GOVP1199904201

65

감성측정평가 시뮬레이터 개발

Development of the Human Sensibility Evaluation Simulator

3차원 시청각 환경 제시기술 개발

Development of 3D Visual/Audio Display System

연구기관
한국과학기술연구원

과 학 기 술 부

제 출 문

과학기술부 장관 귀하

본 보고서를 “감성측정평가 시뮬레이터 개발” 과제 (세부과제 “3차원 시청각 환경 제시 기술 개발”)의 보고서로 제출합니다.

1998.12.

연구 기관명 한국과학기술연구원
연구 책임자 : 고희동 (영상미디어연구센터 책임연구원)
연 구 원 김현석, 김래현, 안재홍, 박경동, 진종욱,
조현제, 장상철, 박창훈, 서형준, 박문호
(영상미디어연구센터 연 구 원)

위탁연구기관:

최윤철 (연세대학교 교 수)
유석종, 고명철, 장택수, 우상옥
(연세대학교 석 박 사 과 정)

정만호 (청주대학교 교 수)
유호식, 김종재, 허태연
(청주대학교 석 박 사 과 정)

김남균 (전북대학교 교 수)
김동욱, 한후석, 오승곤, 박상수
(전북대학교 석 박 사 과 정)

김정현 (포항공과대학 교 수)
이지영, 김혜정, 서진석
(포항공과대학 석 박 사 과 정)

이남식 (국제산업디자인대학원 교 수)
신기진, 신찬수
(국제산업디자인대학원 석사과정)

여 백

요 약 문

Ⅰ 제 목

3차원 시청각 환경 제시 기술 개발

Ⅱ. 연구개발의 목적 및 필요성

본 연구의 목적은 감성측정평가 시뮬레이터가 필요로 하는 여러 가지 형태의 모의 환경 제시 기술중 3차원 시각 및 청각 환경 제시와 관련있는 제반 기술들을 연구하며, 궁극적으로는 다양한 제품이나 환경에 대한 감성측정 평가시에 필요로 하는 물리적인 또는 가상적인 3차원 시청각 환경들을 신속, 편리하게 저작할 수 있고, 또한 원활한 감성측정평가를 지원하기 위해 저작된 3차원 시청각 환경들을 실시간으로 적절히 조정, 통제하며 피실험자에게 제시할 수 있는 3차원 시청각 환경 제시 시스템을 개발하는 것이다.

감성측정평가 시뮬레이터를 개발하는데 있어서 다양한 감성측정평가 대상 제품이나 환경에 대해 필요한 수준의 현실감을 제공하면서 제작이나 실험자에게 제시하는 방법이 매우 효율적인 시스템 요구된다. 이에 대한 해결방안으로 컴퓨터 기술과 가상현실 기술을 기반으로 하여 감성측정평가 대상 제품이나 환경에 대한 인공현실감을 구현해 주는 3차원 시청각 환경 제시 시스템의 개발이 필요하다.

한편, 3차원 시청각 환경 제시 시스템은 다중매체 기술, 실시간 시뮬레이션 기술, 가상현실 기술 등을 기반으로 하기 때문에 이러한 기술들을 기반으로 하는 과학, 설계, 엔지니어링, 의료, 교육/오락, 훈련, 방송 등과 같은 타 산업분야에 대한 파급효과도 매우 클 것으로 사료된다.

Ⅲ. 연구개발의 내용 및 범위

3차원 시청각 환경 제시 시스템은 크게 시청각 환경 저작 시스템과 시청각 환경 제시 및 통제 시스템으로 구성된다. 즉, 다양한 제품이나 환경에 대한 감성측정 평가에 필요한 물리적 또는 가상적인 3차원 시각 및 청각 환경을 신속하고 편리하게 저작할 수

있는 시스템과 원활한 감성측정 평가를 지원하기 위하여 시청각 환경 저작 시스템을 통해 저작된 3차원 시청각 환경을 실시간 대화식으로 적절히 조정 및 통제하며, 피실험자에게 제시할 수 있는 시스템이 그것이다.

3차원 시청각 환경 저작 시스템 개발에는 먼저 지형/도로, 장애물, 건물, 사람, 자동차 등과 같은 각종 시각적인 객체들을 작성하고, 편집하는 기능을 수행하는 시각객체 편집기 개발, 각종 객체들이 갖는 고유한 소리와 이들간 충돌에 의해 발생하는 각종 효과 음향들을 작성하고, 편집하는 기능들을 수행하는 청각객체 편집기 개발, 가상환경내 각종 객체들이 갖는 고유한 움직임들을 저작하는 기능을 수행하는 객체행위 저작기 개발, 그리고 시각 및 청각 편집기와 객체행위 저작기 등을 통해 독립적으로 저작된 각종 객체들을 사용해 3차원 시청각 환경을 저작하는 기능을 수행하는 시청각 환경 저작기 개발 등이 포함된다.

또한, 3차원 시청각 환경 제시 및 통제 시스템 개발에는 입력 장치들과 연계해 원활한 감성측정 평가를 지원하기 위해 실시간 대화방식으로 3차원 시청각 환경의 제시를 조정 및 통제하는 기능을 수행하는 시청각 환경 제어기 개발, 시청각 환경 내에서 일어날 수 있는 각종 사건들의 처리 방법들을 저작하는 기능을 수행하는 시청각 환경 사건 저작기 개발, 그리고 3차원 시청각 환경 제시 시스템의 구성 요소들을 하나의 시스템으로 통합해 주며, 외부의 각종 입력 장비들과 시청각 환경내 동적인 객체의 물리적인 특성들을 모사해 주는 각종 시뮬레이션 모듈과의 인터페이스를 제공하는 시청각 환경 커널 개발 등이 포함된다.

