게, 그리고 가까운 물체는 크게 화면에 나타내는 프로젝션을 의미한다.

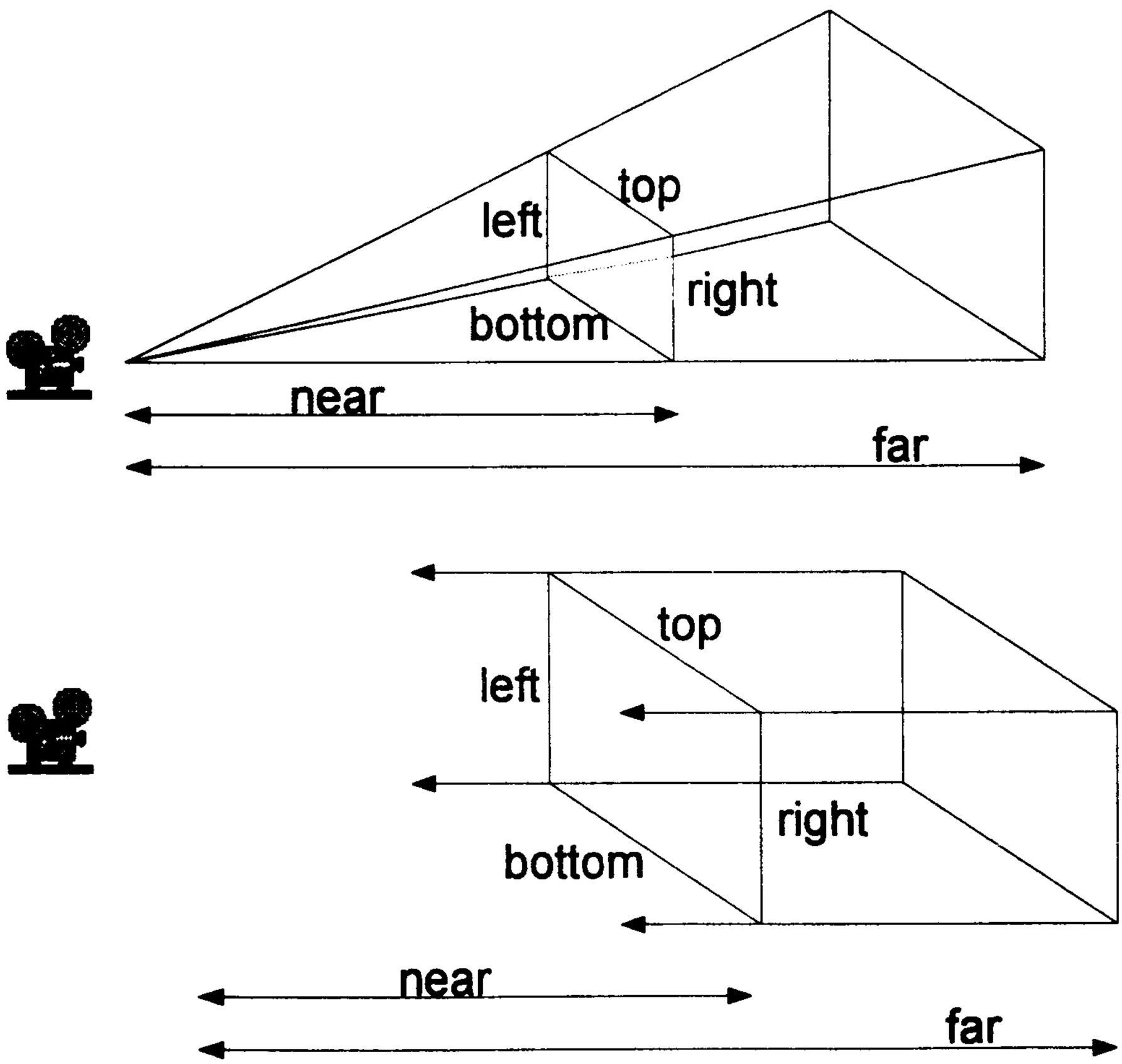


그림 3.14 OpenGL에서의 프로젝션 방법

마지막으로 뷰포트 변환의 경우를 살펴보자. 뷰포트는 화면에서 그림이 실제로 그려질 영역을 나타낸다. 즉, 윈도우의 전체에 그림을 나타낼 것인가, 아니면 일부분에 나타낼 것인가를 지정하는 것으로 glViewport( ) 함수를 이용한다.

가. Zooming

"SurtART"는 정사영 프로젝션(orthographic projection)을 사용하였다. 그 이유는 상대적 크기를 갖는 image보다 실제 치수를 갖는 image를 원하였기 때문이다. 따라서 화면에 보이는 영역의 결정은 아래 그림과 같이 left, right, top, bottom, near, far의 값으로 결정된다.

zoom기능은 마우스의 왼쪽 버튼을 드래그해서 zoomin할 영역을 선택하면 선택된 영역을 확대하여 보여주는 역할을 한다. 먼저 마우스로 드래그하여 선택된 영역의 폭과 높이를 계산하여 화면비율(1:1.4)에 맞는 가를 확인하여 만일 그것이 맞지 않으면 큰 쪽에 맞추어 다른 쪽의 높이나 폭을 다시 재계산해 준다. 그런 다음 그림이 그려질 스크린 크기와 실제의 viewing volumn 크기를 구한다. 그리고 새롭게 계산된 영역으로 viewing volumn의 영역을 바꾸어 주면 되는 것이다. 즉, top, bottom, left, right 값을 마우스가 선택한 영역의 적당한 값으로 계산해 주면 된다.

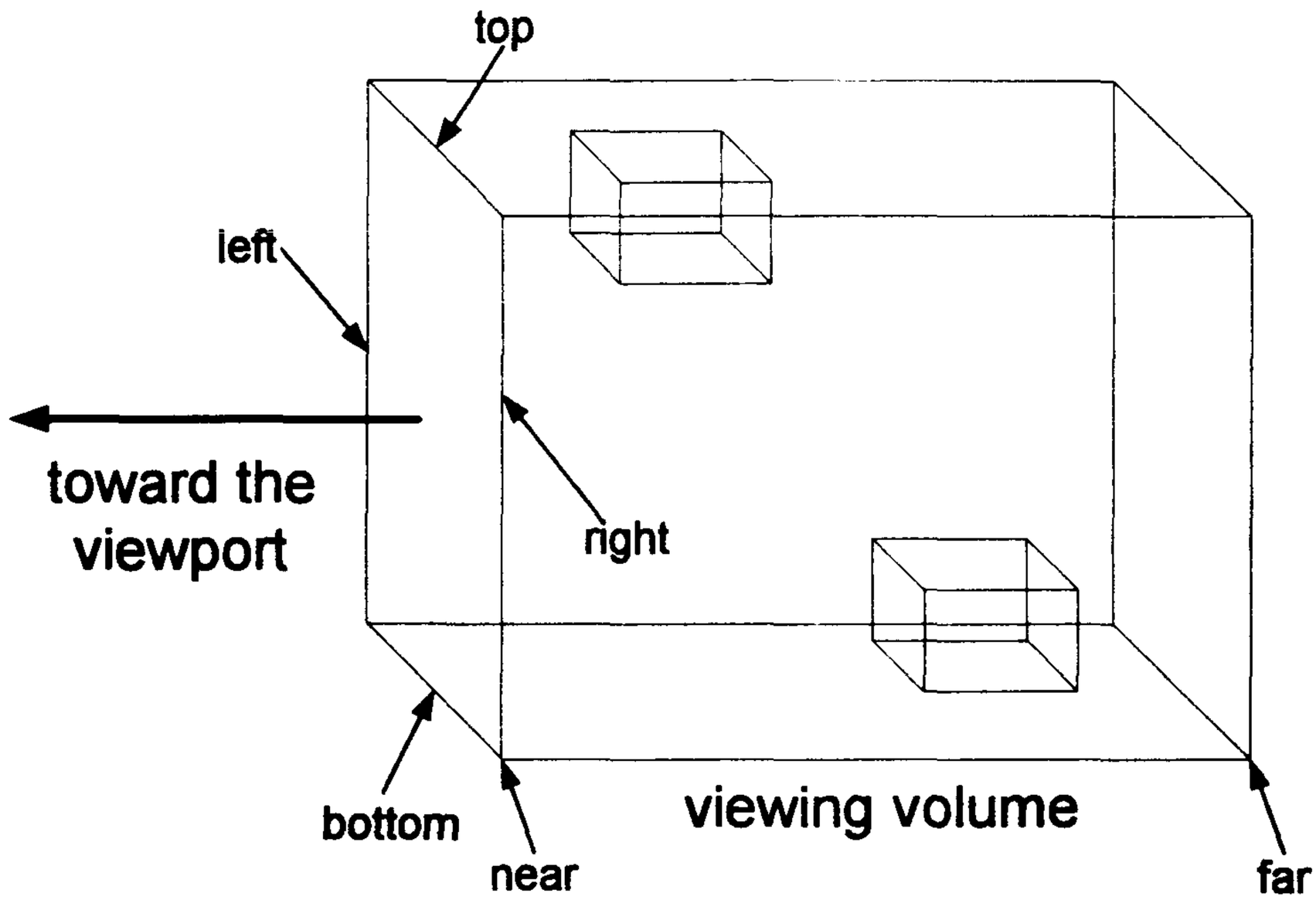


그림 3.15 OpenGL에서의 orthographic projection

루틴명 : void CSurfartView::OnDisplayZoomIn()

void CSurfartView::OnDisplayZoomOut()

나. Panning

Panning기능은 마우스의 왼쪽 버튼을 드래그해서 움직일 방향과 움직일 양을 선택하면 선택된 양만큼 화면을 이동시켜주는 역할을 한다. Zooming 기능에서 설명한 바와 거의 같은 방법을 이용한다. 즉, viewing volumn 중 left, right, top, bottom값을 변화시킨다는 것은 같다. 다만, 새롭게 계산된 left, right, top, bottom값을 가지고 실제 viewing volumn에서 움직여진 양을 계산하여서 원래의 left, right, top, bottom값에 더해준다는 점이 다르다.

루틴명 : void CSurfartView::OnDisplayPanning()

다. Rotation

rotation 기능은 gluLookAt() 함수의 eye position 값과 upvector값을 변화시켜 viewpoint의 변환에 따라 화면을 회전시키는 기능을 한다. 앞서 설명했듯이 OpenGL에서는 눈의 위치와 바라보는 위치를 지정하고, 눈의 위쪽이 향한 방향 즉 upvector값도 지정하게 된다. 이것을 쉽게 gluLookAt()함수를 이용하여 사용하는데 이 함수는 결과적으로 두 번의 회전 모델 행렬을 이용해 만들어 진다. 따라서 mouse의 움직임에 따라 x축의 움직인 양과 y축의 움직인 양을 가지고, eye position 값과 upvector값을 계산해 주는 것이다.

루틴명 : void CSurfartView::OnDisplayRotation()

여 백

제 4 장 데이터 및 파일 구조

제 1 절 개 요

본 연구에서 개발한 SurfART 시스템은 곡면 모델링 시스템(surface modeling system)으로, 사용자의 모델링 작업으로 생성되는 모델을 유지, 저장, 복원할 수 있는 일정한 데이터 구조와 방법이 필요하다. 일반적으로 곡면 모델링 시스템에서 생성되는 데이터는 곡면의 모양을 표현하는 형상 정보(geometry)와 곡면 간의 연결 관계를 나타내는 위상 정보(topology)가 생성된다. 본 연구에서도 이러한 정보들을 일관된 하나의 구조로 표현하고, 효율적으로 관리할 수 있는 데이터 구조를 설계하고, 구현하였다. 또한, 데이터 구조를 관리하는 API(Application Procedural Interface) 함수를 개발하여 응용 프로그램 작성에 편리하게 사용할 수 있도록 하였다.

본 연구에서 개발한 데이터 구조는 다음과 같은 특성을 갖고 있다. 첫째로, 비다양체(non-manifold) 형상을 표현할 수 있도록 하였다. 일반적으로 형상을 모델링하는 과정에서는 비다양체 형상이 발생된다. 또한, SurfART시스템의 주 기능이 측정장치로부터 입력된 점 데이터를 이용하여 복잡한 자유곡면(free-formed surface), 특히 복합곡면(sculptured surfaces)과 트림곡면(trimmed surface)을 생성하는 작업이므로 비다양체 형상이 필연적으로 발생하게 된다. 그러므로 데이터 구조는 이러한 형상을 수용할 수 있어야 한다. 비다양체 모델을 표현할 수 있는 데이터 구조는 개념 설계는 물론 제품생산에 필요한 모든 형상 설계를 하나의 모델링으로 수용할 수 있는 장점을 갖고 있다. 둘째로, 형상 표현의 기본식으로 NURBS(Non-Uniform Rational B-Spline)를 사용하였다. Ferguson, Bezier와 같은 다항식 표현(polynomial-based expression) 방법은 복잡한 곡면 형상의 표현, 수정, 곡면간의 연결 등이 용이하지 않은 단

점이 있다. 그러나 NURBS는 해석 곡선, 곡면(analytic curve, surface)을 정확히 표현할 수 있고, 곡면의 형상을 제어할 수 있는 요소가 가장 많은 표현 방법이다. 또한, NURBS는 형상 데이터 교환의 표준인 IGES(Initial Graphics Exchange Specification)에도 채택되어 있으므로 상용 CAD/CAM 시스템과의 데이터 교환이 용이한 장점도 있다. 그러나 다항식 표현 방법도 수치적 계산이 빠른 장점이 있으므로, 이러한 장점을 이용하기 위하여 NURBS를 다항식 표현으로 변환하는데 필요한 정보를 데이터 구조에 갖고 있도록 하였다. 세째로, 측정기로부터 입력된 점 데이터와 내부적으로 생성되는 점 데이터를 구분하여 표현하였다. 측정기로부터 입력된 점 데이터는 곡선이나 곡면을 생성한 후 삭제되는 데이터이므로 이들을 내부적으로 생성되는 점 데이터와 구분하여 형상 정보만 저장함으로써 많은 점 데이터를 편리하게 관리하고, 저장되는 데이터 양이 최소화될 수 있도록 하였다. 넷째로, 객체 지향 프로그래밍(object-oriented programming) 방법론을 적용하여 데이터 구조를 설계 하였고, 대표적인 객체 지향 프로그래밍 언어인 C++를 사용하여 구현하였다. 일반적으로 객체 지향 프로그래밍은 시스템의 설계 및 개발을 용이하게 하고, 개발된 시스템의 확장성 및 재이용을 극대화할 수 있는 장점이 있다. 다섯째로, 데이터 구조를 관리하는 API 함수를 제공하여 응용 프로그램 구현이 용이하게 하였다. 데이터를 생성, 수정, 삭제하는 많은 응용 프로그램들이 동일한 API를 사용함으로써 응용 프로그램 작성이 용이할 뿐 만 아니라 모델링 과정에도 데이터 구조를 일관되게 유지할 수 있다. 또한, 데이터의 입력, 수정, 삭제가 오일러 작업자(Euler operator)를 통하여 수행되도록 하여 모델의 위상 정보가 합당하게 유지되도록 하였다.

제2절에서는 데이터 구조를 설계하는데 사용되는 기본 이론으로, 형상 모델을 표현하는 방법과 기존에 개발되어 있는 타 형상 모델링 시스템의 데이터 구조를 알아보고, 제3절과 제4절에서는 본 연구에서 개발한 SurfART 시스

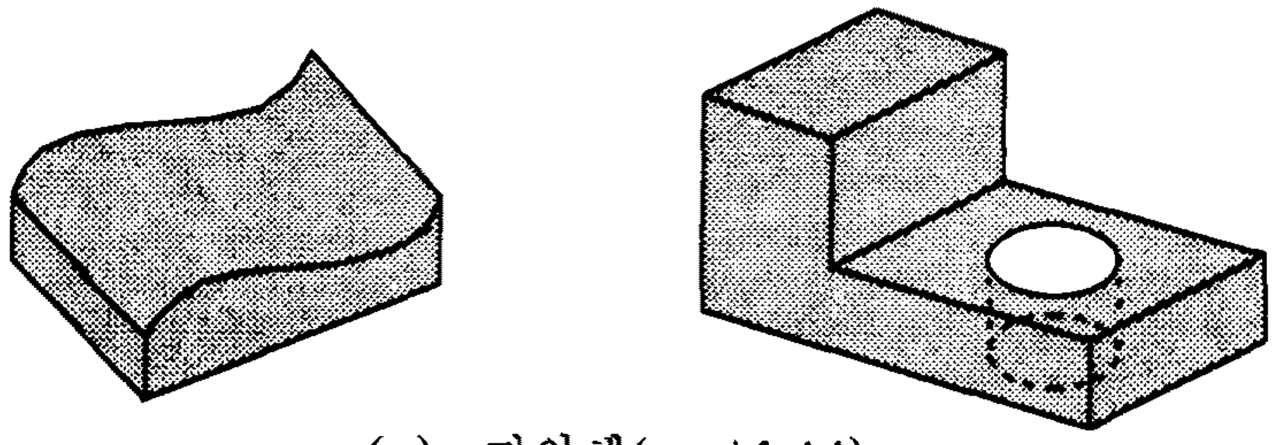
템의 데이터 구조, 구현에 사용한 클래스, 데이터 구조를 관리하는 API 함수와 오일러 작업자 등에 대하여 설명한다. 제5절은 SurfART 시스템에서 사용하는 데이터 파일의 구조와 데이터 구조에 들어 있는 데이터를 파일로 저장하고 복원하는 API 함수에 관하여 기술한다.

제 2 절 형상 모델링 시스템의 데이터 구조

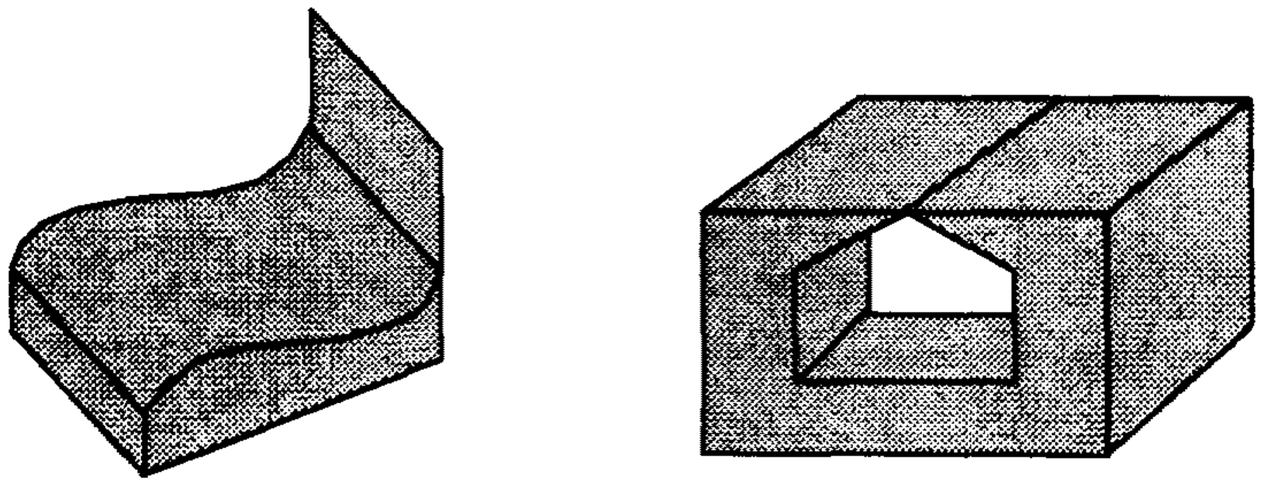
형상 모델링 시스템은 형상을 표현하는 방법과 저장되는 정보에 따라 와이어 프레임(wireframe) 모델링, 곡면(surface) 모델링, 솔리드(solid) 모델링 등으로 발전 되었고 최근에는 이 들을 하나로 통합하는 비다양체 형상 모델링(non-manifold geometric modeling) 시스템이 개발되고 있다. 일반적으로 모델로부터 얻어야 만 되는 정보에 따라 적절한 방법을 이용하나, 2D 도면 작성에는 와이어 프레임 모델링, 형상의 외형 정보만을 필요로 하는 경우에는 곡면 모델링, 공학적 해석을 위해서는 솔리드 모델링을 사용하여 형상을 모델링 한다. 곡면 모델링은 형상의 NC 가공 정보를 얻을 수 있는 모델을 최소의 정보량으로 생성할 수 있으므로 가전, 자동차, 항공기 등과 같이 자유 곡면으로 이루어진 부품이나 금형의 모델링에 많이 사용되고 있다. 그러나 상용되고 있는 대부분의 곡면 모델링 시스템은 형상 요소의 형상 정보만을 순차적으로 저장하거나 위상 정보를 저장하기 위하여 별도의 데이터 구조를 갖고 있으므로 형상 요소간의 인접 정보를 얻기 위해서는 형상 정보를 이용한 수치적 연산이나 추가적인 자료 탐색을 해야만 한다. 물론, 솔리드 모델링 시스템은 형상 정보와 위상 정보를 하나의 데이터 구조에 갖고 있으나 모델링 과정에서 발생되는 비다양체 형상은 표현할 수 없다. 비다양체 형상 모델링 시스템은 모델의 형상 정보와 위상 정보는 물론, 모델링 과정에서 발생되는 비다양체 형상을 완전하게 표현할 수 있다. 그러나 곡면 모델링에서는 필요 없는 정보도 저장되어 모델의 정보량이 커지는 단점이 있다. 그러므로 본 연구에서는 비다양체 형상 모델링의 데이터 구조를 바탕으로 곡면 모델에서는 필요 없는 부분을 제거하여 저장되는 데이터의 양이 최소화 되도록 데이터 구조를 설계하였다.

1. 형상 모델의 구분

일반적으로 형상 모델링 시스템에서 생성되는 모델은 크게 다양체(manifold)와 비다양체(non-manifold)로 구분한다. 3차원 공간상에 존재하는 임의의 도형 상에 점의 위치를 지정하기 위해서 n개의 독립 변수를 필요로 하는 도형을 n차원 도형이라 한다면, 3차원 공간 상의 점(point)은 0차원, 곡선(curve)은 1차원, 곡면(surface)은 2차원 도형이 된다. 다양체란 임의의 한 차원 도형으로만 모델링 되는 형상을 의미하고, 비다양체란 여러 차원의 도형으로 모델링 되는 복합 다양체를 의미한다[4-1]. 따라서 그림 4.1의 (a)와 같이 실존하는 모든 물체는 다양체이며 모델링의 최종 결과도 다양체이다. 그러나 모델링하는 과정에서는 비다양체 형상이 생성될 수 있으며, 개념 설계에서는 비다양체 형상이 모델링의 최종 결과가 될 수도 있다. 그러므로 비다양체 형상을 수용할 수 있는 데이터 구조는 와이어 모델, 곡면 모델, 솔리드 모델을 하나의 구조로 표현하고 관리할 수 있게 된다.



(a) 다양체(manifold)



(b) 비다양체(non-manifold)

그림 4.1 다양체와 비다양체 (Manifold vs. Non-manifold)

2. 형상 모델의 표현

일반적으로 형상 모델을 일관된 구조로 표현하는 방법은 경계 표현(Boundary representation) 법이 많이 사용되며, 경계 표현으로 형상 모델의 위상을 표현하는 방식은 크게, 인접 그래프(incidence graph)와 정렬된 위상 표현(ordered topological representation) 방식으로 구분할 수 있다[4-2]. 인접 그래프 방식은 모델의 형상 요소를 각각의 노드(node)로 표현하고 각 노드를 잇는 선으로 형상 요소간의 인접 관계를 표현하는 방식이다. 반면에 정렬된 위상 표현은 위상 요소(topological entity)를 사용하여 모델의 위상 정보, 경계 정보, 인접 관계 등을 함축적으로 표현하는 방식이다. 인접 그래프에 비하여 정렬된 위상 표현 방식은 데이터 구조의 일관성을 유지하기 쉽고, 필요한 정보를 데이터 구조로부터 쉽게 추론해 낼 수 있으므로 대부분의 데이터 구조들이 이 방식을 사용하고 있다. 다양체와 비다양체를 경계 표현 관점에서 비교하면 다양체는 하나의 모서리를 오직 2개의 면만이 공유할 수 있으나, 비다양체는 두 개 이상의 면들이 하나의 모서리를 공유할 수 있다는 점이 가장 큰 차이이다. 그러므로 비다양체를 표현할 수 있는 데이터 구조는 이러한 인접 관계를 수용할 수 있고, 다양체는 물론 2차원 박판(sheet)나 공간상에 독립된 정점(isolated point)도 표현이 가능하다.

가. 모델의 위상 요소

비다양체 모델의 위상 정보를 표현하는데는 기본 위상 요소(basic topological entity)와 보조 위상 요소(additional topological entity)들을 사용한다[4-3]. 기본 위상 요소는 개발된 모든 데이터 구조들이 사용하는 요소이고, 보조 위상 요소는 곡면간의 인접 관계를 표현하는데 사용되는 요소로 데이터 구조 마다 다르며, 데이터 구조의 특징을 파악할 수 있는 요소이다.

(1) 모델(model)

하나 이상의 공간 영역으로 구성된 3차원 위상 모델링 공간(topological modeling space)으로 형상 모델링 시스템에서 독립된 형상의 위상 정보를 관리하는 단위이다.

(2) 영역(region)

공간에 존재하는 닫혀진 볼륨으로, 형상 요소가 정의되지 않아도 모델에는 적어도 하나의 영역이 존재하는 것으로 가정한다. 그림 4.2는 내부가 빈 상자형 모델을 예로 기본 위상 요소들을 표현한 것으로, R_0가 가상의 경계로 닫혀지고 무한대 크기의 볼륨을 갖는 가상적인 영역이다.

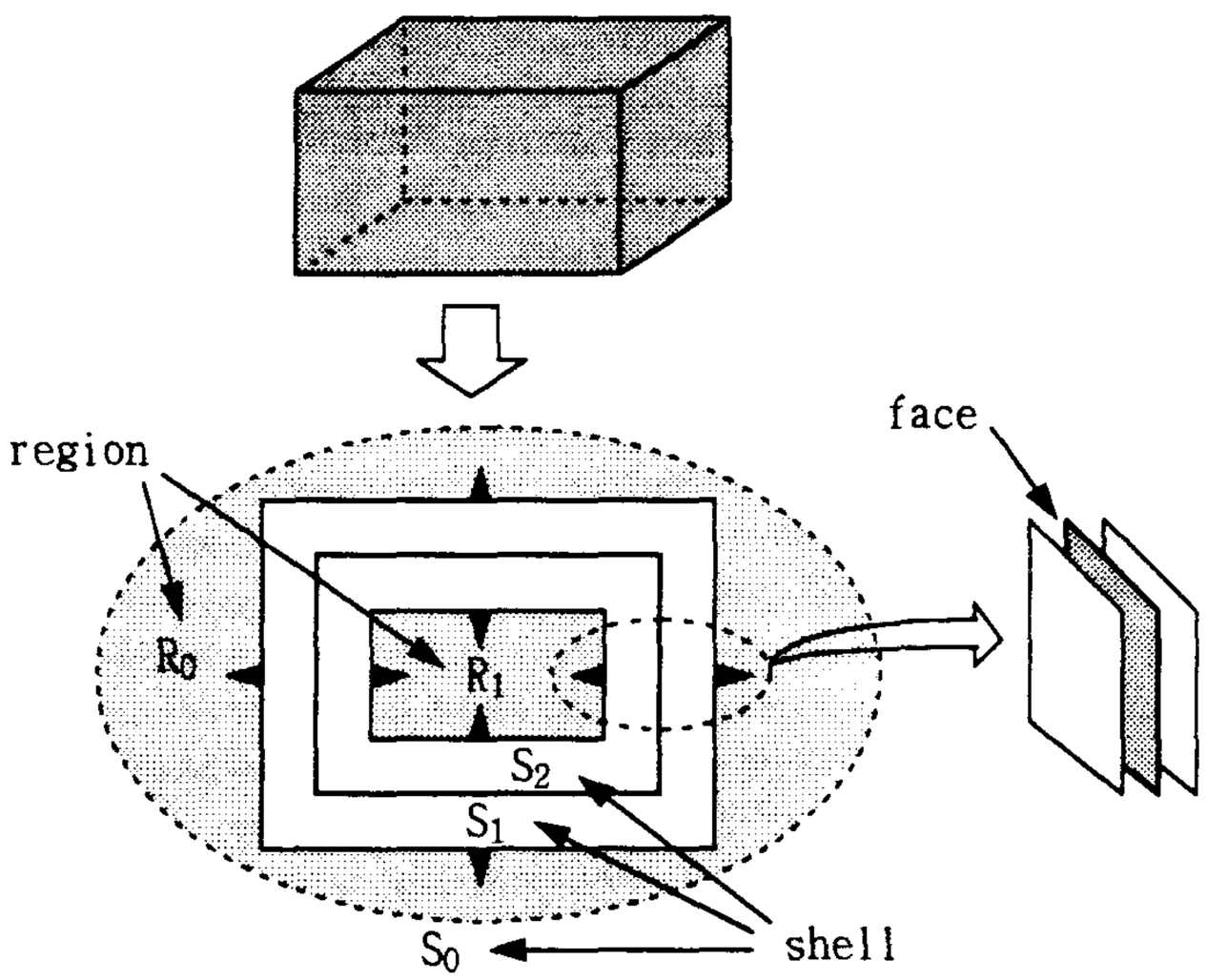


그림 4.2 영역, 쉘, 면 요소간의 관계
(Relation among region, shell and face entities)

(3) 쉘(shell)

영역의 경계 곡면을 의미하며 방향성을 갖고 있다. 쉘은 항상 영역의 안쪽

을 향하는 방향을 갖게 되며, 바깥쪽 쉘(peripheral shell)과 안쪽 쉘(void shell)로 구분된다. 그림 4.2에서 S_0는 무한대 영역인 R_0의 가상적인 바깥쪽 쉘이며, S_1은 R_0의 안쪽 쉘이다. 또한, S_2는 R_1의 바깥쪽 쉘이다.

(4) 면(face)

쉘의 경계를 구성하는 요소로 방향성을 갖고 있으며 경계 지워 지고 스스로 닫히지 않는 곡면을 의미한다. 면은 루프에 의하여 경계 지워 지며 적어도 하나의 루프를 갖는다. 면의 방향은 루프에 의하여 결정된다. 그림 4.3은 면과 루프의 관계를 표현한 것으로 면 F_0는 루프 L_0, L_1, L_2, L_3를 경계로 정의되는 곡면이다.

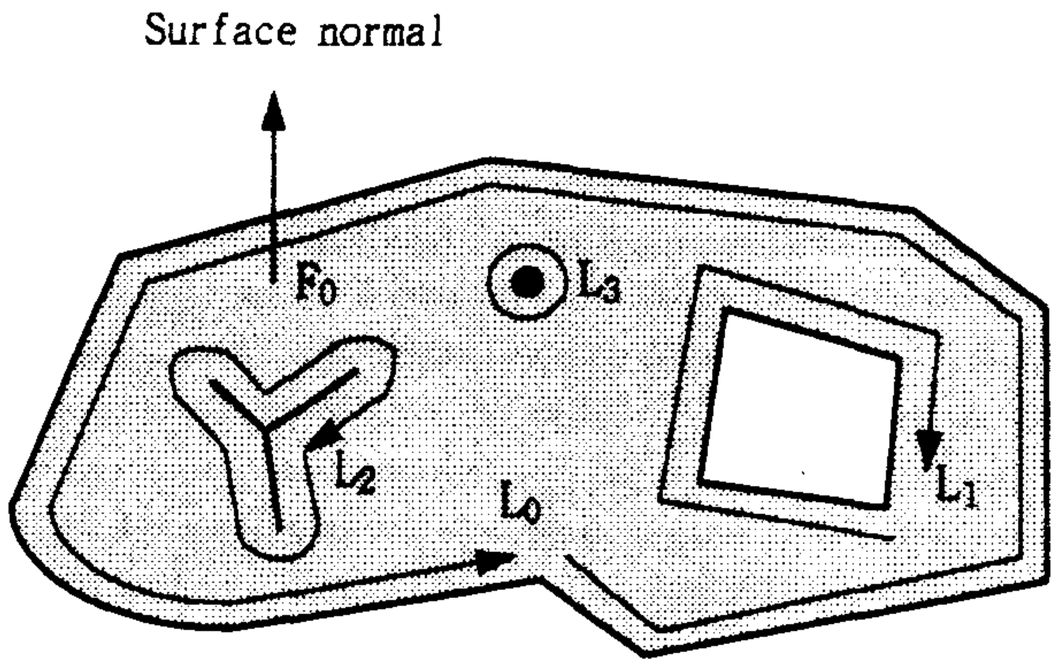


그림 4.3 면과 루프의 관계 (Relation between face and loop)

(5) 루프(loop)

루프는 방향성을 갖으며 곡면의 연결된 경계를 의미한다. 루프는 방향에 따라 바깥쪽 루프(peripheral loop)와 안쪽 루프(hole loop)로 구분된다. 바깥쪽 루프는 곡면의 법선 방향을 기준으로 반시계 방향이고, 안쪽 루프는 시계 방향이다. 그림 4.3에서 L_0는 바깥쪽 루프이고 L_1, L_2, L_3등은 안쪽 루프이다. 그러나 L_3와 같이 곡면 위에 단독으로 존재하는 정점은 방향성이 없는 안쪽 루프로 간주한다.

(6) 모서리(edge)

모서리는 경계 지워지고 스스로 닫히지 않는 곡선을 의미하며 방향성을 갖는다. 모서리는 곡선의 시작점과 끝점으로 경계 지워지고 시작점에서 끝점을 향하는 방향을 갖는다. 이 모서리의 방향은 인접한 면들의 순서를 정할 때 기준이 된다.

(7) 꼭지점(vertex)

꼭지점은 곡선의 시작점과 끝점을 의미한다. 그림 4.4에서와 같이 공간상에 독립적으로 존재하는 정점도 꼭지점이며, 이 꼭지점은 면이나 쉘의 경계로 취급된다.

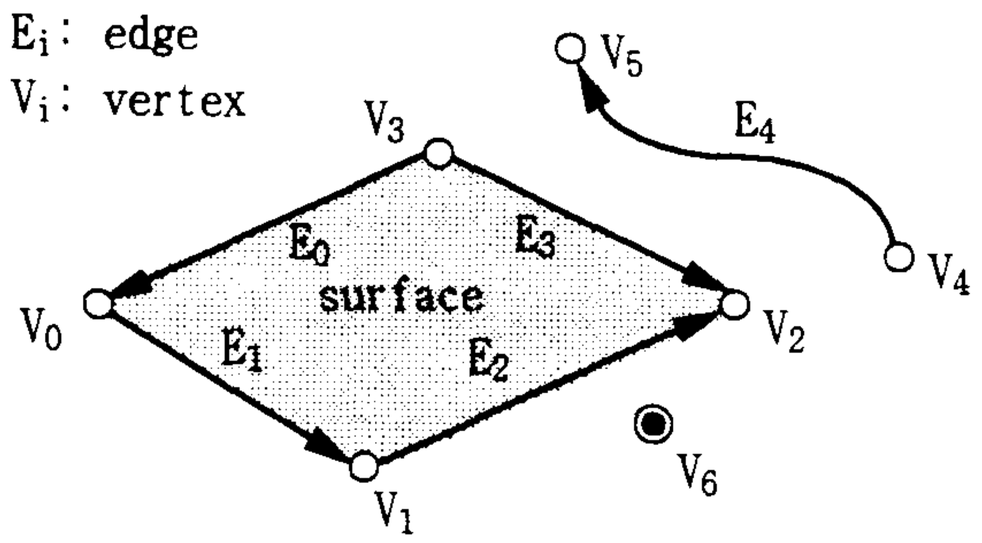
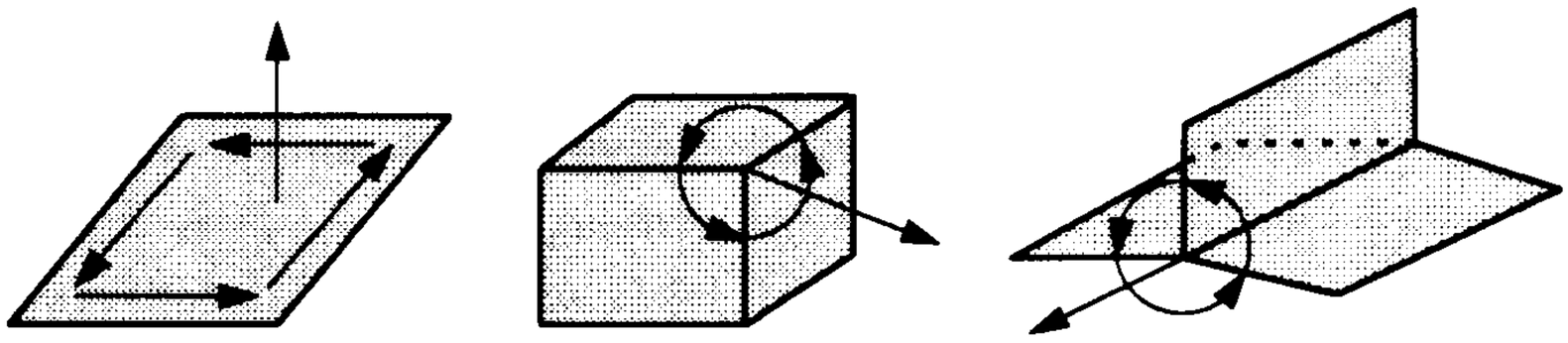


그림 4.4 모서리와 꼭지점의 관계
(Relation between edge and vertex)

나. 인접 관계의 표현

비다양체 모델을 표현하기 위해서는 그림 4.5와 같이 3가지 인접 관계를 표현할 수 있어야 한다[4-4]. 그림에서 (a)는 면의 경계를 나타내는 모서리들의 연결 관계를 표현하는 것으로 루프 순환(loop cycle)이라 한다. (b)는 하나의 꼭지점에 연결된 모서리들의 연결 관계를 표현하는 디스크 순환(disk cycle)이고, (c)는 하나의 모서리를 공유하는 면들의 연결 관계를 표현하는 방

사 순환(radial cycle)이다. 방사 순환은 (c)와 같이 모서리의 방향을 기준으로 반 시계 방향으로 면들의 연결 순서를 표현한다. 다양체 모델은 (a)와 (b)만으로 표현할 수 있으나, 비다양체 모델은 하나의 모서리를 여러 개의 면이 공유함으로 (c)가 필요하다. 또한, 다양체 모델일 경우에는 각 꼭지점에 대하여 디스크 순환이 단 하나만 존재하지만, 비다양체 모델은 여러 개가 존재할 수 있다.



(a) Loop cycle (b) Disk cycle (c) Radial cycle

그림 4.5 모델 표현의 3가지 순환 (Three cycles in model representation)

다. 모델 표현의 예

모델의 위상을 일관된 형태로 표현하는 연구는 Winged-edge, Half-edge, Radial-edge 구조 등으로 발전되어, 현재는 비다양체를 모델링할 수 있는 상용 형상 모델링 시스템의 데이터 구조도 개발되고 있다. 그러므로 기존에 개발된 구조를 통하여 형상 모델의 데이터 구조와 개념을 이해할 수 있으므로 설명을 하고자 한다. Winged-edge와 Half-edge 구조는 다양체 만을 표현할 수 있는 데이터 구조이고, Radial-edge 구조는 비다양체도 표현할 수 있는 데이터 구조이다. Winged-Edge 데이터 구조[4-5]는 그림 4.6과 같이 solid, face, loop, edge, vertex등과 같은 위상 요소를 계층적으로 표현한 구조이다. 이 구조는 임의의 모서리를 중심으로 이 모서리에 연결되는 4개의 모서리를 표현하여 모델의 위상 정보를 표현하였다. 그러나 하나의 모서리가 양쪽의 면에 의해 공유됨으로 데이터의 중복이 발생하고, 구현 프로그램이 복잡한 단점이 있다.

Half-edge 데이터 구조[4-6]는 Winged-edge의 단점을 해결하기 위하여 그림 4.7과 같이 하나의 모서리를 두개의 HalfEdge로 나누어 표현하여 모서리의 정보가 중복되지 않도록 하였다.

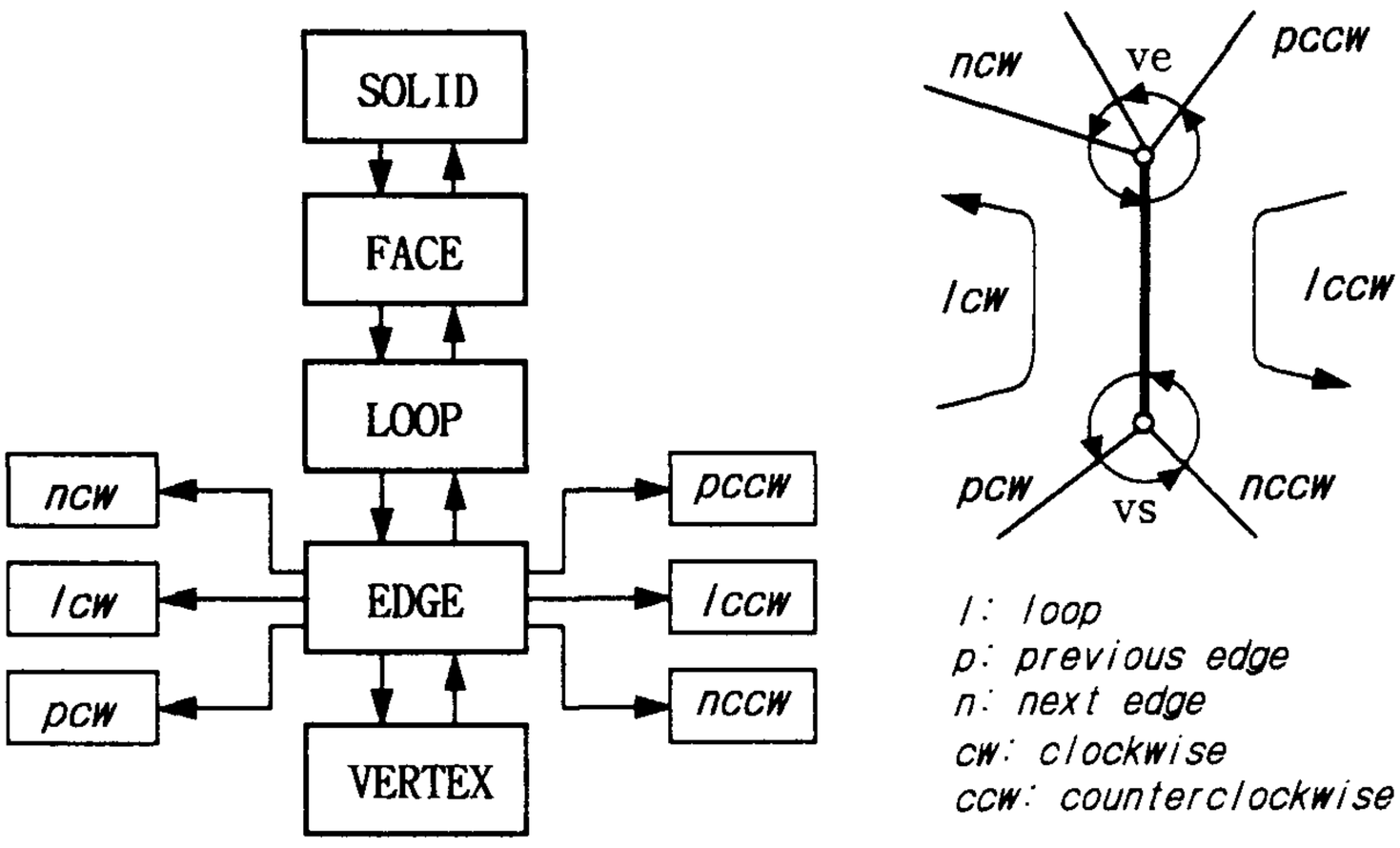


그림 4.6 Winged-edge data structure

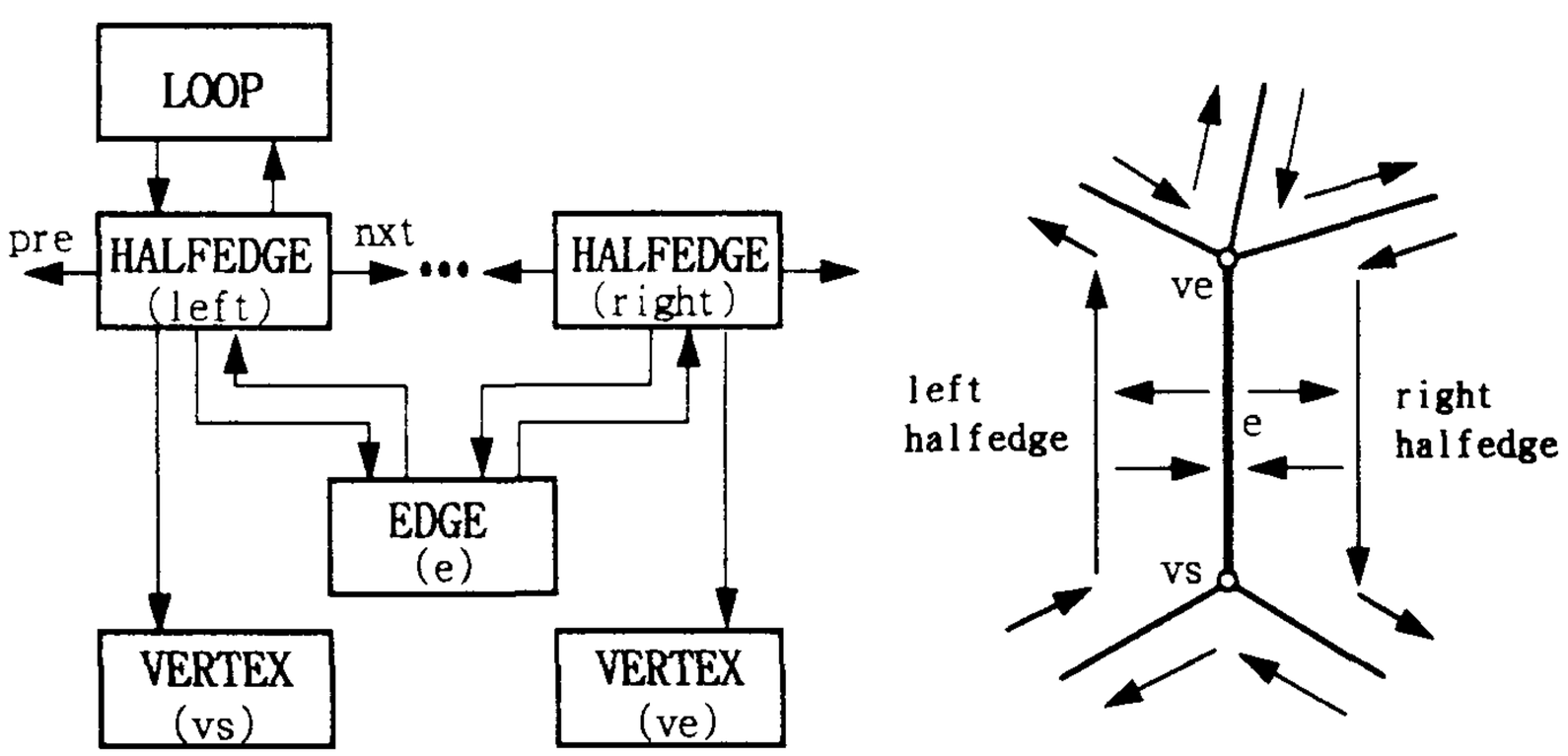


그림 4.7 Half-edge data structure

비다양체 모델의 위상을 표현하기 위해서는 그림 4.2와 같이 하나의 형상 요소가 위상적으로 두 번 표현된다. Weiler는 이러한 비다양체 모델의 특성과 면의 인접 관계를 나타내는 방사 순환을 표현하기 위하여 그림 4.8과 같이 face-

use, loop-use, edge-use, vertex-use 등과 같은 보조 위상 요소를 도입하여 radial-edge 데이터 구조[4-7]를 개발하였다. 이 구조는 공간상에 독립된 정점은 그림 4.8의 ①과 같이 꼭지점 쉘(vertex shell)로, 곡면상에 존재하는 정점은 ②와 같이 꼭지점 루프(vertex loop)로, 공간상에 독립적으로 존재하는 곡선은 ③과 같이 와이어 모서리(wireframe edge)로 표현하였다.

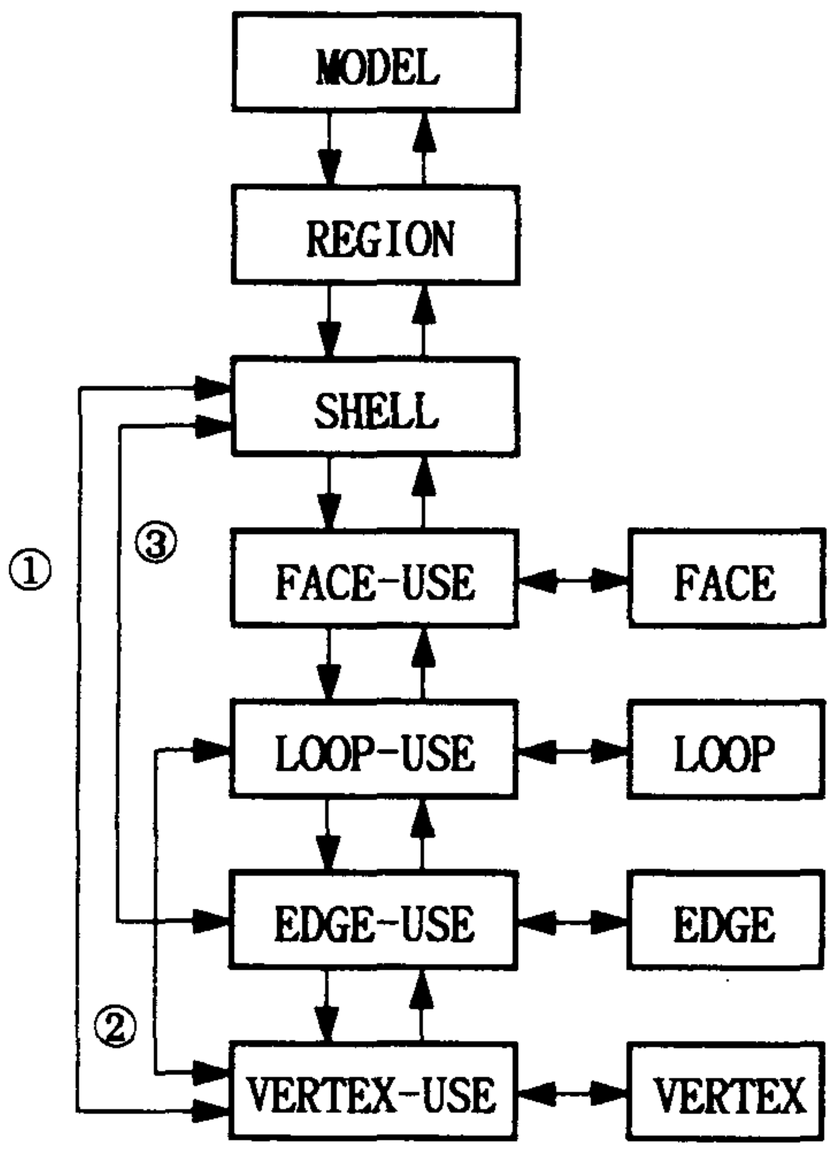


그림 4.8 Radial-edge data structure

Face-use는 쉘을 구성하는 면을 나타내는 요소로, 하나의 면이 두 번 사용되므로 또 다른 face-use를 가르키는 face-use mate pointer를 갖고 있다. Loop-use는 face-use의 경계를 나타내는 요소로 그림 4.9와 같이 면의 경계로 사용되는 edge-use를 연결하여 루프 순환(loop cycle)을 표현한다. Edge-use는 모서리가 loop-use로 사용됨을 나타내는 요소로서 그림 4.10과 같이 방사 순환을 표현하기 위해 edge-use radial pointer와 edge-use mate pointer를 갖고 있다. Edge-use radial pointer는 모서리를 공유하는 면들의 공유 모서리를 나타

내는 edge-use들을 연결하는 pointer이고, edge-use mate pointer는 동일한 모서리를 사용하는 2개의 edge-use를 서로간에 연결시키는 pointer이다. Vertex-use는 edge-use의 경계를 표현하는 요소로 그림 4.11과 같이 하나의 꼭지점에 연결된 모든 모서리의 edge-use들을 vunext와 vulast pointer로 연결하여 디스크 순환을 표현한다.

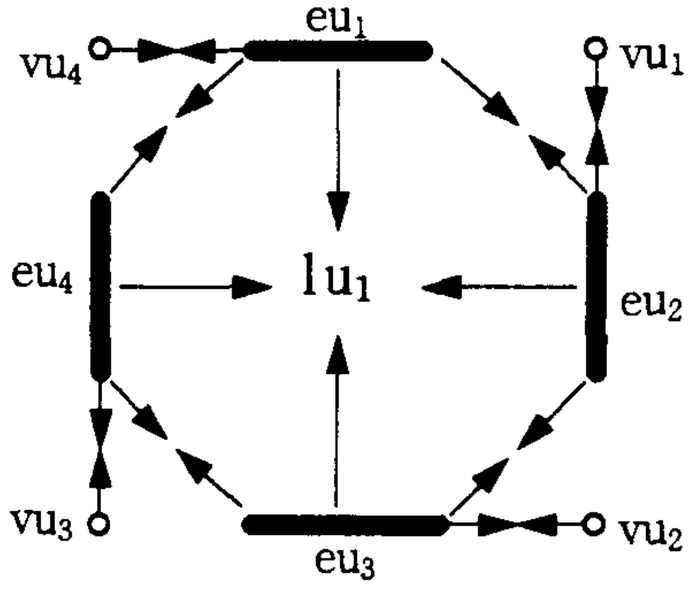


그림 4.9 Loop cycle in radial-edge data structure

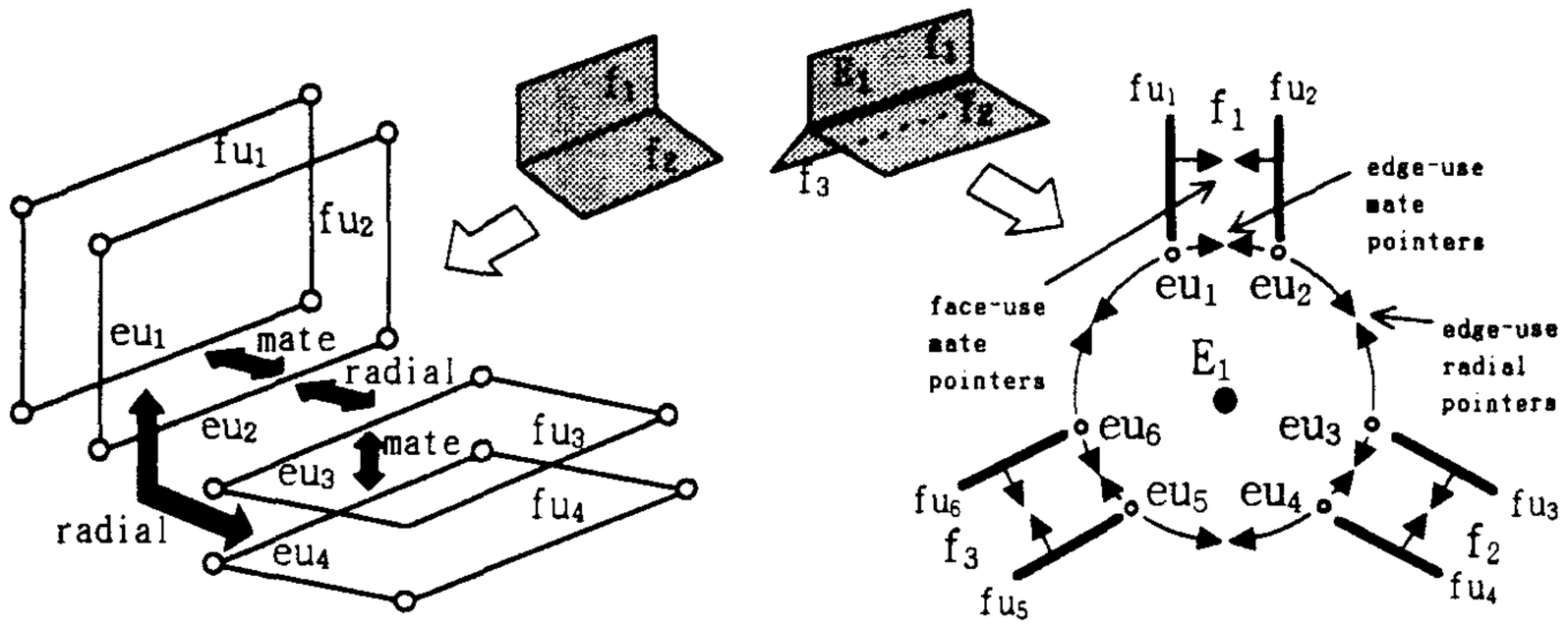


그림 4.10 Radial cycle in radial-edge data structure

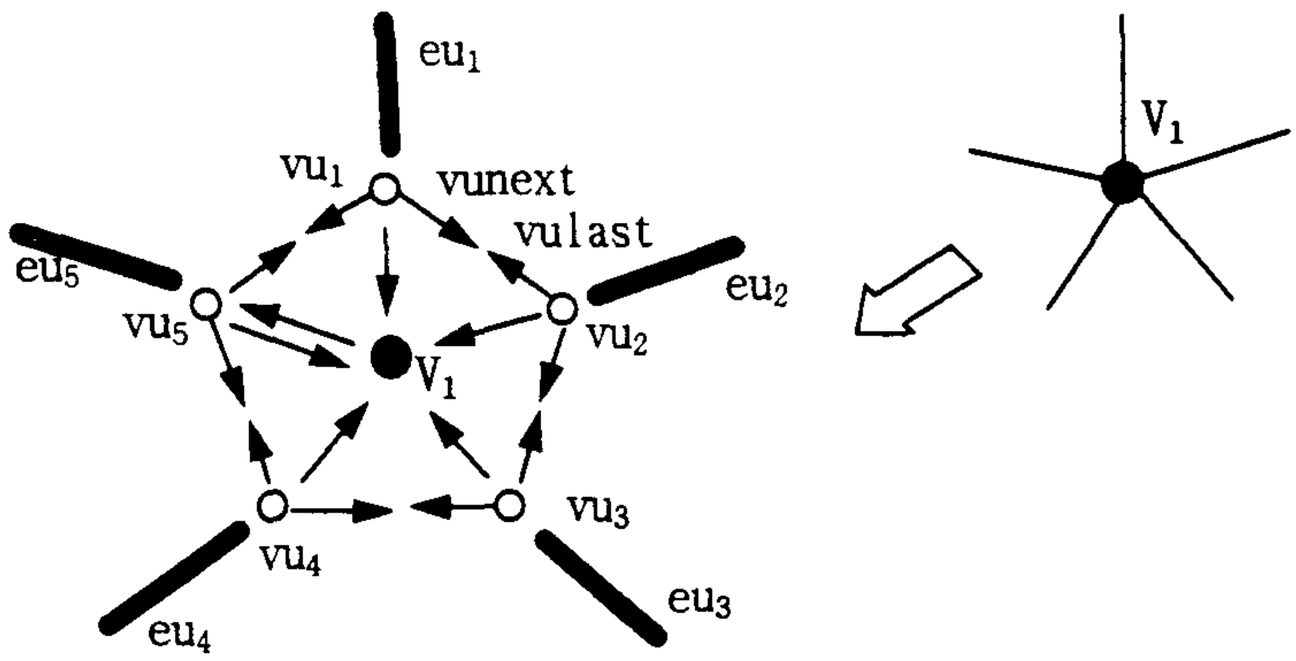


그림 4.11 Disk cycle in radial-edge data structure

3. 기존 데이터 구조

현제까지 제안된 비다양체 모델의 데이터 구조 중 대표적인 것은 앞에서 설명된 Radial-edge와 Vertex-based B-rep, Compact B-rep, ACIS 등이 있다. Radial-edge 데이터 구조는 모서리에서 면들의 인접 관계를 표현하는데 중점을 둔 구조로 방사 순환 정보를 직접적으로 갖고 있으므로 하나의 모서리를 공유하는 면들을 쉽게 추출 할 수 있다. 그러나 face-use가 loop-use를 통해 자신의 경계를 완전하게 저장함으로 단지 방향만 반대인 면을 이중으로 저장하여 데이터 양이 증가된다. 또한, 하나의 꼭지점을 다수의 영역이 공유할 경우에는 꼭지점에서 영역간의 인접 관계를 추출하기가 어려운 단점이 있다. 이러한 단점을 해결하기 위하여 Choi는 Vertex-based B-rep[4-8]을 제안 하였다. 이 구조는 zone과 disk라는 보조 위상 요소를 도입하여 꼭지점에서의 직접적으로 저장한다. 따라서 모델의 모서리와 꼭지점에서 비다양체적 특징을 모두 수용할 수 있고, 하나의 꼭지점을 여러 개의 영역이 공유할 경우에도 인접 관계를 쉽게 얻을 수 있으나, 데이터 구조가 복잡한 단점이 있다. Lee등은 Radial-edge의 단점인, 데이터 중복으로 인한 데이터 양의 증가와 Vertex-based B-rep의 단점인 데이터 구조의 복잡성을 해소 하고자, 면을 표현의 중심으로 하여 Compact B-rep[4-4]을 제안하였다. Compact B-rep의 기본 위상 요소는 Radial-edge와 동일하고 면과 영역의 관계를 나타내는 부분 면(partial face), 모서리와 면의 관계를 나타내는 부분 모서리(partial edge), 그리고 꼭지점과 모서리의 관계를 표현하는 부분 꼭지점(partial vertex) 등의 부분 위상 요소(partial topological entity)를 도입하였다. 이 구조는 Radial-edge에서 face-use 대신에 부분 면을 사용하고, loop-use를 제거하여 면의 경계를 루프로 표현함으로써 부분 면이 사용될 때마다 면의 경계가 중복적으로 저장되지 않도록 하였다. Spatial Technology사에서 상용화한 ACIS[4-9]의 데이터 구조

도 비다양체 모델을 수용할 수 있다. ACIS의 기본 위상 요소는 타 구조와 유사하나, 공간상의 곡선을 표현하기 위하여 wire라는 요소를 추가하여 하였다. 또한, 보조 위상 요소로 부분 면과 유사한 coedge만을 도입하여 데이터 구조가 간결하나 부분 면과 같은 위상 요소가 없어 데이터의 중복이 발생하는 단점도 갖고 있다.

제 3 절 SurfART의 데이터 구조

1. 데이터 구조의 설계

곡면 모델링 과정은 점, 곡선, 곡면 등이 혼재하는 비다양체 모델이 생성된다. 그러므로 이러한 모델을 관리하기 위해서는 비다양체 모델을 수용할 수 있는 데이터 구조가 필요하다. 그러나 곡면 모델링은 완전한 솔리드 모델을 생성하지 않고 대상의 곡면 형상만 표현할 수 있으면 되므로 영역(region)의 개념은 사용되지 않으며, 하나의 곡면이 위상적으로 두 번 사용되지 않아도 된다. 또한, 하나의 꼭지점을 다수의 영역이 공유하는 상황도 발생하지 않으므로 완전한 디스크 순환을 표현할 필요가 없다. 그러나 본 연구에서는 솔리드 모델링으로의 확장성을 고려하여 솔리드 모델도 표현할 수 있도록 설계하였으나, 면의 정보가 중복되는 단점은 갖고 있다. 그림 4.12는 본 연구에서 개발한 SurfART 곡면 모델링 시스템의 데이터 구조를 계층적으로 표현한 것으로 방사 순환을 표현하기 위하여 ACIS와 유사한 coedge를 사용하고, 디스크 순환은 각각의 꼭지점에 연결되는 모서리를 관리하는 멤버 객체로 표현하였다. 또한, 모델링 과정에서는 공간상에 고립된 정점, 어떤 곡면과도 연결되어 있지 않은 곡선, 곡면상에 존재하며 어떤 곡선과도 연결되어 있지 않은 정점 등이 빈번히 발생함으로 Compact-Brep과 같이 불필요한 위상 요소를 사용하지 않고 각각을 외 꼭지점 쉘(single vertex shell), 와이어 모서리(wire edge), 외 꼭지점 루프(single vertex loop) 등으로 표현 하였다. 또한, SurfART 시스템은 곡선이나 곡면을 생성하기 위하여 측정기로부터 입력된 많은 점 데이터를 이용하게 된다. 그러나 이러한 점 데이터는 곡선, 곡면 등을 생성한 후에는 삭제되는 임시적인 데이터이므로 외 꼭지점 쉘과 구별하여 형상 정보만을 관리할 수 있도록 하였다.

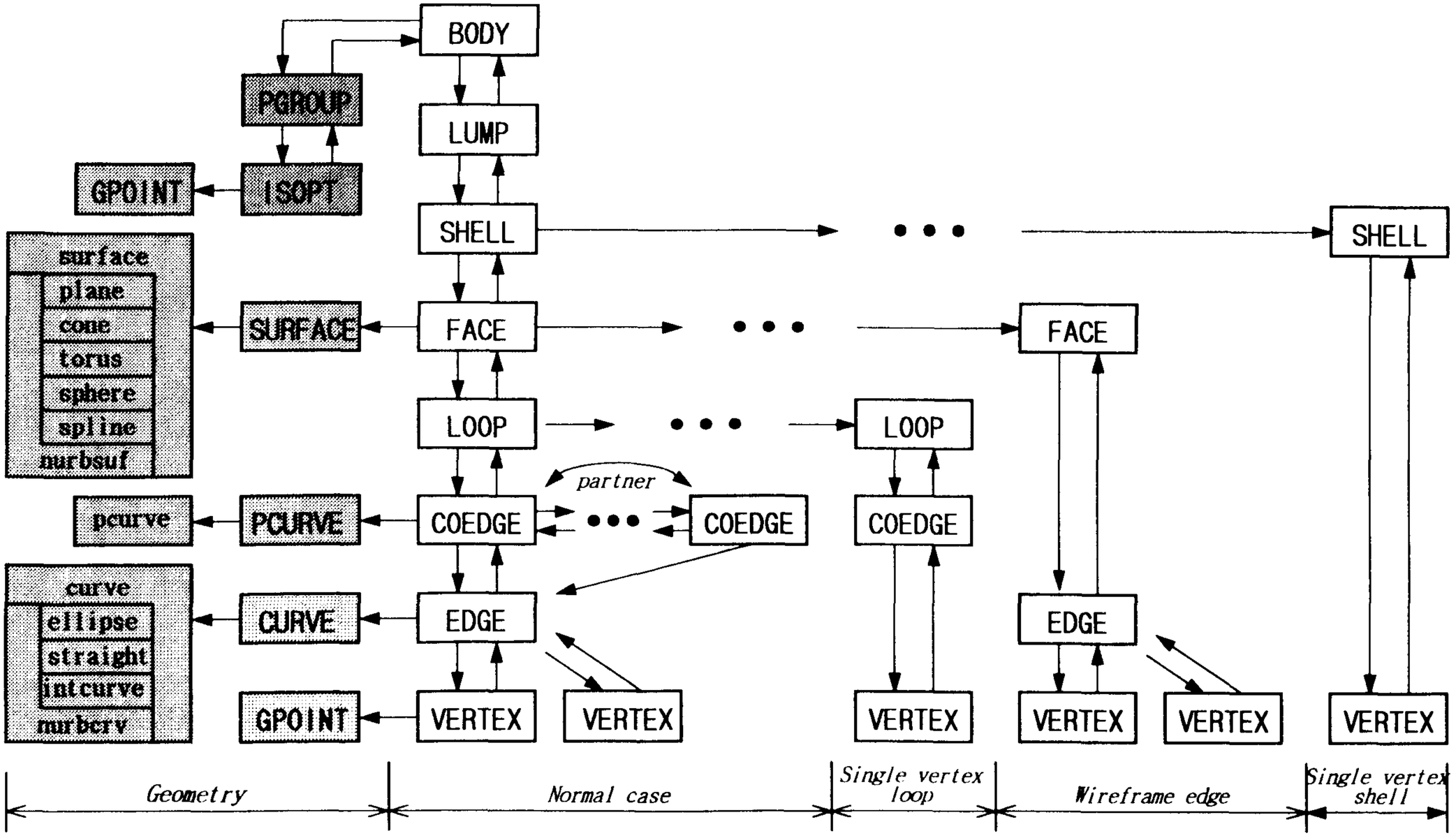


그림 4.12 SurfART의 데이터 구조 (The data structure of SurfART system)

2. 위상 요소의 정의

본 데이터 구조에서 모델의 위상 정보를 표현하기 위하여 사용되는 위상 요소는 ACIS와 유사하나, ACIS에서 곡면과 연결이 없는 곡선을 표현하기 위해 사용되었던 WIRE와 SUBSHELL 등은 없다. 또한, 시스템의 특성 상 임시적인 점 데이터들을 많이 사용함으로 이들을 꼭지점 쉘과 구분하여 표현하기 위하여 PGROUP과 ISOPT를 도입하였다. SurfART 시스템의 데이터 구조에서 사용하는 위상 요소의 정의는 다음과 같다.

가. BODY

최상위 요소로 위상 모델링 공간에 포함된 모든 위상 요소의 저장고 역할을 하며, 하나 이상의 LUMP로 구성된다. SurfART 시스템에서 독립된 하나의 모델을 관리하는 단위이다.

나. LUMP

공간에 존재하는 닫혀진 볼륨으로 그림 4.13와 같이 다른 LUMP의 허 공간(void)에도 포함될 수 도 있다. 모델링을 시작하는 시점에는 가상의 경계로 닫혀지고 무한대의 크기의 볼륨을 갖는 LUMP가 존재한다.

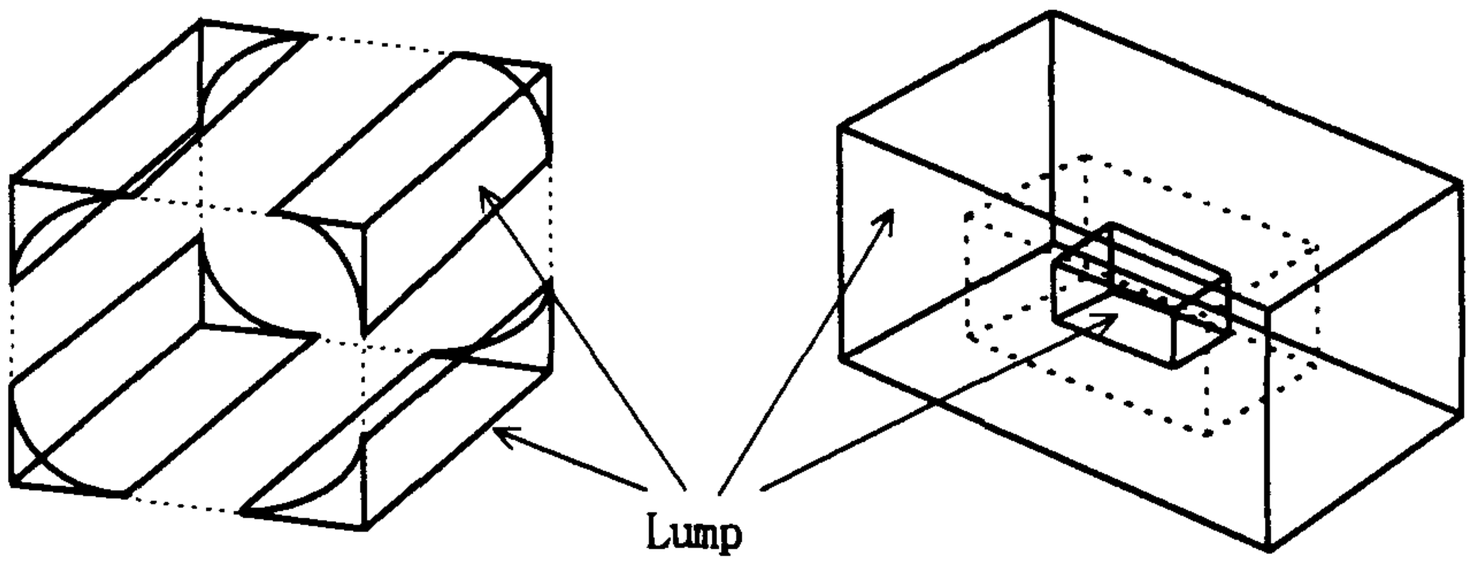


그림 4.13 Lumps in example model

다. SHELL

연결된 FACE들의 집합으로 다른 SHELL과 연결이 없어야 한다. SHELL은 open될 수도 있으며 LUMP의 허공간에도 존재할 수 있다. 모델링 작업으로 closed된 SHELL이 발생되면 이 SHELL은 새로운 LUMP를 생성하게 된다. 공간상에 독립적으로 존재하는 점 하나로도 SHELL을 구성할 수 있다.

라. FACE

공간상에 정의된 곡면을 표현하는 요소로 방향성을 갖는다. 곡면은 스스로 교차되지 않아야 하며 경계 지워져야 한다. FACE의 방향은 해당 FACE로 구성된 SHELL이 경계하는 LUMP의 바깥쪽을 향하므로, 이 방향이 곡면의 법선(normal) 방향과 동일한지, 반대 인지로 나타낸다. 또한, 솔리드 모델과 박판 모델의 곡면을 구분하기 위하여 sidebit 멤버를 갖고 있으며 각각을 single-sided와 double-sided로 표현한다. 곡면과 연결이 없는 곡선은 곡면이 퇴화된(degenerated) 것으로 간주하여 방향성이 없는 하나의 FACE로 표현한다.

마. LOOP

연결된 경계 곡선들을 표현하는 요소로 방향성을 갖으며 open될 수도 있다. LOOP는 FACE의 경계를 표현하는 요소로 FACE의 외부를 경계하는 LOOP를 외부 루프(outer loop), 내부를 경계하는 LOOP를 내부 루프(inner loop)라 한다. 외부 루프는 곡면의 법선 방향을 기준으로 반 시계방향을 갖고 내부 루프는 시계방향을 갖는다. 곡면상에 존재하는 점은 그림 4.14의 L_2와 같이 방향성이 없는 LOOP로 표현한다.

바. COEDGE

경계 곡선을 공유하는 곡면의 수 만큼 경계 곡선을 구분하여 표현하

는 요소로 LOOP를 구성하며 LOOP의 따른 방향성을 갖는다. COEDGE는 Radial-edge 데이터 구조의 edge-use와 기능적으로 유사하며, 그림 4.14에서와 같이 동일한 경계 곡선을 표현하는 COEDGE C_2와 C_8을 partner 관계로 연결하여 방사 순환을 표현하는 요소이다.

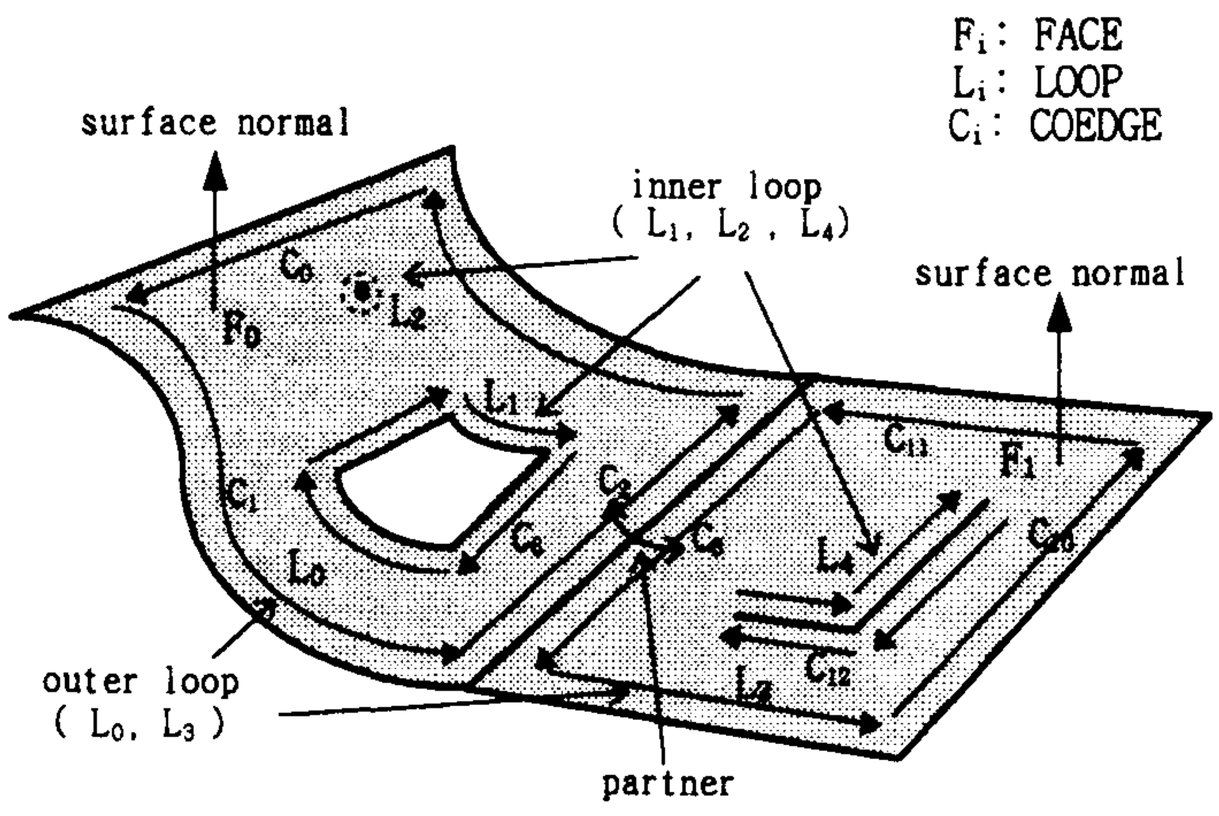


그림 4.14 Relations among FACE, LOOP and COEDGE

사. EDGE

곡선을 표현하는 요소로 방향성을 갖고있다. 곡선은 스스로 교차되지 않아야 하며 항상 양 끝점으로 경계 지워져야 한다. EDGE의 방향은 시작 VERTEX에서 끝 VERTEX를 향하며, 이 방향이 곡선의 방향과 동일하면 forward로, 반대면 reversed로 나타낸다.

아. VERTEX

모델에 정의되는 점을 표현하는 요소로 모델 내에서 유일해야 한다. 곡선의 양 끝점, 곡면 위에 있는 점, 공간상에 독립된 점 등이 위상 요소 VERTEX로 표현된다.

자. PGROUP

그룹(group)으로 나누어진 점 데이터들을 표현하는 요소로 위상 개념은 없고, 그림 4.15와 같이 그룹이 입력되는 순서대로 연결하여 표현한다. 그룹화가 되지 않은 점 데이터는 하나의 PGROUP으로 표현한다

차. ISOPT

측정된 점을 표현하는 요소로 측정기로부터 입력된 점과 꼭지점 셀을 구분하기 위하여 사용된다. ISOPT는 그림 4.15와 같이 점 데이터가 입력되는 순서대로 연결하며 하나 이상의 ISOPT가 PGROUP을 생성한다.

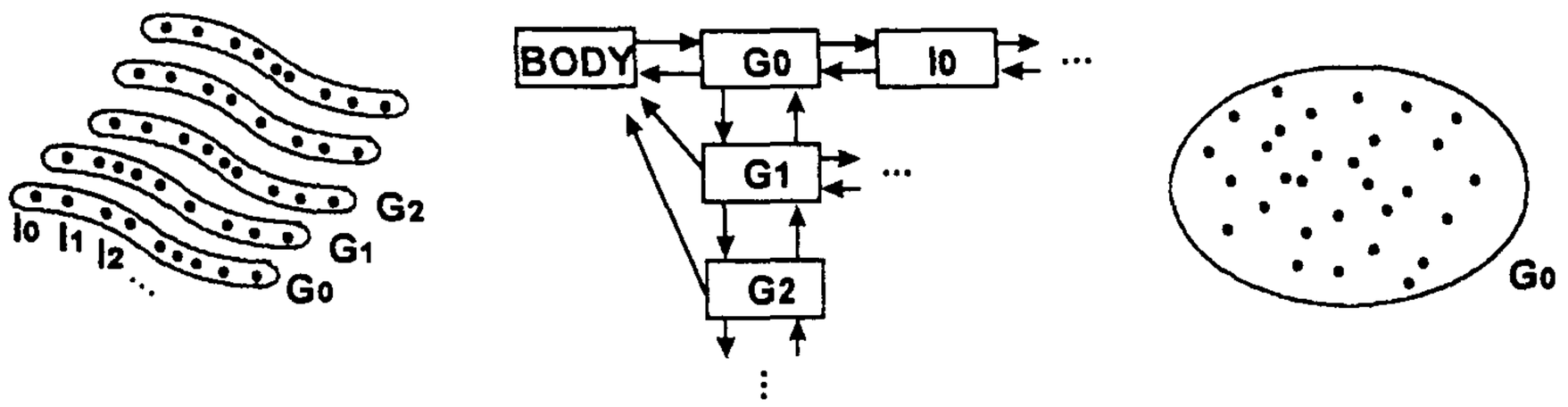


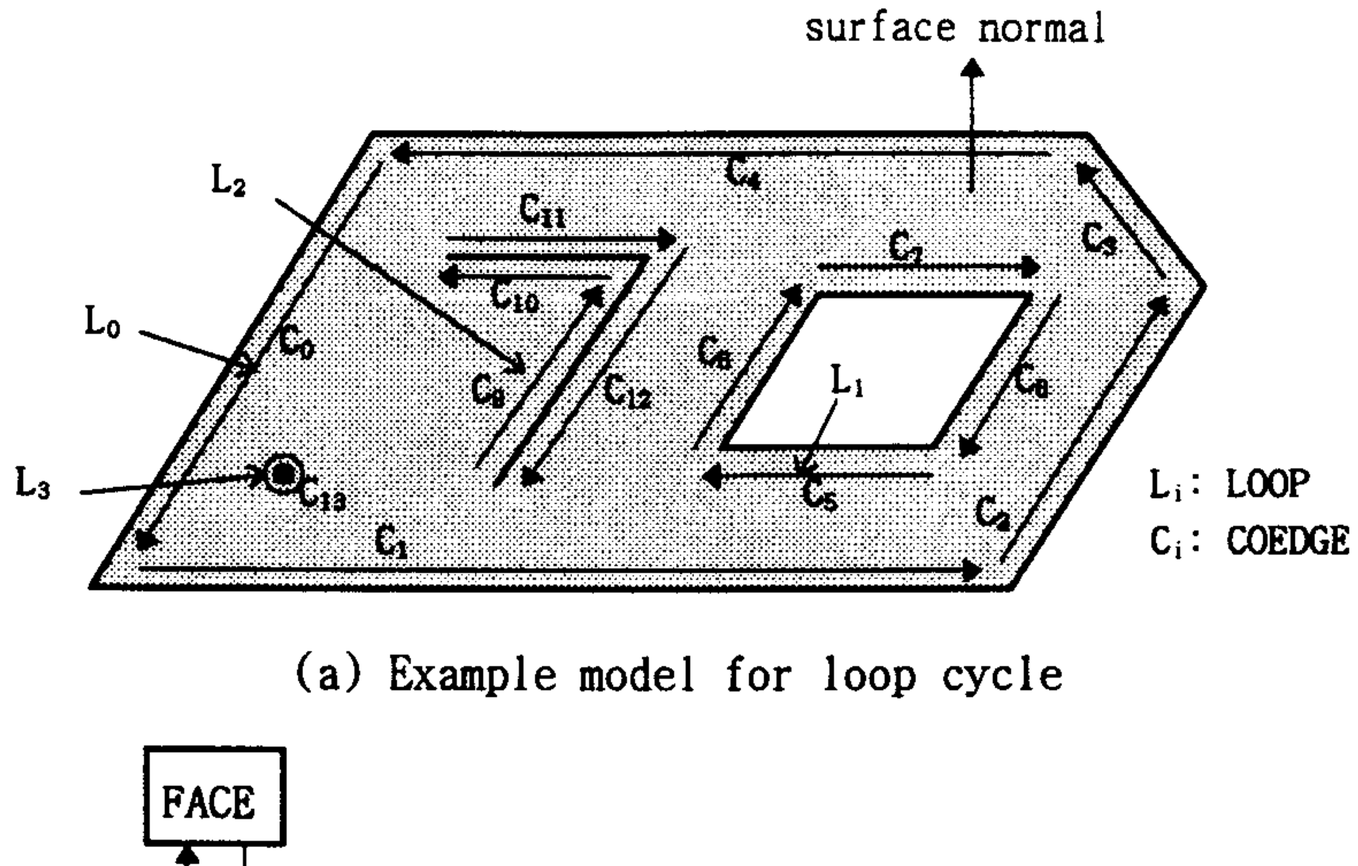
그림 4.15 Definition of PGROUP and ISOPT

3. 순환(cycle)의 표현

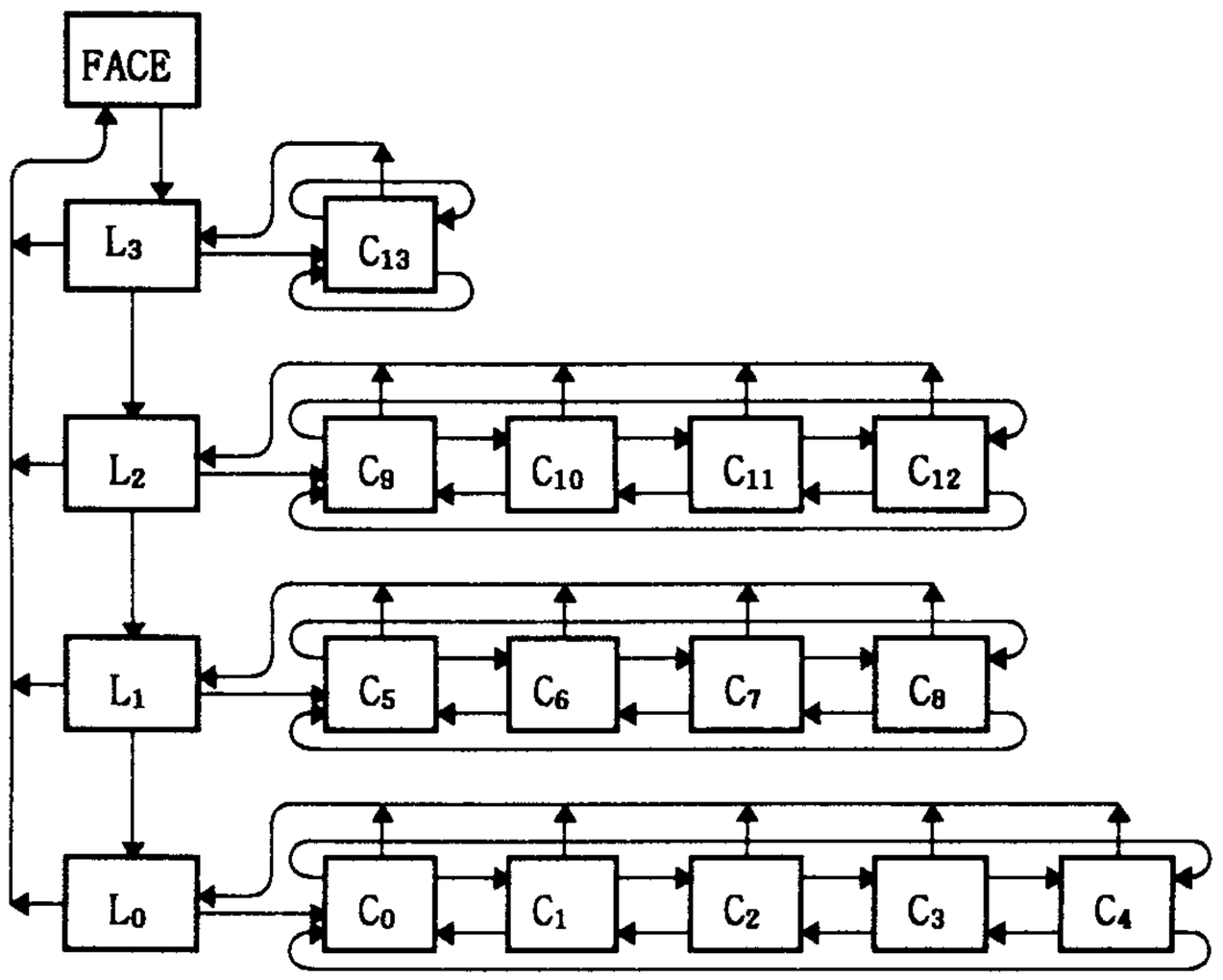
비다양체 모델을 수용할 수 있는 데이터 구조는 루프 순환과 디스크 순환은 물론 방사 순환도 표현할 수 있어야 한다. 루프 순환은 곡면의 경계를 이루는 곡선들의 연결 관계를 표현하는 방법이고 디스크 순환은 하나의 점을 공유하는 곡선이나 곡면들의 연결 관계를 표현하는 방법이다. 또한, 방사 순환은 하나의 경계 곡선을 공유하는 곡면들의 연결 관계를 표현하는 기법이다. 본 데이터 구조도 이와 같은 3개의 순환을 표현할 수 있도록 되어있다. 루프 순환과 방사 순환은 모든 순환 정보를 데이터 구조에 직접적으로 표현한다. 그러나 디스크 순환은 모델링 과정에서 많이 발생하지 않으므로, 모든 정보를 완전하게 표현하는 Vertex-based B-rep과는 달리 일부 정보만 직접적으로 표현하고 완전한 순환 정보는 이들을 이용하여 추출하는 방법을 사용하였다. 본 데이터 구조에서 3가지 순환을 표현하는 방법은 다음과 같다.

가. 루프 순환(loop cycle)

루프 순환은 FACE의 경계를 표현하는 방법으로 그림 4.16과 같이 곡면상의 곡선을 표현하는 COEDGE들을 연결하여 표현한다. COEDGE는 하나의 곡선이 곡면의 경계로 사용됨을 나타내는 요소로 루프의 종류에 따라 방향이 결정된다. 루프는 외부 루프(그림 4.16에서 L_0)이면 곡면의 법선에 대하여 반 시계 방향, 외부 루프(그림 4.16에서 L_1, L_2)이면 시계 방향을 갖으므로 해당 루프를 구성하는 모든 COEDGE는 루프의 방향과 동일한 방향을 갖는다. 곡면상에 존재하는 점도 방향성이 없는 하나의 COEDGE로 구성된 루프(그림 4.16에서 L_3)이고, 곡면상에 존재하며 곡면의 경계로 사용되지 않는 곡선은 그림 4.16의 L_2와 같이 하나의 곡선을 표현하는 2개의 COEDGE들을 연결하여 open된 루프로 표현한다.



(a) Example model for loop cycle



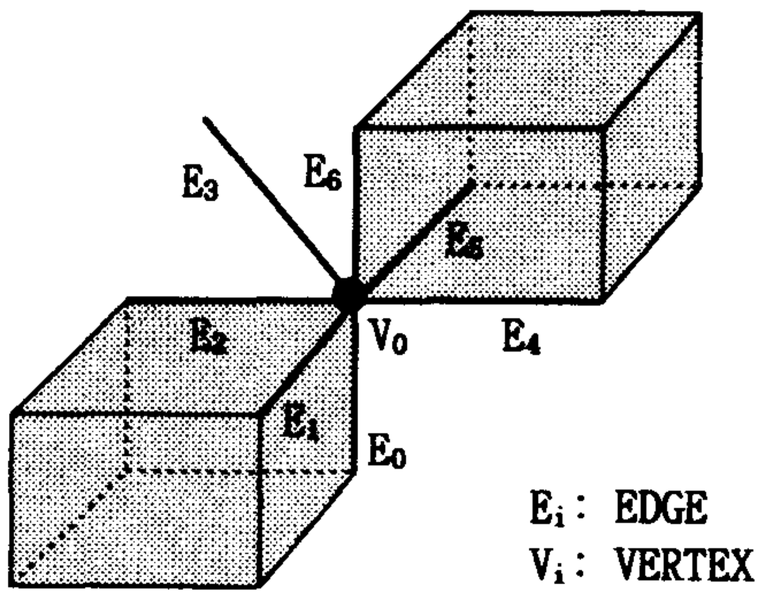
(b) Representation of topological entities for loop cycle

그림 4.16 Representation of loop cycle in data structure

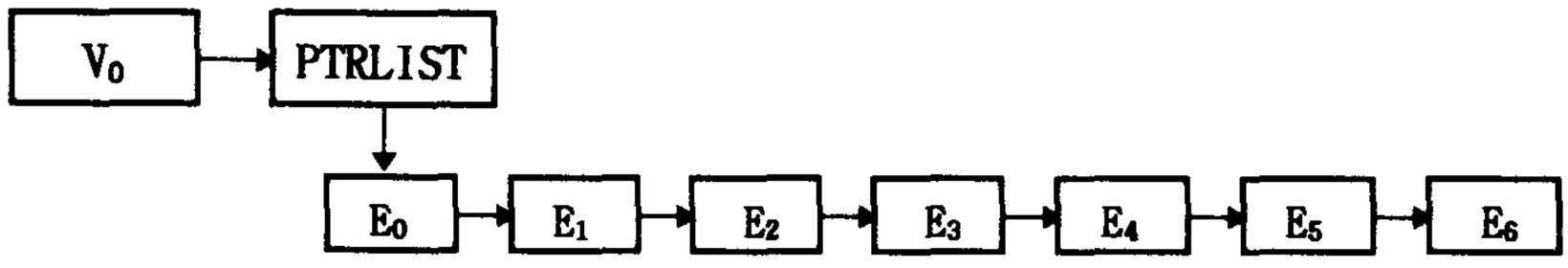
나. 디스크 순환(disk cycle)

디스크 순환은 하나의 VERTEX를 공유하는 EDGE, FACE, LUMP등을 데이터 구조에 직접적으로 표현하여 이들간의 연결 관계를 쉽게 추출할 수 있게 하는 순환이다. 본 데이터 구조에서는 위상 요소를 리스트로 관리할 수 있는 PTRLIST 클래스를 VERTEX의 멤버로 사용하여 그림 4.17과 같이 VERTEX V_0를 공유하는 모든 EDGE들의 포인터(pointer)를 직접적으로 저장하도록 되었다. 그러

므로 임의의 VERTEX를 LUMP가 공유하는 경우에는 저장된 EDGE를 이용하여 해당 EDGE의 LUMP를 쉽게 찾을 수 있으며, 이러한 기능을 수행하는 API가 있다.



(a) Example model for disk cycle



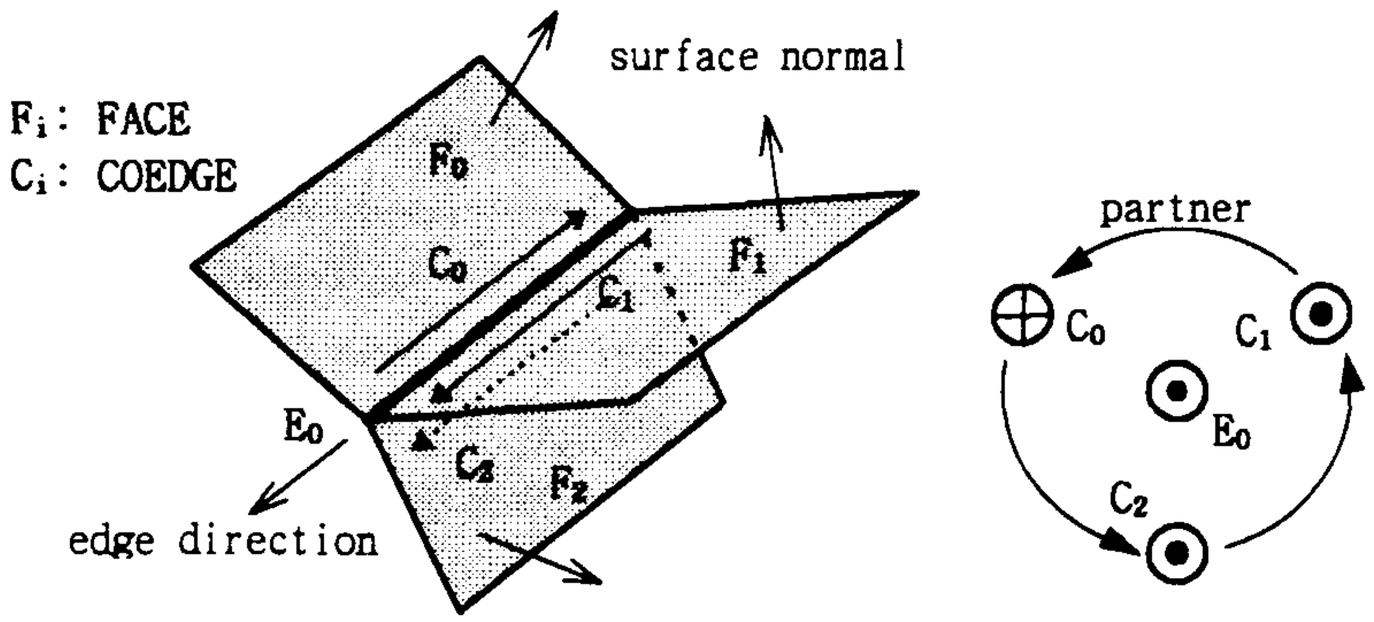
(b) Representation of topological entities for disk cycle

그림 4.17 Representation of disk cycle in data structure

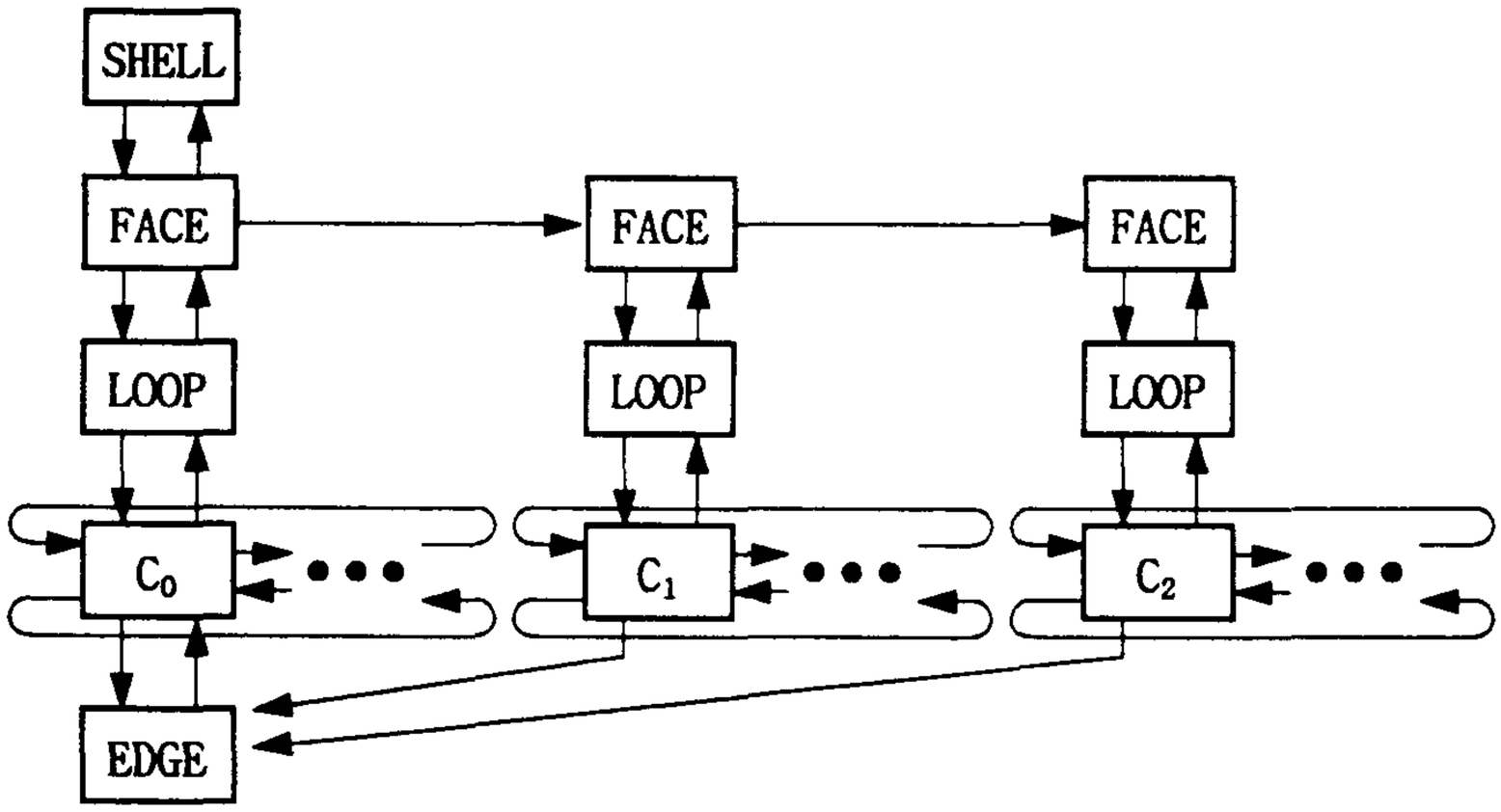
다. 방사 순환(radial cycle)

방사 순환은 하나의 모서리를 공유하는 FACE들을 데이터 구조에 직접적으로 표현하여 곡면들간의 연결 관계를 추출하는데 사용하는 순환이다. 본 데이터 구조에서는 방사 순환을 표현하기 위해 COEDGE를 사용한다. COEDGE 요소는 EDGE가 LOOP의 구성원임을 나타내는 요소로 멤버로 파트너 COEDGE를 갖도록 되어 있다. 파트너 COEDGE는 EDGE의 방향을 기준으로 오른손 법칙에 따라 가장 가까이에 있는 곡면의 LOOP를 구성하는 COEDGE이다. 또한, 공유되지 않는 EDGE의 COEDGE는 파트너가 없으며(null), EDGE가 곡면상에 존재하는 경우에는 해당 EDGE가 LOOP의 구성원임을 나타내는 2개의 COEDGE가 서로간의 파트너 COEDGE이다. 그림 4.18은 EDGE E_0를 공유하는 3개의 곡면들을 방사 순환으로

표현한 것으로 FACE F_0의 경계로 사용되는 C_0, F_1의 경계로 사용되는 C_1, F_2의 경계로 사용되는 C_2가 그림 4.18의 (a)와 같이 파트너 관계로 연결된다. 그러므로 임의의 FACE에 연결된 곡면들의 정보가 필요한 경우에는 해당 FACE의 모든 COEDGE에 대하여 파트너 COEDGE를 얻고 파트너 COEDGE의 FACE를 찾으면 되므로 형상 정보를 이용한 수치적 계산이 없이 빠르게 얻을 수 있다.



(a) Example model for radial cycle



(b) Representation of topological entities for radial cycle

그림 4.18 Representation of radial cycle in data structure

4. 형상 요소의 표현

형상 모델링 시스템에서 모델을 표현하기 위해서는 형상 요소간의 인접 관계를 나타내는 위상 정보뿐만 아니라 모델의 모양(shape)을 결정하는 형상 요소를 정의할 수 있는 방법과 이들을 위상 요소와 연결시켜 표현할 수 있는 데이터 구조가 필요하다. 형상 모델링 시스템에서 형상 요소란 공간상에 정의되는 점(point), 곡선(curve), 곡면(surface) 등을 의미한다. 모델링 시스템에서 곡선은 3차원 공간상에서 정의되는 3차원 곡선과 곡면의 파라메트릭 도메인(parametric domain)에서 정의되는 도메인 곡선이 사용된다. 곡선과 곡면을 수학적으로 정의하는 방식은 많이 있으나 본 데이터 구조에서는 NURBS를 사용하였다. NURBS는 저장해야 할 데이터가 많다는 단점은 있으나, 자유 곡선, 곡면은 물론, 구면(spherical surface), 원통면(cylindrical surface)과 같은 해석 곡선, 곡면(analytic surface)도 정확하게 정의할 수 있고, 정의된 형상을 변형할 수 있는 요소를 가장 많이 갖고 있는 방식이므로 형상 모델링 시스템에서 곡선, 곡면의 표현식으로 많이 사용된다. 또한, NURBS는 OpenGL과 같은 상용 그래픽 라이브러리에서 곡면 표현식으로 사용되므로 형상의 그래픽 디스플레이가 편리한 장점이 있다. 모델링 시스템의 데이터 구조에서 형상 요소들은 위상 요소와 일정한 관계를 갖고 있으며 데이터 구조 상에 직접적으로 표현된다. 본 데이터 구조에서도 형상 요소와 위상 요소간의 관계를 표현하기 위하여 그림 4.12에서와 같이 중간적인 요소(geometric entity)로 GPOINT, CURVE, PCURVE, SURFACE등을 사용하고 있다. 이들은 정의된 형상 요소와 일대일 관계를 갖고, 위상 요소와는 일대다 관계를 갖으므로 임의의 형상 요소가 몇 번 사용되었는지를 보관한다.

가. 점(point)

점은 3차원 좌표값으로 정의되는 요소로 본 데이터 구조에서는 점 데이터를 2가지로 구분한다. 하나는 시스템의 point메뉴를 이용하여 생성한 점이나 곡선의 양 끝점과 같이 위상을 표현하기 위하여 내부적으로 생성되는 점등이 이에 해당된다. 이러한 점은 데이터 구조에서 하나의 VERTEX의 형상 데이터로 표현되며 모델 내에서 유일해야 한다. 다른 하나는 점 데이터를 import하는 기능에 의하여 생성되는 점이다. 이 점 데이터는 위상 개념이 없는 ISOPT의 형상 데이터로 표현된다. 점은 공간상의 위치를 정의하는 position을 데이터로 갖는 GPOINT로 표현되며, 그림 4.19와 같이 VERTEX의 형상 정보로 사용되는 점 데이터와 위상 개념이 없이 ISOPT와 연결되어 형상 데이터만 관리되는 점 데이터 등의 두 가지가 사용된다.

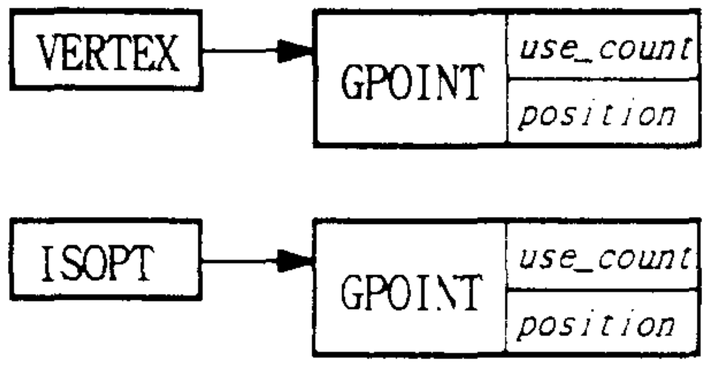


그림 4.19 Representation of point entity

나. 곡선(curve)

본 데이터 구조에서 사용하는 모든 곡선은 NURBS로 정의되며 위상에 따라 두 가지로 구분하여 사용한다. 하나는 곡면의 경계로 사용되지 않는 곡선으로 와이어 모서리의 형상 정보로 사용된다. 이러한 곡선은 모양에 따라 원, 원호, 타원 등을 표현하는 ellipse, 직선을 표현하는 straight, 자유 곡선을 표현하는 intcurve등으로 구분하여 사용하였다. 둘째는 곡면의 경계로 사용되는 곡선으로 정상적인 모서리의 형상 정보로 사용된다. 이러한 곡선은 정확하게 곡면 위에 존재하도록 정의를 해야 하나, 실제적으로 이러한 정의는 불가능하므로 해당 곡면의 u-v 도메인 상에서 곡선을 정의하여 곡면의 경계를 정확하

게 얻을 수 있도록 하였다. 그러므로 곡면의 경계로 사용되는 하나의 곡선은 공간상에서 NURBS로 정의된 3차원 곡선과 곡면의 도메인 상에서 NURBS로 정의된 도메인 곡선 등의 두 가지로 표현되며, 3차원 곡선은 EDGE의 형상 정보로, 도메인 곡선은 COEDGE의 형상 정보로 사용된다. 본 데이터 구조에서 곡면의 경계로 사용되는 곡선은 그림 4.20과 같이 두 가지 방식으로 표현된다.

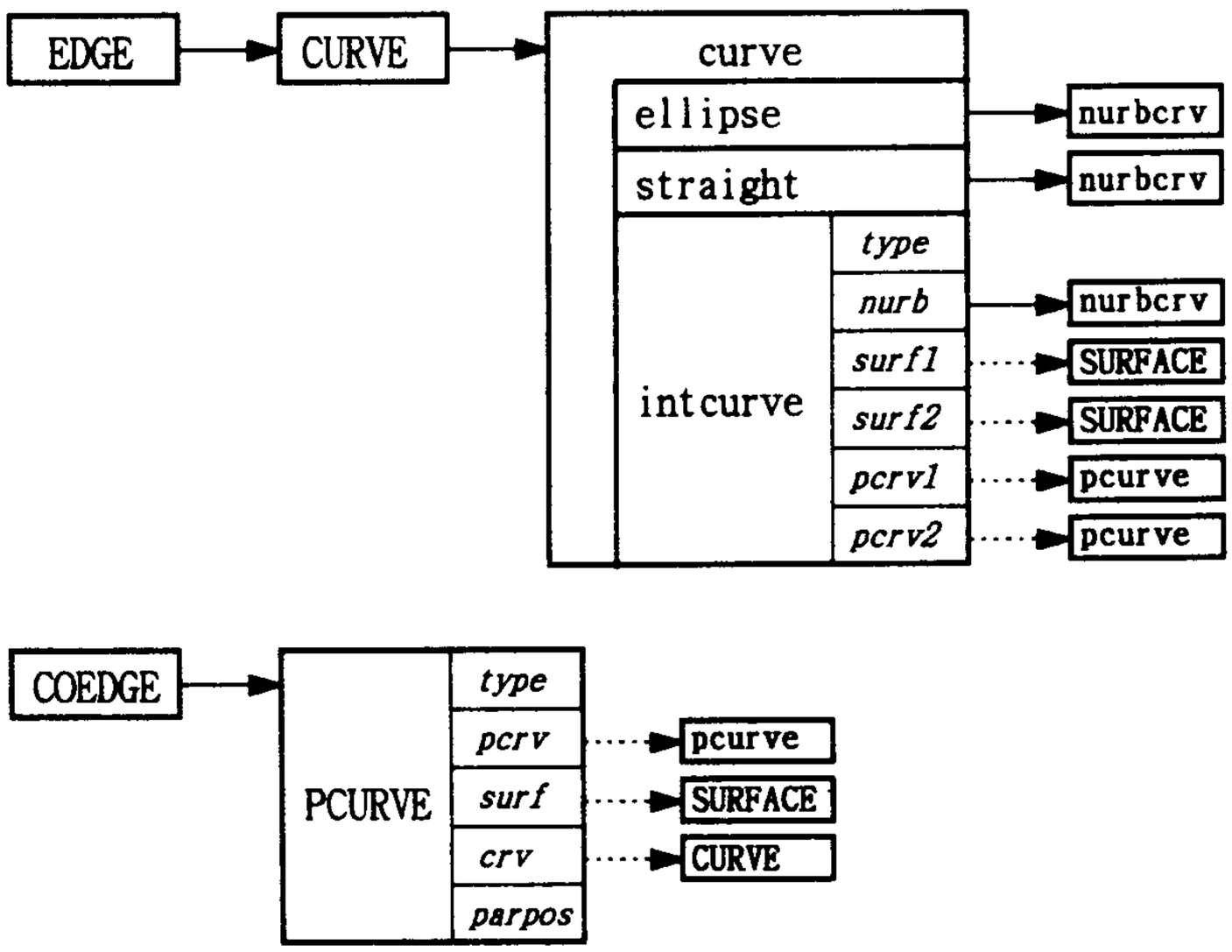


그림 4.20 Representation of curve entity

하나는 곡선을 정의하는 intcurve에 3차원 곡선과 도메인 곡선을 NURBS로 정의하는 pcurve를 함께 갖고 있도록 표현하는 것이다. intcurve에 있는 type은 '0', '1', '2'를 값으로 갖을 수 있으며, '0'은 pcurve를 갖고있지 않음 나타내고, '1'은 3차원 곡선을 곡면 위에 투영하여 생성한 곡선과 같이 pcurve가 하나만 존재하는 경우를 나타낸다. 또한, '2'는 곡면과 곡면의 교차(intersection)에 의하여 생성된 곡선처럼 두개의 pcurve가 정의되는 경우를 의미한다. 다른 하나는 pcurve와 해당 곡면을 PCURVE에서 직접적으로 포인팅하

게 표현하는 것이다. PCURVE의 type은 '0', '1', '-1', '2', '-2', '3'을 갖으며, '0'은 PCURVE가 직접적으로 pcurve와 pcurve가 정의되는 SURFACE를 가리키고 있음을 나타내고, '3'은 곡면상에 존재하는 점과 같이 곡선의 정의가 존재하지 않고, PCURVE가 곡면의 도메인 상에서 점의 위치를 나타내는 (u, v)를 갖고 있음을 의미한다. '1'과 '2'는 pcurve가 CURVE에 있음을 의하며, '1'은 intcurve의 pcrv1이, '2'는 pcrv2가 COEDGE의 형상 정보로 사용되는 pcurve임을 나타낸다. 그리고 '-1', '-2'는 pcurve가 정의된 방향과 반대로 사용됨을 나타낸다.

다. 곡면(surface)

본 데이터 구조에서 사용되는 모든 곡면은 NURBS로 정의되며 FACE의 형상 정보나 도메인 곡선을 정의하기 위하여 사용된다. 곡면은 그림 4.21과 같이 모양에 따라 평면을 나타내는 plane, 원통면이나 원추면을 나타내는 cone, 토러스 면을 나타내는 torus, 구면을 나타내는 sphere, 자유 곡면을 나타내는 spline등으로 구분하여 사용한다.

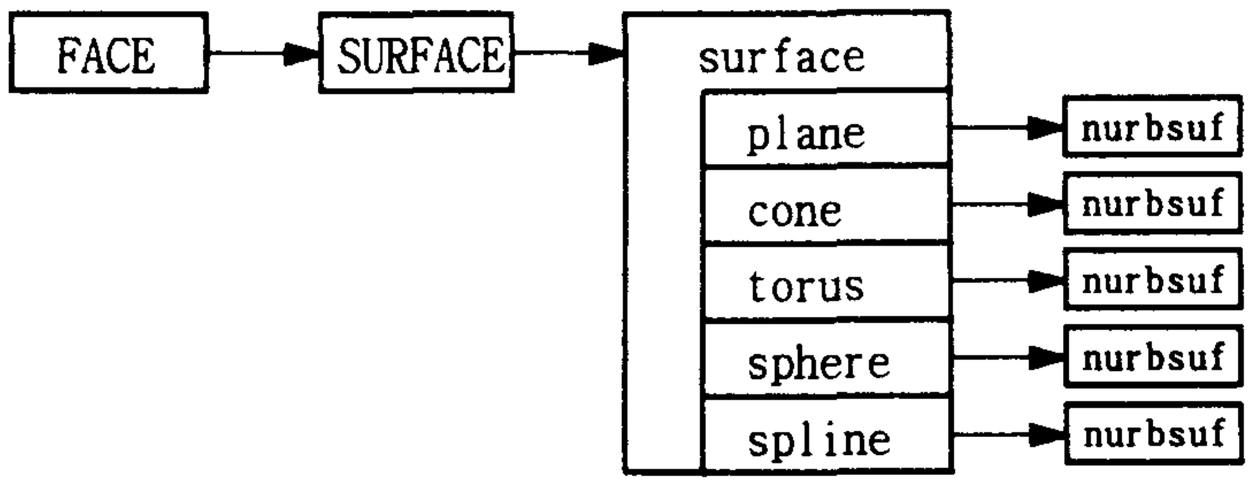


그림 4.21 Representation of surface entity

5. 데이터 구조의 구현

본 데이터 구조는 각각의 특성을 갖고 있는 위상 요소와 형상 요소들의 연결 관계와 소속 관계로 표현되어 있다. 그러므로 이러한 특성을 쉽게 표현할 수 있으며, 대표적인 객체 지향 프로그래밍 언어인 C++로 데이터 구조를 구현하였다. C++는 객체의 특성을 정의할 수 있는 데이터와 해당 객체를 갖고 수행할 수 있는 작업(operation) 등으로 객체를 정의하는 클래스(class)를 사용한다. 또한, 클래스는 상속성(inheritance)을 갖고 있으므로 객체간의 소속 관계를 쉽게 구현할 수 있으며, 데이터 구조의 구현에도 이러한 특성을 이용하였다. 즉, 클래스를 사용하여 각각의 위상 요소와 형상 요소들을 정의하였고, 곡선, 곡면 등의 정의와 같이 소속 관계를 표현하는데는 클래스의 상속성을 이용하였다. 그리고 요소간의 연결 관계는 포인터(pointer)를 이용하였다. 그림 4.22는 데이터 구조를 구현하는데 사용된 클래스들의 상속 관계를 표현한 그림으로 형상 요소의 클래스 정의는 APPENDIX B에, 위상 요소의 클래스 정의는 APPENDIX C에 있다. 그림에서 ENTITY는 모든 위상 요소의 기반 클래스(base class)이고, curve, surface는 각각 곡선, 곡면의 기반 클래스이다. 또한, nurbcrv, nurbsuf, pcurve는 곡선과 곡면을 NURBS로 정의하는 클래스로 기반 클래스로 정의하였다. ATTRIB는 위상 요소나 형상 요소의 속성(attribute)을 표현하는데 사용되는 클래스로 ENTITY의 파생 클래스(derived class)이다. 현재 구현된 속성 요소는 COLOR_ATTRIB만 있으며,형상 요소의 색깔을 설정하는데 사용된다. 속성 요소를 추가할 경우에는 ATTRIB의 파생 클래스로 정의하면 사용할 수 있다. TRANSFORM은 모델의 전역 변환 행렬(global transformation matrix)을 저장하는 클래스로 ENTITY의 파생 클래스이다. 데이터 구조에서 SHELL, FACE, EDGE, VERTEX등의 위상 요소는 상하 연결 관계를 표현할 때 타입이 다른 위상 요소들이 연결된다. 예를 들면, FACE는 경계를 표현하는 LOOP와

연결되는 경우도 있고, 와이어 곡선을 표현할 경우에는 EDGE와 연결된다. 이러한 문제는 클래스의 상속성을 이용하면 데이터를 추가하지 않아도 쉽게 해결할 수 있다. C++에서 파생된 클래스는 자동적으로 기반 클래스의 타입으로 변환되므로 이러한 연결 관계를 표현하는 데이터를 ENTITY 타입으로 정의하고, 파생 클래스에서 자신의 실제 타입을 갖고 있으면 된다.

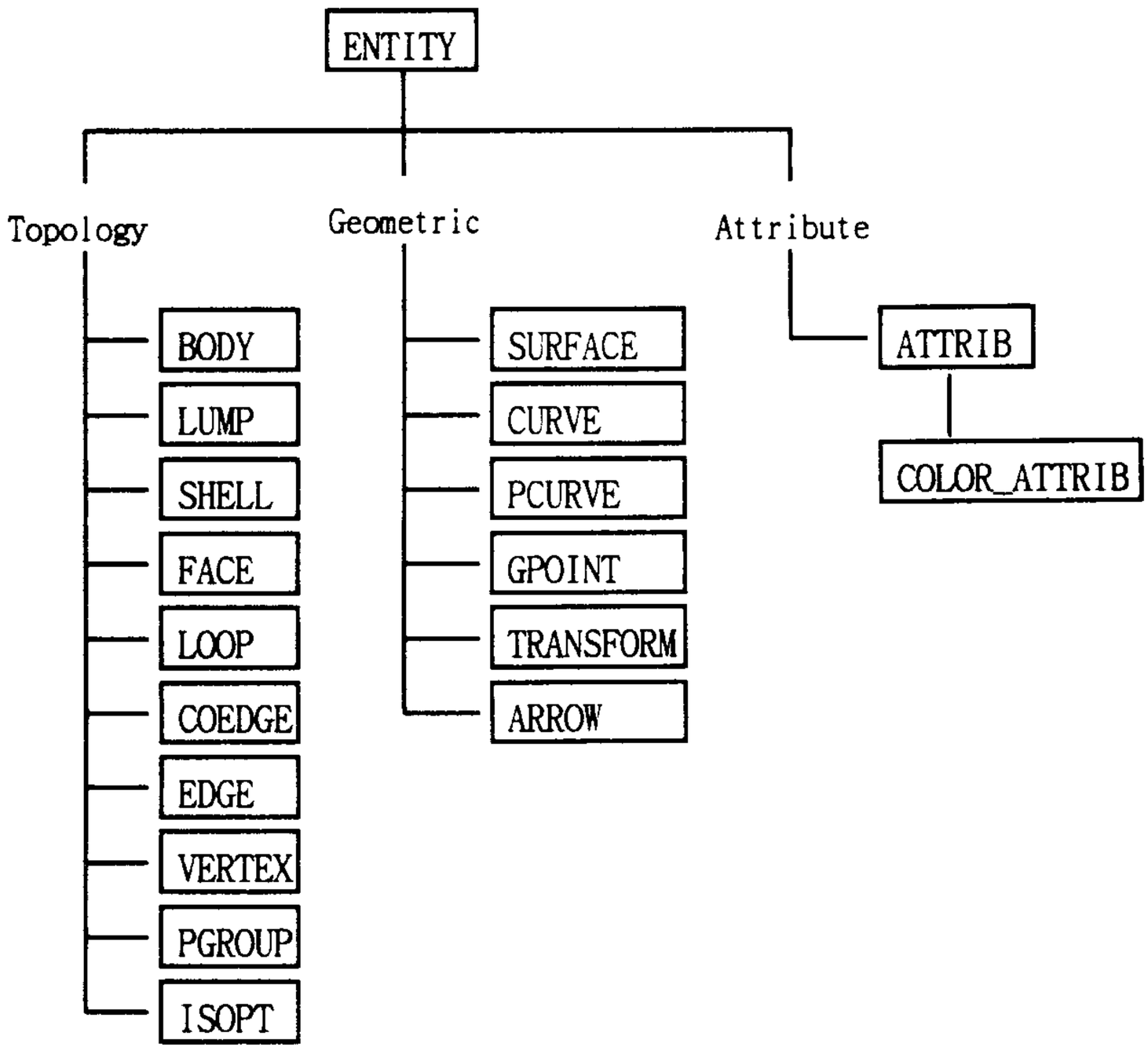


그림 4.22 Derivation of ENTITY class

제 4 절 데이터 구조의 관리

1. 개 요

데이터 구조의 관리란, 시스템 사용자의 모델링 작업에 의하여 생성되는 형상 요소를 데이터 구조에 맞게 저장하거나, 이미 데이터 구조에 저장되어 있는 데이터를 조회하여 응용 프로그램에서 활용할 수 있도록 하는 것이다. 본 연구에서도 이러한 기능을 수행하는 API들을 제공하여 데이터 구조를 파악하지 않고도 모델링을 수행하는 응용 프로그램을 쉽게 작성할 수 있도록 하였다. 데이터 구조를 관리하는 API들은 크게 두 가지로 분류할 수 있다. 하나는 데이터 구조를 조회하는 API로, 임의의 형상 요소나 위상 요소와 관련된 정보를 데이터 구조로부터 찾아내는 기능을 수행하는 함수이다. 다른 하나는 데이터 구조에 저장되어 있는 데이터를 변경하는 API로, 형상 요소의 추가, 삭제, 수정 기능을 수행한다. 데이터 구조를 조회하는 API는 쉽게 구현할 수 있으나, 데이터를 변경하는 API는 그렇지 않다. 본 데이터 구조가 형상 정보에 따른 위상 정보를 동시에 저장하고 있으므로, 형상 정보의 추가, 삭제, 수정에 따라 합당한 위상 정보를 유지해야 하기 때문이다. 그러므로 데이터를 변경하는 API는 모델의 위상 정보를 합당하게 유지하면서 데이터를 추가, 삭제, 수정하는 오일러 작업자(Euler operator)를 사용하여 구현하였다. 오일러 작업자는 데이터 구조에 데이터를 추가, 삭제, 수정하는 기능을 직접적으로 수행하므로 데이터 구조와 응용 프로그램을 연결 해주는 중간 역할을 담당한다. 또한, 오일러 작업자의 사용은 데이터 구조를 관리하는 API를 손쉽게 구현하거나 수정할 수 있게 해준다. 일반적으로 모델링 시스템은 사용자의 명령에 가능한 빠르게 결과를 보여줘야 한다. 특히, 형상 요소의 선택이나 그래픽 디스플레이는 빠르게 수행되어야 한다. 본 연구에서도 이러한 작업을 신속하게 수행할 수 있도록

display-list를 사용 하였다. Display-list는 객체들을 리스트로 관리하는 PTRLIST 클래스 타입으로 객체를 생성하고, 데이터 구조에서 GPOINT, CURVE, SURFACE를 형상 정보로 사용하는 VERTEX, ISPOT, PGROUP, EDGE, FACE등의 포인터를 저장하여 데이터 구조를 조회하지 않고 형상 요소의 선택이나 디스플레이를 빠르게 수행할 수 있도록 하였다.

2. 오일러 작업자(Euler operator)

오일러 작업자는 형상 모델링 시스템에서 생성되는 모델이 수학적으로 항상 합당한 위상을 갖도록 하는 중요한 도구이다. 또한, 데이터 구조의 특성과 복잡성으로부터 상위 레벨의 작업들을 분리시키는 효과가 있으며, 역작업이 존재하기 때문에 앞서 실행한 명령을 취소시키는 것이 용이하다. 그러므로 많은 형상 모델링 시스템에서 데이터 구조를 관리하는 기능으로 오일러 작업자를 사용하고 있다. 오일러 작업자는 오일러-포앙카레(Euler-Poincare) 식[4-6]을 근간으로 형상 모델의 경계 표현법에서 사용되는 위상 요소들의 특성과 종류를 적용하여 식을 수정하고, 수정된 식을 만족하면서 데이터 구조의 위상 요소를 추가, 수정, 삭제하도록 구현한 함수이다. 오일러 작업자는 다양체 솔리드 모델링에서 사용하기 시작하여 최근에는 비다양체 모델의 위상 정보를 관리하는 도구로 확장하여 사용하고 있다.

가. 비다양체 모델의 오일러-포앙카레 식

오일러 작업자의 근간이 되는 오일러-포앙카레 식은 다음과 같다.

$$\sum_{i=0}^{n}(-1)^i a_i = \sum_{i=0}^{n}(-1)^i p_i \qquad (4.1)$$

식(4.1)에서 n은 차원 수, a_i는 i 차원 단체들의 갯수, p_i는 i 차원의 베티

수(Betti number)이다. 본 데이터 구조에서는 점, 곡선, 곡면과 곡면들의 조합으로 이루어진 솔리드까지 표현할 수 있으므로 식(4.1)에서 n은 2이다. 그러므로 a_0는 0차원 단체의 갯수이므로 점의 갯수를 의미하며, 본 데이터 구조에서 사용하는 위상 요소로는 VERTEX의 갯수(V)라고 할 수 있다. a_1은 1차원 단체의 갯수이므로 EDGE의 갯수(E)이다. 또한, a_2는 곡면의 갯수, 즉 FACE의 갯수(F)라고 할 수 있다. 이들을 사용하여 식(4.1)을 정리하면 다음과 같다.

$$V - E + F = P_0 - P_1 + P_2 \qquad (4.2)$$

식(4.2)에서 우변의 변수들은 0, 1, 2차원 베티 수들이다. 이들을 형상 모델에 적용하면, P_0는 연결된 도형들의 갯수를 나타낸다. P_1은 그림 4.23과 같이 곡면상에서 한 점으로 수축시킬 수 없는 원의 갯수와 곡선으로 이루어진 회로(cycle)의 갯수를 나타낸다.

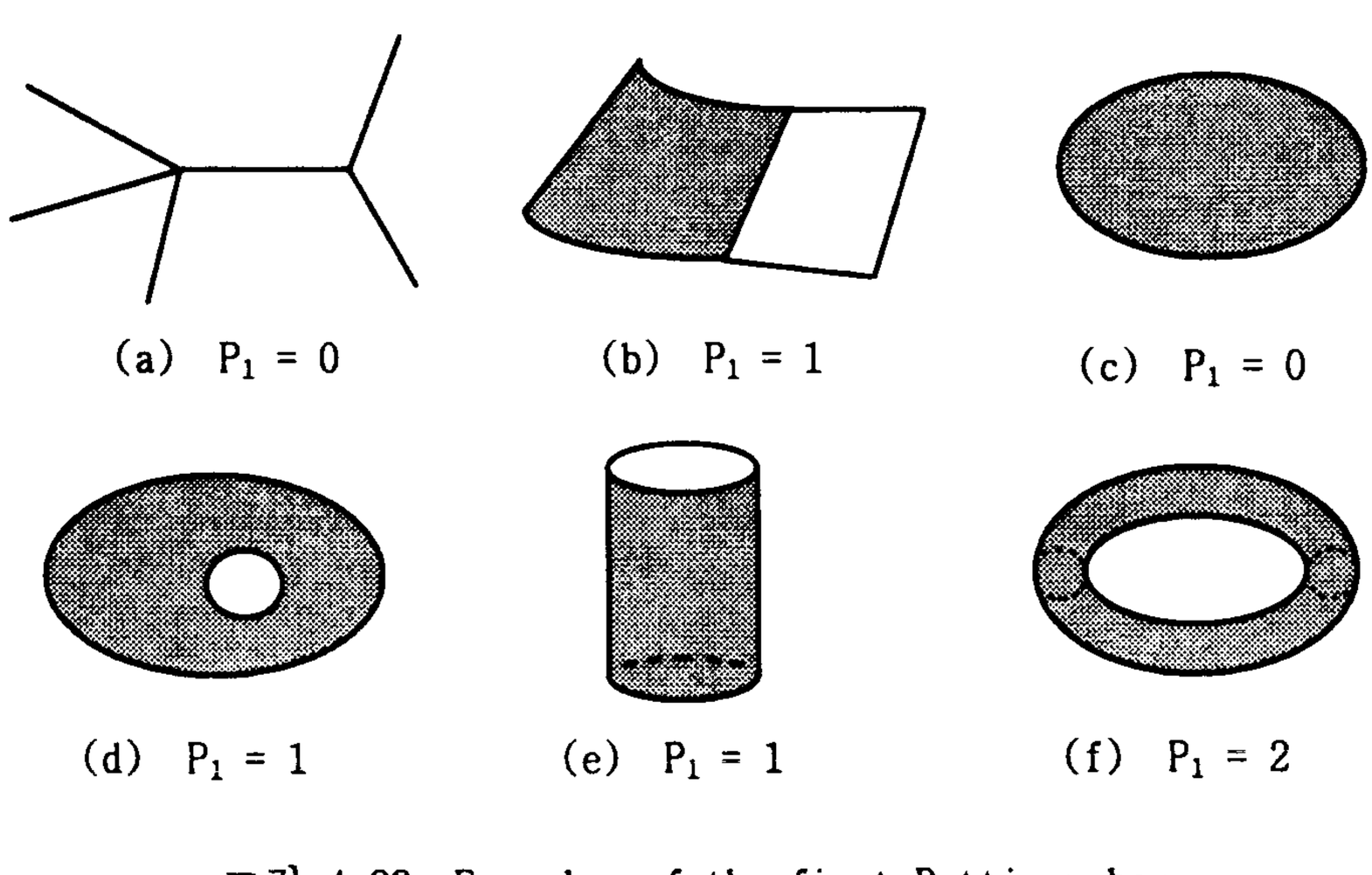


그림 4.23 Examples of the first Betti number

P_2는 연결된 면들에 의하여 만들어지는 영역의 갯수이다. 이 들을 본 데이터 구조에서 사용하는 요소로 표현하면 P_0는 SHELL의 갯수(S)이며, P_2는 무한대

영역 나타내는 LUMP를 제외한 LUMP의 갯수(P)이다. P_1은 위상 요소로 표현되지 않으며 임의의 갯수(C)로 나타내면 식(4.2)는 다음과 같이 된다.

$$V - E + F = S - C + P \tag{4.3}$$

식(4.3)은 임의의 면이 하나의 루프로 구성되는 경우에만 적용할 수 있다. 그러나 대부분의 모델은 안쪽 루프(inner loop)를 갖는 면으로 구성되므로 이를 고려해야 한다. 안쪽 루프를 갖는 임의의 면에서 안쪽 루프를 제거하기 위해서는 안쪽 루프와 바깥쪽 루프를 연결하는 모서리를 추가하면 안쪽 루프를 제거할 수 있다. 그러므로 모델에서 안쪽 루프를 모두 제거하기 위해서는 안쪽 루프의 갯수(L)만큼 모서리를 추가하면 되며, 식(4.3)에서 모서리의 갯수 E는 E+L로 치환된다. 이를 식(4.3)에 대입하여 정리하면 다음과 같은 식을 얻을 수 있다.

$$V - E + F - L = S - C + P \tag{4.4}$$

식(4.4)가 본 연구에서 사용하고 있는 위상 작업자의 바탕이 되는 오일러-포앙카레 식(Euler-Poincare formula)이다. 예를 들어 다양체 솔리드 모델만을 표현한다면, 영역의 수와 연결된 곡면의 수가 같으므로 2차원 베티 수는 항상 0 차원 베티 수와 같다. 또한 1차원 베티 수는 모델에 있는 관통 구멍(through hole)의 갯수(H)의 두 배만큼 존재한다. 따라서 식(4.4)의 우변은 2S + 2H로 치환할 수 있으며, 다양체 솔리드 모델의 위상을 관리하는데 근간이 되었던 식을 얻을 수 있다.

$$V - E + F - L = 2(S - H) \tag{4.5}$$

나. 오일러 작업자(Euler operator)

비다양체 모델에 대한 오일러 작업자란 오일러-포앙카레 식인 식(4.4)를 만족하도록 위상 요소를 추가, 삭제, 수정하는 함수이다. 식(4.4)는 7개의 변수로 구성되어 있으므로, 이 변수들을 축으로 하는 공간을 가상하고, 임의의 위치를 지정한다면 독립적으로 사용될 수 있는 변수는 6개가 된다. 따라서 비다양체 모델의 위상을 관리하기 위해 이론적으로 필요한 최소한의 오일러 작업자는 역작업을 포함해서 12개가 된다. 그러나 이것들 만으로는 모델링을 효율적으로 구현할 수 없으므로 이에 덧붙여 8개의 오일러 작업자를 추가하였고, 모델링 작업의 시작 및 종료 시에 BODY와 무한대 영역을 나타내는 LUMP를 생성시키고 삭제하는 MBP(Make Body and lumP)와 KBP(Kill Body and lumP)를 사용하였다. 표 4-1이 이들을 정리한 것이며, APPENDIX C에 함수의 프로토타입이 있다.

표 4-1 Topological operators

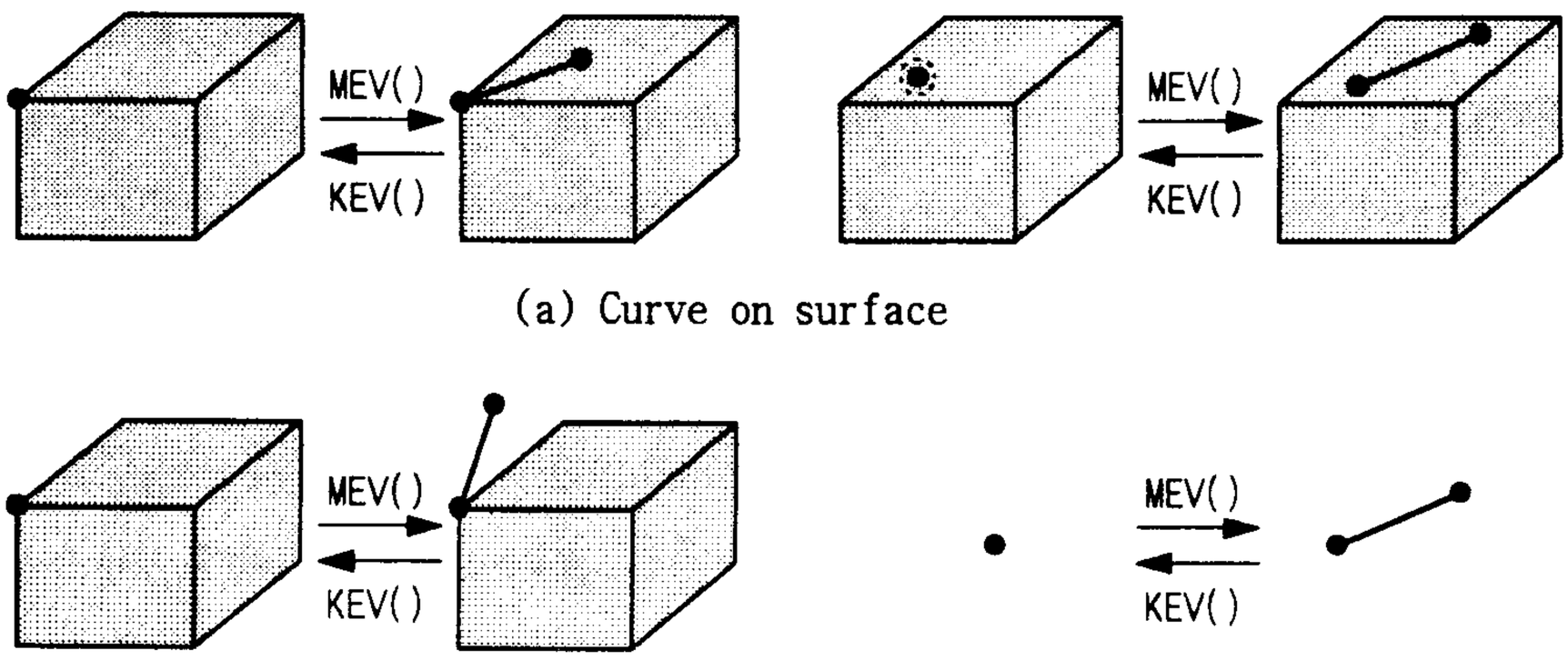
(P: lump, J: join)

Class.	Name	Description	V	E	F	L	S	C	P
Basic Euler operators	MEV (KEV)	Make edge and vertex	1	1	0	0	0	0	0
	MEC (KEC)	Make edge and cycle	0	1	0	0	0	1	0
	MFKC (KFMC)	Make face, kill cycle	0	0	1	0	0	-1	0
	MFP (KFP)	Make face and lump	0	0	1	0	0	0	1
	MVS (KVS)	Make vertex and shell	1	0	0	0	1	0	0
	MVL (KVL)	Make vertex and loop	1	0	0	1	0	0	0
Additional Euler operators	SEMV (JEKV)	Split edge, make vertex	1	1	0	0	0	0	0
	MEF (KEF)	Make edge and face	0	1	0	0	1	0	0
	KEML (MEKL)	Kill edge, make loop	0	-1	0	1	0	0	0
	KEMS (MEKS)	Kill edge, make shell	0	-1	0	0	1	0	0
Additional topological operators	MBP (KBP)	Make body and lump							

(1) MEV(), KEV()

MEV() 함수는 곡선과 곡선의 한 끝점을 형상 정보로 갖는 VERTEX를

입력 받아 새로운 EDGE와 VERTEX를 생성하여 데이터 구조에 곡선을 저장하는 기능을 수행한다. 새로 생성되는 VERTEX는 이미 VERTEX로 사용되지 않는 곡선의 끝점을 형상 정보로 사용하며, 곡선의 형상이 폐곡되어 양 끝점이 같거나 스스로 교점이 발생하는 곡선은 입력으로 사용할 수 없다. 생성되는 EDGE이 방향은 입력된 VERTEX에서 새로 생성되는 VERTEX로 향하도록 지정된다. MEV()함수는 그림 4.24 (a)와 같이 곡선이 곡면 위에 존재할 때에는 EDGE의 부모(parent)로 COEDGE를 생성하여 VERTEX가 속해 있는 LOOP에 추가된다. (b)와 같이 곡선이 공간상에 존재하는 경우에는 와이어 모서리로 추가된다. KEV()함수는 MEV()의 역작업을 수행하는 함수로 입력된 EDGE와 VERTEX를 데이터 구조에서 삭제한다.



(a) Curve on surface

(b) Wire curve

그림 4.24 MEV(), KEV() operators

(2) MEC(), KEC()

MEC()함수는 그림 4.25와 같이 곡선과 곡선의 양 끝점을 형상 정보로 사용하는 두 VERTEX를 입력받아 새로운 EDGE를 추가한다. 추가된 EDGE에 의하여 모델상에 회로(cycle)가 형성되어 식(4.4)가 만족된다. 두 VERTEX는 입력된 SHELL을 구성하는 FACE나 와이어 EDGE의 경계가 되는 VERTEX이어야 한다.

KEC()은 MEC()의 역작업자로 EDGE를 입력받아 이를 삭제한다. 삭제될 EDGE는 회로를 구성하는 EDGE이어야 한다.

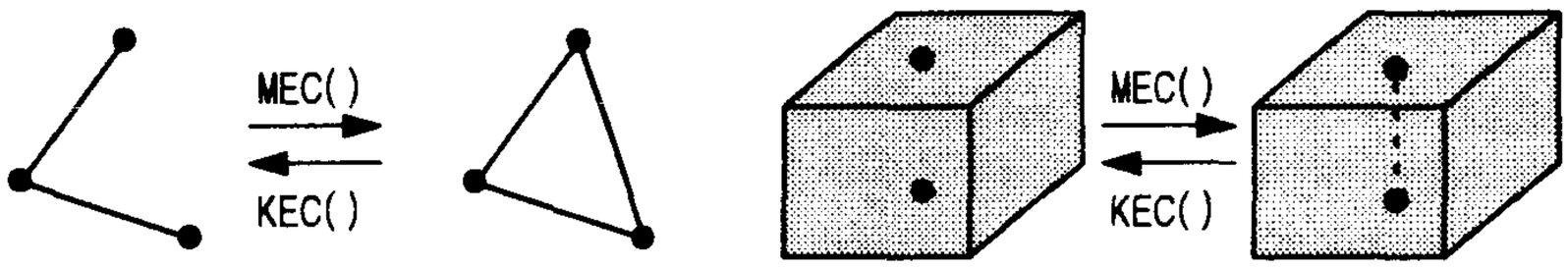


그림 4.25 MEC(), KEC() operators

(3) MFKC(), KFMC()

MFKC()은 그림 4.26과 같이 회로를 구성하는 모든 EDGE, 이들이 속한 SHELL, FACE의 형상 정보로 사용할 곡면 등을 입력받아 입력된 EDGE들을 사용하여 LOOP를 만들고, 이 LOOP를 경계로 하는 FACE를 데이터 구조에 추가한다. 입력된 EDGE들의 형상 정보로 사용된 곡선들은 입력된 곡면 상에 존재해야 한다. LOOP의 방향은 곡면의 법선 방향에 대하여 반시계 방향으로 설정된다. 새 FACE의 경계에서의 방사 순환은 이 함수가 종료된 후에 FACE의 경계 EDGE들을 순서대로 찾아 방사 순환을 만드는 함수를 호출하여 표현한다. KFMC()은 MFKC()의 역작업자로 입력된 FACE를 삭제하고 FACE의 경계 EDGE들로 회로를 구성한다. 방사 순환의 조정은 이 작업자 내에서 수행한다.

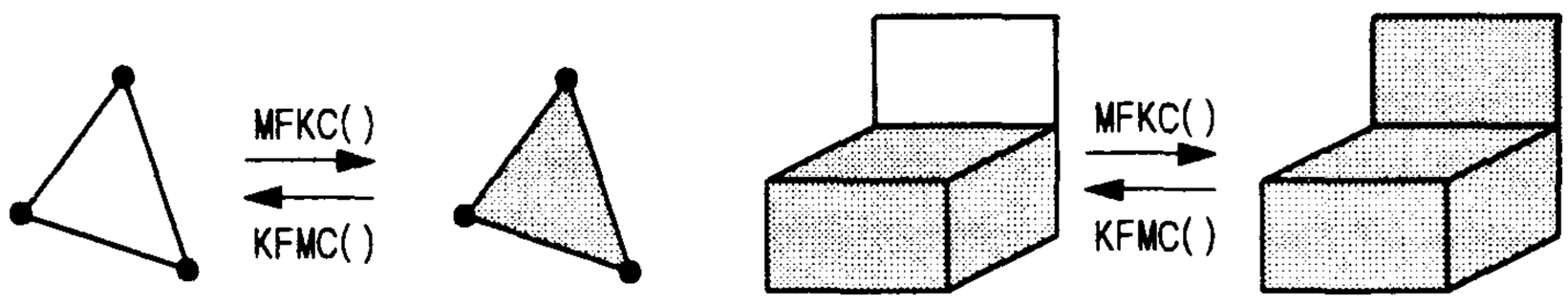


그림 4.26 MFKC(), KFMC() operators

(4) MFP(), KFP()

MFP()는 그림 4.27과 같이 회로를 구성하는 모든 EDGE, 이들이 속한 SHELL, 생성될 FACE의 형상 정보로 사용될 곡면을 입력받아 FACE를 생성하여

새로운 LUMP를 데이터 구조에 추가한다. 그러므로 FACE가 생성되어 밀폐된 영역이 발생되는 회로의 EDGE들이 입력되어야 한다. KFP()는 MFP()의 역작업자로 삭제될 FACE와 해당 LUMP를 입력받아 이를 삭제하고 회로를 구성한다.

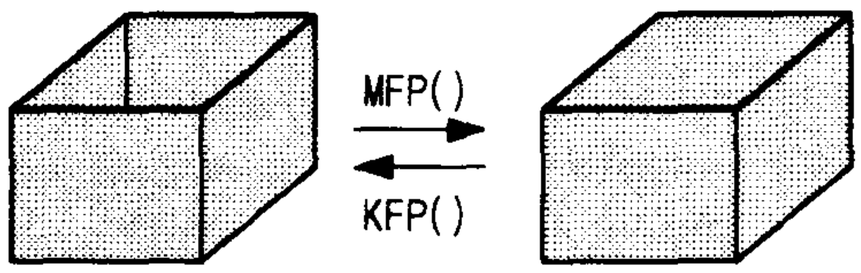


그림 4.27 MFP(), KFP() operators

(5) MVS(), KVS()

MVS() 작업자는 그림 4.28과 같이 LUMP와 점 데이터를 입력받아 입력된 점을 형상 정보로 하는 VERTEX를 생성하고, 주어진 LUMP내에 생성된 VERTEX를 외 꼭지점 쉘로 데이터 구조에 추가한다. KVS()은 MVS의 역작업자로 입력된 외 꼭지점 쉘을 삭제한다.

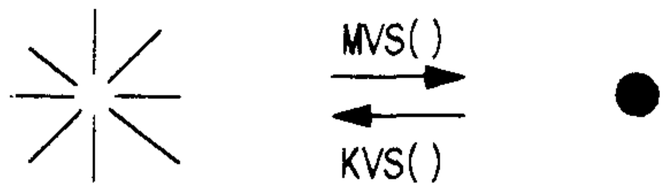


그림 4.28 MVS(), KVS() operators

(6) MVL(), KVL()

MVL()는 그림 4.29와 같이 곡면상에 존재하는 점 데이터와 해당 곡면을 형상 정보로 사용하는 FACE를 입력 받아 FACE내에 외 꼭지점 루프를 생성하고, KVL()는 이러한 루프를 삭제하는 작업자이다.

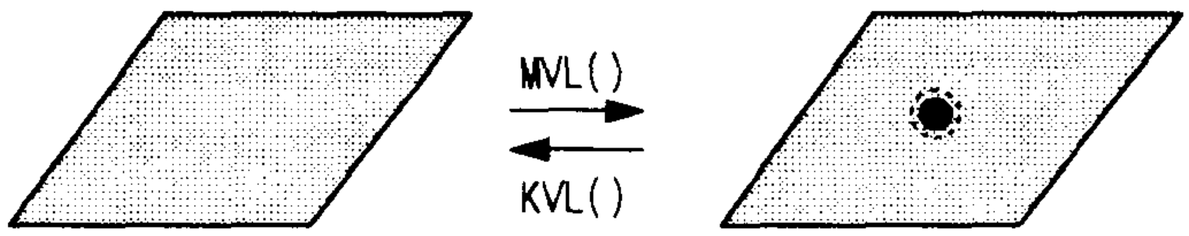


그림 4. 29 MVL(), KVL() operators

(7) SEMV(), JEKV()

SEMV()는 그림 4.30과 같이 입력된 EDGE를 두개로 분할하고, 두 EDGE를 연결하는 VERTEX를 생성한다. 두 EDGE의 형상 정보는 분할되기 전에 EDGE의 형상 정보로 사용되는 곡선을 분할하여 입력하여야 한다. 분할된 EDGE가 곡면의 경계로 사용되는 경우에는 COEDGE도 분할하여 LOOP를 조정한다. JEMV()는 SEMV()의 역작업자로 타 EDGE와 합쳐질 EDGE와 연결될 EDGE와 공유되는 VERTEX를 입력 받아, 해당 VERTEX를 삭제하고 두 EDGE를 하나의 EDGE로 만든다.

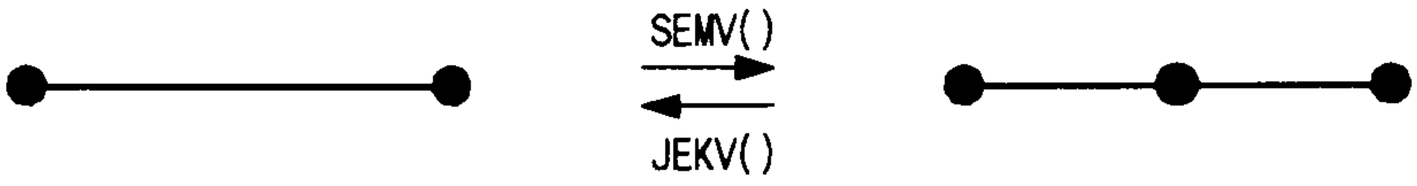


그림 4.30 SEMV(), JEMV() operators

(8) MEF(), KEF()

MEF()는 그림 4.31과 같이 FACE의 바깥 경계로 사용되는 LOOP와 이 LOOP에 속해 있는 두 VERTEX를 입력받고, 이들을 잇는 EDGE를 생성하여 FACE를 둘로 분할하는 작업이다. 이때 원래 FACE에 안쪽 LOOP가 존재하는 경우에는 포함 관계에 따라 옮겨준다. 그러므로 생성된 EDGE의 형상 정보로 사용되는 곡선은 FACE상의 곡선과 입력된 VERTEX이외에 타 교점이 발생되지 않아야 한다. KEF()는 MEF()의 역작업자로 입력된 EDGE를 삭제하고 EDGE를 공유하는 두 FACE를 합치는 기능을 수행한다. 그러므로 입력된 EDGE는 서로 다른 두 FACE와 인접해 있는 EDGE이어야 한다.

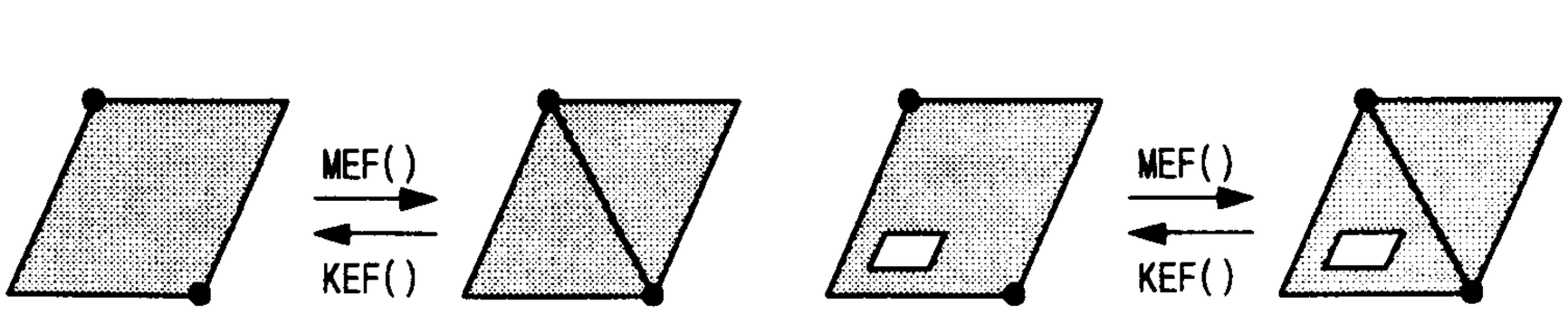


그림 4.31 MEF(), KEF() operators

(9) MEKL(), KEML()

MEKL()는 동일한 FACE상에서 서로 다른 LOOP상에 있는 두 VERTEX를 연결하는 EDGE를 생성하여 두 LOOP를 하나로 합치는 기능을 수행한다. KEML()는 MEKL()의 역작업자로 그림 4.32의 왼쪽과 같은 돌출 모서리(strut edge)나 오른쪽과 같은 해협 모서리(isthmus edge)를 삭제하여 하나의 LOOP를 둘로 분할한다. 특히, 돌출 모서리가 삭제되는 경우에는 외 꼭지점 루프가 생성될 수도 있다.

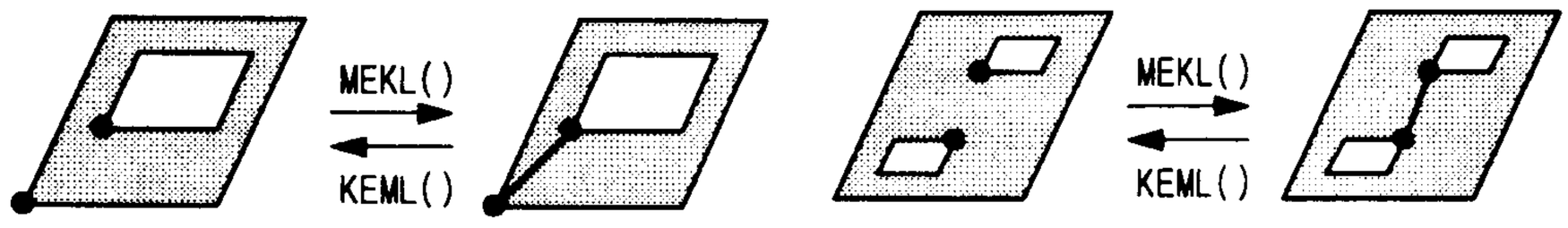


그림 4.32 MEKL(), KEML() operators

(10) MEKS(), KEMS()

MEKS()은 그림 4.33과 같이 서로 다른 SHELL에 속한 두 VERTEX를 연결하는 와이어 EDGE를 생성하여 두개의 SHELL을 하나로 합치는 기능을 수행한다. KEMS()은 MEKS()의 역작업자로 와이어 EDGE를 삭제하여 하나의 SHELL을 둘로 분할한다. 그러므로 삭제될 EDGE는 자신의 두 꼭지점 사이에 있는 유일한 연결 경로이어야 한다. 그림 4.33의 오른쪽과 같이 삭제될 모서리가 돌출 와이어 모서리인 경우에는 외 꼭지점 SHELL이 생성된다.

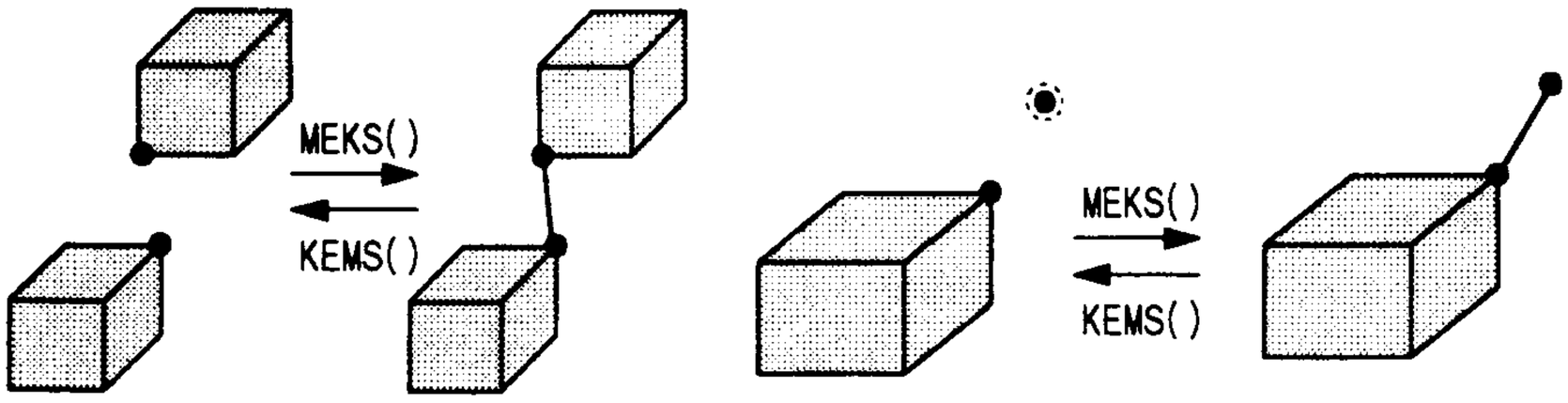


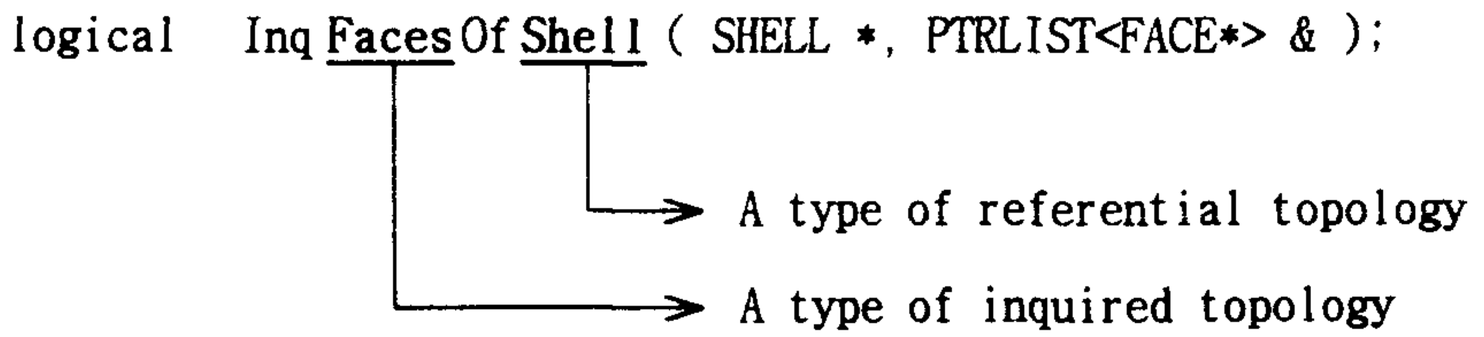
그림 4.33 MEKS(), KEMS() operators

(11) MBP(), KBP()

MBP()는 모델링 초기에 데이터를 저장할 위상 모델링 공간을 만드는 기능을 수해하는 함수로 무한대의 영역을 갖는 가상의 LUMP로 구성된 BODY를 생성한다. KBP()는 MBP()의 역작업자로 새로운 모델을 생성하기 위하여 기존의 모델을 데이터 구조에서 삭제하는 기능을 수행한다.

3. 데이터 조회

데이터 구조로부터 필요한 정보를 효율적이고, 빠르게 조회하는 기능은 데이터 구조를 관리하는데 필수적인 기능이며, 데이터를 추가, 수정, 삭제하는 작업에서도 이용된다. 또한, 응용 프로그램에서도 직접적으로 호출하는 기능이다. 그러므로 본 연구에서는 데이터 구조로부터 데이터를 찾아내는 API를 구현하여 모델을 생성하는 응용 프로그램에서 편리하게 사용할 수 있도록 하였다. 데이터를 조회하는 API는 크게 3 종류로 구분할 수 있다. 하나는 위상 정보를 조회하는 API로 각각의 위상 요소를 입력하여 관련된 위상 정보를 조회하는 기능을 수행한다. 다음에 있는 API는 임의의 SHELL에 포함되는 모든 FACE를 조회하는 함수로, 위상 정보를 조회하는 대부분의 API가 이와 같은 형식을 갖으며 APPENDIX B에 구현된 함수들이 나열되어 있다.



위 형식에서와 같이 조회되는 위상 요소와 조회의 기준이 되는 위상 요소의 타입을 이용하여 함수의 이름을 결정하였다. 이러한 API는 기준 위상 요소의 포인터를 입력받아 객체를 리스트로 관리하는 PTRLIST 클래스를 사용하여 조회된

위상 요소를 넘겨 준다. 함수 실행에 오류가 있는지, 없는지는 logical 타입으로 넘어오는 리턴 값으로 판단할 수 있다. 특히, 데이터 구조상에 특이하게 저장되는 외꼭지점 쉘이, 와이어 모서리, 외꼭지점 루프 등을 조회하는 함수도 따로 구현되어 있다. 위상 요소를 조회하는 API는 입력된 위상 요소를 기준으로 다음 단계의 위상 요소를 조회하고, 각각의 조회된 요소를 기준으로 다음 단계의 위상 요소를 조회하는 API를 호출하는 과정을 통하여 필요한 정보를 찾도록 구현되어 있다. 둘째는 형상 정보를 조회하는 API로 위상 요소인 FACE, COEDGE, EDGE, VERTEX, ISOPT 등을 입력하여 각각의 형상 정보, 즉 점 데이터나 NURBS로 정의된 곡선, 곡면 등을 얻는 기능을 수행한다. 물론, 이러한 위상 요소를 알고 있으면 멤버 함수를 이용하여 GPOINT, CURVE, COEDGE, SURFACE 등을 쉽게 얻을 수 있다. 그러나 형상 데이터를 갖는 position, nurbcrv, pcurve, nurbsuf 등을 얻기 위해서는 데이터 구조상에서 여러 단계를 거쳐야 한다. 특히, COEDGE의 형상 정보인 pcurve를 얻기 위해서는 아래와 같이 PCURVE와 intcurve의 타입에 따라 많은 과정을 수행해야 한다. 그러므로 이러한 기능들을 하나의 함수로 구현하면 응용 프로그램이 간단해지고, 필요한 정보를 편리하게 얻을 수 있다.

```
/*********************************************************************************/
logical GetPcurveDataOfCoedge(COEDGE *ce, pcurve *&gp, par_pos &sp, par_pos &ep,
        SURFACE *&surf)
/*********************************************************************************/
// Function:
//     Get the related data with coedge - a pointers of parameter curve
//     and a supporting surface, start and end parameter of pcurve on the domain
//     of supporting surface
//
// Input:
//     ce:     A pointer to coedge
//
// Output:
//     gp:     A pointer reference to the pcurve of coedge
//     sp:     A par_pos reference to the start position of parameter curve
//     ep:     A par_pos reference to the end position of parameter curve
```

```
//     surf:   A pointer reference to the supporting surface of pcurve
/*************************************************************************/
{
      // Check input coedge ═══════════════════════════════════════════
      PCURVE *pcrv = ce->geometry();
      .....
      // Get pcurve data of coedge ═══════════════════════════════════
      switch( pcrv->index() )
      {
         case 0:
         gp = pcrv->pcurve_def();
         ......
         break;

         case  1:
         gp = ((intcurve *)(pcrv->ref_curve()->geo_curve()))->pcrv1();
         ......
         break;

         case -1:
         gp = ((intcurve *)(pcrv->ref_curve()->geo_curve()))->pcrv1();
         ......
         break;

         ......

         default:
         return FALSE;
      }

      return TRUE;
}
```

세째는 오일러 작업자를 위하여 위상 요소간의 관계나 임의의 점 데이터, 곡선, 곡면 등이 이미 데이터 구조상에 위상 요소나 형상 요소로 사용되고 있는지를 조회하는 함수들이 구현되어 있으며, 예를 들면 다음과 같은 함수들이다.

```
logical DoesEdgeMakeCycle(EDGE *);
logical DoesEdgeLinkShell(EDGE *);
logical InqDirOfLoop(LOOP *, LOOPDIR &);
logical GetRelationBetween2Loops(LOOP *, LOOP *, SURFACE *, LOOPREL &);
logical IsPositionVertexOfBody(position *, VERTEX *&);
logical IsCurveEdgeOfBody(nurbcrv *, EDGE *, EDGE *&);
```

4. 데이터 추가/삭제

형상 모델링 시스템에 사용되는 대부분의 응용 프로그램들은 최종 결과를 데이터 구조에 기록하게 되므로 데이터 구조를 관리하는 기능에는 데이터를 조회하는 기능은 물론, 데이터를 추가, 수정, 삭제하는 기능이 있어야 한다. 데이터를 추가하는 기능은 응용 프로그램에서 생성된 형상 요소를 데이터 구조에 맞게 저장하는 기능이고, 수정하는 기능은 이미 저장된 형상 요소의 데이터를 수정하는 기능이다. 또한, 데이터를 삭제하는 기능은 필요 없는 데이터를 데이터 구조에서 제거하는 기능이다. 형상 모델링 시스템의 응용 프로그램에서 이러한 기능들은 서로 연관되어 사용된다. 특히, 본 데이터 구조와 같이 형상 정보와 위상 정보를 동시에 저장하는 구조에서는 한 형상 요소의 변화에 의하여 데이터의 조회, 추가, 수정, 삭제 작업이 동시에 수반되는 경우가 대부분이므로 응용 프로그램 내에서 이러한 복잡한 작업들을 하나하나 수행하는 것은 불가능하다. 그러므로 본 연구에서는 그림 4.34와 같이 데이터를 추가, 수정, 삭제 등을 수행하는 API를 구현하여 응용 프로그램 작성에 편리하게 사용할 수 있도록 하였다.

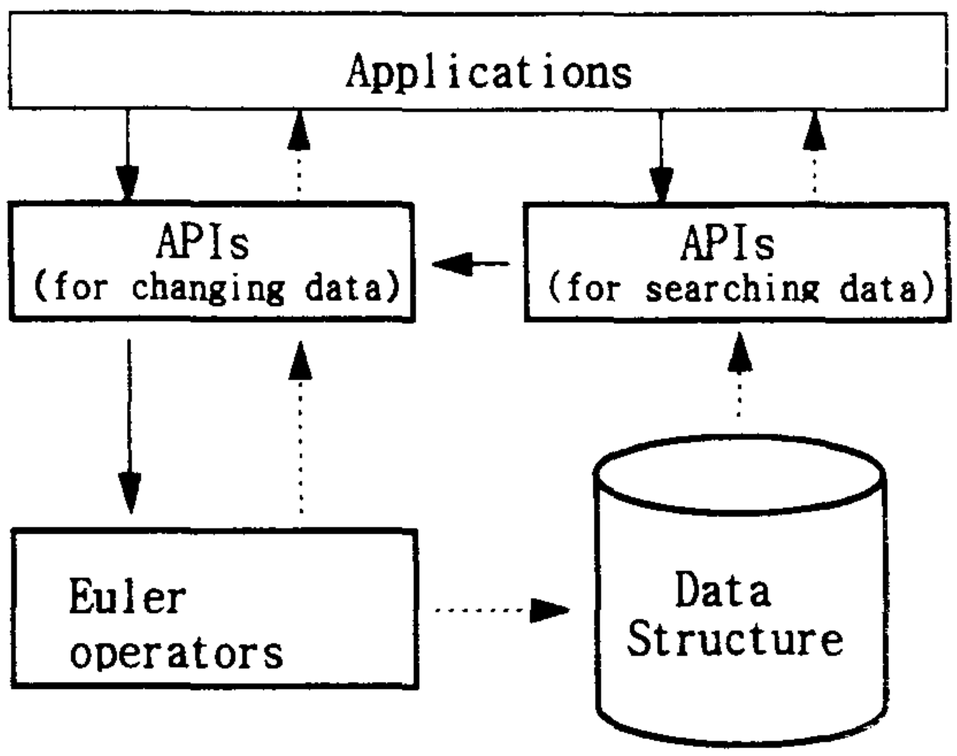


그림 4.34 Interface between application and data structure

그림 4.34와 같이 데이터를 추가, 수정, 삭제하는 API는 최종적으로 오일러 작업자를 사용하여 데이터 구조 내의 데이터를 변화시키도록 되어있다. 오일러 작업자는 모델의 위상을 수학적으로 합당하게 유지하면서 데이터 구조를 관리할 수 있는 편리한 도구 이나, 추가 또는 삭제될 형상 요소와 저장된 데이터의 관계에 따라 적합한 작업자를 선택하여 사용해야만 한다. 데이터 구조와의 인터페이스를 위하여 구현된 주요 API는 APPENDIX C에 있으며, 이들은 형상 요소별로 데이터를 추가하거나 삭제할 수 있도록 되어있다.

가. 곡면을 추가하는 API

곡면을 추가하는 API는 surface 클래스로 표현된 곡면을 입력받아 오일러 작업자 MFKC()을 사용하여 데이터 구조에 FACE를 생성하는 기능을 수행하는 함수로 개략적인 알고리즘은 다음과 같다.

logical SApi_InsFaceToDB(surface *gsurf, FACE *&nface)

{

① 입력 데이터를 검사함

② 곡면 gsurf의 경계 곡선들을 공간상의 곡선과 도메인 곡선으로 생성함 (공간상의 곡선은 nurbcrv, 도메인 곡선은 pcurve로 표현함)

③ 입력된 곡면 gsurf를 멤버로 하는 연결 요소 SURFACE 클래스를 생성함

④ 각각의 경계 곡선에 대하여 ②에서 생성한 nurbcrv, pcurve와 SURFACE를 사용하여 intcurve 클래스를 생성함

⑤ ④에서 생성한 intcurve를 형상 정보로 하는 EDGE를 데이터 구조에 추가하고 리턴되는 EDGE의 포인터를 PTRLIST 타입의 elist에 저장함 (EDGE의 추가는 SApi_InsWireEdgeToDB(intcurve *, EDGE *&) 이용)

⑥ ⑤에서 생성한 EDGE의 SHELL과 elist를 입력으로 오일러 작업자 MFKC(SHELL *, PTRLIST &, SURFACE *, FACE *&)를 호출하여 데이터

구조에 FACE를 추가함

⑦ 추가된 FACE의 모든 EDGE에 대하여 SetRadialCycle()함수를 사용하여 방사 순환을 조정함

⑧ current color로 COLOR_ATTRIB 클래스를 생성하고, 추가된 곡면에 속성 요소로 이를 부여함

⑨ 형상 요소의 display와 picking을 신속하게 수행하기 위하여 관리되는 display-list WeListOB, EdListOB, FaListOB에 내부적으로 생성된 경계 곡선의 EDGE와 FACE를 추가함

⑩ 모델의 상태를 Save-Status로 변경하고 FACE의 포인터를 리턴함

}

나. 곡면을 삭제하는 API

모델로부터 곡면을 삭제하는 API는 삭제할 곡면을 형상 정보로 사용하는 FACE를 입력받아 데이터 구조에서 이를 제거하는 기능을 수행하는 함수이다. 곡면과 FACE를 연결하는 SURFACE 클래스는 여러 FACE의 형상 정보로 사용될 수 있으므로 정수 타입의 use_count를 멤버로 갖고 있으며, 해당 SURFACE가 FACE의 형상 정보로 사용될 때마다 1씩 증가하고, 해당 SURFACE를 형상 정보로 사용하는 FACE가 삭제될 때는 1씩 감소하게 되어 있으며, use_count가 0(zero)이 되면 데이터 구조에서 사라진다. 그러므로 임의의 SURFACE를 형상 정보로 사용하는 FACE가 삭제 되어도 SURFACE는 삭제되지 않을 수 있다. 특히, SURFACE가 pcurve의 도메인으로 사용될 때는 pcurve가 삭제 되어야만 데이터 구조에서 사라지게 된다. 이러한 작업은 오일러 작업자가 담당하는 것이 아니고 SURFACE 클래스의 삭제자(destructor)가 수행한다. 곡면을 삭제하는 API는 오일러 작업자 KFMC()로 입력된 FACE를 삭제하여 모델에 회로(cycle)을 생성하고, EDGE를 삭제하는 API를 호출하여 생성된 회로를 구성하는 EDGE들을 삭제하

는 과정을 통하여 입력된 FACE와 FACE의 형상 정보로 사용하는 곡면을 display list에서 제거하고, 곡면이 더 이상 사용되지 않으면 데이터 구조에서 완전히 제거한다.

```
logical SApi_DelFaceOnDB(FACE *face )
{
        ① 삭제할 수 있는 FACE가 입력 되었는지를 검사함
        ② 입력된 FACE에 inner loop가 있으면 이를 제거함
           (오일러 작업자 MEKL()를 사용하여 inner loop와 outer loop간에
           EDGE를 추가하면 inner loop의 EDGE들이 outer loop의 EDGE로 변경되
           고 위상적으로 inner loop는 사라짐)
        ③ 오일러 작업자 KFMC()을 사용하여 FACE를 제거하고, FACE의 EDGE들로
           모델에 회로를 구성함
        ④ 회로를 구성한 각각의 EDGE들을 삭제함
           (EDGE를 삭제하는 API는 SApi_DelEdgeOnDB(EDGE *) 임)
        ⑤ Display-list EdListOB, FaListOB에서 삭제된 EDGE와 FACE의 포인터를
           제거함
        ⑥ 모델의 상태를 Save-Status로 변경하고 리턴함
}
```

제 5 절 SurfART의 데이터 파일

1. 개 요

형상 모델링 시스템에 의하여 모델링 된 모델을 NC가공 정보 생성, 수치해석, 급속 조형(RP: Rapid Prototyping)을 이용한 제품 제작 등의 응용 작업에서 활용할 수 있도록 데이터 파일로 저장하거나 이미 저장된 데이터 파일로부터 모델을 복원하는 기능은 타 시스템과 마찬가지로 필수적인 기능이다. 모델링 시스템에서 데이터를 저장하는 기능은 데이터 구조에 들어 있는 모델의 데이터를 일정한 형식을 갖는 파일로 쓰는 작업이고, 데이터를 복원하는 기능은 일정한 형식으로 쓰여진 파일을 읽어 데이터 구조에 맞게 모델을 재생하는 작업이다. 일반적으로 컴퓨터에서 사용하는 파일의 종류는 텍스트 파일(text file)과 이진 파일(binary file)이 있다. 텍스트 파일은 ASCII 문자로 쓰여진 파일이고, 이진 파일은 이진수로 구성된 일련의 바이트로 쓰여진 파일이다. 이진 파일은 텍스트 파일보다 디스크 공간을 적게 차지하고, 파일 입출력 속도가 빠른 장점이 있다. 또한, 이진 파일은 일반적으로 사용되는 파일 에티터(editor)로 읽을 수 없으므로 모델링 시스템의 데이터 파일로는 이진 파일을 사용하는 것이 유리하다. 그러나 텍스트 파일은 에티터로 읽어 데이터를 확인할 수 있으므로 시스템을 개발하면서 디버깅을 할 때에는 편리하다. 그러므로 본 연구에서도 데이터 파일을 텍스트 파일로 입출력 하도록 구현하였으며, 디버깅이 완료되면 이진 파일로 변경해야만 한다. SurfART에서 사용하는 데이터 파일은 2개가 있다. 하나는 모델을 입출력하는 파일로 확장자는 .SAM을 사용한다. 다른 하나는 위상 개념이 없는 수많은 점 데이터를 모델과 분리하여 입출력하는 파일로 확장자는 .SAP를 사용한다. 모델 파일은 열기(open)나 저장(save) 메뉴로 입출력 되고, 점 데이터 파일은 점 데이터를 export, import하

는 메뉴에 의하여 입출력 된다.

2. 모델 파일의 구조

SurfART에서 모델을 표현하는 데이터 구조는 위상 정보, 형상 정보, 속성 정보 등은 일정한 클래스로 정의된 요소로 표현되어 있고, 요소간의 상관 관계는 포인터로 연결되어 있다. 따라서 각 요소는 요소별로 각각의 멤버 데이터를 파일에 쓰고, 읽으면 되나, 상관 관계를 표현하는 포인터는 데이터 파일을 입출력할 때마다 변경되는 값이므로 직접 입출력할 수 없다. 그러므로 본 시스템의 데이터 파일은 그림 4.35와 같이 요소가 파일에 쓰여지는 순서로, 각각의 요소에 번호(identification index)를 부여하고, 상관 관계를 나타내는 포인터를 이 번호로 대치하여 파일에 쓰도록 하였다. Null 포인터는 -1로 부여된다. 복원할 때에도 이 번호를 사용하여 요소간의 상관 관계를 연결한다.

< Heading Section >

```
SURFART 1.0 MODEL FILE
1 1 5 6 24 16 13 5 24 24 13 42
```

< Topology Section >

```
#0 body $-1 $0 $-1 #
#0 lump $-1 $-1 $0 $0 #
#0 shell $-1 $-1 $F0 $0 #
#0 face $-1 $1 $L0 $0 $0 forward single null #
...
#0 loop $-1 $-1 $0 $0 #
....
#0 coedge $-1 $3 $1 $6 $E0 $0 $0 forward #
...
#0 edge $-1 $0 $1 $C6 $0 forward 0.000000 1.000000 #
....
#0 vertex $-1 $E5 $0 3 #
....
```

그림 4.35 Structure of the model file

< Surface Section >

```
#0 spline-surface $0 {
{ spline forward
{ nurbs 4 4 open open 8 15 4 11
    0.000000
    ......
    50.000000 -0.000000 0.000000 1.000000
    ......
    0.000000 -50.000000 50.000000 1.000000
}
}
} #
......
```

< Curve Section >

```
#0 intcurve-curve $5 {
{ intcurve $1 $-1
{ nurbc 4 open 8 4
    0.000000
    ......
    0.000000 -50.000000 0.000000 1.000000
    ......
}
0.000000
{ para-curve 4 none 8 4
    0.000000
    .......
    0.000000 8.000000 1.000000
    ......
}
{ null }
}
} #
......
```

< Pcurve Section >

```
#0 pcurve $-1 $-1 $16 {
{ ref 1 0.000000 0.000000 }
} #
......
```

그림 4.35 Structure of the model file (Continued)

< Point Section >

```
#0 point $29 0.000000 -50.000000 0.000000 #
.....
```

< Attribute Section >

```
#0 attrib $-1 $-1 { color 255 255 255 } #
..... #
```

< Transform Section >

```
#0 transform $-1 {
   1.000000 0.000000 0.000000 0.000000
   0.000000 1.000000 0.000000 0.000000
   0.000000 0.000000 1.000000 0.000000
   0.000000 0.000000 0.000000 1.000000
} #
```

그림 4.35 Structure of the model file (Continued)

그림 4.35와 같이 하나의 모델을 저장하는 데이터 파일은 표제 부분(heading section), 위상 부분(topology section), 곡면 부분(surface section), 곡선 부분(curve section), 도메인 곡선 부분(pcurve section), 점 데이터 부분(point section), 속성 부분(attribute section), 변환 정보 부분(transform section) 등으로 구분되며, 파일에 순서대로 기록된다. 각각의 부분에 기록되는 요소와 형식은 다음과 같다.

가. 표제 부분(heading section)

표제 부분은 모델을 복원할 때 입력된 파일이 SurfART의 모델 파일임을 판단하는 정보가 기록되는 부분으로 SurfART의 모델 파일임을 나타내는 설명과 해당 파일에 기록되어 있는 각 요소의 갯수가 쓰여진다.

나. 위상 부분(topology section)

위상 부분은 데이터 구조에서 일정한 클래스로 표현된 모든 위상 요소를 기록하는 부분으로 BODY, LUMP, SHELL, FACE, LOOP, COEDGE, EDGE, VERTEX 순으로 기록된다. 각 요소의 시작과 끝은 '#' 기호로 표현되고 데이터 구조에서 포인터로 연결되는 요소는 '$' 기호 다음에 연결된 요소의 번호가 기록된다.

① BODY

'#'body_index "body" $attribute_index $lump_index $transform_index
'#' "₩n"

② LUMP

'#'lump_index "lump" $attribute_index $next_lump_index $shell_index
$body_index '#' "₩n"

③ SHELL

'#'shell_index "shell" $attribute_index $next_shell_index $child_index
$lump_index '#' "₩n"

④ FACE

'#'face_index "face" $attribute_index $next_face_index $child_index
$shell_index $surface_index sense_data side_data, orientation_data
'#' "₩n"

⑤ LOOP

'#'loop_index "loop" $attribute_index $next_loop_index $coedge_index
$face_index '#' "₩n"

⑥ COEDGE

'#'coedge_index "coedge" $attribute_index $previous_coedge_index
$next_coedge_index $partner_coedge_index $child_index $loop_index
$pcurve_index sense_data '#' "₩n"

⑦ EDGE

```
'#'edge_index "edge" $attribute_index $start_vertex_index
  $end_vertex_index $parent_index $curve_index sense_data
  start_parameter_data end_parameter_data '#' "¥n"
```

⑧ VERTEX

```
'#'vertex_index "vertex" $attribute_index $parent_index $point_index
  use_count_data '#' "¥n"
```

다. 곡면 부분(surface section)

곡면 부분은 모델에 정의된 곡면의 데이터를 기록하는 부분으로 다음과 같은 형식으로 기록된다. 아래 형식에서 surface_type은 곡면을 형상에 따라 구분하는 데이터로 "plane", "cone", "torus", "sphere", "spline" 중에 하나가 기록된다.

```
'#'surface_index surface_type "-surface" $attribute_index '{' "¥n"
'{' surface_type sense_data "¥n"
'{' "nurbs" u_order_data v_order_data u_style_data v_style_data
   no_u_knot_data no_v_knot_data no_u_control_point_data
   no_v_control_point_data "¥n"
   u_knot_data "¥n"
   v_knot_data "¥n"
   control_points_data "¥n"
'}' "¥n"
'}' "¥n"
'}' '#' "¥n"
```

라. 곡선 부분

곡선 부분은 모델에 정의된 모든 3차원 곡선을 기록하는 부분으로 다음과 같은 형식으로 기록된다. Curve_type에는 "straight", "ellipse", "intcurve" 중에 하나가 쓰여지며, curve_style에는 "none", "open", "closed",

"periodic"등이 있다. Curve_type이 intcurve인 경우에는 pcurve가 존재할 수 있으며, 이러한 pcurve는 곡선 부분에 3차원 곡선과 함께 기록된다. pcurve가 없는 경우에는 "{ null }"로 기록된다.

```
'#' curve_index curve_type "-curve" $attribute_index '{' "¥n"
'{' curve_type $surface_1_index $surface_2_index "¥n"
'{' "nurbc" order_data curve_style no_knot_data no_control_points_data
    knot_data "¥n"
    control_points_data "¥n"
'}' "¥n"
fitting_tolerance_data "¥n"
'{' "para-curve" order_data curve_style no_knot_data
 no_control_points_data "¥n"
    knot_data "¥n"
    control_points_data "¥n"
'}' "¥n"
'{' "para-curve" order_data curve_style no_knot_data
 no_control_points_data "¥n"
    knot_data "¥n"
    control_points_data "¥n"
'}' "¥n"
'}' "¥n"
'}' '#' "¥n"
```

마. 도메인 곡선 부분(pcurve section)

모델에서 COEDGE의 형상 정보로 사용하는 도메인 곡선을 기록하는 부분으로 도메인 곡선이 표현되는 타입에 따라 다음과 같이 3가지 방법으로 기록된다. 첫째는 PCURVE가 직접적으로 pcurve를 갖는 경우이고, 둘째는 pcurve가 없이 곡면상에 도메인 위치만을 갖고 있는 경우이다. 세째는 pcurve가 intcurve로 표현된 3차원 곡선에 함께 기록되는 경우이다. 복원할 때에는 surface_index와 curve_index를 이용하여 PCURVE의 타입을 결정한다.

```
① '#'pcurve_index "pcurve" $attribute_index $surface_index
  $curve_index '{' "₩n"
  '{' "type" def_type_data '}' "₩n"
  '{' "para-curve" order_data curve_style no_knot_data
  no_control_points_data "₩n"
     knot_data "₩n"
     control_points_data "₩n"
  '}' "₩n"
  '}' '#' "₩n"

② '#'pcurve_index "pcurve" $attribute_index $surface_index
  $curve_index '{' "₩n"
  '{' "type" def_type_data '}' "₩n"
  '{' parpos_u_data parpos_v_data '}' "₩n"
  '}' '#' "₩n"

③ '#'pcurve_index "pcurve" $attribute_index $surface_index
  $curve_index '{' "₩n"
  '{' "ref" def_type_data  tolerance_u_data tolerance_v_data '}' "₩n"
  '}' '#' "₩n"
```

바. 점 데이터 부분(point section)

점 데이터 부분은 데이터 구조에서 VERTEX의 형상 정보로 사용되는 점 데이터를 기록하는 부분으로 다음과 같은 형식으로 쓰여진다. ISOPT의 형상 정보로 사용되는 점 데이터는 이 부분에 기록되지 않는다.

```
'#'point_index "point" $attribute_index position_x_data
position_y_data position_z_data '#' "₩n"
```

사. 속성 요소 부분(attribute section)

모델에서 사용되는 모든 속성 요소가 기록되는 부분으로 현재 사용되

고 있는 COLOR_ATTRIB는 다음과 같은 형식으로 기록되며, 새로운 속성 요소를 추가할 때에는 클래스에 일정한 형식으로 데이터를 입출력하는 멤버 함수를 정의해야 한다.

```
'#'attribute_index "attrib" $previous_attribute_index
 $next_attribute_index '{' "color" red_value_data green_value_data
 blue_value_data '}' "₩n"
```

아. 변환 정보 부분(transform section)

모델의 전역 변환 행렬(global transformation matrix)을 기록하는 부분으로 다음과 같은 형식으로 쓰여지며, 하나의 모델에는 하나의 전역 변환 행렬이 사용된다.

```
'#'transform_index "transform" $attribute_index '{' "₩n"
   4x4 matrix_data
'}' "₩n"
```

3. 점 데이터 파일의 구조

점 데이터 파일은 SurfART에서 측정 데이터와 같이 수많은 점 데이터를 편리하게 입출력할 수 있도록 하는 파일로 데이터 구조에서 ISOPT의 형상 정보로 사용되는 점 데이터들이 여기에 기록된다. SurfART는 여러 가지 형식의 점 데이터를 입력할 수 있는 기능을 갖고 있으나, 입력된 점 데이터를 파일로 저장하는 기능은 단 하나만 있으며 이 때 생성되는 파일이 *.SAP 확장자를 갖는 점 데이터 파일이다. 점 데이터 파일은 단지 그룹으로 나누어진 점 데이터만을 기록하는 파일이므로 특별한 형식을 갖고 있지는 않으며 그림 4.36과 같은 간단하게 표제 부분(heading section)과 점 데이터 그룹 부분(point group

section)으로 구성된다. 표제 부분에는 파일이 SurfART의 점 데이터 파일임을 나타내는 설명과 파일에 기록된 점 데이터 그룹의 총 갯수가 쓰여진다. 점 데이터 그룹 부분에는 해당 그룹에 속한 모든 ISOPT의 형상 정보들이 기록된다.

```
< Heading Section >
SURFART 1.0 POINTS FILE
4

< Pgroup Section >
#PG 0
#0 -2.350000 0.000000 10.641358 #
#1 -2.350000 1.000000 10.649899 #
......
......

#PG 1
#0 0.000000 0.000000 10.445805 #
#1 0.000000 1.000000 10.463498 #
.....
.....

....
....
```

그림 4.36 Structure of the points file

4. 데이터 파일의 저장/복원

SurfART에서 사용하는 데이터 파일은 모델 파일과 점 데이터 파일이 있다. 이러한 파일들을 생성하거나 디스크에 있는 파일을 시스템으로 복원하는 기능은 API로 구현되어 있으며, 각각은 메뉴와 연결되어 사용자가 편리하게 사용할 수 있다. 모델 파일을 저장/복원하는 기능은 파일 열기/저장(open/save) 메뉴에 의하여 수행되며, 점 데이터 파일을 저장/복원하는 기능은 파일

export/import 메뉴에 의하여 수행된다. 모델 파일을 저장/복원하는 기능에서 가장 중요한 것은 데이터 구조에서 포인터로 표현된 요소간의 연결 관계를 저장하고 복원하는 방법으로 구현된 API에서는 다음의 알고리즘과 같이 데이터 구조에 있는 모든 요소를 조회하여 조회된 순서대로 번호를 부여하여 파일에 기록한다. 이 때 포인터로 표현된 데이터는 해당 요소의 번호로 대치되어 기록된다. 파일을 복원할 때에는 파일에서 순서대로 모든 요소의 데이터를 읽어 각각의 요소를 생성한 후, 번호로 읽혀진 부분을 포인터로 대치한다.

```
logical SApi_WriteBodyToFile( BODY *body, const char *filename )
{
    ① 입력 데이터 검사(body, file name)하고 파일을 생성함
    ② 입력된 body의 모든 요소를 조회하여 요소별로 PTRLIST 클래스로
       보관함
    ③ Heading 정보를 파일에 기록함
    ④ 위상 요소(BODYs, LUMPs, SHELLs, FACEs, LOOPs, COEDGEs, EDGEs,
       VERTEXs)들을 순서대로 번호를 부여하여 파일에 기록함
    ⑤ 형상 요소(SURFACEs, CURVEs, PCURVEs, GPOINTs)들을 순서대로
       번호를 부여하여 파일에 기록함
    ⑥ 속성 요소(COLOR_ATTRIBs)를 순서대로 번호를 부여하여 파일에
       기록함
    ⑦ 변환 정보(TRANSFORM)를 파일에 기록함
    ⑧ 파일을 close하고 return함
}
```

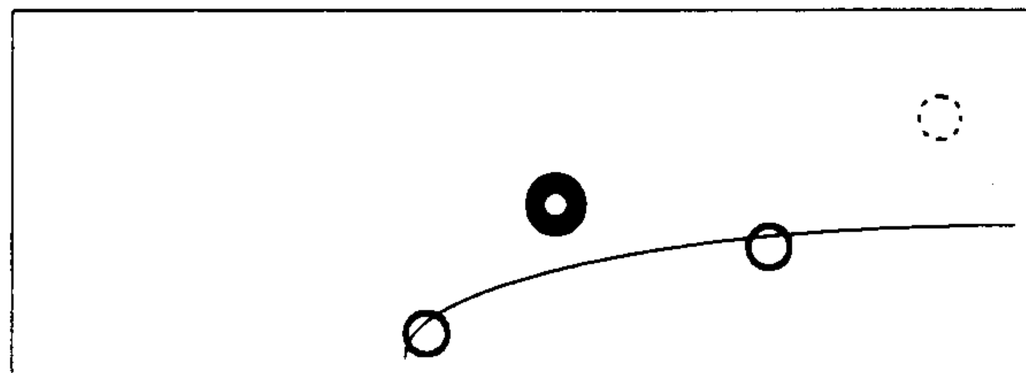
여 백

제 5 장 측정 데이터의 Fairing

제 1 절 개 요

측정 데이터를 이용하여 analytic 한 자료(data)들에 대하여 그 자료그룹(data group)의 형상에 따라 수식적인 계산을 통하여 3 차원적 직선(straight line), 3 차원적인 평면 (plane), 2 차원적인 원(circle), 2 차원적인 호(arc) 등으로 표현하게 되고 이들 각의 수식적인 계산을 통하여 모든 점들을 가장 만족 하는 Analytic 한 형상을 찾아 내게 되고 이 형상을 화면을 통하여 모델링(modeling) 하게 된다. 뿐 만 아니라, 형상 자체가 Analytic 하지 않을 경우 측정 자료(digitizing data or scanning data)를 이용하여 실제 형상에 가까운 자료(data)를 얻기 위하여 Filtering 을 하게 되고 이 Filtering 방법은 세 가지로 median, average, Gaussian 방법에 의해 계산하게 되는데 사용자(user)에 의하여 그 용도에 따라 선택하게 되고 각각의 루틴 들이 호출되어 오류(error)가 포함되어 있는 모든 점(point)을 만족하는 형상을 구현하게 된다. 그 형상 자료(data) 자체의 양이 아주 방대 함으로 형상의 모델링(modeling)을 하는데 과다한 메모리를 차지하게 되므로서처리 하는 시간도 길어 질 수 밖에 없다. 그런 이유에서도 필요한 만큼에 자료(data)를 제외하고 제거하게 되는 data reduction 을 하므로 처리속도의 증대 또는 할당메모리의 양을 줄일 수 있는 장점을 가질 수 있다. 이 data reduction 은 형상 자체가 불규칙 부분에서는 그 원래의 형상을 보존하기 위하여 많은 자료(data)를 보존할 필요성이 있게 되고 형상이 규칙적인 부분에서는 적은 자료(data)를 가지고도 형상을 표현 하는데 문제점이 없기 때문에 자료(data)의 양을 줄일 필요가 있게 된다. 하지만 규칙적인 부분이 라도 일정한 간격(interval)의 자료(Data)는 보존 하게 되므로 서 형상을 가지고 surface 를 생성하게 되는데 용

이 하게 되므로 서 일정한 간격(interval)의 자료(data)는 그대로 보존 하고 있도록 data reduction을 하고 있다. 그림 5.1 은 data reduction의 간단한 예를 보여 주고 있다[5-1].



● 제거 되어져야 할 점 중 일정한 간격에 존재하는 point data

◌ 제거 될 point data

○ 제거 되지 않는 point data

그림 5.1 data reduction의 예

제 2 절 직선(straight line)의 수학적 표현

일반적으로 최소자승 법(least square method)을 이용하여 계산하게 되는데 그림 5.2 은 점(point)들의 정규화(normalize)를 나타내고 있는데 아래에 있는 수식을 이용하여 아래 normalize 식과 같이 정규화(normalize)하여 각 점들의 사이의 거리를 0 과 1 사이로 매핑하여 x, y, z 에 대하여 각각의 최소 자승법(Least-square-method)을 적용하여 각각의 계수 값을 계산하여 그 계수 값들을 이용하여 3 차원적인 직선을 구현 할 수 있다. 아래의 그림에서와 같이 정규화(normalize)는 식 (5.1)을 이용하여 t0, t1, t2,…등을 구 할 수 있다 [5-2][5-3].

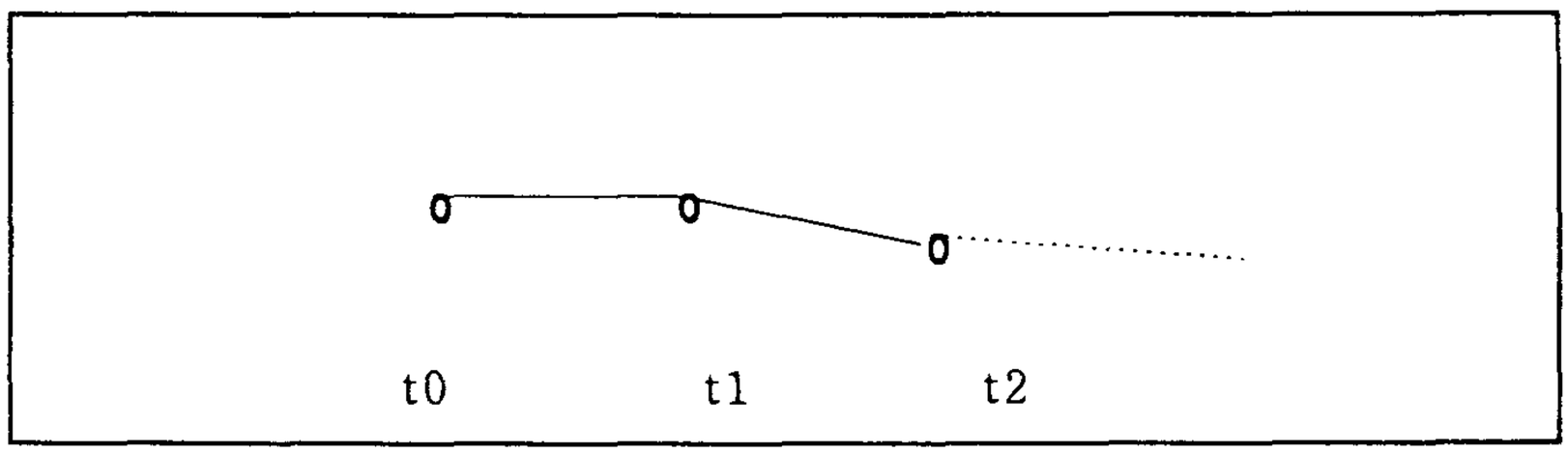


그림 5.2 point 간의 거리의 Normalize

$$t0 + d = t1$$

$$t1 + d = t2$$

$$d = (x(i) - x(i-1))^2 + (y(i) - y(i-1))^2 + (z(i) - z(i-1))^2 \ldots.$$

(5.1)

Normalize : $t0 = 0$

$t1 = d(t1)/ d$

$t2 = d(t2)/ d$

x 축의 좌표를 보면

$$\begin{bmatrix} \sum_{i=1}^{n} x \\ \sum_{i=1}^{n} (xt) \end{bmatrix} = \begin{bmatrix} k\_x0 \\ k\_x1 \end{bmatrix} \begin{bmatrix} N, \sum_{i=1}^{n} t \\ \sum_{i=1}^{n} t, \sum_{i=1}^{n} (t \times t) \end{bmatrix} \qquad (5.2)$$

식 (5.2)를 이용하여 계수 값 k_x0, k_x1 을 구하게 되고

y 축의 좌표를 보면

$$\begin{bmatrix} \sum_{i=1}^{n} y \\ \sum_{i=1}^{n} (yt) \end{bmatrix} = \begin{bmatrix} k\_y0 \\ k\_y1 \end{bmatrix} \begin{bmatrix} N, \sum_{i=1}^{n} t \\ \sum_{i=1}^{n} t, \sum_{i=1}^{n} (t \times t) \end{bmatrix} \qquad (5.3)$$

식 (5.3)을 이용하여 계수 값 k_y0, k_y1 을 구하게 되고

z 축의 좌표를 보면

$$\begin{bmatrix} \sum_{i=1}^{n} z \\ \sum_{i=1}^{n} (zt) \end{bmatrix} = \begin{bmatrix} k\_z0 \\ k\_z1 \end{bmatrix} \begin{bmatrix} N, \sum_{i=1}^{n} t \\ \sum_{i=1}^{n} t, \sum_{i=1}^{n} (t \times t) \end{bmatrix} \qquad (5.4)$$

식 (5.4)를 이용하여 계수 값 k_z0, k_z1 을 구하게 되고

각 각의 축에서 구해진 계수 값 k_x0, k_x1, k_y0, k_y1, k_z0, k_z1 을 이용하여 두 점(data)의 좌표(coordinate value)인 (k_x0, k_y0, k_z0)와 (k_x1, k_y1, k_z1)를 얻을 수 있고 이 좌표를 이용하여 직선을 구현 하게

되면 이 직선은 3차원적으로 모든 점(point)에서의 거리를 최소화(minimize) 시키는 직선을 구현 하게 된다. 직선을 3차원적으로 구현 하기 위해 매개 변수를 이용하였고 이 매개 변수의 의미는 3차원적인 두 점간의 거리에 대한 x, y, z의 좌표 값에 비율을 나타낸다.