Ⅳ. 연구개발결과

상기의 연구들은 1995년 12월부터 총 3년간에 걸쳐 연차적으로 수행되었으며, 1차년도에는 선진국의 관련자료 수집과 협력체제 구축 및 개발 플랫폼의 기본사양 등을 결정하였다. 시청각 환경 제시 시스템의 개략적 구조를 설계 하였으며 주요 기능을 정립하였고, 시청각 환경 제시 시스템의 기본 운영체계 및 관리체계를 구축하였다.

2차 년도에는 지속적인 관련자료 수집 및 통합 시스템의 구조를 설계하였고, 아울러 시각과 청각 환경의 저작과 제시를 위한 주요 기능 모듈들을 개발하였다. 이에 더불어 차량 시뮬레이터 중심의 시뮬레이터 검증을 위한 방법론 연구를 완료하였으며 입체영상 제시 관련 홀로그램 연구 수행 및 주거환경 대상의 데모 시스템을 구축하였다.

3차 년도에는 기존에 개발된 시각과 청각 환경의 저작과 제시를 위한 모듈들을 보완, 발전시키고, 추가기능 모듈들을 개발하였으며, 시각, 청각, 행위모사 등의 서브 시스템들을 통합하였다. 타 모의환경들(운동감,촉감,후각,열환경)과의 효율적 통합을 고려한 소프트웨어 구조를 설계하였으며 프로토타입 시스템으로 자전거 주행 시스템을 개발하였고 개발된 시청각 환경 제시 시스템의 검증 및 보완 작업을 중점적으로 수행하였다.

Ⅴ. 연구개발결과의 활용계획

3차원 시청각 환경 제시 시스템은 1차적으로 감성측정평가 시뮬레이터에서 필요로 하는 다양한 3차원 시청각 환경들을 저작하고, 제시하는데 활용될 것이다. 또한, 3차원 시청각 환경 제시 시스템을 가상 프로토타이핑 분야에 활용함으로써 제품이나 환경의 설계 및 생산에 소요되는 시간, 노력 및 비용 등을 현저히 절감할 수 있을 뿐만 아니라 제품의 질 향상에도 크게 기여할 수 있을 것이다.

3차원 시청각 환경 제시 시스템 개발을 통해 습득한 가상현실 기술과 시뮬레이션 관련 기술들을 앞으로 체험형 교육, 오락 및 훈련 등을 위한 각종 시뮬레이터(자동차, 항공기, 전차, 선박 등) 개발에 직접적으로 활용할 수 있을 것으로 기대된다. 또한, 3차원 시청각 환경 제시 시스템의 시청각 환경 저작 시스템을 기반으로 개발된 가상 스튜디오(Virtual Studio)는 향후 다양한 멀티미디어 응용분야에서 창의적인 응용 사례들을 구축하는데 활용할 계획이며, 공동 개발사인 문화방송에서는 선거방송, 스포츠 중계, 뉴스 보도 방송 등에 이미 활용한 바 있으며, 앞으로 어린이 교육/놀이 프로그램 제작, 일기예보, 이벤트성 프로그램 제작 등에 확대하여 활용할 계획이다.

한편, 본 연구과제에서 개발한 3차원 시청각 환경 제시 시스템은 오늘날 다양한 산업분야에서 적용하려는 가상현실 기술이 안고 있는 대부분이 요소 기술들을 포함하고 있기 때문에 본 연구를 통해 습득한 기술과 경험은 가상현실 기술의 적용사례 개발에도 곧 바로 활용할 수 있을 것으로 기대된다.

여 백

SUMMARY

Objective of this research is to develop a display technology which renders and presents 3D visual and audio environment effectively. This study is part of development of the Human Sensibility Evaluation Simulator which generates various simulated environment, evaluate and analyze consumer products or living environmental factors.

Main details of research are modeling of 3D virtual environment, developing a real time management system of the simulated 3D visual and audio environment, Ergonomic design of 3D visual and audio system, and evaluation of the reality of 3D visual and audio environment.

Next sub-researches are accomplished for this research.

- Study of Synchronization System for Visual and Sound Media
- Representation Technique of 3-dimensional Stereoscopic Visual Parameters
- Development of Evaluating Method for Driving Simulator
- A Study on Behavioral Specification and Simulation for Constructing Virtual Environment
- A Study on Development of Design Technique for Virtual Reality System in Human Engineering.

여 백

CONTENTS

여 백

목 차

제 1 장 서 론

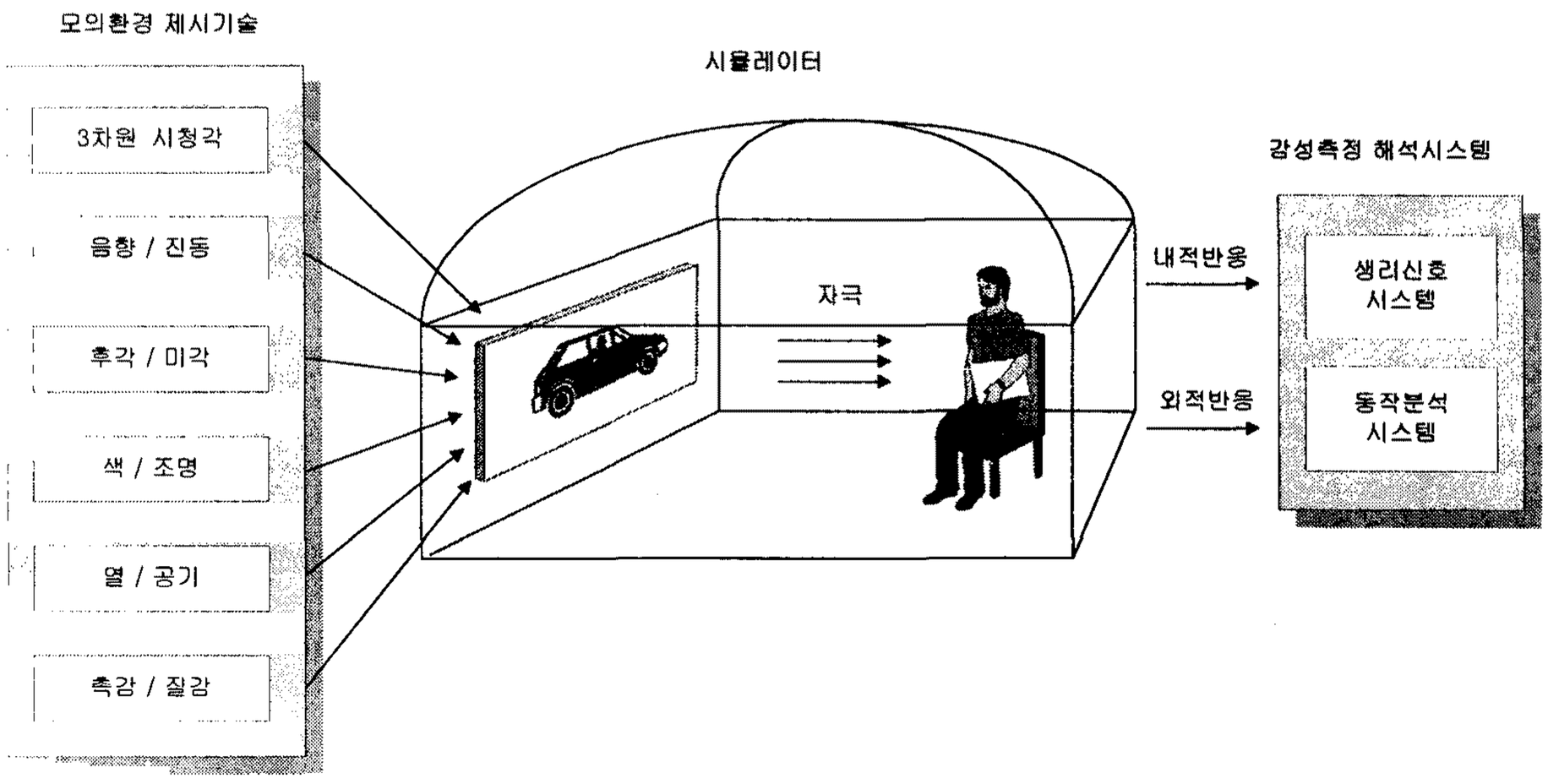
오늘날 대부분의 기업들은 점차 제품기술이 대중화 되고, 보편화 됨에 따라 종전의 품질, 기능 및 가격 중심의 경쟁체제에서 벗어나, 제품의 디자인이나 사용의 편리성과 만족성 등을 중시하는 감성중심의 제품개발에 많은 관심을 가지고 있다[1]. 그러나, 감성공학에 대한 국내에서의 연구가 일천한 관계로 기업들의 감성제품 개발에 활용할 수 있는 가장 기초적인 감성지표 데이터베이스는 물론 감성측정평가에 필요한 시설이나 장비 및 시스템 등이 전무한 실정이었다.

이러한 배경하에서 장차 기업체, 연구소 및 학계 등에서의 감성공학에 대한 다양한 요구들을 충족시키고, 제품 개발에 감성공학 기술을 적극 활용할 수 있는 기반을 마련함으로써 국산 제품의 경쟁력을 제고하기 위해 '95년 12월 1일부터 선도기술개발사업(G7 프로젝트)의 일환으로 "감성공학기술 개발 사업"은 시작되었다. 사업의 궁극적인 목표는 제품의 경쟁력과 삶의 질을 향상시키기 위한 인간 중심의 제품과 환경 설계 응용기술의 기반을 구축하는데 있었다. 이를 위하여 감성요소기술 개발, 감성측정평가 시뮬레이터 개발, 그리고 감성의 제품 및 환경 응용기술 개발 등과 같은 3가지의 구체적인 실천목표가 설정되었고, 각각은 다수의 단위 과제들을 포함하는 대과제의 형태로 연구가 진행되었다[2].