제 3 절 평면(plane)의 수학적 표현

평면의 구현 역시 최소 자승 법(least-square-method)에 의하여 구현할 수 있는데 각 점(point)들에서 최소(minimize)의 거리에 존재하는 평면을 찾아내게 된다. 그림 5.3는 그것을 나타내고 있다.

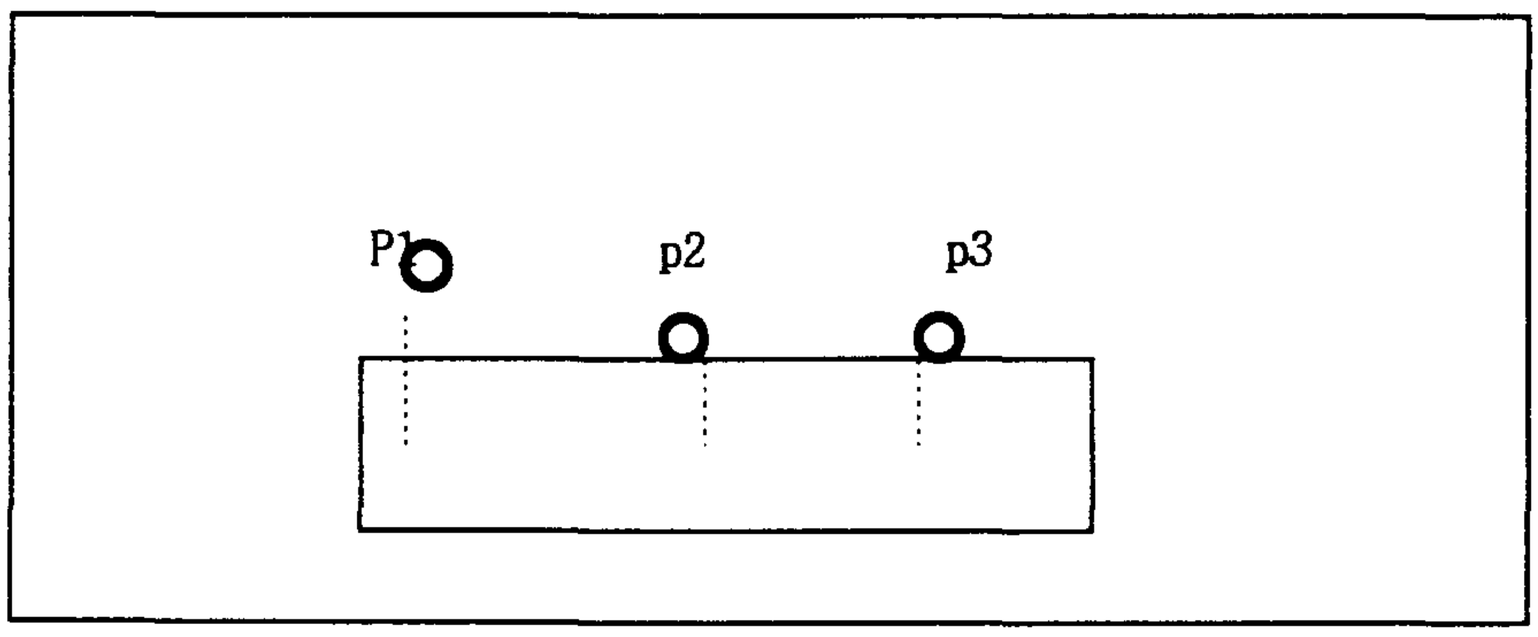


그림 5.3 각 점들에서 최소 거리의 평면

이 경우는 아래 식에서 표현 하고 있는 행렬식 X와 같이 표현하게 되고 식 (5.5)를 이용하여 행렬식 U 값들을 계산 하게 되는데 행렬식 U는 계수 값(coefficient)인데 이 식을 계산 하여 계수(coefficient)값을 구해 내기 위하여 식 (5.6), Cholesky Equation을 사용 하여 계산 하게 된다[5-4].

$$XU = B \qquad (5.5)$$

$$XX^{T}U = X^{T}B \ : \text{Cholesky Equation} \qquad (5.6)$$

$$X = \begin{bmatrix} x1, y1, z1 \\ x2, y2, z2 \\ \\ \\ xn, yn, zn \end{bmatrix}$$ 각 점들의 좌표의 행렬식

X^T = X 의 Transpose

$$U = \begin{bmatrix} u1 \\ u2 \\ \vdots \\ un \end{bmatrix}$$ 계수의 행렬식 , $$B = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$$

식 (5.6)에 의해 얻어진 계수 값들은 가지고 각 점(point)들에서 최소 거리(minimal length)에 있는 평면에 존재 하는 세 점(three point)을 얻어 내게 되고 이것들을 이용하여 평면을 구현 하게 된다.

제 4 절 원(circle)의 방법론

각 점(point)들을 이용하여 각각의 거리가 최소인 중심점을 구해 내고, 중심점을 이용 하여 모든 점까지의 거리가 최소인 반지름을 구하게 되므로서 오차(error)가 포함되어 점들을 가장 만족 하는 원을 구할 수 있다. 그림 5.4 은 각 점 (point)들에서 중심을 구하는 방법을 나타내고 있다.

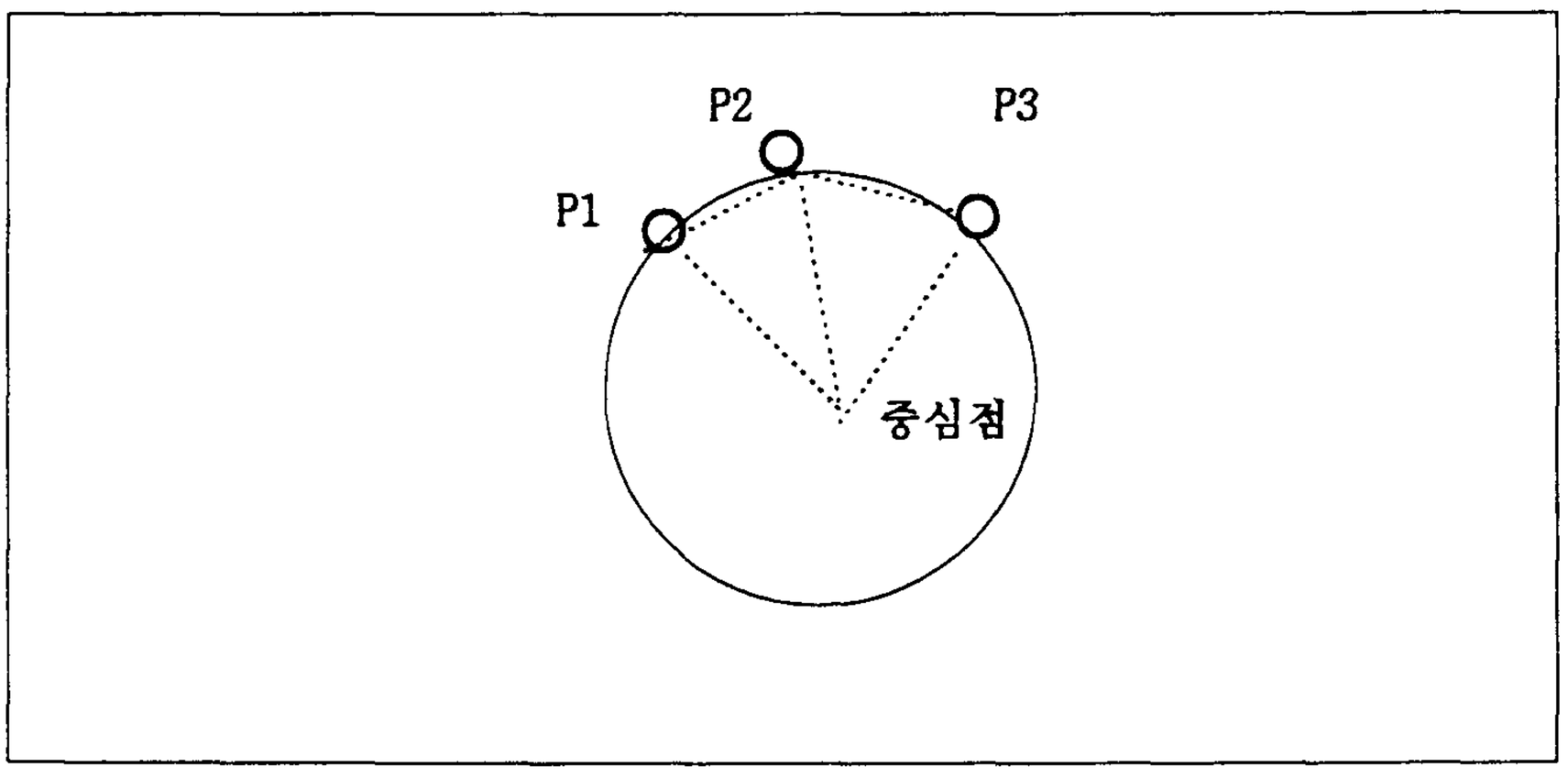


그림 5.4 원(circle)에서 각 점(point)에서 기하적으로 중심점을 찾는 방법

위의 그림에서와 같이 얻어진 중심점(center point)에서 부터 각 점(point)들에 거리의 평균 값을 반지름(radius)을 이용하여 원(circle)을 구현하게 된다.

그 기본적인 알고리즘(algorithm)은 아래와 같다.

(기본 알고리즘)

i) 각 각의 선택된 점들(point) 자료 (data) pi 를 입력 받는다.

ii) 점들의 자료(point data)를 한 점씩 진행 한다.

iii) 세 점(point)에서 양끝 점과 중심점을 있는 직선을 두개를 구현하고 그 직선부터 수직인두개의 직선을 이용하여 중심점을 찾아 낸다.

iv) 이 중심점으로 부 터 각 점(point)들의 평균 거리를 구하여 반지름(radius)을 찾아 낸다.

v) iv)에서 구해진 값들을 이용하여 원을 구현한다.

제 5 절 호(arc)의 방법론

각 점(point)들을 이용하여 각각의 거리가 최소인 중심점을 구해 내고, 중심점(center point)을 이용 하여 모든 점까지의 거리가 최소인 반지름을 구하게 되므로서 오차 (error)가 포함되어 점들과 양끝 점들을 가장 만족하는 호(arc)를 구할 수 있다. 그림 5.5은 각 점들에서 호(arc) 중심을 구하는 방법을 나타내고 있다.

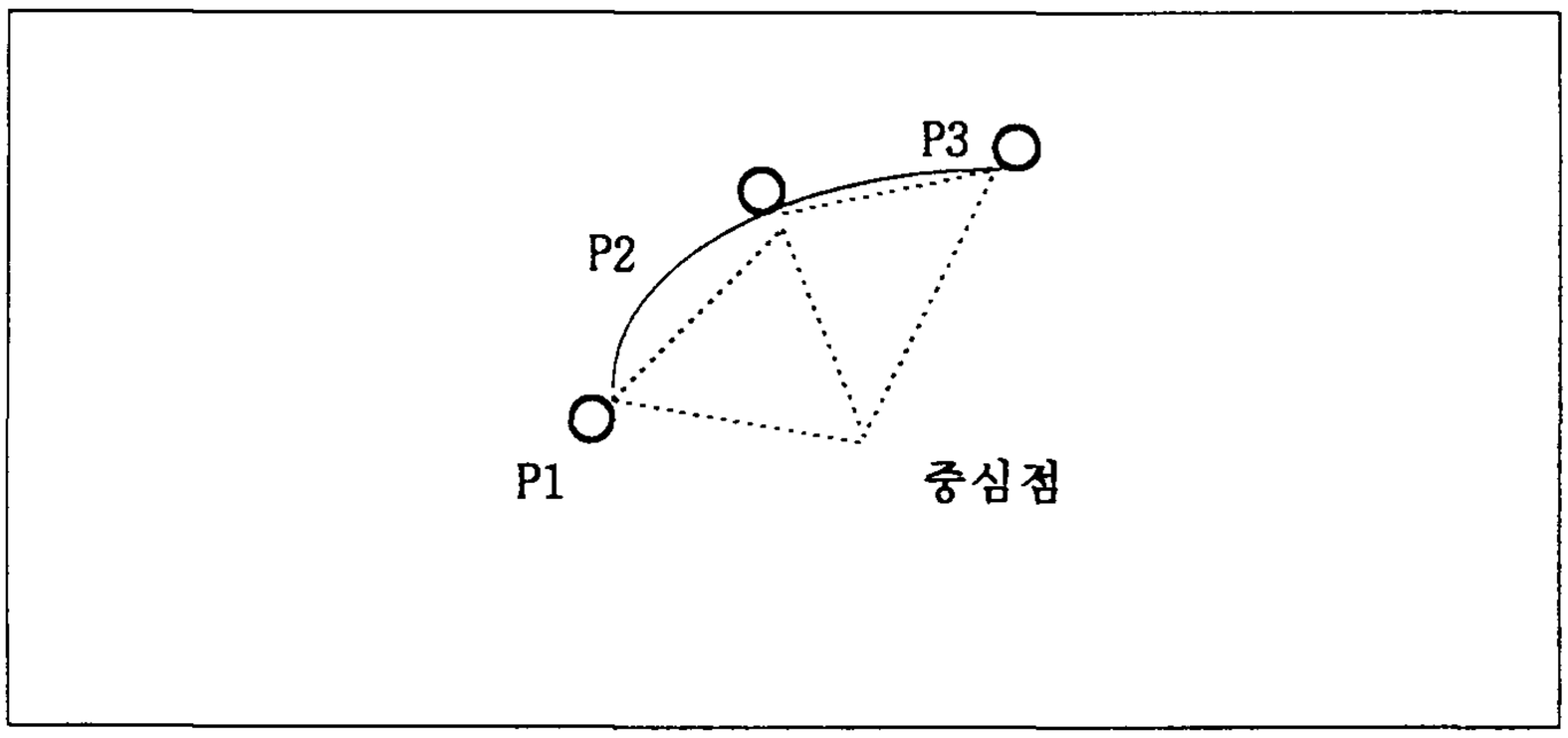


그림 5.5 호(arc)에서 각 점(point)에서 기하적으로 중심점을 찾는 방법

위의 그림에서와 같이 얻어진 중심점(center point)에서 부터 양끝 점을 포함 한 각 점(point)들에 거리의 평균 값을 반지름(radius)을 이용하여 호(arc)를 구현 하게 된다[5-5].

그 기본적인 알고리즘(algorithm)은 아래와 같다.

(기본 알고리즘)

i) 각 각의 선택된 점들(point) 자료 (data) Pi 를 입력 받는다.

ii) 점 (point)들의 자료(point data)를 한 점씩 진행 한다.

ii) 세 점(point)에서 양끝 점과 중심점을 있는 직선을 두개를 구현하고 그 직선부터 수직인 두개의 직선을 이용하여 중심점을 찾아 낸다.

iv) 이 중심점으로 부 터 양끝 점을 포함한 각 점(point)들의 평균 거리를 구하여 반지름(radius)을 찾아낸다.

v) iv)에서 구해진 값들을 이용하여 호를 구현한다.

제 6 절 Filtering 방법론

측정 자료(scanning data)가 포함하고 있는 오차(error)라든지 측정 시 불규칙적인 반사등으로 일어 나게 되는 오차(error)부분을 없애고 정형화 하는데 목적이 있다. 특히 median filtering 방법, average filtering방법, Gaussian filtering에 대하여 자세히 기술적인 측면을 기술한다.

1. Median filtering

각 점 군(point group)의 점(point)들을 p(i), p(i+1), ... ,p(i+n)라 정의할 때 세 점(point)의 통계적 median value를 취하므로 오차(error)를 smooth하게 정형하는 방법이다. 이 방법은 2차원 상에서 각 각의 세 점(three point)에 대하여 통계적(statistical) median 값을 취하게 된다[5-1].

세 점(point), pi, p(i+1), p(i+2)라 가정하고 세 점(point)의 좌표가 pi = (3,4) , p(i+1) = (4,6), p(i+2) = (5,5)라 할 때

x축을 중심으로 볼 때

pi, p(i+1), p(i+2) 통계적 median값은 3, 4, 5 중 4가 되고

y축을 중심으로 볼 때

pi, p(i+1), p(i+2) 통계적 median값은 4, 5, 6 중 5가 된다.

그래서 결과적으로 (4, 5)의 점을 취하게 되고 그 점으로 이동 시키게 된다.

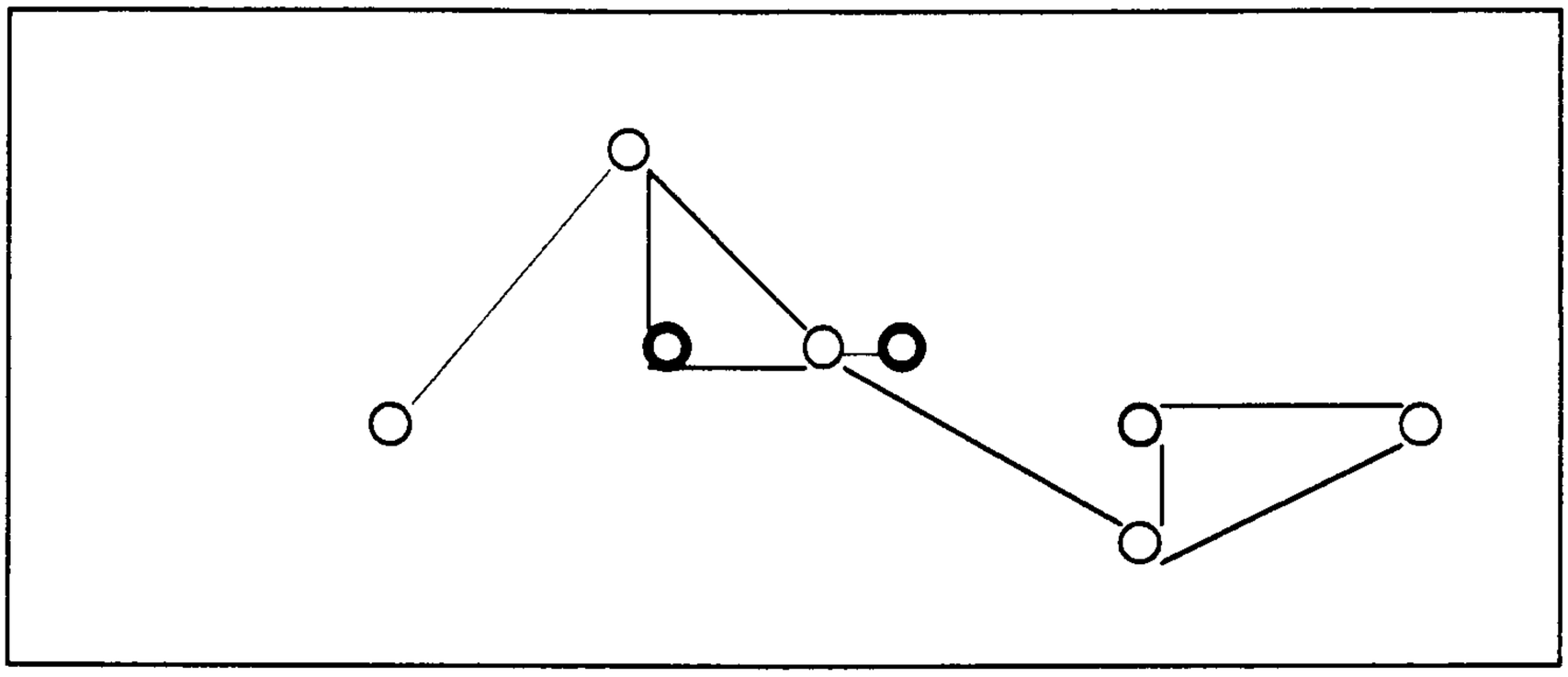


그림 5.6 Median filtering의 point의 변화

그 기본적인 알고리즘(algorithm)은 아래와 같다.

(기본 알고리즘)

i) 점의 자료(point data) pi를 입력 받는다.

ii) 점의 자료(point data)를 한 점씩 진행 한다.

iii) 세 점(point)의 통계적 median value를 계산한다.

iv) 세 점(point)의 중간의 점(point)을 median value에 의해 수정하여 구현 한다.

2. Average filtering

각 점 군(point group)의 점(point)들을 p(i), p(i+1), ... ,p(i+n)라 정의 할 때 세 점(point)의 통계적 무게중심 (average value)을 취하므로 오차(error)를 smooth하게 정형하는 방법이다[5-1].

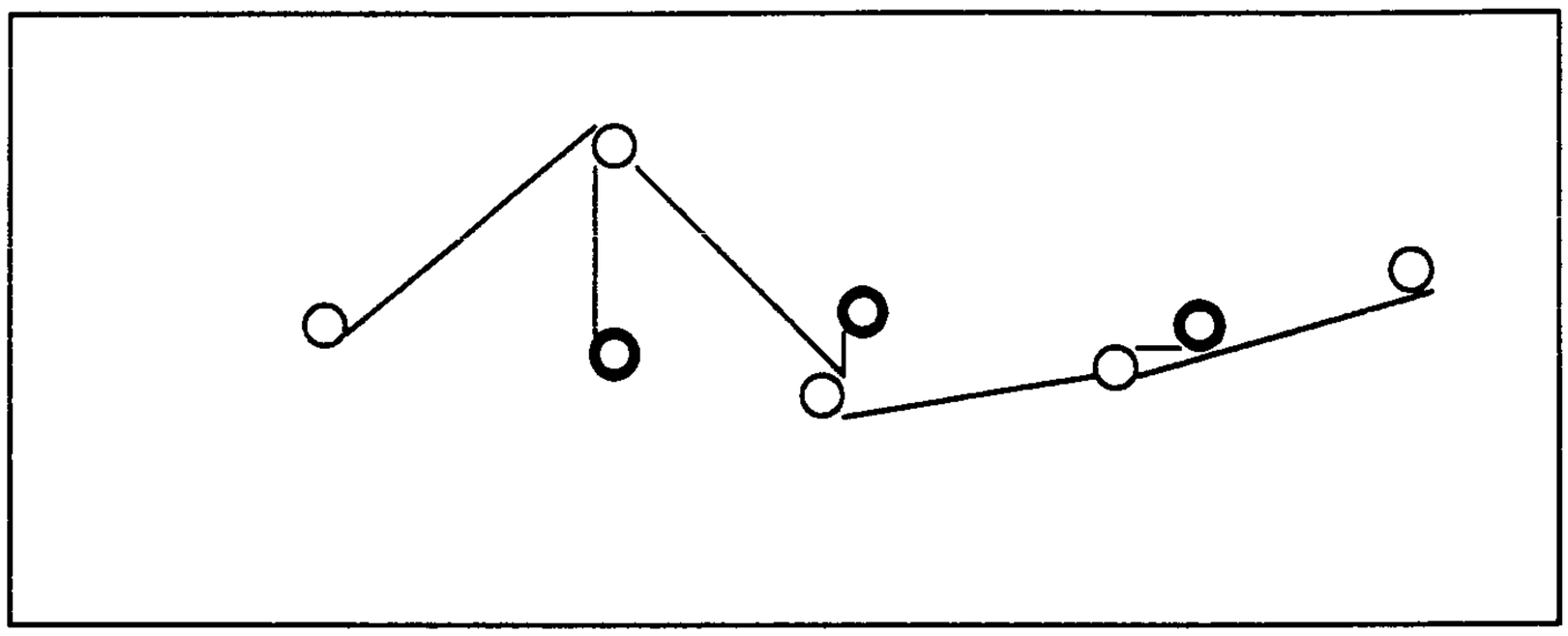


그림 5.6 Average filtering의 point 변화

이 방법은 2차원 상에서 각각의 세 점(three point)에 대하여 통계적(statistical) 평균(average) 값을 취하게 된다.

세 점(point), pi, p(i+1), p(i+2)라 가정하고 세 점(point)의 좌표가 pi = (3,4) , p(i+1) = (4,6), p(i+2) = (5,5)라 할 때

x축을 중심으로 볼 때

pi, p(i+1), p(i+2) 통계적 평균(statistical average)값은 {pxi+px(i+1)+px(i+2)}/3 (3+4+5)/3 = 4

y축을 중심으로 볼 때

pi, p(i+1), p(i+2) 통계적 평균(statistical average)값은 {pyi+py(i+1)+py(i+2)}/3 (4+4+5)/3 = 4.333 이다.

그래서 결과적으로 (4, 4.333)의 점을 취하게 되고 그 점으로 이동 시키게 된다.

그 기본적인 알고리즘(algorithm)은 아래와 같다.

(기본 알고리즘)

i) 점의 자료(point data) pi 를 입력 받는다.

ii) 점의 자료(point data)를 한 점씩 진행 한다.

iii) 세 점(point)의 통계적 무게중심 (average value)을 계산 한다

iv) 세 점(point)의 중간의 점(point)을 무게 중심 (average value)에 의해 수정하여 구현 한다.

3. Gaussian filtering

각 point group 의 point 들을 p(i), p(i+1), ... ,p(i+n)라 정의 할 때 세 점(point)의 통계적 무게 중심(average value)을 취하고 중간 점(point)과 무게중심과의 가중치(weight)를 계산하여 가중치를 고려한 중심점을 찾아 이동시키므로 서 오차(error)를 smooth 하게 정형하는 방법 이다[5-1]. 아래 그림 4.7 에서와 같이 점들이 이동 하게 되고 원래의 점 군 (point group)으로 형성된 곡선 보다 smooth 한 곡선을 얻게 됨을 알 수 있다. 물론 아주 오차(error)를 많이 포함하고 있는 점들이 점 군(point group)사이 존재 할 때는 그 결과가 그리 좋지는 못하게 된다.

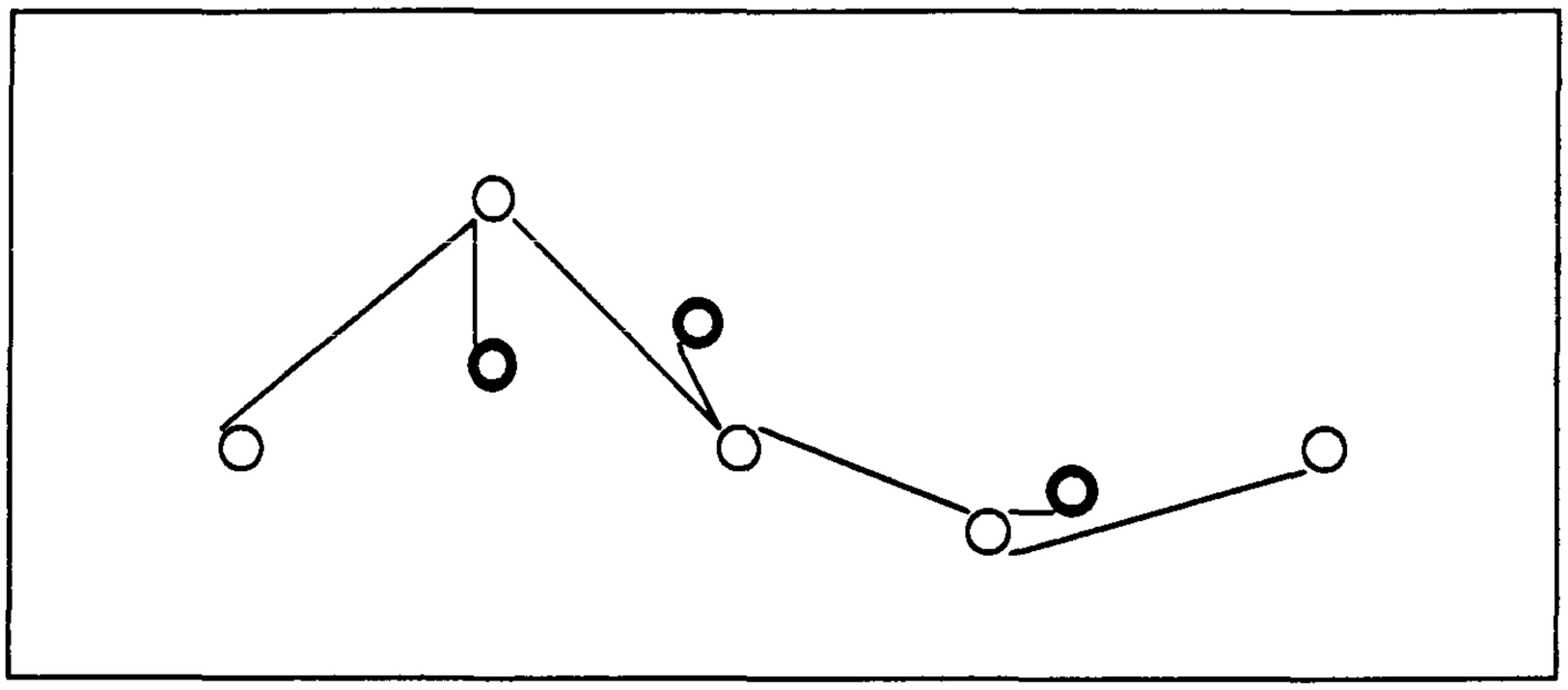


그림 5.7 Gaussian filtering의 point 변화

이 방법은 2차원 상에서 각 각의 세 점(three point)에 대하여 통계적 (statistical) 평균(average) 값을 가중치(weight)를 고려하여 취하게 된다.

세 점(point), pi, p(i+1), p(i+2)라 가정하고 세 점(point)의 좌표가 pi = (3,4) , p(i+1) = (4,6), p(i+2) = (5,5)라 할 때

x축을 중심으로 볼 때

pi, p(i+1), p(i+2) 통계적 평균(statistical average)값은

{pxi+px(i+1)+ px(i+2)}/3 (3+4+5)/3 = 4

y축을 중심으로 볼 때

pi, p(i+1), p(i+2) 통계적 평균(statistical average)값은

{pyi+py(i+1)+ py(i+2)}/3 (4+4+5)/3 = 4.333 이다.

그래서 (4, 4.333)의 점에 가중치(weight)를 고려하여 취하게 되고 그 점으로 이동 시키게 된다.

그 기본적인 알고리즘(algorithm)은 아래와 같다.

(기본 알고리즘)

i) 점의 자료(point data) pi 를 입력 받는다.

ii) 점의 자료(point data)를 한 점씩 진행 한다.

iii) 세 점 point 의 통계적 무게 중심(average value)을 계산 한다

iv) 세 점(point)의 중간의 점(point)을 무게 중심(average value)과 그 point 의 x,y 좌표를 계산한다.

v) 가중치가 고려된 새로운 중심점으로 수정하여 구현한다.

제 7 절 Data reduction 의 방법론

점의 자료(data point)의 수를 최소화 하여 형상을 표현하고자 하는 방법으로써 주로 사용자(user)가 정의한 chordal deviation 의 제약과 일정한 간격(interval)에 존재하는 제약을 고려하여 point data 의 수를 줄이게 된다. 이 두 가지 제약 조건이 만족되는 point data 를 제외하고 나머지 자료를 제거 함으로 data 양을 축소 시킴 으로서 모델링 시 메모리 공간을 적게 차지한다든지 자료(data)를 처리하는 시간적의 이득을 얻을 수 있다[5-1].

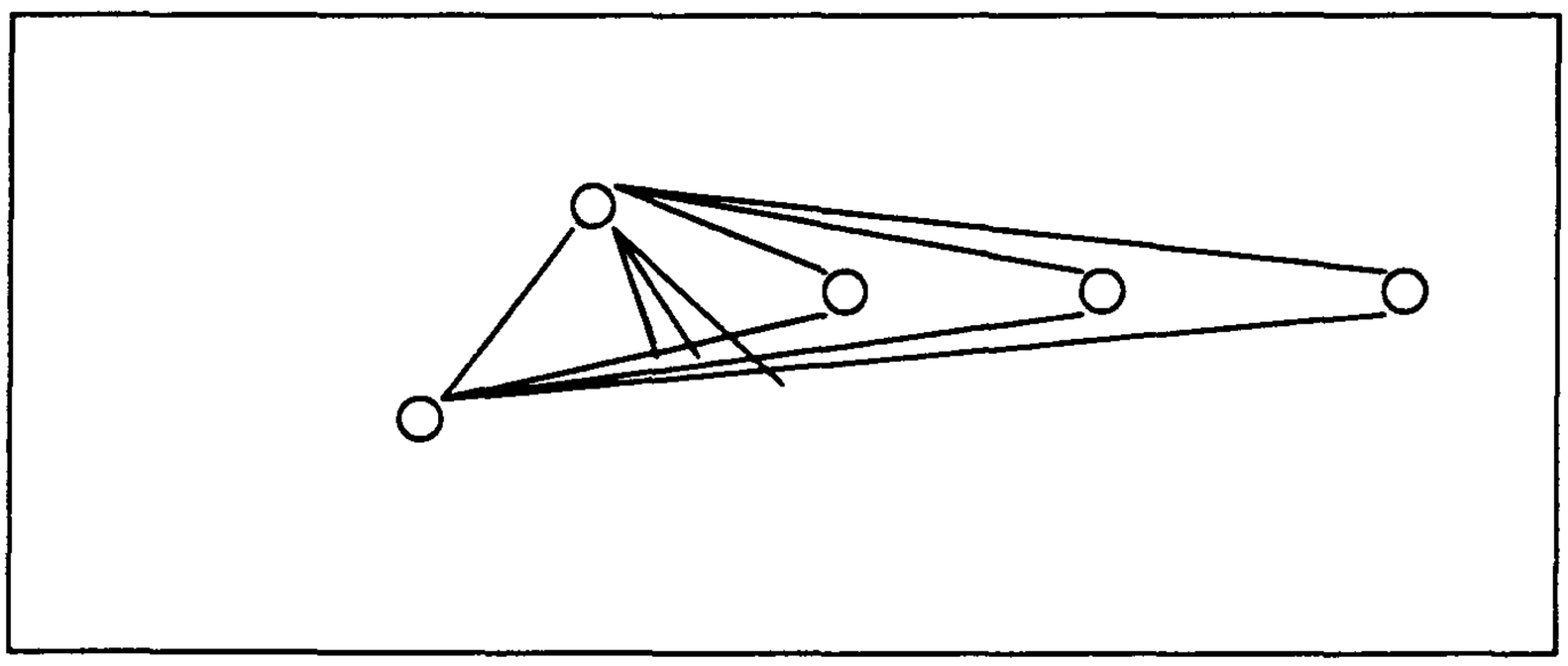


그림 5.8 Chordal deviation 을 이용한 data reduction

1. Data Reduction 의 방법

point data 를 일정한 간격(interval)상의 점(point)은 제거하지 않고 그 점(point)을 제외하고는 모든 점(point)에서 chordal deviation 을 계산하여 사용자(user)에 의해 정의된 value 이하 일 때는 점(point)을 제거 함으로써 점(point)의 수를 줄 일수 있다. 그림 4.8 에서와 같이 양 끝점(point)으로 있는 직선과 중간 점(point)을 잇는 수직선과의 교차점으로 부터 중간 점(point)

과의 거리 즉 chordal deviation 를 사용자(user)에 의해서 정의된 한계 범위라든지 아니면 default 로 정의된 한계 범위에 의해서 점들의 자료(point data)를 줄이게 된다[5-1]. 이 한계 범위를 결정 하는 데는 몇 가지 고려 할 사항이 있다. 그 자료(data) 자체가 전반적으로 굴곡의 변화가 많이 있는 형태 라면 한계 범위를 크게 하므로 서 좋은 결과를 얻을 수 있는 가능성이 높아지고 만약 반대의 경우로 전반적으로 굴곡이 완만한 형태 라면 한계 범위를 작게 함으로서 좋은 결과를 얻을 수 있는 가능성이 높다. 하지만 자료(data)의 형태로서 구간 별로 오류(error)가 어느 정도 포함되어 있어 형상의 굴곡에 영향을 미치는지를 알아 내는데 문제점을 가지고 있다. 이러한 부분을 앞으로 더 연구하여야 할 부분으로 고려 되어야 한다.

그 기본적인 알고리즘(algorithm)은 아래와 같다.

(기본 알고리즘)

i) 점의 자료(point data) pi 를 입력 받는다.

ii) 점의 자료(point data)를 pi 점(point)을 기준점으로 하고 또 다른점들을 p(i+1),p(i+2)부 터 한 점씩 진행 시킨다.

iii) 세 점(point)의 chordal deviation 을 계산한다

iv) 사용자(user)에 의해 chordal deviation 에 limit value 를 정의 된다.

v) 세 점(point)의 중간의 점(point)의 chordal deviation 이하 일 경우 그 점(point)을 제거 하는데 일정한 간격(interval) 에 존재할 경우 제거 하지 않는다.

vi) 기준점은 제거되지 않는 점으로 계속 이동 하며 ii)와 같이 진행되면서 iii) 에서 v)까지 절차를 반복 한다.

제 8 절 Fairing을 위한 API function

1. 직선(straight line)

입력 : pts: 선택된(click) 점(point)들의 좌표 값 [position],

➔ 선택된(click) 점의 좌표 값을 제공 한다.

NoPts: 선택된(click) 점(point)들의 수 [long]

➔ 선택된(click) 점들의 숫자를 제공 하므로 서 그 숫자에 의해 점들의 입력 회수를 결정 한다.

출력 : pts1, pts2: 내부 계산에 의한 두 점(two point) [position]

입력된 pts, NoPts에 의해 내부적으로 최소 자승 법(least-square method)에 의해 계산 하고 두 점(point)을 출력 한다.

2. 평 면(plane)

입력 : Pts: 선택된(click) 점(point)들의 좌표 값 [position],

➔ 선택된(click) 점의 좌표 값을 제공 한다.

NoPts: 선택된(click) 점(point)들의 수 [long]

➔ 선택된(click) 점들의 숫자를 제공 하므로 서 그 숫자에 의해 점들의 입력 회수를 결정 한다.

출력 : pts1, pts2,pts3: 내부 계산에 의한 세 점(three point) [position]

➔ 입력된 pts, NoPts에 의해 내부적으로 최소 자승 법(least-square method)에 의해 계산 하고 두 점(point)을 출력 한다.

3. 원(circle)

입력 : Pts: 선택된(click) 점(point)들의 좌표 값 [position],

➔ 선택된(click) 점의 좌표 값을 제공 한다.

NoPts: 선택된(click) 점(point)들의 수 [long]

➔ 선택된(click) 점들의 숫자를 제공 하므로 서 그 숫자에 의해 점들의 입력 회수를 결정 한다.

출력 : Cen: 원의 중심점(center position) [position],

➔ 입력에서 주어진 선택된 점들의 기하학적인 계산에 의해 중심점을 구해 내고 이 점을 이용 하여 원을 구현 한다.

Rad: 원의 반지름(radius)[long]

➔ 선택된 모든 점들을 만족 하는 반지름을 계산하고 이 값을 이용 하여 원을 구현 한다.

4. 호(arc)

입력 : pts: 선택된(click) 점(point)들의 좌표 값 [position],

➔ 선택된(click) 점의 좌표 값을 제공 한다.

NoPts: 선택된(click) 점(point)들의 수 [long]

➔ 선택된(click) 점들의 숫자를 제공 하므로 서 그 숫자에 의해 점

들의 입력 회수를 결정 한다.

출력 : cen: 호의 중심점(center position) [position],

➔ 입력에서 주어진 선택된 점들의 기하학적인 계산에 의해 중심점을 구해 내고 이 점을 이용 하여 원을 구현 한다.

Rad: 호의 반지름(radius)[long]

➔ 모든 선택 점 들을 만족하는 반지름을 구해 내고 이 점을 이용하여 호를 구현 한다.

pts1, pts2: 양 끝점(two side point) 좌표 값 [double]

➔ 호를 구성하는 끝 두 점과 중심점을 이용 하여 호를 구현 한다.

5. Filtering

입력 : FiPts: 선택된(click) 점 군(point group)들의 좌표 값 [position],

➔ 점 군 들을 선택 하여 점 군의 각각의 점들을 입력으로 이용 한다.

NoFiPts: 선택된(click) 점 군(point group)의 점(point)의 수 [long]

➔ 선택 된 점 군의 점의 수를 이용하여 점들의 수 만큼 입력 자료를 반복 하여 계산 한다.

출력 : FiPts: 계산에 의해 얻어진 점 군(point group)들의 좌표 값[position],

➔ 내부적 계산에 의해 새로운 점 군을 계산하여 출력 한다.

NoFiPts: 계산에 의해 얻어진 점 군(point group)의 점(point)의 수 [long]

6. Data reduction

입력 : RePts: 선택된(click) 점 군(point group)들의 좌표 값 [position],
ReNoPts: 선택된(click) 점 군(point group)의 점(point)의 수 [long]

출력 : RePts: 계산에 의해 줄여진 점 군(point group)들의 좌표 값 [position],
ReNoPts: 계산에 의해 줄여진 점 군(point group)의 점(point)의 수 [long]

여 백

제 6 장 응용 곡선,곡면의 생성

제 1 절 곡면 모델 : NURP

1. NURP(Non-Uniform Rational Power-Basis Polynomial)

가. 특징

NURBS type의 곡선/곡면/Edge(curve on surface)와 완벽한 상호 변환 가능하고, 메모리가 많이 요구되나 계산 속도 빠르다.

(1) 기존의 Rational polynomial과 NURP와의 비교

기존의 rational polynomial은 uniform한 경우와 하나의 patch 내에 있을 경우에는 NURBS와 정확하게 일치하지만, edge가 non-uniform한 patch간에 존재할 경우 기존의 rational polynomial에서는 왜곡이 발생하는데 non-uniform을 처리하는 NURBS와 비교해 볼 때, patch와 patch사이의 간격이 커지면 커질수록 그 에러는 커진다(그림 6.1). 이런 왜곡으로 인한 에러의 누적이 계속될 경우 곡면의 형상에 아주 나쁜 결과를 초래할 수 있기 때문에 non-uniform한 경우를 처리할 수 있도록 patch간의 간격을 표현하는 knot vector를 고려한 rational polynomial인 NURP를 이 연구에서 제안한다. 그림 6.2에서는 uniform rational polynomial과 non-uniform rational polynomial의 evaluation 과정을 보여주고 있는데 uniform rational polynomial일 경우 patch의 간격과는 관계없이 parameter를 uniform하게 계산함으로써 non-uniform한 곡면에서 patch와 patch사이에 왜곡이 발생한다. 하지만 NURP 곡면은 patch 간격을 고려한 knot vector를 사용함으로써 왜곡이 없는 NURBS와 동일한 곡면을 얻을 수 있다. 그리고 그림 6.3에서처럼 NURP 곡면의 모든 점들은

tangent 연속을 만족함으로 Ferguson 곡면이나 chord-length B-spline 곡면에 비해 훨씬 부드러운 곡면을 얻을 수 있다.

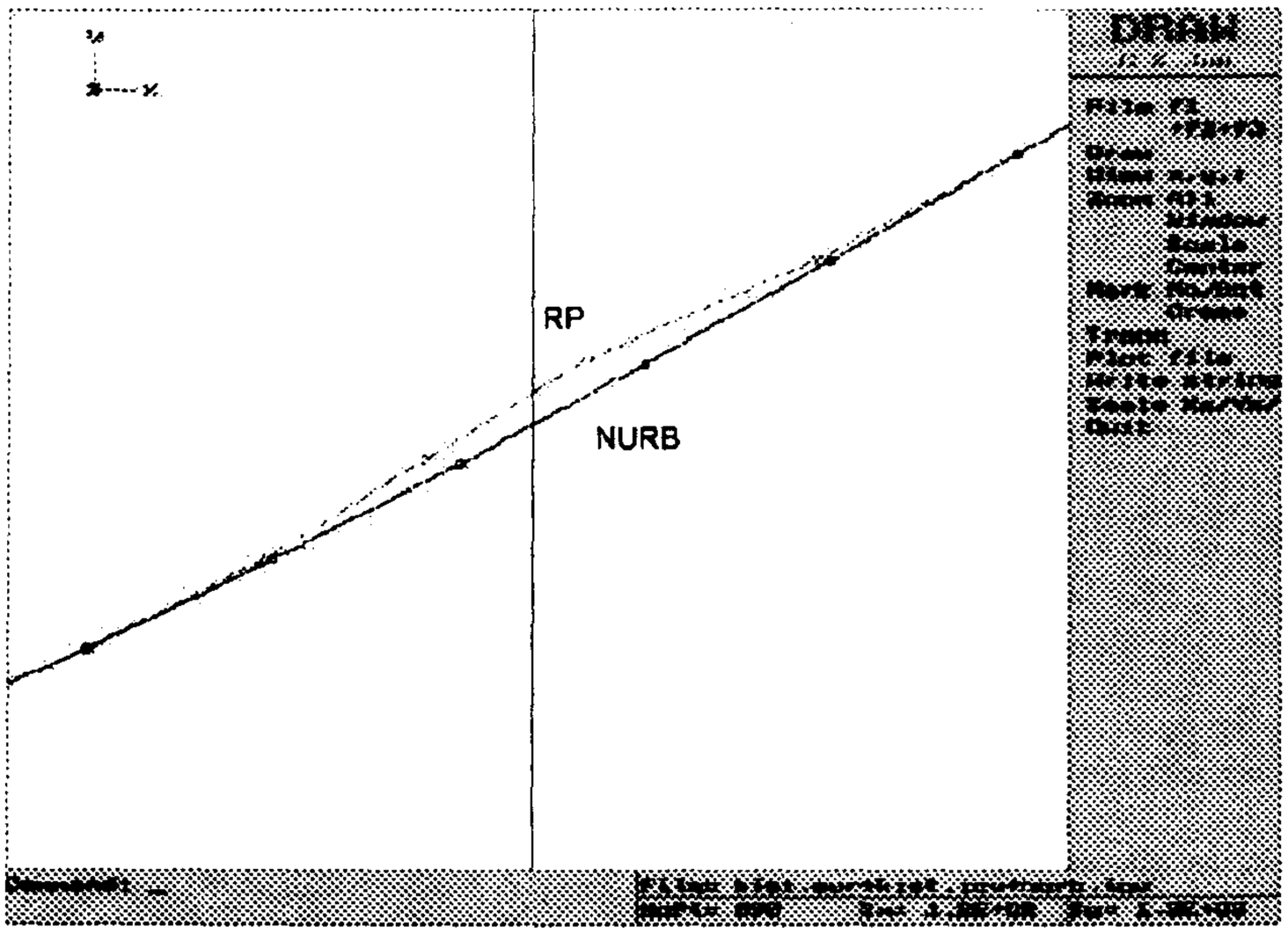
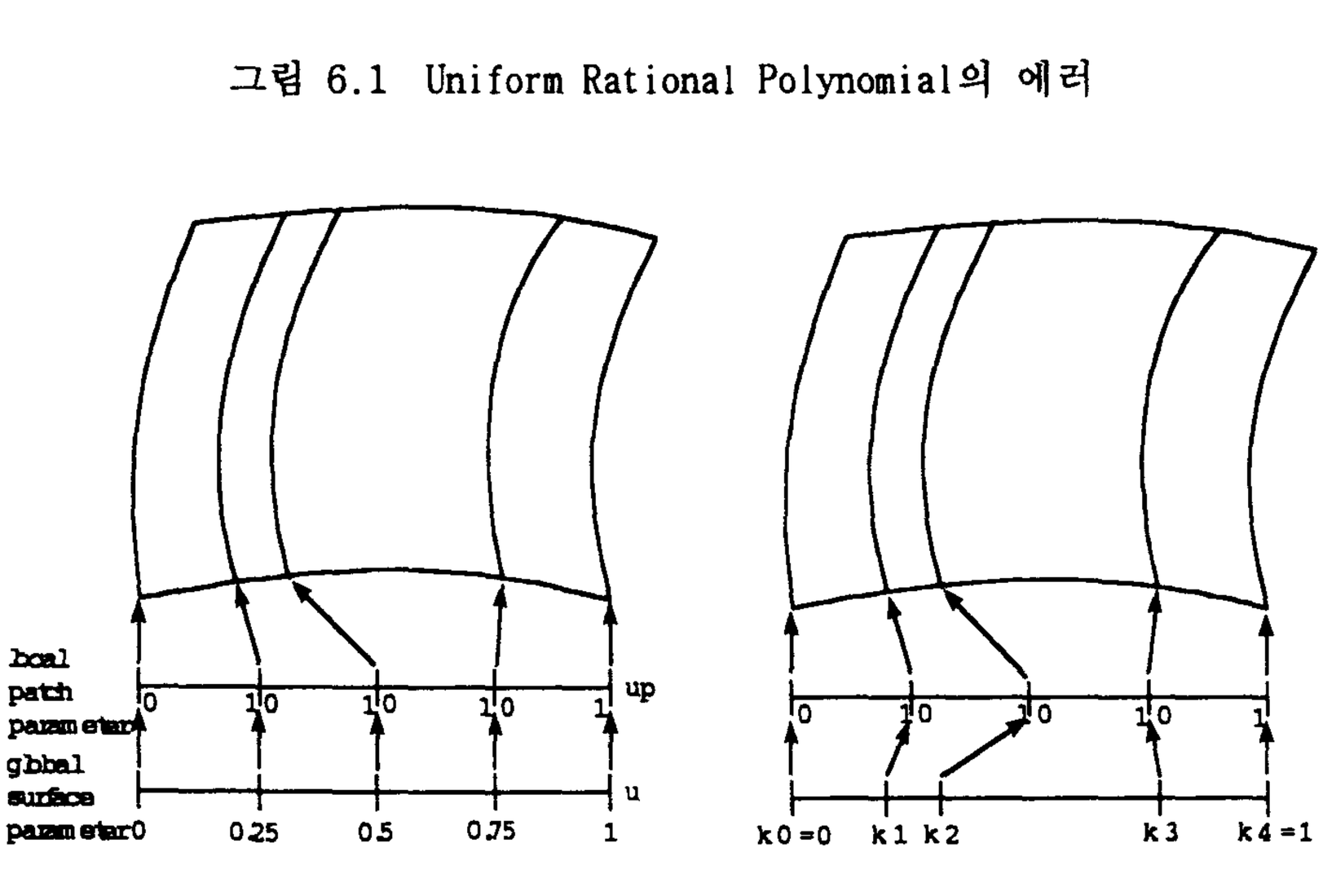
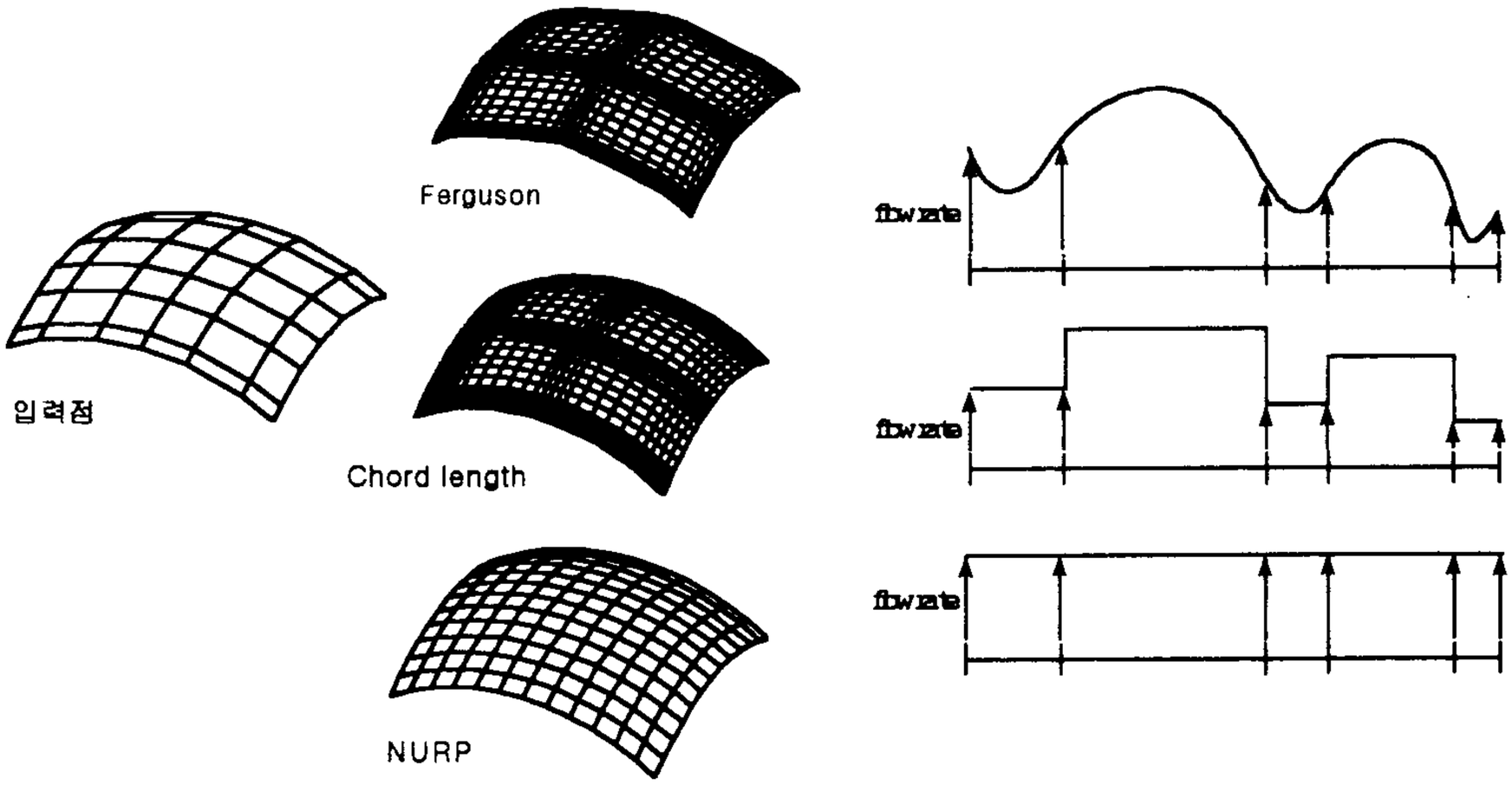


그림 6.1 Uniform Rational Polynomial의 에러



[Uniform Rational Polynomial] [Non-uniform Rational Polynomial]

그림 6.2 Rational Polynomial과 NURP와의 Evaluation 과정의 비교



[곡면] [Flowrate]

그림 6.3 Ferguson/현길이 Spline/NURP의 곡면과 flowrate의 비교

나. NURP 자료 구조

(1) 곡선

Type	변 수	비 고
int	*iRatCv	0 = Non-rational 1 = Rational
int	*iUniformCv	0 = Uniform 1 = Non-uniform
int	*NoCvSeg	Segment 개수
int	**Ndeg	order(degree+1)
double	*CvCoef	Surface Coefficients

double	**KnotCv	knot vector(0 ~ 1)

(2) Edge

Type	변 수	비 고
int	*iRatEdg	0 = Non-rational 1 = Rational
int	*iUniformEdg	0 = Uniform 1 = Non-uniform
int	*iNoEdgSeg	Segment 개수
int	**EdgOdr	order(degree+1)
double	*EdgCoef	Surface Coefficients
double	**KnotEdg	knot vector(0 ~ 1)

(3) 곡면

Type	변 수	비 고
int	*iRatSf	0 = Non-rational 1 = Rational
int	*iUniformSf	0 = Uniform 1 = Non-uniform
int	*NiPat *NjPat	U방향 patch 개수 V방향 patch 개수
int	**NuDeg **NvDeg	U방향 order(degree+1) V방향 order(degree+1)

double	*Sfcoef	Surface Coefficients
double	**U_knotSf	u방향 knot vector(0 ~ 1)
double	**V_knotSf	v방향 knot vector(0 ~ 1)

2. Ferguson 곡선의 NURP 곡선 변환

가. 현길이 스플라인 곡선

n개의 점열 $r_0, r_1, \cdots, r_{n-1}$ 을 지나는 곡률 연속의 현길이 스플라인 곡선으로 보간하면 다음과 같은 Ferguson Basis 함수를 갖는 곡선을 얻을 수 있다.

$$r_i(u) = UCS_i$$

여기서 $U = \begin{bmatrix} 1 & u & u^2 & u^3 \end{bmatrix}$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 3 & -2 & -1 \\ 2 & -2 & 1 & 1 \end{bmatrix}$$

$$S_i = \begin{bmatrix} r_i \\ r_{i+1} \\ w_i t_i \\ t_i \end{bmatrix}$$

$$w_0 = 1$$

$$w_i = \frac{|r_{i+1} - r_i|}{|r_i - r_{i-1}|} \quad : \text{for } i = 0, 1, 2, \cdots, n-1$$

$$r_i(u) = UA_i = UCS$$

$$\therefore A_i = CS_i$$

나. NURP 곡선의 knot vector

점열 $r_0, r_1, \cdots, r_{n-1}$ 을 non-uniform rational polynomial로 보간하기 위해서는 점열의 현길이를 기준으로 knot span을 결정하게 되는데 전체적인 절차는 아래와 같다.

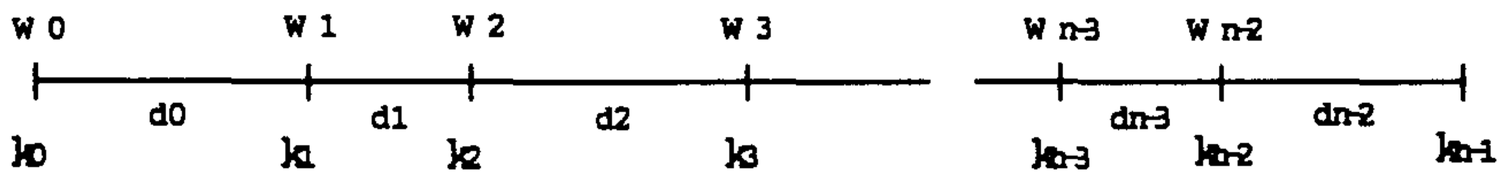


그림 6.4 Chord-Length Ferguson의 weight와 knot vector의 관계

weight : w_i

$$w_i = \frac{d_i}{d_{i-1}}, \quad \text{for } i = 0,1,2,\cdots,n-1$$

knot : k_i

$$k_i = k_{i-1} + \prod_{j=0}^{i-1} w_j \text{ , for } i = 0,1,2,\cdots,n-1$$

$$k_0 = 0 < k_1 < k_2 < \cdots < k_{n-1} = 1$$

3. Ferguson 곡면의 NURP 곡면 변환

가. 현길이 스플라인 곡면

(m×n)개의 점군 $\{r_{ij} : i=0,1,2,\cdots,m-1 ; j=0,1,2,\cdots,n-1\}$을 곡률 연속으로 보간하는 현길이 스플라인 곡면의 데이터는 다음과 같다.

$\{r_{ij} : i=0,1,2,\cdots,m-1 ; j=0,1,2,\cdots,n-1\}$: 점

$\{s_{ij} : i=0,1,2,\cdots,m-1 ; j=0,1,2,\cdots,n-1\}$: u방향 접선 벡터

$\{t_{ij} : i=0,1,2,\cdots,m-1 ; j=0,1,2,\cdots,n-1\}$: v방향 접선 벡터

$\{x_{ij} : i=0,1,2,\cdots,m-1 ; j=0,1,2,\cdots,n-1\}$: twist 벡터

$\{w_{ij} : i=0,1,2,\cdots,m-2\}$: u방향 현길이 비

$\{q_{ij} : j=0,1,2,\cdots,n-1\}$: v방향 현길이 비

이때 곡면 patch의 식은 다음과 같다.

$$r_{ij}(u,v) = UCQ_{ij}C^tV^t$$

여기서 $U=\begin{bmatrix}1 & u & u^2 & u^3\end{bmatrix} : u \in [0,1]$

$V=\begin{bmatrix}1 & v & v^2 & v^3\end{bmatrix} : v \in [0,1]$

$$C=\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\\ -3 & -3 & -2 & -1\\ 2 & -2 & 1 & 1\end{bmatrix}$$

$$Q_{ij}=\begin{bmatrix}r_{i,j} & r_{i,j+1} & q_j t_{i,j} & t_{i,j+1}\\ r_{i+1,j} & r_{i+1,j+1} & q_j t_{i+1,j} & t_{i+1,j+1}\\ w_i s_{i,j} & w_i s_{i,j+1} & w_i q_j x_{ij} & w_i x_{i,j+1}\\ s_{i+1,j} & s_{i+1,j+1} & q_j x_{i+1,j} & x_{i+1,j+1}\end{bmatrix}$$

$$r_{ij}(u,v) = UA_{ij}Vt = UCQ_{ij}C^TV^T$$

$$\therefore A_{ij} = CQ_{ij}C^T$$

나. NURP 곡면의 knot vectors

전체적인 절차는 곡선의 경우와 knot vector가 u, v방향 두 개가 존재한다는 것을 제외하고는 동일하다.

4. NURBS 곡선/곡면/Edge의 NURP 곡선/곡면/Edge 변환

NURBS 곡선을 NURP의 곡선으로의 변환에는 두 가지 방법이 있다. 모든 차수에 대해 적용 가능한 NURBS의 control points를 Bezier control points로 decomposition하는 방법과 3차까지 정의된 NURBS coefficient matrix를 이용하는 방법이 있다.

가. NURBS 곡선의 NURP 곡선 변환

(1) 변환 절차

NURBS 곡선의 control points를 bezier control points로 decomposition 시킨 후 n차 bezier coefficients matrix와 n차 bezier control points를 행렬 곱하면 NURP의 coefficients를 생성할 수 있고, NURBS의 knot vectors를 0에서 1사이의 값으로 만든 다음 NURP의 knot vector에 저장하면 된다.

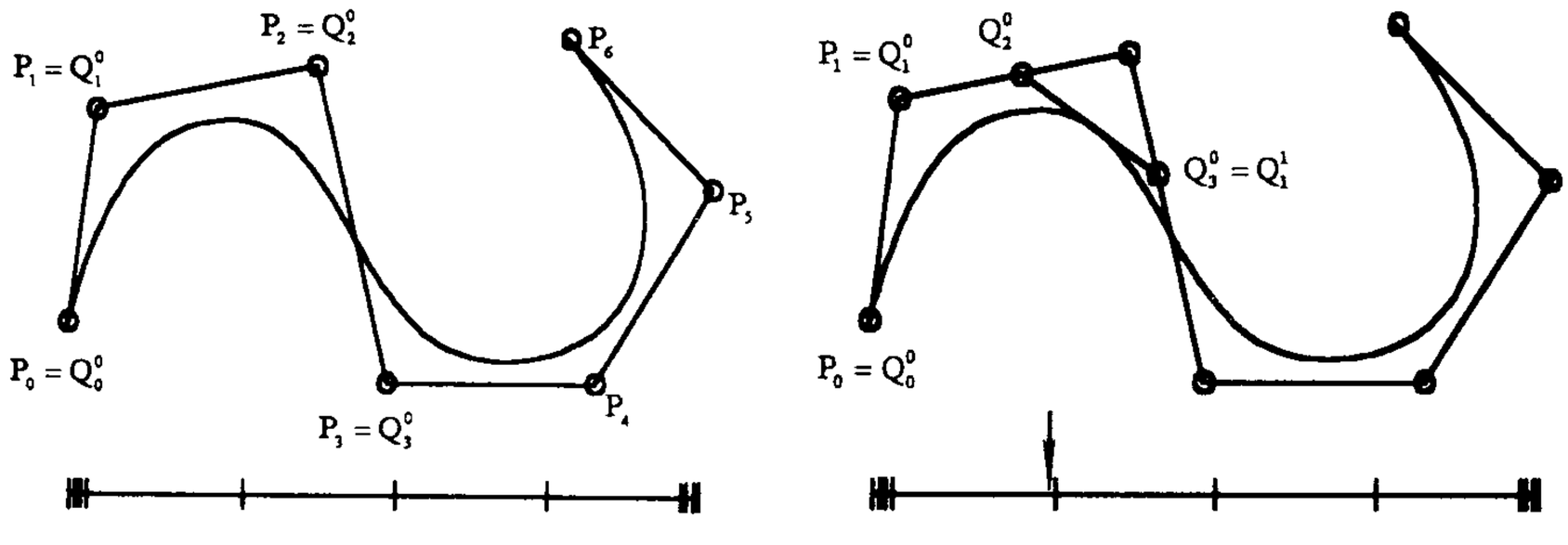
(2) 곡선 Decomposition

일반적인 3차 NURBS 곡선일 경우에 첫 번째와 마지막에서는 order(degree+1)개 만큼 knot이 중복되어있고, 그 나머지에서는 하나이거나 order보다는 적은 개수의 knot이 중복되어진다. 여기서 만약 첫 번째와 마지막

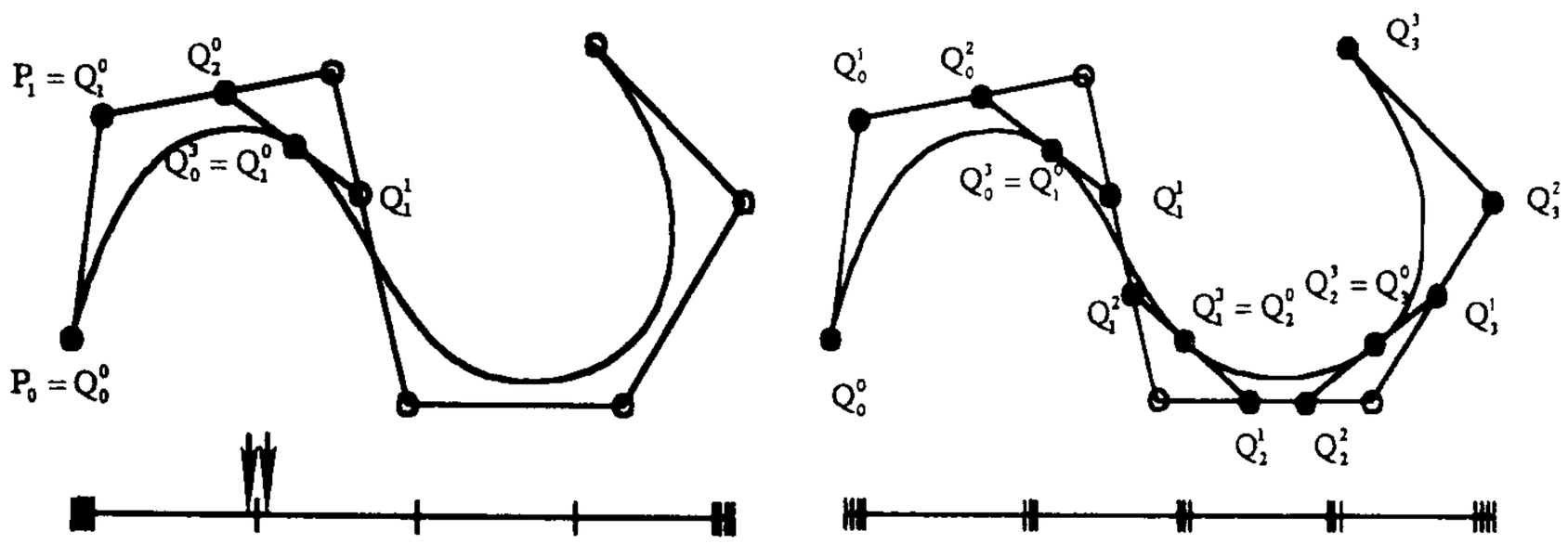
knot를 제외한 knot에 degree만큼의 중복을 시키면 Bezier control points를 구할 수 있다. 이와 같은 knot의 성질을 이용하면 각각의 segment를 분해할 수 있는데 이 과정을 decomposition이라고 한다. 여기서 bezier control points의 계산은 다음과 같이 정의된다.

k : knot index, p : degree

$$Q_i^w = a_i P_i^w + (1 - a_i) P_{i-1}^w \quad \text{where } a_i = \begin{cases} 1 & i \le k - p \\ \dfrac{u - u_1}{u_{i+p} - u_i} & k - p + 1 \le i \le k \\ 0 & i \ge k + 1 \end{cases}$$



(a) 3차 NURBS Curve　　(b) First knot insertion



(c) Second knot insertion　　(d) 3차 Bezier Curve

그림 6.5　곡선 Decomposition의 절차

(3) knot의 변환

NURP의 knot는 중복 knot를 제외시킨 patch+1만큼만을 가진다. 따라서 NURBS의 knot를 그대로 적용하지는 못하고 중복이 없는 knot만을 찾아서 knot의 크기가 0~1사이의 값을 가지도록 각각의 knot를 전체 knot의 크기로 나누어준다.

order : degree + 1

U : NURBS knot vectors

m : NURBS knot의 개수

mult : NURBS multiple knot의 개수

knot : NURP knot(0~1사이의 값을 가진다).

n : NURP knot의 개수

```
n = no_knot - 2 * order + 1 - mult;

knot[0]     = 0.0

knot[1]     = U[order] / U[no_knot - order] - U[order - 1];


knot[n - 1] = U[m - order - 1] / U[no_knot - order] - U[order - 1];

knot[n]     = 1.0
```

나. NURBS 곡면의 NURP 곡면 변환

전체적인 절차는 곡선의 경우와 knot vector가 u, v방향 두 개가 존재한다는

것을 제외하고는 동일하다.

다. NURBS Edge의 NURP Edge 변환

z값이 없다는 것을 제외하고는 곡선의 변환 과정과 동일하다

제 2 절 곡면과 곡면의 교선

1. Surface/surface Intersection

곡면과 곡면간의 교선을 구하는 문제는 Geometric processing에서 가장 중요한 문제의 하나이다. 이의 대표적인 용도를 나열하면 다음과 같다.

- blend surface의 형성 (이 경우는 offset surface간의 교선이 필요하다.)
- 근래 차세대 곡면 모델러로서 이야기되고 있는 Unified Solid modeler에서 필요한 trimmed surface의 형성
- NC 가공을 위한 tool path의 생성 (Cartesian machining)
- 설계된 곡면의 검증 (곡면의 단면을 얻거나 곡면간의 간섭 검사)

Intersection routine은 CAD/CAM시스템에 있어서 가장 기본적이고도 많이 이용되므로 전체 System의 성능에 많은 영향을 미친다. 따라서 Intersection routine은 다음과 같은 요구 조건을 만족해야 한다.

- accuracy
- robustness
- efficiency
- memory Storage의 절약

일반적으로 두 개의 평활한 곡면간의 Intersection의 결과는 다음과 같은 경우로 나누어진다.

(i) 전혀 만나지 않는 경우

(ii) 한 개 또는 다수의 접점

(iii) 한 개 또는 다수의 교선

(iv) 한 개 또는 다수의 접선

(v) 한 개 도는 다수의 접면

(vi) (ii), (iii), (iv), (v)의 조합

두 개의 곡면의 intersection이 (i) 또는 (iii)의 경우에 있는 것이 '일반적'이며 이렇지 않을 경우 두 개의 자유곡면 사이의 intersection을 구하는 것은 훨씬 더 어렵고 시간이 많이 소요되는 algorithm이 필요하게 된다. 본 연구는 '일반적'인 경우의 intersection을 구하는 문제만을 다루고 있다.

Surface/Surface Intersection(이하 SSI로 부름) 방법은 4개의 범주 즉, Algebraic method, lattce evaluation method, subdivision method, 그리고 tracing method등으로 구분할 수 있는데, 곡면 모델러에서 흔히 쓰이는 Parametric surface에 대해서는 subdivision method와 tracing method가 가장 많이 이용된다.

Subdivision method는 Bezier나 B-spline surface와 같이 convex-hull Property가 있는 곡면에 대하여 robustness가 큰 방법을 알려져 있다[6-8][6-9]. 그 외의 곡면에 대해서는 별도의 조작이 필요하며 이때는 robustness와 efficiency가 많이 떨어지게 된다.

Tracing method에서는 이미 구해진 교선 상의 한 점에서 곡면의 미분 기하적 성질에 의하여 제어되는 어떤 방향으로 한 'step'만큼 이동하여 교선 상의 다른 한 점을 찾게되는데 efficiency에 큰 특징이 있으며 곡면의 수학적 표현 형태에 크게 영향을 받지 않는다. 그러나 이 방법은 곡면이 Smooth하지 못하여 곡면의 미분 기하적 성질의 변화가 급격하면 tracing에 성공하지 못하는 경우

가 있고, 교선이 여러 개 존재할 때 각 교선에 대하여 각각 하나 이상의 출발점을 발견하지 못하게 된다. 이런 robustness에 관한 결점을 보완하기 위하여 많은 연구가 진행되었으며 Barnhill 등의 연구가 가장 대표적이라 할 수 있다. 본 연구에서는 Barnhill의 SSI Algorithm을 이용하였으며 이의 특징은 다음과 같다.

(1) 출발점을 발견하는 과정을 도입함으로써 robustness를 크게 향상시켰으며, SSI의 용도에 따라 tolerance를 조정함으로써 efficiency와 robustness를 모두 만족시키는 최적의 절충이 가능하다.

(2) 사각 매개변수 영역에서 정의된 곡면이면 곡면의 수학적 표현 방식에 전혀 영향받지 않는 일반성이 있다. 특히 blending surface 형성 시 쓰이는 offset surface간의 intersection에도 전혀 수정 없이 이용 가능하다.

(3) 본 Algorithm은 C^1 Continuity를 만족하는 surface에 대하여 개발되었으나 implementation 방식에 따라서 C^0 surface에 대해서도 만족스러운 결과를 얻을 수 있다.

나. SSI Algorithm의 개요

본 연구에서 이용한 SSI알고리즘은 다음과 같은 4단계로 이루어져 있다.

(1) Mesh Generation

각 surface domain상에서 직사각형 격자를 형성한다. 이 때 첫 번째 곡면은 격자점을 직선으로 연결하는 network로 근사 되며, 두 번째 곡면은 하나의 사각 격자를 두 개의 삼각형으로 분할하는 triangular polyhedron면으로 근사 된다. mesh의 크기는 efficiency와 robustness에 많은 영향을 미친다. 즉

mesh가 많으면 robustness는 증가되나 efficiency가 많이 떨어진다.

(2) Detection

첫 번째 곡면의 network의 모든 선분과 두 번째 곡면의 polyhedron의 모든 삼각형면 사이의 intersection point를 초기점으로 첫 번째 곡면의 등매개변수 곡선과 두 번째 곡면의 교점을 구한다. 이 점들은 다음 단계인 Tracing의 출발점이 된다.

Detection 된 point를 저장하는 Data Structure (IIP-list)는 tracing과정에서의 efficiency와 memory size에 많은 영향을 미치게 된다.

(3) Tracing

Detection 과정에서 구한 초기점(IIP-List에 저장)을 출발점으로 하여 Newton-Raphson방식에 의하여 교선을 추적한다. 교선의 추적 방향은 출발점의 앞과 뒤 양쪽으로 추정하며 한쪽 방향의 추적은 다음과 같은 조건이 만족되면 추적을 마친다.

(i) tracing방향이 존재하지 않는 경우

(ii) tracing방향의 변화가 급격한 경우

(iii) 두 곡면 중 어느 한 곡면의 domain으로부터 나가는 경우

(iv) tracing이 출발점으로 되돌아오는 경우 (교선이 폐곡선인 경우)

이 중에서 (iv)를 제외하고는 반대 방향으로도 추적한다. 추적과정에서는 한 step이 진행될 때마다 그 선분에 가까이 있는 초기점을 IIP-list로부터 제거하고, 한 교선의 추적이 끝나면 IIP-list에서 다른 점을 초기점으로 하여 다른 교선을 찾는다. 본 과정은 IIP-list가 완전히 비게 되면 끝난다.

(4) Optimization and Curve fitting

Tracing과정에서 얻은 점열을 주어진 공차에 맞도록 솎아내어 Data의 개수를

줄이고 교선의 용도에 따라 이를 fitting하여 곡선의 식을 얻는다. trimmed surface의 경계를 얻고자 하는 경우는 domain상에서 보간 하여 'edge'를 구한다.

다. 프로그램 수행 예

그림 6.6은 두 개의 자유곡면간의 교선을 구한 것이고 그림 6.7는 두 개의 원통 면간의 교선을 구한 것이다.

그림 6.7의 접점 근방에서는 교선의 방향이 급격히 변화되므로(격임) 접점에서 tracing을 마치게 된다.

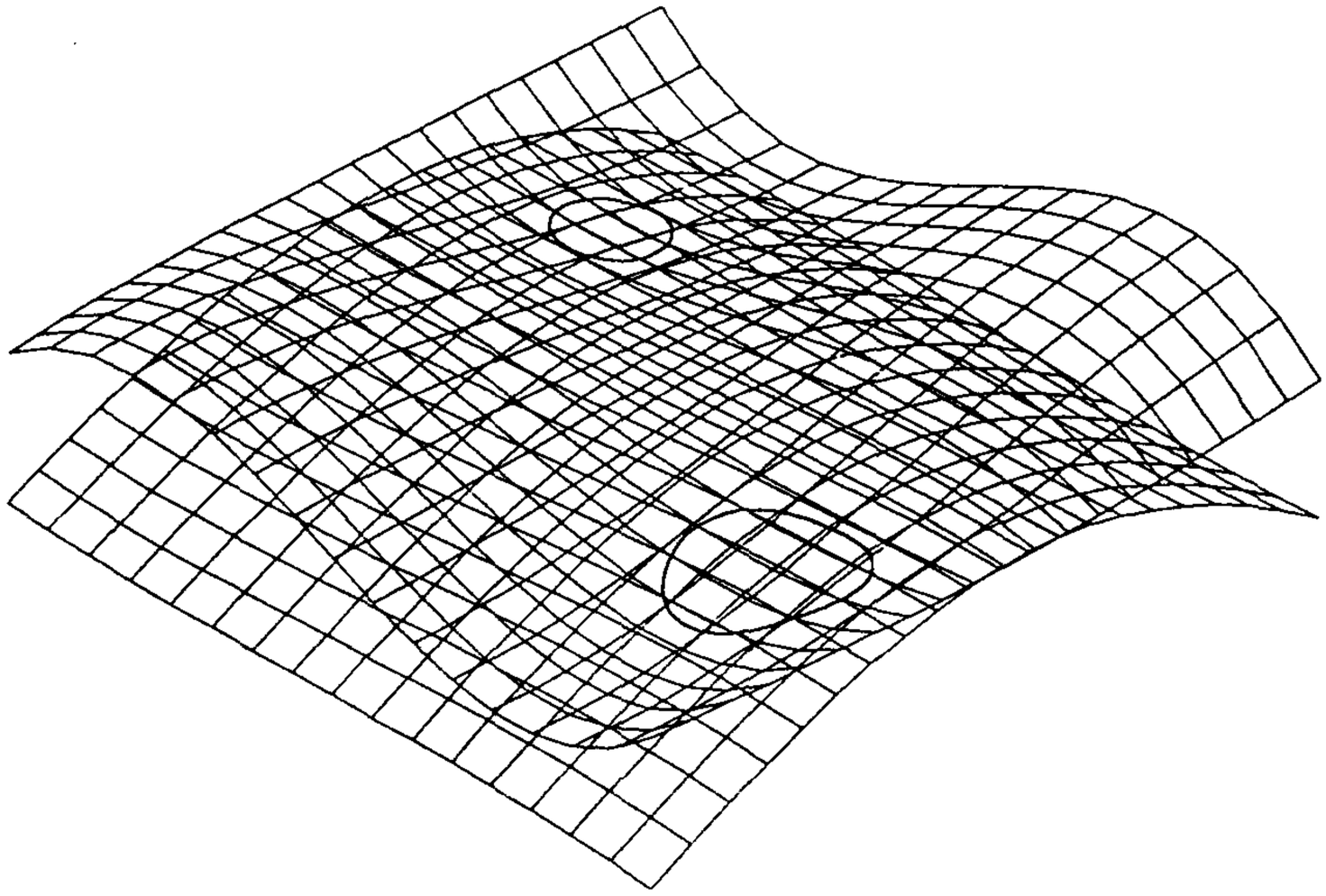


그림 6.6 두 자유곡면 간의 교선

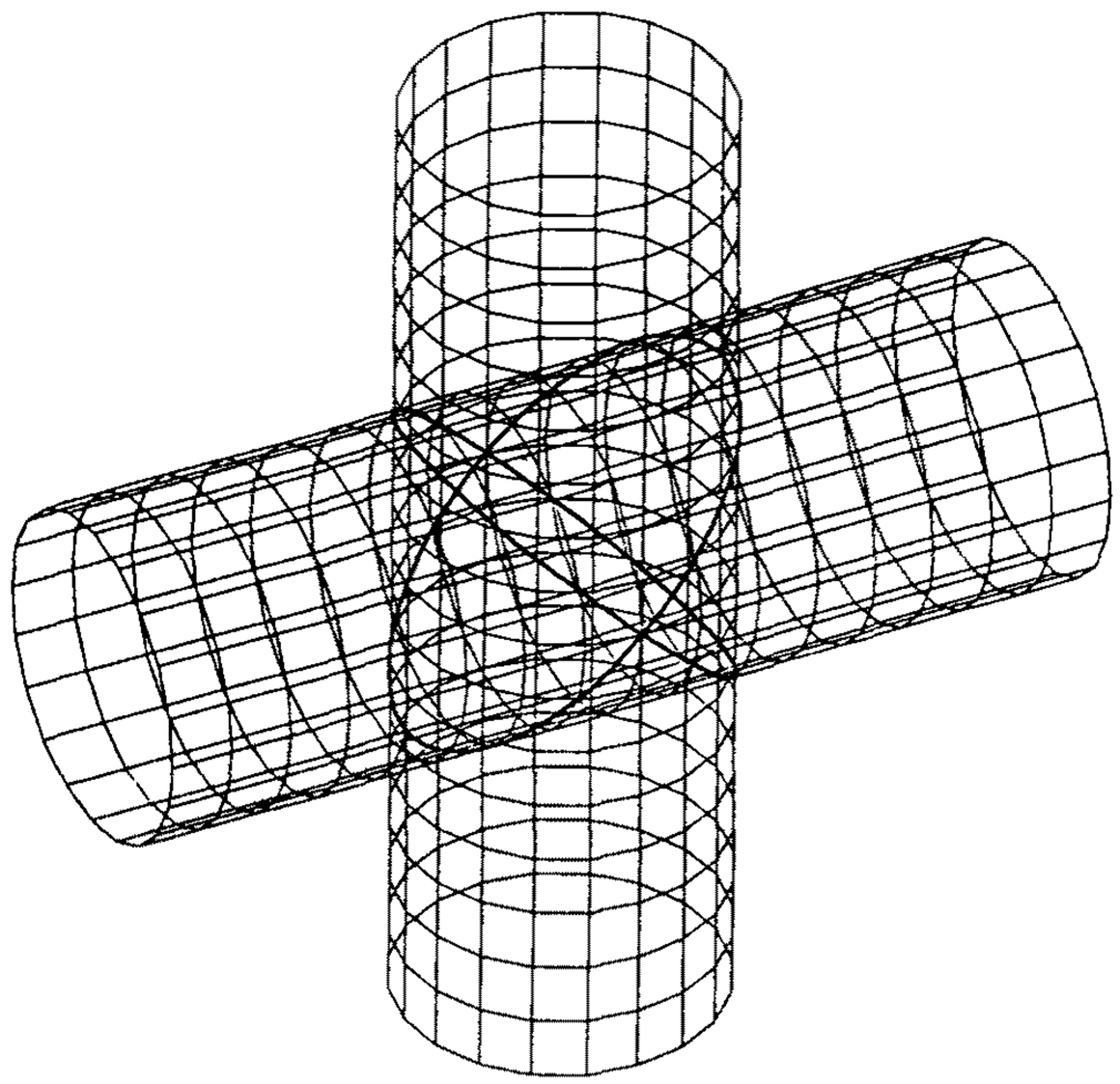


그림 6.7 두 원통면 간의 교선

2. Curve Projection

Curve Projection에서는 공간상의 곡선을 곡면 위로 투영하였을 때 곡선 위에 놓이는 Cartesian space와 domain space상에서 얻는다. 이 결과는 NC가공의 영역을 지정하거나 trimmed surface의 경계로 얻는데 이용될 수 있다.

가. Curve Projection Algorithm의 개요

Curve Projection의 과정을 두 단계로 나누어진다.

(1) Making Ruled surface

주어진 곡선을 Projection방향으로 sweeping되는 ruled surface를 형성한다.

(2) Surface/surface Intersection

앞에서 형성한 ruled surface와 주어진 surface와의 intersection curve를 구한다.

나. 적용 예

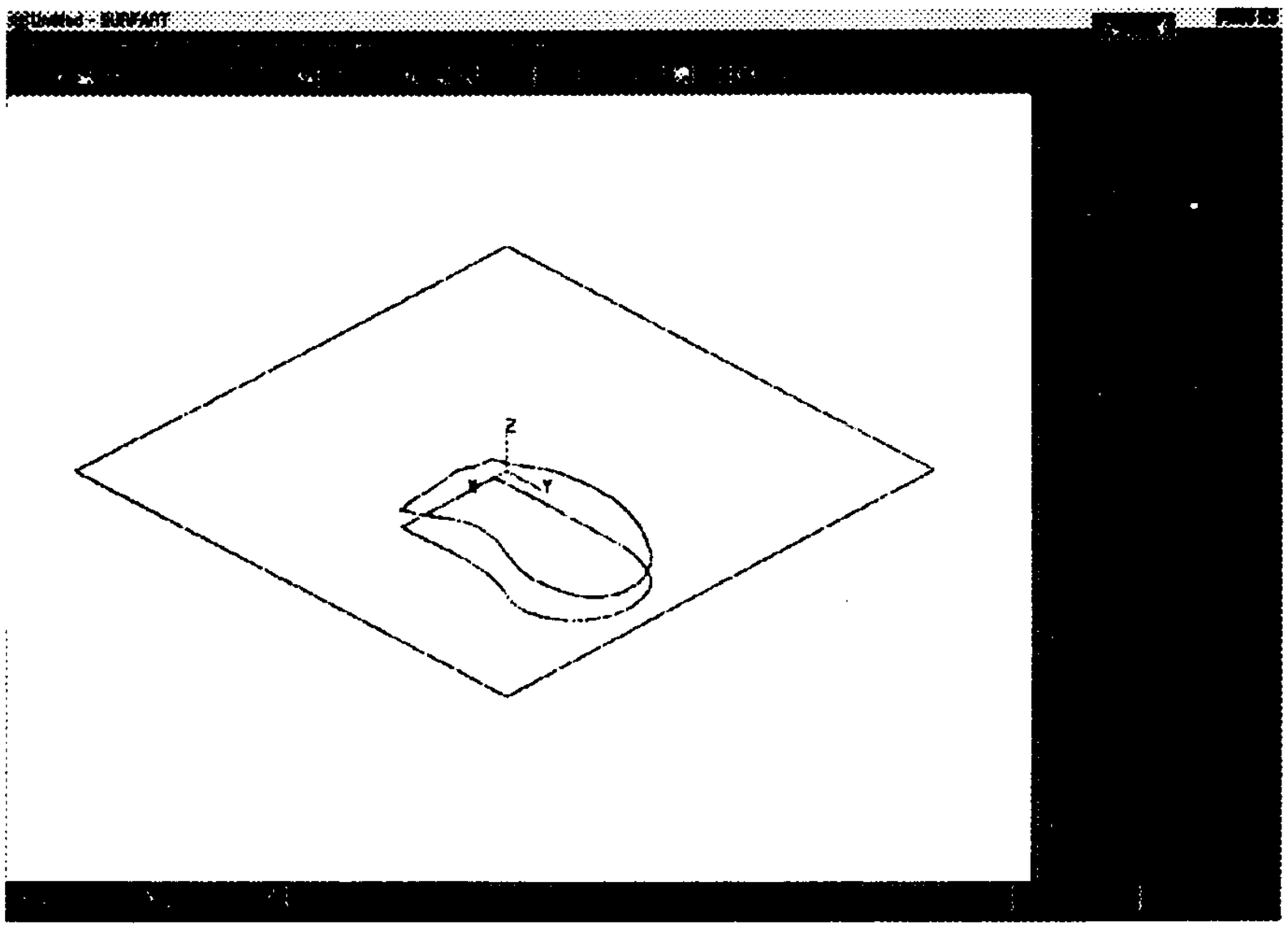


그림 6.8 곡선의 projection

3. Face/surface Intersection

Face와 surface 간의 교선을 구하는 알고리즘은 surface/surface intersection(SSI)을 수행한 후 결과로 나온 교선 중 face에 속하지 않은 부분을 트리밍 하는 과정이다. 그 구체적인 절차는 다음과 같다.

1) Input : face, surface1

2) surface2 ← face의 supporting surface

3) Intersection curve list ← SSI(surface1, surface2)

4) face에 의한 Intersection curve의 trimming

4-1) Intersection curve와 face boundary의 교점을 구함

① face의 boundary(domain curve)를 line segment로 근사

② Initial intersection point를 구함

← line/line intersection(face의 boundary, intersection curve)

③ Exact intersection point를 구함

← surface/edge intersection(surface1, face boundary)

4-2) Intersection curve 중 face 외부를 trimming

5) Output : trimmed intersection curve

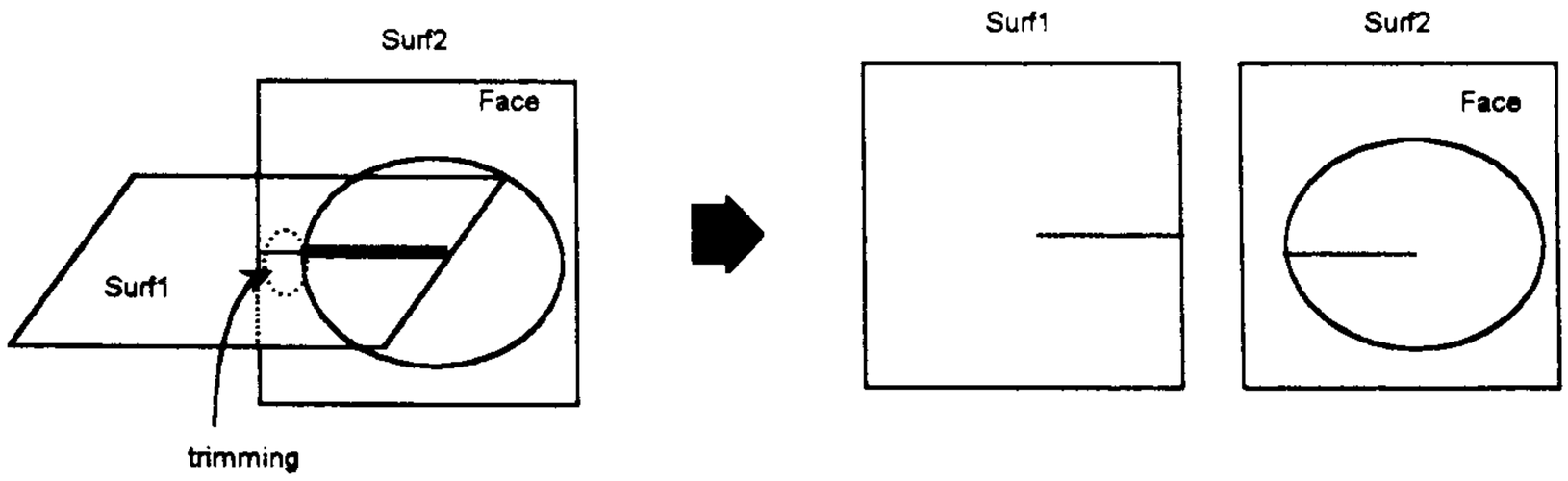


그림 6.9 Face/Surface intersection

4. Face/face Intersection

Face와 face간의 intersection은 각 face에 대한 base surface를 가지고 SSI를 수행한 후 각 face별로 intersection curve를 트리밍하는 과정을 통해서 얻을 수 있다. 여기서 intersection curve를 트리밍하는 방식은 앞의 face/surface intersection에서와 동일하다.

1) Input : face1, face2

2) surface1 ← face1의 supporting surface

   surface2 ← face2의 supporting surface

3) Intersection curve list ← SSI(surface1, surface2)

4) Intersection curve trimming

   4-1) face1에 의한 intersection curve의 trimming

   4-2) face2에 의한 intersection curve의 trimming

5) Output : trimmed intersection curve

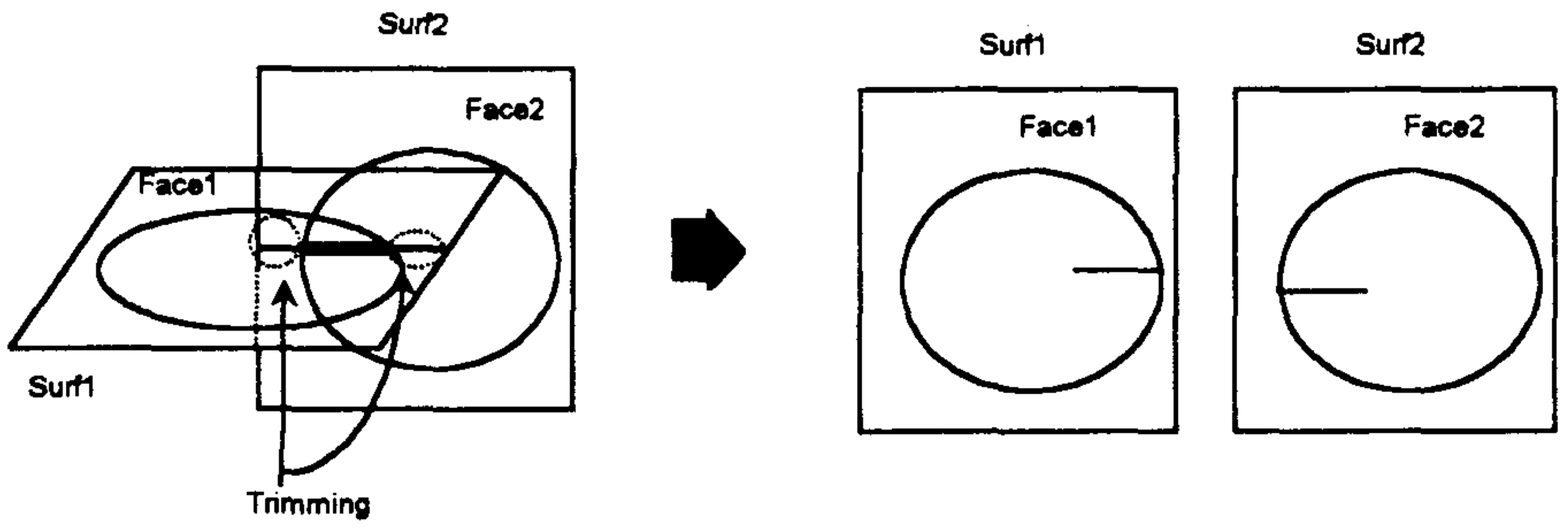


그림 6.10 Face/Face intersection

제 3 절 블렌드 곡면의 생성

1. 고정 반경 모서리 블렌드 곡면(Constant Radius Edge Blending)의 생성

Constant Radius Edge(CRE) Blending이란 일정한 반경의 구를 두 곡면에 동시에 접하도록 굴렸을 때 형성되는 envelope surface를 구하는 것이다(이를 'rolling-ball' method라 한다.). 수학적으로 보면 구의 중심이 지나는 곡선은 구의 반경만큼 offset된 두 곡면의 교선에 해당되며, 구가 곡면에 접하며 지나는 boundary curve는 offset곡면의 교선을 각 곡면의 법선 방향으로 곡면에 투영한 곡선이다. 본 연구는 주상윤[6-6] 의한 연구를 바탕으로 본 모델러에 맞도록 수정하여 구현하였다. 주상윤의 연구는 VC^1 continuity를 위하여 blend surface의 표현 방식이 base surface (Blending되는 곡면)에 의존되므로 기존 modeler의 surface 표현 방식과 상이하다. 본 연구에서는 blend surface의 표현 방식을 base surface와 같은 형태를 갖도록 하였다. 이렇게 하면 C^0 continuity를 만족시키지 못하게 되나 주어진 공차를 만족하므로 실용적인 측면에서는 문제가 없다. 또한 효율적 응용을 위하여 blend surface의 boundary curve에 해당되는 'edge'(base surface 상의 곡선)을 제공하고 있다. 본 연구는 특히 efficiency의 향상에 역점을 두었다.

가. CRE Blending Algorithm의 개요

Constant Radius Edge Blending은 다음과 같은 2단계로 이루어진다.

(1) 블렌드 곡면의 경계 곡선 생성

두 base surface의 offset surface의 교선을 구한다. 교선의 domain값에 해

당되는 base surface 위의 곡선이 구가 굴러가면서 base surface와 접하는 곡선이며, blending surface의 boundary curve가 된다. 이 단계는 이미 설명한 SSI routine이 수정 없이 이용된다.

(2) 블렌드 곡면 보간

구하고자 하는 blend surface의 단면 (boundary curve에 수직한)이 주어진 반경을 갖는 원호가 되도록 surface를 형성한다. 본 연구에서는 chord-length surface fitting절차가 이용되었다[6-13].