한편, 감성공학기술 개발 사업의 3가지 구체적인 실천목표를 살펴보면 첫번째가 감성요소기술 개발로 인간의 감성특성을 파악하고, 감성에 관련된 심리, 생리지표 등을 개발하며, 감성을 제품설계에 반영시키는데 필요한 감성관련 제반요소 기술을 개발함으로써 국내에 감성공학 연구 기반을 구축하는데 있다. 아울러, 중장기적으로 감성공학 분야의 인력수급을 원활하게 하고, 산업의 선진화를 뒷받침하는 것을 목표로 한다. 두번째는 감성측정평가 시뮬레이터 개발로 제품과 환경에 대한 인간 감성의 반응 특성을 파악하기 위하여 인공적으로 물리적인 환경을 자유롭게 바꿀 수 있는 실험시설의 확보를 목표로 모의환경 제시 기술과 감성의 생리, 심리적 측정평가 소프트웨어를 개발하고, 주요 산업제품 및 환경 평가에 활용될 감성측정평가 시뮬레이터를 개발하는데 있다. 세번째는 감성의 제품 및 환경 응용기술 개발로 감성요소기술이나 감성측정평가 시뮬레이터와 연계하여 우리나라 경제를 주도하고 있으며, 감성공학의 파급효과가 큰 자동차, 가전/정

보산업 등의 주요 제품이나 주거 및 작업환경, 생활에 직접 응용될 수 있는 기술을 개발하여 제품 경쟁력을 높이고 삶의 질을 향상하기 위한 기술 기반을 확립하는데 있다[3].

상기와 같은 배경하에서 감성공학기술 개발 사업에서는 제품이나 환경에 대한 인간의 감성반응을 측정 및 평가할 수 있는 공동의 실험평가 시설인 "감성측정평가 시뮬레이터"의 개발을 추진하게 되었다. 감성측정평가 시뮬레이터는 제품이나 환경에 대한 인간의 감성반응 특성을 파악하기 위하여 인공적으로 물리적 환경을 자유롭게 바꿀 수 있는 실험시설로 모의환경 제시기술, 모의실험 장치(시뮬레이터) 및 감성측정 해석 시스템 등으로 구성된다. 이중 모의환경 제시기술은 실험실 공간에서 인간의 감성반응 특성을 효율적으로 파악할 수 있도록 다양한 환경이나 상황을 인위적으로 제시해 주는 기술로서 3차원 시청각 환경 제시 기술, 음향 및 진동 환경 제시 기술, 후각 환경 제시 기술, 색/조명 환경 제시 기술, 열 환경 제시 기술, 촉각 및 질감 제시 기술 등이 있다[4].



<그림 1> 감성측정평가 시뮬레이터 개념도

3차원 시청각 환경 제시 기술은 최근에 급속도로 발전하고 있는 가상현실(Virtual Reality) 기술을 이용하여 감성측정평가 대상 제품이나 환경에 대한 시각 및 청각 정보들을 입체적이고, 현실감 있게 모사해 주는 기술이다. 인간이 외부 세계로부터 획득하는 정보들 중에서 시각을 통해 획득하는 정보가 전체 정보량의 80%를 상회한다고 하니 감성측정평가 시뮬레이터의 현실감 구현에 시청각 환경 제시 기술이 차지하는 비중이 그만큼 크다고 말할 수 있다. 이러한 3차원 시청각 환경 제시 기술은 실험환경의 현실감

증대 차원에서 궁극적으로는 다른 모의환경 제시 기술들과 통합되어 피측정자에게 제시되어야 하고, 원활한 감성측정 및 평가를 위하여 실험환경은 피측정자 또는 실험 통제관(Controller) 등에 의해 실시간으로 제어될 수 있어야 한다[5].

제 1 절 연구개발의 목적

감성측정평가 시뮬레이터가 필요로 하는 여러 가지 형태의 모의환경 제시 기술 중에 3차원 시청각 환경 제시와 관련 있는 제반 기술들을 연구하며, 궁극적으로는 다양한 제품이나 환경에 대한 감성측정 평가시에 필요로 하는 물리적인 또는 가상적인 3차원 시청각 환경들을 신속, 편리하게 저작할 수 있고, 또한 원활한 감성측정평가를 지원하기 위해 저작된 3차원 시청각 환경들을 실시간으로 적절히 조정, 통제하며 피측정자에게 제시할 수 있는 3차원 시청각 환경 제시 시스템을 개발하는 데 있다.

제 2 절 연구개발의 필요성

1. 기술적 측면

오늘날 선진국에서는 대부분의 가전제품, 자동차, 토목, 건축 및 주거환경 등에 대한 설계가 CAD(Computer-Aided Design) 시스템에 의해 수행되고 있는 상황에서 CAD 시스템에 의해 작성된 각종 데이터들을 인간의 시각 및 청각 기관에 3차원 입체적으로 제시해 줌으로써 인공 현실감을 창출하려는 가상현실 기술이 급속하게 발전하고 있다. 가상현실이 창출하는 모의환경을 경험하는 사람은 실제처럼 설계물을 조작해 보거나 가상환경에 거주해 보는 등과 같은 인간과 설계물 사이의 상호작용을 통해 다양하고 직접적인 체험을 시도해 볼 수 있다.

감성공학에서 이러한 3차원 시청각 모의환경 제시기는 여러 사람들에게 동일한 조건의 모의환경을 반복하여 제공할 수 있으므로 해당 설계물이 다양한 사람에게 미치는 영향을 제어실험(Controlled Experiment)할 수 있는 하나의 실험장치로 활용할 수 있다. 마치 입자가속기를 통해 고 에너지 물리학 연구 분야에서 다양한 가설을 시험, 증명하듯이

3차원 시청각 모의환경 제시기는 감성공학 분야에서 다양한 환경이 인간에게 미치는 영향에 관한 감성지표 규명을 위한 필수적인 실험도구로써 역할을 할 수 있을 것이다.

또한, 모의환경 제시기와 사용자 사이에 상호작용을 하면서 설계변경을 제시하고, 이를 반영한 새로운 설계물이 짧은 시간에 사용자에게 다시 제시될 수 있는 감성 디자인 체계로 발전시 킬 수 있다. 따라서 3차원 시청각 모의환경 제시기는 감성공학 연구에 참여하는 연구자들이 언제나 사용할 수 있는 공동의 실험시설로 활용될 수 있으므로 연구 시작단계부터 조속히 구축하여 운영되는 것이 바람직할 것으로 사료된다.

한편, 3차원 시청각 환경 제시 시스템을 구축하기 위해서는 CAD 기술, 가상현실 기술, 시뮬레이션 기술, 인간과 가상환경 간의 상호작용 기술, 전문가들 사이의 협업을 위한 네트워크 기술, 가상환경의 현실감 향상을 위한 운동감/작동감 제시 기술, 그리고 감성지표와 관련한 데이터베이스 기술 등과 같은 미래 정보기술의 총아 분야들이 모두 망라되어 있다. 그러므로 본 연구를 통해 얻을 수 있는 기술적인 차원의 의의로는 가상환경 제시 및 평가 시설을 구축, 감성측정 평가 및 제품 응용기술을 개발하여 21세기 국가적 차원의 핵심 정보 기술들에 대한 독자적인 기반을 마련하고 독창적인 응용기술을 확보할 수 있을 것이다[6],[7],[8].

2. 경제/산업적 측면

3차원 시청각 모의환경 제시기의 기반기술은 멀티미디어, 컴퓨터그래픽스, CAD, 실시간 시뮬레이션 및 가상현실 기술 등을 포함하는 복합적인 기술 분야이다. 비록, 실시간 시뮬레이터의 시장은 탈 냉전시대에서 그 동안 군사응용 분야에 집중되어 있던 시장을 민간 시장체제로 전환하는데 많은 어려움을 가지고 있으며 새로운 돌파구 마련이 필요할 것으로 사료된다.

한편, 3차원 시청각 모의환경 제시기는 가상 프로토타이핑(Virtual Prototyping) 분야에 활용함으로써 제품이나 환경의 설계 및 생산에 소요되는 시간, 노력 및 비용 등을 현저히 절감할 수 있을 뿐만 아니라, 제품의 질 향상에도 크게 기여할 수 있을 것으로 판단된다. 그리고, 제품생산에 소요되는 노력과 비용 및 시간을 절약할 수 있고, 다양한 분야의 전문가들과 소비자들의 참여를 유도함으로써 제품의 기능적 신뢰성 향상과 제품의 다양화를 도모할 수 있다.

또한, 제품을 구매하고자 하는 고객에게 가상의 모델을 통해 시/공간의 제약을 받지 않고 가장 효율적인 프리젠테이션을 할 수 있게 됨으로써 제품의 판매, 홍보에도 크게 기여할 수 있다. 바이어는 언제, 어디에 있든지 컴퓨터를 통해 제품을 볼 수 있고, 동작시켜 볼 수 있을 것이다. 따라서 시/공간적으로 판매시장이 크게 확장되는 효과를 가져 올 수 있을 것이다.

한편, 3차원 시청각 환경 제시기 개발을 통해 습득한 가상현실 기술과 시뮬레이션 관련 기술들은 앞으로 체험형 교육, 오락 및 훈련 등을 위한 각종 시뮬레이터(자동차, 항공기, 전차, 선박) 개발에 직접적으로 활용할 수 있을 뿐만 아니라, 의료, 과학, 엔지니어링, 디지털 라이브러리, 레져, 스포츠, 방송 등과 다양한 산업분야의 기반기술로 이들 분야의 발전에 크게 기여할 것이다. 또한, 본 과제를 통해 축적된 기술과 경험을 토대로 가상현실 기술의 산업화, 이를 활용한 고 부가가치 산업 및 제품의 창출이 가능할 것으로 기대된다.

3. 사회/문화적 측면

다가오는 2000년대의 정보화 사회에서 3차원 시청각 모의환경 제시기는 사람들로 하여금 다양한 정보를 가정에서 입수하여 경험하게 하는데 기여할 수 있을 것이다. 특히, 안방에서 가상세계로의 여행을 실현해 본다든지 지리적으로 서로 멀리 떨어져 있는 여인들이 마치 동일한 데이트 장소에 있는 것과 같은 환경을 조성해 줄 수가 있을 것이다.

또한, 장차 고령화 사회에서 중요한 사회적 문제로 대두될 노약자나 장애자의 사회활동 범위 확대와 그들로 하여금 사회의 능동적인 일원으로 참여할 수 있는 여건과 환경을 조성하는데 본 연구에서 개발하고자 하는 3차원 시청각 모의환경 제시기의 관련 기술들이 핵심적인 역할을 수행할 수 있을 것이다.