다. 프로그램 수행 예

그림 6.11은 parametric equation으로 표현된 두 원통면 사이에 CRE blend surface를 형성한 것을 보여주고 있으며, 그림 6.12는 접하는 두 원통면간의 CRE blending을 보여주고 있는데 두 원통 면의 접점에서 blend surface는 degeneration 된다(블렌드 곡면을 생성하는 함수는 부록 E를 참조).

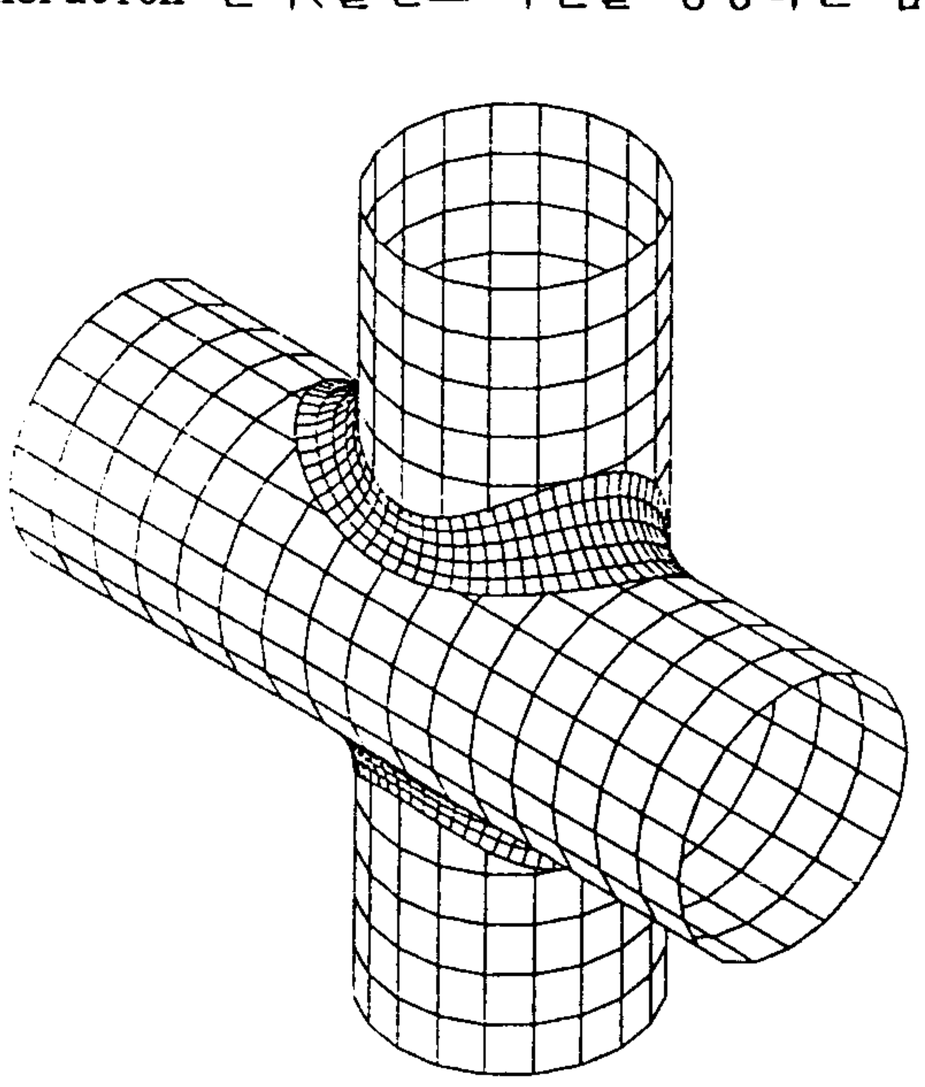


그림 6.11 두 원통면 간의 CRE Blending

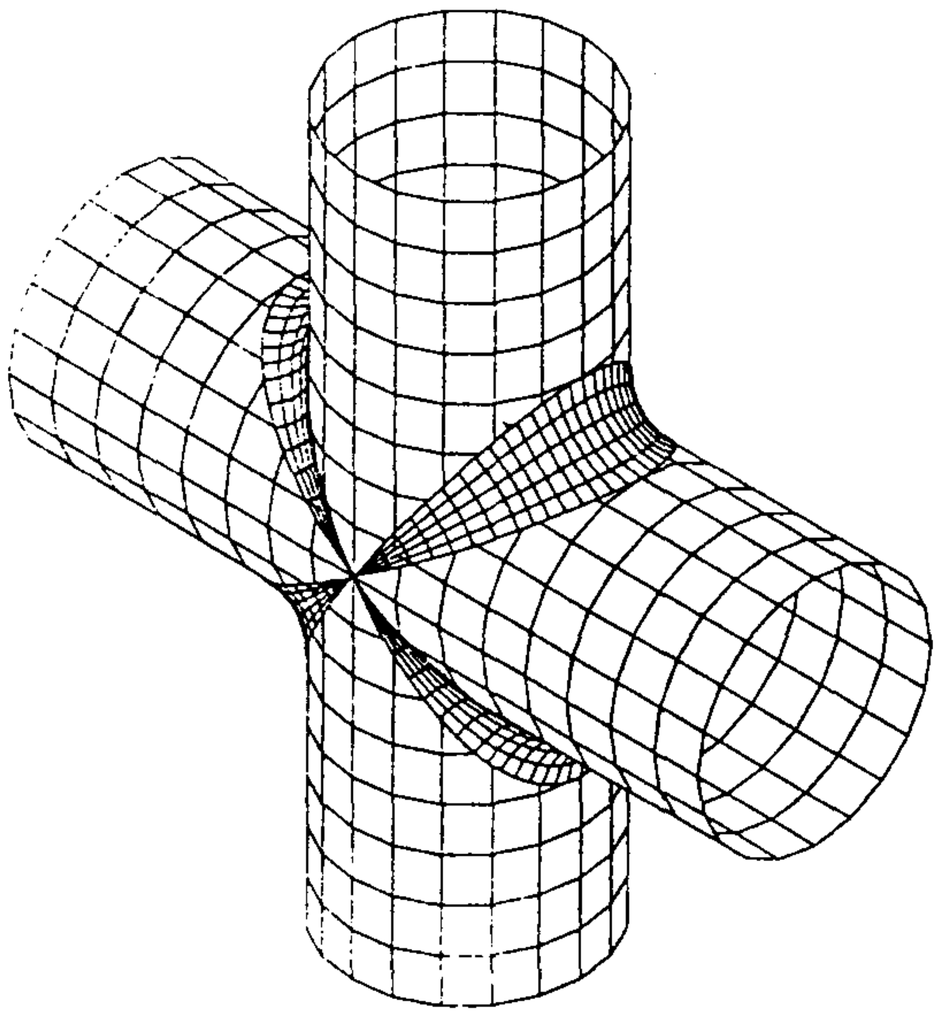


그림 6.12 접하는 두 원통면 간의 CRE Blending

2. 가변 반경 모서리 블렌드 곡면(Variable Radius Edge Blending)의 생성

Variable Radius Edge(VRE) Blending은 'rolling ball'이 두 surface에 동시에 접하도록 굴러가면서 그 반경이 변하는 경우이다. 기본적으로는 CRE blending과 비슷하나 변화하는 반경의 입력방법 및 반경 변화에 따른 교선의 수정을 추가하여야 한다.

가. VRE blending Algorithm의 개요

VRE blending에 대해서는 4단계로 나누어 설명하고자 한다.

(1) Radius-Law

Rolling-ball의 반경 변화는 굴러가는 길이에 대한 함수로 주어져야 하는데 본 연구에서는 입력 반경과 끝 반경을 입력 받아 굴러가는 길이를 매개변수

로 놓았을 때 반경 변화가 직선이 되게 하였다.

(2) 블렌드 곡면의 초기 경계곡선

Radius Law의 평균값으로 CRE와 같은 blend surface의 경계곡선을 구한다.

(3) 블렌드 곡면의 경계곡선 조정

(2)의 단계에서 얻어진 곡선을 초기점으로 하여 Radius Law에 맞도록 새로운 경계곡선을 구한다. 이때는 (2)단계에서 얻어진 offset surface간의 교선의 길이를 기초로 parameterization하여 이 parameter에 맞는 반경 Ri를 얻고, 그 교선에 수직한 평면과 두 base 곡면의 offset곡면 (offset 량 = Ri)과의 교점을 search하여 원하는 경계곡선을 구한다.

(4) 블렌드 곡면의 보간

CRE의 경우와 같다.

나. 프로그램 수행 예

그림 6.13은 평면과 구면 사이에서 radius가 곡선 형태를 따라 변하는 VRE blending을 보여 주고 있다(블렌드 곡면을 생성하는 함수는 부록 E를 참조)

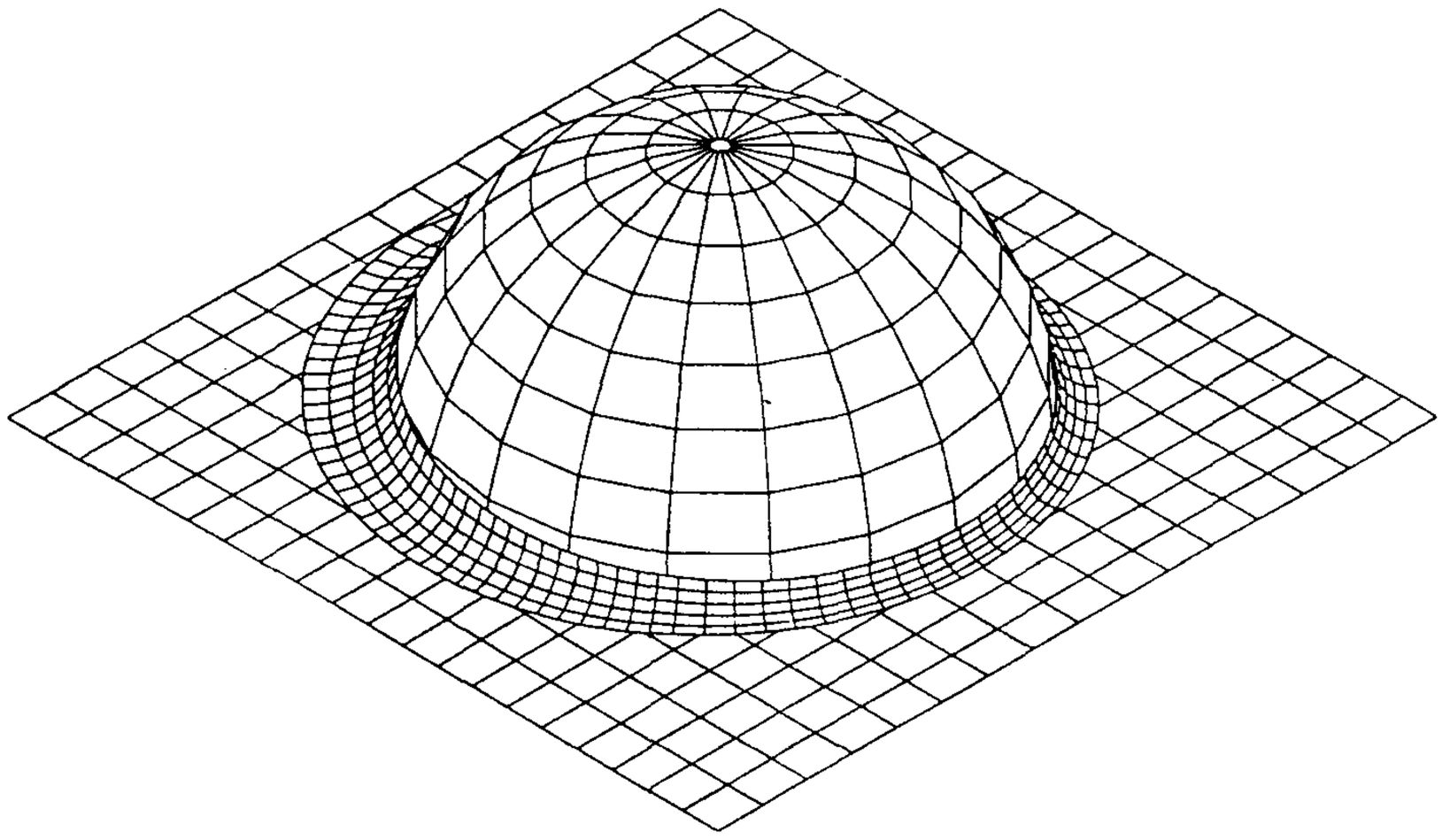


그림 6.13 VRE Blending

3. 코너 블렌드 곡면(Corner Blending)의 생성

Corner bending이란 rolling-ball을 곡면 사이에 굴렸을 때 얻어지는 envelope surface를 구하는 것이며, 이 과정은 CRE blending에 세 곡면의 corner부분의 처리가 추가된다.

가. Corner blending Algorithm의 개요

이 과정은 3단계로 나누어진다.

(1) Generation of blend surface boundaries

2개의 곡면을 한 쌍으로 하고 세 쌍의 곡면 조항에 대하여 CRE blend surface의 경계곡선을 구한다.

(2) Corner finding

각 곡면의 domain상에서 다른 두 곡면과의 교선과의 교점을 구한다. 이 교점들은 corner surface의 vertex가 된다.

(3) surface blending

두 곡면간에는 edge blend surface를 형성하고 세 곡면간의 corner 부위에는 degenerate surface(삼각형 형상의 사각 patch를 형성한다.

나. 프로그램 수행 예

그림 6.14는 한 평면과 원면, 구면에 대한 corner blending을 보여주고 있다(블렌드 곡면을 생성하는 함수는 부록 E를 참조).

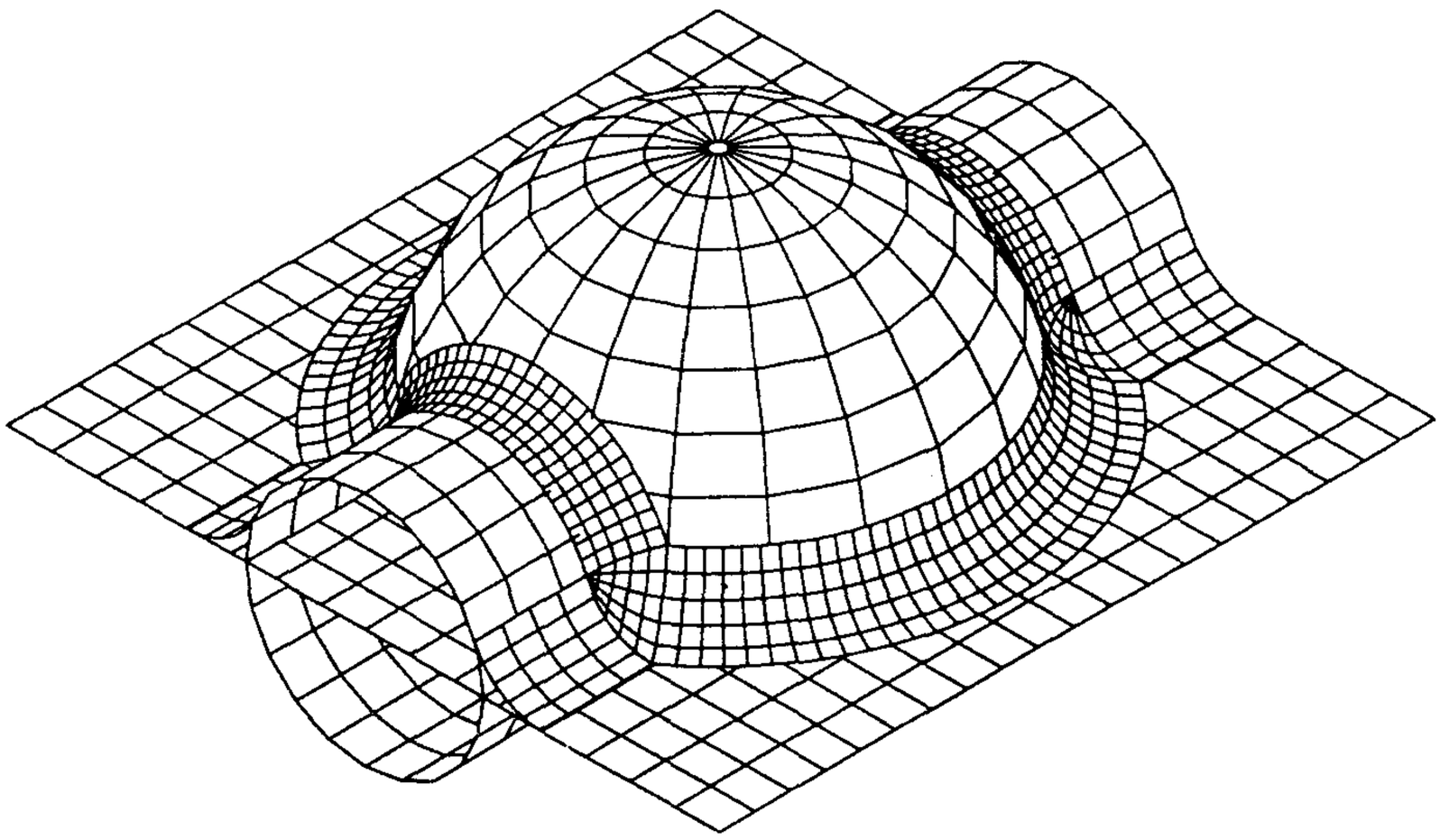


그림 6.14 Corner Blending

4. 연장된 Edge Blend 곡면의 형성

Edge blending을 수행한 후에는 생성된 블렌드 곡면을 가지고 base 곡면을 트리밍한다. 하지만 만약 두 base 곡면 크기가 다르면 생성된 블렌드

곡면의 boundary가 base 곡면의 boundary에 닿지 않는 경우가 발생하게 되고, 그렇게 되면 블렌드 곡면을 가지고서 base 곡면을 트리밍할 수가 없다. 따라서 이미 생성된 블렌드 곡면이 base 곡면의 boundary에 닿도록 블렌드 곡면을 연장하는 과정이 필요하다. 이를 위해서는 먼저 블렌드 곡면이 boundary에 닿지 않는 base 곡면이 있으면 다른 한 곡면(boundary에 도달하는 곡면)의 끝에 ruled surface를 만들어서 추가적인 블렌딩을 수행한 후 base 곡면의 boundary에 맞추어 트리밍하면 된다. 이때의 트리밍은 uv-domain상에서 직선으로 이루어진다. 이러한 과정을 정리하면 다음과 같다.

1) Input : surface1, surface2

2) Intersection curve list ← offset surface intersection(surface1, surface2)

3) FOR (각 intersection curve) {

3-1) IF (intersection curve가 open) goto 3-2

ELSE goto 3

3-2) FOR (intersection curve의 시작/끝) {

IF (intersection curve의 시작(끝)이

surface1의 boundary에는 도달하고

surface2의 boundary에는 도달하지 않으면) {

surface1의 boundary에 ruled surface를 만듦

offset surface intersection(ruled surface, surface2)

} ELSE IF (intersection curve의 시작(끝)이

surface2의 boundary에는 도달하고

surface1의 boundary에는 도달하지 않으면) {

surface2의 boundary에 ruled surface를 만듦

offset surface intersection(surface1, ruled surface)

}

새로 구한 intersection curve list에서 surface boundary에 도달하는 하나의 point만을 원래의 intersection curve list에 추가

}

}

4) blend surface ← intersection curve list를 가지고 blending

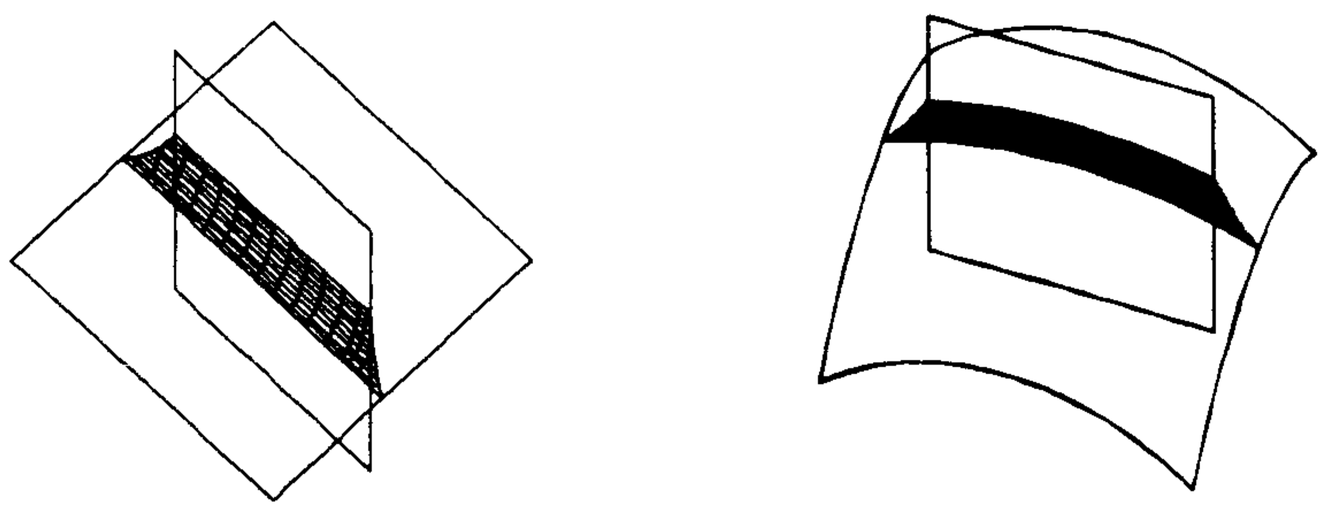
5) 연장된 blend surface를 trimming

5-1) Intersection curve list에서 각 base surface의 boundary에 도달하는 point들을 찾음

5-2) blend surface의 boundary 중 두 base surface를 잇는 boundary는 5-1에서 구한 point들을 uv-domain상에서 직선으로 연결함으로써 연장된 blend surface를 trimming

6) blend surface에 의한 base surface의 trimming

7) Output : 연장된 blend surface, trimmed base surfaces



(a) (b)

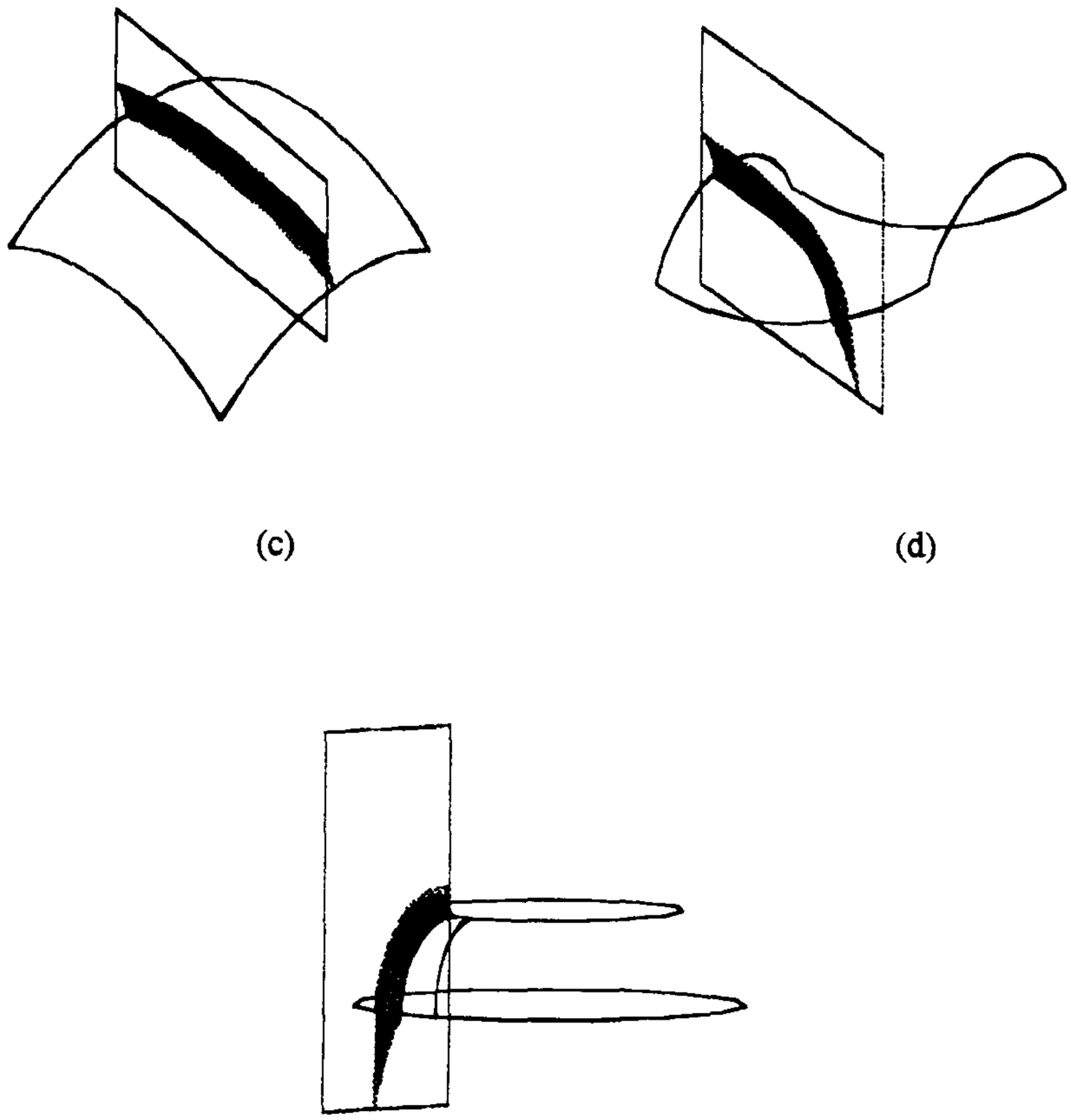


그림 6.15 확장된 블렌드 곡면의 예

제 4 절 이동곡면의 생성

1. 이동 곡면(sweep surface)의 정의

이동곡면(sweep surface)이란 2차원 단면곡선이 어떤 규칙을 갖고 안내곡선(또는 경계곡선)을 따라 이동할 때 생기는 궤적으로 정의된다. 이동곡면은 도면에 표기된 2차원 곡선 정보로부터 3차원 곡면형상을 만드는데 널리 쓰이고 있다.

많은 상업용 CAD/CAM 시스템이 이동곡면을 지원하고 있으나 시스템 내부에서의 처리 방법은 잘 공개하지 않고 있다[6-6][6-8][6-13]. 이동곡면에 관한 연구는 Choi와 Lee[6-7][6-10]에 의해 연구된 바 있다. 이들의 연구는 도면에 나타난 곡면 정보로 부터 설계자의 의도를 충실히 반영할 수 있는 곡면을 형성할 수 있으나, 일반적인 CAD/CAM 시스템에서 많이 쓰는 매개변수 스플라인 곡면 형태로 표현되지 않으므로 IGES 등을 통한 데이터 교환이 불가능하고, 복합 이동곡면의 경우는 접평면 연속을 만족시키지 못한다.

한편 곡선망으로부터 매개변수 스플라인 곡면을 형성하는 방법으로는 Coons 곡면, Gordon 곡면, Gossling의 DUCT 곡면, Hermite blended surface, Woodward의 skinning 곡면 등이 있으나 이들은 설계 의도의 반영에 한계를 갖고 있다[6-2][6-9].

본 연구에서는 곡률연속을 만족하는 쌍3차(bicubic)의 매개변수 스플라인 곡면 형태로 이동곡면을 표현하는데 중점을 두고 있다.

여기서 말하는 매개변수 스플라인 곡면(parametric spline surface)은 Coons, Bezier, Ferguson, B-spline 곡면들을 통칭하는 것으로 이들은 서로 쉽게 변환된다. 또한 본 연구에서는 기존의 이동곡면 형성 절차를 변경함으로

써 보다 원활한 곡면을 형성할 수 있음을 보여주고 있다.

2. 이동곡면의 형성 요소

이동곡면 형성에 필요한 요소로는 4가지 기하요소와 3가지 형성규칙이 있으며 이들은 사용자로부터 직접 입력받거나 경우에 따라서는 시스템 내부에서 결정할 수도 있다. 우선 기하 요소부터 살펴 보면 다음과 같다(그림 6.16 참고).

가. 단면곡선(section curve ; SC) : 단면(section plane) 상에 놓인 평면 곡선. 1개 이상 주어질 수 있다.

나. 경계곡선(boundary curve ; BC) : 단면곡선을 제외한 곡면의 경계를 이루는 3차원 공간 곡선이 주어지지 않을 수도 있으며 다수 개일 수도 있다.

다. 안내곡선(guide curve ; GC) : 단면의 이동을 안내하는 곡선으로 이동규칙을 적용하는 기준이되는 3차원 공간 곡선이다. BC가 이를 겸할 수 있다. 이동규칙 중 대응이동인 경우는 2개, 나머지는 1개가 필요하다.

라. 안내평면(guide plane) : 단면의 이동을 안내하는 평면으로 항상 1개가 주어져야 한다. 이 평면과 단면의 교선은 단면을 지역 좌표계로 나타내는 기준이 되며, 이동규칙 중 평행, 회전, 대응이동은 단면이 안내평면에 항상 수직을 유지하며 이동한다.

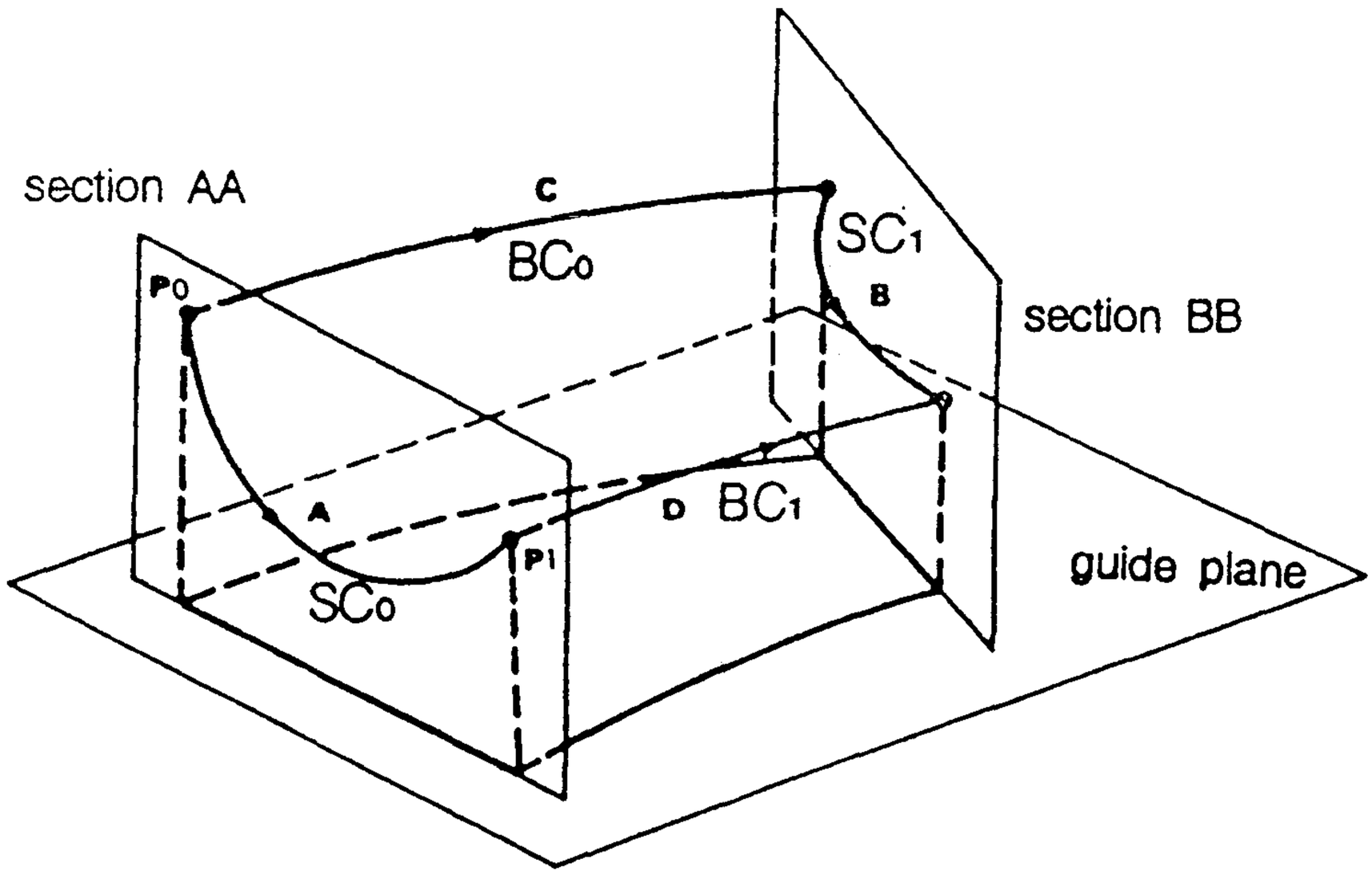


그림 6.16 이동곡면 형성의 개념

3. 이동곡면의 형성 규칙

이동곡면의 특징은 몇 가지 곡면형성 규칙을 선택함으로써 다양한 형상을 얻을 수 있다는데 있으며, 이들 규칙은 CAGD(computer-aided geometric design) 이전의 전통적인 곡면 형성 방법에 기초하고 있으므로 설계자가 그 형상을 쉽게 예측할 수 있다.

가. 이동규칙(sweeping rule) : 단면이 안내곡선을 따라 이동하는 규칙이다. 이동규칙에는 다음의 네가지 종류가 있다.(그림 6.17 참고)

a. 평행이동(parallel sweeping) : 단면이 평행 이동한다.

b. 회전이동(rotational sweeping) : 단면이 안내평면에 수직한 회전축을 중심으로 회전이동 한다.

c. 법선이동(normal sweeping) : 단면이 안내곡선과 수직을 유지하며 이동한

다.

d. 대응이동(synchronized sweeping) : 단면이 두 안내곡선의 곡선 길이 비에 대응되도록 이동한다.

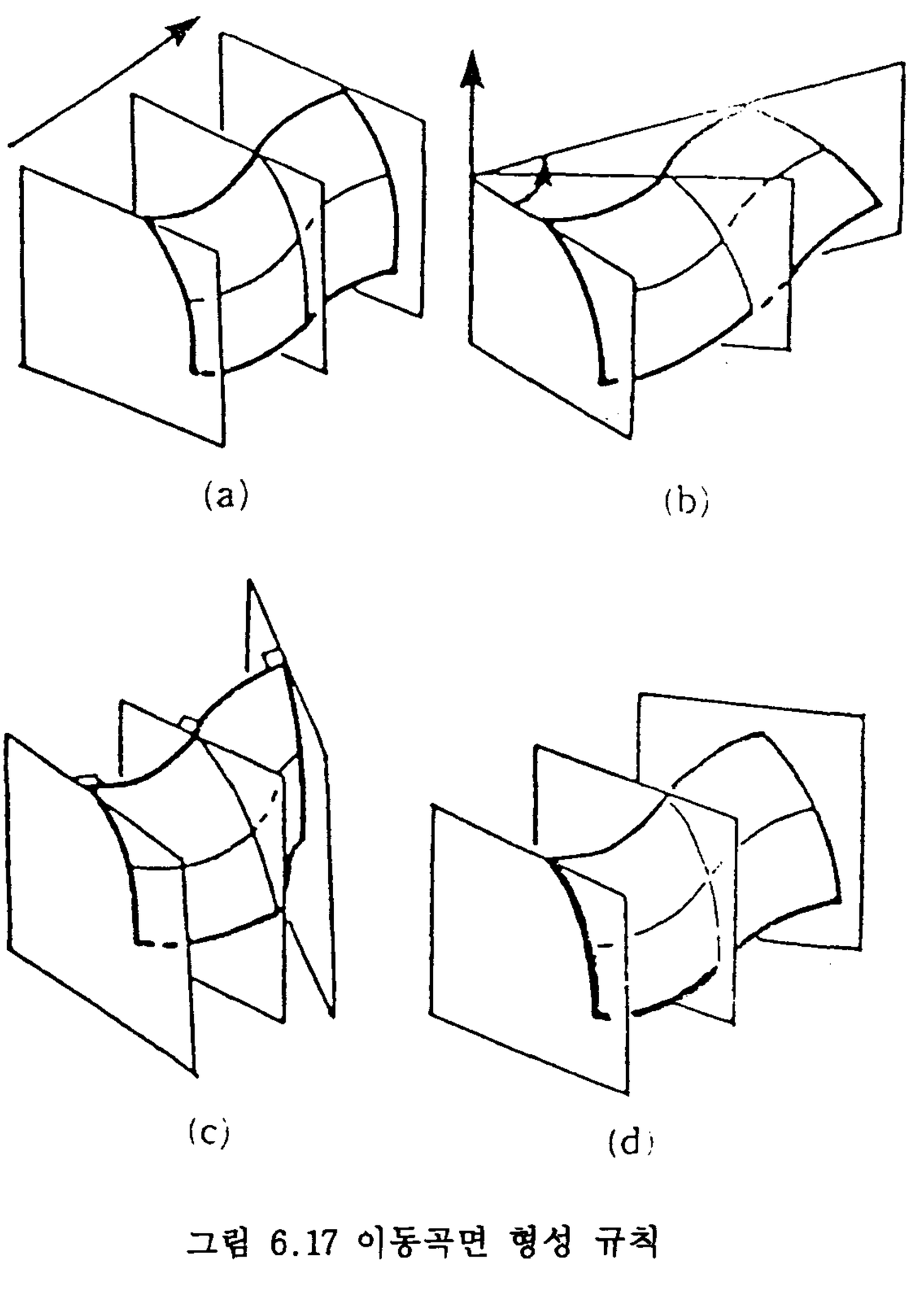


그림 6.17 이동곡면 형성 규칙

나. 혼합규칙(blending rule) : 단면곡선이 2개 이상 주어진 경우 임의의

중간 단면에서는 각각의 단면 곡선을 이동 거리에 따라 적절히 혼합하여야 한다.

a. 선형 혼합(linear blending) : 선형 혼합함수를 이용하여 주어진 두 단면 곡선을 혼합하여 중간 단면곡선을 만든다. 단면곡선이 2개 주어진 경우에 이용될 수 있다.

b. 3차 혼합(cubic blending) : 단면곡선이 3개 이상 주어지는 경우 전체적인 단면곡선의 형상변화를 고려하여 중간 단면곡선을 생성한다. Ferguson fitting 이나 Chord-length spline fitting에서와 같은 3차 혼합함수를 이용한다.

다. 수정규칙(compensation rule) : 경계곡선이 두개 이상 주어진 경우 안내곡선 상의 임의 위치로 이동되고 혼합된 중간 단면곡선의 그물점(node point;곡선망의 교점)은 경계곡선과 만나지 않는다. 이를 일치시키기 위한 중간 단면곡선의 수정방법은 다음과 같은 것이 있다.

a. Affine 변환을 이용한 수정 : 중간 단면곡선의 평행이동 변환, 회전 변환 축척변환을 통하여 중간 단면 곡선을 수정한다. 이는 단면의 모양을 쉽게 예측할 수 있는 방법이나, 경계곡선이 다수 개인 경우 접평면 연속을 만족 시키기 어렵다.

b. 차이의 보간에 의한 수정 : 경계곡선과 중간 단면의 교점을 $\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2, \cdots, \mathbf{b}_m$ 이라 하고, 이동된 단면곡선을 $\mathbf{p}(v)$, 그의 그물점을 $\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_m$ 이라 할 때 각 점에서의 차이벡터는 $\mathbf{d}_i = \mathbf{b}_i - \mathbf{p}_i$, $i = 1, 2, \cdots, m$ 로 정의 된다. $\mathbf{d}_0, \mathbf{d}_1, \mathbf{d}_2, \cdots, \mathbf{d}_m$ 을 보간한 곡선을 $\mathbf{d}(v)$ 라 하면 수정된 단면곡선 $\mathbf{b}(v)$ 는 다음과 같다.

$$b(v) = p(v) + d(v)$$

$d(v)$의 보간시 $m = 2$ 이면 선형보간을 행하고, $m \geq 3$ 이면 cubic spline 보간을 이용한다. 경계곡선이 3개 이상인 경우 G^2 연속 조건을 만족시킬 수 있다.

4. 매개변수 스플라인 곡면식에 의한 이동곡면의 모델링 절차

본 연구에서 입력받는 곡선은 곡면의 표현 형식(함수의 종류)이나 차수(degree)에 제한이 없으며 여러 개의 segment로 이루어진 복합 곡선(composite curve) 이다. 이러한 단면곡선, 경계곡선 등의 기하요소와 형성 규칙으로부터 이동곡면을 표현하고자 한다. 일반적인 CAD/CAM 시스템의 표준적 곡면 표현 형식인 매개변수 스플라인 곡면식에 의한 이동곡면의 모델링 절차는 다음과 같다.

가. 단면곡선 remeshing : 모든 단면곡선들이 같은 갯수의 마디점(mesh points)을 갖도록 곡선 길이를 기준으로 remeshing 한다[6-1][6-13]. remeshing된 곡선은 주어진 곡선과 근사 공차를 만족하는 3차 현길이 스플라인[6-13]이 된다(그림 6.18 참고). 본 과정에 대한 보다 자세한 절차는 참고문헌[6-11]에 기술되어 있다.

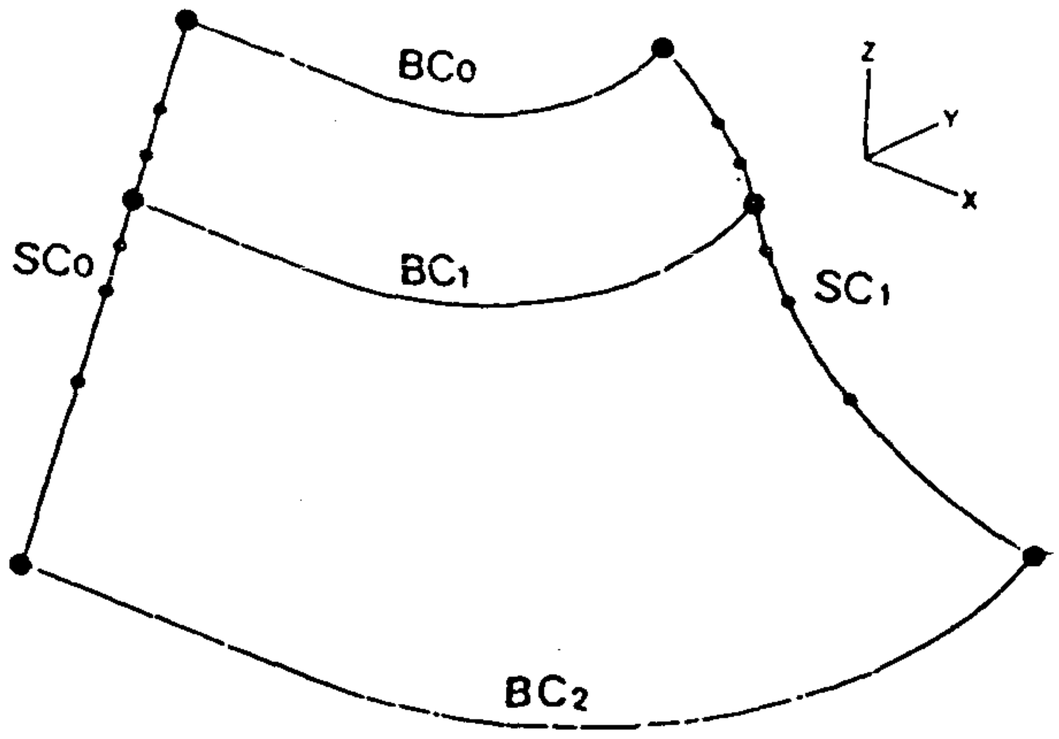


그림 6.18 단면곡선 remeshing

나. 경계곡선과 안내곡선의 remeshing : 모든 경계곡선과 안내곡선이 같은 개수의 마디 점을 갖도록 경계곡선과 안내곡선에 새로운 점을 삽입한다. 삽입되는 점의 위치는 곡선의 근사오차와 이동규칙을 적용하여 얻어진다. 그림 6.19는 회전이동 규칙을 적용한 경우의 경계곡선 remeshimg 결과를 보여주고 있다.

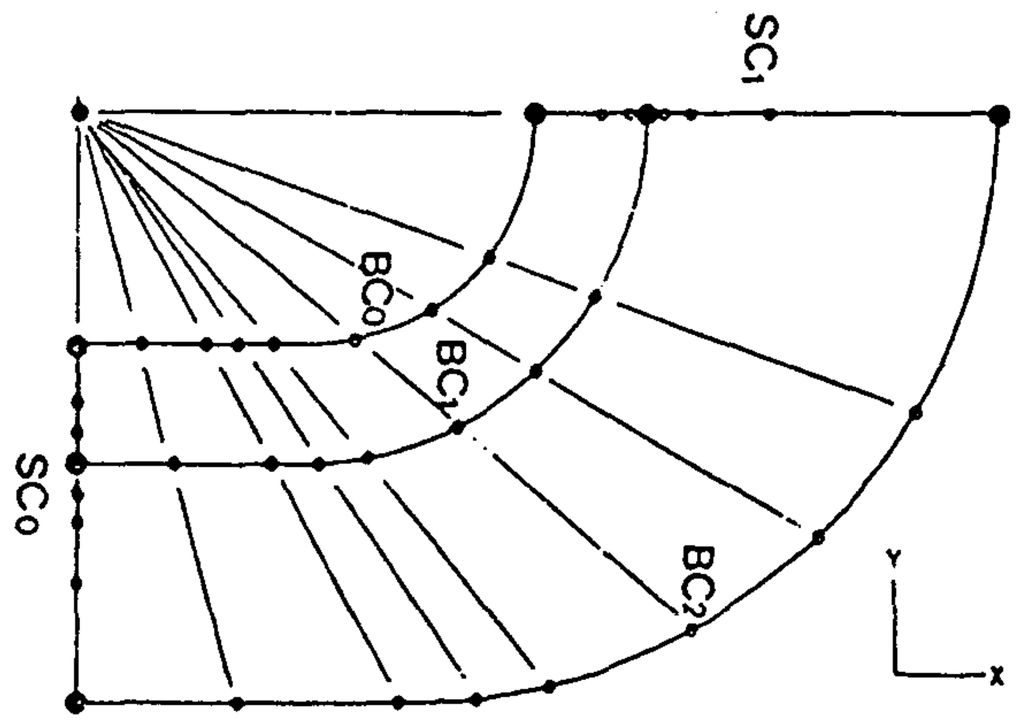


그림 6.19 안내곡선과 경계곡선의 remeshing

다. 단면곡선의 혼합 : 초기 단면 곡선을 혼합(blending)하여 '나.'의 과정에서 얻어진 안내곡선의 마디점에서의 중간 단면곡선을 얻는다. 그림 6.20(b)와 같이 초기 단면을 경계곡선의 평균길이 만큼 띄워 평행하게 나열하고 초기 단면곡선들을 혼합규칙에 따라 보간하여 곡면을 생성한다. 이 곡면과 중간 단면의 교선이 중간 단면곡선이 된다.

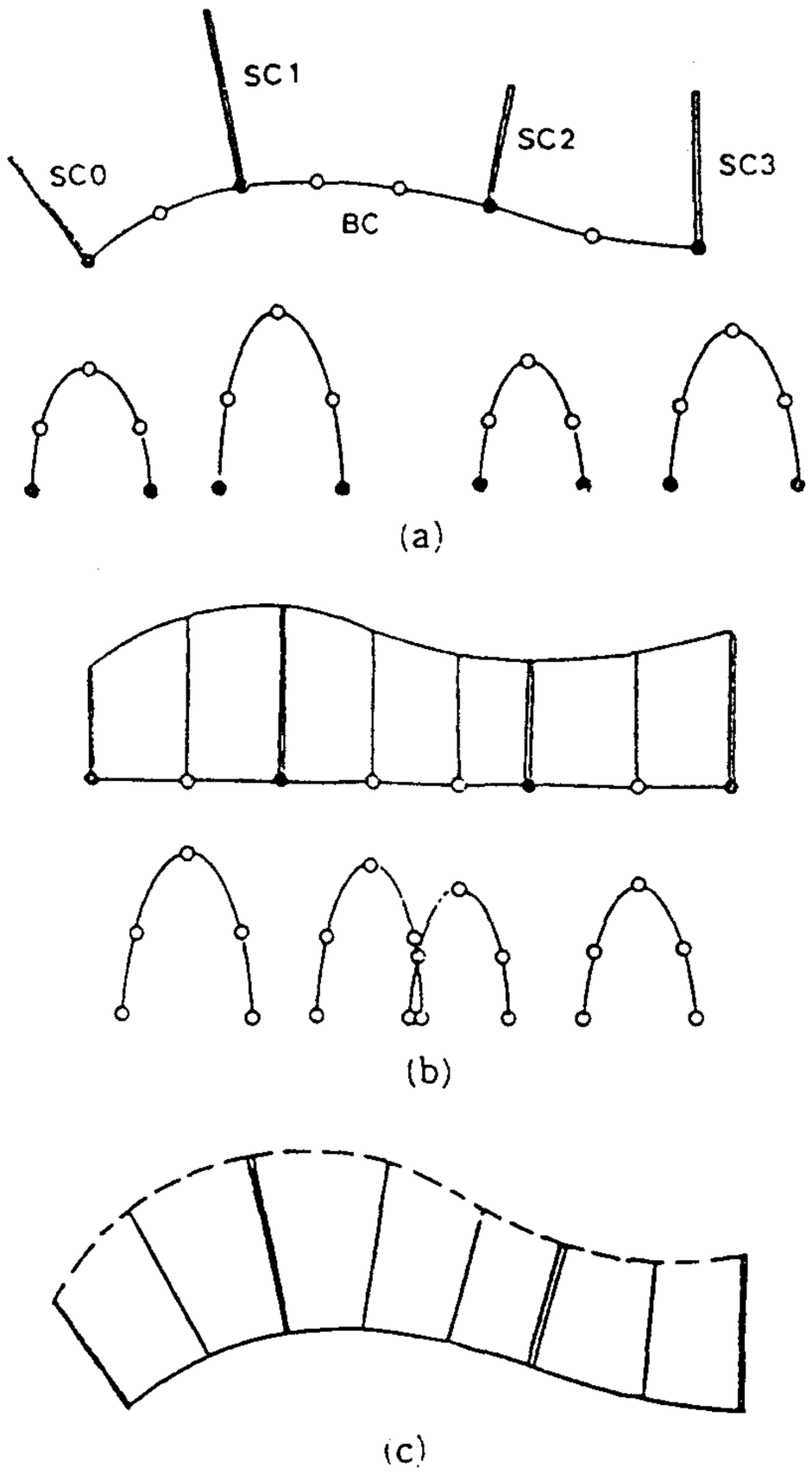


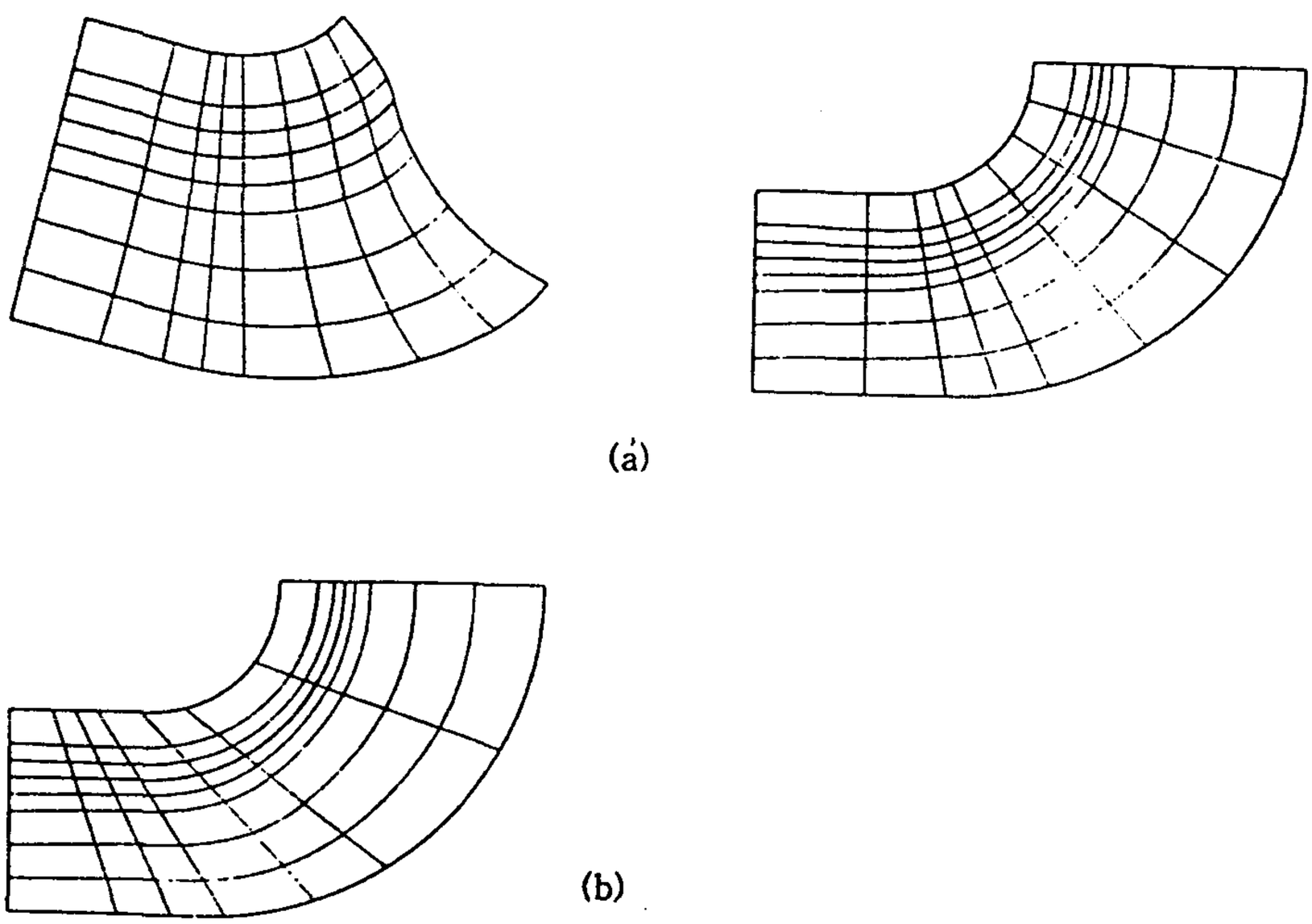
그림 6.20 단면곡선의 확장

(4) 중간단면 곡선의 수정 : 중간단면 곡선의 그물점이 경계곡선과 일치하도록 수정규칙에 따라 중간단면곡선을 수정한다.

(5) 단면곡선의 이동 : 이동규칙에 따라 단면곡선들을 경계곡선 상의 위치로 이동변환 한다(그림 6. 20(c)).

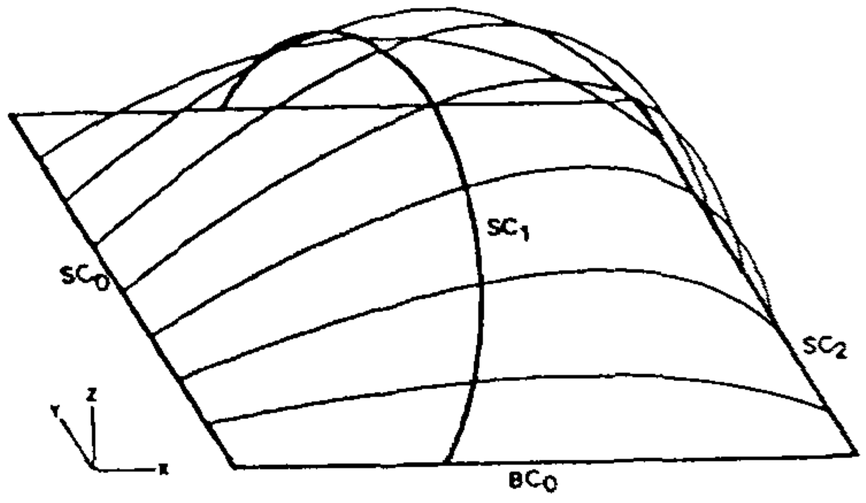
(6) 곡면 보간 : 이상과 같은 과정을 거치면 초기단면과 중간단면에 놓인 곡선들은 3차원 공간 상에서 사각 배열의 마디점을 갖게 된다. 이 곡선 데이터를 Hermite 보간을 이용한 곡률연속의 현길이 곡면[6-13]으로 보간하여 곡면의 식을 얻는다.

그림 6.21(a,b)는 그림 6.18에 표시된 곡선으로부터 형성된 법선, 회전 이동곡면을 보여주고 있다. 각 그림은 곡면의 등매개곡선을 그린 것인데 등매개곡선들이 이동규칙을 정확히 따르고 있음을 보여준다(Sweep 곡면을 생성하는 함수는 부록 E를 참조)

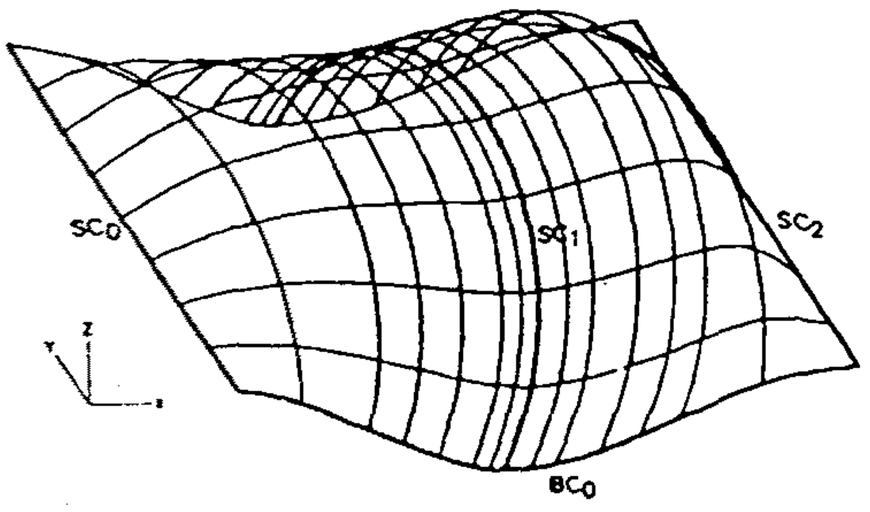


(a)

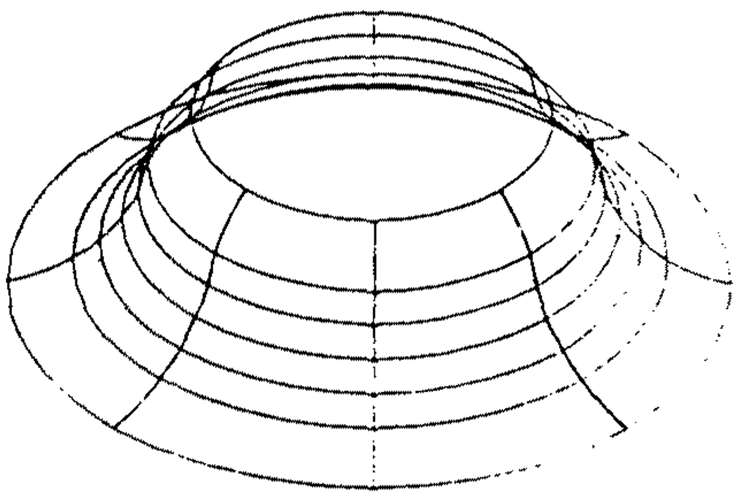
(b)



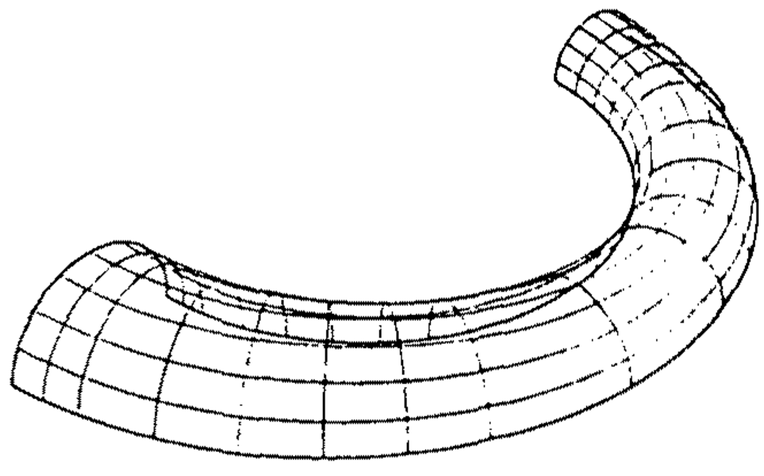
(c)



(d)



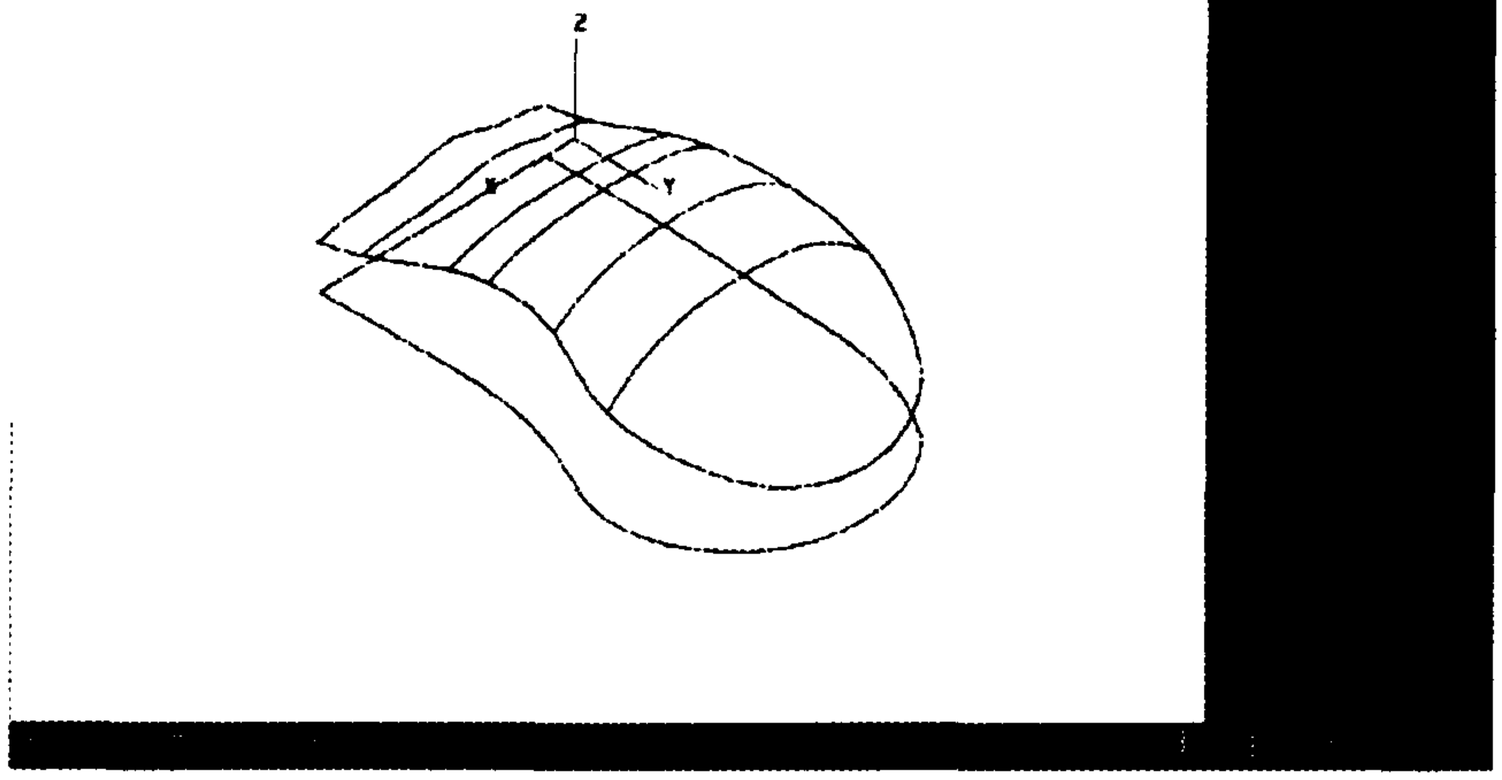
(e)



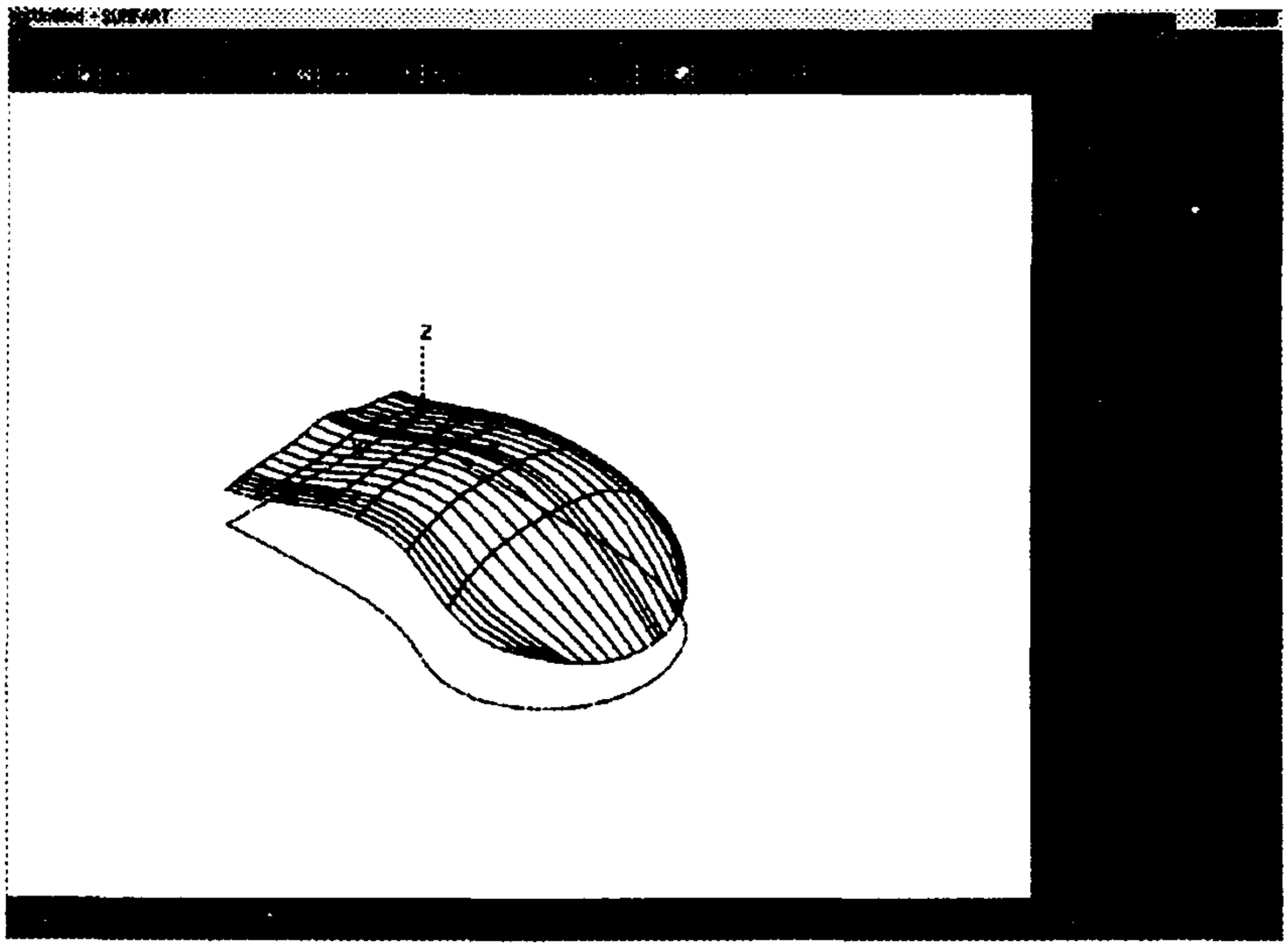
(f)

그림 6.21 이동곡면의 예

그림6.21(c)는 3개의 단면곡선과 1개의 경계곡선(직선)으로 형성한 평행이동 곡면이며, 그림6.21(d)는 (c)의 경계곡선을 직선에서 곡선으로 대체한 것이다. 두 그림 모두 패치의 경계선을 그린 것이다. (c)에서는 경계곡선을 remeshing하지 않고도 곡면을 보간 할 수 있는 경우이며 (d)는 경계곡선이 remeshing 되었는데 중간 단면곡선이 매우 자연스럽게 형성되고 있음을 보여주고 있다.



(a) section curves



(b) skinning

그림6.22 Skinnig을 이용한 곡면 모델링

제 5 절 자동 트림곡면 생성

1. 트림곡면(trimmed surface)의 정의

트림곡면(trimmed surface)이란 곡면상의 곡선(curve-on-surface)에 의해 영역이 분리된 곡면을 말한다. 트림 곡면의 생성 방법으로는 공간상의 곡선을 곡면에 투영시키거나(curve projection), 곡면간의 교차곡선을 구하거나(surface intersection), 블렌딩 곡면이나 필렛팅(filletting) 곡면의 경계를 이용하거나, 곡면의 iso-parametric 곡선을 이용할 수 있는데 내부적으로는 곡면상의 곡선(curve-on-surface)을 트림곡면의 나이프(knife - 이것을 트리밍 곡선이라 하자.)로 이용한다.

그림6.23에서 S는 기준곡면(base surface), f는 face, h는 face내부의 hole 즉, closed loop을 나타내며, f를 육지라고 생각하면 h는 호수, S는 바다에 비유될 수 있다. 바다 또는 호수에 떠있는 섬(island)들은 별도의 face로 간주한다.

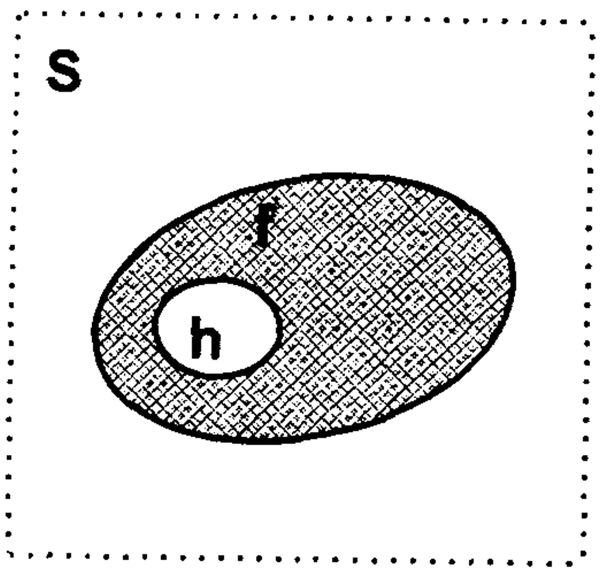


그림 6.23 내부 loop이 있는 face

트림곡면이 경계를 이루는 경계곡선(boundary curve)은 방향성을 갖고 있는데 경계곡선의 왼쪽을 트림곡면으로 간주하는 것이 일반적이다. 그림6.24에서 face의 외부 loop은 반시계 방향, 내부 loop은 시계 방향을 갖는다.

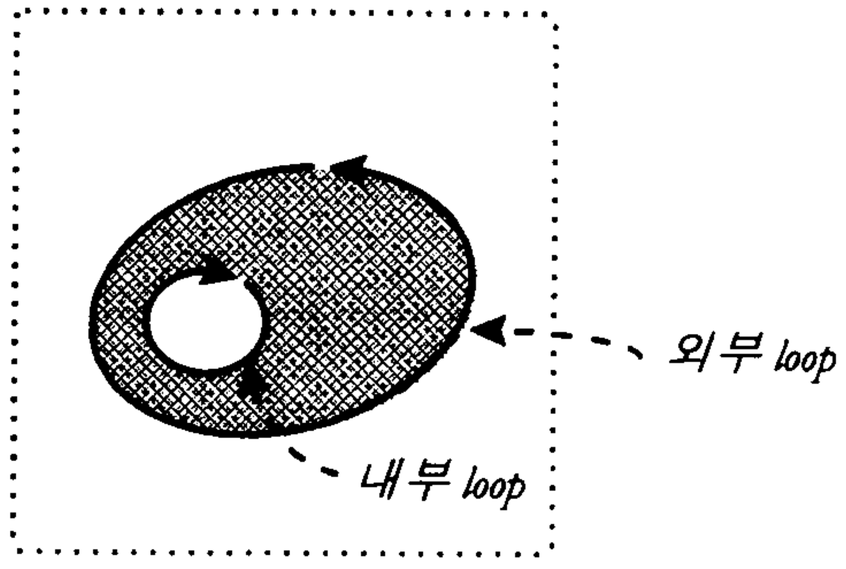


그림 6.24 loop의 방향성

트리밍 알고리듬에 대한 기존 연구로는[Farouki 87]과 [전용태 93] 등이 있으며 Farouki는 B-rep에서의 트림곡면에 대한 정의를 내렸다. Farouki 알고리즘은 트리밍 곡선의 self-intersection, cusp을 고려하였고, 트림곡면의 island도 표현 가능하다는 특징을 갖고 있다.

2. 트림곡면이 생성되기 위한 조건

트림곡면(trimmed surface)이 생성되기 위해서는 트리밍 곡선(trimming curve)이 곡면(또는 face)을 두 개 이상의 영역으로 확실히 분리할 수 있어야 하는데 이를 달리 말하면 트리밍 곡선으로 새로이 '닫혀진(closed)' 경계를 형성해야 한다는 것이다.

트림곡면이 생성되기 위해 트리밍 곡선이 만족해야할 두 가지 조건을 정리하면 다음과 같다(이 절에서 '곡선'은 특별한 언급이 없는 한 '트리밍 곡선'을, '곡면'은 '곡면' 또는 'face'와 같은 뜻으로 쓰인다).

조건1. 트림곡면이 생성되기 위한 조건

곡선의 self-intersection은 발생하지 않는다(그림6.25).

개곡선(open curve)인 경우

(곡선의 두 끝점이 모두 곡면의 동일한 외부영역에 존재)

AND

(곡면의 경계와 짝수 번 교차)

이며,

폐곡선(closed curve)인 경우

(곡면의 내부영역과 폐곡선으로 형성된 영역간에 교차영역이 존재)

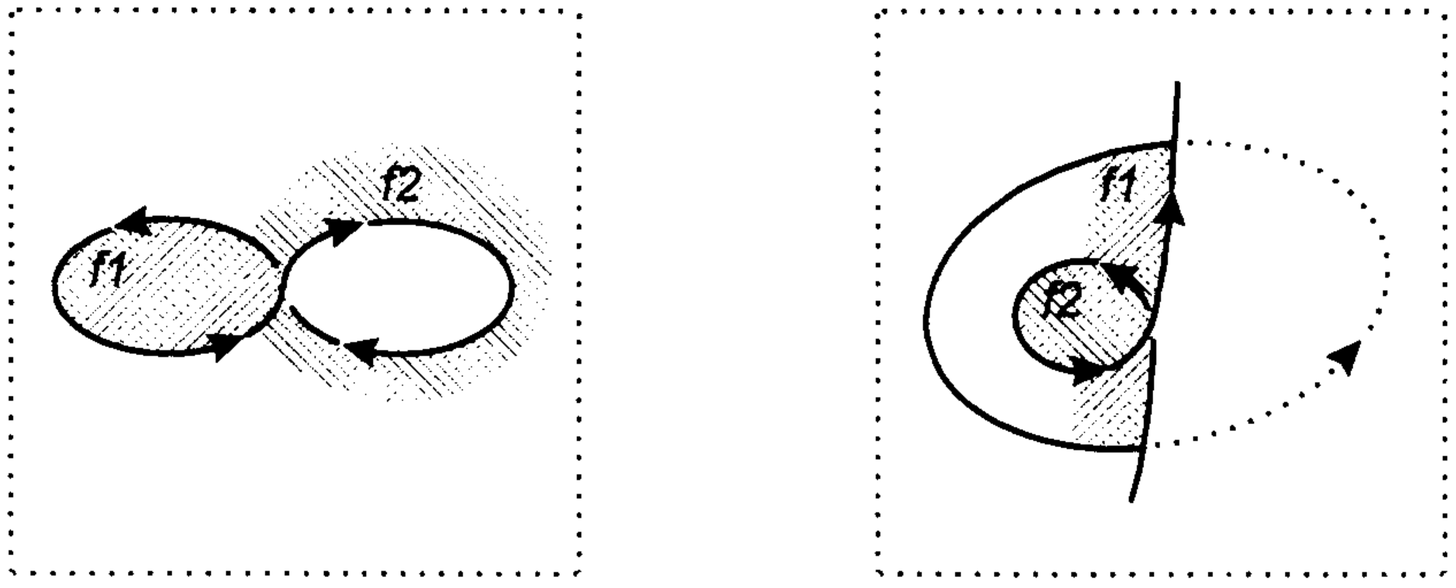


그림 6.25 곡선의 self-intersection

조건1에서 '곡면의 동일한 외부영역'이라 함은 그림6.26의 S 또는 h 영역을 가리킨다.

그림6.26은 트림곡선이 개곡선일때 트림곡면이 생성될 조건이 만족하는 경

우(○로 표시)와 만족하지 않는 경우(×로 표시)로 나타내었다.

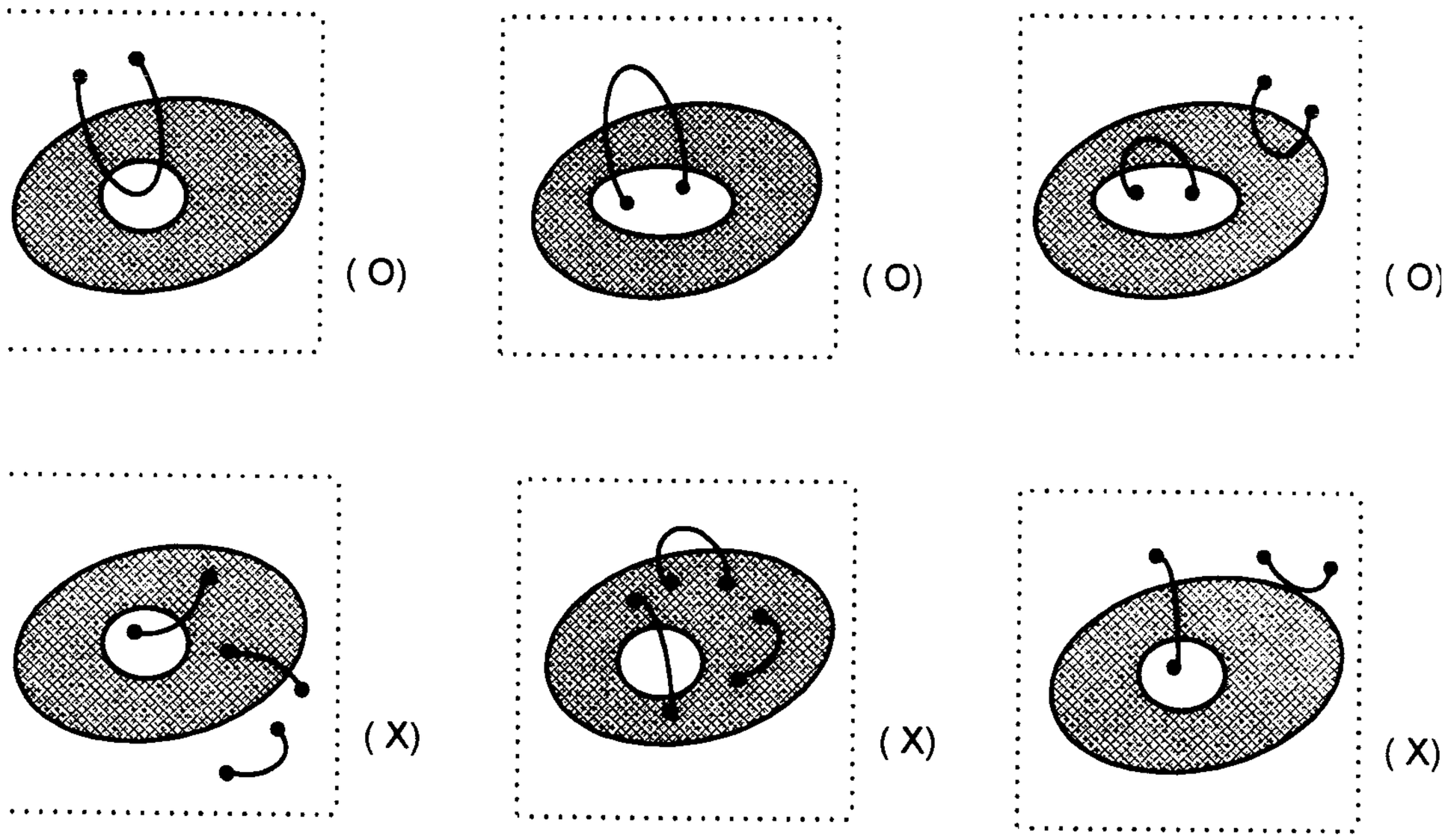
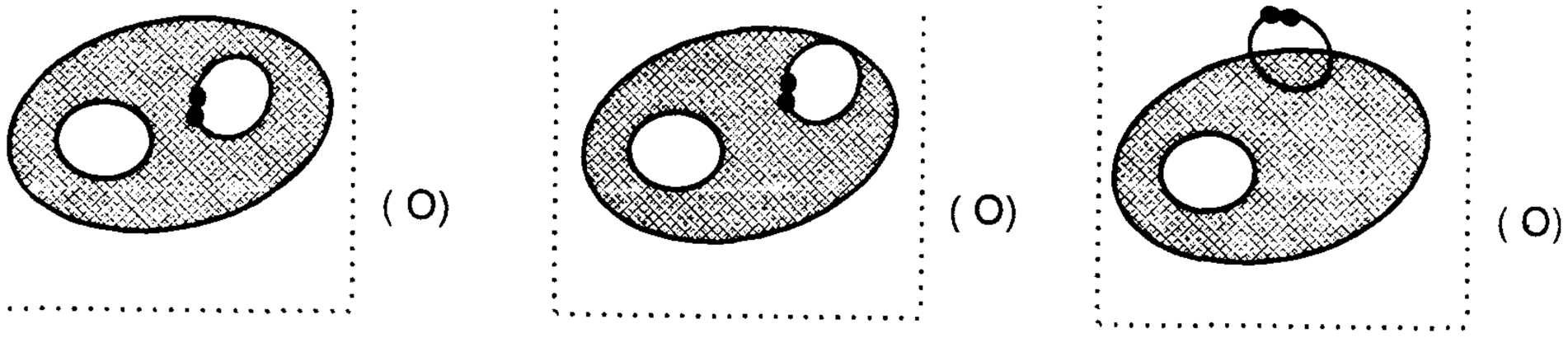


그림 6.26 트림곡면의 생성 조건을 만족하는 개곡선과 만족하지 않는 개곡선

곡면의 내부영역과 폐곡선의 내부영역간의 교차영역이 존재하는 경우는 그림 6.27처럼 폐곡선 전체가 곡면과 교차영역이 되는 경우(a)(b)와 폐곡선의 일부분이 교차 영역이 되는 경우(c)가 있다. c의 경우는 트림곡선이 경계 edge에 의해 트리밍되어 두 개 이상의 개곡선이 되므로 내부적으로는 개곡선인 경우와 동일한 트림곡면 생성절차를 거치게 된다.



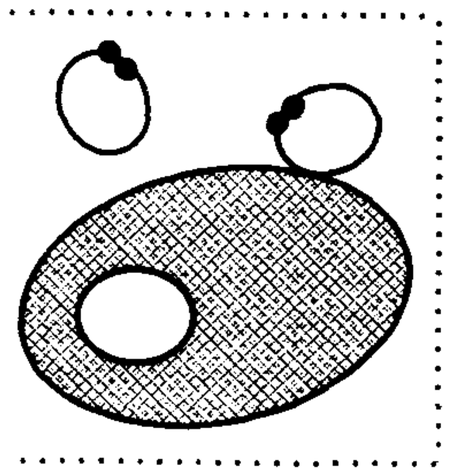
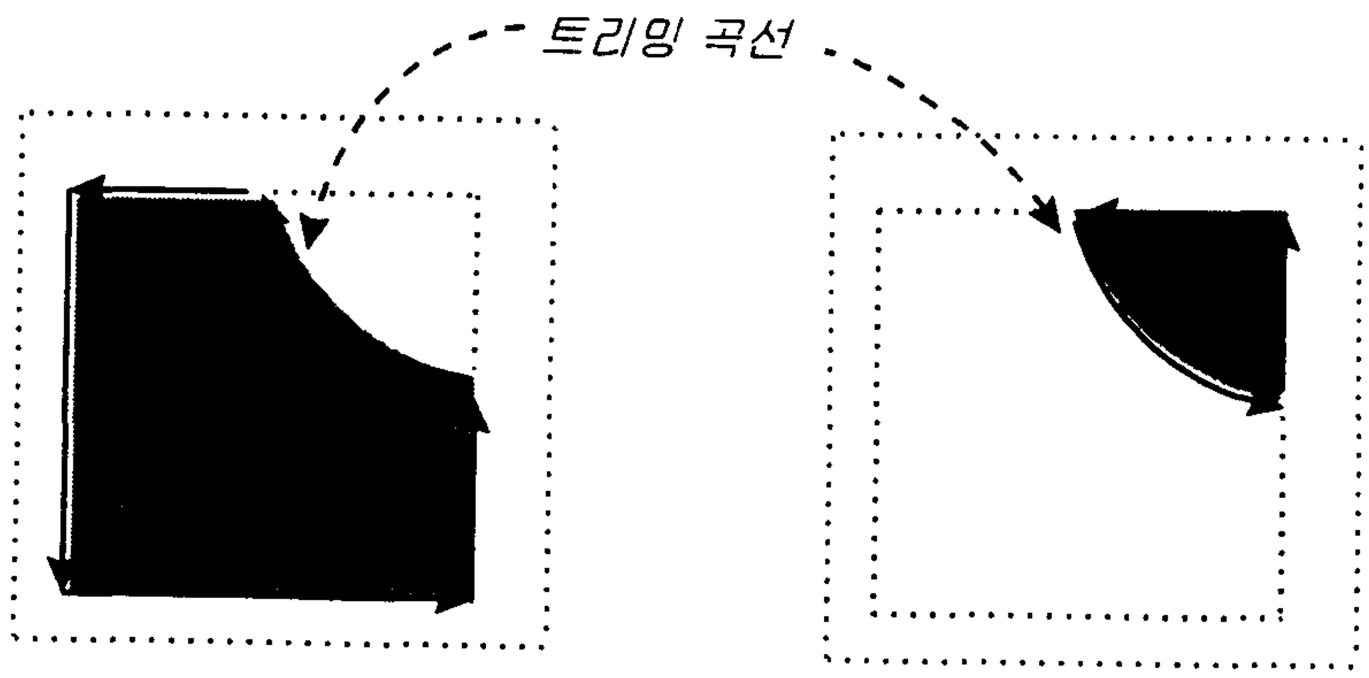


그림 6.27 트림곡면 생성될 조건을 만족하는 폐곡선과 만족하지 않는 폐곡선

트리밍 곡선의 방향에 따라 생성되는 트림 곡면이 달라지는데, 특히 트리밍 곡선이 폐곡선이면서 시계방향이면 기존 곡면에 hole이 생기고 반시계 방향이면 새로운 트림곡면이 생성된다(그림 6.28).



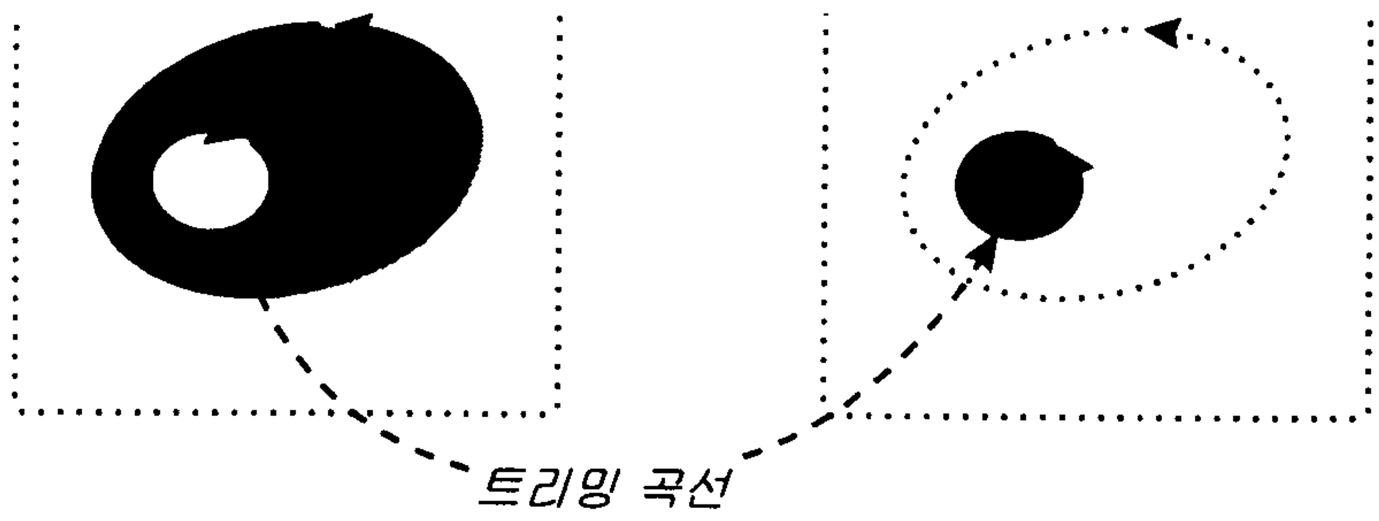


그림 6.28 트리밍 곡선의 방향에 따른 트림곡선의 생성

3. 트림곡면 생성 알고리즘

트림곡면 생성 알고리즘은 3단계를 거친다

```
step 1 :

  for (모든 트리밍 곡선)
  {
      for (face의 경계를 이루는 모든 edge)
      {
          경계 edge와 트리밍 곡선과의 교점을 구한다.
          if( 교점이 존재)
              교점에서 트리밍 곡선과 edge를 splitting(그림 6.29).
      }
  }
```

```
step 2 :
  face 영역 밖의 트리밍 곡선을 제거한다(그림 6.30).
step 3 :
  for (face의 경계를 형성하지 않는 트리밍 곡선)
 {
     if (closed curve & 시계방향)
         기존 face의 내부 loop으로 등록한다(그림 6.31).
     Else
     {
         while (closed loop이 형성 안됨)
             가까운 edge나 트리밍 곡선을 찾아 연결한다(그림 6.32).
         새로운 face를 생성한다(그림 6.33).
     }
 }
```

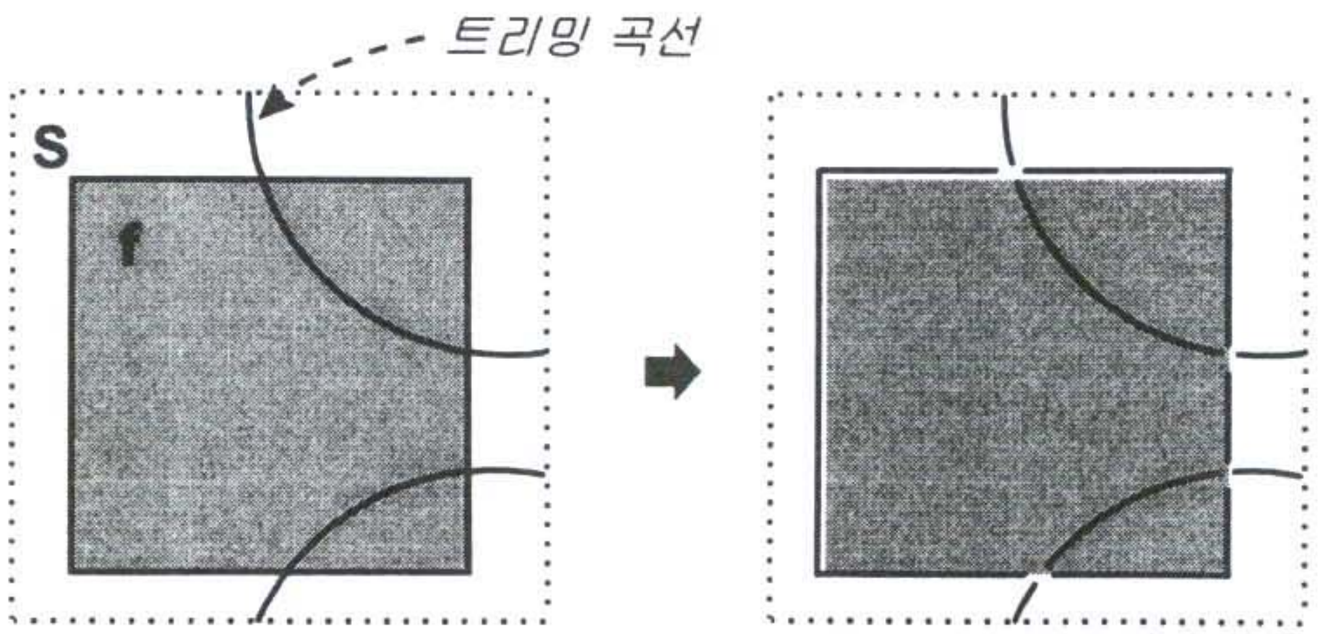


그림 6.29 step 1 : 트리밍 곡선과 edge의 splitting

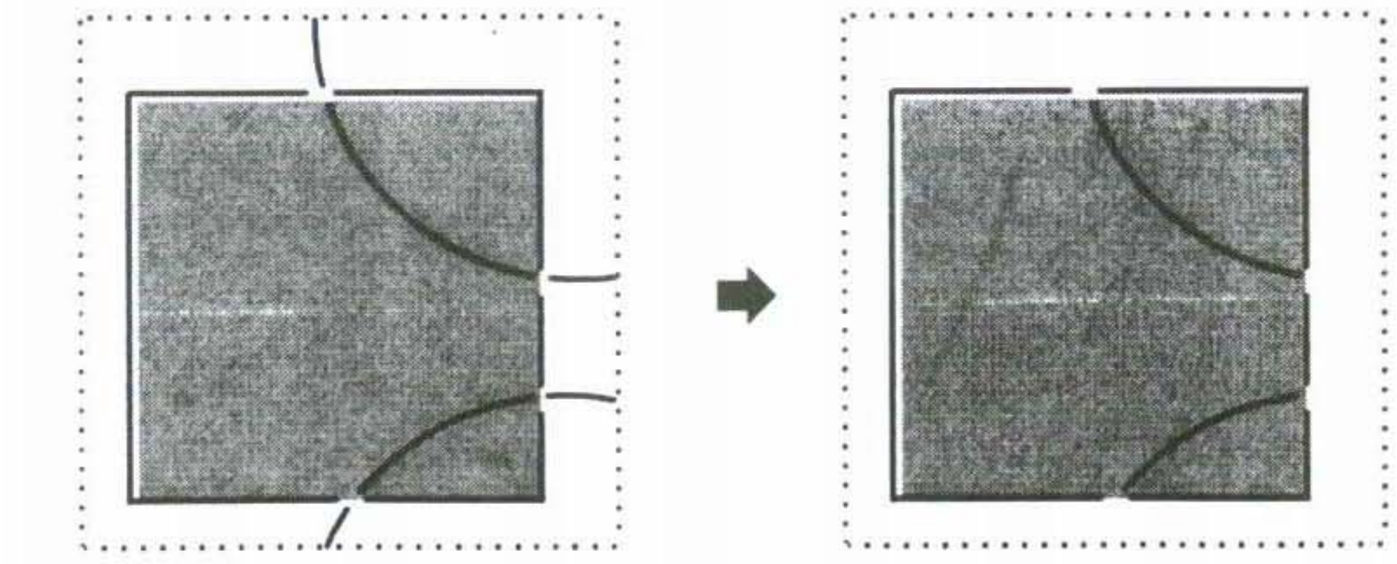


그림 6.30 step 2 : face 영역 밖의 트리밍 곡선 제거

그림 6.31 step 3 : face의 내부 loop 생성

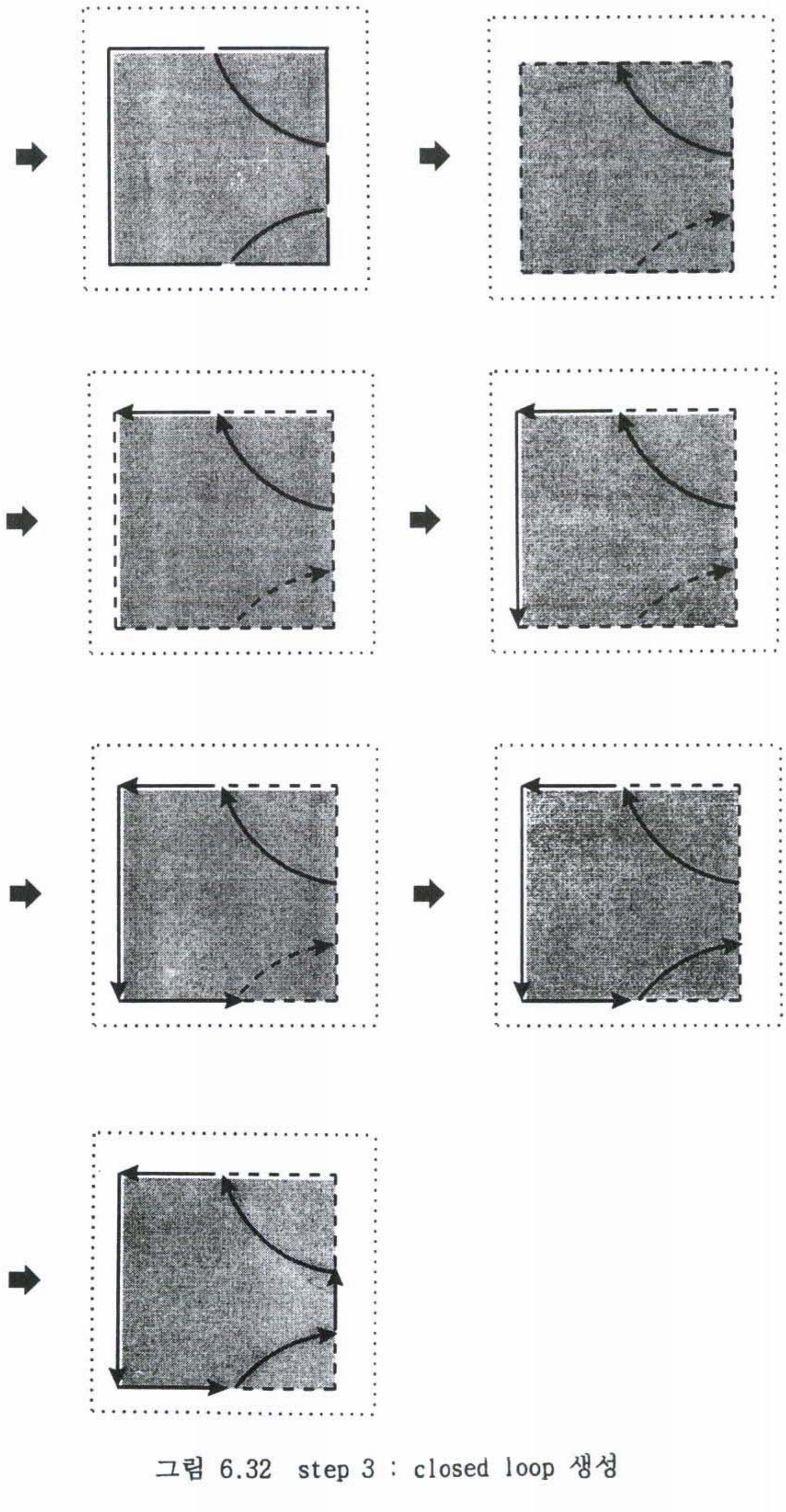


그림 6.32 step 3 : closed loop 생성

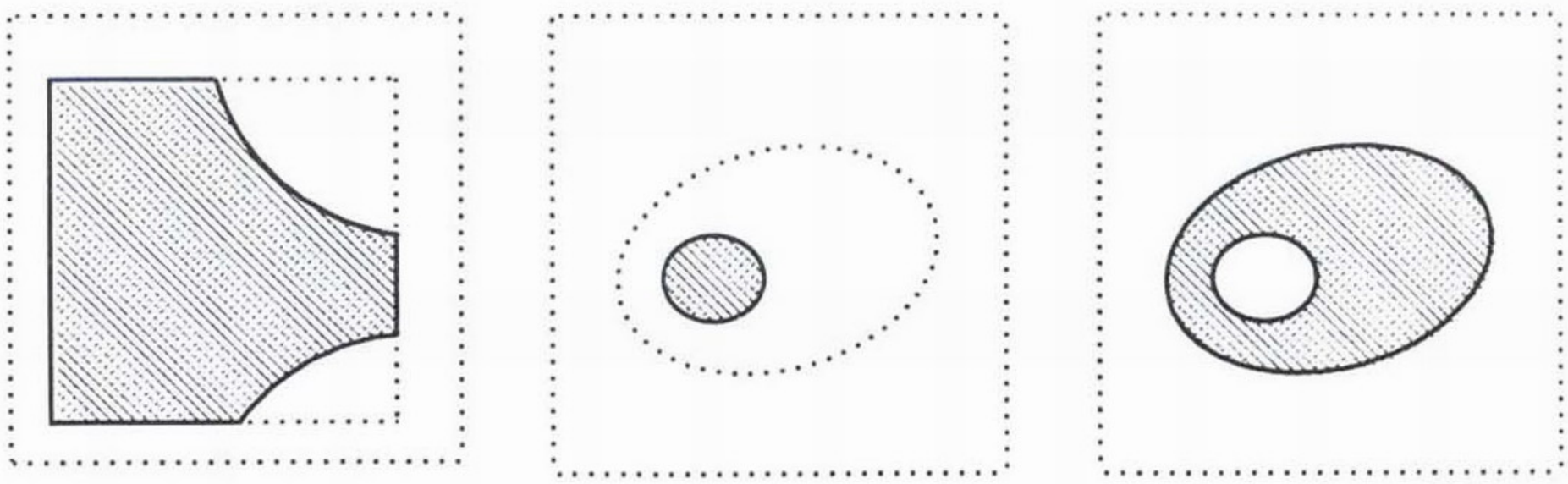


그림 6.33 step 3 : 새로운 face의 생성

Step3에서 '가까운 edge나 트리밍 곡선을 찾아' 연결할 때 허용오차 내에 드는 곡선이 2개 이상 존재할 수도 있다(그림 6.34). 이때 최근에 연결된 edge가 e1일 때, e2와 e3가 다음에 연결될 edge 후보가 된다. face는 edge의 왼쪽에 존재하므로 e2와 e3중 e1의 왼쪽에 놓인 e2를 연결 할 edge로 선택한다.