한편, 개발 시스템은 궁극적으로 네트워크를 통해 지리적으로 흩어져 있는 다양한 분야의 사람들을 제품생산에 참여시킴으로써 제품생산에 직접적으로 관련이 있는 엔지니어와 디자이너 뿐만 아니라, 의사결정 권한을 가진 관리자, 제품의 성능을 시험하는 테스터, 그리고 소비자들을 제품생산 과정에 적극적으로 개입하게 하여 체계적이고, 신

속하며 효율적인 제품개발이 가능하게 함은 물론, 이들간의 의사소통 과정 자체를 매우 효율적으로 바꿀 수 있다.

그리고, 제품에 대한 소비자의 욕구가 경제적인 제품을 선호하는 실용성 욕구에서 개인적인 것을 중시하는 감성욕구로 변화하여 소비자는 다른 제품과 차별화된 기능을 가진 제품, 자신이 좋아하는 디자인을 가진 제품, 자신만의 개인용 제품을 원하고 있다. 따라서 제품의 속성 역시 기능 중심에서 인터페이스 중심으로, 제품 중심에서 사용자 중심으로 변해가고 있는 추세이다[9].

한편, 상기와 같은 시스템이 사회 전반의 기능분야에 보편화 됨으로써 전문가 집단의 재택근무가 현실로 다가 올 것이며, 지리적인 제약과 시간적인 제약을 뛰어 넘어 전문가들의 활발한 토론문화가 정착될 것으로 생각된다. 이렇게 되면 현재 분야별로 부족한 전문가 문제를 어느 정도 극복할 수 있을 뿐만 아니라, 소비자들도 그들의 요구들을 가정이나 직장에서 직접적인 방법으로 매우 손쉽게 제시할 수 있을 것으로 전망된다.

제 3 절 연구개발의 범위

3차원 시청각 환경 제시 시스템은 크게 시청각 환경 저작 시스템과 시청각 환경 제시 및 통제 시스템으로 구성된다. 즉, 다양한 제품이나 환경에 대한 감성측정 평가에 필요한 물리적 또는 가상적인 3차원 시각 및 청각 환경을 신속하고 편리하게 저작할 수 있는 시스템과 원활한 감성측정 평가를 지원하기 위하여 시청각 환경 저작 시스템을 통해 저작된 3차원 시청각 환경을 실시간 대화식으로 적절히 조정 및 통제하며, 피실험자에게 제시할 수 있는 시스템이 그것이다.

3차원 시청각 환경 저작 시스템 개발에는 먼저 지형/도로, 장애물, 건물, 사람, 자동차 등과 같은 각종 시각적인 객체들을 작성하고, 편집하는 기능을 수행하는 시각객체 편집기 개발, 각종 객체들이 갖는 고유한 소리와 이들간 충돌에 의해 발생하는 각종 효과 음향들을 작성하고, 편집하는 기능들을 수행하는 청각객체 편집기 개발, 가상환경내 각종 객체들이 갖는 고유한 움직임들을 저작하는 기능을 수행하는 객체행위 저작기 개발, 그리고 시각 및 청각 편집기와 객체행위 저작기 등을 통해 독립적으로 저작된 각종 객체들을 사용해 3차원 시청각 환경을 저작하는 기능을 수행하는 시청각 환경 저작기 개발 등이 포함된다.

또한, 3차원 시청각 환경 제시 및 통제 시스템 개발에는 입력 장치들과 연계해 원활한 감성측정 평가를 지원하기 위해 실시간 대화방식으로 3차원 시청각 환경의 제시를 조정 및 통제하는 기능을 수행하는 시청각 환경 제어기 개발, 시청각 환경 내에서 일어날 수 있는 각종 사건들의 처리 방법들을 저작하는 기능을 수행하는 시청각 환경 사건 저작기 개발, 그리고 3차원 시청각 환경 제시 시스템의 구성 요소들을 하나의 시스템으로 통합해 주며, 외부의 각종 입력 장비들과 시청각 환경내 동적인 객체의 물리적인 특성들을 모사해 주는 각종 시뮬레이션 모듈과의 인터페이스를 제공하는 시청각 환경 커널 개발 등이 포함된다.

한편, 3차원 시청각 환경 제시 시스템 개발과 직접 또는 간접적으로 관련성 있는 연구들도 아울러 수행되었다. 즉, 입체영상 제시방법과 HMD 조정 연구, 시뮬레이터 검증수법 개발, 지형정보 데이터베이스 구축 및 가상현실 시스템의 인간 공학적 설계기법 연구 등이 그것이다.

상기의 연구들은 1995년 12월부터 총 3년간에 걸쳐 연차적으로 수행되었으며, 1차 년도에는 선진국의 관련자료 수집과 협력체제 구축 및 개발 플랫폼의 기본사양 등을 결정하였다. 2차 년도에는 지속적인 관련자료 수집 및 통합 시스템의 구조를 설계하였고, 아울러 주요 기능 모듈들을 개발하였다. 3차 년도에는 기존 개발 모듈들을 보완, 발전시키고, 추가기능 모듈들을 개발하였으며, 시각, 청각, 행위모사 등의 서브 시스템들을 통합하고, 프로토타입 시스템으로 자전거 주행 시스템을 개발하고, 시스템의 검증 및 보완 작업을 중점적으로 수행하였다. 본 연구의 연차별 주요 연구개발 범위를 정리하면 아래의 <표 1>과 같다.

<표 1> 연차별 주요 연구내용

년 도	주 요 연 구 내 용
1차 년도	• VR, 시뮬레이터 관련 기초자료 수집 및 해외 구축사례 조사 • 상용 3차원 청각 제시 시스템 조사 • 시스템 설계 및 기본 운영체제 구축 • 기본 하드웨어 및 소프트웨어 개발환경 선정

2차 년도	• 시스템의 하드웨어 및 소프트웨어 체계 구축 • 시각 및 청각 시스템의 설계와 주요 기능모듈 개발 • 시각 및 청각 시스템의 통합 소프트에어 구조 설계 • 시뮬레이터 검증방법 및 입체영상 제시방법 연구
3차 년도	• 시각 및 청각 시스템의 추가기능 개발과 기존모듈 보완발전 • 가상객체의 행위모사 시스템의 개발 • 시각, 청각 및 객체행위 시스템의 통합, 검증 및 평가 • HMD 조정 및 VR 시스템의 설계 및 검증방법 연구

제 2 장 국내외 기술개발 현황

본 연구에서 개발하려는 3차원 시청각 환경 제시 시스템은 궁극적으로 감성측정평가 시뮬레이터를 구축하기 위한 노력의 일환이다. 감성측정평가 시뮬레이터에서 3차원 시청각 환경 제시 시스템이 담당해야 할 역할은 감성측정평가 대상 제품이나 환경에 대한 시각 및 청각 정보들에 대해 인공 현실감을 구현하는 것이다. 즉, 설계 및 개발하려는 제품이나 환경에 대해 외형적인 특성, 기능적인 특성 및 구조적인 특성 등을 컴퓨터 시스템 상에 구현한 다음, 이를 전문가 또는 사용자들에게 제시하여 그들의 감성적인 반응을 측정하고 평가하는 것이다. 이는 제품이나 환경을 본격적으로 생산하기 이전에 설계단계에서 감성적인 평가를 가능하게 함으로써 보다 인간 친화적인 제품이나 환경을 개발할 수 있을 것으로 기대된다.

한편, 본 연구와 연관 있는 요소기술 분야는 매우 다양하고 복합적이다. 그러나, 요소 기술들이 통합된 시스템 측면에서 관련기술 분야를 살펴보면 크게 가상현실 시스템과 시뮬레이터 시스템을 꼽을 수 있다. 가상현실 기술은 시청각 정보를 컴퓨터 상에 구현하는 기술이고, 시뮬레이터 기술은 보다 높은 현실감을 구현하기 위해 필요한 움직임과 같은 부가적인 정보를 구현하기 위한 기계적인 시스템들을 포함하는 기술이다.

그러나, 최근 들어 이들 두개의 기술분야가 상호 협조적인 관계를 유지하면서 발전해 각 있기 때문에 이들에 대한 기술적인 영역의 한계를 구분 짓기가 매우 어려운 것이 현실이다. 그러므로 본 연구과제와 관련한 국내외 기술개발 현황은 크게 가상현실 시스템의 개발 현황과 시뮬레이터 시스템의 개발 현황을 살펴 봄으로써 3차원 시청각 환경 제시 시스템 개발과 연관 있는 중요한 요소 기술들의 개발 실태를 파악할 수 있을 것으로 사료된다.

제 1 절 국내 기술개발 현황

현재 국내 가상현실 관련 응용기술에 대한 연구는 선진국의 산.학.연 연합체제 운영과 같이 각 연구 주체들이 정형화된 형태로서 진행되지 못하고 있으며, 주요 연구기관

으로는 KIST, KAIST, ETRI, 원자력연구소, 포항공대 등에서 자체적인 연구 테마로 수행하고 있다. 아직 국내의 가상현실의 연구개발 능력이나 소프트웨어 기술 잠재력은 선진 기술과 어깨를 나란히 할 정도로 발전하였으나, 이를 집약시키는 가상현실 실용화 프로젝트 관리 및 시스템 통합에 관한 노하우의 축적은 미약한 실정이다

국내 가상현실 관련 활동은 실용적인 응용기술 보다는 핵심 기반기술 관련 선행 연구 중심으로 이루어지고 있으며, 아직 상용화 사례는 발표된 바가 없으나 일부 기반 기술을 기업의 요구에 따라 단편적인 적용사례는 다양한 응용분야에서 도출되고 있다. 그 실례 중에 하나가 KIST와 문화방송이 개발한 가상 스튜디오 시스템이다. 특히, 현재 감성공학 선도기술 개발 사업에서 가상현실 기술은 시청각 뿐만 아니라 후각, 힘각, 체감 등 오감을 총체적으로 제시하는 연구는 세계적으로도 첫 시도라고 할 수 있으며 또한, 가상현실에서 인체에 미치는 영향 평가를 통한 가상현실 설계 시스템 구축은 아마도 세계에서 최초이며 유일한 것이다.

본 과제에서 개발할 감성측정평가 시뮬레이터는 인간감성에 관한 연구용 시뮬레이터라고 말할 수 있다. 이러한 연구용 시뮬레이터의 수요는 적용분야가 확산되면서 소비자 차원에서의 소요 보다는 단기적으로 감성공학 기술을 발전시키는 데 기여할 것으로 판단된다.