그림 6.34 후보 edge가 다수 개 존재하는 경우

본 모델러에서 자동 트리밍은 블렌드 곡면 생성시 옵션으로 적용된다. 따라서 특별한 API 함수로 존재하지 않는다.

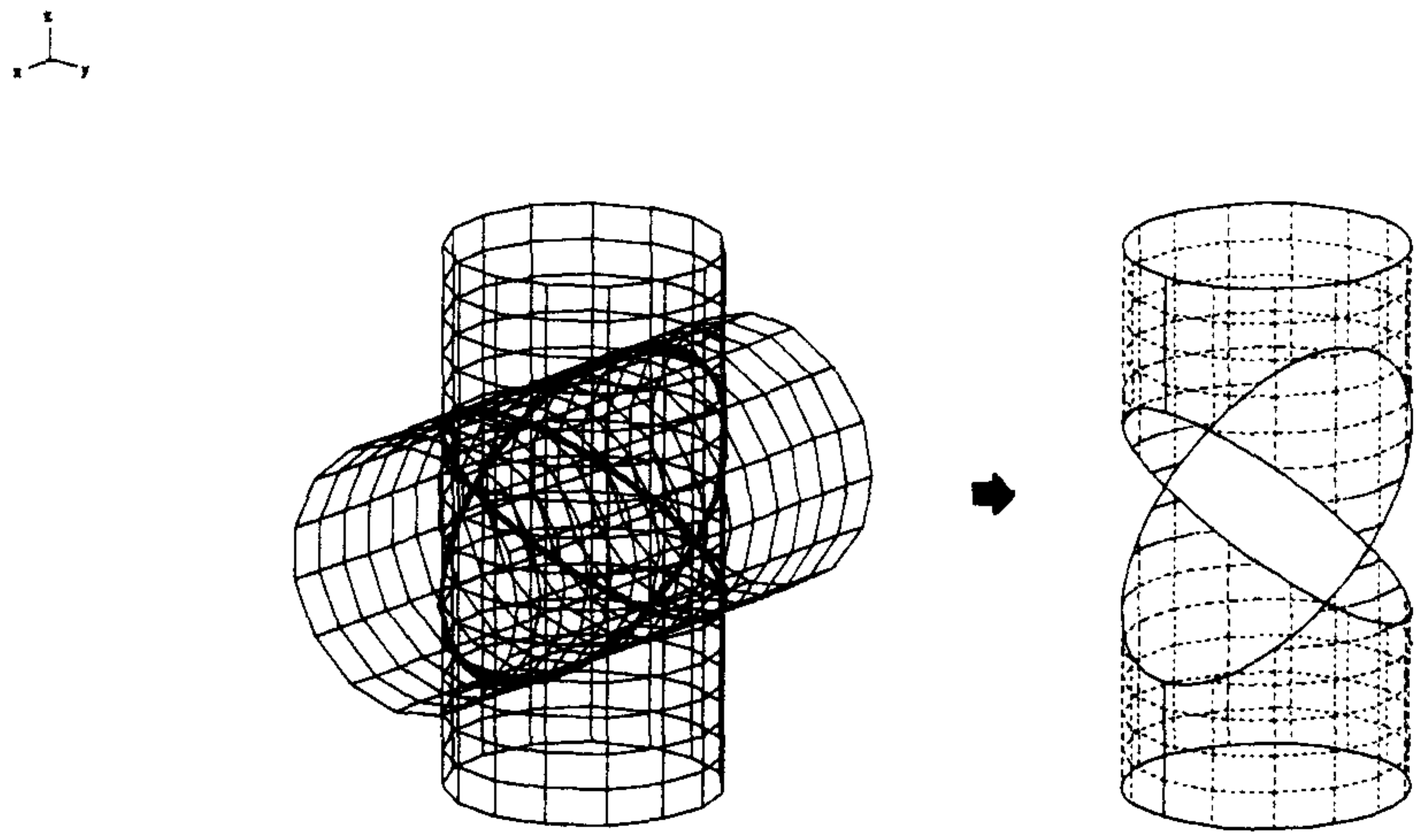


그림 6.35 곡면간의 교선과 트리밍1

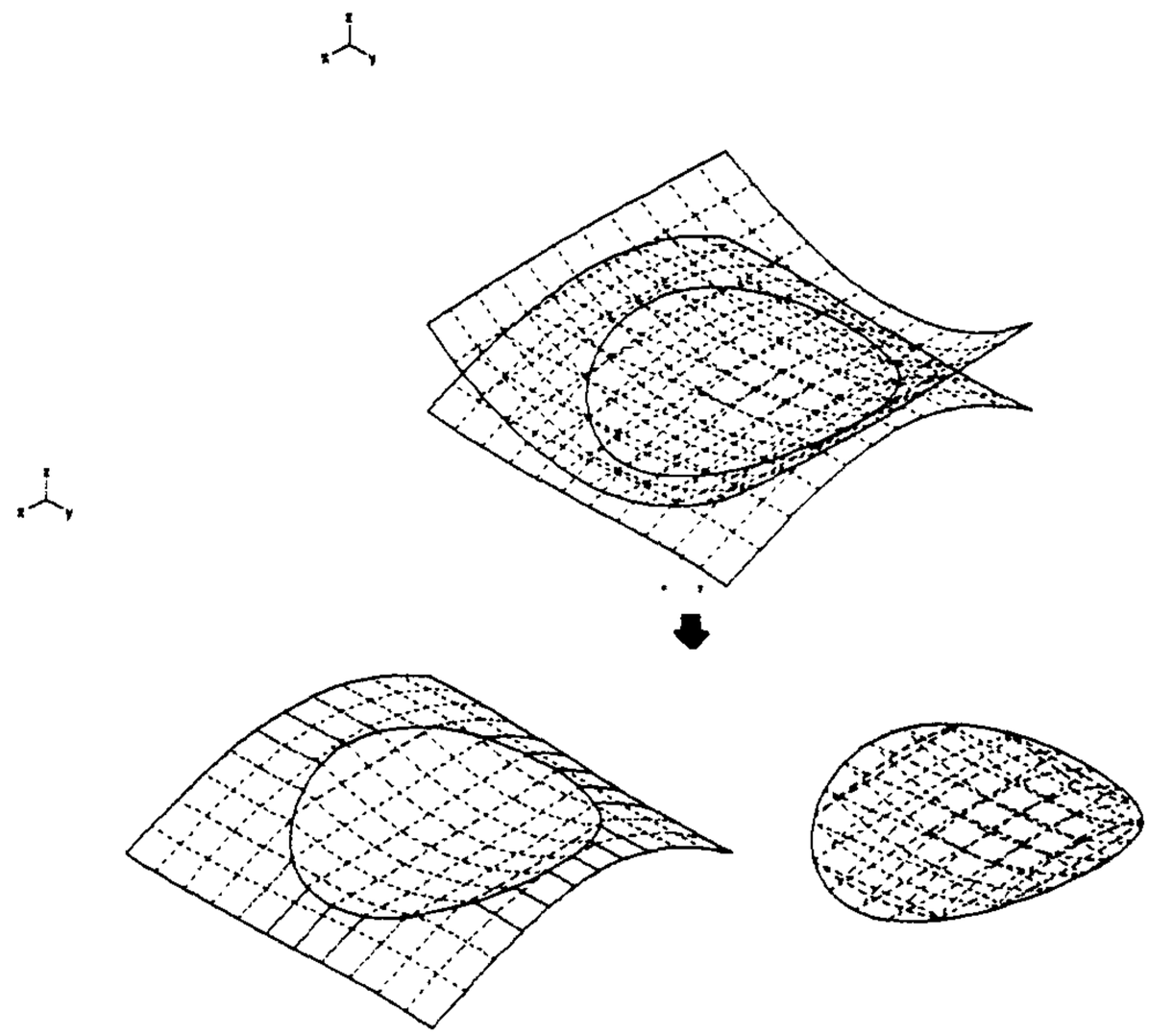


그림 6.36 곡면간의 교선과 트리밍2

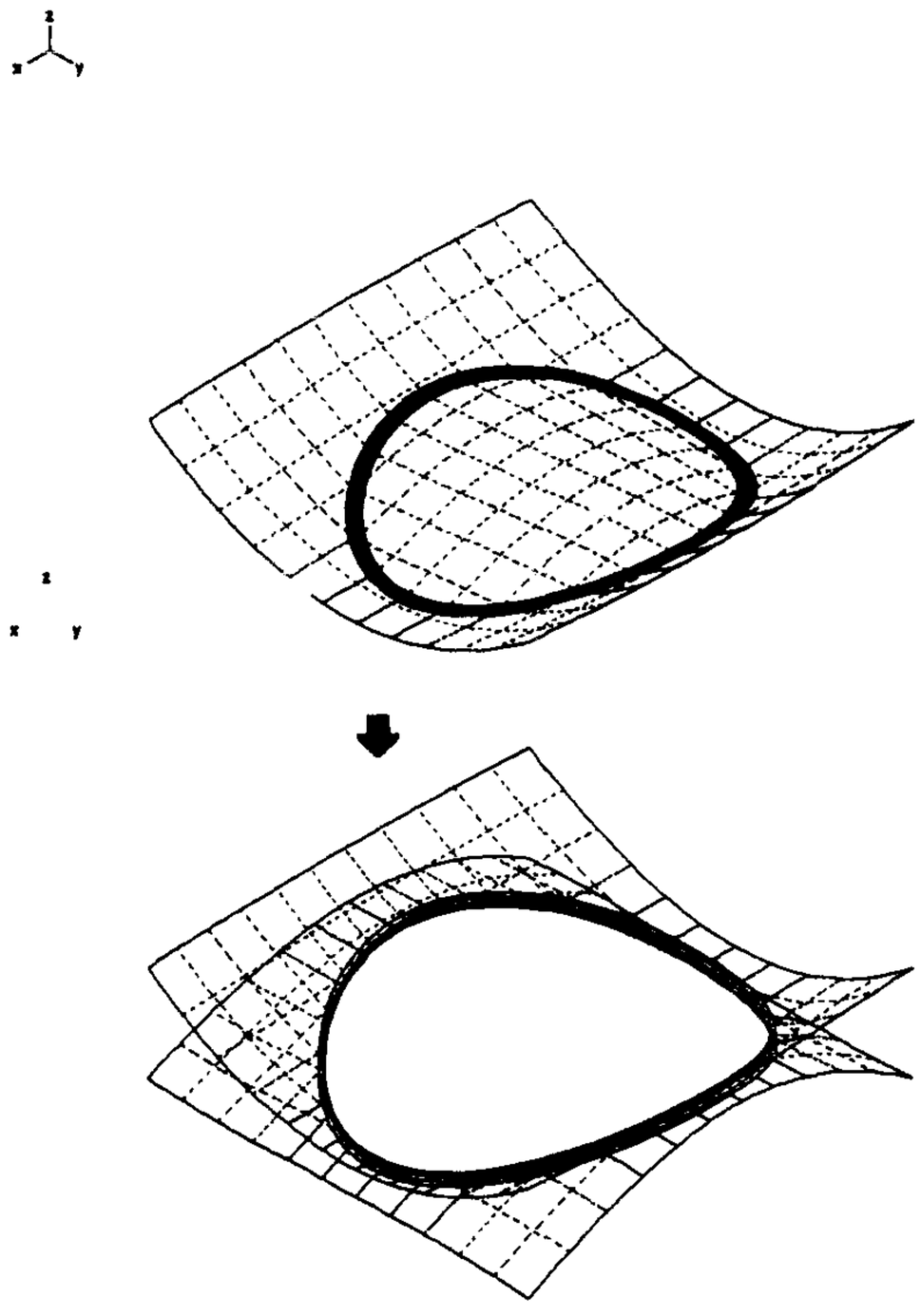


그림 6.37 블렌드 곡면의 트리밍

제 7 장 타 시스템과의 인터페이스

제 1 절 IGES Interface

IGES(Initial Graphics Exchange Specification)는 1980년도에 미국 상무성의 국가표준국(NBS, National Bureau of Standards)에서 만들어진 그래픽 정보의 교환을 위한 미국의 표준규격이다. IGES는 제품정의 데이터의 수치적 표현 및 교환을 위한 정보구조를 제공하는 것을 목적으로 하고 있다. IGES에서 표현하는 데이터의 기본단위를 entity라고 하는데, entity는 geometry, annotation 및 structure entity의 3가지 영역으로 나뉜다. geometry entity는 curve, surface 및 solid를 표현하며, annotation entity는 지시선, 치수, 심벌, 단면도 등의 도면정보를 나타낸다. 그리고 structure entity는 문자의 크기, line의 굵기, 색상의 정의, grouping 관계, FEM(Finite Element Model)의 요소 등을 나타낸다. neutral 파일을 이용한 이기종간 형상데이터의 교환을 위한 수단으로, 다른 어떤 파일 형식보다도 널리 사용되고 있어 실질적인 세계표준이라 볼 수 있다. 이 보고서에서는 IGES version 5.0에 기초를 두고 있다.

1. IGES 파일구조

1 ····	···· 72 73	···· 80	
주석(file 명, 작성자 등)			Start Section
Software 환경에 대한 정보			Global Section
데이터의 색인 및 속성 정보 fixed format			Directory Entry Section (DE Section)
실제의 데이터 값(좌표, 파라미터 값 등) free format			Parameter Data Section (PD Section)
S n_1G n_2D n_3P n_4	T	1	Terminate Section

※ n_1, n_2, n_3, n_4는 각 Section의 Line 개수를 나타낸다.

각 Section에 대한 자세한 설명은 아래와 같다.

가. Start Section

(1) 용도: file의 내용에 대한 임의의 주석을 기록하는 부분이다.

(2) 내용: file 명, 대상 데이터 명, 작성자, 작성일시 등 필요하다고 생각되는 임의의 내용을 기록할 수 있다.

나. Global Section

(1) 용도: IGES file을 만든 Software 환경에 대한 정보를 기록하는 부분이다.

(2) 내용: IGES version 번호, Preprocessor version 번호, file 명,

작성자, 작성일시, 정수표현 bit수, 단정도 실수 배정도 실수의 지수의 최대값 및 유효자리수, 측정 단위, 최대의 선폭 등 총 24개의 데이터를 기록한다.

다. Directory Entry Section

(1) 용도: file에 기록되어 있는 모든 형상·비형상 Entity에 대한 속성정보를 기록하 는 부분으로, Entity들에 대한 색인의 역할을 한다.

(2) 내용: Entity Type number, PD Section에 대한 포인터, 선의 종류, 선폭, 색상 등 총 20개의 데이터를 기록한다.

1 8	9 16	17 24	25 32	33 40	41 48	49 56	57 64	65 72	73 80
Entity Type Number #	Parameter Data	Structure #,	Line Font Pattern #,	Level #,	View 0,	Transform Matrix 0,	Label Display Assoc. 0,	Status Number #	Sequence Number D#
Entity Type Number #	Line Weight Number #	Color Number #,	Parameter Line Count #	Form Number #	사용 않음	사용 않음	Entity Label	Entity Subscript Number #	Sequence Number D#+1

#: 정수 : PD 포인터 : DE 포인터
0, : 영 또는 포인터 #, : 정수 또는 포인터 : 구현시 관심대상

라. Parameter Data Section

(1) 용도: DE Section에서 정의된 Entity들에 대한 실제 데이터를 기록하는 부분이다.

(2) 내용: Entity들의 실제 데이터를 기록한다. 즉 3차원 좌표값, 파라미터값, DE Section을 참조하는 포인터 등을 기록한다. 그리고 어떤 Entity들은 Associativity Entity 및 Property Entity에 대한 포인터를 기록하기도 한다. Associativity Entity는 여러 Entity들이 논리적으로 연관관계를 가지는 경우에 그 관계를 정의하는 Entity이고, Property Entity는 Entity와 관련된 비형상 수치정보 및 문자정보를 정의하는 Entity이다.

마. Terminate Section

(1) 용도: 위의 4개의 Section에 사용된 line의 수를 Section별로 기록한다.

(2) 내용: 각 Section별로 Section 구분문자와 총 line의 수를 기록한다.

바. Entity 종류에 따른 PD Section의 구성

(1) Composite Curve Entity(# 102)

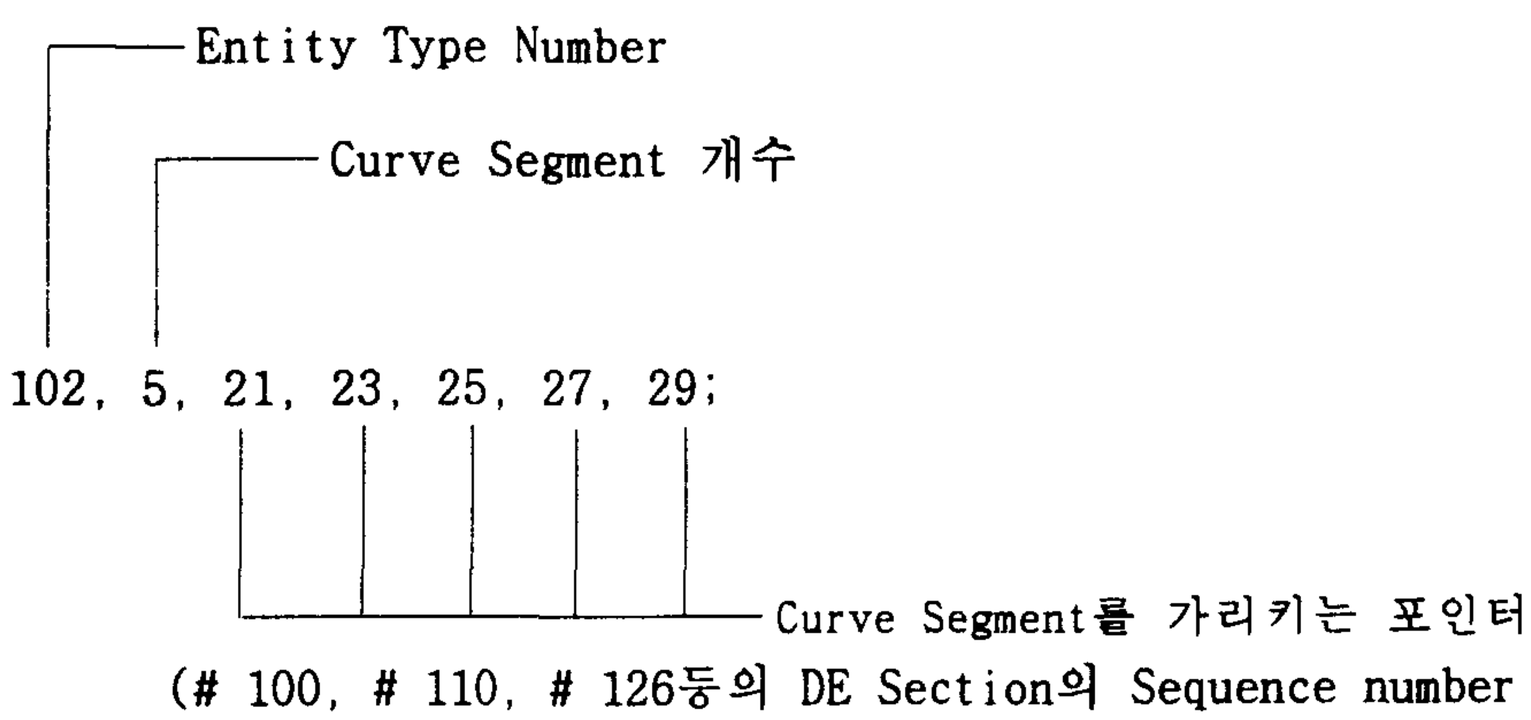
Curve segment의 조합으로 복잡한 모양의 curve를 표현하기 위한 Entity이다. 사용되는 curve segment에는 Circular Arc(# 100), Line(# 110), B-spline curve(# 126)등이 있다.

① 구성 데이터

Curve segment의 개수와 각 segment에 대한 포인터

순서	이름	수형	의미
1	N	정수	Curve segment의 개수
2	DE1	포인터	DE Section의 sequence #
.	.	.	.
.	.	.	.
.	.	.	.
N+1	DEN	포인터	DE Section의 sequence #

② 예



※ # 100은 Circular Arc Entity

110은 Line Entity

126은 Rational B-spline Curve Entity

(2) Rational B-Spline Curve Entity(# 126)

① 곡선식

$$R(t) = \frac{\sum_{I=0}^{k} N_{M+1,I}(t) W_I V_I}{\sum_{I=0}^{k} N_{M+1,I}(t) W_I}$$

M = Degree of basis function

② 구성 데이터

순서	이름	수형	의미	비고
1	K	정수	Upper index of sum	①곡선식 참조
2	M	정수	Degree of basis function	①곡선식 참조
3	PROP1	정수	0: Non-Planar, 1: Planar	
4	PROP2	정수	0: Open Curve, 1: Closed Curve	
5	PROP3	정수	0: Rational Curve 1: polynomial	
6	PROP4	정수	0: Non-Periodic 1: periodic	
7 7+A	T(-M) T(N+M)	실수	Knot Sequence	N=K-M+1 A=N+2M (K+M+2)개
8+A 8+A+K	W(0) W(K)	실수	Weights	(K+1)개 ①곡선식 참조
9+A+K 11+A+4K	V_i	실수	Control Points	3(K+1)개 ①곡선식 참조
12+A+4K 13+A+4K	V(0) V(1)	실수	Starting Parameter Value Ending Parameter Value	
14+A+4K	XNORM	실수	Unit Normal(X value)	PROP1 = 1 일 때만 해당됨
15+A+4K	YNORM	실수	Unit Normal(Y value)	
16+A+4K	ZNORM	실수	Unit Normal(Z value)	

(3) Rational B-Spline Curve Entity(# 126)

① 곡면식

$$R(t) = \frac{\sum_{I=0}^{k1}\sum_{J=0}^{k2} N_{M1+1,I}(u)\,N_{M2+1,J}(v)\,W_{JI}V_{IJ}}{\sum_{I=0}^{k1}\sum_{J=0}^{k2} N_{M1+1,I}(u)\,N_{M2+1,J}(v)\,W_{JI}}$$

M = Degree of basis function

② 구성 데이터

순서	이름	수형	의미	비고
1	K1	정수	Upper index of first sum	①곡면식 참조
2	K2	정수	Upper index of second sum	①곡면식 참조
3	M1	정수	Degree of first set of basis function	①곡면식 참조
4	M2	정수	Degree of second set of basis function	①곡면식 참조
5	PROP1	정수	0: Not Closed 1: Closed in first parametric variable direction	
6	PROP2	정수	0: Not Closed 1: Closed in second parametric variable direction	
7	PROP3	정수	0: Rational 1: polynomial	
8	PROP4	정수	0: Non-Periodic in first parametric variable direction 1: periodic in first parametric variable direction	
9	PROP5	정수	0: Non-Periodic in second parametric variable direction 1: periodic in second parametric variable direction	
10 10+A	S(-M1) S(N1+M1)	실수	First Knot Sequence	N1=K1-M1+1 A=N1+2M1 (K1+M1+2)개
11+A 11+A+B	T(-M2) T(N2+M2)	실수	Second Knot Sequence	N2=K2-M2+1 B=N2+2M2 (K2+M2+2)개
12+A+B 13+A+B … 11+A+B+C	W(0,0) W(1,0) … W(K1,K2)	실수	Weights	C=(K1+1)×(K2+1)개 ①곡면식 참조
12+A+B+C 11+A+B+4C	V_{ij}	실수	Control Points	3C개 ①곡면식 참조
12+A+B+4C	U(0)	실수	Starting Value for first parametric direction	
13+A+B+4C	U(1)	실수	Ending Value for first parametric direction	
14+A+B+4C	V(0)	실수	Starting Value for second parametric direction	
15+A+B+4C	V(1)	실수	Ending Value for second parametric direction	

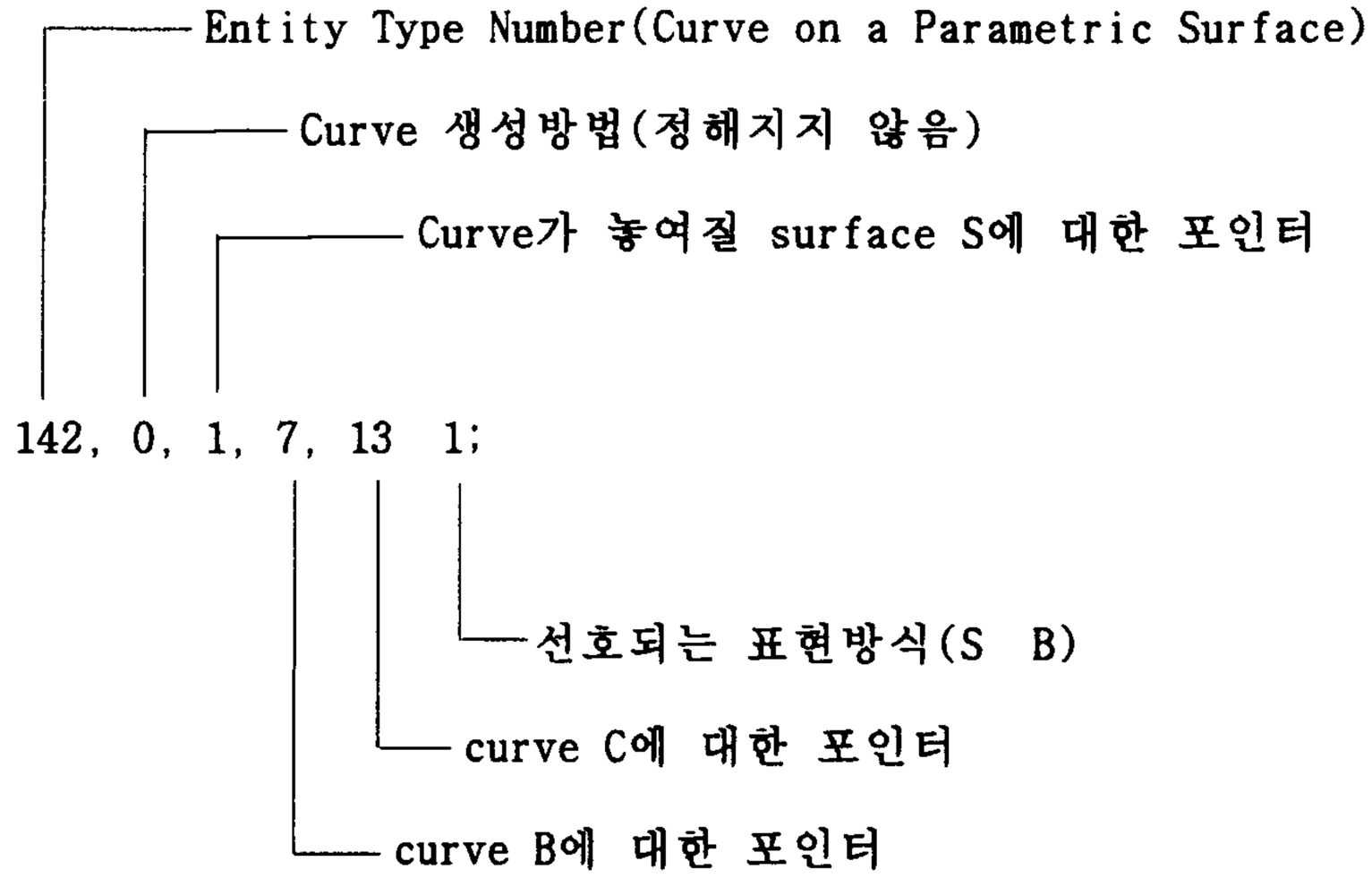
(4) Curve on a Parametric Surface Entity(# 142)

① 구성 데이터

Surface S에 대한 포인터, t-축에서 u-v domain상으로의 curve B에 대한 포인터, t-축에서 surface S상으로의 curve C에 대한 포인터, surface상에서 curve가 생성된 방법 및 보내는 쪽의 system이 선호하는 curve의 표현 방식

순서	이름	수형	의미
1	CRTN	정수	Surface상에서 curve가 생성된 방법 0: 결정되지 않음 1: Projection 2: Intersection 3: Isoparametric Curve
2	SPTR	정수	Curve가 놓여지는 surface S에 대한 포인터, S=S(u, v)
3	BPTR	포인터	Domain curve B에 대한 포인터 B = B(t) = (u(t),v(t))
4	CPTR	포인터	3D curve C에 대한 포인터 C = C(t) = (x(t),y(t),z(t))
5	PREF	정수	보내는 쪽의 시스템이 선호하는 표현방식 0: 결정되지 않음 1: S B (surface S와 curve B의 조합) 2: C (mapping C and indicate that on S) 3: C와 S B가 동일하게 선호됨

② 예



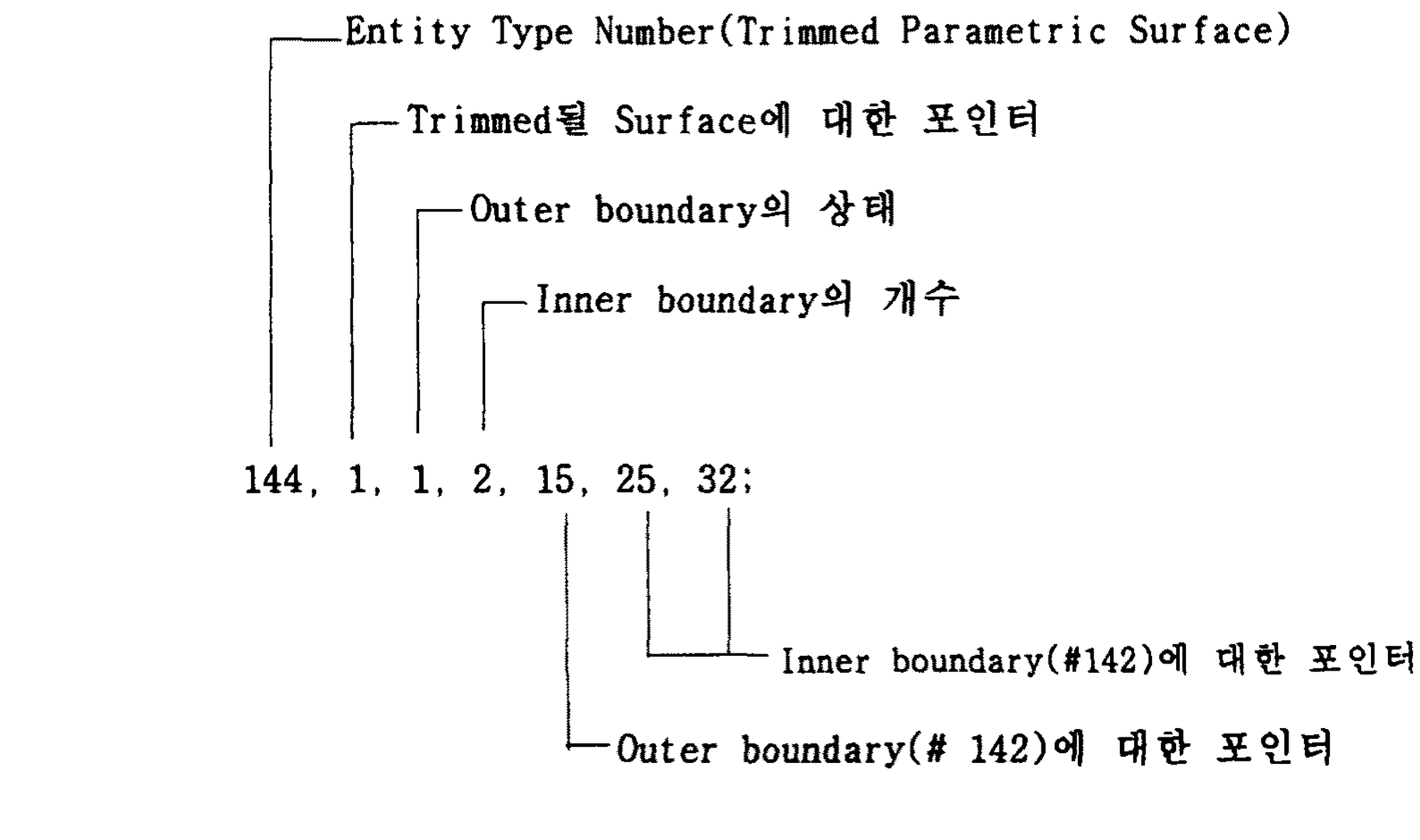
(5) Trimmed(Parametric) Surface Entity(# 144)

① 구성 데이터

Trimmed될 surface에 대한 포인터, outer boundary curve에 대한 포인터, inner boundary curve에 대한 포인터

순서	이름	수형	의미
1	PTS	포인터	Trimmed될 surface entity에 대한 포인터
2	N1	정수	0: If the outer boundary is the boundary of D(u-v domain), 즉 untrimmed 경우 1: Otherwise
3	N2	정수	Inner boundary를 구성하는 simple closed curve의 개수
3+N1	PTO1	포인터	Outer boundary를 구성하는 simple closed curve(# 142)에 대한 포인터
4+N1 3+N1+N2	PTI1 PTIN2	포인터	Inner boundary를 구성하는 simple closed curve(# 142)에 대한 포인터(N2 개)

② 예



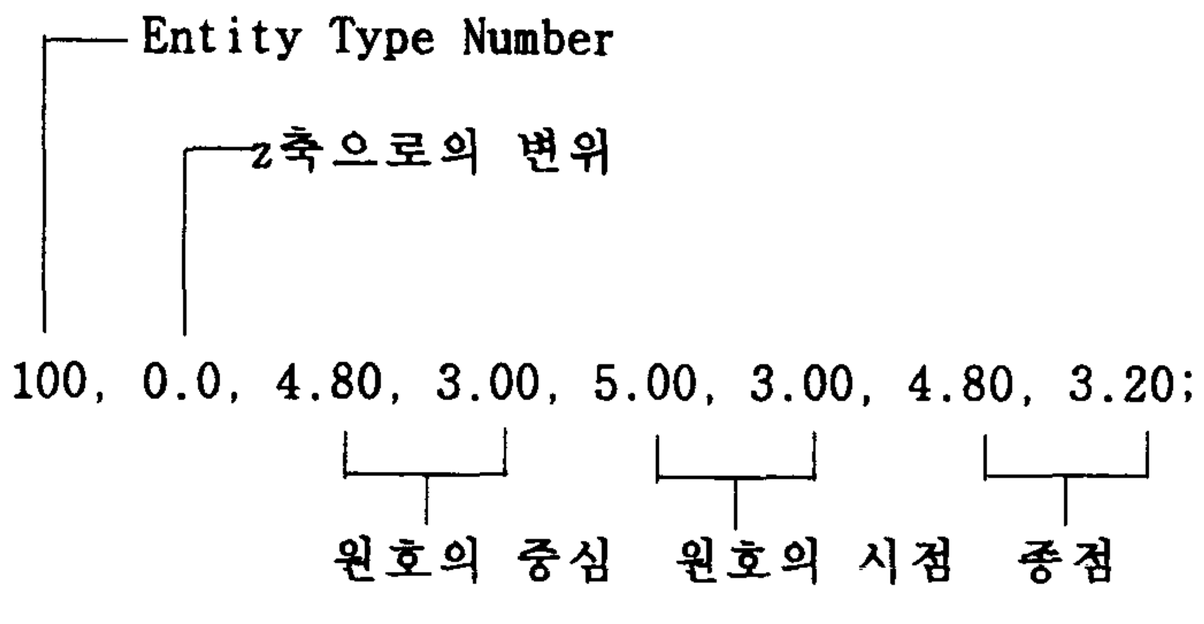
(6) Circular Arc Entity(# 100)

원호를 표현하기 위한 Entity이다. 이 Entity의 경우 xy 평면에 평행하면서 z축으로의 변위를 가진 원호를 표현한다. 이러한 원호가 3차원 공간상의 임의의 위치에 놓이기 위 해서는 좌표의 변환이 필요하다. 흔히 이 Entity의 DE field에서 Transformation Matrix Entity(# 124)를 가리키는 포인터를 보게 되는데, 이는 바로 circular arc의 좌표변환을 위한 것이다.

① 구성 데이터

xy-평면으로부터 z축으로의 변위, 원호의 중심, 원호의 시점 및 종점의 좌표

② 예



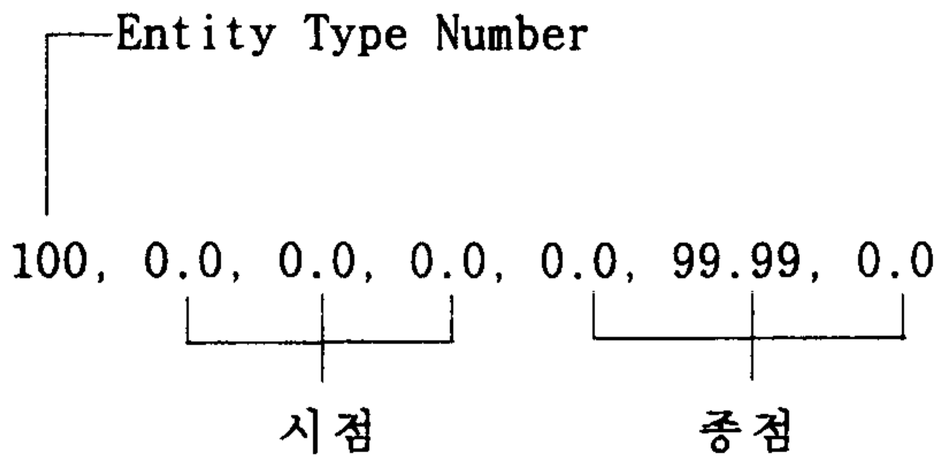
(7) Line Entity(# 110)

3차원 공간상의 두 점을 잇는 선분을 정의하기 위한 Entity이다.

① 구성 데이터

시점과 종점의 좌표

② 예



(8) Transform Matrix Entity(# 124)

좌표변환이 필요한 경우에 이용할 변환행렬의 정의를 위한 Entity이다.

① 용도: 회전이동과 평행이동의 변환요소를 정의

② 구성 데이터: 변환행렬의 요소들

순서	이름	수형	설명
1	R11	실수	첫 번째 행
2	R12		
3	R13		
4	T1		
5	R21	실수	두 번째 행
6	R22		
7	R23		
8	T2		
9	R31	실수	세 번째 행
10	R32		
11	R33		
12	T3		

③ 구성요소의 의미

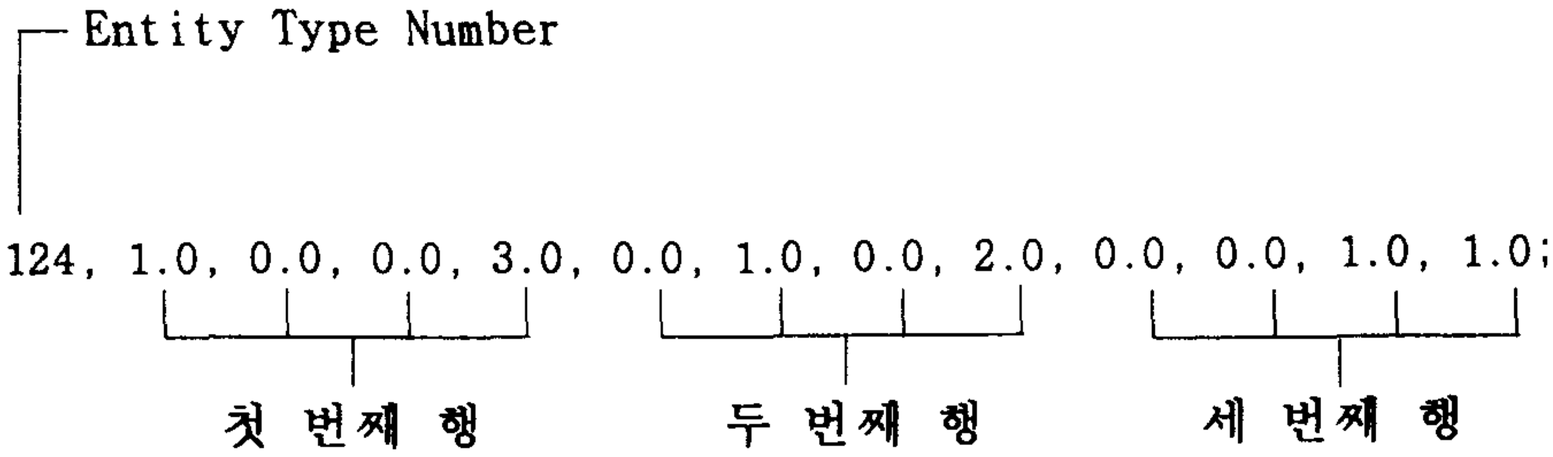
위 표의 순서대로 행렬의 원소가 기록되면, 다음과 같은 회전이동행렬 R과 평행이동 행렬 T가 정의된다.

$$R = \begin{vmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{vmatrix} \quad , \quad T = \begin{vmatrix} T_1 \\ T_2 \\ T_3 \end{vmatrix}$$

만약 Transformation Matrix를 참조하는 Entity의 3차원 좌표 값이 (X_{output} , Y_{output} , Z_{output})라면 변환 결과는 다음과 같다.

$$\begin{vmatrix} X_{output} \\ Y_{output} \\ Z_{output} \end{vmatrix} = \begin{vmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{vmatrix} \begin{vmatrix} X_{output} \\ Y_{output} \\ Z_{output} \end{vmatrix} + \begin{vmatrix} T_1 \\ T_2 \\ T_3 \end{vmatrix}$$

④ 예


Entity Type Number
124, 1.0, 0.0, 0.0, 3.0, 0.0, 1.0, 0.0, 2.0, 0.0, 0.0, 1.0, 1.0;
첫 번째 행
두 번째 행
세 번째 행

2. IGES Interface의 구조

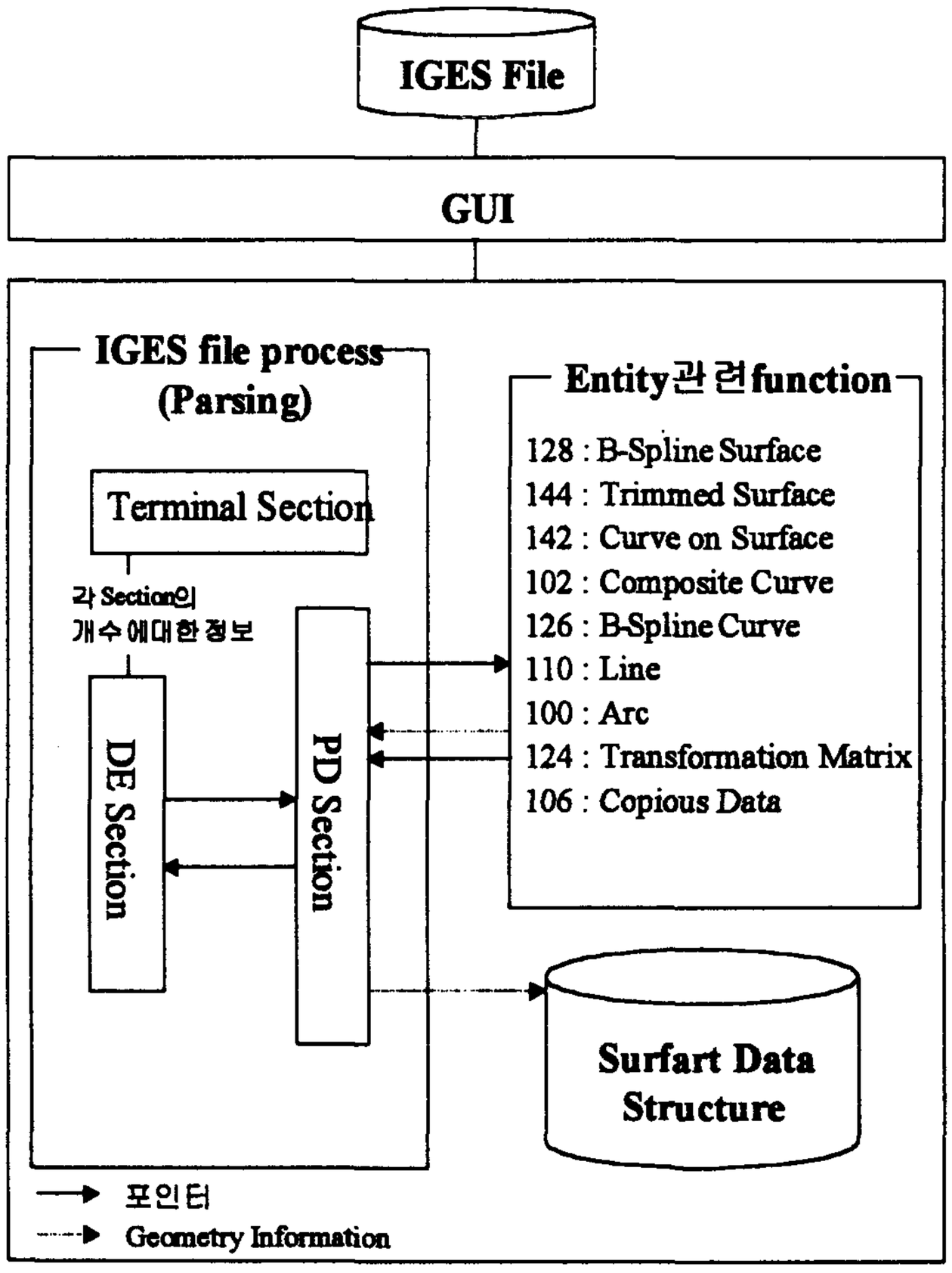
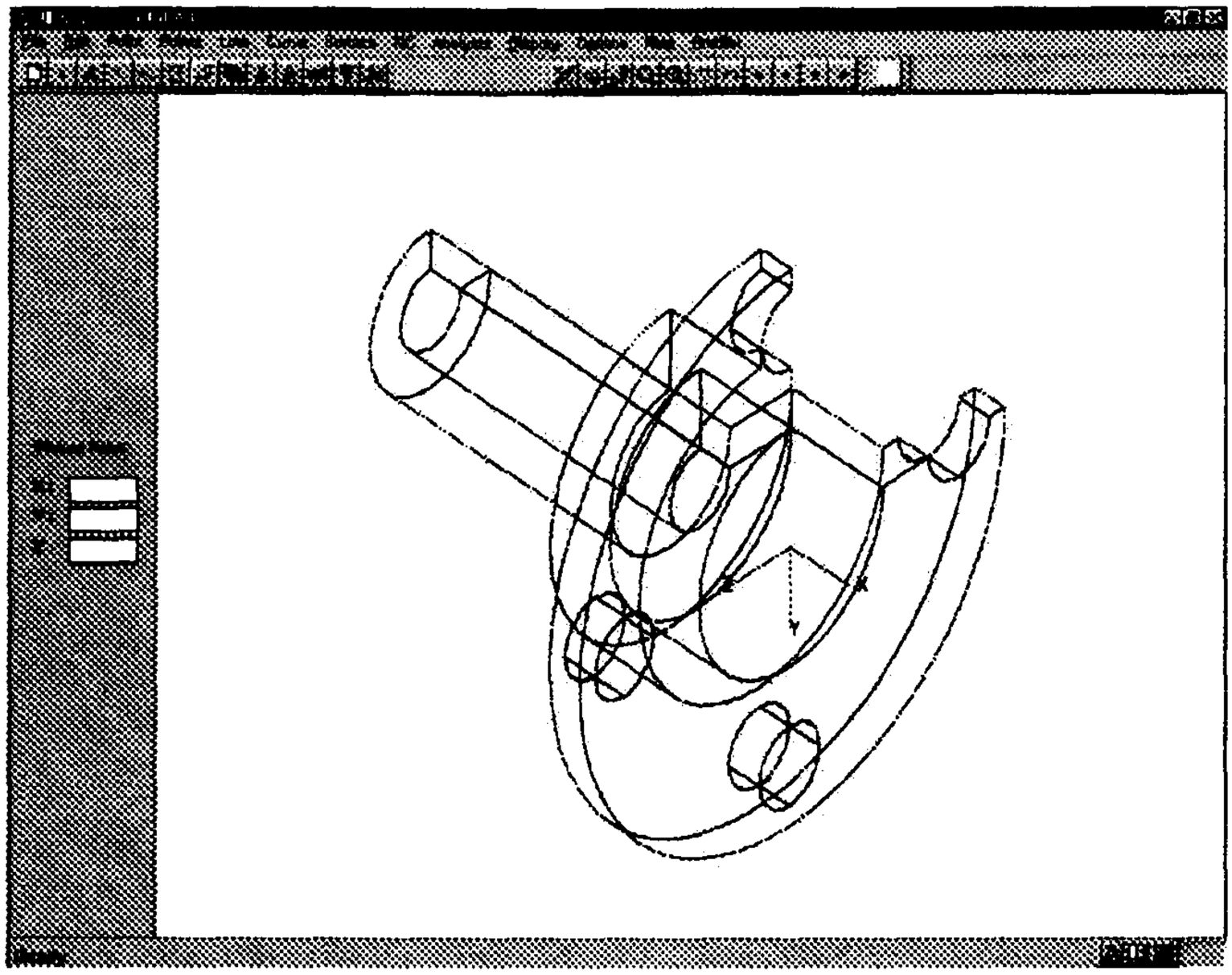


그림7.1 IGES Interface 구조

3. 적용 사례

아래에 있는 그림은 Pro-Engineer에서 모델링된 것으로 SurfART에서 IGES import한 예들이다.



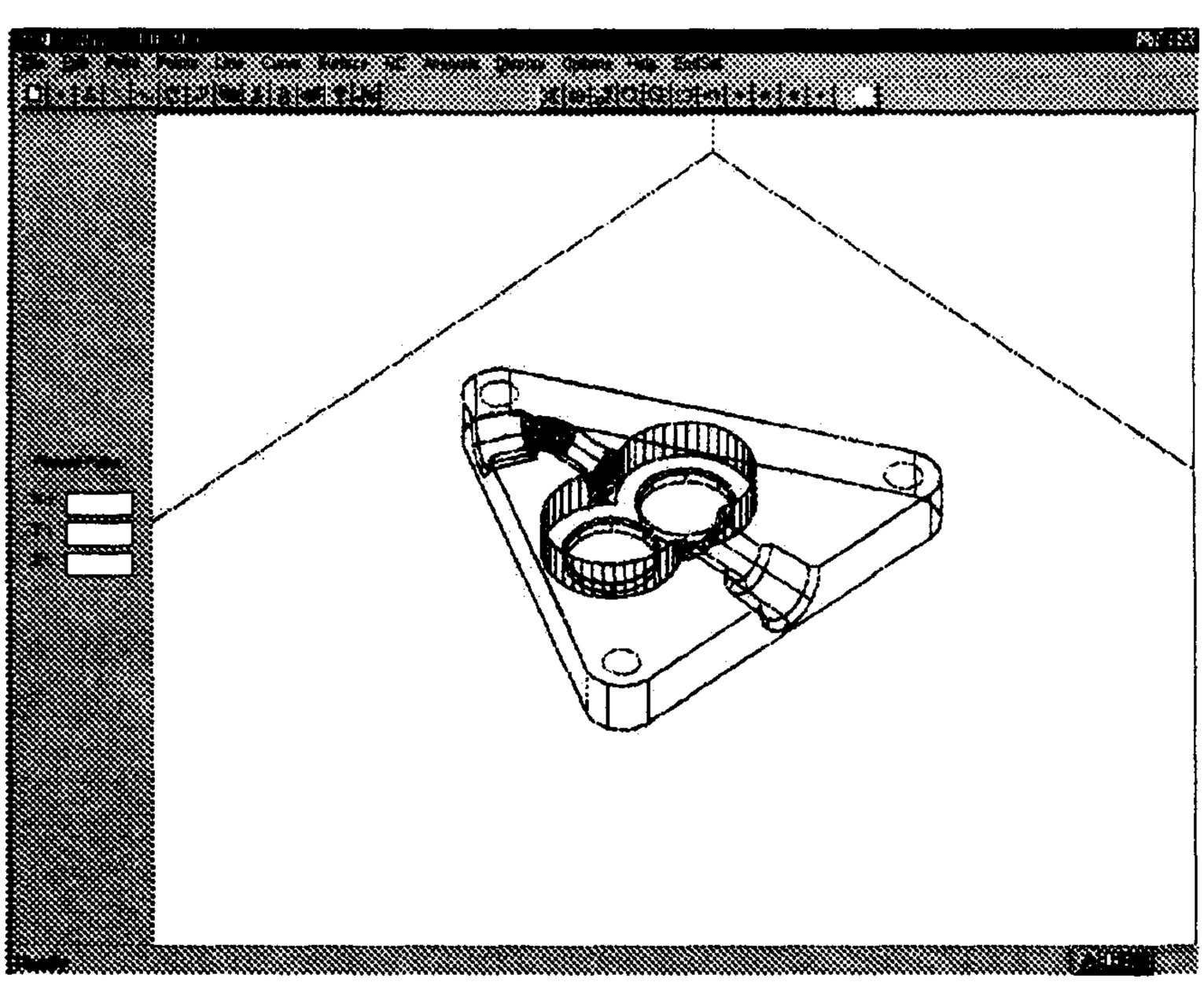


그림7.2 SurfART에서 IGES import

제 2 절 ZES Interface

1. ZES 파일 형식(version 2.2)

가. ZES 파일 개요

CATIA CAD/CAM 시스템의 한 모델 중 NC가공에 필요한 정보만을 추출하여 타 시스템과 데이터를 주고받기 위한 간이 정보 파일

나. ZES가 포함하는 ELEMENT

ZES는 다음과 같은 CATIA의 Space Geometric Elements만을 포함한다.

CATIA Elements		ZES Elements
A. POLYNOMIAL SURFACE	[Type = (5,1)]	(5,0)
B. FACE	[Type = (6,1)]	(6,4), (6,5)
Supporting Polynomial Surface	[Type = (5,1)]	(5,0)
Supporting Plane	[Type = (4,1)]	(4,0)
Supporting NURBS Surface	[Type = (47,1)]*	(5,0) - ①
Bounding Edge	[Type = (12,1)]	(12,0)
C. POLYNOMIAL CURVE	[Type = (3,1)]	(3,0)
D. NURBS CURVE	[Type = (46,1)]*	(3,0) - ②
E. NURBS SURFACE	[Type = (47,1)]*	(5,0) - ①
F. EXACT SOLID	[Type = (17,1)]*	(6,4),(6,5) - ③
G. SKIN(위상학적 정보를 갖는 FACE들의 집합)	[Type = (13,1)]	(6,4),(6.5) - ③

(비고) · ZES Version 2.0에 새롭게 추가 되었음

· ZES Version 2.2에 새롭게 추가 되었음

① POLYNOMIAL SURFACE로 근사

② POLYNOMIAL CURVE로 근사

③ FACE로 분해함

다. 파일 구조

ZES파일은 크게 3개의 SECTION으로 구성되어 있으며 이는 다음과 같다.

A. 개시 SECTION

B. 데이터 SECTION

C. 종료 SECTION

(1) 개시 SECTION

개시 SECTION은 ZES파일의 머리 부분에 존재하는 80바이트의 3개 LINE으로 구성되어 있으며 다음과 같은 구조로 되어 있다.

A. 첫 번째 LINE

열번호	설 명	비 고
1 - 72	모델(파일) 이름	
73 - 80	개시 SECTION의 SEQUENCE 번호	S0000001

B. 두 번째 LINE

열번호	설 명	비 고
1 - 72	COMMENT	
73 - 80	개시 SECTION의 SEQUENCE 번호	S0000002

C. 세 번째 LINE

열번호	설 명	비 고
1 - 8	파일을 작성한 회사명	HMC
9 - 16	CAD/CAM 시스템 명	CATIA
17 - 24	시스템 버전	VER 3.1
25 - 32	처리한 날짜	MM/DD/YY
33 - 40	처리한 시간	HH:MM:SS
41 - 72	사용 안함	
73 - 80	개시 SECTION의 SEQUENCE 번호	S0000003

(2) 종료 SECTION

종료 SECTION은 ZES파일의 끝부분에 존재하는 80바이트의 1개 LINE으로 구성되어 있으며 다음과 같은 구조로 되어 있다.

열번호	설 명	비 고	
1 - 8	-999	I8	
9 - 16	FACE의 개수	I8	
17 - 24	독립적인 Polynomial SURFACE의 개수	I8	
25 - 32	`FILE END'	A8	
33 - 40	Polynomial CURVE의 개수	I8	
41 - 48	NURBS CURVE의 개수	I8	*2
49 - 56	FILE의 총 LINE수	I8	
57 - 64	독립적인 NURBS SURFACE의 개수	I8	*2
65 - 72	EXACT SOLID의 개수	I8	*2
73 - 80	종료 SECTION의 SEQUENCE 번호	E0000001	

(참고) 1. 이 SECTION은 데이터 SECTION의 ID-LINE과 동일한

READ/WRITE FORMAT임

2. ZES Version 1.2로부터 추가되었음. (ver1.2에서는 `사용

안함')이었음

NURBS CURVE의 개수 추가

NURBS SURFACE의 개수 추가

EXACT SOLID의 개수 추가

3. FACE, 독립적 Poly_SURFACE, Poly_CURVE의 개수는 ZES로 출

력되는 요소의 개수임(즉 CATIA원래 요소 + 변환된 요소).

4. NURBS CURVE/SURFACE와 EXACT SOLID의 개수는 원래 CATIA

모델에 포함된 개수로 ZES file에는 위의 요소들로 변환됨.

(3) 데이터 SECTION

데이터 SECTION은 각 ELEMENT들의 수치 데이터를 가지는 SECTION으로 이는 1개의 ID-LINE과 나머지의 데이터 BLOCK으로 구성되어 있다. ID-LINE은 ELEMENT TYPE, 간단한 ATTRIBUTE, 데이터 BLOCK의 길이 등의 정보를 가지고 있으며 다음과 같이 구성되어 있다.

A. ID-LINE

열번호	설 명	비 고
1 - 8	ELEMENT TYPE	I8
9 - 16	ELEMENT SUB-TYPE	I8
17 - 24	ELEMENT의 ID-번호	I8
25 - 32	ELEMENT의 ID-STRING	A8
33 - 40	COLOR	I8
41 - 48	SHOW(1) or NOSHOW(0)	I8
49 - 56	데이터 BLOCK의 길이	I8
57 - 72	사용 안함	
73 - 80	데이터 LINE의 SEQUENCE 번호	I0000001

** 각 항의 설명

· ELEMENT TYPE : ELEMENT의 TYPE을 나타내는 정보로 다음과 같다.

· ELEMENT SUB-TYPE : ELEMENT의 2차 TYPE을 나타낸다.

Geometry Element의 Type과 Sub-Type은 다음과 같다.

Elements	Type	Sub_type	비 고
POLYNOMIAL CURVE	3	0	
PLANE	4	0	
POLYNOMIAL SURFACE	5	0	
FACE	6	4	Plane supporting
		5	Surface supporting
EDGE	12	0	

· ELEMENT ID 번호 : 각 ELEMENT의 고유한 ID-번호이다.

· ELEMENT ID-STRING: 각 ELEMENT의 고유한 ID-STRING이다.

· COLOR : 각 ELEMENT의 색깔

· SHOW OR NOSHOW : ELEMENT가 화면에 나타나 있는 경우 1,
아니면 0 이다.
FACE의 SUPPORTING ELEMENT와 EDGE는
모두 0으로 되어 있다.

· 데이터 BLOCK 길이: 다음에 나오는 데이터 BLOCK의 길이를 나타낸다. 이 수는 ID-LINE을 포함하지 않는 데이터 BLOCK만의 길이이다.

B. 데이터 BLOCK : 다음 장에서 자세히 설명한다.

라. 각 ELEMENT별 데이터 SECTION의 데이터 BLOCK의 구조

데이터 BLOCK의 한 LINE은 두 종류가 있다. 그 하나는 9개 이내의 정수(INTEGER)가 각각 8 BYTE(I8) 영역에 순차적으로 수록되어 있는 것이며, 다른 하나는 3개 이내의 실수(REAL)가 각각 24 BYTE(E24.16) 영역에 수록되어 있다. 끝부분 73-80 COLUMN은 SEQUENCE 번호를 수록하는 영역이다.

TYPE 1 :

1-8	9-16	17-24	25-32	33-40	41-48	49-56	57-64	65-72	73-80
I8	I8	I8	I8	I8	I8	I8	I8	I8	Dnnnnnnn

TYPE 2 :

1 - 24	25 - 48	49 - 72	73 - 80
E24.16	E24.16	E24.16	Dnnnnnnn

(1) PLANE : TYPE(4), SUB-TYPE(1)

PLANE은 독립된 ELEMENT로는 존재하지 않으며 FACE의 SUPPORTING ELEMENT로 존재한다. 데이터 BLOCK의 구조는 다음과 같다.

1	PTX	PTY	PTZ	D0000001
2	V1X	V1X	V1X	D0000002
3	V2Y	V2Y	V2Y	D0000003

** 설 명 **

· (PTX, PTY, PTZ) : PLANE상의 원점

· (V1X, V1Y, V1Z) : 이 PLANE의 원점에 좌표축을 정의했을 때
X-축의 방향 VECTOR

· (V2X, V2Y, V2Z) : Y-축의 방향 벡터

(2) POLYNOMIAL SURFACE : TYPE(5), SUB-TYPE(1)

POLYNOMIAL SURFACE는 독립적으로 존재할 수도 있고 FACE의 SUPPORTING ELEMENT로도 존재할 수 있다. FACE의 SUPPORTING ELEMENT로 존재할 경우는 ID-LINE의 SHOW NOSHOW ATTRIBUTE는 0으로 되어 있다. 한 SURFACE상에 두개 이상의 FACE가 존재할 경우 각 FACE마다 SURFACE의 정보를 출력하므로 같은 ID의 동일한 SURFACE가 파일 내에 여러 개 수록되어 있을 수도 있다. 데이터 BLOCK은 다음과 같다.

										73-80
1	NI	NJ								D0000001
2	XMIN			YMIN			ZMIN			D0000002
3	XMAX			YMAX			ZMAX			D0000003

4	NTAB	NDU	NDV	NPU	NPV					D0000004
5	XMIN			YMIN			ZMIN			D0000005
6	XMAX			YMAX			ZMAX			D0000006
7	CX(1)			CY(1)			CZ(1)			D0000007
8	CX(2)			CY(2)			CZ(2)			D0000008
9	CX(3)			CY(3)			CZ(3)			D0000009

. . . . . .

32	NTAB	NDU	NDV	NPU	NPV					D0000032
33	XMIN			YMIN			ZMIN			D0000033
34	XMAX			YMAX			ZMAX			D0000034
35	CX(1)			CY(1)			CZ(1)			D0000035
36	CX(2)			CY(2)			CZ(2)			D0000036

** 설 명 **

· NI　　:　U방향 PATCH 수

· NJ　　:　V방향 PATCH 수

· XMIN/XMAX, YMIN/YMAX, ZMIN/ZMAX　:　SURFACE의 최소/최대값

· NTAB　:　한 PATCH의 TABLE길이 = 3 + NDU X NDV

· NDU　:　1 + U방향 PATCH의 차수

· NDV　:　1 + V방향 PATCH의 차수

· NPU, NPV : U, V방향의 DISCRETIZATION POINT수

(화면에 SURFACE를 나타낼 때 사용되는 수치)

· CX(), CY(), CZ()　:　PATCH의 POLYNOMIAL COEFFICIENT MATRIX

각 PATCH TABLE은 다음 순서대로 배열되어 있다.

1 : I = 1　　J = 1

2 : I = 1　　J = 2

3 : I = 1　　J = 3

. . . . . .

N : I = NI　J = NJ

예를 들어 한 SURFACE가 U방향으로 1 PATCH, V방향으로 2 PATCH, 그리고 첫 번째 PATCH가 U방향으로 4차, V방향으로 2차라면

NI = 1,　　NJ = 2

NDU = 5,　　NDV = 3

NTAB = 3 + NDU X NDV = 18

그리고 CX(), CY(), CZ()의 길이는 15이며, X값은 다음 식으로 구한다.

$$X = CX(1) + CX(2)u + \ldots + CX(5)u^4 + CX(6)v + CX(7)uv + \ldots + CX(10)u^4v + CX(11)v^2 + CX(12)uv^2 + \ldots + CX(15)u^4v^2$$

(3) FACE : TYPE(6), SUB-TYPE(1)

FACE는 PLANE이나 POLYNOMIAL SURFACE, NURBS SURFACE의 부분으로서 이 부분은 EDGE로 둘러 쌓여 있다.

73 - 80

1	NDOM	NOUT	NIN1	NIN2	NIN3	. .	. . .	. . .	NIN7	D0000001
2	NIN8	. . .	. .							D0000002

** 설 명 **

· NDOM : DOMAIN의 개수

· NOUT : 바깥 DOMAIN의 EDGE개수

· NINi : i 번째 안쪽 DOMAIN의 EDGE 개수

위 BLOCK 다음에는 SUPPORTING ELEMENT의 데이터 BLOCK이 출력되고 그 다음에 각 EDGE들의 데이터 BLOCK이 출력된다. 이때 각 EDGE는 바깥쪽 DOMAIN의 EDGE 데이터가 먼저 출력되고 그 다음 안쪽 DOMAIN의 EDGE가 순서대로 출력된다. 즉 예를 들어 한 FACE가

NDOM = 3, NOUT = 5, NIN1 = 3, NIN2 = 1

라면 출력되는 전체 FACE 데이터 BLOCK은

1 : FACE ID-LINE

2 : FACE 데이터 BLOCK(위의 모양)

3 : BASE ELEMENT의 BLOCK (SURFACE)나 PLANE의 ID-LINE과 DATA BLOCK

4 : 5개의 바깥쪽 EDGE BLOCK

5 : 3개의 안쪽 EDGE BLOCK

6 : 1개의 안쪽 EDGE BLOCK

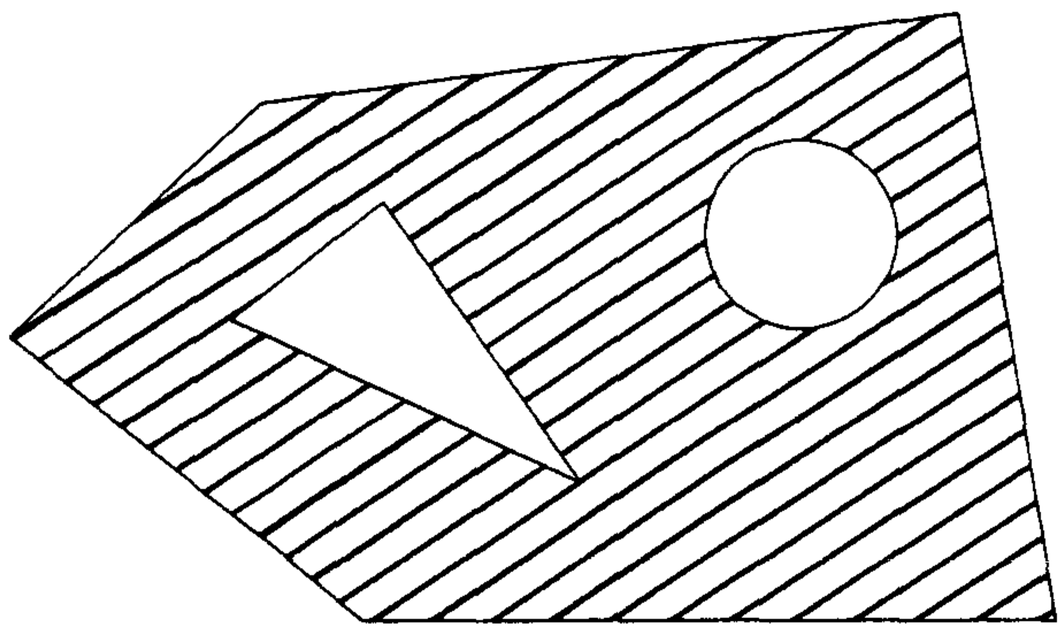


그림7.3 FACE element

(4) EDGE-TYPE(12), SUB-TYPE(1)

EDGE는 FACE의 BOUNDARY를 형성하는 일종의 CURVE로서 SURFACE나 PLANE의 PARAMETER U, V를 구하기 위한 POLYNOMIAL u = FU(w), v = FV(w)로 표현된다. 한 SUPPORTING ELEMENT상에 FACE가 여러 개 존재하고 이 중 두 개의 FACE가 한 EDGE를 공유할 경우에는 같은 ID의 동일한 EDGE가 각 FACE마다 출력되므로 한 파일 내에 동일한 EDGE정보가 한 개 이상 수록되어 있을 수도 있다. 데이터 BLOCK은 다음과 같다.

										73 - 80
1	NARC									D0000001
2	W1			W2						D0000002
3	NTAB	NDW	IU	IV	NP					D0000003
4	CU(1)			CV(1)						D0000004
5	CU(2)			CV(2)						D0000005

. . . . . .

21	NTAB	NDW	IU	IV	NPW					D0000021
22	CU(1)			CV(1)						D0000022
23	CU(2)			CV(2)						D0000023

. . . . . .

. . . . . .

** 설 명 **

- NARC : 전체의 ARC 수
- W_1, W_2 : 첫 번째와 마지막 ARC에서의 LIMIT PARAMETER값
- NTAB : 한 ARC의 TABLE길이 = 1 + NDW
- NDW : 1 + ARC차수
- IU, IV : 이 ARC가 존재하는 SURFACE상의 PATCH 번호
- NP : DISCRETIZATION POINT수
- CU(), CV() : U와 V의 POLYNOMIAL COEFFICIENT VECTOR

예를 들어 한 EDGE가 SUPPORTING SURFACE의 PATCH 번호 I = 1, J = 2에 있고 2개의 ARC로 구성되어 있으며 차수가 5일 경우

NARC = 2

W_1 = 0.2, W_2 = 0.8

NDW = 1 + 5 = 6

NTAB = 1 + NDW = 7

IU = 1, IV = 2

가된다. 또 U와 V는 다음과 같이 계산된다.

$$u = CU(1) + CU(2)w + CU(3)w^2 + \ldots + CU(6)w^5$$

$$v = CV(1) + CV(2)w + CV(3)w^2 + \ldots + CV(6)w^5$$

SUPPORTING ELEMENT가 PLANE일 경우에는 U, V값은 PLANE의 상대 좌표계상의 좌표 값이 된다. 즉

$$\begin{vmatrix} x' \\ y' \\ z' \end{vmatrix} = \begin{vmatrix} u \\ v \\ 0 \end{vmatrix}$$

PLANE의 정보, (PTx, PTy, PTz), (V1x, V1y, V1z), (V2x, V2y, V2z)가 주어졌을 때 상대 좌표축 상의한 점 P' = (x', y', z') = (u, v, 0)을 절대 좌표 축좌표로 환산하는 식은

$$P = T \times P'$$

$$\begin{vmatrix} x \\ y \\ z \\ 1 \end{vmatrix} = \begin{vmatrix} V_{1x} & V_{2x} & V_{3x} & PT_x \\ V_{1y} & V_{2y} & V_{3y} & PT_y \\ V_{1z} & V_{2z} & V_{3z} & PT_z \\ 0 & 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} x' \\ y' \\ 0 \\ 1 \end{vmatrix}$$

이다.

(5) POLYNOMIAL CURVE : TYPE(3), SUB-TYPE(1)

POLYNOMIAL CURVE는 3차원 공간상에 독립적으로 존재하는 기하 요소로서 데이터 BLOCK은 다음과 같다.

	1	NARC	IARC1	IARC2						D0000001
	2	W_1			W_2					D0000002
1st	3	NTAB	NDEG	NP						D0000003
arc	4	TABX(1)			TABY(2)			TABZ(1)		D0000002
		.								
		.								
		.								
	9	TABX(NDEG)			TABY(NDEG)			TABZ(NDEG)		D0000009
	10	Xmin			Ymin			Zmin		D0000010
	11	Xmax			Ymax			Zmax		D0000011
2nd	12	NTAB	NDEG	NP						D0000012
arc	13	TABX(1)			TABY(2)			TABZ(1)		D0000013
		.								
		.								
		.								
		TABX(NDEG)			TABY(NDEG)			TABZ(NDEG)		D000 . . .
		Xmin			Ymin			Zmin		D000 . . .
		Xmin			Ymax			Zmax		D000 . . .

.
.
.

** 설 명 **

· NARC : 곡선을 구성하는 ARC의 총 개수

· IARC1, IARC2 : 실제 이용되는 곡선의 시작과 끝 ARC

· W_1 , W_2 : 실제 이용되는 곡선의 시작과 끝 PARAMETER 값

(0 W_1 , W_2 1)

· NTAB : 한 ARC의 TABLE 길이 = 3 + NDEG

· NDEG : ARC의 차수 + 1

· NP : 화면에 곡선을 그릴 때 사용되는
discretization point의 개수

· TABX(), TABY(), TABZ() : 곡선 ARC 다항식의 계수

예를 들어 아래 그림과 같이 3개의 arc로 구성되어 있고 차수가 5인 경우는

NARC = 3 , IARC1 = 1, IARC2 = 2

W_1 = 0.5 , W_2 = 1

NTAB = 3 + 6 = 9

NDEG = 6

이고, 곡선 arc의 식은 아래와 같다.

$$x = TABX(1) + TABX(2)u + \ . \ . \ . \ + TABX(6)u^5$$

$$y = TABY(1) + TABY(2)u + \ . \ . \ . \ + TABY(6)u^5$$

$$z = TABZ(1) + \ TABZ(2)u + \ . \ . \ . \ + TABZ(6)u^5$$

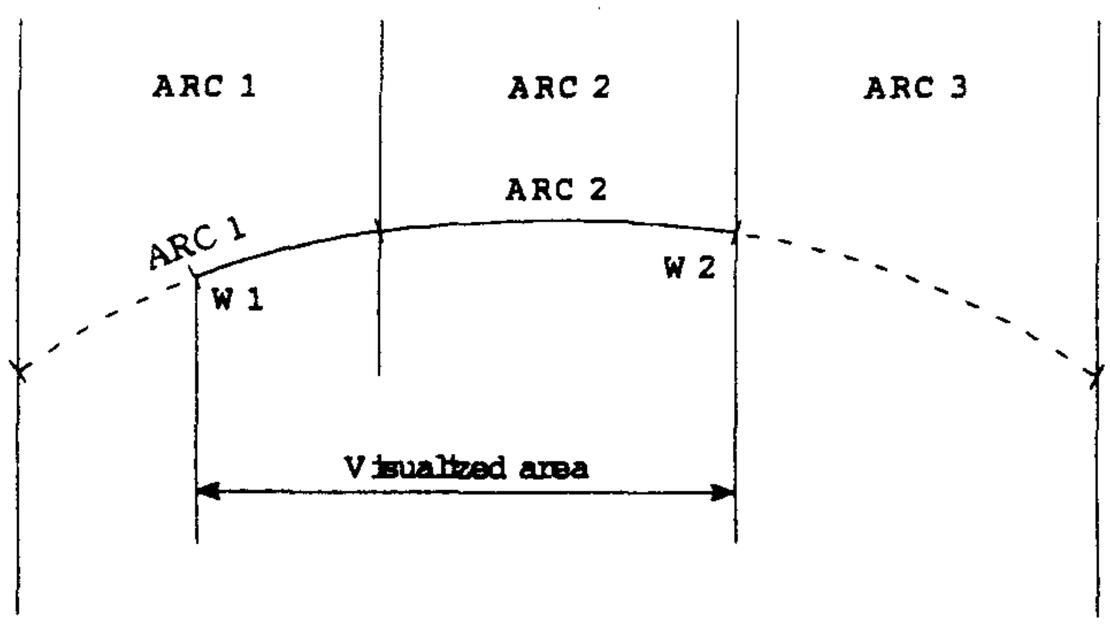


그림7.4 Curve element

마. 요약

ZES File의 전체 구조는 다음과 같다.

1	72	73	80
START SECTION	Model Name	S0000001	
	Comment	S0000002	
	Company System Version Data Time	S0000003	
DATA SECTION	EXACT SOLID ID-line	I0000001	
	EXACT SOLID Data-Block	D0000001 D000 …	
	FACE ID-line	I0000001	
	FACE Data-Block	D0000001 D000 …	
	Suppo-rting Element sub-Block	Supporting SURFACE/PLANE/ NURBS SURFACE ID-line	I0000001
		Supporting SURFACE/PLANE/ NURBS SURFACE Data-Block	D0000001 D000 …
	EDGE sub-Block	EDGE ID-line	I0000001
		EDGE Data-Block	D0000001 D000 …
		EDGE ID-line	I0000001
		EDGE Data-Block	D0000001 D000 …
		• (EDGE Blocks) •	I0000001 D0000001 D000 …
		• (FACE Blocks) •	I0000001 D0000001 D000 …
	EXACT SOLID Blocks		I000 … D000 …
	Stand alone FACE, POLY-NOMIAL CURVE/ SURFACE & NURBS Block	Stand alone FACE Blocks	I000 … D000 …
		Stand alone POLYNOMIAL SURFACE Blocks	I000 … D000 …
		Stand alone NURBS SURFACE Blocks	I000 … D000 …
		Stand alone CURVE Blocks	I000 … D000 …
		Stand alone NURBS CURVE Blocks	I000 … D000 …
END SECTION	-999	E0000001	

참고

· FACE는 EXACT SOLID의 구성 요소로 나타날 수도 있고 독립적(Stand alone)으로도 존재할 수 있다.

· 또한 SURFACE 역시 독립적으로도, FACE의 Supporting Element로도 나타날 수 있다.

2. ZES Interface 구조

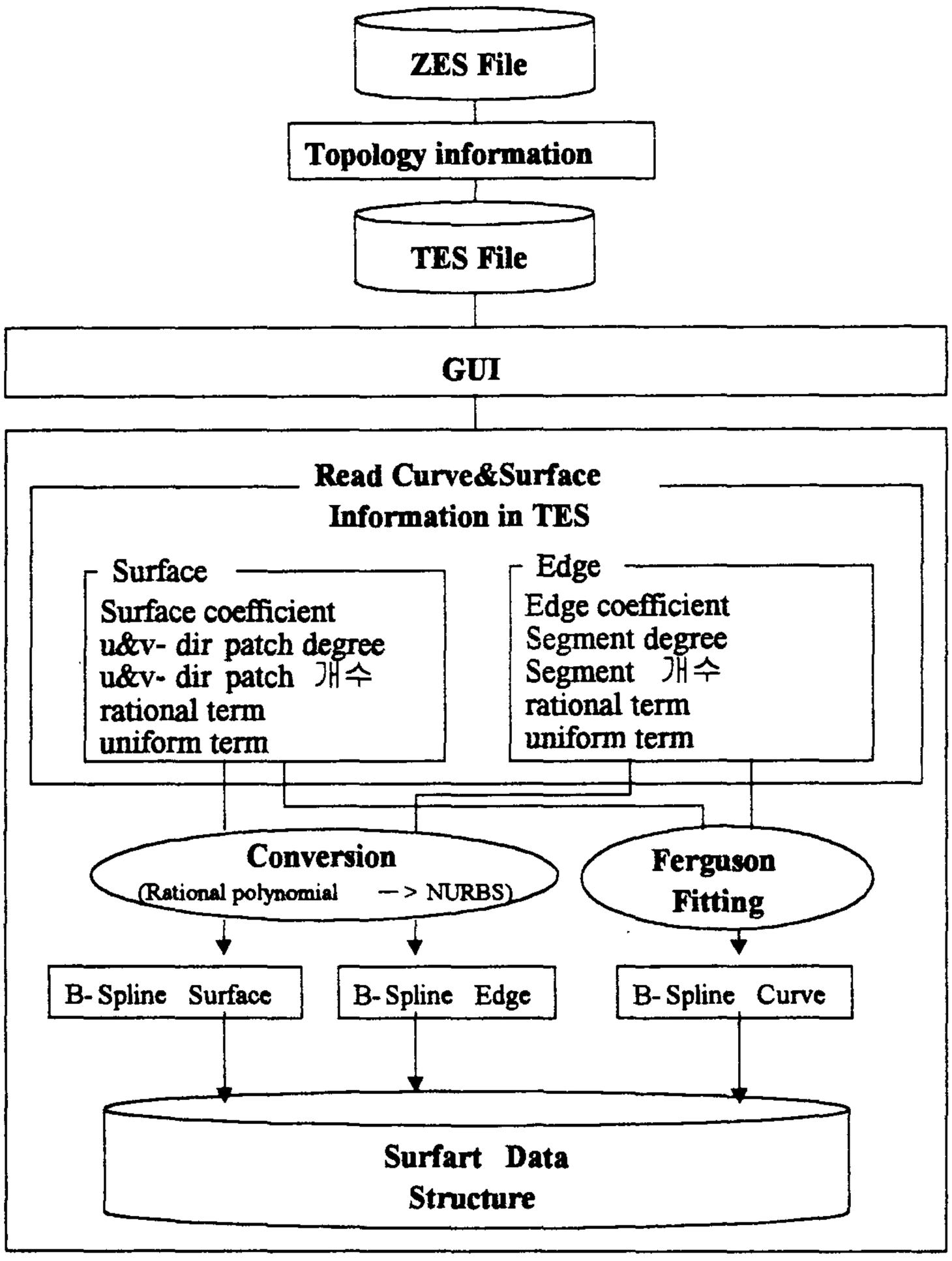


그림7.5 ZES Interface

가. Conversion

Rational Polynomial 곡선/곡면/Edge를 NURBS로 변환하는데 있어서 주의할 점은 rational polynomial에서는 곡선/edge segment와 곡면의

patch의 차수가 각각 다르게 표현될 수 있다는 것인데, NURBS 곡선/곡면/Edge는 모든 segment의 차수가 동일해야 하므로 변환과정에서 주의해야 한다.

(1) 곡선/Edge

① Control points의 계산

NURBS control point는 NURBS matrix가 3차까지 정의되어져 있기 때문에 NURBS control point와 동일하고 임의의 차수에 적용 가능한 bezier control point를 구한다.

곡선식 : $r_j(t) = UA = UMR_j$

Bezier control point : $R_j = M^{-1}A$

위 식에서 곡선/edge coefficients를 이용하여 Bezier control points를 쉽게 구해낼 수 있다. 만약 segment의 차수가 서로 다를 경우에는 segment중 가장 높은 차수를 찾아 NURBS의 차수로 정의한다. 이렇게 되면 bezier matrix의 행렬과 최고 차수보다 낮은 차수의 coefficients matrix의 행렬이 일치하지 않게 되는데 이때 coefficient matrix의 나머지 부분을 0으로 놓고 구하면 된다.

② knot의 계산

NURBS 곡선을 bezier 곡선으로 decomposition한 결과와 동일한 knot을 가지는데 첫 번째와 마지막 knot는 order개만큼 중첩을 시키고 그 나머지 knot에 대해서는 차수 개만큼 중첩을 시킨다.

(2) 곡면

전체적인 절차는 u v 방향이라는 것을 제외하면 곡선과 동일하다.

3. 적용 사례

CATIA에서의 모델(첫 번째 그림)을 SurfART에서 ZES import한 예(두 번째 그림)이다.

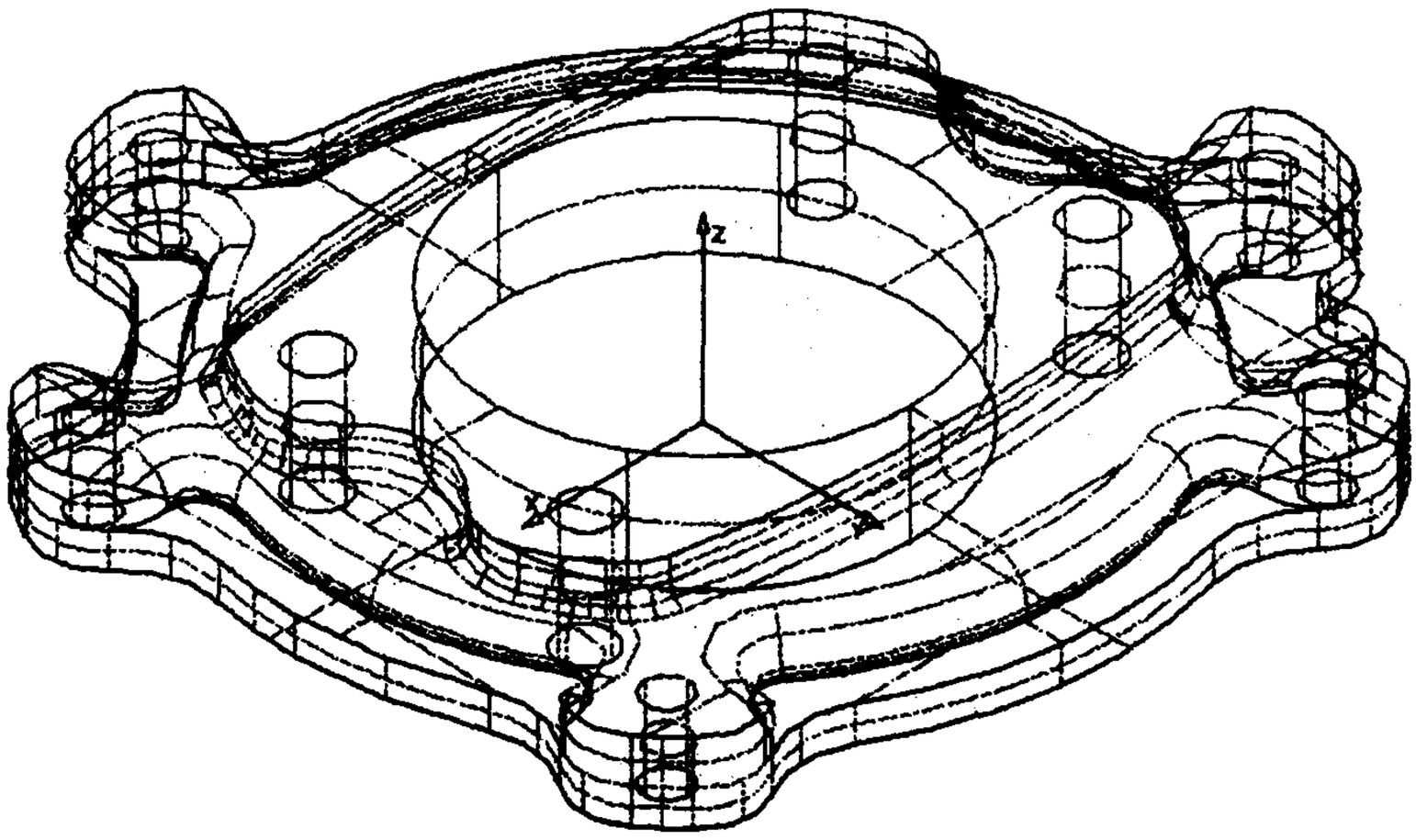


그림7.6 CATIA model

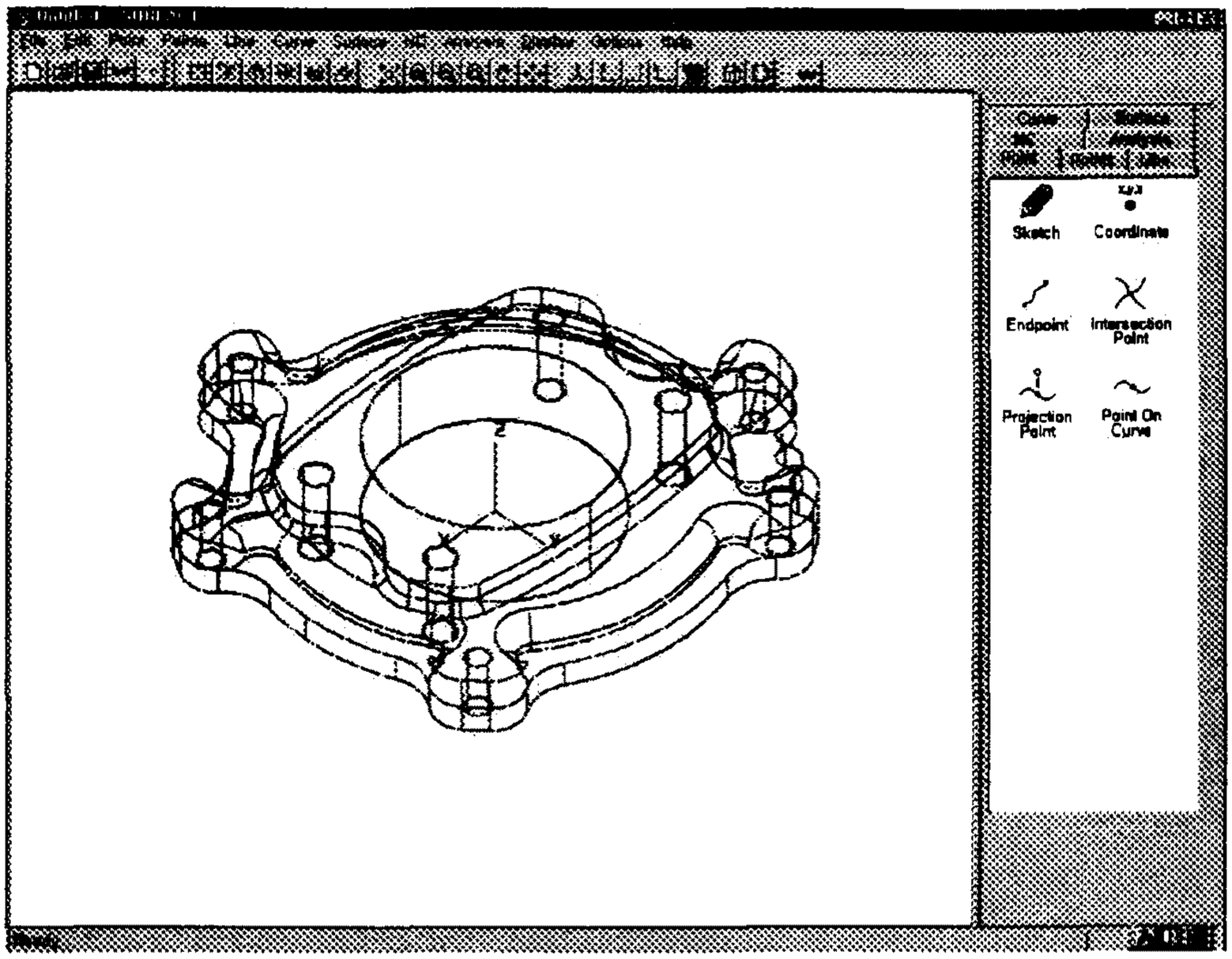


그림7.7 SurfART에서 ZES import

여 백

제 8 장 NC 가공

제 1 절 개요

NC가공은 곡선이나 곡면으로 표현된 형상의 데이터를 기반으로 가공 데이터를 산출하여 CNC 공작기계에 입력하여 수치적으로 가공하는 것을 말한다. 이러한 형상은 모델링의 방식이나 기하학이나 위상기하학과 같은 특성을 갖기 때문에 각 형상의 특징을 살리면서 정밀하게 가공하기 위해서는 형상의 특성을 고려한 가공 경로가 필요하다. 또한 공작 기계, 절삭 공구, 가공 재료에 따라 절삭 조건을 고려할 때 양질의 가공 경로를 구할 수 있다.

제 2 절 가공 조건

가공 조건은 NC가공에서 물리적 특성에 해당한다. 기하학적으로 충분히 고려된 가공 경로라도 실제 가공을 하면서 발생하는 절삭성, 절삭력, 진동, 마모 등을 고려하지 않으면 가공상 불량이 발생하게 된다. 이러한 물리적 특성은 수학적 특성과 함께 신중하게 고려할 부분으로 다음과 같은 것들이 대표적이라고 볼 수 있다.

가공 조건을 선정하는 것은 사용자의 경험을 바탕으로 이루어지고 있다. 따라서 물리적 특성을 사용자가 경험을 바탕으로 지정할 수 있도록 하고 있다.

1. 공구의 크기와 형상 (Cutter dimension and shape)

기계 가공에서 적정한 공구를 선정하는 것은 여러 가지 가공 조건을 결정하는 것 중에서 매우 중요하다. 즉 기계 가공에서 적정한 공구를 선정하면 가공 속도를 높이고 가공 비용을 절감할 수 있다. 황삭 가공에서는 필렛 엔드밀(fillet endmill)이나 플랫 엔드밀(flat endmill)이 볼 엔드밀(ball endmill)보다 절삭성이 좋아 유리하다. 정삭 가공에서는 볼 엔드밀로의 가공이 표면 조도에서 더 많은 잇점을 가지고 있다. 그 외에도 특수한 목적을 위한 다양한 공구들이 있다. 공구의 내마모성을 높인 코팅 공구나 피삭재와 공구의 마찰을 줄이

는 노즈 볼 엔드밀(nose ball endmill) 등이 있다. 정삭 가공과 잔삭 가공에서는 가늘고 긴 공구의 약한 강성을 보강하기 위해서 테이퍼(taper)공구를 사용한다. 고속도 가공을 위해서는 고속의 절삭 속도로 인한 열마모를 견딜 수 있는 CBN, 다이아몬드 공구가 필요하다.

가공 기술의 측면에서 적정한 크기의 공구를 선정하는 것은 가공에서 아주 중요하다. 공구의 크기에 따라 가공 시간과 정밀도에 큰 영향을 끼치기 때문이다. 공구의 반경이 크면 절삭 깊이를 크게 할 수 있고 이송도 크게 할 수 있어 생산성을 크게 할 수 있다. 공구의 크기가 크면 마모가 적게 되어 공구의 수명이 길어져서 경제 절삭 속도가 높아지므로 생산성이 높게 된다. 황삭, 정삭, 잔삭 가공은 사실상 공구의 크기에 의해 결정되는 것이라고 말할 수 있다. 생산성과 정 밀도를 기준으로 이 두 가지를 평가한 중에 가장 효율이 좋은 공구의 크기를 선정한다.

본 연구에서 적용 가능한 공구는 그림 8.1과 같으며 볼 엔드밀, 필렛 엔드밀 그리고 플랫 엔드밀이다. 이들 공구는 일반적으로 현장에서 널리 쓰이고 있는 것들이다. 볼 엔드밀은 가공 형상을 양호하게 가공하기 때문에 주로 사용되고 있다. 여기에서는 공구의 직경과 라운딩 반경 및 길이를 선정하여 입력한다.

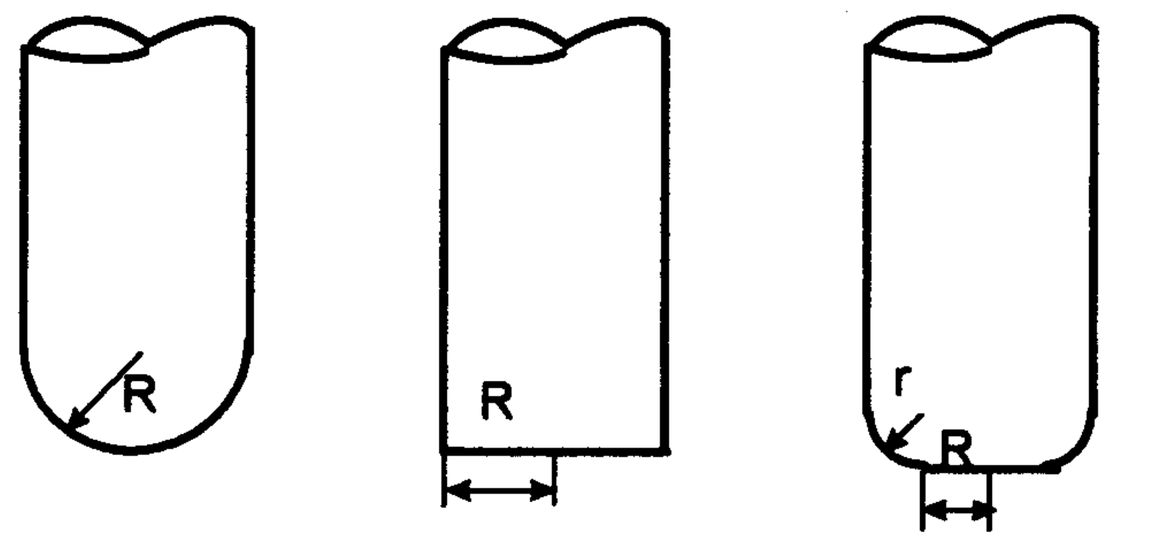


그림 8.1 절삭 공구의 종류

위의 그림에서 R은 공구의 반경이고, r은 공구의 fillet 반경이다. R, r의 상관관계로 공구의 종류를 분별할 수 있다.

2. 가공 경로 간격 (Path interval)

가공 경로의 생성 방식은 Parametric, Cartesian 가공 경로를 이용하였다. 가공 경로 간격은 가공의 소요 시간과 표면 정밀도에 크게 영향을 미친다. 가공 경로 간격의 크기를 용도에 맞게 적정하게 설정하는 것은 제품의 생산성과 정밀도에 직접적인 영향을 미친다. 가공 경로 간격을 과절삭력이 발생하지 않는 범위 내에서 되도록 크게 설정하여 황삭에서 가공 시간을 단축을 할 수 있고, 정삭에서 요구하는 표면 정밀도를 만족시키는 간격을 설정하여 정밀도를 만족시킬 수 있다.

그러나, 표면 정밀도의 경우는 곡면의 기울기가 다양하게 변하기 때문에 가공 경로 간격이 달라지고 커스프(cusp)도 달라지므로 자유곡면의 경우 적정한 크기를 설정하는 것이 필요하다.

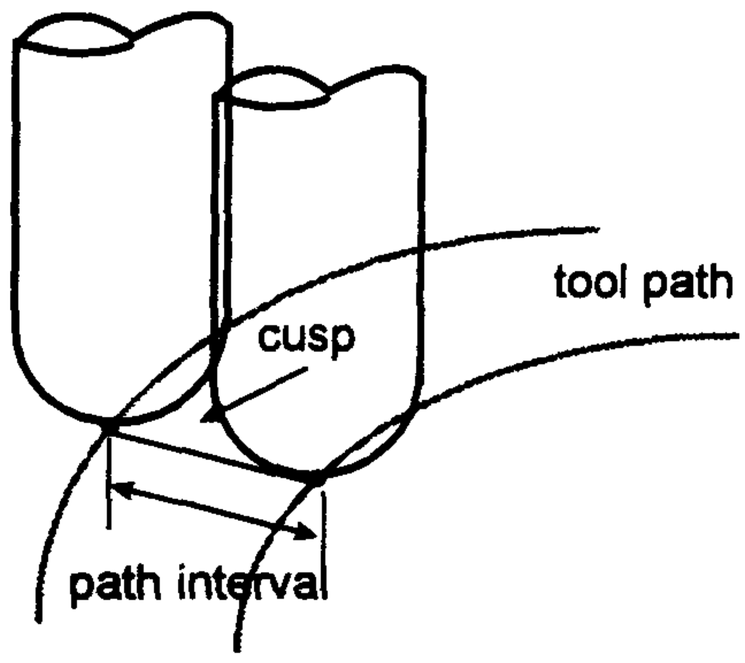


그림 8.2 가공 경로와 표면 조도(height of cusp)

3. 가공 여유 (Cutting tolerance)

가공 여유는 그림 8.3과 같이 곡면의 NC 가공에서 공구는 공구 경로상의 점(CC point)들을 직선 보간하면서 따라가는데 직선 보간으로 인한 오차의 허용치를 지정한 것이다. 오차가 너무 작아도 가공 시간이 길어지므로 적정한 가공 효율을 고려한 오차를 지정해야 한다.

intol 이 정해진 tolerance 보다 크면 과절삭이 되는 것이고, outol 이 tolerance 보다 크면 미절삭이 되어 표면 정밀도가 크게 악화된다. 그러나 NC 기계의 특성상 직선 보간을 할

때 시작점에서 가속되었다가 종착점에서 감속되다가 정지하는 동작을 하기 때문에 가공점이 많을 수록 가공 시간이 길어지게 된다. 무조건 tolerance를 작게 하는 것이 최적의 가공이 될 수 없다는 것이다. 그러므로, 가공 시간과 정밀도를 고려한 적정한 tolerance 설정이 요구된다.

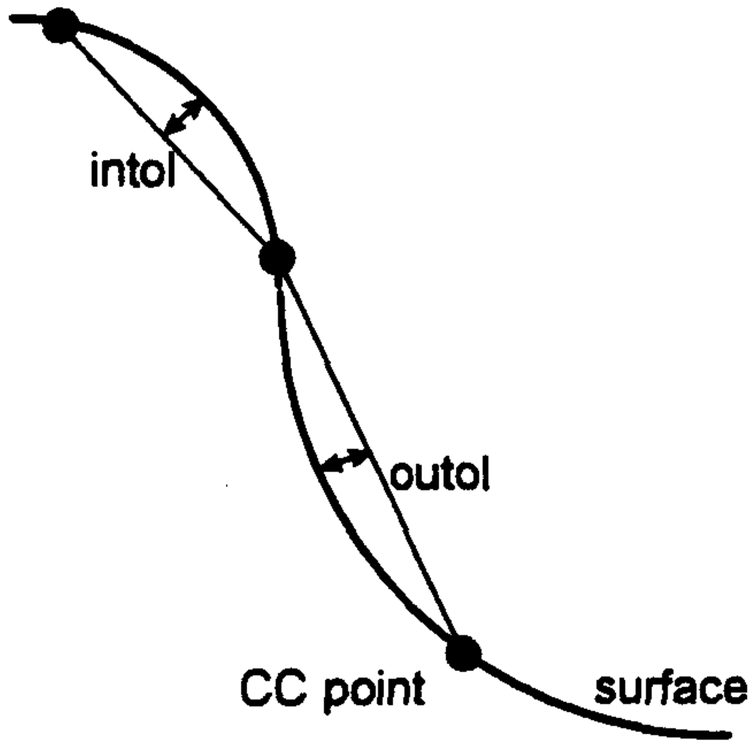


그림 8.3 가공 공차(tolerance)

4. 옵셋 (Surface offset)

곡면의 황삭 가공 시 정삭 여유나 제품의 두께 등을 고려하기 위한 곡면 법선 방향의 두께를 지정한다. 제품 곡면(part surface)을 상하지 않으면서 빠른 황삭이나 사출 제품을 생산하기 위해서 사용하고 있다. 대개 1~2 mm 를 사용하고 있다.

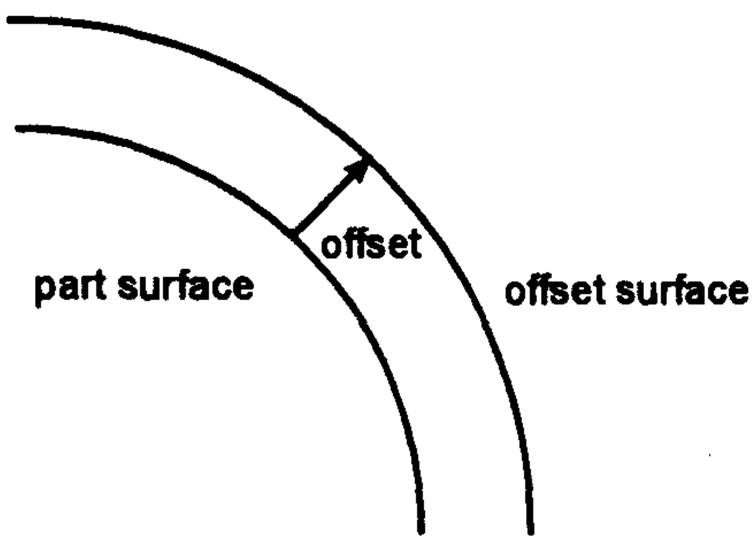


그림 8.4 곡면의 옵셋(offset)

옵셋은 절삭 깊이로도 사용되며 가공 시간과 정밀도, 그리고 절삭력에 큰 영향을 미친다. 대개는 공구의 크기에 따라 절삭 깊이를 정한다. 또한, 절삭 깊이는 가공의 방식, 즉 상향 절삭 (up-milling), 하향 절삭 (down milling)과 가공의 종류(황삭, 중삭, 정삭)에 따라 조정이 된다. 상향 절삭과 황삭에서는 공구와 피삭재간에 떨림이 발생하지 않는 한도 내에서 정하고, 정삭에서는 가공 정밀도를 위하여 공구의 휨이 일어나지 않는 한도 내에서 정한다.

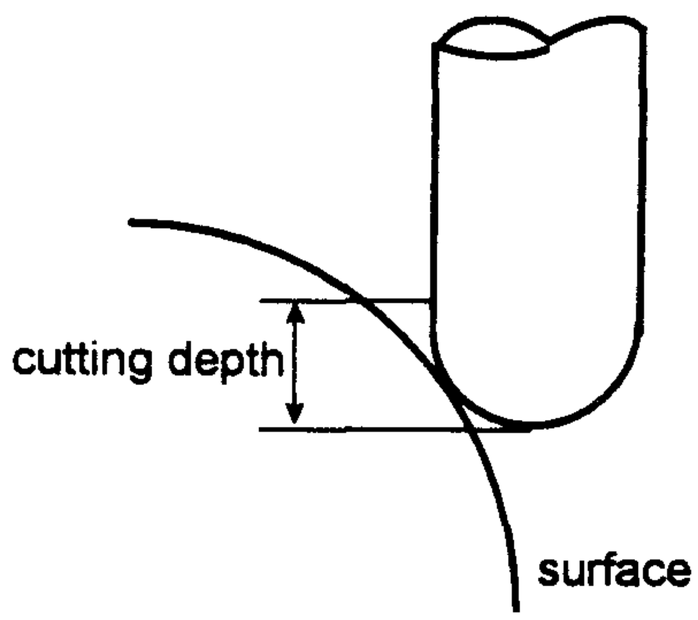


그림 8.5 절삭 깊이

5. 가공 시작점 (Start point)

절삭 시작점은 공구의 출발점으로 공작물 좌표계의 좌표 값을 지정한다. 가공을 마친 후 공구는 그림 8.6 와 같이 이점으로 되돌아 온다. 가공 시작점을 설정하는 것은 상당한 주의가 필요하다. 설계자는 설계의 편의상 가공 시작점을 제품의 내부에 설정하는 것이 관행이기 때문에 가공 시작부터 심각한 문제가 발생된다. 그러므로 가공 시작점이 제품의 외부에 있도록 주의하여 가공 시작점을 설정해야 한다.

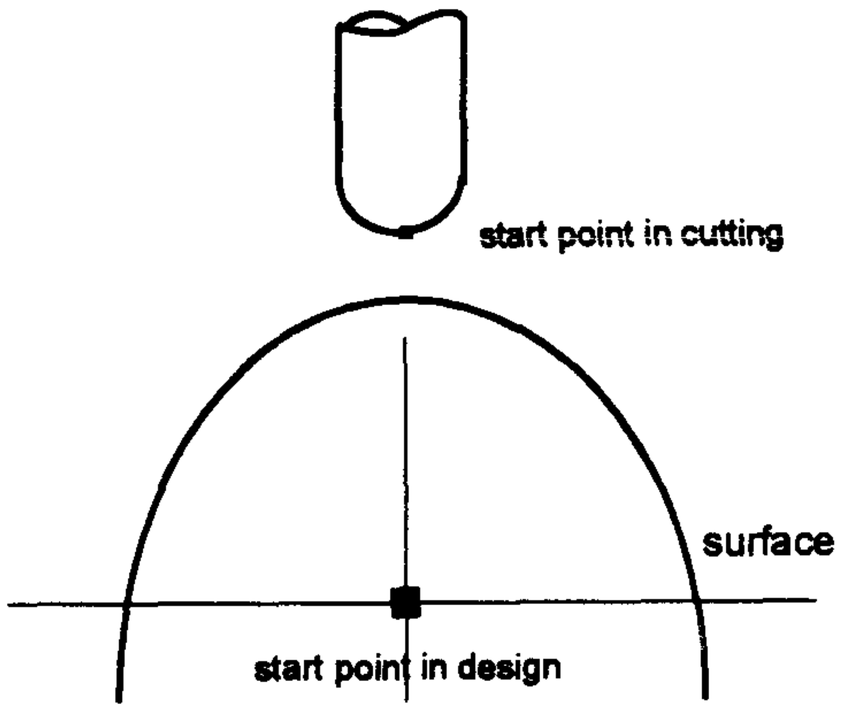


그림 8.6 설계와 가공의 시작점

6. 공구의 이송 높이 (Clearance height)

공구의 이송 높이는 공구가 곡면에 접근하기 전까지 급속 이송으로 이동되는 곡면상의 높이다. 이는 공구가 곡면에 처음 접근하거나 가공 경로를 연결하거나, 공구 경로상에 결점이 있거나, 또는 곡면으로부터 빠져 나올 때에 이용된다. 여기에는 3가지 종류가 있는데 ABS-Z는 곡면상의 점 위의 Z축의 절대값을 지정하는 것이며, INC-Z는 Z축 방향으로 증분 치를, T-축은 곡면상의 점에서 공구 축 방향으로의 증분치를 나타낸다.

7. 공구의 접근 높이 (Approach height)

공구가 곡면을 가공하기 위하여 곡면에 접근할 때는 급속 이송에서 절삭 이송으로 바뀌어야 한다. 이때 어느 정도의 높이에서부터 절삭 이송을 할 것인지를 곡면 위의 상대 값으로 지정한다. 접근 방향은 공구의 이송 높이의 종류에 의해 Z방향 또는 공구 축 방향으로 결정된다.

8. 가공 진행 방식 (Zig/Zig-zag)

가공 경로의 진행 방향은 그림 8.7과 같이 지그(Zig) 또는 지그재그(Zigzag)방식 중 어느 것으로 할 것인가를 지정한다. 가공 방식과 절삭 방향에 따라 가공 시간과 가공 표면의 질이 결정된다. 지그재그 방식으로 가공을 할 경우에 가공 시간을 단축시킬 수 있다.

그러나 절삭력의 변화가 심하고 표면 정밀도가 일정하지 않다. 지그 방식으로 공구가 진행하면 가공 시간은 길어지지만 절삭력이 일정하여 안정된 절삭을 행할 수 있고 표면 정밀도가 좋아진다.

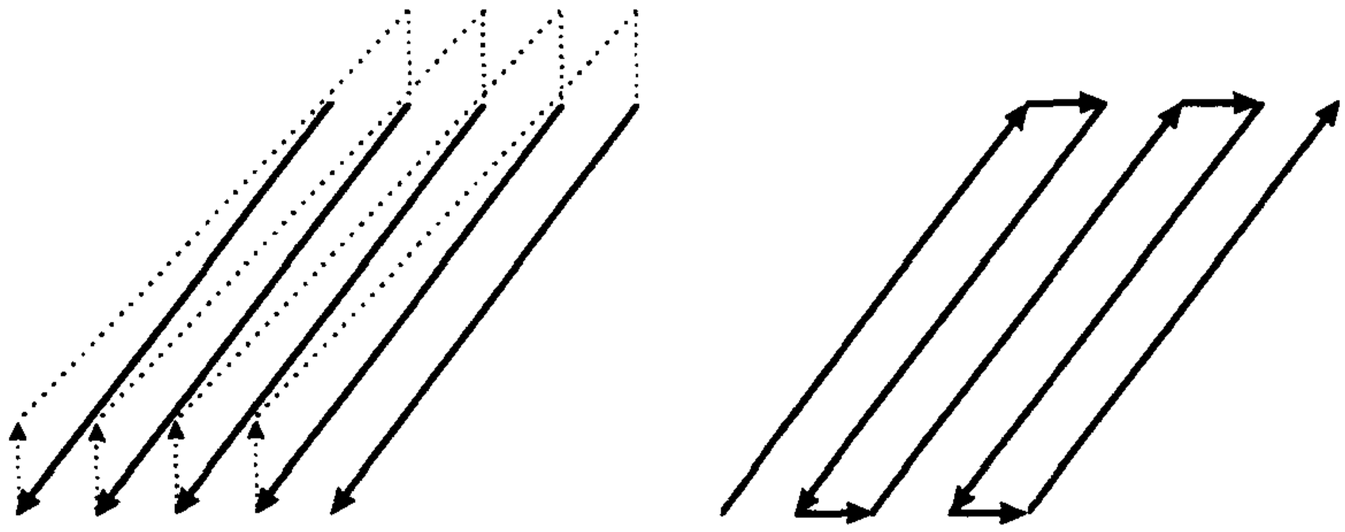


그림 8.7 Zig / Zig - zag 방식

9. 경로 연결 방법 (Path connection)

이웃하는 2 경로의 연결 방법을 지정한다. D는 앞 경로의 끝점과 다음 경로의 시작점을 직선으로 연결하고, Z은 연결 도중 곡면을 과절삭하지 않도록 Z축 이동과 XY평면상의 움직임으로 나누어서 연결하는 것이며, J는 공구를 점프(jump)시켜 연결한다. 그림 8.8에서 직선은 절삭 이송, 점선은 급속 이송을 나타낸다.

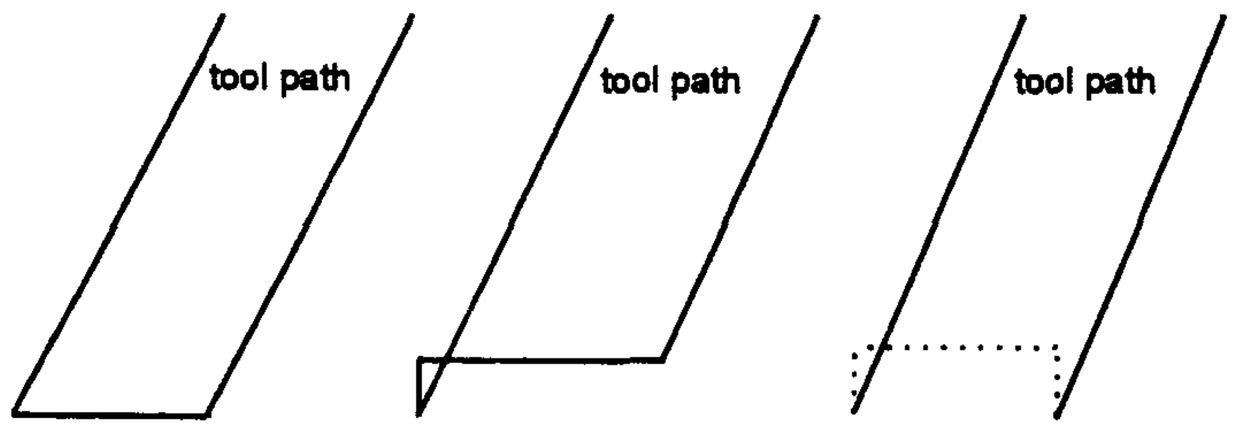


그림 8.8 가공 경로의 연결 방식

10. 이송 (Feed rate)

이송은 절삭 가공 속도와 절삭력을 조절하는 대표적인 인자로서 생산성에 중요한 영향을 미치는 인자이다. 황삭에서는 떨림을 방지하기 위해 일정한 절삭율을 유지하도록 조정되며 정삭에서는 공구의 휨이 정해진 공차 안에 있도록 조정된다. 밀링 가공의 방식(up-milling 이나 down-milling)과 가공의 방향(upward milling 이나 downward milling)에 따라 이송을 변경하여 절삭력을 조정할 수 있다. 그러나 이송은 너무 느려도 기계적 틈제거 마모 때문에 공구의 수명이 저하된다. CNC 밀링 가공에서 이송의 또 다른 영향으로는 왕복 단차이다. 이는 지그재그 가공에서 이웃하는 가공 경로의 절삭 방향이 서로 반대 방향일 때에 발생한다. 이송이 크게 되면 표면 조도가 커지므로 왕복 단차를 피해야 한다. 특히 정밀도를 중요시하는 정삭에서는 가공 방향이 반복해서 반대로 바뀌는 지그재그 가공 을 피해야만 한다. 또한 공구의 바깥 지름, 날 길이, 절삭 깊이, 절삭 유제, 피삭재 등을 고려하여 이송을 적정 영역 내에서 결정해야만 한다. 날 길이가 길 수록, 절삭 깊이가 클 수록, 공구의 바깥 지름이 작을 수록 작은 이송을 주어야 한다. 이송도 현장에서 사용되고 있는 공식을 적용한다. 또한 절삭 이송 속도를 곡면의 가공 부위에 달리 줄 수 있도록 하였고 절삭력의 변화를 일정한 범위내에 있게 하여 안정한 절삭을 행하도록 모두 5 개의 부위로 나누어 입력하는데 그 내용은 다음과 같다.

F1 : 곡면 내부의 절삭 속도

F2 : 곡면으로의 접근 속도

F3 : 첫 path 의 절삭 속도

F4 : 마지막 path 의 절삭 속도

F5 : 가공 경로의 연결 시의 절삭 속도

F1 은 보통 절삭 가공 속도이고, F2, F3 ,F5 은 절삭력이 커지거나 충돌이 있기 때문에 낮은 속도이고, F5 은 가공을 행하지 않기 때문에 급속 이동을 하는 이송 속도이다.

11. 공구의 회전속도 (Spindle speed)

공구의 회전 속도는 가공 속도에 큰 영향을 미친다. 절삭 속도가 너무 빠르면 공구 마모가 심해지고 너무 느리면 공구 날이 파괴되기 때문에 적정한 회전 속도를 정해야 한다. 공구 회전 속도를 국제 표준인 rpm 단위로 입력한다.

12. 공구의 간섭 (Interference check)

공구 간섭은 공구가 가공물을 설계대로 가공하지 않고 제품을 파먹는 현상이다. 이런 현상은 가공물의 질을 해치기 때문에 반드시 제거해야 한다. 공구 간섭은 직경이 큰 공구를 사용하는 황삭에서 크게 발생하고 정삭은 최종적으로 제품을 만드는 단계이기 때문에 간섭 제거는 필수적이다. 본 시스템은 정삭에서 볼 엔드밀을 사용할 경우에만 공구 간섭을 제거할 수 있다.

13. 후처리기 (Post-processing name)

NC post-processing 을 위하여 사용하고자 하는 NC 콘트롤러에 해당되는 후처리 설정파일(post-processor configuration file) 이름을 입력한다. 가장 많이 사용되고 있는 FANUC 의 콘트롤러를 기준으로 이용하였다.

14. 가공 조건의 설정

위에서 설명한 가공 조건들은 대화상자를 통해서 설정하도록 하였다. 가공 조건들은 사용자의 경험이 중요하기 때문에 사용자가 직접 설정하도록 하였다. 위의 설명을 기초로 대화상자의 값을 설정하면 도움이 될 수 있다.

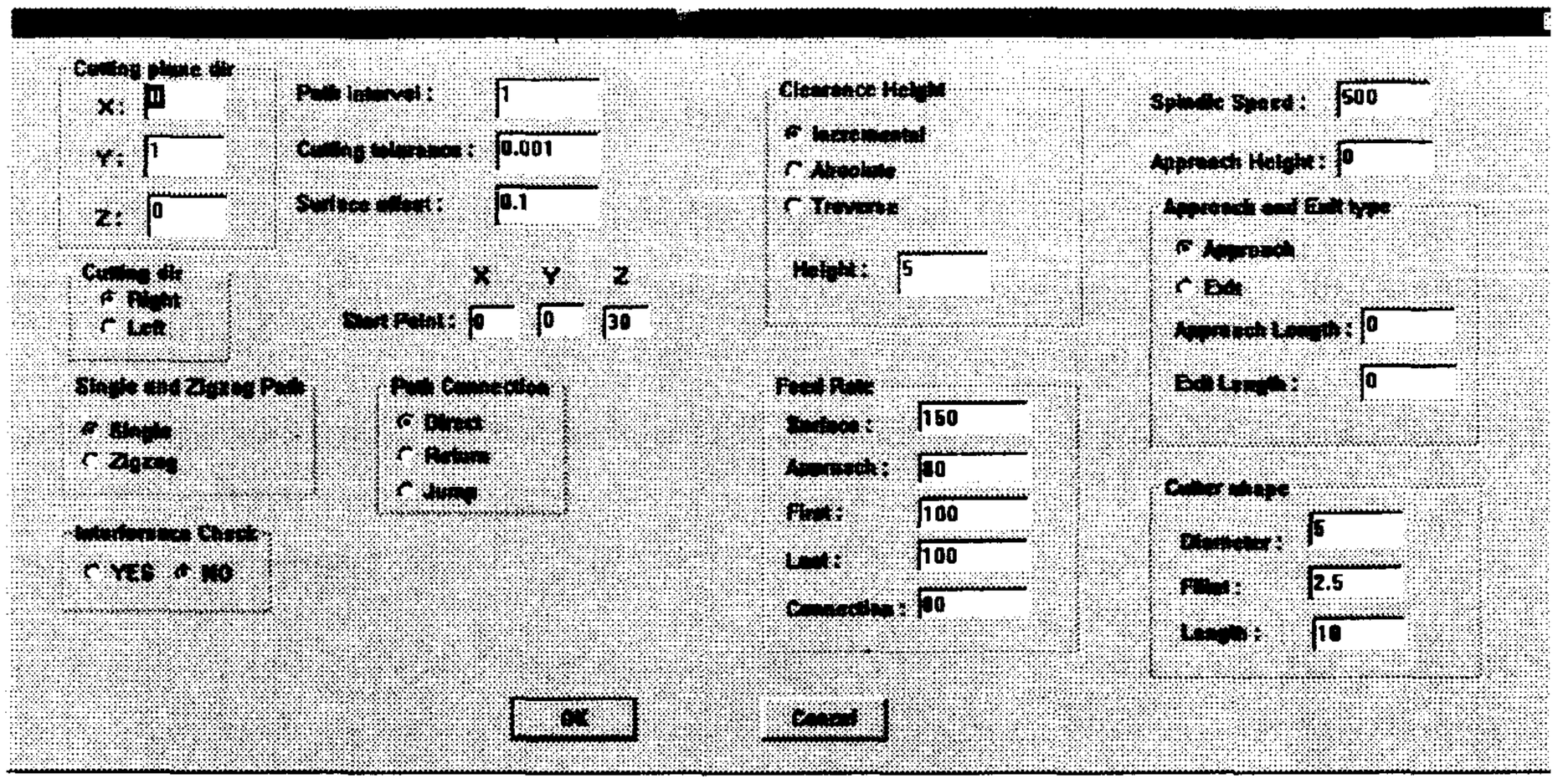


그림 8.9 절삭 조건을 결정하는 Dialogue box

제 3 절 가공 경로

가공 경로는 모델링에서 생성한 형상을 가공하기 위해서 절삭 공구가 지나가야 하는 경로를 말한다. 가공 경로의 생성은 아래의 그림 8.10과 같은 절차가 필요하다. 그림에서 보는 것처럼 가공 경로는 절삭 공구, 형상의 특징, 공작 기계의 특성 등에 영향을 받기 때문에 각 단계에서 필요한 조건들을 고려해야 한다.

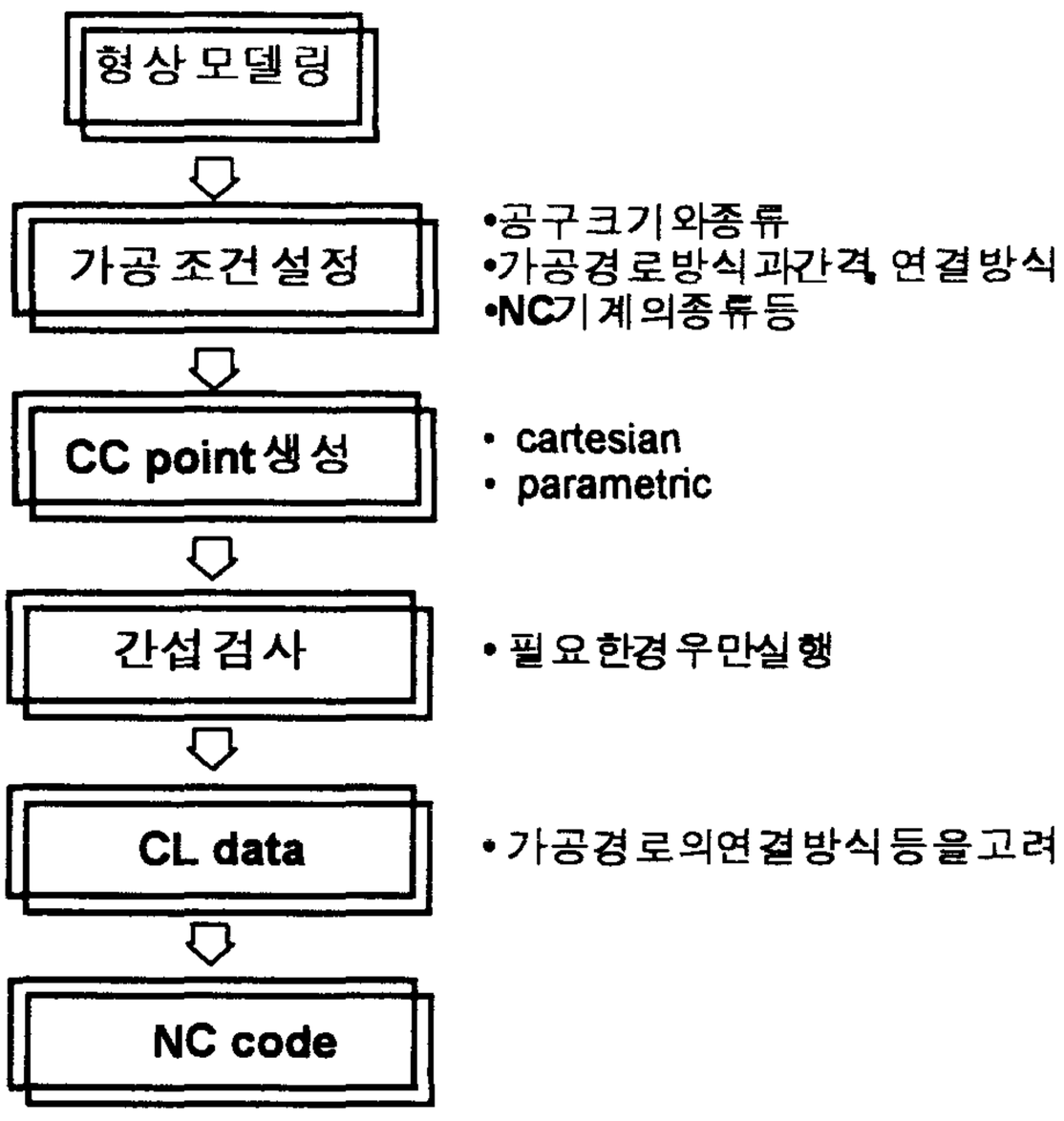


그림 8.10 가공 경로 생성의 절차

가공 조건들을 설정하는 단계에서는 공구 크기와 종류, 가공 경로의 방식과 간격, 경로의 연결 방식, NC 기계의 종류, 가공 시작점 등을 설정한다. CC 생성 단계에서는 Cartesian 방식이나 parametric 방식을 이용해서 CC 를 구한다. 간섭 검사 단계는 절삭 공구가 형상을 과절삭하지 않도록 간섭 검사를 수행하기 때문에 많은 시간을 소요한다. 간섭이 발생하는 형상에 대해서만 간섭 검사를 수행하여 가공 경로 시간을 단축할 수 있다. 간섭 검사 후 공구의 선단(tip)의 위치를 계산하여 CL data 를 생성한다. 그리고 공작 기계가 CL data 를 인식할 수 있도록 NC code 화한다.

이러한 절차을 거쳐서 생성된 NC code 를 공작 기계에 입력하면 모델링한 형상을 NC 가공할 수 있게 된다.

1. 황삭 가공 경로

황삭 가공은 최종 제품의 정밀도를 크게 고려하지 않고 거칠게 가공하여 가공 시간의 단축을 목표로 하는 가공 방식이다. 빠른 이송 속도와 절삭 깊이가 중요한 가공 조건

으로 작용한다. 본 연구에서는 일반적인 가공 경로 방식인 Cartesian 가공 경로와 안정적이고 신속한 가공 방식인 황삭 가공 경로를 지원하도록 한다.

가. CARTESIAN 가공 경로

Cartesian 가공 경로는 모든 형상의 NC 가공에 사용된다는 일반성이 장점이다. 이런 장점때문에 가장 많이 사용되고 있다. 곡면과 절단 평면의 교차 곡선을 이용한 CC-Cartesian 과 옵셋 곡면과 절단 평면의 교차 곡선을 이용한 CL-Cartesian 이 있는데 여기서는 CC-Cartesian 을 이용한다. CC-Cartesian 은 아래 그림과 같다.

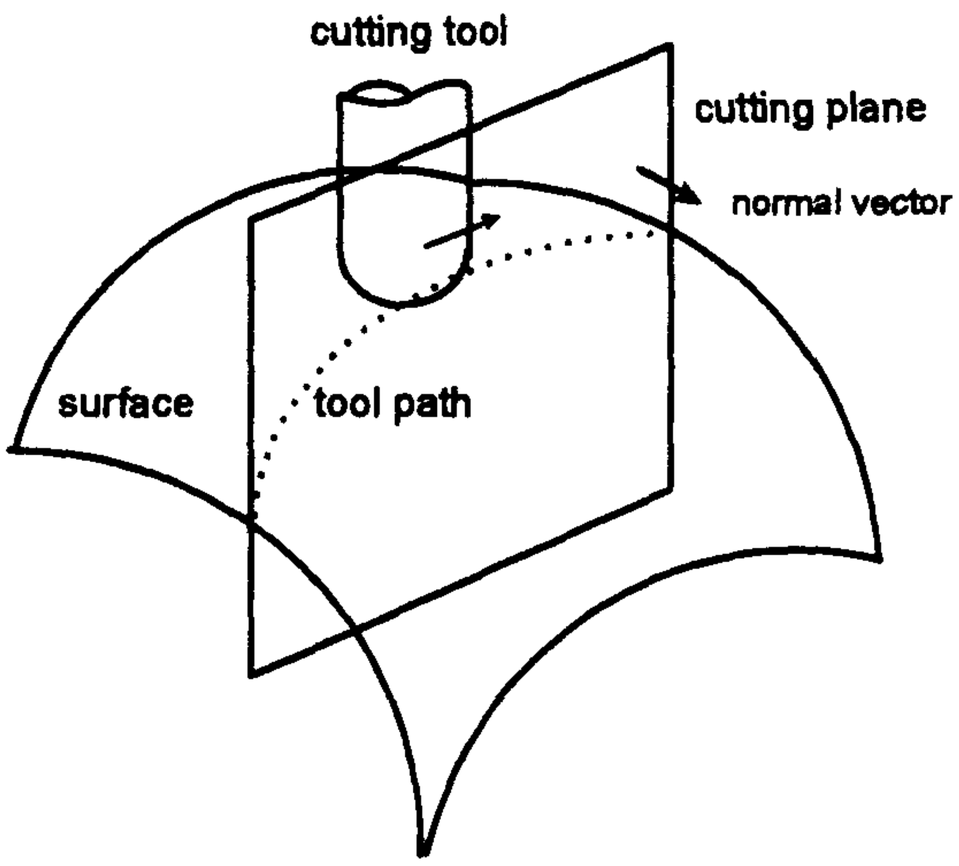


그림 8.11 Cartesian 가공 경로의 생성

절단 평면과 곡면의 교차 곡선은 점선으로 그려졌는데 이 점들과 절삭 공구가 접하면서 이동하여 가공을 수행한다. 절단 평면은 법선 벡터(normal vector)을 따라 이동하면서 가공 경로의 CC 를 생성한다.

(1) 가공 범위 설정

모델은 face 의 집합으로 이루어 졌기 때문에 각 face 와 원점간의 거리를 계산해서

절단 평면을 생성할 범위를 설정한다. 각 face 의 경계(boundary)의 점들과 사용자가 설정한 절단 평면의 방향 벡터(normal vector)를 이용하여 절단 평면을 생성한다. 이 절단 평면과 원점(*origin*)의 거리를 계산해서 절단 평면, 즉 가공 범위를 설정한다. 그림 8.12 에서 절단 평면은 *cutting plane 0 ~ cutting plane 1* 이다. 이 사이를 가공 간격 만큼씩 이동하여 절단 평면을 생성하고 face 와 교차 곡선을 구하여 CC point 를 생성한다.

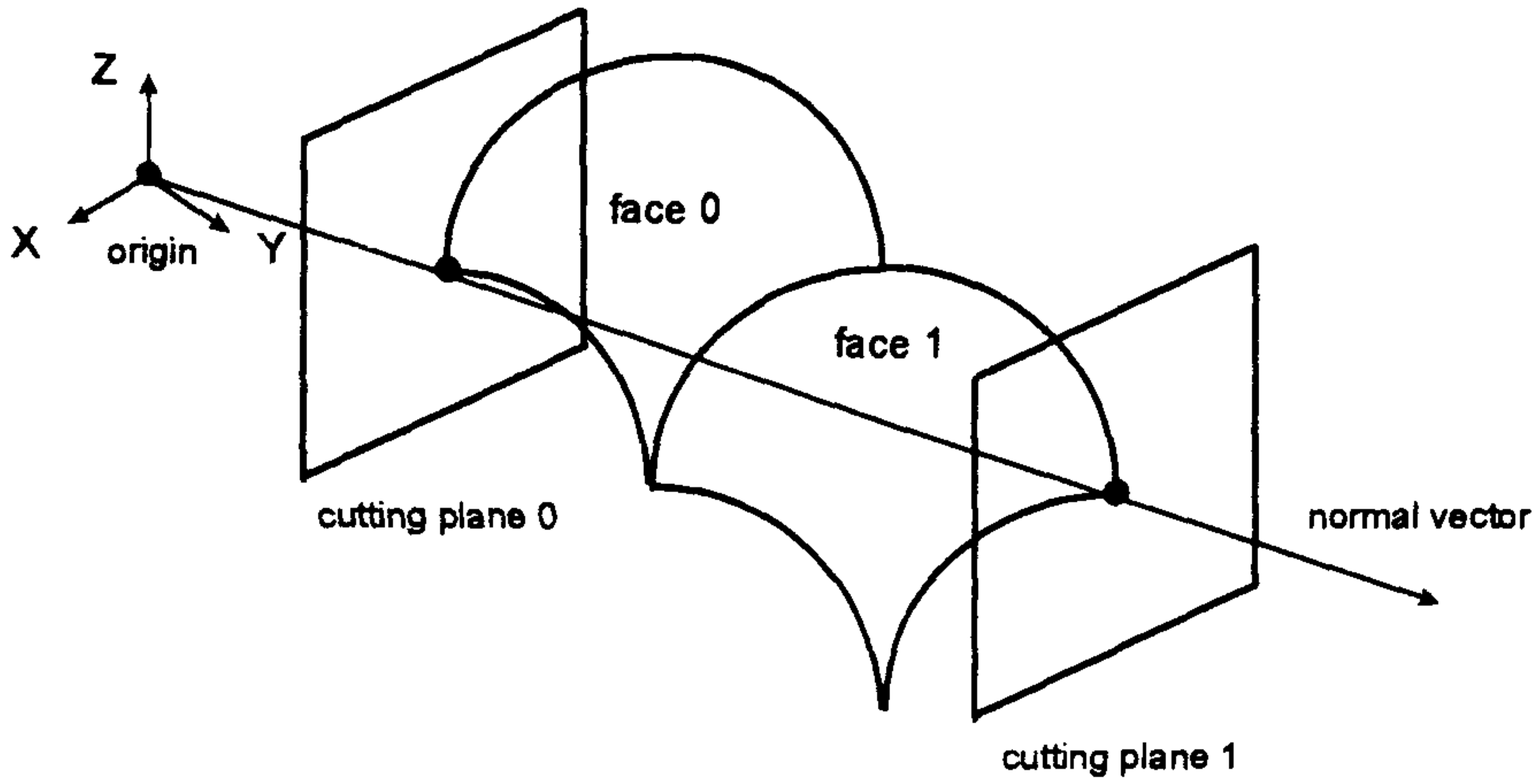


그림 8.12 절단 평면의 설정 범위

(2) face/face 교차 곡선

절단 평면은 위상이 face 이고, 형상의 곡면도 face 의 집합으로 이루어졌기 때문에 face/face 교차 곡선으로 CC point 를 구한다. face 를 단위로 한 것은 통합 모델을 생성할 경우 trim 곡면을 사용하게 되는데 이 trim 곡면을 가공하기 위해서이다. 작동 원리는 5 장의 face/face 교차 곡선을 이용하였기 때문에 그 장을 참조하면 된다.

(3) CC point 의 최적화와 CL data 생성

(2)에서 구한 CC point 의 열중 점간 거리가 너무 큰 경우는 절삭 이송으로 이동하지 않고 급속 이송으로 이동해야 한다. 이러한 CC 점의 처리는 가공 속도를 빠르게 하고 과절삭을 피하게 한다. 이러한 경우는 face 와 face 가 상당한 거리만큼 떨어진 모델에서 발

생한다.

정리한 CC 점과 그 점의 법선 벡터를 이용하여 NC 공작 기계가 CC 점에 접하도록 하는 공구 위치(CL)를 인식하도록 하는 CL data 를 생성한다. CL data 의 생성 작동은 아래의 그림과 같고 계산은 아래의 식을 이용한다.

$$CL = CC + R(n-t) \qquad (8.1)$$

여기서, R 은 절삭 공구의 반경, n 은 CC 에서의 법선 벡터, t 은 공구 축 벡터이다.

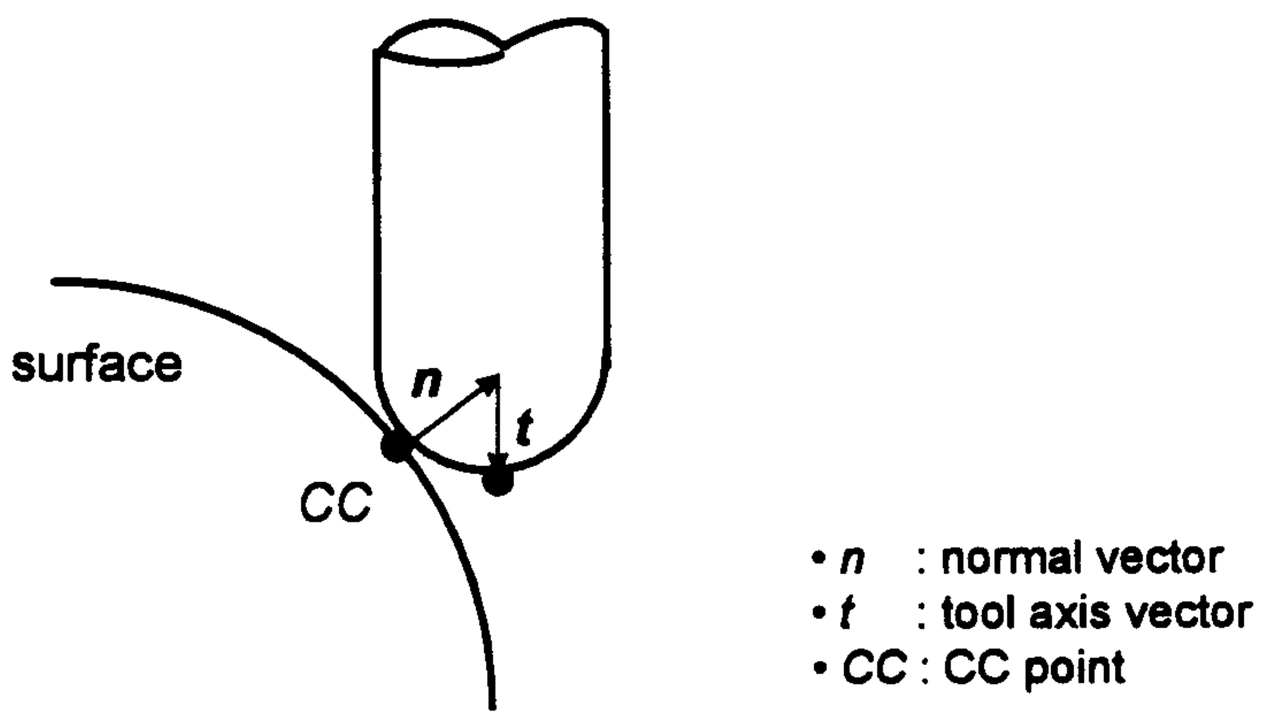


그림 8.13 CL data 생성

(4) 간섭 회피

간섭 회피는 절삭 공구가 형상을 과절삭하는 것을 방지하여 양질의 제품을 생산하는 방식이다. 간섭 회피는 CC data 를 이용해서 곡면을 삼각형으로 이루어진 다면체 모델로 변환하고 공구의 중심과 모델의 삼각형과의 거리 검사를 하여 공구 반경과 비교함으로써 수행된다. CC 점들을 절삭 공차가 만족하도록 구했기 때문에 이 비교는 효율적인 간섭 회피를 수행할 수 있다.

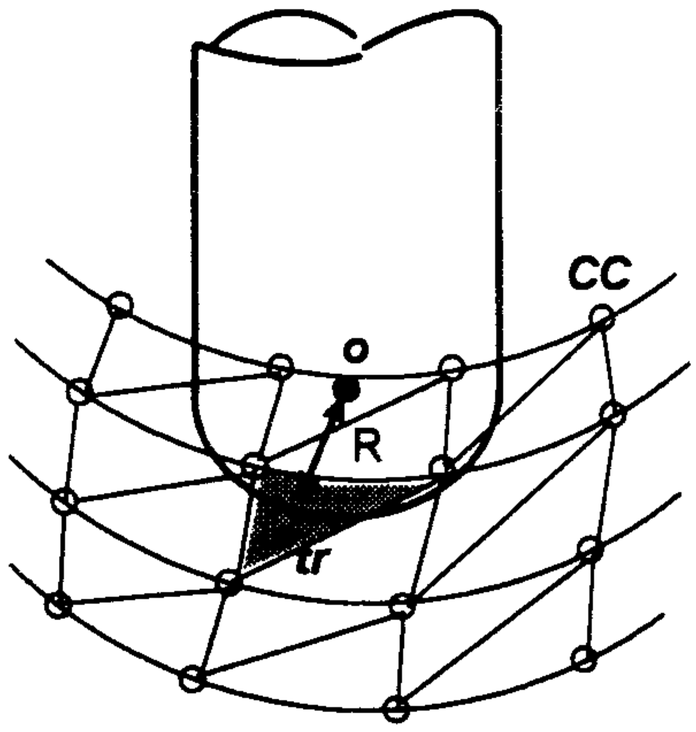


그림 8.14 간섭 회피

그림 8.14 와 같이 공구 중심점 *o* 와 삼각형 *tr* 의 거리가 공구의 반경 *R* 보다 크면 간섭이 없고, 공구의 반경보다 작으면 간섭이 발생하는 부분이다. 공구 반경보다 작은 곡률 반경을 가진 곡면의 부분을 가상으로 간섭이 발생하는 부분으로 인식했다가 위의 방법으로 정확한 간섭 검사를 행한다. 간섭이 일어난 부분은 간섭 영역의 삼각형들의 법선 벡터의 합의 방향으로 공구를 이동하여 간섭을 피한다.

(5) 적용 예

아래의 그림은 Cartesian 가공 경로를 마우스 형상에 적용한 것이다. 그림 8.15 은 마우스 형상의 CC 점들이고, 그림 8.16 은 마우스 형상을 가공하기 위한 CL data 이다. 실선은 공구가 실제로 가공하는 경로이고 점선은 공구가 가공하지 않고 급속 이동하는 부분이다. 절삭 공구의 직경은 10mm 이고 여유 공차는 0.0mm 이다.

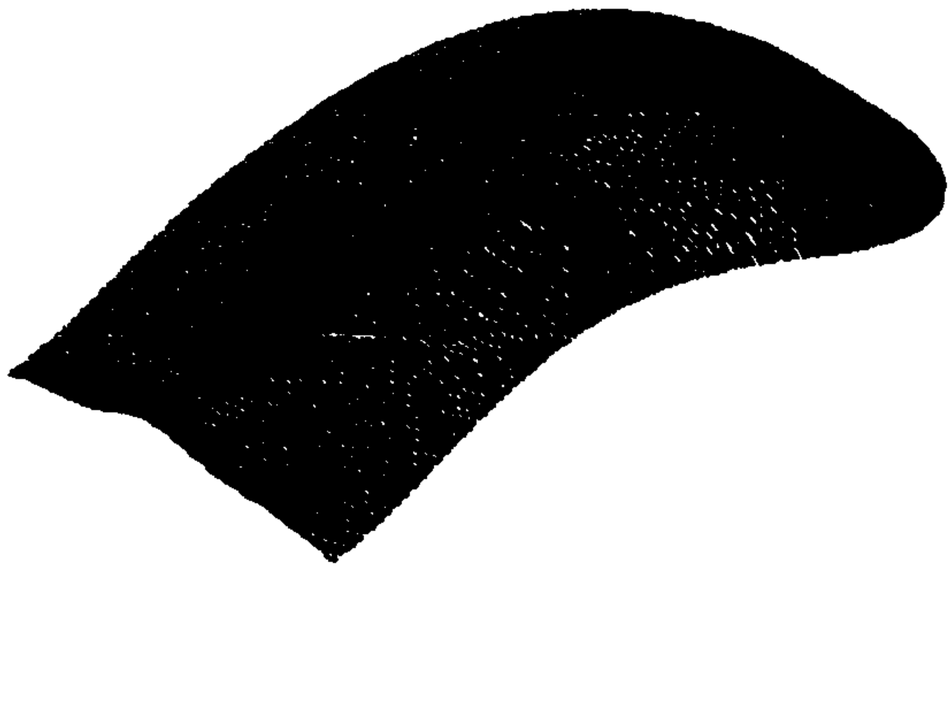


그림 8.15 마우스 형상의 CC data

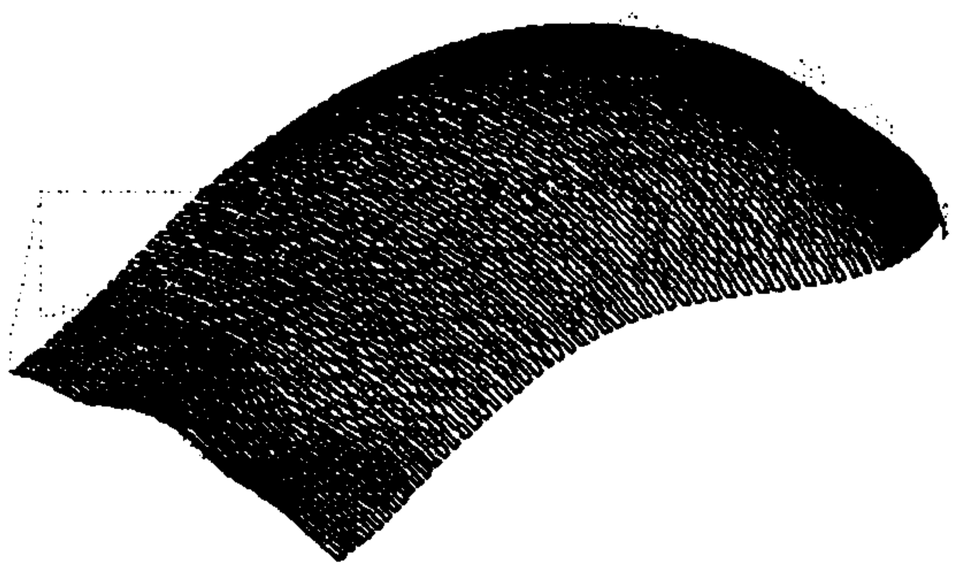


그림 8.16 마우스 형상의 CL data

2. 정삭 가공 경로

정삭 가공은 마무리 가공으로 설계상 요구하는 표면 정밀도와 형상 정밀도를 만족하는 제품을 생산하는 가공 경로이다. 정삭 가공 경로는 Cartesian 가공 경로와 parametric 가공 경로의 2 가지 종류를 생성하였다.

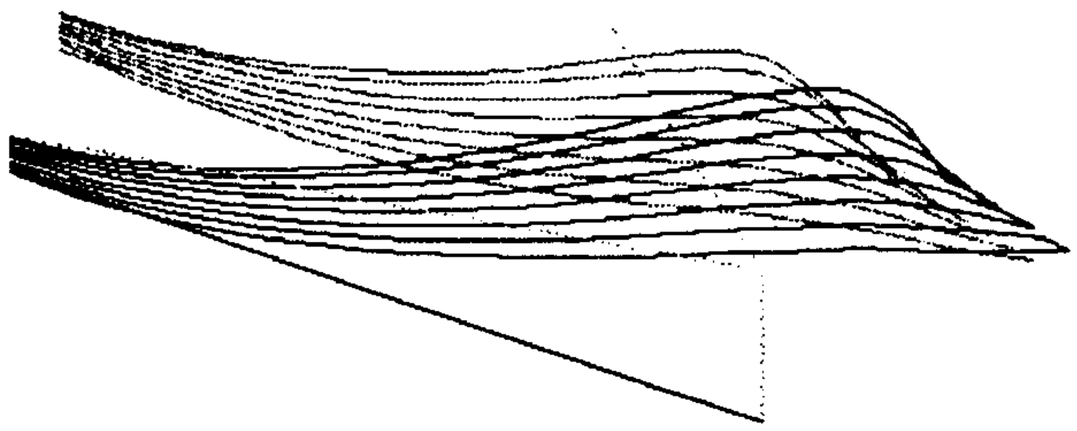


그림 8.20 임펠러의 parametric 가공 경로

3. 가공 경로의 시뮬레이션

가공 경로의 시뮬레이션은 가공 경로를 컴퓨터 상에서 그래픽으로 미리 검증한다는 의미로 중요하다. NC 가공 자체는 상당한 시간을 소모하는 작업이다. 그러나 시행착오를 적게 하기 위해서는 NC code가 올바르게 생성되었는지 검증할 필요가 있다. 가공 경로의 시뮬레이션은 가공 경로를 따라 공구가 이동하는 모습을 통해서 절삭 중 공구의 동작을 직관적으로 이해하고, 가공 중에 발생할 오류를 미리 감지할 수 있게 한다. 가공 경로는 실제로는 수치이기 때문에 직관적인 검사가 사람으로는 매우 힘들다. 그래서 이 시스템은 가공 경로를 그래픽적으로 처리함으로써 직관적 검사를 가능하게 하였다. 오류 발생 부분은 NC code의 수정으로 정상적 가공 작업을 수행할 수 있다.

4. NC code 생성

가공 경로의 시뮬레이션을 통하여 검증된 가공 경로는 공작 기계의 컨드롤러가 인식할 수 있도록 CL data를 NC code로 변환한다. 컨드롤러은 제작사와 기계의 종류에 따라 다를 수 있지만 세계의 표준이라 할 수 있는 FANUC사에 맞추어 NC code를 생성하여 여러 종류의 공작 기계와 호환성을 유지했다.

방향으로 설정한다.

face가 여러 개로 이루어진 모델인 경우는 face들의 경계를 계산해서 길이가 긴 방향으로 가공 방향을 설정한다. 이런 방식은 face와 face가 만나는 경계에 관계없이 face 간의 연속성을 고려하면서 가공 경로를 생성할 수 있다. 또한 절삭 조건 측면에서는 가공 경로의 변화가 적기 때문에 좀 더 빠르고 안정된 가공을 행할 수 있다.

(2) 가공 경로의 간격을 일정하게 조정

parametric 가공 경로는 매개 변수를 증가시켜서 가공 경로를 생성하기 때문에 곡면식의 영향을 직접적으로 받는다. 예를 들면, 곡면의 한편 경계가 좁고 맞은편의 경계가 넓다고 한다면 가공 경로간의 폭도 경계가 좁은 쪽에서는 좁고, 맞은편에서는 넓어 질 것이다. 가공 경로간 폭이 넓어진다는 것은 표면 정밀도가 악화됨을 의미한다. 이를 방지하기 위해서 전 가공 경로(*path i*)를 참조해서 가공 경로(*path i+1*) 간격이 일정하도록 한다. 그림 8.18에서 보는 것처럼 5번째 점들($f_i, f_{i+1}, \ldots$)의 거리를 계산해서 정해진 가공 경로의 간격을 넘지 않도록 조정하면서 가공 경로를 진행한다.

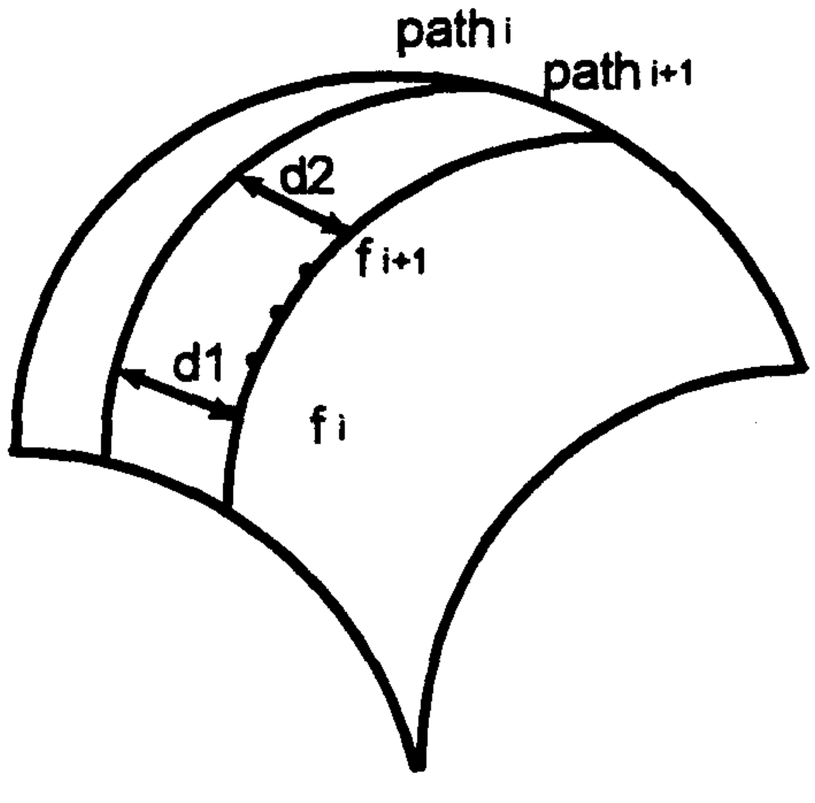


그림 8.18 parametric 가공 경로의 간격을 일정하게 함

(3) CL data 생성

(1), (2)에서 구한 CC data 은 Cartesian 가공 경로와 같은 과정을 통해서 간섭 검사를 행하고 CL data 를 생성한다. 식(8-1)을 이용한다.

(4) 실행의 예

그림 8.19 는 임펠러의 날개(blade)의 형상을 parametric 방식으로 CC data 를 구한 것이다. 그림 8.20 은 임펠러의 날개를 가공하는 가공 경로를 parametric 방식으로 구한 것이다. 실선은 실제 공구가 형상을 가공하는 부분이고 점선은 공구가 급속 이동를 하는 부분이다. 가공 경로 산출에 쓰인 절삭 공구의 직경은 10mm 이다.

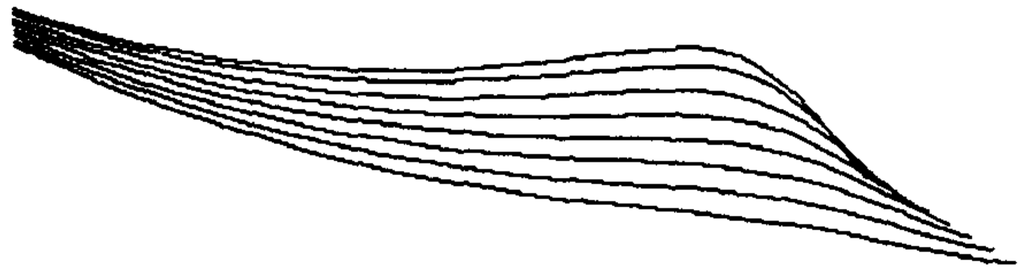


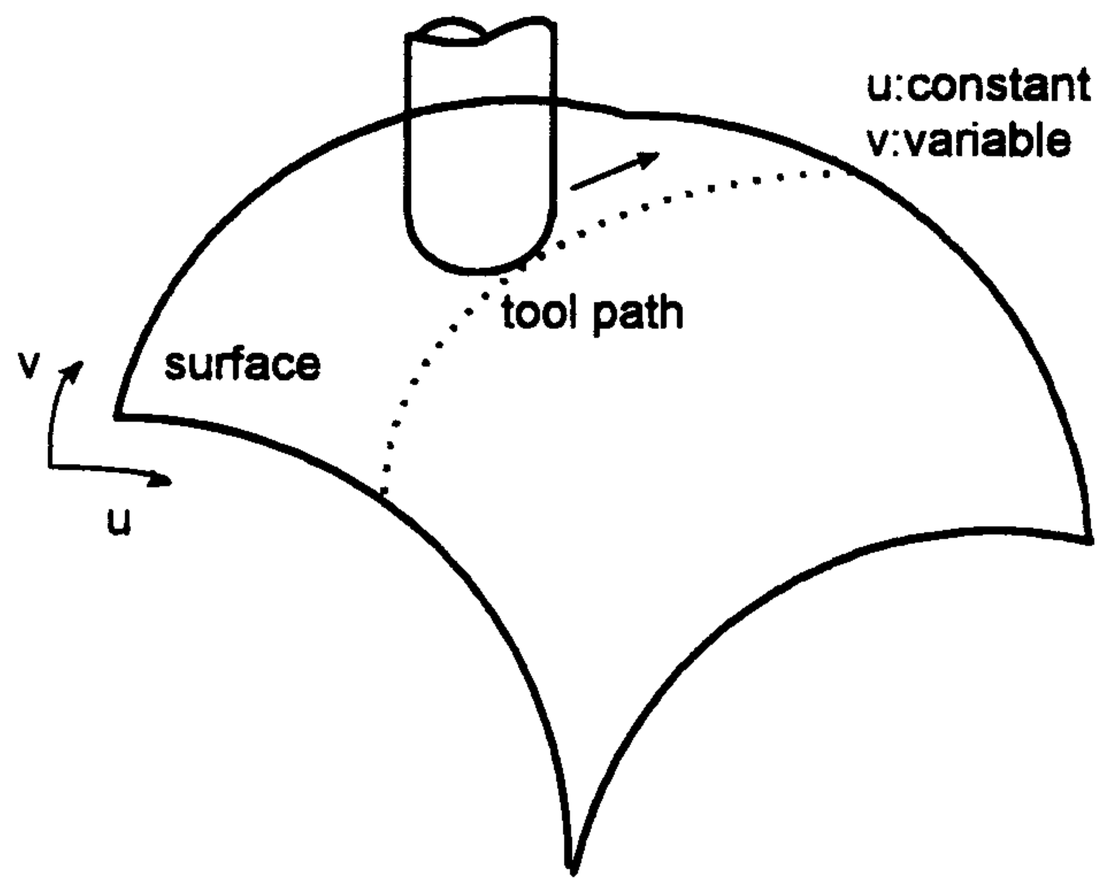
그림 8.19 임펠러 날개의 parametric 방식의 CC data

가. CARTESIAN 가공 경로

가공 경로의 생성 원리와 순서는 '1 황삭 가공 경로의 가 Cartesian 가공 경로'에서 설명한 바와 같다. 다만 절삭 조건을 설정할 때에 양호한 표면 정밀도를 위해서는 공구의 반경과 가공 경로의 간격의 비를 작게 해야 한다. 이 비가 작으면 작을 수록 조밀한 정밀도를 내기 때문이다.

나. PARAMETRIC 가공 경로

parametric 가공 경로는 형상 모델링의 특성을 이용해서 효율적인 가공할 수 있다는 장점을 가지고 있다. 또한 가공 경로를 생성하기가 쉽다. Parametric 가공 경로는 그림 8.17 에서 보는 것처럼 곡면의 매개 변수중 하나를 상수로 고정하고, 다른 매개 변수를 변화시키면서 생성되는 점열을 CC data 로 사용한다.



8. 17 parametric 가공 경로의 생성

(1) 가공 방향

parametric 가공 경로는 모델링 형상이 몇 개의 face 로 구성되었는가에 따라 다르다. 하나의 face 로 이루어졌으면 face 의 경계의 길이를 계산해서 길이가 짧은 방향을 가공

제 9 장 시스템 수행 예

본 연구에서 개발한 SurfART는 CAD/CAM 시스템으로 측정 데이터를 입력하여 곡면 모델을 생성하고, NC 가공 정보를 생성하는 소프트웨어이다. 현재 구현된 SurfART는 WINDOWS-NT에서 커넬(kernel)은 C++로, GUI는 Visual C++로 구현하였고, 형상 모델의 그래픽 디스플레이는 OpenGL을 사용하여 구현하였다. 본 장에서는 개발된 SurfART를 사용하여 금형 부품을 모델링하고, NC 가공 정보를 생성하는 과정을 도시하여 시스템의 기능을 검증하고, 필요한 기능을 추가하거나 수정하는 자료로 이용하고자 한다. 예제 금형 부품은 차량에 쓰이는 브라켓(bracket)으로 그림 9.1과 같은 모양과 크기를 갖고 있다. 그러나 개발된 시스템의 주 목적이 점 구름(point cloud)이라고 할 수 있을 정도로 많은 측정 점 데이터를 사용하여 곡면 모델을 생성하는 것이므로 단지 모양과 측정 점 데이터만 주어지고 부품의 치수는 모르는 것으로 간주한다. 그림 9.2가 예제 부품을 측정한 점 데이터이다. SurfART에서 점 데이터는 import/export메뉴로 읽거나 저장할 수 있으므로 이를 사용하면 측정 데이터를 시스템에 입력할 수 있다. 그림 9.3은 점 데이터를 Z축에서 본 그림이다. 측정 점 데이터를 사용하여 곡면을 생성할 때 가장 바람직한 것은 적은 양의 점 데이터로 원하는 곡면을 정확히 표현하는 것이다. 그러므로 다음과 같은 과정을 수행하게 된다.

첫번째 과정은 측정시 발생된 오류나 오차를 제거하고, 곡면 생성시 사용하지 않아도 되는 점 데이터를 제거하는 작업이다. SurfART는 이러한 작업을 수행하기 위하여 Points 메뉴에 Data Reduction, Filtering 기능을 갖고 있다. 그림 9.4가 이러한 기능을 사용하거나 부품의 모양을 이용하여 필요 없는 점 데이터를 제거한 것이다.

두 번째 과정은 곡면의 경계를 찾아내는 작업이다. 그림 9.3과 그림 9.4를 보면 곡면의 경계가 되는 곳에서는 점 데이터가 조밀하여 점 데이터만으

로도 대략적인 곡면의 모양을 알 수 있다. 그러므로 곡면의 경계는 이러한 점들을 지나는 곡선이나 직선이다. 그러나 많은 점 데이터가 존재하므로 이러한 점들을 모두 지나는 보간(interpolation)으로 곡선을 생성할 수 없고, 최소자승법이나 기타 다른 근사(fitting)방법을 사용해야 한다. 또한, 부품의 모양을 알고 있으므로 해당 경계가 원호, 직선, 자유 곡선 인지를 알 수 있으므로 해당 형상 요소로 경계 곡선을 생성하면 된다. SurfART에서는 이러한 작업을 위하여 Fairing 메뉴에 직선, 원, 원호 등을 생성하는 기능을 갖고 있다. 그림 9.5 - 그림 9.7이 경계 곡선을 생성하는 과정을 나타낸 그림이다.

세 번째 과정은 곡면을 생성하는 작업이다. 곡면은 경계 곡선이나 단면 곡선을 생성하여 skinning으로 생성하면 된다. 그러나 점 데이터가 단면을 나타내도록 측정되지 않은 경우에는 점 데이터를 조정하거나 곡면을 근사하는 방법으로 생성해야 한다. SurfART는 평면, 원통면 등의 해석 곡면 생성 기능을 갖고 있다. 특히, 평면 생성 기능에는 점 데이터를 근사하여 생성할 수 있으므로 측정 데이터로부터 평면을 쉽게 찾아 낼 수 있다. 또한 자유 곡면을 생성하는 방법으로는 점 데이터의 보간(interpolation), skinning, sweeping, blending기능을 갖고 있다. 예제 모델은 면이 평면으로 되어 있고, 모서리가 블랜딩된 모양이므로, 먼저 점 데이터로부터 해당 평면들을 찾고, 경계 곡선을 평면에 투영하여 평면을 투영 곡선으로 트리밍하는 방법으로 모델링을 하였다. 그림 9.8과 그림 9.9능 평면을 찾아 경계 곡선을 평면에 투영하는 과정이고, 그림 9.10은 이러한 작업을 완료한 상태이다. 그림 9.11은 평면을 트리밍하기 위하여 투영된 경계 곡선을 분할하여 필요 없는 부분을 삭제하는 과정이고, 그림 9.12는 경계 곡선의 트리밍 작업이 완료된 그림이다. 그림 9.13은 경계 곡선으로 평면을 트리밍한 결과를 나타내는 그림이다. 그림 9.14는 평면의 경계 곡선을 단면 곡선으로 하여 skinning 곡면을 생성하여 평면 사이를 연결하는 과정이며, 그림 9.15는 이러한 과정을 완료한 그림이다. 또한, 그림 9.16은 모

델을 shading한 그림이다. 그림 9.17은 예제 모델의 일부분을 NC 가공하는 가공 경로를 나타낸 그림이다.

현재 SurfART는 디버깅 과정에 있어서 예제 모델을 완벽하게 모델링할 수 없었다. 특히, 블랜딩시 수반되는 트리밍에 오류가 많아 모델링 과정에서 블랜딩 곡면을 생성하지 못하였다. 블랜딩과 트리밍 기능은 주어진 조건에 따라 수행되지 않는 경우가 많아 알고리즘을 수정해야 한다. 또한, 많은 점 데이터부터 경계 곡선이나 평면을 찾아내는 기능을 보강해야 하며, 일정한 형식이 없이 측정된 데이터를 reduction하거나 filtering하는 기능도 추가해야만 현장에서 사용할 수 있을 것이다.

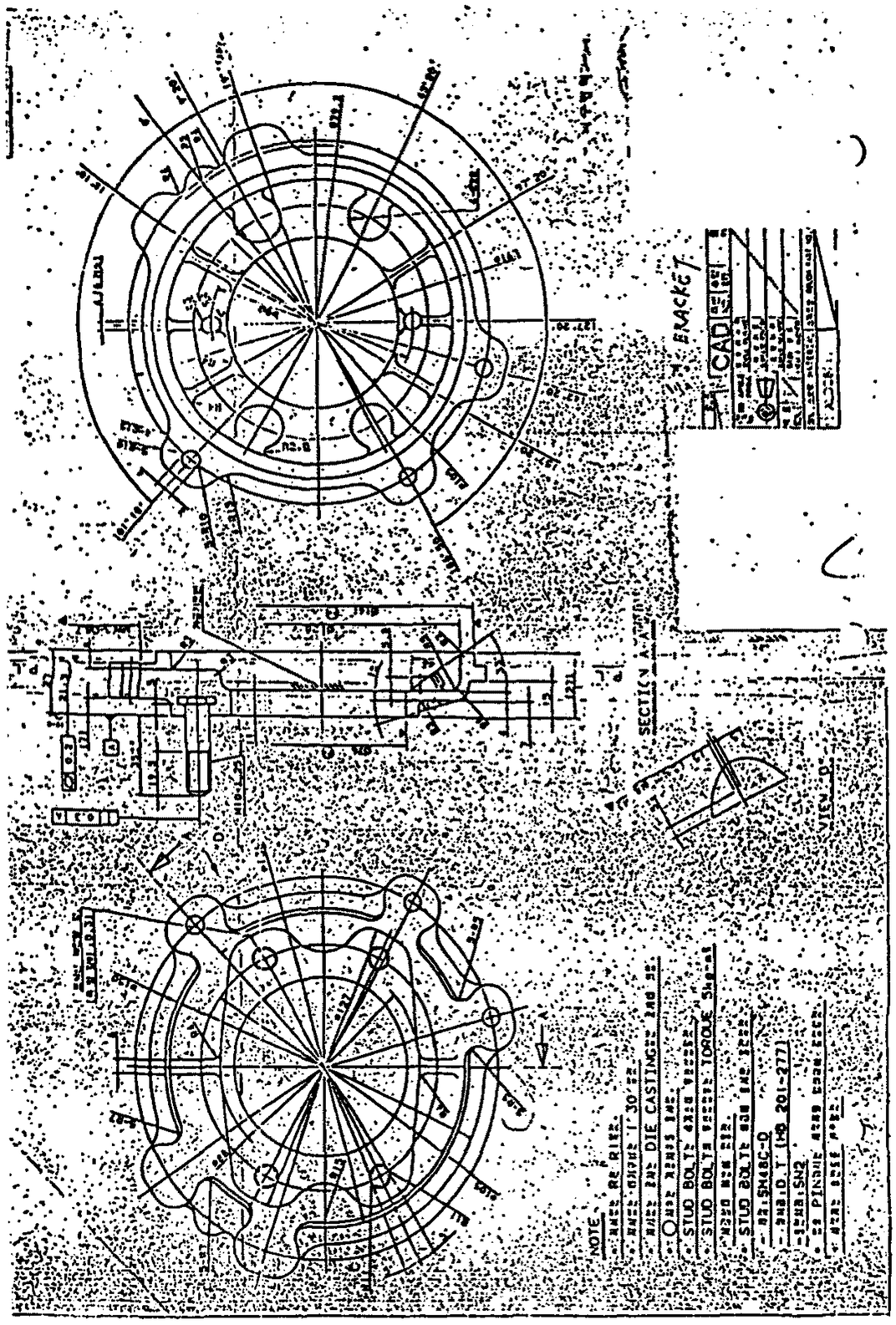


그림 9.1 예제 모델의 도면

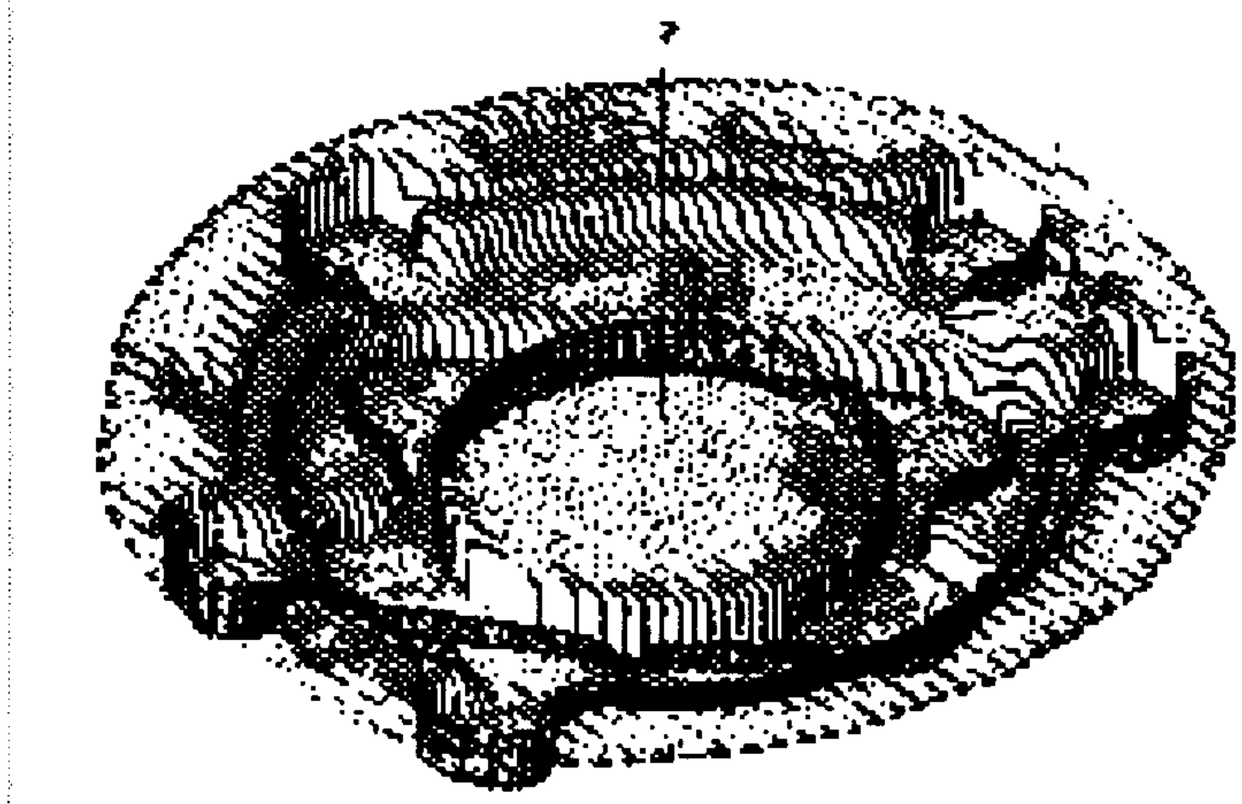


그림 9.2 브라켓을 측정한 점 데이터

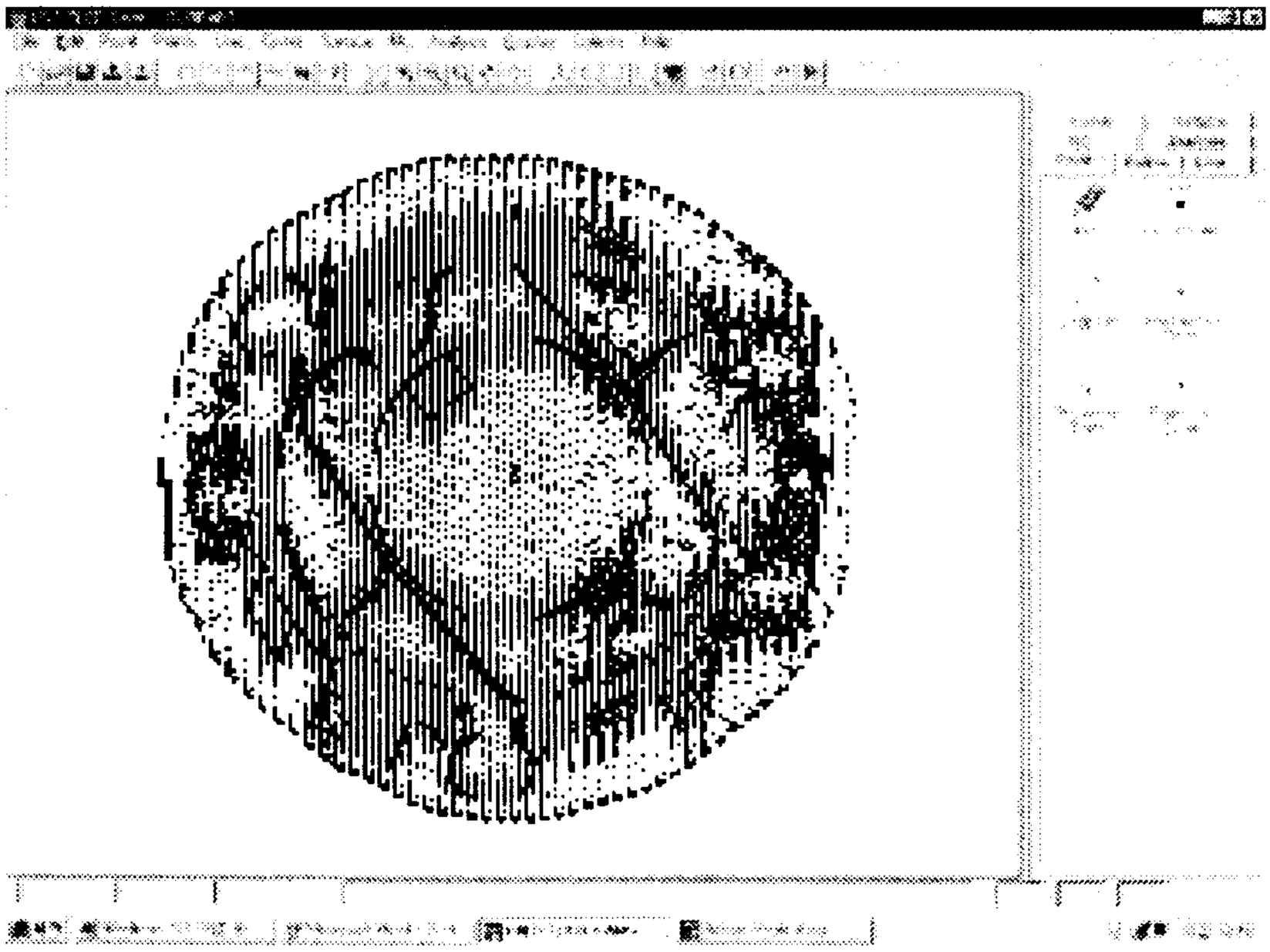


그림 9.3 X-Y평면에 투영한 점 데이터

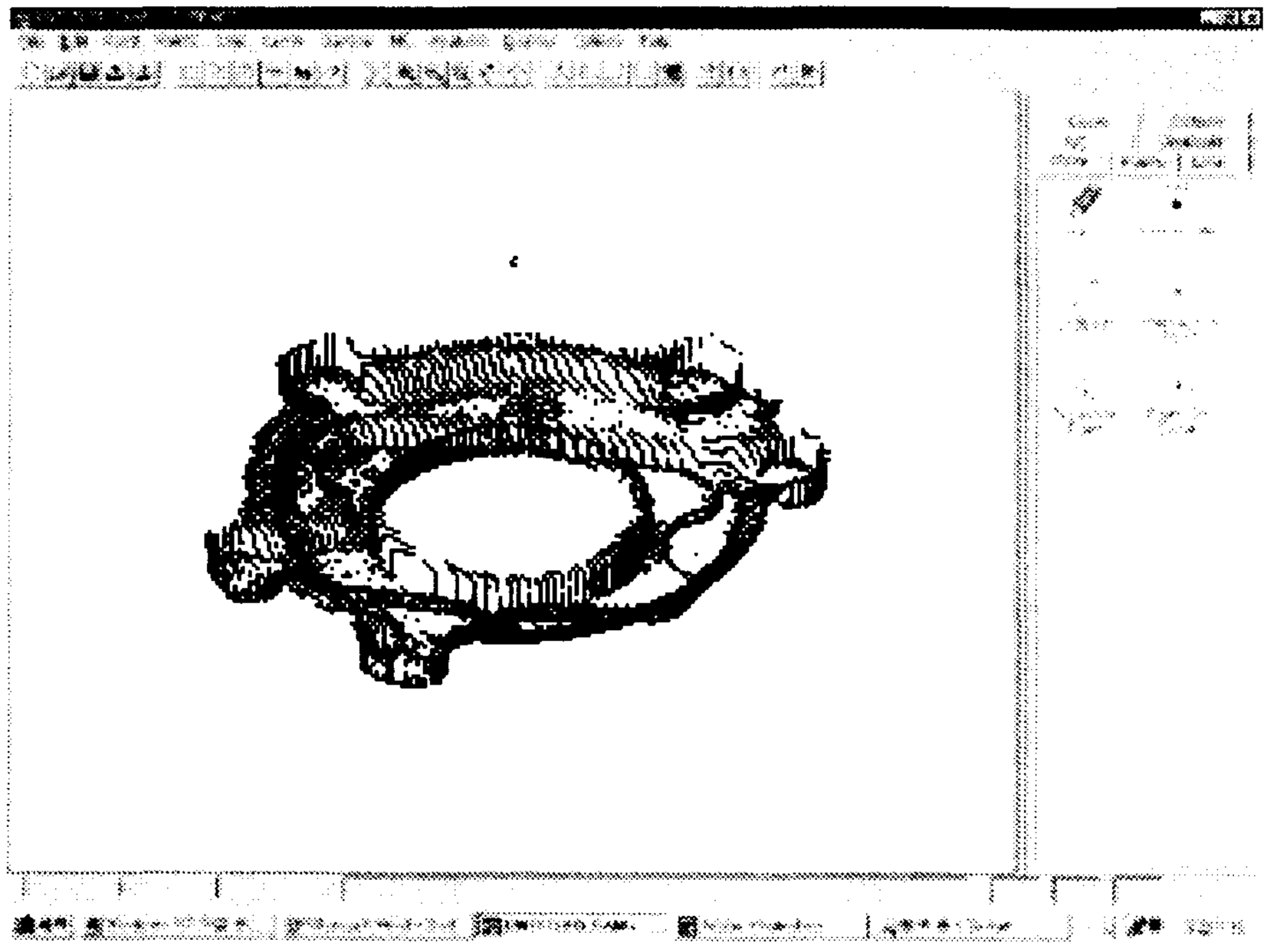


그림 9.4 측정 데이터의 reduction 과 fairing

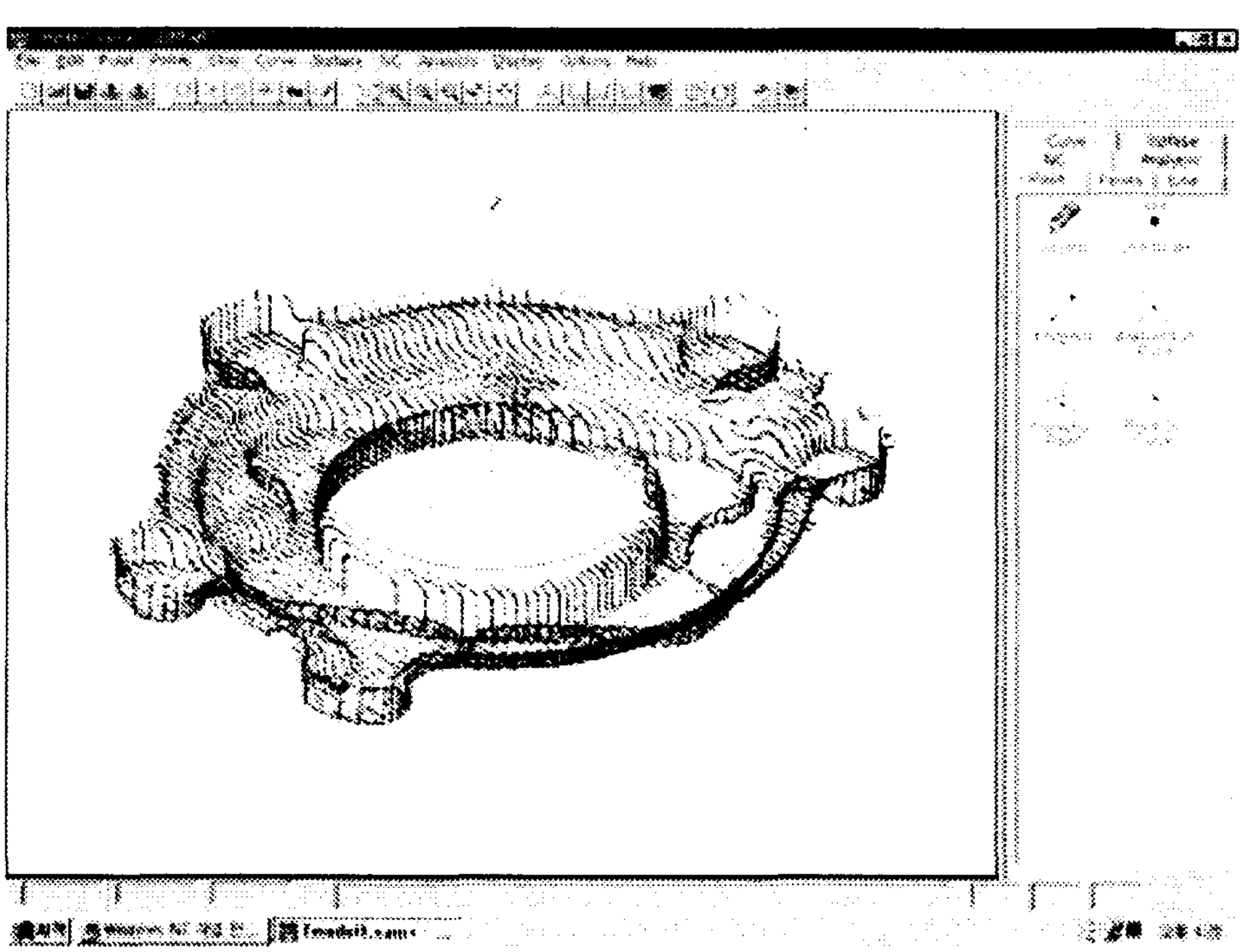


그림 9.5 경계 곡선의 생성 (I)

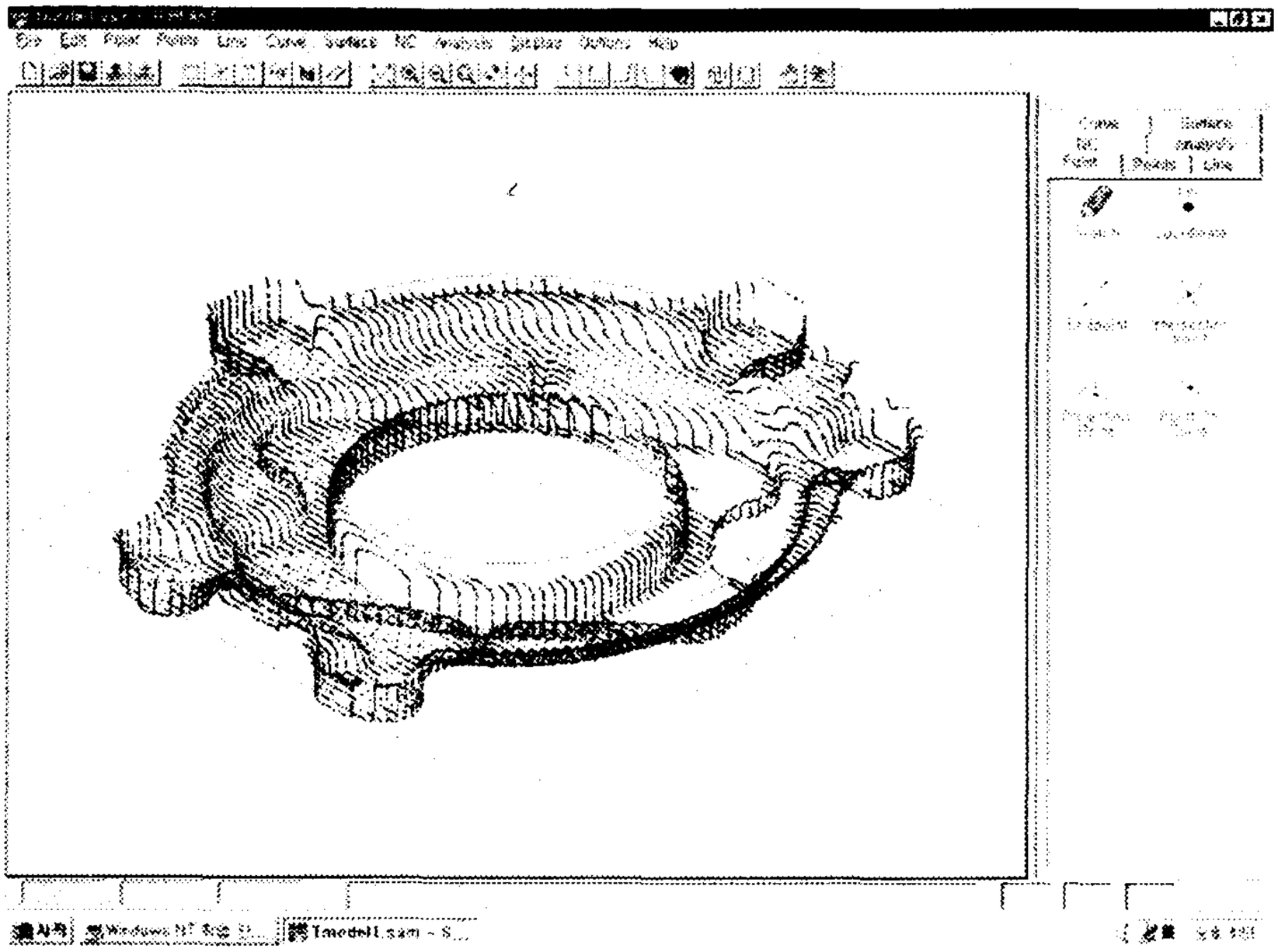


그림 9.6 경계 곡선의 생성 (II)

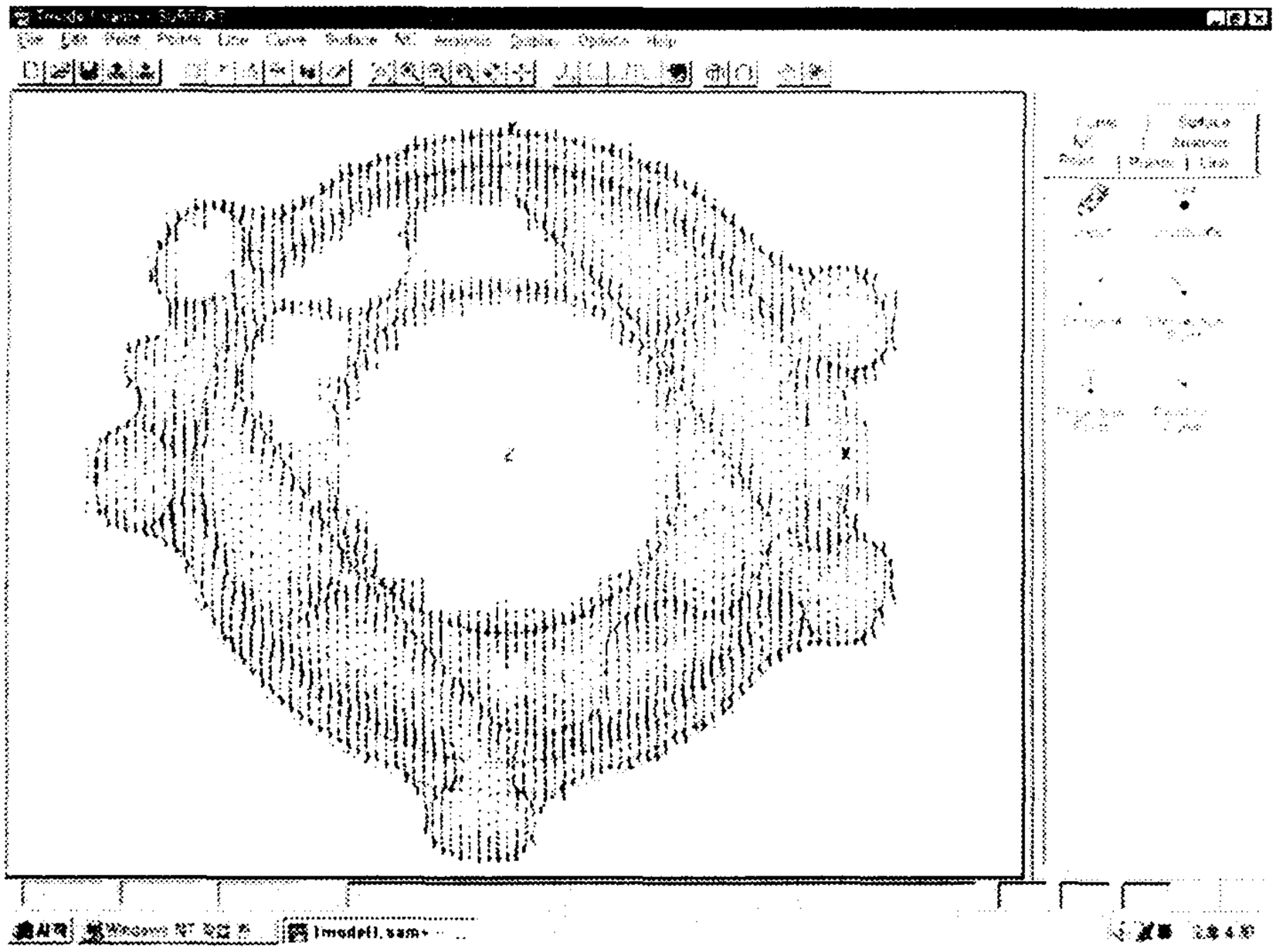


그림 9.7 경계 곡선의 생성 (III)

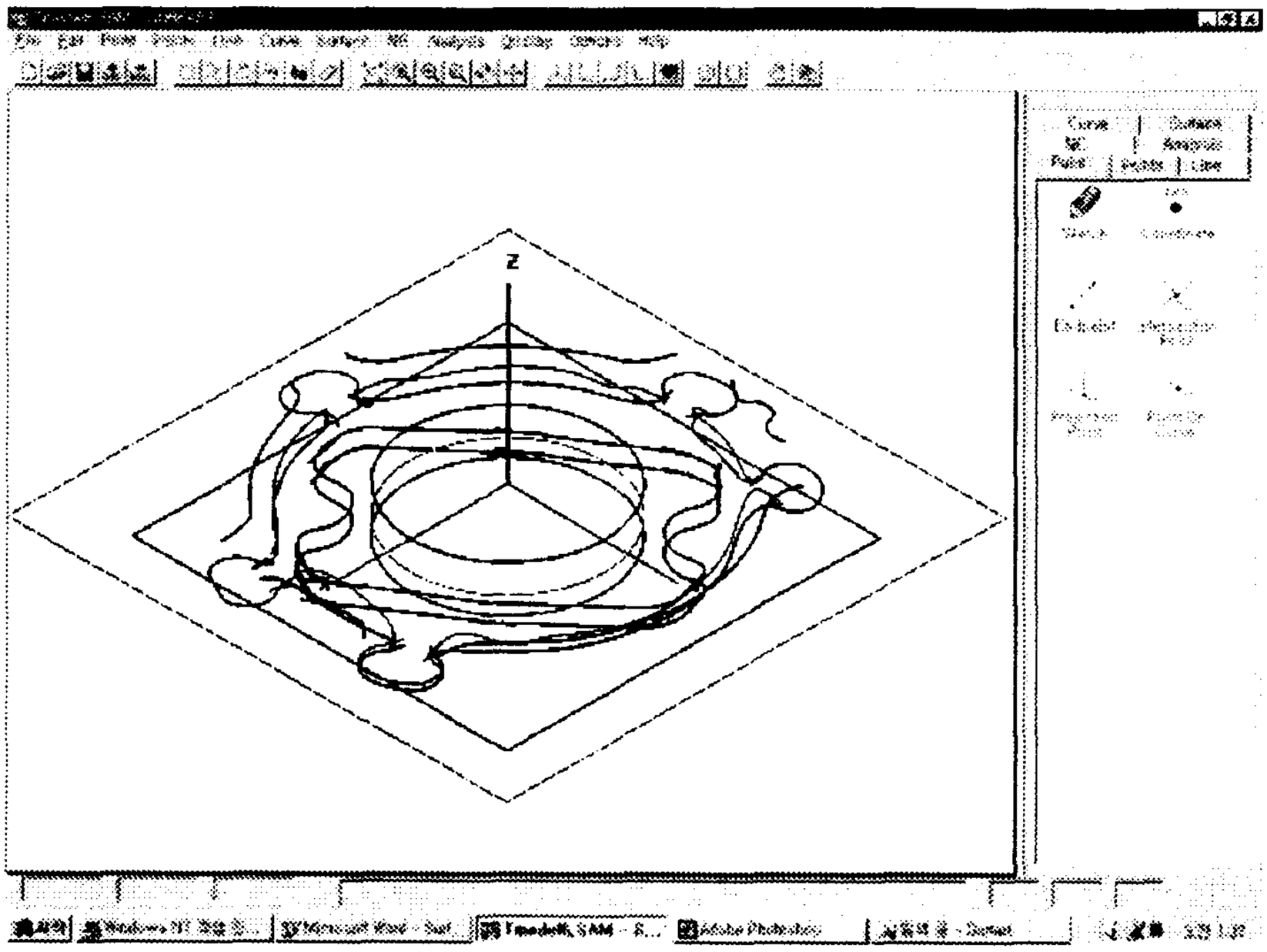


그림 9.8 평면에 투영된 경계 곡선 (I)

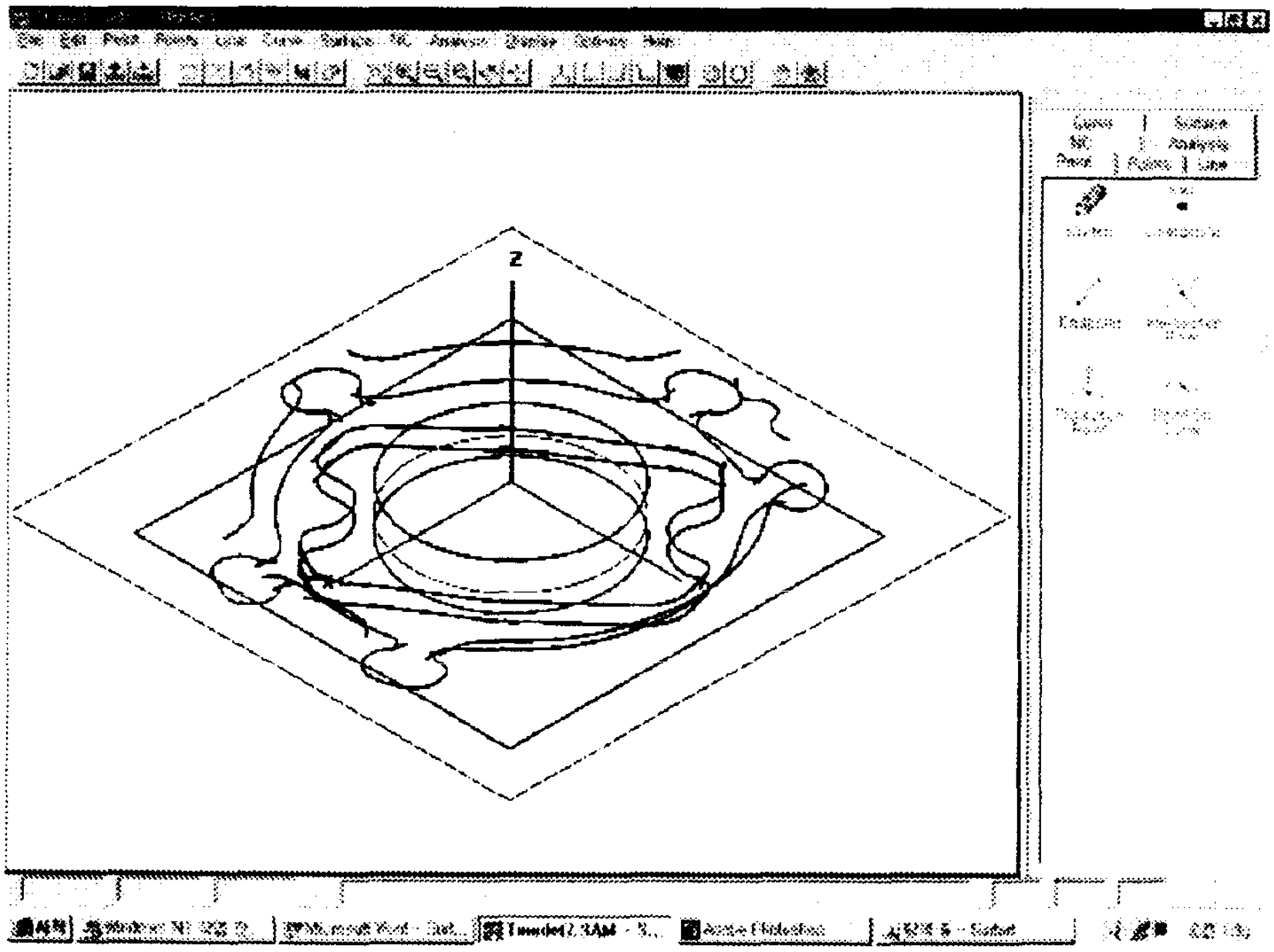


그림 9.9 평면에 투영된 경계 곡선 (II)

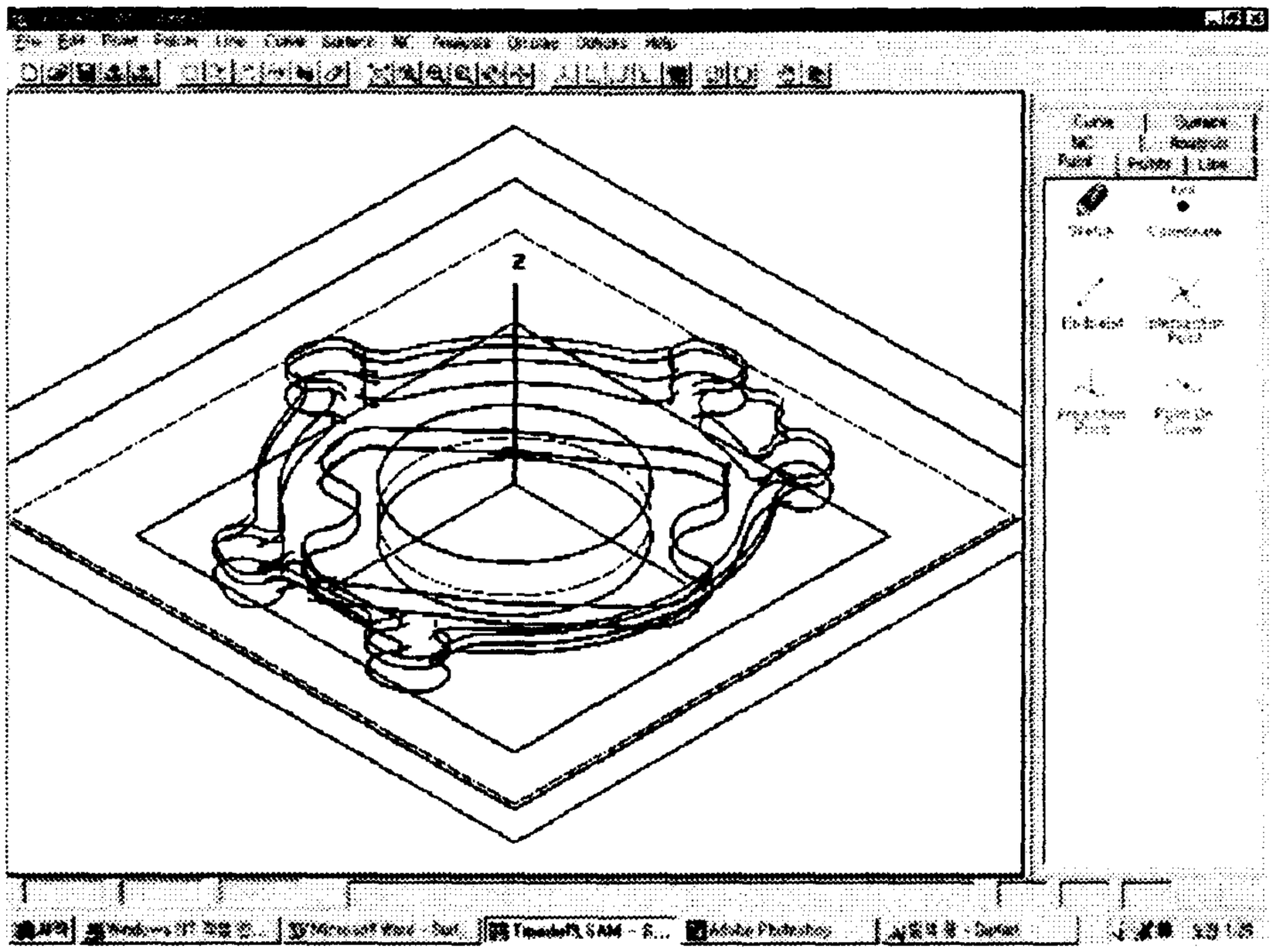


그림 9.10 경계 곡선 투영이 완료된 상태

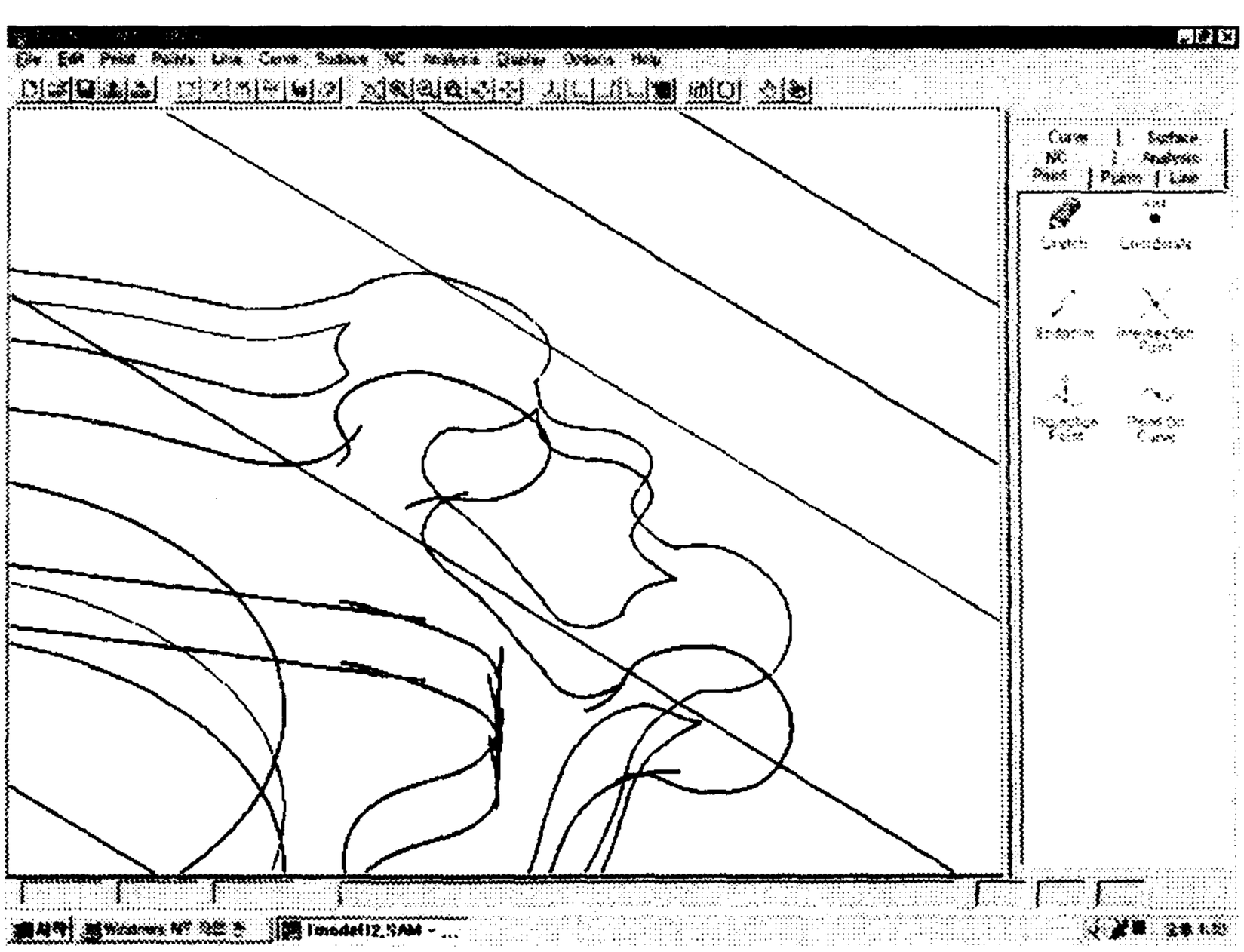


그림 9.11 경계 곡선의 트리밍

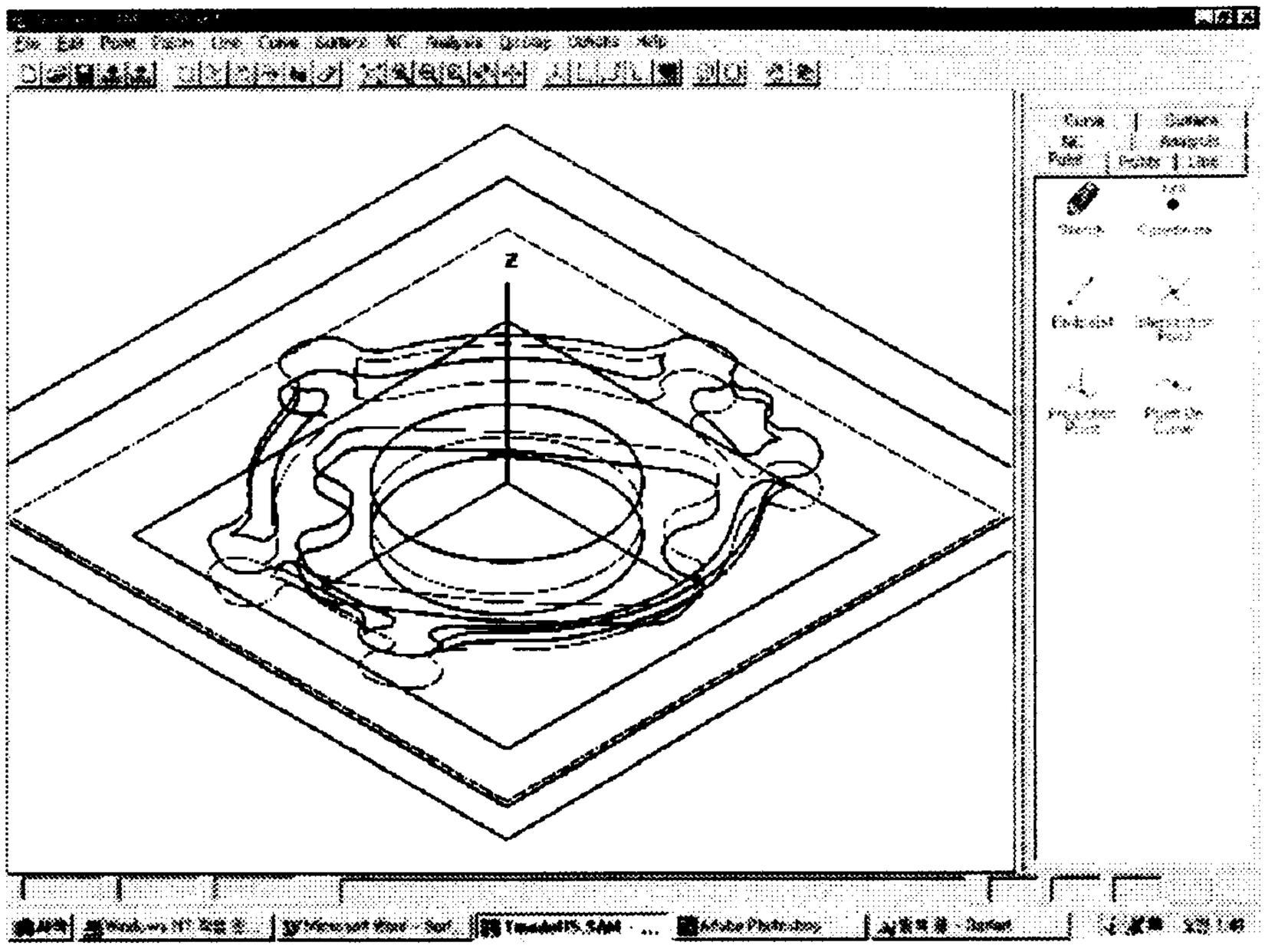


그림 9.12 경계 곡선의 트리밍이 완료된 상태

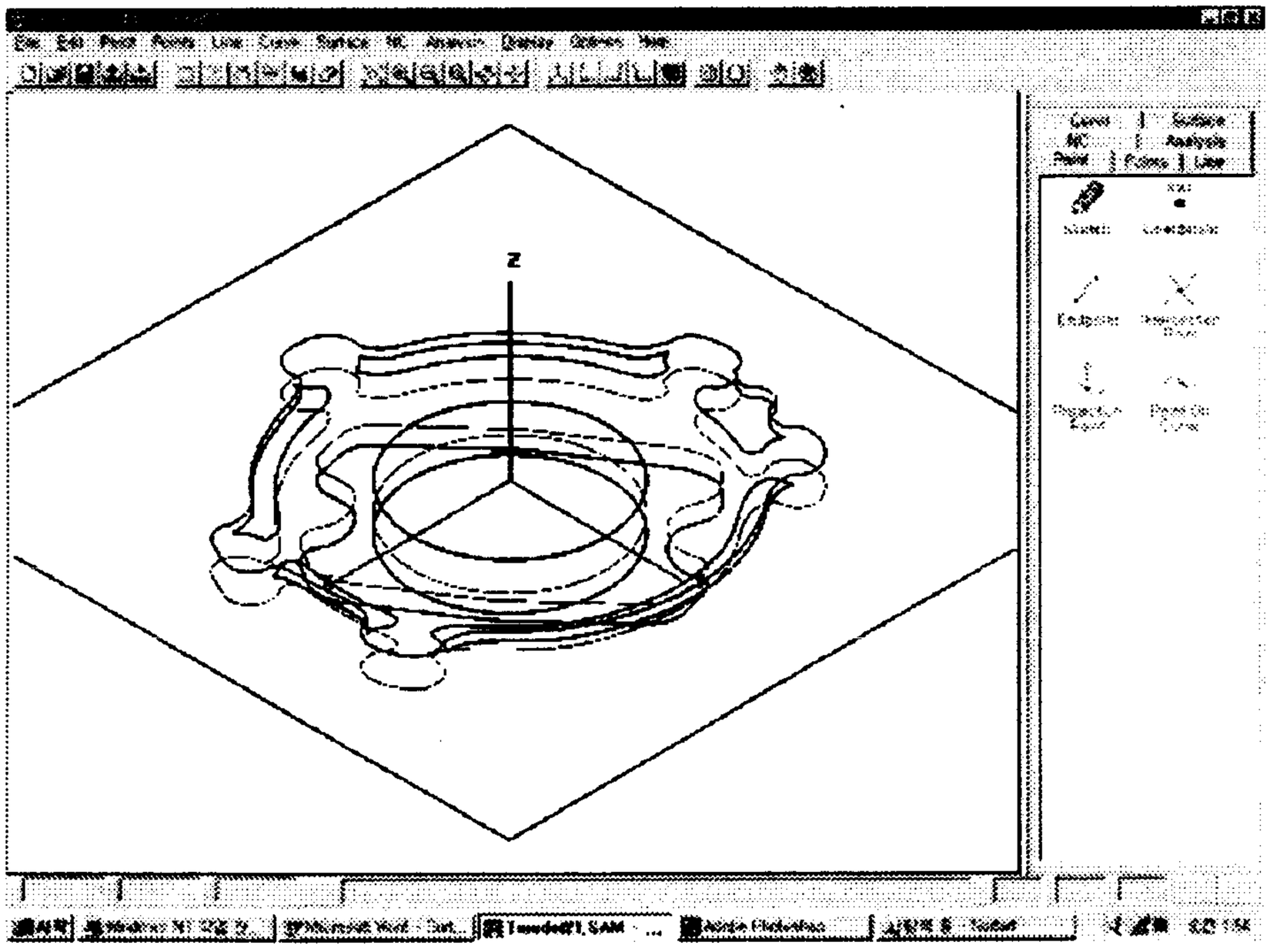


그림 9.13 평면의 트리밍

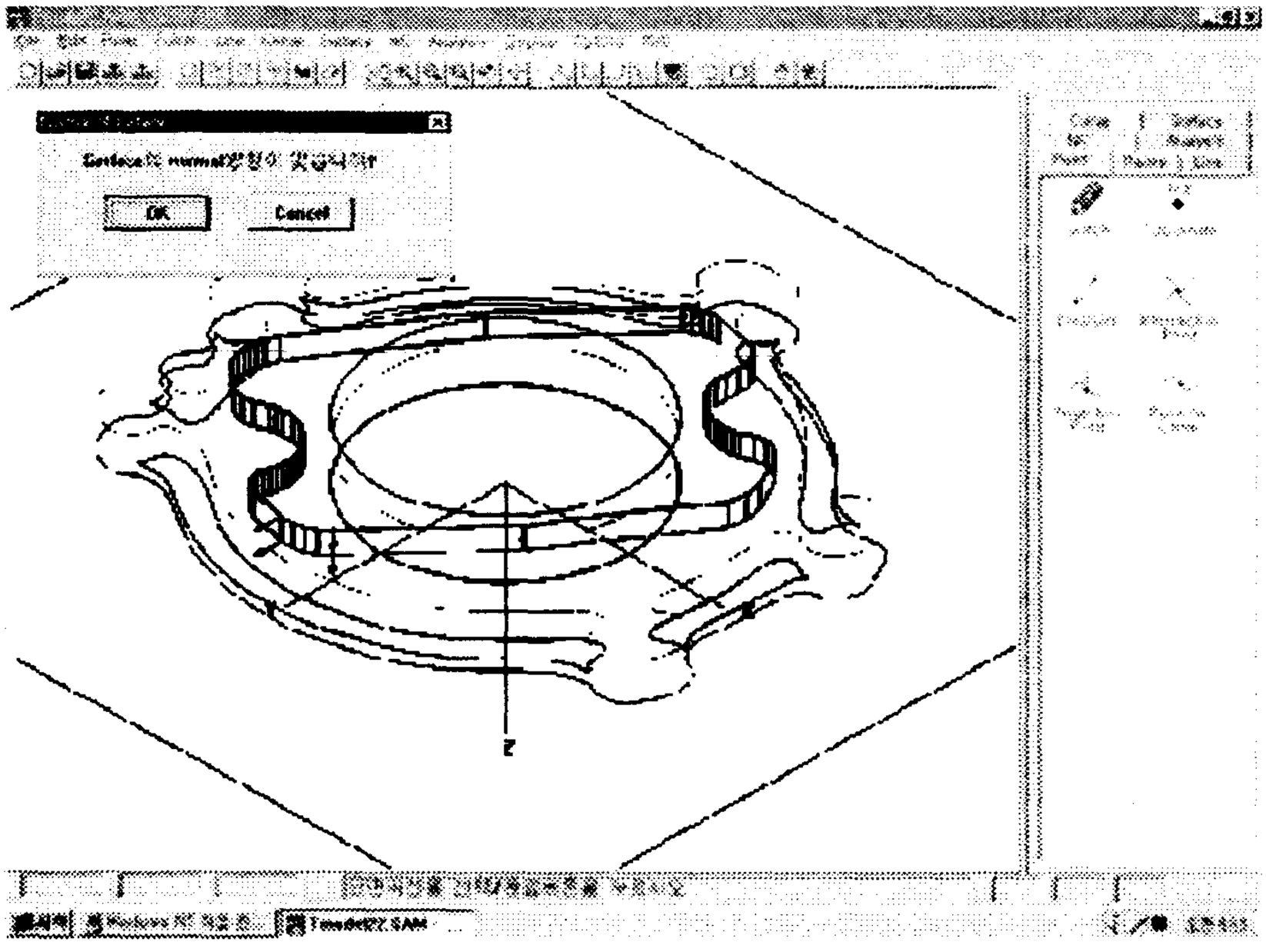


그림 9.14 Skining 곡면의 생성

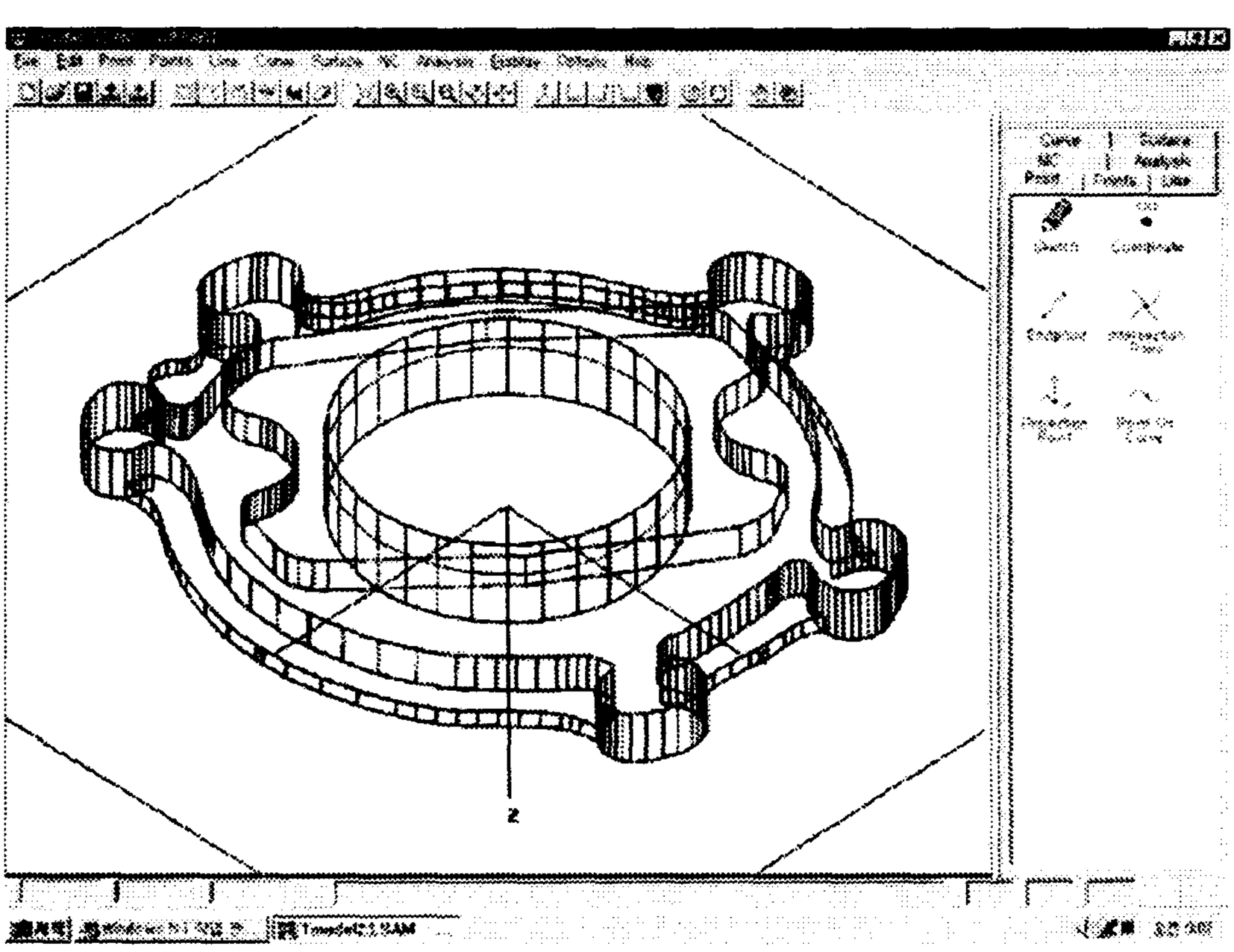


그림 9.15 모델의 완성

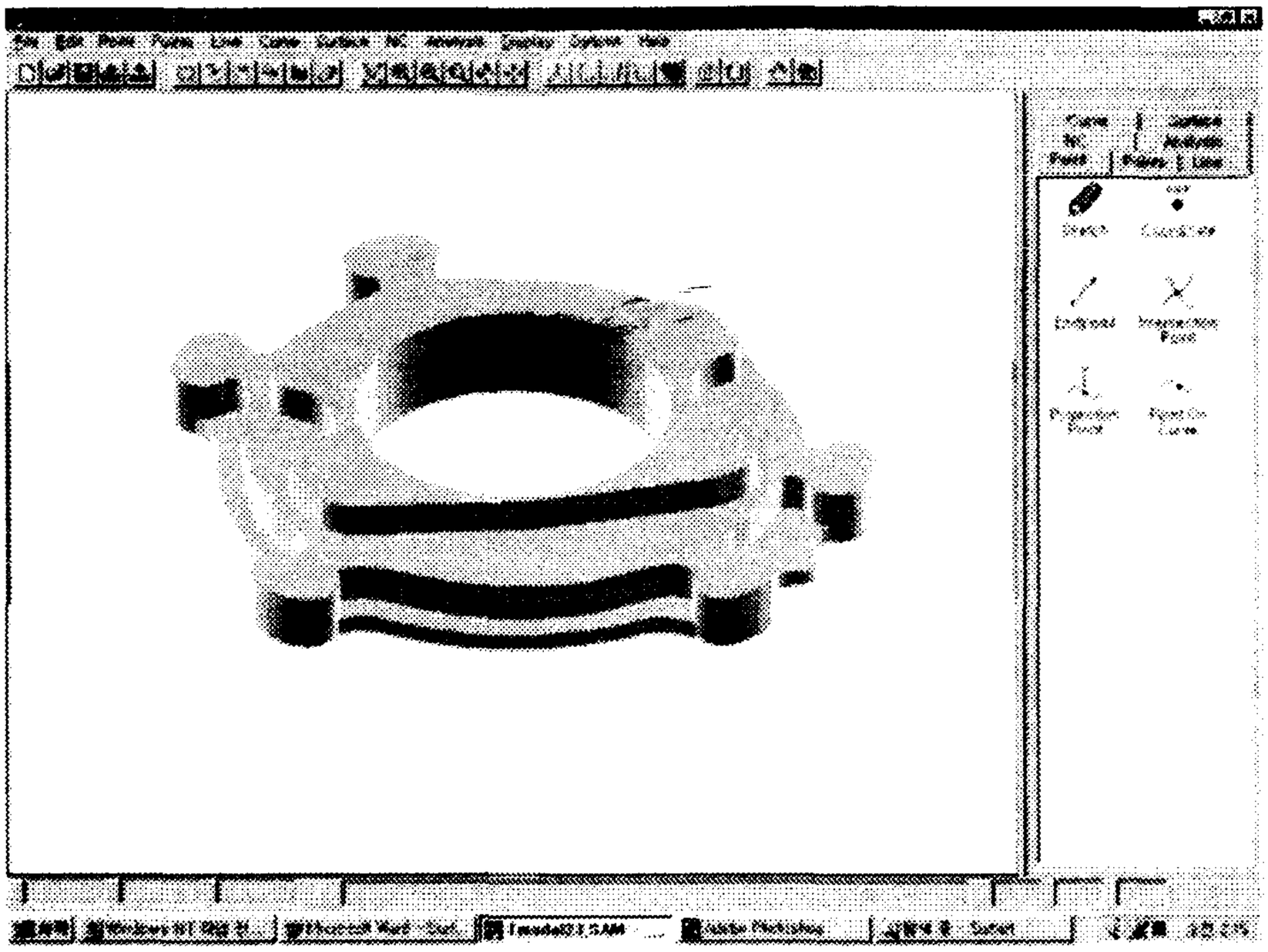


그림 9.16 Shading된 모델

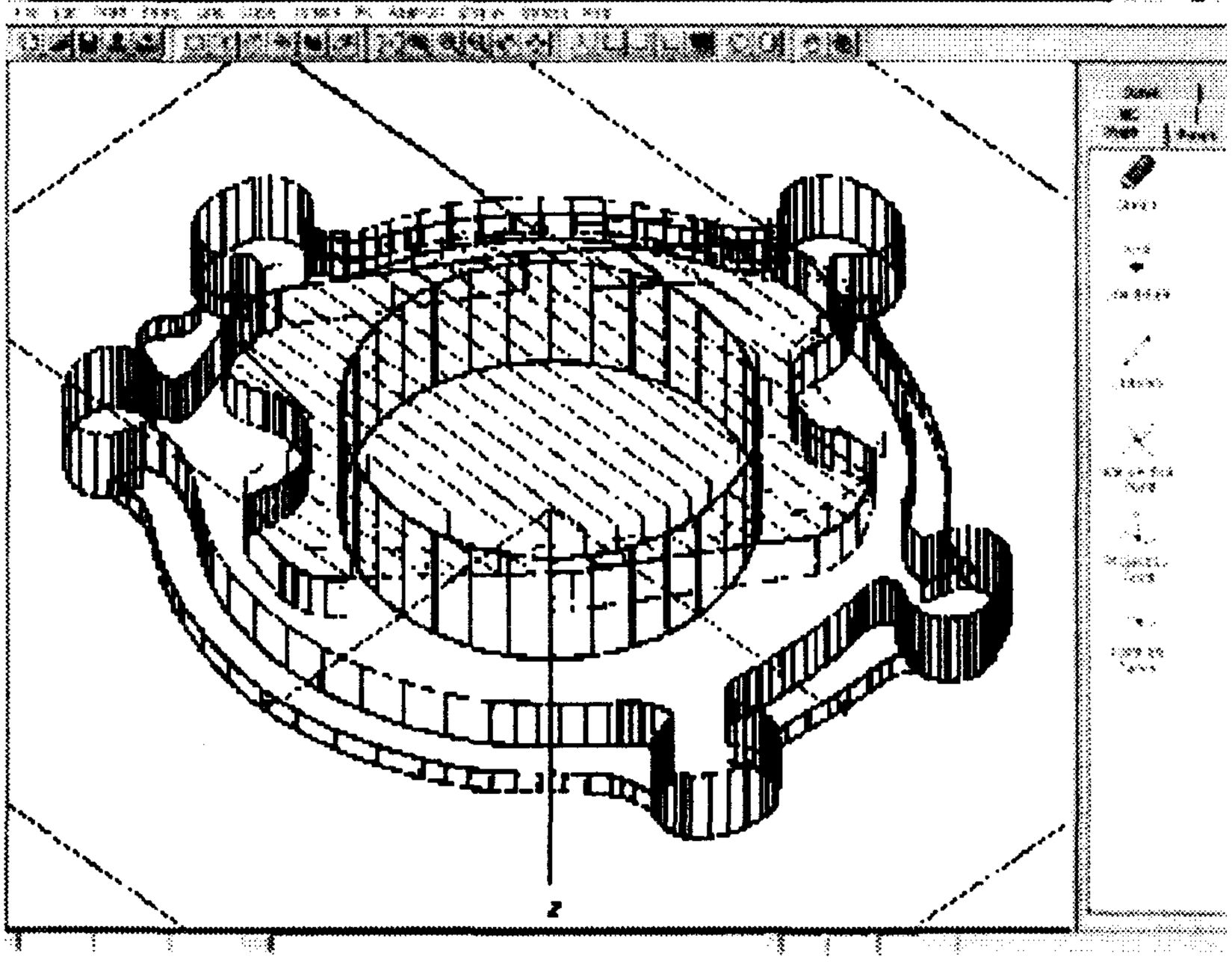


그림 9.17 NC 가공 경로의 생성

제 10 장 결론

본 연구는 금형의 제품 개발기간 단축을 위하여, 측정한 정보로부터 모델링을 하고, 설계해석을 한 후 형상을 최적화 하고 가공하며, 가공 상태를 Monitoring 하여 최적의 생산을 할 수 있도록 하기 위한 일련의 하드웨어 및 소프트웨어 개발을 위한 과제의 세부 과제로 진행 되었다. PC 환경에서 측정된 데이터를 이용하여 곡면을 모델링 하고 가공정보를 생성하는 소프트웨어를 개발하였다. WINDOWS 환경에서 OpenGL Graphics Library와 Visual C++ 프로그래밍 언어를 사용하여, 범용성을 가지는 소프트웨어를 개발하였다. 주요 기능으로는 측정데이터의 입력 및 처리, 곡선, 곡면 모델링, NC 기능이 포함되어 있다. 자료구조는 타 시스템과의 Interface 및 향 후 Solid Modeling으로의 확장을 고려하여 NURBS곡면을 형상정의의 기본으로 하고 Topology는 Non-manifold 구조를 갖도록 개발하였다. IGES, DXF, ZES 등을 통하여 타 시스템과의 형상정보 교환도 가능토록 하였다.

개발된 소프트웨어는 향 후 금형업체를 위한 NC전용 소프트웨어, 측정데이터 처리 소프트웨어, NC Controller의 Shop-floor Programming 소프트웨어, 특정제품의 모델링 및 가공 전용 소프트웨어 개발의 기반 소프트웨어로 쓰일 수 있도록 할 예정이다. 이를 위하여, 프로그램의 Documentation을 더욱 철저히 하고, 데이터 처리 및 계산의 효율을 향상시키며, 프로그램의 Error를 완전히 없애는 작업을 계속 수행할 것이다. 또한 본 소프트웨어를 대학의 연구팀에 나누어 주어 형상설계 관련 연구의 기반이 되도록 하고자 한다. 국내 CAD/CAM 수준에서 범용의 소프트웨어를 개발하여 판매하는 것은 경쟁력이 없으나 전용화되고 차별화된 소프트웨어를 개발하는 경우 국제 시장에서도 경쟁을 할 수 있을 것이다.

여 백

APPENDIX

A. 메뉴 ID와 메시지 핸들러 함수

B. 기본 모듈에 관련된 클래스와 API

C. 데이터 구조 및 파일에 관련된 클래스와 API

D. 측정 데이터의 fairing과 관련된 API

E. 응용 곡선, 곡면의 생성과 관련된 API

F. IGES, ZES 인터페이스 관련 API

G. NC에 관련된 중요한 함수

여 백

A. 메뉴 ID와 메세지 핸들러 함수

표2.6 메뉴 ID와 핸들러 함수

ID	함수 리스트	소속 클래스
ID_FILE_CUSTOMDRAW	OnFileCustomdraw	CSurfartView
ID_FILE_CUSTOMNEW	OnFileCustomdnew	CSurfartView
ID_FILE_CUSTOMOPEN	OnFileCustomopen	CSurfartView
ID_FILE_CUSTOMSAVE	OnFileCustomsave	CSurfartView
ID_FILE_SAVE_AS	OnFileSaveAs	CSurfartView
ID_FILE_EXPORT_POINT	OnFileExportPoint	CSurfartView
ID_FILE_IMPORT_DXF	OnFileImportDxf	CSurfartView
ID_FILE_IMPORT_IGES	OnFileImportIges	CSurfartView
ID_FILE_IMPORT_POINT	OnFileImportPoint	CSurfartView
ID_FILE_IMPORT_ZES	OnFileImportZes	CSurfartView
ID_EDIT_COLOR	OnEditColor	CSurfartView
ID_EDIT_DELETEONE	OnEditDeleteone	CSurfartView
ID_EDIT_DELETETYPE_CURVE	OnEditDeletetypeCurve	CSurfartView
ID_EDIT_DELETETYPE_PGROUP	OnEditDeletetypePgroup	CSurfartView
ID_EDIT_DELETETYPE_POINT	OnEditDeletetypePoint	CSurfartView
ID_EDIT_DELETETYPE_SURFACE	OnEditDeletetypeSurface	CSurfartView
ID_EDIT_TRANSFORM_ROTATION	OnEditTransformRotation	CSurfartView
ID_EDIT_TRANSFORM_SCALE	OnEditTransformScale	CSurfartView

ID_EDIT_TRANSFORM_TRANSLATION	OnEditTransformTranslation	CSurfartView
ID_POINT_KEYBOARD	OnPointKeyboard	CSurfartView
ID_POINTS_AVERAGEFILTERING	OnPointsAveragefiltering	CSurfartView
ID_POINTS_DATAREDUCTION	OnPointsDatareduction	CSurfartView
ID_POINTS_FAIRING_CIRCLE_3DPLANE	OnPointsFairingCircle3dplane	CSurfartView
ID_POINTS_FAIRING_CIRCLE_XYPLANE	OnPointsFairingCircleXyplane	CSurfartView
ID_POINTS_FAIRING_CIRCLE_YZPLANE	OnPointsFairingCircleYzplane	CSurfartView
ID_POINTS_FAIRING_CIRCLE_ZXPLANE	OnPointsFairingCircleZxplane	CSurfartView
ID_POINTS_FAIRING_CIRCULAR_ARC_3DPLANE	OnPointsFairingCircularArc3dplane	CSurfartView
ID_POINTS_FAIRING_CIRCULAR_ARC_XYPLANE	OnPointsFairingCircularArcXyplane	CSurfartView
ID_POINTS_FAIRING_CIRCULAR_ARC_YZPLANE	OnPointsFairingCircularArcYzplane	CSurfartView
ID_POINTS_FAIRING_CIRCULAR_ARC_ZXPLANE	OnPointsFairingCircularArcZxplane	CSurfartView
ID_POINTS_FAIRING_PLANE_3DPLANE	OnPointsFairingPlane3dplane	CSurfartView

ID_POINTS_FAIRING_STRAIGHT_3DPLANE	OnPointsFairingStraight3dplane	CSurfartView
ID_POINTS_GAUSSIANFILTERING	OnPointsGaussianfiltering	CSurfartView
ID_POINTS_MEDIANFILTERING	OnPointsMedianfiltering	CSurfartView
ID_LINE_OFFSET	OnLineOffset	CSurfartView
ID_LINE_POINTTOPOINT	OnLinePointtopoint	CSurfartView
ID_LINE_POINTTOPOINT_SELECT	OnLinePointtopointSelect	CSurfartView
ID_LINE_POINT_DIRECTION	OnLinePointDirection	CSurfartView
ID_LINE_POINT_DIRECTION_SELECT	OnLinePointDirectionSelect	CSurfartView
ID_CURVE_ARC_RAD_CEN	OnCurveArcRadCen	CSurfartView
ID_CURVE_ARC_STPT_ENDPT	OnCurveArcStptEndpt	CSurfartView
ID_CURVE_CIRCLE_RAD_CEN	OnCurveCircleRadCen	CSurfartView
ID_CURVE_CIRCLE_THREE_PTS	OnCurveCircleThreePts	CSurfartView
ID_CURVE_CIRCLE_THREE_PTS_SELECT	OnCurveCircleThreePtsSelect	CSurfartView
ID_CURVE_CURVEONASURFACE	OnCurveCurveonasurface	CSurfartView
ID_CURVE_EDGE_CLOSURE	OnCurveEdgeClosure	CSurfartView
ID_CURVE_ELLIPSE_RAD_CEN	OnCurveEllipseRadCen	CSurfartView
ID_CURVE_INTERPOLATION_GPOINT	OnCurveInterpolationOfPoints	CSurfartView
ID_CURVE_INTERPOLATION_PGROUP	OnCurveInterpolationOfPgroup	CSurfartView
ID_CURVE_INTERSECTION	OnCurveIntersection	CSurfartView

ID_CURVE_TOOLS_SPLIT	OnCurveToolsSplit	CSurfartView
ID_SURFACE_CONSTANT_RADIUS_BLENDING	OnSurfaceConstantRadiusBlending	CSurfartView
ID_SURFACE_CORNER_BLENDING	OnSurfaceCornerBlending	CSurfartView
ID_SURFACE_CYLINDER_BASE_TOP	OnSurfaceCylinderBaseTop	CSurfartView
ID_SURFACE_CYLINDER_TWO_CIRCLE	OnSurfaceCylinderTwoCircle	CSurfartView
ID_SURFACE_INTERPOLATION	OnSurfaceInterpolation	CSurfartView
ID_SURFACE_NORMAL_SWEEPING	OnSurfaceNormalSweeping	CSurfartView
ID_SURFACE_OF_REVOLUTION	OnSurfaceOfRevolution	CSurfartView
ID_SURFACE_PARALLEL_SWEEPING	OnSurfaceParallelSweeping	CSurfartView
ID_SURFACE_PLANE_POINT_NORMAL	OnSurfacePlanePointNormal	CSurfartView
ID_SURFACE_PLANE_THREE_PTS	OnSurfacePlaneThreePts	CSurfartView
ID_SURFACE_PLANE_THREE_PTS_SELECTION	OnSurfacePlaneThreePtsSelection	CSurfartView
ID_SURFACE_RULED	OnSurfaceRuled	CSurfartView
ID_SURFACE_SKINNING	OnSurfaceSkinningOfCurves	CSurfartView
ID_SURFACE_SYNCHRONIZED_SWEEPING	OnSurfaceSynchronizedSweeping	CSurfartView
ID_SURFACE_TRIMMING	OnSurfaceTrimming	CSurfartView
ID_DISPLAY_MODE_FULLSHADING	OnDisplayModeFullshading	CSurfartView

ID_DISPLAY_MODE_QUICKSHADIN G	OnDisplayModeQuickshading	CSurfartView
ID_DISPLAY_MODE_WIREFRAME	OnDisplayModeWireframe	CSurfartView
ID_DISPLAY_VIEW_FRONT	OnDisplayViewFront	CSurfartView
ID_DISPLAY_VIEW_PERSPECTIVE	OnDisplayViewPerspective	CsurfartView
ID_DISPLAY_VIEW_RIGHT	OnDisplayViewRight	CsurfartView
ID_DISPLAY_VIEW_TOP	OnDisplayViewTop	CsurfartView
ID_DISPLAY_VIEW_USER	OnTransformationLookat	CsurfartView
ID_DISPLAY_ZOOM_ZOOMALL	OnTransformationZoomin	CsurfartView
ID_DISPLAY_ZOOM_ZOOMIN	OnDisplayZoomZoomin	CsurfartView
ID_DISPLAY_ZOOM_ZOOMOUT	OnDisplayZoomZoomout	CsurfartView
ID_DISPLAY_ZOOM_PREVIOUS	OnDisplayZoomPrevious	CsurfartView
ID_DISPLAY_ZOOM_ROTATION	OnDisplayZoomRotation	CsurfartView
ID_DISPLAY_ZOOM_PANNING	OnDisplayZoomPanning	CsurfartView
ID_DISPLAY_ZOOM_MOVE_LEFT	OnDisplayZoomMoveLeft	CsurfartView
ID_DISPLAY_ZOOM_MOVE_RIGHT	OnDisplayZoomMoveRight	CsurfartView
ID_DISPLAY_ZOOM_MOVE_UP	OnDisplayZoomMoveUp	CsurfartView
ID_DISPLAY_ZOOM_MOVE_DOWN	OnDisplayZoomMoveDown	CsurfartView
ID_DISPLAY_ELEMENTS_ALL	OnDisplayElementsAll	CsurfartView
ID_DISPLAY_ELEMENTS_CURVES	OnDisplayElementsCurves	CsurfartView
ID_DISPLAY_ELEMENTS_POINTS	OnDisplayElementsPoints	CsurfartView
ID_DISPLAY_ELEMENTS_SELECTI ON	OnDisplayElementsSelectio n	CsurfartView

ID_DISPLAY_ELEMENTS_SURFACE S	OnDisplayElementsSurfaces	CsurfartView
ID_NC_CARTESIAN_CUT	OnNcCartesianCut	CsurfartView
ID_NC_CONTOUR_CUT	OnNcContourCut	CsurfartView
ID_NC_FINISH_CARTESIAN_CUT	OnNcFinishCartesianCut	CsurfartView
ID_NC_FINISH_PARAMETRIC_CUT	OnNcFinishParametricCut	CsurfartView
ID_NC_POST_PROCESS	OnNcPostProcess	CsurfartView
ID_NC_SIMULATE_FINISH	OnNcSimulateFinish	CsurfartView
ID_NC_SIMULATE_MODEL	OnNcSimulateModel	CsurfartView
ID_NC_SIMULATE_ROUGH	OnNcSimulateRough	CsurfartView
ID_NC_SIMULATE_STOP	OnNcSimulateStop	CsurfartView
ID_NC_SURFACE_SELECT	OnNcSurfaceSelect	CsurfartView
ID_ENDSEL	OnEndSelection	CsurfartView
ID_SELCANCEL	OnselCancel	CsurfartView
ID_OPTIONS_COLORS	OnOptionsColors	CsurfartView
ID_OPTIONS_SELETIONMETHOD	OnOptionsSeletionmethod	CsurfartView
ID_APP_EXIT	OnAppExit	CsurfartView
ID_APP_ABOUT	OnAppAbout	CsurfartApp
ID_FILE_CUSTOMOPEN	OnFileCustomopen	CsurfartApp
ID_FILE_CUSTOMLIST	OnFileCustomlist	CsurfartApp

표2.7 아이콘 ID

Icon ID	ImageList의 Resource	해당되는 ListCtrl
IDI_SKETCH, IDI_COORDINATE, IDI_ENDPOINT, IDI_INTERSECTION, IDI_PROJECTION, IDI_POINTONCURVE,	nPointResource	pPointCtrl
IDI_REDUCTION, IDI_FILTERAVERAGE, IDI_FILTERMEDIAN, IDI_FILTERGAUSSIAN, IDI_FITSTRAINGHTXY, IDI_FITSTRAINGHTYZ, IDI_FITSTRAINGHTZX, IDI_FITSTRAINGHT3D, IDI_FITCIRCLE, IDI_FITARCXY, IDI_FITARCYZ, IDI_FITARCZX, IDI_FITARC3D, IDI_FITPLANE, IDI_SEGMENTATION, IDI_MERGE, IDI_ADDPOINT, IDI_POJECTTOPLANE, IDI_PROJECTTOBESTPLANE,	nPointsResource	pPointsCtrl

IDI_POINTCOORDINATE, IDI_POINTSELECTION, IDI_DIRECTIONCOORDINATE, IDI_DIRECTIONSELECTION, IDI_HORIZONTAL, IDI_VERTICAL, IDI_ANGLE, IDI_TANGENT, IDI_POOFFSET, IDI_SPLIT, IDI_REVERSE, IDI_EXTENTION,	nLineResource	pLineCtrl

IDI_RADIUSCENTER, IDI_COORDINATEPOINTS, IDI_SELECTINPOINTS, IDI_ARCCENTER, IDI_STARTENDPOINTS, IDI_ELLIPSE, IDI_FITTING, IDI_INTERPOLATIONPOINTS, IDI_INTERPOLATIONPGROUP, IDI_CURVEINTERSECTION, IDI_SURFACEBOUNDARY, IDI_ISOPARAMETRIC, IDI_CURVEPROJECTION, IDI_OFFSET, IDI_EDGECLOSURE, IDI_TOOLSPLIT, IDI_TOOLSJOIN, IDI_TOOLBLEND, IDI_TOOLSFAIRING, IDI_TOOLSREVERSE, IDI_TOOLSEXTENTION, IDI_TOOLSCONTROLPOINTS, IDI_TOOLSINSERTREMOVEKNOTS	nCurveResource	pCurveCtrl

IDI_PLANEPOINTNORMAL, IDI_PLANEPOINTSCOORDINATE, IDI_PLANEPOINTSSELECTION, IDI_CYLINDERBASETOP, IDI_CYLINDERTWOCIRCLE, IDI_SURFACEFITTING, IDI_SURFACEINTERPOLATION, IDI_BOUNDARYCURVESPOINTS, IDI_COONS, IDI_RULED, IDI_SKINNING, IDI_TRIMMING, IDI_BLENDINGEDGE, IDI_BLENDINGCORNER, IDI_SURFACEOFREVOLUTION, IDI_SWEEPINGPARALLEL, IDI_SWEEPINGNORMAL, IDI_SWEEPINGSYNCHRONIZED, IDI_SURFACEOFFSET, IDI_SURFACEEXTENTION, IDI_SURFACEFAIRING, IDI_SURFACEREVERSE, IDI_CONTROLPOINTS, IDI_INSERTREMOVEKNOTS,	nSurfaceResource	pSurfaceCtrl

IDI_SELECTSURFACE, IDI_ROUGHCUTCONTOUR, IDI_ROUGHCUTCARTESIAN, IDI_FINISHCUTCARTESIAN, IDI_FINISHCUTPARAMETRIC, IDI_SIMULATEMODEL, IDI_SIMULATEROUGH, IDI_SIMULATEFINISH, IDI_SIMULATESTOP, ID_POSTPROCESS,	nNCResource	pNCCtrl
IDI_ANGULARDEVIATION, IDI_CHORDALDEVIATION, IDI_COORDINATE, IDI_DISTANCE, IDI_TANGENT, IDI_CURVATURE, IDI_RADIUSOFCURVATURE,	nAnalysisResource	pAnalysisCtrl

표2.8 다이얼로그 박스 ID와 사용용도

다이얼로그 박스 ID	사 용 용 도
IDD_ABOUTBOX	Help 메뉴에서 About Surface항목 지원
IDD_CIRCLE	원중심과 반지름을 이용한 circle 생성
IDD_CIRCLE_ARC_CENTER_RAD	원중심과 반지름을 이용한 원호 생성
IDD_CIRCLE_ARC_ST_END	시작점과 끝점을 이용한 원호 생성
IDD_CIRCLE_THREE_PT	세점을 이용한 circle생성

IDD_CORNER_BLENDING	Conrner Blending
IDD_CRE_BLENDING	Constant Radius Edge Blending
IDD_CURVE_PARAMETER	곡선상에 파라메터값으로 점을 생성
IDD_CURVE_PROJECTION	곡선을 곡면에 투영하기 위한 방향입력
IDD_CYLINDER_RAD_HEIGHT	반지름과 높이를 이용한 실린더 생성
IDD_ELLIPSE	타원 생성
IDD_FINISH_CUT_CARTESIAN	Cartesian Finish Cut
IDD_FINISH_CUT_PARAMETRIC	Parametric Finish Cut
IDD_LINE_DIR	시작점과 방향벡터를 이용한 직선 생성
IDD_LINE_OFFSET	offset 벡터를 이용한 직선 생성
IDD_LOOKAT	물체를 보는 방향 지정
IDD_NORMAL_CURVE	곡선의 탄젠트 방향 지정
IDD_NORMAL_SURFACE	곡면의 normal 방향 지정
IDD_POINT_KEYBOARD	x,y,z좌표값으로 point 생성
IDD_POST_PROCESSING	대상 기계 지정
IDD_ROTATE	형상요소의 회전 조정
IDD_ROUGH_CUT_CARTESIAN	Cartesian Rough Cut
IDD_ROUGH_CUT_CONTOUR	Contour Rough Cut
IDD_SCALE	형상요소의 크기 조정
IDD_STRAIGHT	직선 생성
IDD_STRAIGHT_PT_DIREC	시작점과 방향벡터를 이용한 직선 생성
IDD_SWEEP_NORMAL	Normal Sweep
IDD_SWEEP_PARALLEL	Parallel Sweep
IDD_SWEEP_ROTATIONAL	Rotational Sweep

IDD_SWEEP_SYNCRONIZED	SynCronized Sweep
IDD_TRANSLATE	형상요소의 위치 조정
IDD_PICKINGMETHOD	picking 단위 지정
IDD_PLANE	평면 생성
IDD_PLANE_THREE_PT	세점으로 이용한 평면 생성
IDD_OPTION_COLOR	global, shading, selection color 지정
IDD_MYDIALOGBAR	리스트 콘트롤이 있는 다이얼로그바
IDD_STATUS	상태바를 위한 다이얼로그바

표2.9 다이얼로그 박스를 위한 클래스와 화일

다이얼로그 박스 ID	클 래 스	베이스클래스	헤더 화일	시행 화일
IDD_ABOUTBOX	CaboutDlg	Cdialog	-	surface.cpp
IDD_CIRCLE	CcircrDlg	Cdialog	circrdlg.h	circrdlg.cpp
IDD_CIRCLE_ARC_CENTER_RAD	CarcraDlg	Cdialog	arcradlg.h	arcradlg.cpp
IDD_CIRCLE_ARC_ST_END	CarcseDlg	Cdialog	arcsedlg.h	arcsedlg.cpp
IDD_CIRCLE_THREE_PT	Ccir3pDlg	Cdialog	cir3pDlg.h	cir3pdlg.cpp
IDD_CORNER_BLENDING	CcnrblDlg	Cdialog	cnrbldlg.h	cnrbldlg.cpp
IDD_CRE_BLENDING	CcreblDlg	Cdialog	crebldlg.h	crebldlg.cpp
IDD_CURVE_PARAMETER	CcrvpmDlg	CDialog	crvpmdlg.h	crvpmdlg.cpp

IDD_CURVE_PROJECTION	CprojdDlg	CDialog	projddlg.h	projddlg.cpp
IDD_CYLINDER_RAD_HEIGHT	CcybteDlg	CDialog	cybtedlg.h	cybtedlg.cpp
IDD_ELLIPSE	CelcmrDlg	CDialog	elcmrdlg.h	elcmrdlg.cpp
IDD_FINISH_CUT_CARTESIAN	CfcartDlg	CDialog	fcartdlg.h	fcartdlg.cpp
IDD_FINISH_CUT_PARAMETRIC	CfparaDlg	CDialog	fparadlg.h	fparadlg.cpp
IDD_LINE_DIR	ClindiDlg	CDialog	lindidlg.h	lindidlg.cpp
IDD_LINE_OFFSET	ClinofDlg	CDialog	linofdlg.h	linofdlg.cpp
IDD_LOOKAT	CvprojDlg	CDialog	vprojdlg.h	vprojdlg.cpp
IDD_NORMAL_CURVE	CcrvnrDlg	CDialog	crvnrdlg.h	crvnrdlg.cpp
IDD_NORMAL_SURFACE	CsurfnDlg	CDialog	surfndlg.h	surfndlg.cpp
IDD_POINT_KEYBOARD	CptkeyDlg	CDialog	ptkeydlg.h	ptkeydlg.cpp
IDD_POST_PROCESSING	CpostpDlg	CDialog	postpdlg.h	postpdlg.cpp
IDD_ROTATE	CrotaxDlg	CDialog	rotaxdlg.h	rotaxdlg.cpp
IDD_ROUGH_CUT_CARTESIAN	CrcartDlg	CDialog	rcartdlg.h	rcartdlg.cpp
IDD_ROUGH_CUT_CONTOUR	CrcontDlg	CDialog	rcontdlg.h	rcontdlg.cpp
IDD_SCALE	CscaleDlg	CDialog	scaledlg.h	scaledlg.cpp
IDD_STRAIGHT	Cstr2pDlg	CDialog	str2pdlg.h	str2pdlg.cpp
IDD_STRAIGHT_PT_DIREC	CstrpdDlg	CDialog	strpddlg.h	strpddlg.cpp

IDD_SWEEP_NORMAL	CnoswpDlg	CDialog	noswpdlg.h	noswpdlg.cpp
IDD_SWEEP_PARALLEL	CpaswpDlg	CDialog	paswpdlg.h	passwpdlg.cpp
IDD_SWEEP_ROTATIONAL	CroswpDlg	CDialog	roswpdlg.h	roswpdlg.cpp
IDD_SWEEP_SYNCRONIZED	CsyswpDlg	CDialog	syswpdlg.h	syswpdlg.cpp
IDD_TRANSLATE	CtrnavDlg	CDialog	trnavdlg.h	trnavdlg.cpp
IDD_PICKINGMETHOD	CpicmtDlg	CDialog	picmtdlg.h	picmtdlg.cpp
IDD_PLANE	CplnptDlg	CDialog	plnptdlg.h	plnptdlg.cpp
IDD_PLANE_THREE_PT	Cpl3ptDlg	CDialog	pl3ptdlg.h	pl3ptdlg.cpp
IDD_OPTION_COLOR	COptColorDlg	CDialog	optcolordlg.h	optcolordlg.cpp
IDD_MYDIALOGBAR	CMyDialogBar	CDialogBar	mydialogbar.h	mydialogbar.cpp
IDD_STATUS	CDialogBar	CDialogBar	mainfrm.h	mainfrm.cpp

B. 기본 모듈에 관련된 클래스와 API

1. 수치계산용 클래스

```
/*************************************************************************/
#if !defined(POSITION_CLASS)

class position {
        double coord[3];
        friend position operator*(position const &, double );
public:

        position(){}
        position(double xi, double yi, double zi){
                coord[0] = (double) xi;
                coord[1] = (double) yi;
                coord[2] = (double) zi;
        }
        position(double p[3]){
                coord[0] = (double) p[0];
                coord[1] = (double) p[1];
                coord[2] = (double) p[2];
        }

        ~position(){}

        double x() const { return coord[0]; }
        double y() const { return coord[1]; }
        double z() const { return coord[2]; }
        double coordinate(int i) const { return coord[i]; }

        void set_x(double new_x) { coord[0] = new_x; }
        void set_y(double new_y) { coord[1] = new_y; }
        void set_z(double new_z) { coord[2] = new_z; }

        friend vector operator-(position const &, position const&);

        friend position operator+(position const &, vector const &);
        friend position operator+(vector const &, position const &);
        position const &operator+=(vector const &);
        friend position operator-(position const &, vector const &);
        position const &operator-=(vector const &);
        friend double operator%(position const &, vector const &);
        friend double operator%(vector const &, position const &);
        friend position operator*(position const &, unit_vector const &);
```

```
            friend position operator*(unit_vector const &, position const &);
            friend logical operator==(position const&, position const& );

friend position interpolate(double , position const &, position const & );

#if !defined(NO_JOURNAL)
            friend ostream& operator<<(ostream&, position const &);
            friend istream& operator>>(istream&, position &);
#endif

            void debug(short, FILE *fp = debug_file_ptr ) const;

};

#define POSITION_CLASS

#endif
/*****************************************************************************/


/*****************************************************************************/
// define vector
#if !defined(VECTOR_CLASS)

class vector {
            double comp[3];
public:
            vector(){}
            vector(double x, double y, double z){
                        comp[0] = (double) x;
                        comp[1] = (double) y;
                        comp[2] = (double) z;
            }
            vector(double v[3]){
                        comp[0] = (double) v[0];
                        comp[1] = (double) v[1];
                        comp[2] = (double) v[2];
            }

            ~vector(){}
            double x() const { return comp[0]; }
            double y() const { return comp[1]; }
            double z() const { return comp[2]; }
            double component(int i) const { return comp[i]; }

            void set_x(double new_x) {comp[0] = new_x; }
```

```
void set_y(double new_y) {comp[1] = new_y; }
void set_z(double new_z) {comp[2] = new_z; }

friend position const &operator+=(position const &, vector const &);
friend position operator+(position const &, vector const &);
friend position operator+(vector const &, position const &);
friend position operator-(position const &, vector const &);
friend double operator%(position const &, vector const &);
friend double operator%(vector const &, position const &);
friend vector operator-(vector const &);
friend vector operator+(vector const &, vector const &);
vector const &operator+=(vector const &);
friend vector operator-(vector const &, vector const &);
vector const &operator-=(vector const &);
friend double operator%(vector const &, vector const &);
friend vector operator*(vector const &, vector const &);
friend vector operator*(double, vector const &);
friend vector operator*(vector const &, double );
vector const &operator*=(double);
friend vector operator/(vector const &, double);
vector const &operator/=(double);

double len() const;

friend unit_vector normalise(vector const &);
        friend double angle(vector const&, vector const&, vector const&);
        friend int VecUni(vector a, unit_vector &Ua, int &ierr);
        friend int VecUni(vector a1, vector &a2, int &ierr);
        friend int VecPrj(vector a, vector n, vector &b);
        friend void VecDst(vector a, vector b, double &d);

#if !defined(NO_JURNAL)
        friend ostream& operator<<(ostream &, vector const &);
        friend istream& operator>>(istream &, vector &);

#endif

        friend position vectopos(vector &v1)
        {
                return position(v1.x(), v1.y(), v1.z());
        }

friend vector postovec(position &p1)
{
        return vector(p1.x(), p1.y(), p1.z());
}
```

```
        void debug(short, FILE *fp = debug_file_ptr ) const;

};

#define VECTOR_CLASS
#endif
/*****************************************************************************/


/*****************************************************************************/
// define unit-vector
#if !defined( UNITVEC_CLASS)
#define EPS 1.0e-10

class unit_vector: public vector{
public:
        unit_vector(): vector() {}
        unit_vector(double x, double y, double z) {
                set_x(x);
                set_y(y);
                set_z(z);
        }
        unit_vector(double u[3]) {
                set_x(u[0]);
                set_y(u[1]);
                set_z(u[2]);
}

~unit_vector(){}

friend void errmsg(int, char, char *);

friend position operator*(position const &, unit_vector const &);
friend unit_vector operator-(unit_vector const &);
friend double operator%(position const &, unit_vector const &);
friend double operator%(unit_vector const &, position const &);
friend position operator*(unit_vector const &, position const &);

friend logical operator==(unit_vector const&, unit_vector const&);

friend int VecUni(vector a, unit_vector &Ua, int &ierr);

#if !defined(NO_JOURNAL)
        friend istream& operator>>(istream &, unit_vector & );
#endif

        void debug(short, FILE *fp = debug_file_ptr ) const;
```

```
};

#define UNITVEC_CLASS
#endif
/*****************************************************************************/


/*****************************************************************************/
// define parameter
#if !defined( PARAMETER_CLASS )

class parameter {
        double val;
public:
        parameter(){}
        parameter(double p){ val = p; }
        ~parameter(){}

        // get member data
        double value() const { return val; }

        // set member data
        void set_value( double pval ) { val = pval; }

        operator double() const { return val; }
        parameter operator-() const { return -val; }

        friend double operator+(parameter const &p1, parameter const &p2)
        { return p1.val + p2.val; }

        friend double operator+(parameter const &p, double d)
        { return p.val + d; }

        friend double operator+(double d, parameter const &p)
        { return d + p.val; }

        friend double operator-(parameter const &p1, parameter const &p2)
        { return p1.val - p2.val; }

        friend double operator-(parameter const &p, double d)
        { return p.val - d; }

        friend double operator-(double d, parameter const &p)
        { return d - p.val; }
        friend double operator*(parameter const &p1, parameter const &p2)
        { return p1.val * p2.val; }
```

```
friend double operator*(parameter const &p, double d)
{ return p.val * d; }

friend double operator*(double d, parameter const &p)
{ return d * p.val; }

friend double operator/(parameter const &p1, parameter const &p2)
{ return p1.val / p2.val; }

friend double operator/(parameter const &p, double d)
        { return p.val / d; }

        friend double operator/(double d, parameter const &p)
        { return d / p.val; }

        parameter operator+=(double rhs){ return val += rhs; }
        parameter operator-=(double rhs){ return val -= rhs; }
        parameter operator*=(double rhs){ return val *= rhs; }
        parameter operator/=(double rhs){ return val /= rhs; }

//#if !defined(NO_JOURNAL)
        friend ostream& operator<<(ostream&, parameter const &);
        friend istream& operator>>(istream&, parameter&);
//#endif

        void debug(short, FILE *fp = debug_file_ptr ) const;
//      { debug_dist(val, fp); }

};
/*****************************************************************************/


/*****************************************************************************/
//Define parameter position, i.e. in parameter space of a surface

class par_pos {
public:
        parameter u;
        parameter v;
#if defined(osf1)                                       // possible error
        par_pos(): u(0.0), v(0.0) {}
#else
        par_pos() {}
#endif
        //construct a par_pos from two doubles.
        par_pos(double uval, double vval): u(uval), v(vval) {}
```

```
//construct a par_pos from an array of two doubles.
par_pos(double uv[2]): u(uv[0]), v(uv[1]) {}

~par_pos(){}

// get member data
parameter u_val() { return u; }
parameter v_val() { return v; }

double uval() const { return u.value(); }
double vval() const { return v.value(); }

// set member data
void set_u( parameter u_val ) { u = u_val; }
void set_v( parameter v_val ) { v = v_val; }
void set_u( double u_val ) { u.set_value(u_val); }
void set_v( double v_val ) { v.set_value(v_val); }

//simple arithmetic
friend par_pos operator+(par_pos const &, par_vec const &);
friend par_pos operator+(par_vec const &v, par_pos const &p);

friend par_pos operator-(par_pos const &, par_vec const &);
friend par_vec operator-(par_pos const &, par_pos const &);

par_pos const &operator+=(par_vec const &);
par_pos const &operator-=(par_vec const &);

//Scalar product of position with vector
friend double operator%(par_pos const &, par_vec const &);
friend double operator%(par_vec const &v, par_pos const &p);

// Assignment operator
par_pos &operator=(par_pos const &);

// Equality operator
friend logical operator==(par_pos const &, par_pos const &);

//Ouput details of a par_pos.
void debug(short, FILE *fp = debug_file_ptr) const;
};
/*****************************************************************************/

/*****************************************************************************/
//Define a vector in parameter space
```

```
class par_vec {
public:
        parameter du;
        parameter dv;

        par_vec() {}            //Allow unitialised par_vec.

        //Construct a par_vec form two doubles.
        par_vec(double u, double v): du(u), dv(v) {}

//Construct a par_vec from an array of two doubles.
par_vec(double uv[2]): du(uv[0]), dv(uv[1]) {}

//Construct a par_vec from a par_dir. This will be defined inline,
//but later once we have defined par_dir.
par_vec(par_dir const &);

~par_vec(){}

// get member data
parameter du_val() { return du; }
parameter dv_val() { return dv; }

// set member data
void set_du( parameter du_val ) { du = du_val; }
void set_dv( parameter dv_val ) { dv = dv_val; }
void set_du( double du_val ) { du.set_value(du_val); }
void set_dv( double dv_val ) { dv.set_value(dv_val); }

//***Simple arithmetic
par_vec friend operator-(par_vec);
par_vec friend operator+(par_vec const &v1, par_vec const &v2);
par_vec friend operator-(par_vec const &v1, par_vec const &v2);
par_vec friend operator*(par_vec const &v, double d) {
        return v*d;
}
par_vec friend operator*(double d, par_vec const &v) {
        return d*v;
}
par_vec friend operator/(par_vec const &v, double d);
double friend operator%(par_vec, par_vec);
double friend operator*(par_vec, par_vec);

friend double angle(par_vec const&, par_vec const&);
par_vec const &operator+=(par_vec const &);
par_vec const &operator-=(par_vec const &);
par_vec const &operator*=(double);
```

```
        par_vec const &operator/=(double);

        double len() const;

        //Output details of a par_vec.
        void debug(short, FILE *fp = debug_file_ptr) const;
};
/*****************************************************************************/

//Define a direction in parameter direction.

class par_dir : public par_vec {
public:
        par_dir() {}

        // Construct and normalise a par_dir from two doubles.
        par_dir(double d1, double d2) {
                double len = sqrt(d1*d1 + d2*d2);
                du = d1 / len;
                dv = d2 / len;
        }
        // Construct and normalise a par_dir form an array of two doubles.
        par_dir(double uv[2]) {
                double len = sqrt(uv[0]*uv[0] + uv[1]*uv[1]);
                du = uv[0] / len;
                dv = uv[1] / len;
}
par_dir(par_vec const &d) {
                double len = sqrt(d.du*d.du + d.dv*d.dv);
                du = d.du;
                dv = d.dv;
}

~par_dir(){}

par_dir friend operator-(par_dir const &);

friend double operator%(par_pos const &p, par_dir const &u) {
                return p% *(par_vec const *)&u;
        }
        friend double operator%(par_dir const &u, par_pos const &p) {
                return p% *(par_vec const *)&u;
        }
};
/*****************************************************************************/
```

```
/*****************************************************************/
// Header for transf.

// This class records a general transformation of 3D vectors.

#if !defined( TRANSF_CLASS )

#include "kernel.h"
#include "misc\misc.h"

class vector;
class unit_vector;
class position;

class transf
{

        double elem[4][4];

public:

        // Creates the identity transformation.
        transf();

        // Creates transformation by 4x4 double array.

        transf( double [4][4] );

        // makes a copy constructor.

        transf( transf const & );

        // Construction routines

        friend transf scale_transf( double );

        friend transf scale_transf( double, double, double );

        // Data reading routines

        double value(int i, int j) const { return elem[i][j]; }

        // Data changing routines

        void set_value(int i, int j, double value)
```

```
{
        elem[i][j] = value;
}

// Set transformation to rotation by angle about vector.

friend transf rotate_transf( double, vector const & );

// Set transformation to reflection about plane through
// origin and perpendicular to given vector.

friend transf reflect_transf( vector const & );

// Set transformation to translation along vector.

friend transf translate_transf( vector const & );

// Set transformation to carry origin to given position, and
// x and y axes to the given unit vectors.  If the second unit
// vector is not orthogonal to the first, uses instead a
// unit vector in the plane of the two given vectors, that is.

transf coordinate_transf( position const &,
                          unit_vector const &, unit_vector const & );

// Multiply two transformations.

friend transf operator*( transf const &, transf const & );
friend transf operator*( transf const &t1, transf const *t2 )
{
        return t1 * (*t2);
}

transf const &operator*=( transf const & );

// Return the inverse transformation (must be no shear in the
// given transformation).

transf inverse() const;

// Transform a vector.

friend vector operator*( vector const &, transf const & );
friend vector const &operator*=( vector &,transf const & );
// Transform a unit vector.

friend unit_vector operator*( unit_vector const &,
```

```
            transf const & );
friend unit_vector const &operator*=( unit_vector &,
            transf const & );

// Transform a position.

friend position operator*( position const &, transf const & );
friend position const &operator*=( position &, transf const & );

// Output details of a transf.
            void debug( short, FILE * = debug_file_ptr ) const;
};

#define TRANSF_CLASS
#endif
/*****************************************************************************/


/*****************************************************************************/
// Header for interval.
// This class records an interval on the real line.    It is implemented
// as an ordered pair of doubles.

#if !defined( INTERVAL_CLASS )

#include <math.h>

#include "kernel.h"
#include "misc\logical.h"
#include "misc\misc.h"

class interval {

            double low;
            double high;

public:

            interval()  // Construct an empty interval.
            {
                low = 1;
                high = 0;
}
// Construct a zero-length interval from one real value.

interval( double d )
```

```
{
        low = d;
        high = d;
}

// Construct an interval from two real values.

interval( double l, double h )
{
        low = l;
        high = h;
}

// Negate an interval

interval operator-();

// Add two intervals together. Provide double operators as well
// to improve efficiency.

friend interval operator+( interval const &, interval const & );
friend interval operator+( interval const &, double );
friend interval operator+( double, interval const & );
interval &operator+=( interval const & );
interval &operator+=( double d )
{
        low += d;
        high += d;
        return *this;
}

// Subtract two intervals or an interval and a double.

friend interval operator-( interval const &, interval const & );
friend interval operator-( interval const &, double    );
friend interval operator-( double, interval const & );
interval &operator-=( interval const & );
interval &operator-=( double d )
{
        low -= d;
        high -= d;
        return *this;
}

// Multiply an interval by a scalar.

friend interval operator*( interval const &intval, double d );
```

```
friend interval operator*( double d, interval const &intval );
interval &operator*=( double d )
{
        low *= d;
        high *= d;
        return *this;
}

// Divide an interval by a scalar

friend interval operator/( interval const &intval, double d )
{
        return interval( intval.low / d, intval.high / d );
}

interval &operator/=( double d )
{
        low /= d;
        high /= d;
        return *this;
}

// Arithmetic comparisons.

friend logical operator<( interval const &i, double d )
{
        return i.high < d;
}
friend logical operator<( double d, interval const &i )
{
        return d < i.low;
}
friend logical operator<( interval const &i1, interval const &i2 )
{
        return i1.high < i2.low;
}

friend logical operator>( interval const &i, double d )
{
        return i.low > d;
}
friend logical operator>( double d, interval const &i )
{
        return d > i.high;
}
friend logical operator>( interval const &i1, interval const &i2 )
{
```

```
        return i1.low > i2.high;
}

friend logical operator<=( interval const &i, double d )
{
        return i.high <= d;
}
friend logical operator<=( double d, interval const &i )
{
        return d <= i.low;
}
friend logical operator<=( interval const &i1, interval const &i2 )
{
        return i1.high <= i2.low;
}

friend logical operator>=( interval const &i, double d )
{
        return i.low >= d;
}
friend logical operator>=( double d, interval const &i )
{
        return d >= i.high;
}
friend logical operator>=( interval const &i1, interval const &i2 )
{
        return i1.low >= i2.high;
}

// Construct an interval containing two intervals.

friend interval operator|( interval const &, interval const & );
interval &operator|=( interval const & );

// Find the interval of overlap.

friend interval operator&( interval const &, interval const & );
interval &operator&=( interval const & );

// Return useful information.

logical empty() const { return low > high; }
logical scalar() const { return fabs(low - high) <= resabs; }

// Data changing
void set_low( double l ) { low = l; }
void set_high( double h ) { high = h; }
```

```
double start() const { return low; }
double end() const { return high; }
double mid() const { return (low+high) / 2.0; }

// Return the difference between the high and low ends

double length() const {          return abs(high-low); }

// Print out details of an interval.

        void debug( short, FILE * = debug_file_ptr ) const;
};

#define INTERVAL_CLASS
#endif
/*****************************************************************************/


/*****************************************************************************/
// Header for box.

// This class represents a bounding box.    It is implemented as an
// axis-dependent, axis-aligned rectangular box, given by a triple
// of real intervals.

#if !defined( BOX_CLASS )

#include "kernel.h"
#include "box\interval.h"
#include "math\vector.h"
#include "misc\logical.h"
#include "misc\misc.h"

class box {

        interval xb;        // extent of box in x direction
        interval yb;        // extent of box in y direction
        interval zb;        // extent of box in z direction

public:

// An uninitialised box is empty (because all its intervals are).

box();
// Construct an infinitesimal box from one position.

box( position const & );
```

```
// Construct a box from two positions.

box( position const &, position const & );

// Construct a box as a copy of another box

box( box const & );

// Extract the constituent data from the box, as the ends of the
// leading diagonal.

position low() const;
position high() const;

// Return axis bound
interval &xbound() { return xb; }
interval &ybound() { return yb; }
interval &zbound() { return zb; }

// Combine two boxes, i.e. a box that encloses the two given boxes

friend box operator|( box const &, box const & );

// Compound one box into another, i.e. extend this box until it
// also encloses the given box.

box & operator|=( box const & );

// Find the overlap of two boxes, i.e. the intersection

friend box operator&( box const &, box const & );

// Limit one box by another, i.e. form the intersection of this
// box with the given box, and make the result this box.

box & operator&=( box const & );

// Determine whether two boxes overlap.
// Returns true if either box is null, or if all the intervals
// of one box overlap the corresponding intervals of the other.

friend logical operator&&( box const &, box const & );
// Determine point containment, i.e. returns true if point is
// contained within this box (or if this box is null).

logical operator<<( position const & ) const;
```

```
// Determine if this box entirely enclosed into given box.
// Returns true if this box is null, false if given box is null,

logical operator>>( box const & ) const;

// Determine if box encloses given position.

friend logical operator>>( position const &p, box const &b )
{ return b << p; }

            // Determine if given box encloses this box.

            logical operator<<( box const &b )
            { return b >> *this; }

            // Determine whether two boxes are coincident

            logical operator==( box const &b ) const;

            // Output details of a box.

            void debug( short, FILE * = debug_file_ptr ) const;
};

#define BOX_CLASS
#endif
/*****************************************************************************/
```

2. 형상요소의 클래스

```
/*****************************************************************************/
// Class for NURB( Non-Uniform Rational B-spline ) curve.
// It has a number of data members for storing NURB curve, and all curves in SURFART
// have this class as member, ie. all curves are stored as class nurbcrv

#if !defined( NURBCRV_CLASS )

#include "kernel.h"
#include "math\vector.h"
#include "math\transf.h"

// Class nurbcrv declaration

class nurbcrv {
```

```
        // The order of NURB crve
        long order;

        // A number of knot vectors
        long no_knot;
        // The knot vectors
        double *knot;
        // A number of control points
        long no_ctlpt;
        // The arrsy of control points
        position *ctlpt_net;
        // The array of weights
        double *weight;
        // The style of NURB curve
        CRVSTYLE style;
        // Polynoimal curve id
        int poly_crvid;
public:
        // Default constructor
        nurbcrv();
        // Copy constructor
        nurbcrv(nurbcrv const &);
        // Construct nurbcrv from details of NURB
        nurbcrv(long, long, double *, long, position *, double *);
```

```
// Construct nurbcrv from details of NURB(polycrv id)
nurbcrv(long, long, double *, long, position *, double *, int);
// Default destructor
~nurbcrv();
// Data reading routines
long get_order() const { return order; }
long get_no_knot() const { return no_knot; }
double *get_knot() const { return knot; }
long get_no_ctlpt() const { return no_ctlpt; }
position *get_ctlpt_net() const { return ctlpt_net; }
double *get_weight() const { return weight; }
position get_ctlpt_net( long );
double get_weight( long );
CRVSTYLE get_style() const { return style; }
int get_poly_crvid() const {return poly_crvid; }
// Data changing routines
void set_order( long );
void set_no_knot( long );
void set_knot( double * );
void set_no_ctlpt( long );
void set_ctlpt_net( position * );
void set_weight(double * );
void set_poly_crvid(int );
// Assignment operator
```

```
nurbcrv &operator=( nurbcrv const & );

// Reverse the direction of curve

void reverse();

// Evaluate a curve at a given parameter value

void eval(parameter, position &, vector &, vector &,
          logical = FALSE, logical = FALSE);

// Get only position from NURB evaluator

void eval(parameter, position &);

// Transform NURBS curve
friend nurbcrv operator*(nurbcrv const &, transf const &);

// Check whether two curves are same

logical same( nurbcrv* );

// Check whether curve is degenerated

logical degenerated();

          // Save and restore data of nurbcrv

          void save_data( FILE * ) const;

          void restore_data( FILE *);

          // Print out details of nurbcrv

          void debug(short, FILE *fp = debug_file_ptr) const;

};

#define NURBCRV_CLASS
#endif
/*************************************************************************/

/*************************************************************************/
// Class for NURB(Non-Uniform Rational B-spline) surface.
// It has a number of data members and function for storing and managing
```

```
// a NURB surface, All surfaces in SURFART have this class as member, ie
// all surfaces are stored as class nurbsuf

#if !defined( NURBSUF_CLASS )

#include "kernel.h"
#include "math\vector.h"
#include "math\transf.h"

// Class nurbsuf declaration

class nurbsuf {

        // The u,v-direction order of NURB surface

        long u_order;

        long v_order;

        // The number of knot vectors and vectors array of u,v-direction

        long nu_knot;

        double *u_knot;

        long nv_knot;

        double *v_knot;

        // The number of control points of u,v-direction

        long nu_ctlpt;

        long nv_ctlpt;

        // The array for control points and weights

        position *ctlpt_net;

        double *weight;

        // The style of u, v direction curve of NURB surface

        CRVSTYLE u_style;
        CRVSTYLE v_style;

        // Polynoimal Surface id
```

```
        int poly_sfid;
public:
        // Default constructor
        nurbsuf();
        // Copy constructor
        nurbsuf(nurbsuf const &);
        // Construct nurbsuf from details
        nurbsuf(long, long, long, double *, long, double *, long, long,
                position *, double * );
        // Construct nurbsuf from details(polysfid)
        nurbsuf(long, long, long, double *, long, double *, long, long,
                position *, double *, int );
        // Default destructor
        ~nurbsuf();
        // Data reading routines
        long get_u_order() const { return u_order; }
        long get_v_order() const { return v_order; }
        long get_nu_knot() const { return nu_knot; }
        double *get_u_knot() const { return u_knot; }
        long get_nv_knot() const { return nv_knot; }
        double *get_v_knot() const { return v_knot; }
        long get_nu_ctlpt() const { return nu_ctlpt; }
        long get_nv_ctlpt() const { return nv_ctlpt; }
        position *get_ctlpt_net() const { return ctlpt_net; }
        double *get_weight() const { return weight; }
```

```
position get_ctlpt_net(long, long);

double get_weight(long, long);

CRVSTYLE get_u_style() const { return u_style; }

CRVSTYLE get_v_style() const { return v_style; }

int get_poly_sfid() const { return poly_sfid; }

// Data changing routines

void set_u_order(long);

void set_v_order(long);

void set_nu_knot(long);

void set_u_knot(double *);

void set_nv_knot(long);

void set_v_knot(double *);

void set_nu_ctlpt(long);

void set_nv_ctlpt(long);

void set_ctlpt_net(position *);

void set_weight(double *);

void set_poly_sfid(int );

// Assignment operator

nurbsuf &operator=( nurbsuf const & );

// Reverse the normal direction of NURB surface

void reverse( );

// Evaluate a surface at a given parameter value
void eval(par_pos const &, position        &, vector & = *(vector *)NULL,
          vector & = *(vector *)NULL, unit_vector & = *(unit_vector *)NULL,
          logical = FALSE, logical = FALSE, logical = FALSE);
```

```
// Transform NURBS surface
friend nurbsuf operator*(nurbsuf const &, transf const &);

// Write the data of nurbsuf to file

void save_data( FILE * ) const;

// Read the data of NURB surface from file

void restore_data( FILE * );
        // Print out details of nurbsuf

        void debug(short, FILE *fp = debug_file_ptr) const;

};

#define NURBSUF_CLASS
#endif
/*****************************************************************************/


/*****************************************************************************/
// Header for (lower-case) curve.
// This is an abstract base class for classes representing specific
// curve shapes. It provides a large variety of virtual
// functions for generic interaction with curves.

#if !defined( GEOCURVE_CLASS )

#include "kernel.h"
#include "misc\misc.h"
class transf;

// curve declaration

class curve {

        int crvtype;

public:

        curve();                    // Force creation of all curves to be by
constructor
```

```
virtual ~curve();       // Ensure any derived class destructor gets
                        // a say when we destroy a curve.

// Get drivation type. ie. curve type

int type() const { return crvtype; }

// Set drivation type. ie curve type

void set_type(int);

// Make a copy of the given curve.   This function is not virtual
// so as to handle a NULL "this".

curve *copy_curve() const {
            return this == NULL ? NULL : make_copy();
}

// Make a copy of the given curve. This is avirtual
// function to ensure that each derived class defines its own.

virtual curve *make_copy() const;

// Return a string identifying the curve type

virtual char *type_name() const;

// Reverse the sense of the curve.

virtual curve &negate();

// Assignment operator

curve &operator=( curve const &);

// Transform a curve.

virtual curve &operator*=( transf const & );

// Print out a description of the curve.

            virtual void debug( short, FILE * = debug_file_ptr ) const;
};

#define GEOCURVE_CLASS
#endif
/*************************************************************************/
```

```
/*********************************************************************/
// Header for straight, which represents an infinite straight line.

// A straight line is represented by a point on it and a tangent
// direction. It also has a scale factor for the parametrisation, so
// that the parameter values can be made invariant under
// transformation.

// The parametric equation of the line is
//                          point = root_point + t * param_scale * direction
// where t is the parameter.

#if !defined( straight_CLASS )

#include "kernel.h"
#include "curve\crvdef.h"
#include "curve\nurbcrv\nurbcrv.h"
#include "math\vector.h"

class transf;
class nurbcrv;

// Straight definition proper.

class straight: public curve {

// A point through which the straight line passes.

position root_point;

// The tangent along the line

unit_vector direction;

// Scaling factor for parametrisation, to allow fixed parameters
// despite transformation.

double param_scale;

// The NURB form of straight

        nurbcrv    *nurbcrv_ptr;
public:

        // Default constructor
```

```
straight();

// Construct a straight line from another.

straight(straight const &);

// Construct a straight line from a point, direction and parameter
// scaling.

straight(position const &, unit_vector const &, double = 1.0);

// Construct a straight from NURB form

straight(nurbcrv    *);

// Virtual destructor

~straight();

// Data reading routines

position const &rootpoint() const { return root_point; }

unit_vector const &direct() const { return direction; }

double paramscale() const { return param_scale; }

nurbcrv *equation() const { return nurbcrv_ptr;}

// Data changing routines

void set_rootpoint(position const &);

void set_direct(unit_vector const &);

void set_paramscale(double);

void set_nurb(nurbcrv *);

// Make a copy of this straight on the heap, and return a
// pointer to it.

virtual curve *make_copy() const;
// Return a string identifying the curve type

virtual char *type_name() const;
```

```
// Negate this straight line (i.e. negate the direction).

virtual curve &negate();

// Assignment operator

straight &operator=( straight const & );

// Transform this straight line by the given transf.

virtual curve &operator*=( transf const & );
void save_data( FILE * ) const;
void restore_data( FILE * );

// Output details of the curve.

virtual void debug( short, FILE * = debug_file_ptr ) const;
};

#define straight_CLASS
#endif
/*********************************************************************/


/*********************************************************************/
// Header for ellipse.

// An ellipse is recorded by its centre, normal, major radius vector
// (the displacement from the centre to one end of the major diameter)
// and radius ratio (the ratio between the minor and major axis
// lengths, exactly 1 for a circle).  The sense of the ellipse is
// clockwise around the direction of its normal.

#if !defined( ELLIPSE_CLASS )

#include "kernel.h"
#include "curve\crvdef.h"
#include "curve\nurbcrv\nurbcrv.h"
#include "math\vector.h"
#include "math\transf.h"
#include "misc\logical.h"

class box;
class ellipse: public curve {

        // Private ellipse members
```

```
    position center_pt;                     // Center point
    unit_vector normal_vec;         // Normal vector
    vector major_axis_vec;                  // Major axis
    vector minor_axis_vec;                  // Minor axis
    double radius_ratio_val;        // Radius ratio
    nurbcrv *nurbcrv_ptr;                   // Record a NURB form of ellipse
public:
    // Essentially uninitialised ellipse.
    ellipse();
    // Construct an ellipse by copying an existing one
    ellipse( ellipse const & );
    // Construct an ellipse by nurb form of ellipse
    ellipse( nurbcrv    * );
    // Construct an ellipse from its centre, normal, radius vector
    // and radius ratio.
    ellipse( position const &, unit_vector const &,
                        vector const &, vector const &, double );
    // Virtual destructor
    ~ellipse();
    // Data reading routines.
    position const &center() const    { return center_pt; }
    unit_vector const &normal() const { return normal_vec; }
    vector const &major_axis() const { return major_axis_vec; }
    vector const &minor_axis() const { return minor_axis_vec; }
    double radius_ratio() const { return radius_ratio_val; }
```

```
nurbcrv *equation() const { return nurbcrv_ptr; }

// Data changing routines.

void set_center( position const & );

void set_normal( unit_vector const & );

void set_major_axis( vector const & );

void set_minor_axis( vector const & );

void set_radius_ratio( double );

void set_nurb( nurbcrv * );

// Make a copy of this ellipse

virtual curve *make_copy() const;

// Return a string identifying the curve type

virtual char *type_name() const;

// Return an ellipse with the opposite sense from "this" one.

virtual curve &negate();

// Assignment operator

ellipse &operator=( ellipse const & );

// Transform this ellipse by the given transf, in place.

virtual curve &operator*=( transf const & );

// Check whether this is circle

logical circular() const;
void save_data(FILE *) const;
void restore_data( FILE *);

        // Output details of ellipse.
        virtual void debug(   short, FILE * = debug_file_ptr ) const;

};
```

```
#define ELLIPSE_CLASS
#endif
/*****************************************************************************/


/*****************************************************************************/
// Header for intcurve, which is the general representation of any
// curve without an explicit "equation" but defined by reference to
// other geometric entities, for example the intersection of two
// surfaces.

#if !defined( INTCURVE_CLASS )

#include "kernel.h"
#include "geom\surface.h"
#include "curve\crvdef.h"
#include "curve\nurbcrv\nurbcrv.h"
#include "curve\pcurve\pcrvdef.h"
#include "math\vector.h"

// Defining a class intcurve for general interpolation curve
// and interpolated intersection curve between two surfaces,
// ie. a fitted curve on one surface, or an exact spline curve.

class intcurve: public curve {

// The type of intcurve

short intcrv_type;     // 0: unknown
                       // 1: interpolation curve without defining surface
                       // 2: interpolation curve with defining surface
                       // 3: intersection curve of two surfaces

// Object-space approximation to the true curve

nurbcrv   *nurbcrv_ptr;

// Precision to which the spline approximates to the true
// object-space curve.

double fitol_data;

// Up to two surfaces defining the true curve.
SURFACE *surf1_data;

SURFACE *surf2_data;
```

```
        // Parametric-space curves with respect to the given surfaces.
        // Only non-NULL if the corresponding surface exists and is parametric.

        pcurve *pcrv1_data;

        pcurve *pcrv2_data;

public:

        // Default constructor

        intcurve();

        // Copy constructor.

        intcurve( intcurve const & );

        // Construct a intcurve from interpolation curve or intersection curve

        intcurve( nurbcrv *. double );

        intcurve( nurbcrv *, double, SURFACE *, pcurve * );

        intcurve( nurbcrv *, double, SURFACE *, SURFACE *, pcurve *, pcurve * );

        // Default destructor

        ~intcurve();

        // Data reading routines

        short type() const { return intcrv_type; }

        nurbcrv *equation() const { return nurbcrv_ptr; }

        double fittol() const { return fitol_data; }

        SURFACE *surf1() const { return surf1_data; }

        SURFACE *surf2() const { return surf2_data; }

        pcurve *pcrv1() const { return pcrv1_data; }

        pcurve *pcrv2() const { return pcrv2_data; }

        // Data changing routines.
```

```
void set_type( short );

void set_nurb( nurbcrv * );

void set_fitol( double );

void set_surf1( SURFACE * );

void set_surf2( SURFACE * );

void set_pcrv1( pcurve * );

void set_pcrv2( pcurve * );

// Make a copy of this intcurve

virtual curve *make_copy() const;

// Return a string identifying the curve type

virtual char *type_name() const;

// Return an intcurve with the opposite sense from "this" one.

virtual curve &negate();

// Assignment operator

intcurve &operator=( intcurve const & );

// Transform this intcurve by the given transf, in place.

virtual curve &operator*=( transf const & );

void save_data( FILE * ) const;
void restore_data( FILE * );

// Output details of the intcurve

virtual void debug( short level, FILE * = debug_file_ptr ) const;

};

#define INTCURVE_CLASS
#endif
/*********************************************************************/
```

```
/*****************************************************************/
// Class for parametric curve on surface.

#if !defined( PCRV_CLASS )

#include "kernel.h"
#include "surface\nurbsuf\nurbsuf.h"
#include "curve\nurbcrv\nurbcrv.h"
#include "math\vector.h"
#include "box\box.h"

// Class pcurve declaration

class pcurve {

        // The order of 2D NURB crve

        long order;

        // The number of knot vectors and array

        long no_knot;

        double *knot;

        // The number of control points and array

        long no_ctlpt;

        par_pos *ctlpt_net;

        // The array of weights

        double *weight;

        // The style of NURB curve

        CRVSTYLE style;

        // Polynoimal pcurve id

        int poly_pcrvid;
public:

        // Default constructor

        pcurve();
```

```
// Copy constructor

pcurve(pcurve const &);

// Construct a pcurve from nurbcrv pointer

//pcurve::pcurve( nurbcrv * );

// Construct a pcurve from details of 2D NURB curve and defining surface

pcurve( long, long, double *, long, par_pos *, double * );

pcurve( long, long, double *, long, par_pos *, double *, int );

// Default destructor

~pcurve();

// Data reading routines

long get_order() const { return order; }

long get_no_knot() const { return no_knot; }

double *get_knot() const { return knot; }

long get_no_ctlpt() const { return no_ctlpt; }

par_pos *get_ctlpt_net() const { return ctlpt_net; }

double *get_weight() const { return weight; }

par_pos get_ctlpt_net( long );

double get_weight( iong );

CRVSTYLE get_style() const { return style; }

int get_poly_pcrvid() const {return poly_pcrvid; }

box &bound();

// Evaluation of parametric curve;

void eval(nurbsuf *, parameter, par_pos&, par_vec&, par_vec&,
                logical = FALSE, logical = FALSE );
```

```
void eval( parameter, par_pos & );

// Data changing routines

void set_order( long );

void set_no_knot( long );

void set_knot( double * );

void set_no_ctlpt( long );

void set_ctlpt_net( par_pos * );

void set_weight(double * );

void set_poly_pcrvid(int );

// Assignment operator

pcurve &operator=( pcurve const & );

// Write the data of pcurve to file

void save_data( FILE * ) const ;

// Read data of pcurve from file

void restore_data( FILE * );

// Print out details of nurbcrv

void debug(short, FILE *fp = debug_file_ptr) const;

};

#define PCRV_CLASS
#endif
/*************************************************************************/

/*************************************************************************/
// Header for (lower-case) surface.

// The generic surface is defined as a portmanteau for the specific
// surfaces. It contains type of surface and virtual functions for
// the derived types to elaborate. Some functions are pure, forcing
```

```
// derived classes to define their own versions, and preventing the
// construction of an object of the base class, and some have default
// definitions which may be useful in derived classes.

#if !defined( GEOSURFACE_CLASS )

#include "kernel.h"
#include "math\vector.h"
#include "misc\logical.h"

class curve;
class transf;
class interval;
class box;

// surface declaration

class surface {

        int surftype;

public:

        surface();      // Force creation of all surfaces to be by constructor
        virtual ~surface();     // Ensure any derived class destructor gets
                                                        // a say when we destroy a
surface.

        // Get drivation type. ie. surface type

        int type() const { return surftype; }

        // Set drivation type. ie surface type

        void set_type(int);

        // Make a copy of the given surface.    This function is not virtual
        // so that we can handle a NULL "this".    It calls make_copy().

        surface *copy_surf() const {
                return this == NULL ? NULL : make_copy();
}

// Make a copy of the given surface. This is a virtual
// function to ensure that each derived class defines its own.
```

```
virtual surface *make_copy() const;

// Return a string identifying the surface type

virtual char *type_name() const;

// Reverse the sense of the surface.

virtual surface &negate();

// Assignment operator

surface &operator=( surface const & );

// Transform a surface.

virtual surface &operator*=( transf const & );

// Print out details of the surface.

virtual void debug(   short, FILE * = debug_file_ptr ) const;

};

#define GEOSURFACE_CLASS
#endif
/*********************************************************************/


/*********************************************************************/
// Header for (lower-case) plane.

// The plane is instanced in the PLANE entity.

#if !defined( PLANE_CLASS )

#include "kernel.h"
#include "surface₩surfdef.h"
#include "math₩vector.h"

class nurbsuf;

// Define plane proper.

class plane: public surface {
```

```
position root_point_val;  // A point through which the plane
                          // passes

unit_vector normal_vec;   // The normal to the plane

vector u_dir;             // A vector in the plane which gives
                          // the direction and scaling of u
                          // parameter lines.

logical reverse_v;        // By default the v direction is the
                          // cross product of normal with u_dir.
                          // If this is TRUE, the v direction must
                          // be negated.

// Record a NURB form of a plane.

nurbsuf *nurbsuf_ptr;

public:

// Permit uninitialised planes, though default the parametric
// information.

plane();

// Construct a plane as a copy of another one

plane( plane const & );

// Construct a plane with NURB form ofr plane

plane(nurbsuf *);

// Construct a plane with given position and normal.

plane( position const &, unit_vector const &, nurbsuf *    );

// Virtual destructor

~plane();

// Data reading routines.
```

```
position const &root_point() const { return root_point_val; }

unit_vector const &normal() const { return normal_vec; }

vector const &u_direction() const { return u_dir; }

logical v_reverse() const { return reverse_v; }

nurbsuf *equation() const { return nurbsuf_ptr; }

// Data changing routines.

void set_root_point( position const & );

void set_normal( unit_vector const & );

void set_u_dir( vector const & );

void set_reverse_v( logical );

void set_nurb(nurbsuf *);

// Make a copy of the given surface.

virtual surface *make_copy() const;

// Return a string identifying the surface type

virtual char *type_name() const;

// Reverse the sense of the plane.

virtual surface &negate();

// Assignment operator

plane &operator=( plane const & );
// Transform a plane.

virtual surface &operator*=( transf const & );

void save_data( FILE * ) const;
```

```
        void restore_data( FILE * );

        // Print out details of plane.

        virtual void debug(       short,  FILE * = debug_file_ptr   )
const;
};

#define PLANE_CLASS

#endif
/*****************************************************************************/


/*****************************************************************************/
// Class definition for a cone, representing an elliptical cone at
// any position or orientation in space. As special cases, the
// cross-section may be circular, or the "cone" may be a cylinder,
// or both.

// A cone is recorded by a bottom cross-sectional ellipse and
// top cross-sectional ellipse.

// If the bottom ellipse and top ellipse are same, it represents
// elliptical plane,

#if !defined( CONE_CLASS )

#include "kernel.h"
#include "surface₩surfdef.h"
#include "curve₩ellipse₩elldef.h"
#include "misc₩logical.h"

class nurbsuf;

// Define cone proper.

class cone: public surface {

        ellipse base;                   //    Bottom    cross-sectional
ellipse

        ellipse top;                    // Top cross-sectional ellipse
```

```
        logical rev;                        // Status of surface normal
        nurbsuf *nurbsuf_ptr;     // NURB form of conical surface
public:
        // Default constructor
        cone();
        // Construct a cone as a copy of another one
        cone( cone const & );
        // Construct a cone with given ellipse as base, top and NURB
        // form
        cone( ellipse const &, ellipse const &, nurbsuf * );
        // Construct a cone with given ellipse as base and top
        cone( ellipse const &, ellipse const & );
        // Construct a cons with NURB form
        cone( nurbsuf * );
        // Virtual destructor
        ~cone();
        // Data reading routines.
        ellipse const &get_base() const { return base; }
        ellipse const &get_top() const { return top; }
        logical reversed() const { return rev; }
        nurbsuf *equation() const { return nurbsuf_ptr; }
        // Data changing routines
```

```
    void set_base( ellipse const & );

    void set_top( ellipse const & );

    void set_rev( logical );

    void set_nurb( nurbsuf * );

    // Make a copy of this cone on the heap, and return a pointer
    // to it.

    virtual surface *make_copy() const;

    // Return a string identifying the cone type

    virtual char *type_name() const;

    // Negate this cone.

    virtual surface &negate();

    // Assignment operator

    cone &operator=( cone const & );

    // Transform this cone by the given transf.

    virtual surface &operator*=( transf const & );

    // Simple classification routines, defined here as a convenience,
    // and also to help ensure consistency.

    logical cylinder() const;

    logical flat() const;

    void save_data(FILE *) const;
    void restore_data( FILE * );

        // Print out details of cone.

        virtual void debug( short, FILE * = debug_file_ptr ) const;
};
```

```
#define CONE_CLASS
#endif
/*****************************************************************************/


/*****************************************************************************/
// Header for spline surface.

// A spline holds a pointer to a class nurbsuf for spline surface,

#if !defined( SPLINE_CLASS )

#include "kernel.h"
#include "surface₩surfdef.h"

class nurbsuf;

// Define class spline

class spline: public surface {

        // Record a NURB form of spline surface

        nurbsuf *nurbsuf_ptr;

        logical rev;

public:

        // Default constructor

        spline();


        // Construct a spline from a spline.

        spline( spline const & );

        // Construct a spline with a NURB form

        spline( nurbsuf * );
```

```
// Default destructor

virtual ~spline();

// Return the surface equation.

nurbsuf *equation() const { return nurbsuf_ptr; }

logical reversed() const { return rev; }

// Data changing routine.

void set_nurb( nurbsuf * );

void set_rev( logical );

// Make a copy of this cone on the heap, and return a pointer
// to it.

virtual surface *make_copy() const;

// Return a string identifying the spline type

virtual char *type_name() const;

// Negate this spline.

virtual surface &negate();

// Assignment operator

spline &operator=( spline const & );

// Transform this spline by the given transf.

virtual surface &operator*=( transf const & );

void save_data( FILE * ) const;

void restore_data( FILE * );

// Print out details of spline.

virtual void debug(       short, FILE * = debug_file_ptr ) const;
```

```
};

#define SPLINE_CLASS
#endif
/*****************************************************************************/
```

3. 곡선, 곡면 생성 기능 API

```
/*****************************************************************************/
logical MakeLineByNURBS( position *spos, position *epos, straight *&line)
/*****************************************************************************/
//      <Fuction>
        Make line with NURBS form

//      <Input parameter>
        spos: Start position of line
        epos: End position of line

//      <Output parameter>
        line: Created line

//      <Return value>
        TRUE: No error
        FALSE: Error
/*****************************************************************************/

/*****************************************************************************/
logical MakeEllipseByNURBS( position* cen, unit_vector* norm, unit_vector* maj_axis,
        unit_vector* min_axis, double maj_rad, double min_rad, ellipse*& ell)
/*****************************************************************************/
//      <Fuction>
        Make ellipse as NURBS form

//      <Input parameter>
        cen: Center position
        norm: Normal of base plane
        maj_axis: Major axis vector
        min_axis: Minor axis vector
        maj_rad: Radius of major axis direction
        min_rad: Radius of minor axis direction

//      <Output parameter>
        ell: Created ellipse
```

```
//      <Return value>
          TRUE: No error
          FALSE: Error
/*****************************************************************************/


/*****************************************************************************/
logical GetEllipseDataFromCircleOf3Positions( position* pos1, position* pos2, position*
          pos3, position& cen, unit_vector& norm, unit_vector& maj_axis, unit_vector&
          min_axis, double &radius )
/*****************************************************************************/
//      <Fuction>
          Calculate control points of circle

//      <Input parameter>
          pos1, pos2, pos3: Three positions on circle

//      <Output parameter>
          norm: Normal of base plane
          cen: Center position of ellipse
          maj_axis: Major axis vector of ellipse
          min_axis: Minor axis vector of ellipse
          rad: Major and minor radius of ellipse

//      <Return value>
          TRUE: No error
          FALSE: Error
/*****************************************************************************/


/*****************************************************************************/
logical MakeArcByNURBS( position* cen, position* stpt, unit_vector* norm, double rad,
          double angle, ellipse*& arc )
/*****************************************************************************/
//      <Fuction>
          Make arc with NURBS form

//      <Input parameter>
          cen: Center of arc
          stpt: Start point of circular arc
          norm: Normal of base plane
          rad: Radius of circular arc
          angle: Angle of circular arc

//      <Output parameter>
          arc: Created arc
```

```
//      <Return value>
         TRUE: No error
         FALSE: Error
/*****************************************************************************/

/*****************************************************************************/
logical CalculateAnglePtToPtOfArc(position *cen,position *stpt,position *endpt,
         unit_vector *norm,double rad,double &angle)
/*****************************************************************************/
//      <Fuction>
         Calculate angle between two points of circular arc

//      <Input parameter>
         cen: center of circular arc
         stpt: start point of circular arc
         endpt: end point of circular arc
         norm: select (xy,yz,zx) plane
         rad: radius of circular arc

//      <Output parameter>
         angle: angle of circular arc

//      <Return value>
         TRUE: No error
         FALSE: Error
/*****************************************************************************/

/*****************************************************************************/
logical NurbCurveInterpolation( long no_jpt, position *pts, long order,
         enum ENDCON endtype, nurbcrv *&nu_curve )
/*****************************************************************************/
//      <Function>
         Curve interpolation using NURBS( Non-Uniform Rational B-spline )
         where as, weight is 1.

//      <Input parameter>
         no_jpt: the number of joint points
         pts: pointer of junction points
         order: order of curve
         endtype: type of end condition

//      <Output parameter>
         nu_curve: reference of class nurbcrv pointer

//      <Return value>
         TRUE: No error
```

```
        FALSE: Error
/*****************************************************************************/

/*****************************************************************************/
logical MakePlaneByNURBS( unit_vector *norm, position *pos, double u_size, double
        v_size, plane *&pln)
/*****************************************************************************/
//      <Fuction>
        Calculate control points of plane

//      <Input parameter>
        norm: Normal vector of plane
        pos: A point on plane
        u_size: u direction size of plane
        v_size: v direction size of plane

//      <Output parameter>
        pln: Created plane

//      <Return value>
        TRUE: No error
        FALSE: Error
/*****************************************************************************/

/*****************************************************************************/
logical CalculateControlPointsOfCylinder(unit_vector *base_norm,position *base_cen,
        double base_maj_rad,double base_min_rad,unit_vector *top_norm, position
        *top_cen, double top_maj_rad,double top_min_rad, surface *&nurb_suf)
/*****************************************************************************/
//      <Fuction>
        Calculate control points of cylinder

//      <Input parameter>
        base_norm,top_norm: select base(top) plane
        base_cen,top_cen: center of base(top) ellipse
        base_maj_rad,top_maj_rad: major radius of base(top) ellipse
        base_min_rad,top_min_rad: minor radius of base(top) ellipse

//      <Output parameter>
        nurb_suf: gernerated cylinder

//      <Return value>
        TRUE: No error
        FALSE: Error
/*****************************************************************************/
```

```
/*****************************************************************************/
logical CalculateControlPointsOfCylinderFromTwoCircle(position *AllPts, nurbsuf
        *&nurb)
/*****************************************************************************/
//      <Fuction>
        Calculate control points of cylinder from two circle

//      <Input parameter>
        AllPts: position of circle

//      <Output parameter>
        nurb: gernerated nurb

//      <Return value>
        TRUE: No error
        FALSE: Error
/*****************************************************************************/

/*****************************************************************************/
logical NurbSurfaceInterpolation( long nu_jpt, long nv_jpt, position *pts,
        long u_order, long v_order, enum ENDCON endtype, nurbsuf *&nu_surf )
/*****************************************************************************/
//      <Function>
        Surface interpolation using NURBS( Non-Uniform Rational B-spline )
        where as, weight is 1.

//      <Input parameter>
        nu_jpt: the number of joint points in u-direction
        nv_jpt: the number of joint points in v-direction
        jpt_net: array for junction points
        u_order: order of surface in u-direction
        v_order: order of surface in v-direction
        endtype: type of end condition

//      <Output parameter>
        nu_knot: the number of knots in u-direction
        u_knot: array for knots in u-direction
        nv_knot: the number of knots in v-direction
        v_knot: array for knots in v-direction
        nu_ctlpt: the number of control points in u-direction
        nv_ctlpt: the number of control points in v-direction
        ctlpt_net: array for control points of surface

//      <Return value>
        TRUE: No error
        FALSE: Error
/*****************************************************************************/
```

4. 형상요소의 변환기능 API

```
/*****************************************************************************/
logical SApi_CreIsoptByTranslatePoint(ISOPT *iso, vector *trvec, logical DelFlag,
        ISOPT *&niso)
/*****************************************************************************/
//      <Fuction>
        Translate Point with isopt

//      <Input parameter>
        iso: Translation 할 point
        trvec: Translation vector
        DelFlag: Translation option

//      <Output parameter>
        niso: 생성된 isopt

//      <Return value>
        TRUE: No error
        FALSE: Error
/*****************************************************************************/

/*****************************************************************************/
logical SApi_CrePgroupByTranslatePgroup(PGROUP *pgr, vector *trvec, logical DelFlag,
        PGROUP*& npgr)
/*****************************************************************************/
//      <Fuction>
        Translate Pgroup

//      <Input parameter>
        pgr: Translation 할 pgroup
        trvec: Translation vector
        DelFlag: Translation option

//      <Output parameter>
        npgr: 생성된 pgroup

//      <Return value>
        TRUE: No error
        FALSE: Error
/*****************************************************************************/

/*****************************************************************************/
logical SApi_CreVertexByTranslatePoint(VERTEX *vtx, vector *trvec, logical DelFlag,
```

```
        VERTEX *&nvtx)
/*****************************************************************************/
//      <Fuction>
        Translate Point with vertex

//      <Input parameter>
        vtx: Translation 할 point
        trvec: Translation vector
        DelFlag: Translation option

//      <Output parameter>
        nvtx: 생성된 vertex

//      <Return value>
        TRUE: No error
        FALSE: Error
/*****************************************************************************/

/*****************************************************************************/
logical SApi_CreEdgeByTranslateCurve(EDGE *edge, vector *trvec, logical DelFlag,
        EDGE *&nedge)
/*****************************************************************************/
//      <Function>
        Translate curve

//      <Input parameter>
        crv: Translation 할 curve
        trvec: Translation vector
        DelFlag: Translation option

//      <Output parameter>
        nedge: Translation 된 curve

//      <Return value>
        TRUE: No error
        FALSE: Error
/*****************************************************************************/

/*****************************************************************************/
logical SApi_CreFaceByTranslateSurface(FACE *face, vector *trvec, logical DelFlag,
        FACE *&nface)
/*****************************************************************************/
//      <Fuction>
        Translate surface

//      <Input parameter>
        face: Translate 할 surface
```

```
        trvec: Translation vector
        DelFlag: Translation option

//      <Output parameter>
        nface: Translation 된 surface

//      <Return value>
        TRUE: No error
        FALSE: Error
/*****************************************************************************/
/*****************************************************************************/
logical SApi_CreIsoptByRotatePoint(ISOPT *iso, vector *axis, double angle,logical
        DelFlag, ISOPT *&niso)
/*****************************************************************************/
//      <Fuction>
        Rotate Point

//      <Input parameter>
        iso: Rotation 할 point
        axis: Rotation axis
        angle: Rotation angle
        DelFlag: Rotation option

//      <Output parameter>
        niso: Rotate 된 point

//      <Return value>
        TRUE: No error
        FALSE: Error
/*****************************************************************************/

/*****************************************************************************/
logical SApi_CrePgroupByRotatePgroup(PGROUP *pgr, vector* axis, double angle,
        logical DelFlag, PGROUP*& npgr)
/*****************************************************************************/
//      <Fuction>
        Rotate Pgroup

//      <Input parameter>
        pgr: Rotation 할 pgroup
        axis: Rotation axis
        angle: Rotation angle
        DelFlag: Rotation option

//      <Output parameter>
        npgr: Rotate 된 pgroup
```

```
//      <Return value>
         TRUE: No error
         FALSE: Error
/*****************************************************************************/

/*****************************************************************************/
logical     SApi_CreVertexByRotatePoint(VERTEX *vtx, vector *axis, double angle,
             logical DelFlag, VERTEX *&nvtx)
/*****************************************************************************/
//      <Fuction>
         Rotate Point

//      <Input parameter>
         vtx: Rotation 할 point
         axis: Rotation axis
         angle: Rotation angle
         DelFlag: Rotation option

//      <Output parameter>
         nvtx: Rotate 된 point

//      <Return value>
         TRUE: No error
         FALSE: Error
/*****************************************************************************/

/*****************************************************************************/
logical SApi_CreEdgeByRotateCurve(EDGE *edge, vector *axis, double angle,
        logical DelFlag, EDGE *&nedge)
/*****************************************************************************/
//      <Function>
         Rotate curve

//      <Input parameter>
         edge: Rotation 할 curve
         axis: Rotation axis
         angle: Rotation angle
         DelFlag: Rotation option

//      <Output parameter>
         nedge: Rotation 된 curve

//      <Return value>
         TRUE: No error
         FALSE: Error
/*****************************************************************************/
```

```
/*************************************************************************/
logical     SApi_CreFaceByRotateSurface(FACE *face, vector *axis, double angle,
             logical DelFlag, FACE *&nface)
/*************************************************************************/
//      <Fuction>
         Rotate surface

//      <Input parameter>
         face: Rotate 할  surface
         axis: Rotation axis
         angle: Roatation angle
         DelFlag: Rotation option

//      <Output parameter>
         nface: Rotate 된  surface

//      <Return value>
         TRUE: No error
         FALSE: Error
/*************************************************************************/

/*************************************************************************/
logical SApi_CreIsoptByScalePoint(ISOPT *iso, double xfactor, double yfactor, double
         zfactor, logical DelFlag, ISOPT *&niso)
/*************************************************************************/
//      <Fuction>
         Scale Point

//      <Input parameter>
         iso: Scale 할  point
         xfactor, yfactor, zfactor:   Local Scale factor
         DelFlag: Scale option

//      <Output parameter>
         niso: Scale 된  point

//      <Return value>
         TRUE: No error
         FALSE: Error
/*************************************************************************/

/*************************************************************************/
logical SApi_CreIsoptByScalePoint(ISOPT *iso, double factor, logical DelFlag, ISOPT
         *&niso)
/*************************************************************************/
//      <Fuction>
```

Scale Point

// <Input parameter>
iso: Scale 할 point
factor: Overall Scale factor
DelFlag: Scale option

// <Output parameter>
niso: Scale 된 point

// <Return value>
TRUE: No error
FALSE: Error
/***/

/***/
logical SApi_CrePgroupByScalePgroup(PGROUP *pgr, double xfactor, double yfactor,
double zfactor, logical DelFlag, PGROUP*& npgr)
/***/
// <Fuction>
Scale Pgroup

// <Input parameter>
pgr: Scale 할 pgroup
xfactor,yfactor,zfactor: Local Scale factor
DelFlag: Scale option

// <Output parameter>
npgr: Scale 된 pgroup

// <Return value>
TRUE: No error
FALSE: Error
/***/

/***/
logical SApi_CrePgroupByScalePgroup(PGROUP *pgr, double factor, logical DelFlag,
PGROUP*& npgr)
/***/
// <Fuction>
Scale Pgroup

// <Input parameter>
pgr: Scale 할 pgroup
factor: Overall Scale factor
DelFlag: Scale option

```
//      <Output parameter>
         npgr: Scale 된 pgroup

//      <Return value>
         TRUE: No error
         FALSE: Error
/*****************************************************************************/

/*****************************************************************************/
logical SApi_CreVertexByScalePoint(VERTEX *vtx, double xfactor, double yfactor,
         double zfactor, logical DelFlag, VERTEX *&nvtx)
/*****************************************************************************/
//      <Fuction>
         Scale Point

//      <Input parameter>
         vtx: Scale 할 point
         xfactor,yfactor,zfactor: Local Scale factor
         DelFlag: Scale option

//      <Output parameter>
         nvtx: Scale 된 point

//      <Return value>
         TRUE: No error
         FALSE: Error
/*****************************************************************************/

/*****************************************************************************/
logical SApi_CreVertexByScalePoint(VERTEX *vtx, double factor, logical DelFlag,
         VERTEX *&nvtx)
/*****************************************************************************/
//      <Fuction>
         Scale Point

//      <Input parameter>
         vtx: Scale 할 point
         factor: Overall Scale factor
         DelFlag: Scale option

//      <Output parameter>
         nvtx: Scale 된 point
//      <Return value>
         TRUE: No error
         FALSE: Error
/*****************************************************************************/
```

```
/*****************************************************************************/
logical SApi_CreEdgeByScaleCurve(EDGE *edge, double xfactor, double yfactor,
        double zfactor, logical DelFlag, EDGE *&nedge)
/*****************************************************************************/
//      <Function>
        Scale curve

//      <Input parameter>
        edge: Scale 할 curve
        xfactor, yfactor, zfactor: Local Scale factor
        DelFlag: Scale option

//      <Output parameter>
        nedge: Scale 된 curve

//      <Return value>
        TRUE: No error
        FALSE: Error
/*****************************************************************************/

/*****************************************************************************/
logical SApi_CreEdgeByScaleCurve(EDGE *edge, double factor, logical DelFlag,
        EDGE *&nedge)
/*****************************************************************************/
//      <Function>
        Scale curve with global sacle factor

//      <Input parameter>
        edge: Scale 할 curve
        factor: Overall Scale factor
        DelFlag: Scale option

//      <Output parameter>
        nedge: Scale 된 curve

//      <Return value>
        TRUE: No error
        FALSE: Error
/*****************************************************************************/

/*****************************************************************************/
logical SApi_CreFaceByScaleSurface(FACE *face, double xfactor, double yfactor,
        double zfactor, logical DelFlag, FACE *&nface)
/*****************************************************************************/
//      <Fuction>
        Scaling surface with Local sacle factor
```

```
//      <Input parameter>
         face: Scale 할 surface
         xfactor, yfactor, zfactor: Local Scale factor
         DelFlag: Scale option

//      <Output parameter>
         nface: Scale 된 surface

//      <Return value>
         TRUE: No error
         FALSE: Error
/*****************************************************************************/

/*****************************************************************************/
logical SApi_CreFaceByScaleSurface(FACE *face, double factor, logical DelFlag,  FACE
        *&nface)
/*****************************************************************************/
//      <Fuction>
         Scaling surface with Overall sacle factor

//      <Input parameter>
         face: Scale 할 surface
         factor: Overall Scale factor
         DelFlag: Scale option

//      <Output parameter>
         nface: Scale 된 surface

//      <Return value>
         TRUE: No error
         FALSE: Error
/*****************************************************************************/
```

C. 데이터 구조 및 파일에 관련된 클래스와 API

1. 위상 요소의 클래스

```
/*****************************************************************************/
/*                                                                           */
/*       SURFART   KIST (Korea Institute of Science and Technology)          */
/*                                                                           */
/*       This program is the property of KIST and is protected by            */
/*       copyright under the laws of Korea. This program is not to be        */
/*       disclosed without written authorization from KIST.  Any use,        */
/*       duplication or disclosure of this program by other than KIST        */
/*       and  their assigned licensees and customers is strictly for-        */
/*       bidden by law.                                                      */
/*                                                                           */
/*       Copyright (c) 1995, 1996, 1997 by                                   */
/*       Korea Institute of Science and Technology                           */
/*       All rights reserved                                                 */
/*                                                                           */
/*****************************************************************************/
// ($$) entity.h   1.0                  10/23/95

// Header for ENTITY.

// ENTITY is the common class for all toplogical entities in data structure.
// It defines the base class for all types.

// We provide a range of macros to declare and define a large number
// of the standard member functions, used by most derived classes.

#if !defined( ENTITY_CLASS )

#include <stdio.h>
#include "kernel.h"


// Macros for simplifying the definition of user entities. There are
// rather a lot of member functions required for any entity, but
// many of them are the same for all (at the source level),
// and some of the others have elements in common. These macros are
// to help with this commonality.


// Declare the common utility routines for the attribute header file.

#define ENTITY_FUNCTIONS( name ) ₩
private: ₩
        virtual ENTITY *make_copy() const; ₩
```

```
public: ₩
        virtual ~name(); ₩
        virtual char *type_name() const; ₩
        virtual unsigned size() const; ₩
        virtual void debug_ent( short, FILE * ) const; ₩
₩
public: ₩
        virtual void lose();

// ============================================================
// Now declare the base class ENTITY, from which all data structure
// objects are derived.
// ============================================================

class ATTRIB;

// The generic data structure entity, of which all the specific types
// are subclasses.

class ENTITY {

        // Derivation type of this class from ENTITY.
        int class_type;

        // Pointer to start of chain of attributes attached to this entity
        // (NULL if entity has no attributes).
        ATTRIB *attrib_ptr;

        // Declare the standard set of generic ENTITY functions
        ENTITY_FUNCTIONS( ENTITY )

public:
        // Entity constructor, initializing common entries
        ENTITY();

        // Data reading routines.
        int type() const { return class_type; }
        ATTRIB *attrib() const { return attrib_ptr; }

        // Member setting function.
        void set_attrib( ATTRIB * );
        void set_type( int );
};

#define ENTITY_CLASS
#endif

/*****************************************************************************/

/*****************************************************************************/
```

```
// ($$) body.h                  1.0                  10/23/95

// Header for BODY.

// This class models a wire, sheet, solid body (which may be several
// disjoint bodies treated as a collection of lumps) or points group.

// Lumps represent partially-surfaced wire-frames, sheets and solids.
// In a partially-surfaced wire-frame, edges will border on zero,
// one or two faces, whereas edges in a sheet will meet one or two (or
// more) faces.  In a manifold solid, every edge is adjacent to two
// faces (a non-manifold solid has edges adjacent to more than two
// faces or more than one set of faces at a vertex).

// A body can in principle contain both wire and lump components.
// The convention is followed that they should not be connected to
// one another (in other words, there should be only one path from
// the body to any edge - via the wire or lump pointer, but not
// both).

// A solid is represented by the boundary of the region of space
// enclosed, modelled as a lump composed of one or more disjoint
// SHELLs.

// The geometry of lumps and wires are given in a local coordinate
// system, related to the universal one by a transformation stored
// with the body.

#if !defined( BODY_CLASS )

#include "kernel.h"
#include "top₩tophdr.h"
#include "top₩entity.h"

class TRANSFORM;
class box;

// BODY declaration.
class BODY: public ENTITY {

        // Pointer to the start of a list of lumps.
LUMP *lump_ptr;

// Pointer to the start of a list of pgroups.
PGROUP *pgroup_ptr;

// This transformation relates the local coordinate system to the
// global one in which the body resides.
TRANSFORM *transform_ptr;
```

```
// Pointer to a geometric bounding region (a box), within which the
// entire body lies (with respect to its internal coordinate system).
// It may be NULL if no such bound has been calculated since the
// body was last changed.
box *bound_ptr;

// Include the standard member functions for all entities.
ENTITY_FUNCTIONS( BODY )

// Now the functions specific to BODY.
// Basic constructor.
BODY();
BODY( LUMP * );    // for a body containing a lump initially
BODY( PGROUP * );  // for a body containing a points group initially

// Data reading routines.
LUMP *lump() const { return lump_ptr; }
PGROUP *pgroup() const { return pgroup_ptr; }
TRANSFORM *transform() const { return transform_ptr; }
box *bound() const { return bound_ptr; }

// Data changing routines.
void set_lump( LUMP * );
void set_pgroup( PGROUP * );
void set_transform( TRANSFORM * );
        void set_bound( box * );
};

#define BODY_CLASS
#endif

/*****************************************************************************/

/*****************************************************************************/

// ($$) lump.h     1.0                    10/23/95

// Header for LUMP.

// The lump is a connected portion of a body.  Usually a body will
// consist of a single lump (which is bounded by an outer shell and
// zero, one or more inner shells representing voids within the body).
// However when an operation such as boolean subtraction or
// intersection returns a body composed of several pieces, each piece
// will be represented by a lump.

#if !defined( LUMP_CLASS )

#include "kernel.h"
```

```
#include "top₩tophdr.h"
#include "top₩entity.h"

class box;

// LUMP declaration.

class LUMP: public ENTITY {

        // List pointer linking all the lumps in a body (last member
        // of list has this pointer NULL).
        LUMP *next_ptr;

        // Start of a list of shells that comprise the lump.
        SHELL *shell_ptr;

        // Pointer to the owning body.
        BODY *body_ptr;

        // Pointer to a geometric bounding region (a box), within which the
        // entire lump lies (with respect to the internal coordinate
        // system of its body).
        // It may be NULL if no such bound has been calculated since the
        // lump was last changed.
        box *bound_ptr;

        // Include the standard member functions for all entities.
        ENTITY_FUNCTIONS( LUMP )

        // Now the functions specific to LUMP.
        // Basic constructor.
        LUMP();

        // Public constructor, which initializes all the class data.
        // The first argument is the start of a lists of shells contained
        // in the lump, and the second argument sets the pointer to
        // the next lump in the owning body.
        // The constructor sets the backpointers in the shells in the given
        // list.  The calling routine must set body_ptr and if desired,
        // bound_ptr, using set_body() and set_bound() [declared below].
        LUMP( SHELL *, LUMP * );

        // Data reading routines.
        LUMP *next() const { return next_ptr; }
        SHELL *shell() const { return shell_ptr; }
        BODY *body() const { return body_ptr; }
        box *bound() const { return bound_ptr; }

        // Data changing routines.
        void set_next( LUMP * );
```

```
        void set_shell( SHELL * );
        void set_body( BODY * );
        void set_bound( box * );

};

#define LUMP_CLASS
#endif

/*************************************************************************/

// ($$) shell.h                1.0                10/23/95

// Header for SHELL.

// The shell is one portion of a lump's boundary - it has no internal
// connection with any other shell.  If a lump has no voids, then
// exactly one shell gives its overall extent; any other shells bound
// voids wholly within the lump.  There is no distinction made in the
// data structure between peripheral and void shells.  In this
// context a shell is closed and bounded.

// A shell is constructed from a collection of "faces", each of
// which is a bounded or unbounded piece of a single geometric surface.

#if !defined( SHELL_CLASS )

#include "kernel.h"
#include "top₩tophdr.h"
#include "top₩entity.h"

class box;

// SHELL declaration proper.

class SHELL: public ENTITY {

        // List pointer linking all the shells in a body (last member
        // of list has this pointer NULL).
        SHELL *next_ptr;

        // Pointer to child entity: one of face, edge or vertex.
        ENTITY *child_ptr;

        // Pointer to the owning lump - shells in separate lumps are
        // themselves entirely separate.
        LUMP *lump_ptr;
```

```
// Pointer to a geometric bounding region (a box), within which the
// entire shell lies (with respect to the internal coordinate
// system of its body).
// It may be NULL if no such bound has been calculated since the
// shell was last changed.
box *bound_ptr;

// Include the standard member functions for all entities.
ENTITY_FUNCTIONS( SHELL )

// Now the functions specific to SHELL.
// Basic constructor.
SHELL();

// Public constructor, which initializes all the class data.
// The first two arguments are the starts of lists of faces,
// and the last is a list of "sister" shells already
// in the current lump.  The calling routine must set lump_ptr and
// if desired, bound_ptr, using set_lump() and set_bound()
// [declared below].
SHELL( FACE *, SHELL * );

// Single vertex shell
SHELL( VERTEX *, SHELL * );

        // Data reading routines.
        SHELL *next() const { return next_ptr; }
        ENTITY *child() const { return child_ptr; }
        LUMP *lump() const { return lump_ptr; }
        box *bound() const { return bound_ptr; }

        // Data changing routines.
        void set_next( SHELL * );
        void set_child( ENTITY * );
        void set_lump( LUMP * );
        void set_bound( box * );

};

#define SHELL_CLASS
#endif

/*****************************************************************************/

/*****************************************************************************/

// ($$) face.h     1.0                    10/23/95

// Header for FACE.
```

```
// A FACE is a bounded portion of a single geometric surface in space
// - the two-dimensional analogue of the BODY.  The boundary is
// represented by one or more LOOPs of edges.  As with a body, a face
// can in principle be in several disjoint parts, but modelling
// algorithms frequently have to determine the disjoint pieces for
// different treatment, so generally each face may be assumed to be
// connected.  It is not meaningful in general to distinguish exterior
// and interior loops of edges, though for certain surface types this
// may be possible, and some algorithms may do so.

// Face "loops" need not necessarily be closed, and if not, either
// open end may be finite or infinite.  If either end is infinite, then
// the face is infinite; if either end is finite, then the face is
// "incompletely-bounded", or just "incomplete".

#if !defined( FACE_CLASS )

#include "kernel.h"
#include "top₩tophdr.h"
#include "top₩entity.h"

class box;

// Type for a marker to distinguish single-sided (i.e. boundary) faces
// and double-sided ones (i.e. sheets or embedded).

typedef logical SIDESBIT;
#define SINGLE_SIDED FALSE
#define DOUBLE_SIDED TRUE

// Type for a marker to distinguish the type of a double-sided face.
// Meaningless for a single-sided face.

typedef logical CONTBIT;
#define BOTH_OUTSIDE FALSE
#define BOTH_INSIDE TRUE

// FACE declaration.

class FACE: public ENTITY {

// Pointer to next face in list contained directly
// by SHELL.
FACE *next_ptr;


// Pointer to child entity: first loop of edges bounding face
// or wireframe edge.
ENTITY *child_ptr;
```

```
// Pointer to shell containing face.
SHELL *shell_ptr;

// Face geometry.  Every surface is directed - that is its normal
// direction is a continuous function of position.  The normal to
// the face may be the same as that of the surface at any position,
// or may be the reverse of it, as determined by sense_data.  This
// member is of an enumerated type, with values FORWARD or REVERSED.
// When a face bounds a region of space, then its normal is always
// chosen to point away from the region bounded - that is it is an
// outward normal.
SURFACE *geometry_ptr;
REVBIT sense_data;

// Faces may also be double-sided, either free surfaces in space
// or embedded in material. Distinguish between single- and double-
// sided faces, and in the latter case between in-space and in-
// material.
SIDESBIT sides_data;
CONTBIT cont_data;

// Pointer to a geometric bounding region (a box), within which the
// entire body lies (with respect to its internal coordinate system).
// It may be NULL if no such bound has been calculated since the
// body was last changed.
box *bound_ptr;

// Include the standard member functions for all entities.
ENTITY_FUNCTIONS( FACE )

// Now the functions specific to FACE.
// Basic constructor.
FACE();

// Public constructor.  The arguments initialise the members
// loop_ptr, next_ptr, geometry_ptr and sense_data respectively.
// It also sets the backpointers (to the face) in the loops which
// therefore must be correctly chained together before this
// constructor is called.
// The calling routine must set shell_ptr to refer to the owning shell,
// and if desired, bound_ptr, using set_shell() and set_bound()
// [declared below].
FACE( FACE *, LOOP *, SURFACE *, REVBIT );


// Virtual face for wireframe edge
FACE( FACE *, EDGE * );

// Data reading routines.
FACE *next() const { return next_ptr; }
```

```
ENTITY *child() const { return child_ptr; }
SHELL *shell() const { return shell_ptr; }
SURFACE *geometry() const { return geometry_ptr; }
REVBIT sense() const { return sense_data; }
REVBIT sense( REVBIT sense ) const
{ if (sense == FORWARD) return sense_data;
  else return sense_data == FORWARD ? REVERSED : FORWARD;
}
SIDESBIT sides() const { return sides_data; }
CONTBIT cont() const { return cont_data; }
box *bound() const { return bound_ptr; }

// Data changing routines.
void set_next( FACE * );
void set_child( ENTITY * );
void set_shell( SHELL * );
void set_geometry( SURFACE * );
void set_sense( REVBIT );
void set_sides( SIDESBIT );
void set_cont( CONTBIT );
void set_bound( box * );

};

#define FACE_CLASS
#endif

/*****************************************************************************/

/*****************************************************************************/

// ($$) loop.h     1.0                    10/23/95

// Header for LOOP.

// A loop represents a connected portion of the boundary of a FACE.
// Despite the name, it may not be a closed loop - see the description
// of FACE (above) for details.

#if !defined( LOOP_CLASS )
#include "kernel.h"
#include "top₩tophdr.h"
#include "top₩entity.h"

class box;

// LOOP declaration.

class LOOP: public ENTITY {
```

```
// Next loop in list bounding a particular face (NULL if this is
// the last loop in the list).
LOOP *next_ptr;

// One "edge" in the loop.  If the loop is closed, no significance
// is placed upon the choice of this edge, which may change during
// modelling operations.  If a loop is not closed, then its start
// coedge pointer always points to the first coedge in the "next"
// list, so that a loop may always be traversed completely from this
// start coedge simply by following the next pointer in each coedge
// until this becomes NULL or returns to the start coedge.
COEDGE *start_ptr;

// Back pointer to containing face.  Each loop may belong to
// only one face.
FACE *face_ptr;

// Pointer to a geometric bounding region (a box), within which the
// loop lies (with respect to the internal coordinate system of its
// owning body).
// It may be NULL if no such bound has been calculated since the
// body was last changed.
box *bound_ptr;

// Include the standard member functions for all entities.
ENTITY_FUNCTIONS( LOOP )

// Now the functions specific to LOOP.
// Basic constructor.
LOOP();

// Public constructor which initializes the record.
// The arguments initialise coedge_ptr and loop_ptr respectively.
// If the given coedge is not NULL, the constructor also sets
// back pointers to the new loop in each coedge of the loop
// (hence the coedges must be correctly linked together
// before this constructor is called).
LOOP( COEDGE *, LOOP * );

// Data reading routines.
LOOP *next() const { return next_ptr; }
COEDGE *start() const { return start_ptr; }
FACE *face() const { return face_ptr; }
box *bound() const { return bound_ptr; }

// Data changing routines.
void set_next( LOOP * );
void set_start( COEDGE * );
void set_face( FACE * );
```

```
        void set_bound( box * );
        void make_bound();

};

#define LOOP_CLASS
#endif

/*****************************************************************************/

// ($$) coedge.h   1.0                  10/23/95

// Header for COEDGE.

// The coedge is closely related to an edge, and provides room to
// store its relationships with adjacent edges and with superior owning
// entities.  The structures formed by these pointers and their
// interpretation depends upon the exact nature of the owning entity.

// The normal case is when the associated edge is part of a
// well-formed solid body shell, being adjacent to exactly two faces.
// There are two coedges, each associated with a loop in one of the
// faces (in principle the two faces could be the same, and even the
// loops could be the same).  All the coedges in each loop are linked
// into a doubly-linked circular list through the next and previous
// pointers, and the two coedges for each edge are linked through
// their partner pointers.

// There are several extensions possible to this simple arrangement.
// Firstly, a loop may not necessarily be closed, for either a
// partially-defined or infinite face boundary.  In this case, the
// next and previous lists are not circular, but terminate with NULL
// pointers.  Secondly, a shell may not be closed, and have "free"
// edges at its boundary.  For such edges, there is only one coedge,
// with a NULL partner pointer.

// A third extension is to non-manifold shells, in which more than
// two faces meet in an edge.  In this case, the partner pointers for
// the coedges (still one for each face) are linked in a circular list.

// An entirely different owning entity is the wire.  An object may
// be a directed or undirect graph, made up of one or more disjoint
// wires, each of which is a collection of connected edges.  In this
// case, each edge has exactly one coedge, and the coedges are linked
// in circular lists around each vertex, using next and previous
// pointers according to which end of the coedge lies at the vertex.

// Finally, a shell may be of mixed dimensionality, containing both
```

```
// faces and unembedded edges.  The unembedded edges are connected
// together as in wires, and are connected to any faces sharing their
// vertices in a similar way.  The only additional rule is that in
// each face, the next list always yields the loop of edges embedded
// in the face - any unembedded edge is connected via the previous
// pointer to one of the face coedges.

#if !defined( COEDGE_CLASS )

#include "kernel.h"
#include "top₩tophdr.h"
#include "top₩entity.h"

// COEDGE declaration.

class COEDGE: public ENTITY {

        // Pointers to provide a doubly-linked list of coedges in a loop,
        // or circular lists at each end in a general unembedded graph.
        COEDGE *next_ptr;
        COEDGE *previous_ptr;

        // Pointer to partner coedge, or NULL if this coedge is unembedded
        // or attached to a free edge.
        COEDGE *partner_ptr;

        // Pointer to child entity: one of edge or vertex
        ENTITY *child_ptr;

        // Relationship between the direction of the coedge and that of the
        // underlying edge.  When embedded in a face, the coedges must run
        // clockwise about the (outward) face normal, that is at any point
        // on the coedge, if the face normal is "upwards" and the coedge
        // tangent is "forwards", then the face lies to the "left".
        REVBIT sense_data;

        // Pointer to the owning loop or wire.  There is always a loop if
        // the coedge is embedded in a face, or a wire if it is part of
        // an unembedded graph.  If the coedge is an unembedded one in a
        // mixed-dimensionality shell, then this pointer is NULL.
        LOOP *loop_ptr;

        // Pointer to the description of the edge geometry referred to the
        // parametric space of the face in which it is embedded.  This will
        // be NULL if the edge is  not embedded, or if the face is not
        // parametrically described.  It may be NULL even if the face is
        // parametric.
        PCURVE *geometry_ptr;

        // Include the standard member functions for all entities.
```

```
ENTITY_FUNCTIONS( COEDGE )

// Now the functions specific to COEDGE.
// Basic constructor.
COEDGE();

// Public constructor, which initializes the record.
// The arguments initialize edge_ptr (and indirectly partner_ptr),
// sense_data, next_ptr and previous_ptr respectively.
// Coedge back-pointers are also set in the two argument coedges,
// but presently only valid if all the coedges are part of
// a conventional simple loop.
COEDGE( EDGE *, REVBIT, COEDGE *, COEDGE * );

// Construct coedge for single vertex loop
COEDGE( VERTEX * );

// Data reading routines.
COEDGE *next() const { return next_ptr; }
COEDGE *next( REVBIT sense ) const
{       if (sense == FORWARD) return next_ptr;
        else return previous_ptr; }
COEDGE *previous() const { return previous_ptr; }
COEDGE *previous( REVBIT sense ) const
{       if (sense == FORWARD) return previous_ptr;
        else return next_ptr; }
COEDGE *partner() const { return partner_ptr; }
ENTITY *child() const { return child_ptr; }
REVBIT sense() const { return sense_data; }
REVBIT sense( REVBIT sense ) const
{       if (sense == FORWARD) return sense_data;
        else return sense_data == FORWARD ? REVERSED : FORWARD; }
LOOP *loop() const { return loop_ptr; }
PCURVE *geometry() const { return geometry_ptr; }

// Data changing routines.
void set_next( COEDGE * );
void set_previous( COEDGE * );
void set_partner( COEDGE * );
void set_child( ENTITY * );
void set_sense( REVBIT );
        void set_loop( LOOP * );
        void set_geometry( PCURVE * );

};

#define COEDGE_CLASS
#endif

/*****************************************************************************/
```

```
/*********************************************************************/

// ($$) edge.h     1.0                    10/23/95

// Header for EDGE.

// The edge class represents the physical edge as recognized by the
// user.  It consists of a bounded portion of a space curve, the bounds
// being given as object-space vertices (any parametric bounds required
// for a parametric curve are recorded with the edge - see below).
// The vertex pointer at either or both ends may be NULL,
// in which case the edge is taken to be unbounded in that direction.
// If the underlying curve is infinite, then so is the unbounded edge;
// if the curve is closed, then the vertex pointers must both be present
// or both NULL, and in the latter case the edge is the whole curve.

// As a special case, the geometry pointer may be NULL, while both
// vertex pointers point to the same vertex.  This is taken to mean that
// the "edge" is in fact an isolated point (for example the apex of a
// cone).

#if !defined( EDGE_CLASS )

#include "kernel.h"
#include "top₩tophdr.h"
#include "top₩entity.h"

class box;
class position;
class parameter;

// EDGE declaration.

class EDGE: public ENTITY {

        // Pointers to the vertices of the edge. They may be NULL, with
        // meanings as described above.
        VERTEX *start_ptr;
        VERTEX *end_ptr;

        // Pointer to one of the coedges lying on this edge.
        // No significance attaches to the choice of coedge,
        // which may change during a modelling operation.
        ENTITY *parent_ptr;

        // The geometric shape of the edge.  Every curve description is of
        // a directed curve; the sense defines whether the edge shares this
        // direction, or has the opposite one.
        CURVE *geometry_ptr;
```

```
REVBIT sense_data;

// Pointer to a geometric bounding region (a box), within which the
// entire edge lies (with respect to its body's internal
// coordinate system).
// It may be NULL if no such bound has been calculated since the
// edge was last changed.
box *bound_ptr;

// Pointer to a parametric bounding region.
parameter *start_param_ptr;
parameter *end_param_ptr;


// Include the standard member functions for all entities.
ENTITY_FUNCTIONS( EDGE )

// Now the functions specific to EDGE.
// Basic constructor.
EDGE();

// Public constructor which initializes the record.
// The arguments initialize start_ptr, end_ptr, geometry_ptr
// and sense_data respectively.
// If necessary, the edge pointers in the two end vertices are set,
// and the use count in the curve geometry is incremented.
EDGE( VERTEX *, VERTEX *, CURVE *, REVBIT );

// Data reading routines.
VERTEX *start() const { return start_ptr; }
VERTEX *end() const { return end_ptr; }
ENTITY *parent() const { return parent_ptr; }
CURVE *geometry() const { return geometry_ptr; }
REVBIT sense() const { return sense_data; }
box *bound() const { return bound_ptr; }
parameter *start_param() const { return start_param_ptr; }
parameter *end_param() const { return end_param_ptr; }

// Data changing routines.
void set_start( VERTEX * );
void set_end( VERTEX * );
void set_parent( ENTITY *, logical = FALSE );
void set_geometry( CURVE * );
void set_sense( REVBIT );
void set_bound( box * );
void set_start_param( parameter * );
void set_end_param( parameter * );

};
```

```
#define EDGE_CLASS
#endif

/*********************************************************************************/

/*********************************************************************************/

// ($$) vertex.h   1.0                    10/23/95

// Header for VERTEX.

// The vertex is the embodiment of the user's view of a corner of a
// face.  It refers to a point in object space, and to one or more
// of the edges which it bounds.

// If all the edges at the vertex are wire (each adjacent to no faces)
// or if all are embedded (each adjacent to two faces), the vertex
// will contain a pointer to one of the edges.  The others can be
// found by following the next, previous and partner pointers of the
// coedges of the edges as appropriate for wires or embedded edges.

// Otherwise the vertex will contain pointers to more than one edge,
// from which all the edges at the vertex can be reached.  This will
// be so, for example, when a body is non-manifold at a vertex, or
// when an unembedded edge dangles from a vertex of an otherwise
// well-formed solid.

#if !defined( VERTEX_CLASS )

#include "kernel.h"
#include "top₩tophdr.h"
#include "top₩entity.h"
#include "misc₩ptrlist.h"

// VERTEX declaration.

class VERTEX: public ENTITY {

        // Pointer to parent entity: one of edge, coedge, shell
        ENTITY *parent_ptr;

// Pointer to position of vertex in space.
GPOINT *geometry_ptr;

// List of edges which are connected to this vertex
PTRLIST<EDGE*> elist;

// Include the standard member functions for all entities.
ENTITY_FUNCTIONS( VERTEX )
```

```
// Now the functions specific to VERTEX.
// Basic constructor.
VERTEX();

// Public constructor, which initializes the record.
// Increments the point's use count to reflect this new use.
VERTEX( GPOINT * );

// Data reading routines.
// Get an edge at a vertex known to contain at most one pointer
// to an edge
ENTITY *parent() const { return parent_ptr; }
unsigned count() const { return elist.count(); }

// Test if the given edge is pointed to by the vertex.
logical edge_linked( EDGE * ) const;

// Get geometry of vertex
GPOINT *geometry() const { return geometry_ptr; }

// Get edge-list of vertex
PTRLIST<EDGE*> *edge_list() { return &elist; }

// Data changing routines.
// Delete any existing pointers to edges in the vertex and then
// place a pointer to the given edge in the vertex.
void set_parent( ENTITY *, logical = FALSE );
void set_edge( EDGE * );
void del_edge( EDGE * );
void set_geometry( GPOINT * );

};

#define VERTEX_CLASS
#endif

/*****************************************************************************/

/*****************************************************************************/

// ($$) pgroup.h   1.0                    11/13/95

// Header for PGROUP.

// A pgroup represents a group of isolated points.
// It is owned by a body.

// Usually a pgroup has a pointer to a start point of the isolated point
// group which was red from a points data file. If the points data file has
// no groups or section, all of points red from points data file were assumed
```

```
// to one group of points. However, if the point data file has sectioned or
// grouped points data, each group or section has a pgroup and is linked by
// next pointer to the pgroup.

#if !defined( PGROUP_CLASS )

#include "kernel.h"
#include "top₩tophdr.h"
#include "top₩entity.h"

class box;

// PGROUP declaration.

class PGROUP: public ENTITY {

        // Next pgroup in list of all pgroups of the owning body.
        PGROUP *next_ptr;  // list pointer to pgroups in a body

        // Start point in isolated points group
        ISOPT *isopt_ptr;

        // Back pointer to owning body.
        // If the pgroup isn't first pgroup in body, this is NULL
        BODY *body_ptr;

        // Pointer to a geometric bounding region (a box), within which the
        // entire body lies (with respect to its internal coordinate system)
        // It may be NULL if no such bound has been calculated since the
        // body was last changed.
        box *bound_ptr;              // bounding box

        // Include the standard member functions for all entities.
        ENTITY_FUNCTIONS( PGROUP )

        // Now the functions specific to WIRE.
        // Basic constructor.
        PGROUP();

        // Public constructor.  The arguments initialize isopt_ptr
        // and next_ptr respectively.  The calling routine must set the
        // owning body using set_body.
        PGROUP( ISOPT *, PGROUP * );

        // Data reading routines.
        PGROUP *next() const { return next_ptr; }
        ISOPT *isopt() const { return isopt_ptr; }
        BODY *body() const { return body_ptr; }
        box *bound() const { return bound_ptr; }
```

```
        // Data changing routines.
        void set_next( PGROUP * );
        void set_isopt( ISOPT * );
        void set_body( BODY * );
        void set_bound( box * );

};

#define PGROUP_CLASS
#endif
```

```
/*****************************************************************************/

/*****************************************************************************/

// ($$) isopt.h                1.0                11/13/95

// Header for ISOPT.

// The isopt is the embodiment of the user's view.
// It refers to only a point which is red from data file.
// If a user creates a geometrically isolated point in object space,
// it is not a ISOPT but a VERTEX.

#if !defined( ISOPT_CLASS )
#include "kernel.h"
#include "top₩tophdr.h"
#include "top₩entity.h"

// ISOPT declaration.

class ISOPT: public ENTITY {

        // Previous isopt in group.
        ISOPT *previous_ptr;

        // Next isopt in pgroup.
        ISOPT *next_ptr;

        // Pointer to one pgroup in which the isopt lies.
PGROUP *pgroup_ptr;

// Pointer to position of isopt in space.
GPOINT *geometry_ptr;

// Include the standard member functions for all entities.
ENTITY_FUNCTIONS( ISOPT )

// Now the functions specific to ISOPT.
```

```
// Basic constructor.
ISOPT();

// Public constructor, which initializes the record.
ISOPT( ISOPT *, GPOINT * );

// Data reading routines.
PGROUP *pgroup() const { return pgroup_ptr; }
ISOPT *previous() const { return previous_ptr; }
ISOPT *next() const { return next_ptr; }
GPOINT *geometry() const { return geometry_ptr; }

// Data changing routines.
// Delete any existing pointers to pgroup in the isopt and then
// place a pointer to the given pgroup in the isopt.
void set_pgroup( PGROUP * );

// Change previous or next isopt
void set_previous( ISOPT * );
void set_next( ISOPT * );

// Change the geometry of point
void set_geometry( GPOINT * );

};

#define ISOPT_CLASS
#endif

/*****************************************************************************/


/*****************************************************************************/

// ($$) surface.h  1.0                    10/24/95

// Header for SURFACE.

// The SURFACE provides the basic framework for the range of surface
// geometries implemented at any time in the modeller.  It provides a
// use-count, so that surfaces may be multiply referenced.


#if !defined( SURFACE_CLASS )

#include "kernel.h"
#include "topWentity.h"
#include "surfaceWsurfdef.h"

class LOOP;
```

```
class transf;
class box;

// SURFACE declaration.

class SURFACE: public ENTITY {

        // The use-count holds the number of faces referencing the SURFACE
        unsigned int use_count_data;

        // Record a geomtric surface.
        surface *surface_ptr;

        // Include the standard member functions for all entities.
        ENTITY_FUNCTIONS( SURFACE )

        // Now the functions specific to SURFACE.
        // Make a generic surface. Initializes the use count to zero.
        SURFACE();

        // Create a SURFACE from a geometric surface.
        SURFACE( surface * );

        // Return the use_count.
        int use_count() const { return use_count_data; }

        // Return the surface pointer
        surface *geo_surface() const { return surface_ptr; }

        // Set surface pointer as given geometric surface
        void set_surface( surface * );

        // Use count manipulation. Either add or subtract one use, and if
        // subtraction causes the use count to fall to zero, then delete
        // the record.
        void add();
        void remove();

        // Transform a surface.
        void operator*=( transf const & );

        // Make a bounding box for this surface, surrounded by a set of loops
        // of edges. For the present we merely find the box containing the
        // whole underlying surface, ignoring the bounding edges. Provided
        // we keep the surface minimal, this is probably sufficient as a
        // long-term solution.
        box make_box( LOOP * ) const;

};
```

```
#define SURFACE_CLASS
#endif

/*********************************************************************************/

/*********************************************************************************/

// ($$) pcurve.h   1.0                    07/16/96

// Header for PCURVE.

// A PCURVE represents a parameter-space approximation to a curve
// lying on a parametrised surface. Because there is only one such
// representation, this class does not need to have derived classes
// for specific geometries.

// Note that we retain a use count in this record.

#if !defined( PCURVE_CLASS )

#include "kernel.h"
#include "top₩entity.h"
#include "top₩coedge.h"
#include "geom₩surface.h"
#include "curve₩pcurve₩pcrvdef.h"
#include "math₩vector.h"
#include "math₩transf.h"
#include "misc₩logical.h"

class CURVE;
class pcurve;
class transf;
class box;


// PCURVE declaration.

class PCURVE: public ENTITY {

        unsigned int use_count_data;

// Details of the PCURVE are found in the pcurve *pcurve_ptr OR
// via the CURVE *curve_ptr (see below), according to the value of
// int def_type.
// A def_type of zero indicates a private definition, supplied by
// pcurve_ptr(below).
// A positive value indicates the pcurve associated with
// the CURVE curve_ptr, shifted in parameter space by off.
// A negative value indicates the pcurve associated with
// the CURVE curve_ptr, negated, shifted in parameter space by off.
```

```
int def_type;      // values: 0, -1, 1, -2, 2, 3

// Definition of an explicit pcurve and defining surface .
par_pos parpos;              // ignored  if def_type isn't three (3)
pcurve *pcurve_ptr;          // ignored (null) if def_type is non-zero

// Defining surface of pcurve
SURFACE *def_surf_ptr;       // ignored (null) if def_type is non-zero

// Definition of an implicit pcurve.
CURVE *curve_ptr; // ignored (null) if def_type is zero.
par_vec off;                 // ignored (zero) if def_type is zero.

// Include the standard member functions for all entities.
ENTITY_FUNCTIONS( PCURVE )

// Now the functions specific to PCURVE.
// Make a bare PCURVE.
PCURVE();

// Make a PCURVE from a pcurve.
PCURVE( par_pos &, SURFACE * );
PCURVE( pcurve *, SURFACE * = NULL );

// Make a PCURVE to point to an existing curve.
PCURVE( CURVE *, int, par_vec const & = *(par_vec *)NULL );

// Use count manipulation, incrementing or decrementing.  If the
// decrement leaves the use count at zero, the record is deleted
// using lose().
void add();
void remove();

// Data reading routines.
int use_count() const { return use_count_data; }
int index() const { return def_type; }
pcurve *pcurve_def() const { return pcurve_ptr; }
par_pos pos() const { return parpos; }
SURFACE *surf_def() const { return def_surf_ptr; }
CURVE *ref_curve() const { return curve_ptr; }
par_vec offset() const { return off; }
box &make_bound();

// Data changing routines
void set_index( int );
void set_par_pos( par_pos & );
void set_surf_def( SURFACE * );
void set_ref_curve( CURVE * );

// Set the pcurve to explicit type.
```

```
// Removes any CURVE referred to by cur (which is set to NULL),
// sets def_type to zero, and puts given pcurve in pcurve_ptr.
void set_pcurve( pcurve *, SURFACE * );

// Set curve_ptr to pcurve of an existing CURVE.
// the logical is given as true if the reversed pcurve is wanted.
// Removes any previous reference in pcurve_ptr or curve_ptr,
// and increments use-count for given CURVE.
void set_pcurve( CURVE *, int, par_vec const & = *(par_vec *)NULL );

                // Negate the PCURVE, either by reversing the pcurve
                // or by reversing the value of a non_zero def_type.
                void negate();

                // Transform the PCURVE. If the definition is a CURVE reference,
                // assume that the curve will be transformed as well, so do nothing.
                void operator*=( transf const & );

                // Write the data of PCURVE to file
                void save_data( FILE * ) const;

                // Read the data of PCURVE from file
                void restore_data( FILE * );

};

#define PCURVE_CLASS
#endif

/*****************************************************************************/

/*****************************************************************************/

// ($$) curve.h             1.0      02/03/95

// Header file describing the modeller data structures

// The CURVE class provides the framework for the range of object-space
// curve types defined at any one time in the system.  As with surfaces,
// it provides a use count, to allow multiple use of curve descriptions.

#if !defined( CURVE_CLASS )

#include "kernel.h"
#include "top₩entity.h"
#include "curve₩crvdef.h"

class GPOINT;
class transf;
class box;
```

```
// CURVE declaration.

class CURVE: public ENTITY {

        // Use count for record, allowing multiple use.  Starts at zero, and
        // if ever it reaches zero again, the record can be deleted.
        // Holds the number of edges referencing the CURVE.
        unsigned int use_count_data;

        // Record geometric curve pointer
        curve *curve_ptr;

        // Include the standard member functions for all entities.
        ENTITY_FUNCTIONS( CURVE )

        // Now the functions specific to CURVE.
        // Make a generic curve and initializes the use count to zero.
        CURVE();

        // Create a CURVE by curve.
        CURVE( curve * );

        // Data accessing member functions
        // Return use count
        int use_count() const { return use_count_data; }

        // Return the curve pointer
        curve *geo_curve() const { return curve_ptr; }

        // Data changing member functions
        // Set geometric curve
        void set_curve( curve * );

        // Use count manipulation. Either add or subtract one use, and if
        // subtraction causes the use count to fall to zero, then delete
        // the record.
        void add();
        void remove();

        // Transform a curve.
        void operator*=( transf const & );

        // Determine a bounding box for the portion of the curve through two
        // points.
        box make_box( GPOINT *, GPOINT * ) const;

};

#define CURVE_CLASS
```

```
#endif
/*****************************************************************************/
/*****************************************************************************/
// ($$) gpoint.h   1.0       02/03/95
// Header for POINT.
// A point records the object space position of a vertex.  There is
// no need for derived types. (This record class could be combined
// with the VERTEX but is kept separate, however, for consistency.)
// Cartesian coordinates are assumed.

#if !defined( GPOINT_CLASS )

#include "kernel.h"
#include "topWentity.h"
#include "mathWvector.h"

class transf;
class position;

// Point definition proper.
class GPOINT: public ENTITY {

        // Use count for record, allowing multiple use.  Starts at zero, and
        // if ever it reaches zero again, the record can be deleted.
        // Holds the number of edges referencing the CURVE.
        unsigned int use_count_data;

        // Coordinates of point, recorded as a position.
        position coords_data;

        // Include the standard member functions for all entities.
        ENTITY_FUNCTIONS( GPOINT )

        // Now the functions specific to POINT.
        // Constructor for a null point, which has to be filled in later.
        GPOINT();

        // Public constructor for a point, given its cartesian coordinates
        GPOINT( double, double, double );

        // Constructor for a point, given a position.
        GPOINT( position const & );

        // Data reading routines.
```

```
int use_count() const { return use_count_data; }
position const &coords() const { return coords_data; }

// Use count manipulation. Either add or subtract one use, and if
// subtraction causes the use count to fall to zero, then delete
// the record.
void add();
void remove();

// Set position into existing point.
        void set_coords( position const & );

        // Write the data of gpoint to file
        void save_data( FILE * ) const ;

        // Read the data of gpoint from file
        void restore_data( FILE * );

        // Transform a point
        void operator*=( transf const & );

};

#define GPOINT_CLASS
#endif
```

```
/*****************************************************************************/
```

2. Euler operators

```
/*****************************************************************************/
logical MVS( LUMP *lump, GPOINT *gpt, SHELL *&shell, VERTEX *&vtx )
/*****************************************************************************/
// <Function>
// Create a single vertex shell on database

// <Input>
// lump: The pointer to lump
// pt:   The pointer to point

// <Output>
// shell:The pointer to created shell
// vtx:  The pointer to created vertex

// <Return value>
// TRUE: No error
// FALSE:Error
/*****************************************************************************/

/*****************************************************************************/
logical KVS( SHELL *shell )
/*****************************************************************************/
// <Function>
// Remove a single vertex shell on database

// <Input>
// shell:The pointer to vertex shell

// <Return value>
// TRUE: No error
// FALSE:Error
/*****************************************************************************/

/*****************************************************************************/
logical MEV( VERTEX *vtx, CURVE *crv, EDGE *&edge, VERTEX *&nvtx )
/*****************************************************************************/
// <Function>
// Create an edge and vertex of curve on database

// <Input>
// vtx:  The pointer to vertex
// crv:  The pointer to curve

// <Output>
// edge: The pointer to created edge
// nvtx: The pointer to created vertex
```

```
// <Return value>
// TRUE: No error
// FALSE:Error
/*****************************************************************************/

/*****************************************************************************/
logical KEV( EDGE *edge, VERTEX *vtx )
/*****************************************************************************/
// <Function>
// Remove an edge and vertex of curve on database

// <Input>
// edge: The pointer to edge
// vtx:  The pointer to vertex

// <Return value>
// TRUE: No error
// FALSE:Error
/*****************************************************************************/

/*****************************************************************************/
logical MEC(SHELL *shell, VERTEX *svtx, VERTEX *evtx, CURVE *crv, EDGE *&edge )
/*****************************************************************************/
// <Function>
// Create a cycle with an edge of curve on database

// <Input>
// shell:The pointer to shell of parent of edge
// svtx: The pointer to start vertex of new edge
// evtx: The pointer to end vertex of new edge
// crv:  The pointer to geometry of edge

// <Output>
// edge: The pointer to created edge

// <Return value>
// TRUE: No error
// FALSE:Error
/*****************************************************************************/

/*****************************************************************************/
logical KEC( EDGE *edge )
/*****************************************************************************/
// <Function>
// Remove an edge with a cycle on database

// <Input>
// edge: The pointer to deleted edge
```

```
// <Return value>
// TRUE: No error
// FALSE:Error
/*****************************************************************************/


/*****************************************************************************/
logical MFKC( SHELL *shell, PTRLIST<EDGE*> elist, SURFACE *surf, FACE *&face )
/*****************************************************************************/
// <Function>
// Remove a cycle and create a face of surface on database

// <Input>
// shell:The pointer to shell which has edges
// elist:The list of edge pointer which will be outer boundary of face
//       and has parametric curve as intcurve class
// surf: The pointer to geometry of surface

// <Output>
// face: The pointer to created face

// <Return value>
// TRUE: No error
// FALSE:Error
/*****************************************************************************/


/*****************************************************************************/
logical KFMC( FACE *face )
/*****************************************************************************/
// <Function>
// Create a cycle and remove a face on database

// <Input>
// face: The pointer to deleted face

// <Return value>
// TRUE: No error
// FALSE:Error
/*****************************************************************************/


/*****************************************************************************/
logical MFP( SHELL *shell, PTRLIST<EDGE*> elist, SURFACE *surf, LUMP *&lump,
             FACE *&face)
/*****************************************************************************/
// <Function>
// Create a lump with face of surface on database

// <Input>
// shell:The pointer to shell which has edges
// elist:The array of edge pointer which will be face
```

```
// surf: The pointer to geometry of surface

// <Output>
// lump: The pointer to created lump
// face: The pointer to created face

// <Return value>
// TRUE: No error
// FALSE:Error
/*****************************************************************************/

/*****************************************************************************/
logical KFP( LUMP *lump, FACE *face)
/*****************************************************************************/
// <Function>
// Remove a face whit lump on database

// <Input>
// lump: The pointer to deleted lump
// face: The pointer to deleted face

// <Return value>
// TRUE: No error
// FALSE:Error
/*****************************************************************************/

/*****************************************************************************/
logical MVL( FACE *face, GPOINT *gpt, LOOP *&loop, VERTEX *&vtx)
/*****************************************************************************/
// <Function>
// Create single vertex loop on database

// <Input>
// face: The pointer to face which has single vertex loop
// gpt:  The pointer to point

// <Output>
// loop: The pointer to created loop
// vtx:  The pointer to created vertex

// <Return value>
// TRUE: No error
// FALSE:Error
/*****************************************************************************/

/*****************************************************************************/
logical KVL( LOOP *loop )
/*****************************************************************************/
// <Function>
// Remove single vertex loop on database
```

```
// <Input>
// loop: The pointer to deleted loop

// <Return value>
// TRUE: No error
// FALSE:Error
/*****************************************************************************/

/*****************************************************************************/
logical SEMV( EDGE *edge, CURVE *crv1, CURVE *crv2, VERTEX *&nvtx, EDGE *&nedge)
/*****************************************************************************/
// <Function>
// Split an edge and create new edge on database

// <Input>
// edge: The pointer to edge which will be splitted
// crv1, crv2:     The geometry of splitted edge

// <Output>
// nvtx: The pointer to created vertex
// nedge:The pointer to created edge

// <Return value>
// TRUE: No error
// FALSE:Error
/*****************************************************************************/

/*****************************************************************************/
logical JEKV( EDGE *edge, VERTEX *vtx, CURVE *crv )
/*****************************************************************************/
// <Function>
// Joint an edge and remove vertex on database

// <Input>
// edge: The pointer to edge which will be jointted
// vtx:  The pointer to vertex which is removed
// crv:  The pointer to curve which will be geometry of edge

// <Return value>
// TRUE: No error
// FALSE:Error
/*****************************************************************************/

/*****************************************************************************/
logical MEF( LOOP *loop, VERTEX *svtx, VERTEX *evtx, CURVE *crv, EDGE *&nedge,
             FACE *&nface )
/*****************************************************************************/
// <Function>
// Split face with creation of new edge on database
```

```
// <Input>
// loop: The pointer to loop of face which don't have inner loop
// svtx: The pointer to start vertex of newly created edge
// evtx: The pointer to end vertex of newly created edge
// crv:  The pointer to geometry of newly created edge

// <Output>
// nedge:The pointer to newly created edge
// nface:The pointer to newly created face

// <Return value>
// TRUE: No error
// FALSE:Error
/*****************************************************************************/

/*****************************************************************************/
logical KEF( EDGE *edge, FACE *face )
/*****************************************************************************/
// <Function>
//  Merge two face with respect to an edge on database

// <Input>
// edge: The pointer to deleted edge
// face: The pointer to deleted face

// <Return value>
// TRUE: No error
// FALSE:Error
/*****************************************************************************/

/*****************************************************************************/
logical KEML( EDGE *edge, LOOP *&loop )
/*****************************************************************************/
// <Function>
// Split a loop with deleting an edge on database

// <Input>
// edge: The pointer to edge of a loop which will be splitted

// <Output>
// loop: The pointer to newly created loop

// <Return value>
// TRUE: No error
// FALSE:Error
/*****************************************************************************/

/*****************************************************************************/
logical MEKL( LOOP *lp1, LOOP *lp2, VERTEX *svtx, VERTEX *evtx, CURVE *crv,
```

```
                EDGE *&edge )
/*****************************************************************************/
// <Function>
// Join two loops with creating an edge on database

// <Input>
// lp1:  The pointer to outer loop which will be joined
// lp2:  The pointer to inner loop which will be joined
// svtx: The pointer to a vertex of loop 1
// evtx: The pointer to a vertex of loop 2
// crv:  The pointer to geometry of newly created edge

// <Output>
// edge: The pointer to newly created edge

// <Return value>
// TRUE: No error
// FALSE:Error
/*****************************************************************************/

/*****************************************************************************/
logical MEKS( SHELL *shell1, SHELL *shell2, VERTEX *svtx, VERTEX *evtx,
              CURVE *crv, EDGE *&edge )
/*****************************************************************************/
// <Function>
// Join two shells with creating an edge on database

// <Input>
// shell1: The pointer to shell which will be joined
// shell2: The pointer to shell which will be joined and then removed
// svtx: The pointer to a vertex of shell1
// evtx: The pointer to a vertex of shell2
// crv:  The pointer to geometry of newly created edge

// <Output>
// edge: The pointer to newly created edge

// <Return value>
// TRUE: No error
// FALSE:Error
/*****************************************************************************/

/*****************************************************************************/
logical KEMS( EDGE *edge, SHELL *&shell )
/*****************************************************************************/
// <Function>
// Split shell with deleting an edge of shell on database

// <Input>
// edge: The pointer to edge which will be deleted
```

```
// <Output>
// shell:The pointer to newly created shell

// <Return value>
// TRUE: No error
// FALSE:Error
/*****************************************************************************/

/*****************************************************************************/
logical MBP( BODY *&body, LUMP *&lump )
/*****************************************************************************/
// <Function>
// Create a body with a lump on database

// <Input>
// None

// <Output>
// body: The pointer to newly created body
// lump: The pointer to newly created lump

// <Return value>
// TRUE: No error
// FALSE:Error
/*****************************************************************************/

/*****************************************************************************/
logical KBP( BODY *body )
/*****************************************************************************/
// <Function>
// Delete a body on database

// <Input>
// body: The pointer to deleted body

// <Return value>
// TRUE: No error
// FALSE:Error
/*****************************************************************************/
```

3. 데이터 구조 관리를 위한 주요 APIs

```
/*****************************************************************************/
logical InqLumpsOfBody( BODY *body, PTRLIST<LUMP*> &LumpList )
/*****************************************************************************/
// <Function>
// Get all lumps of body from DB

// <Input>
// body: A pointer to body

// <Output>
// LumpList: List of lumps

// <Return value>
// TRUE: No error
// FALSE:Error
/*****************************************************************************/

/*****************************************************************************/
logical InqVerticesOfBody( BODY *body, PTRLIST<VERTEX*> &VtxList )
/*****************************************************************************/
// <Function>
// Get all vertices of body from DB

// <Input>
// body: A pointer to body

// <Output>
// VtxList: List of vertices

// <Return value>
// TRUE: No error
// FALSE:Error
/*****************************************************************************/

/*****************************************************************************/
logical InqShellVerticesOfBody( BODY *body, PTRLIST<VERTEX*> &VtxList )
/*****************************************************************************/
// <Function>
// Get all sigle vertex shell of body from DB

// <Input>
// body: A pointer to body

// <Output>
// VtxList: List of single vertex shells
```

```
// <Return value>
// TRUE: No error
// FALSE:Error
/*****************************************************************************/

/*****************************************************************************/
logical InqWiresOfBody( BODY *body, PTRLIST<EDGE*> &WireList )
/*****************************************************************************/
// <Function>
// Get all wireframe edges of body from DB

// <Input>
// body: A pointer to body

// <Output>
// WireList: List of wireframe edges

// <Return value>
// TRUE: No error
// FALSE:Error
/*****************************************************************************/

/*****************************************************************************/
logical InqSurfacesOfBody( BODY *body, PTRLIST<SURFACE*> &SurfList)
/*****************************************************************************/
// <Fuction>
// Get geometry of faces in body

// <Input>
// body: A pointer to body

// <Output>
// SurfList:      List of surface

// <Return value>
// TRUE: No error
// FALSE:Error
/*****************************************************************************/

/*****************************************************************************/
logical InqShellsOfLump( LUMP *lump, PTRLIST<SHELL*>& ShellList )
/*****************************************************************************/
// <Function>
// Get all shells of a lump

// <Input>
// lump: A pointer to lump

// <Output>
// ShellList:     List of shell
```

```
// <Return value>
// TRUE: No error
// FALSE:Error
/*****************************************************************************/

/*****************************************************************************/
logical InqDirOfLoop( LOOP *loop, LOOPDIR &dir )
/*****************************************************************************/
// <Function>
// Inquire direction of loop

// <Input>
// loop: The pointer to loop

// <Output>
// dir:  The direction of loop about surface normal - CCW, CW, OPL.

// <Return value>
// TRUE: No error
// FALSE:Error
/*****************************************************************************/

/*****************************************************************************/
logical InqCoedgesOfEdge( EDGE *edge, PTRLIST<COEDGE*> &coedgelist )
/*****************************************************************************/
// <Function>
// Inquire all coedges of a given edge on database

// <Input>
// edge: The pointer to edge

// <Output>
// coedgelist:     The reference of coedge list

// <Return value>
// TRUE: No error
// FALSE:Error
/*****************************************************************************/

/*****************************************************************************/
logical InqEdgesOfVertex( VERTEX *vtx, PTRLIST<EDGE*> &edgelist )
/*****************************************************************************/
// <Function>
// Inquire all edges which have a given vertex as vertex on database

// <Input>
// vtx:  The pointer to vertex

// <Output>
```

```
// edgelist: The reference of edges list

// <Return value>
// TRUE: No error
// FALSE:Error
/*****************************************************************************/


/*****************************************************************************/
logical SApi_InsFaceToDB(surface *gsurf, FACE *&nface )
/*****************************************************************************/
// <Function>
// Insert surface to data structure as a face

// <Input>
// gsurf:A pointer to surface

// <Output>
// nface:A pointer to newly created face

// <Return value>
// TRUE: No error
// FALSE:Error
/*****************************************************************************/

/*****************************************************************************/
logical SApi_DelFaceOnDB( FACE *face)
/*****************************************************************************/
// <Function>
// Delete face on DB

// <Input>
// edge: A pointer to face which is deleted

// <Return value>
// TRUE: No error
// FALSE:Error
/*****************************************************************************/

/*****************************************************************************/
logical SApi_InsFaceEdgeToDB( FACE *face, curve *gcrv, EDGE *&nedge )
/*****************************************************************************/
// <Function>
// Insert curve to DB as an face edge

// <Input>
// face: A face of supporting surface
// gcrv: Geometry of edge which is inserted

// <Output>
```

```
// nedge:Created edge

// <Return value>
// TRUE: No error
// FALSE:Error
/*****************************************************************************/

/*****************************************************************************/
logical SApi_InsWireEdgeToDB( curve *gcrv, EDGE *&nedge )
/*****************************************************************************/
// <Function>
// Insert curve to DB as a wire edge

// <Input>
// gcrv: Geometry of edge which is inserted

// <Output>
// nedge:Created edge

// <Return value>
// TRUE: No error
// FALSE:Error
/*****************************************************************************/

/*****************************************************************************/
logical SApi_DelEdgeOnDB( EDGE* edge )
/*****************************************************************************/
// <Function>
// Delete edge on DB

// <Input>
// edge: A pointer to edge which is deleted

// <Return value>
// TRUE: No error
// FALSE:Error
/*****************************************************************************/

/*****************************************************************************/
logical SApi_InsVertexToDB( LUMP *lump, GPOINT *pt, SHELL *&shell, VERTEX
                            *&nvtx)
/*****************************************************************************/
// <Function>
// Insert shell vertex of gpoint to a given lump

// <Input>
// lump: Lump of vertex
// pt:   Geometry of vertex

// <Output>
```

```
// shell:Parent of vertex
// vtx:  Created vertex

// <Return value>
// TRUE: No error
// FALSE:Error
/*****************************************************************************/

/*****************************************************************************/
logical SApi_InsPgroupToDB( ISOPT *iso, BODY *body, PGROUP *&pg )
/*****************************************************************************/
// <Function>
// Insert pgroups to DB

// <Input>
// iso:  Instance of start ISOPT of ISOPT group
// body: Pointer to body

// <Output>
// pg:              Created pgroup

// <Return value>
// TRUE: No error
// FALSE:Error
/*****************************************************************************/

/*****************************************************************************/
logical SApi_WriteBodyToFile( BODY *body, const char *filename )
/*****************************************************************************/
// <Function>
// Write all data of body to file

// <Input>
// body:     The pointer to body
// filename:  File name

// <Return value>
// TRUE: No error
// FALSE:Error
/*****************************************************************************/

/*****************************************************************************/
logical SApi_WritePointsToFile( BODY *body, const char *filename )
/*****************************************************************************/
// <Function>
// Write only points data of body to file

// <Input>
// body:     The pointer to body
```

```
// filename: File name

// <Return value>
// TRUE: No error
// FALSE:Error
/*****************************************************************************/

/*****************************************************************************/
logical SApi_ReadBodyFromFile( const char *filename, BODY *& nbody )
/*****************************************************************************/
// <Function>
// Read all data of body from file

// <Input>
// filename:       File name

// <Output>
// nbody:          The pointer to newly created body

// <Return value>
// TRUE: No error
// FALSE:Error
/*****************************************************************************/

/*****************************************************************************/
logical SApi_CrePgroupsOfPointsBySAPFile(const char *filename, BODY *body)
/*****************************************************************************/
// <Function>
// Create Pgroups from point file (*.SAP)

// <Input>
// filename: Input point file name
// body:     The pointer to body which points are added to

// <Return value>
// TRUE: No error
// FALSE:Error
/*****************************************************************************/
```

D. 측정 데이터의 Fairing에 관련된 API

1. Line 생성

```
/*************************************************************************/
logical FairingStraight3DLine(long NoPts, position *pts, position *&pt1,
position *&pt2)
/*************************************************************************/
// <Function>
  입력된 point 들을 이용해 계산 하여 직선을 구현 하는 데 필요한 두 point
  를 구현 하는데 이상 없이 return value 을 제공 하면 TRUE 임.

// <input parameter>
  NoPts:  point 의 수를 나타냄
  pts: point 의 좌표를 가진 pointer

// <Output parameter>
  pt1: 직선을 구현하는 좌표를 가진 pointer
  pt2: 직선을 구현하는 좌표를 가진 pointer

// <Return value>
  TRUE:  No error
  FALSE: Error
/*************************************************************************/
```

2. Plane 생성

```
/*************************************************************************/
logical FairingPlane3Dplane(long NoPts, position *pts, position *&pt1,
position *&pt2, position &pt3)
/*************************************************************************/
// <Function>
  입력된 point 들을 이용해 계산 하여 평면을 구현 하는데  필요한 세 point
  가  이상 없이 return value 를 제공 하면 TRUE 임.

// <input parameter>
  NoPts:  point 의 수를 나타냄
  pts: point 의 좌표를 가진 pointer

// <Output parameter>
  pt1: 평면을 구현하는 좌표를 가진 pointer
  pt2: 평면을 구현하는 좌표를 가진 pointer
```

```
  pt3: 평면을 구현하는 좌표를 가진 pointer

// <Return value>
  TRUE:  No error
  FALSE: Error
/*****************************************************************************/
```

3. Circle 생성

```
/*****************************************************************************/
logical FairingCircleXYplane(long NoPts, position *pts, position *&cen,
long rad)
logical FairingCircleYZplane(long NoPts, position *pts, position *&cen,
long rad)
logical FairingCircleZXplane(long NoPts, position *pts, position *&cen,
long rad)
/*****************************************************************************/
// <Function>
  입력된 point 들을 가지고 계산 하여 각 XY, YZ, ZX 평면에서 원을 구현 하
  는 중심점과 반지름이 이상 없이 return value를 제공 하면 TRUE 임.

// <input parameter>
  NoPts:  point 의 수를 나타냄
  pts: point 의 좌표를 가진 pointer

// <Output parameter>
  cen: 원을 구현하는 좌표를 가진 pointer
  rad: 원을 구현하는 반지름.

// <Return value>
  TRUE:  No error
  FALSE: Error
/*****************************************************************************/
```

4. Arc 생성

```
/*****************************************************************************/
logical FairingArcXYplane(long NoPts, position *pts, position *&cen,
position*&stpt, position, *&end)
logical FairingArcYZplane(long NoPts, position *pts, position *&cen,
position *&stpt, position *&end)
logical FairingArcZXplane(long NoPts, position *pts, position *&cen,
position *&stpt, position *&end)
```

```
/*****************************************************************/
// <Function>
  입력된 point 들을 가지고 계산 하여 각 XY, YZ, ZX 평면에서 호를 구현 하
  는 중심점 과 양끝 점이 이상 없이 return value 제공 하면 TRUE 임.

// <input parameter>
  NoPts:  point 의 수를 나타냄
  pts: point 의 좌표를 가진 pointer

// <Output parameter>
  cen: 호를 구현하는 좌표를 가진 pointer
  stpt: 호를 구현하는 좌표를 가진 pointer

// <Return value>
  TRUE:  No error
  FALSE: Error
/*****************************************************************/
```

5. Data filtering

```
/*****************************************************************/
logical DataFiltering(long NoPts, position *pts, long &NoFiPts, position
*&FiPts)
/*****************************************************************/
// <Function>
  입력된 point 들을 가지고 계산 하여 data filtering 을 이상이 없이 return
  value 를 제공 하면 TRUE 임.

// <input parameter>
  NoPts:  point 의 수를 나타냄
  pts: point 의 좌표를 가진 pointer

// <Output parameter>
  NoFiPts: data filtering 을  구현하는 point 의 군들의 point 수.
  FiPts: data filtering 을  구현하는 point 의 군들의 point 의 좌표를 가진
  pointer

// <Return value>
  TRUE:  No error
  FALSE: Error
/*****************************************************************/
```

6. Data reduction

```
/*****************************************************************************/
logical DataReduction(long NoPts, position *pts, long &ReNoFiPts,
position *&RePts, double ch_dev)
/*****************************************************************************/
// <Function>
  입력된 point 들을 이용해 계산 하여 data reduction 을  이상 없이 return
  value 를 제공하면 TRUE 임.

// <input parameter>
  NoPts:  point 의 수를 나타냄
  pts: point 의 좌표를 가진 pointer

// <Output parameter>
  ReNoFiPts: data reduction 을  구현하는 point 의 군들의 point 수.
  ReFiPts: data reduction 을 구현하는 point 의 군들의 point 의 좌표를 가진
  pointer

// <Return value>
  TRUE:  No error
  FALSE: Error
/*****************************************************************************/
```

E. 응용 곡선,곡면의 생성과 관련된 API

```
/*****************************************************************************/
logical CurveProjection(  nurbcrv *crv,  nurbsuf *surf,  vector *prj_dir,
                          nurbcrv *&nucrv, pcurve *&pcrv,  int &nocrv)
/*****************************************************************************/
//  < Function>
  공간 곡선 crv를 surf 곡면 상으로 prj_dir 방향으로 투영한다. 투영 후 생
  성되는 곡선은 nocrv개이며 nucrv, pcrv는 각각 nurbcrv와 pcurve의 배열로
  저장된다.  nocrv ≥ 1 이면 TRUE,  nocrv = 0 이면 FALSE가 리턴된다.
//  <Input parameter>
   crv:    곡면에 투영할 공간상의 곡선
   surf:   곡선이 투영되는 곡면
   prj_dir: 곡선의 투영 방향

//  <Output parameter>
   nucrv:  곡면에 투영시켜 생성된 공간상의 곡선
   pcrv:   nucrv의 parametric domain 상의 곡선
   nocrv:  nucrv의 개수

//  <Return value>
   TRUE:   No error
   FALSE:  Error
/*****************************************************************************/
```

```
/*****************************************************************/
logical  SurfaceIntersection( SURFACE *surf1, SURFACE *surf2,
         nurbcrv *&nucrvs, pcurve *&pcrvs1, pcurve *&pcrvs2,  int &nicv)
/*****************************************************************/
//  < Function>
   두 곡면이 교차되었을 때 생성되는 교차 곡선을 구한다. 생성된 교차 곡선의
   개수는 nicv개이며 nucrvs, pcrv1, pcrv2는 각각 nicv개의 배열로 저장된다.
   nicrv ≥ 1 이면 TRUE,  nicrv = 0 이면 FALSE가 리턴된다.
//  <Input parameter>
   surf1:  교차곡선을 구할 base surface
   surf2:  교차곡선을 구할 base surface
//  <Output parameter>
   nucrvs:  surf1과 surf2간의 교차 곡선
   pcrvs1:  surf1의 parametric domain 상의 nucrvs.
   pcrvs2:  surf2의 parametric domain 상의 nucrvs.
   nicv:    nucrvs의 개수
//  <Return value>
   TRUE:   No error
   FALSE:  Error
/*****************************************************************/
```

```
/*****************************************************************/
logical FaceFaceIntersection(FACE *face1, FACE *face2,
                int &nicv, nurbcrv *&nucrv,pcurve *&pcrv1,pcurve *&pcrv2)
/*****************************************************************/
//  < Function>
   두 face가 교차되었을 때 생성되는 교차 곡선을 구한다. 생성된 교차 곡선의
   개수는 nicv개이며 nucrvs, pcrv1, pcrv2는 각각 nicv개의 배열로 저장된다.
   nicv ≥ 1 이면 TRUE,  nicv = 0 이면 FALSE가 리턴된다.
//  <Input parameter>
    face1: 교차곡선을 구할 face
    face2: 교차곡선을 구할 face
//  <Output parameter>
    nicv: nucrv의 개수
    nucrv:  face1과 face2간의 교차 곡선
    pcrv1: face1의 parametric domain 상의 nucrv
    pcrv2: face2의 parametric domain 상의 nucrv
//  <Return value>
    TRUE:   No error
    FALSE:  Error
/*****************************************************************/
```

```
/*****************************************************************************/
logical SweepRotational( int noSc, nurbcrv *sc_nucrvs,
                         int noBc, nurbcrv *bc_nucrvs,
                    vector *rotAxis, position *RotCen, nurbsuf *&nusurf)
/*****************************************************************************/
//  < Function>
  축벡터 rotAxis를 중심으로 회전시켜 곡면의 생성시킨다.
//  <Input parameter>
   noSc:  단면곡선의 개수
   sc_nucrvs: 단면곡선
   noBc: 경계곡선의 개수
   bc_nucrvs: 경계곡선
   rotAxis: 회전축 벡터
   RotCen: 회전축 벡터의 중심
//  <Output parameter>
   nusurf: rotaional sweep 혹은 revolution으로 생성된 곡면
//  <Return value>
   TRUE:   No error
   FALSE:  Error
/*****************************************************************************/
```

```
/*****************************************************************/
logical SweepSyncronized( int noSc, nurbcrv *sc_nucrvs,
                          int noBc, nurbcrv *bc_nucrvs,
                     vector *gcPlnNorm,     nurbsuf *&nusurf)
/*****************************************************************/
//  < Function>
   단면곡선을 경계곡선의 같은 파라메터에 해당하는 위치로 이동,변환하여
sweep 곡면을 얻는다.
//  <Input parameter>
   noSc:  단면곡선의 개수
   sc_nucrvs: 단면곡선
   noBc: 경계곡선의 개수
   bc_nucrvs: 경계곡선
   gcPlnNorm: 안내평면의 법선벡터
   RotCen: 회전축 벡터의 중심
//  <Output parameter>
   nusurf:  syncronized sweep으로 생성된 곡면
//  <Return value>
   TRUE:   No error
   FALSE:  Error
/*****************************************************************/
```

```
/*****************************************************************/
logical SweepNormal( int noSc, nurbcrv *sc_nucrvs,
                     int noBc, nurbcrv *bc_nucrvs,
                     nurbcrv *gc_nucrv, vector *scNorm,
                     vector *gcPlnNorm,      nurbsuf *&nusurf)
/*****************************************************************/
//  < Function>
   단면곡선을 경계곡선에 수직하게 이동시켜 곡면을 얻는다.
//  <Input parameter>
   noSc:  단면곡선의 개수
   sc_nucrvs: 단면곡선
   noBc: 경계곡선의 개수
   bc_nucrvs: 경계곡선
   gc_nucrv: 안내곡선
   scNorm: 단면곡선이 놓인 단면 평면의 법선벡터
   gcPlnNorm : 안내평면의 법선벡터
//  <Output parameter>
   nusurf: normal sweep으로 생성된 곡면
//  <Return value>
   TRUE:   No error
   FALSE:  Error
/*****************************************************************/
```

```
/*****************************************************************/
logical SweepParallel(int noSc, nurbcrv *sc_nucrvs,
                      int noBc, nurbcrv *bc_nucrvs,
                      vector *scNorm,       nurbsuf *&nusurf)
/*****************************************************************/
//  < Function>
   단면곡선을 경계곡선을 따라 평행하게 이동시켜 곡면을 얻는다.
//  <Input parameter>
   noSc:  단면곡선의 개수
   sc_nucrvs: 단면곡선
   noBc: 경계곡선의 개수
   bc_nucrvs: 경계곡선
   scNorm: 단면곡선이 놓인 단변평면의 법선벡터
//  <Output parameter>
   nusurf: parallel sweep으로 생성된 곡면
//  <Return value>
   TRUE:   No error
   FALSE:  Error
/*****************************************************************/
```

F. IGES, ZES 인터페이스 관련 API

```
/*****************************************************************************/
logical ReadFACEfromIGESfile(const char *fname,
                             int &no_face, FACE *& firstFace )
/*****************************************************************************/
//  < Function>
   IGES 파일을 읽어 SurfART 모델로 변경한다.

//  <Input parameter>
   fname : import할 IGES 파일명

//  <Output parameter>
    no_face : 모델의 face 개수
    firstFace : SurfART 데이터 스트럭쳐로 변환된 face의 첫번째 포인터

//  <Return value>
   TRUE:   No error
   FALSE:  Error
/*****************************************************************************/
```

```
/*****************************************************************************/
logical ReadFaceFromTESfile(const char *fname,
                            int * no_face, FACE *&firstFace)

/*****************************************************************************/
//  < Function>
    ZES 파일을 읽어 SurfART 모델로 변경한다.

//  <Input parameter>
    fname : import할 ZES 파일명

//  <Output parameter>
    no_face : 모델의 face 개수
    firstFace : SurfART 데이터 스트럭쳐로 변환된 face의 첫번째 포인터

//  <Return value>
   TRUE:   No error
   FALSE:  Error
/*****************************************************************************/
```

G. NC에 관련된 중요한 함수

```
//*************************************************************************
logical CarCut(PTRLIST<FACE*>* FcList, vector Pdir1,char RorL1,
double toler,double paslen)
//*************************************************************************
// <Function>
Cartesian Cutting을 수행함.
// <Input parameter>
        FcList:선택된 face들의 pointer
        Pdir1: 가공 평면의 법선 벡터 방향
        RorL1: 가공 방향
        toler: 가공 여유
        paslen: 가공 경로의 폭
// <Output parameter>
        FILE: *.CC의 파일
              *.IDX의 파일
// <Return value>
        TRUE: No error
        FALSE:Error

//*************************************************************************
logical ParaCut(PTRLIST<FACE*>* FcList, double toler, double paslen)
//*************************************************************************
// <Function>
generate CCdata along Iso-Parameric Curve
// <Input parameter>
```

FcList:선택된 face 들의 pointer

toler: 가공 여유

paslen: 가공 경로의 폭

// <Output parameter>

FILE: *.CC 의 파일

*.IDX 의 파일

// <Return value>

TRUE: No error

FALSE:Error

//**********

void INTF()

//**********

// <Function>

간섭 검사

CL data 계산

// <Input parameter>

FILE: *.CC 의 파일

FILE: *.IDX - *.CC 에 대한 자료

// <Output parameter>

FILE: *.CL 의 파일

// <Return value>

없음

//**********

void PostProc (int ModeCut,char *FileNm)

//**********

// <Function>

NC code 생성

// <Input parameter>

ModeCut: 가공 방식

FileNm : *.CL file 의 이름

FILE: *.CFG - NC 의 종류에 관한 자료

// <Output parameter>

FILE: *.NC 의 파일

// <Return value>

없음

참 고 문 헌

[2-1] Programming with the Microsoft Foundation Class Library, Microsoft Press, 1994.

[2-2] Micky Williams, Understanding Visual C++ 4, 도서출판 삼각형, 1996.

[2-3] Jackie Neider, Tom Davis and Mason Woo, OpenGL Programming Guide, Addison-Wesley Publishing company, 1993.

[2-4] 이상엽, Visual C++ Programming Bible Ver 4.x, 영진출판사, 1997.

[3-1] Michael E. Mortension, Geometric Modeling, John Wiley & Sons, Inc., 1985.

[3-2] 최병규, Surface Modeling for CAD/CAM, 1990.

[3-3] Bruce R. Dewey, Computer Graphics for Engineers, Harper & Row, Publishers, 1988.

[3-4] OpenGL Reference Manual, Addison-Wesley Publishing Company, 1992.

[4-1] 山口富士夫, 形狀處理工學(III), 月刊工業新聞社, 1987.

[4-2] P. Lienhardt, "Topological Models for Boundary Representation: a comparison with n-dimensional generalized maps", "Computer-Aided Design, vol. 23, pp. 59 - 82, 1991.

[4-3] 이상헌, "사출 성형 제품의 설계 및 해석의 통합 환경을 제공하기 위한 특징형상 기반 비다양체 모델링 시스템의 개발", 박사학위 논문, 서울대학교, 1993.

[4-4] 이상헌, 이건우, "비다양체 형상 모델링을 위한 간결한 경계 표현 및 확장된 오일러 작업자", 한국 CAD/CAM 학회 논문집, 제1권, 제1호, pp. 1 - 19, 1996.

[4-5] 이건우, "Representation of Solid(Topology)", 산학협동 공개강좌 교재, 한국과학기술원, pp. 1 - 49, 1990.

[4-6] Mantyla, M., An introduction to SOLID MODELING, Computer Science Press, 1988.

[4-7] Weiler, K., 'The Radial Edge Structure: A Topological Represen-tation for Non-Manifold Geometric Boundary Modeling", Geometric Modeling for CAD Applications, North-Holland, pp. 3 - 36, 1988.

[4-8] Choi, Y., "Vertex-based Boundary Representation of Non-Manifold Geometric Models", Ph.D Thesis, Canegie Mellon Univ., 1989.

[4-9] Spatial Technology Inc., ACIS Geometric Modeler: Programmers Manual, 1993.

[5-1] "Surface User's Guide", Imageware Inc., 1995.

[5-2] Shan S. Kuo, "Computer Application of Numerical Methods", Addison-Wesley Publishing Co. 1972.

[5-3] Chapra Canle, "Numerical Method for Engineerings", McGraw-Hill book Co. 1984.

[5-4] William H. Press, "Numerical Recipes in C", Cambridge Univ. Press, 1992.

[5-5] Michael E. Mortenson, "Computer Graphic Handbook", Industrial Press Inc. 1990.

[5-6] 이재규, C로 배우는 알고리즘, 세화 출판사. 1996.

[6-1] Taylor,H.F., Flemings,M.C. and Wulff,J., 1959, Foundary Engineering, John Wiley & Sons,Inc.

[6-2] Fanuc Ltd., 1982, FAPT DIE-II for Die Sculptured Surface Modelling.

[6-3] Paul,R.P, 1982, Robot Manipulators - Mathematics, Programming and Control, the MIT press.

[6-4] 전차수, 1985, 3차원 측정 데이타로부터 자유곡면 NC가공, 석사학위논문, 한국과학기술원.

[6-5] Farouki,R.T., 1987, "Trimmed-surface algorithm for the evaluation and interrogation of solid boundry representations", IBM J. RES. DEVELOP, Vol31, No3.

[6-6] 주상윤, 1989, "곡면 모델링에서 블렌드 곡면형성에 관한 연구", KAIST 박사 학위 논문.

[6-7] Choi, B.K. and Lee, C.S., 1990, "Sweep Surface Modelling via Coordinate Transformation and Blending", Computer-Aided Design,20(6),pp.371 -378.

[6-8] Choi,B.K, 1991, Surface Modelling for CAD/CAM, Elsevier.

[6-9] 큐빅테크, 1992, 오메가 시스템 사용법 설명서, 큐빅테크.

[6-10] 큐빅테크, 1992, SWEEP III 사용설명서편, 큐빅테크.

[6-11] 한국과학기술연구원, 1992, 프레스 금형의 CAD/CAE 시스템 개발,연구보고서,과학기술처.

[6-12] 전용태, 이숙진, 최재붕, 박세형, 1993, "곡면모델러에서 트리밍곡면생성", 대한기계학회논문집, 17권 6호.

[6-13] 전차수, 주상윤, 전명길, 1994, "허미트 보간을 이용한 곡률연속 현길이 스플라인 곡면", 대한산업공학회지, 20권 1호.

[6-14] Piegl,L., Tiller,W., 1995, The NURBS Book, Springer.

[6-15] 구미정, 1997. 2, "곡면 모델링 커널 개발", 경상대 석사 학위 논문.

[7-1] IGES manual(ver 5.0)

[7-2] ZES manual(ver 2.2)

[8-1] Faux, I.D. and Pratt, M.J., 1981, Computational Geometry for Design and Manufacture, Ellis Horwood

[8-2] 최병규, 김대현, 1985, 자유 곡면 절삭을 위한 경제적인 CL 데이터의 계산, 대한산업공학회지, Vol. 9, No. 2

[8-3] 전차수, 1985, 3차원 측정데이터로부터 자유곡면의 NC가공, KAIST 산업공학과 석사학위논문

[8-4] Choi, B.K. et al, 1988, Compound Surface Modeling and Machining, CAD, Vol. 20, No. 3, pp. 127~136

[8-5] Choi, B.K. and Jun, C.S., 1989, Ball-end Cutter Interference Avoidance in NC Machining of Sculptured Surfaces, CAD, Vol. 21, No. 6, pp. 371~378