한편, 그 동안 국내에서 연구용 시뮬레이터가 개발되거나 외국으로부터 도입된 사례는 없다. 또한, 이 분야에서 가장 선진국이라 말할 수 있는 미국에서도 연구용 시뮬레이터는 어느 규모 이상의 대학에서 인간요소(Human Factors) 연구분야에 적용하고 있는 실정이다. 그러나, 연구용 시뮬레이터의 근간이 되는 가상현실 기술은 '91년도 대전 엑스포 '93의 사전 행사로 거행된 서울 컴퓨터 아트 '91에서 소개 되었고, 현재 KIST, 시스템 공학 연구소(현 ETRI), 삼성종합기술원 등에서 가상현실 시스템을 구축, 운영 중에 있다. 또한, 과학기술원, 포항공대 등에서도 기반기술에 대한 연구를 수행하고 있다.

그 동안 국내 가상현실 기술은 짧은 연구개발 기간에도 불구하고, 외국 부품기술을 국내에서 조립하여 운영하는 기술분야는 선진국 수준과 견줄 만한 정도로 발전하였다. 또한, 응용사례 개발 분야는 가구전시 시스템, 기중기 작동 시스템, 원격 무인 지게차, 분자결합 모의실험 시스템, 가상조립 시스템 등과 같은 다양한 분야에서 수행되었다.

그러나, 이러한 개발 노력들은 많은 사람들의 관심과 이해 부족으로 장기간 체계적인 연구수행이 이루어지지 못하고, 일시적이고 단편적인 연구개발에 그치고 말아 많

은 아쉬움을 남겼다. 이러한 개발 노력들 중에서 그래도 국내 가상현실과 시뮬레이터 분야의 발전에 초석을 마련했다고 평가할 수 있는 KIST의 인공 현실감(Virtual Reality) 시스템 개발과 국민대학교 자동차공학과의 실시간 차량 시뮬레이터 개발을 간략히 살펴보기로 한다.

1. KIST의 가상현실 연구

국내 가상현실 기술의 도입 및 실용화에 그 동안 선두적인 역할을 수행해 온 KIST는 '91년 12월부터 '93년 11월까지 2년간에 걸쳐 '93 대전 엑스포 행사에 출품하기 위해 "EXPO 조작 로보트 시스템"을 개발하였다. 꿈돌이로 명명된 이 로보트 시스템에서는 임의의 대상자의 얼굴 형상을 Moire 기법을 사용하여 3차원적으로 측정한 다음, 그 형상 데이터를 처리한다. 이렇게 처리한 얼굴형상 정보에 대해 컴퓨터 그래픽 기술을 이용하여 다양하게 표정을 변화시키고, 또한 로보트의 작업 프로그램을 작성하는 통합 시스템을 개발하였다[10].

또한, KIST는 '92년 12월부터 '94년 10월까지 약 2년간에 걸쳐 "인공 현실감 시스템 개발"이라는 연구를 시작하였다. 이 연구는 국내 최초 가상현실에 대한 본격적인 연구로 간주되고 있으며, 다양한 산업분야에서 가상현실 응용 시스템을 개발할 때 활용할 수 있는 여러 가지 가상현실 하드웨어 장비와 소프트웨어 개발 도구들을 소개하고 있으며, 조립설계 시스템 개발, 분자 그래픽스 개발 및 원격존재 기술 개발 등과 같은 다양한 응용 시스템 개발을 통해 가상현실 응용 시스템 개발 접근방법과 구축사례를 제시하였다[11].

그 밖에 KIST는 '93년도에 청각 현실감 기술에 관한 연구를 통해 가상환경을 구성하는 청각 데이터베이스를 구축하고, 인터페이스를 개발하였으며, '94년도에는 가상 현실감 시스템 운영체계에 관한 연구를 통해 가상환경 구축 시간을 단축하는 도구를 개발하였다. 또한, 창 밖 시계 비쥬얼 시뮬레이션 툴킷 프로토타입 개발에 관한 연구를 통해서는 공항 주변환경 모델 Import 하고, 시각 데이터베이스를 최적화하며, 실시간 Fly-by 할 수 있는 프로토타입 시스템을 개발하였다. 그리고, '94년 5월부터 '96년 4월까지 2년간에 걸쳐 휴먼 로보트의 World Modeling 기능 개발 연구를 통해 KIST 주위 환경을 모델링하였고, 로보트 보행 시뮬레이션과 주위환경 통합, 실시간 상호작용 기능 등을 개발하여 휴먼 로보트의 임무계획, 계획 디버깅 및 업무정의 등에 활용하였다[12].

한편, KIST는 1996년과 1997년 2차례에 걸쳐 문화방송과 공동으로 가상 스튜디오를 개발하였다. 가상 스튜디오에서는 연기자나 카메라가 가상세트가 창출한 3차원 공간에서 마음대로 움직이면서 3차원 영상물을 마음대로 만들 수 있다. 단, 물리적인 세트에서가 아니라 파란바탕의 세트에서 연기자는 움직이며, 이를 따라 다니는 카메라의 움직임, 초점거리 등을 감지하며, 가상세트와 합성함으로써 시청자는 물리적인 세트에서 출연한 것과 마찬가지의 효과를 가상세트에서 볼 수 있다. 가상세트는 물리적인 세트에 비해 여러 가지 장점을 가지고 있다. 즉, 물리적인 세트는 세트를 디자인하면 목공소에서 세트를 제작하게 되고, 그 세트를 방송국 스튜디오에 설치하게 된다. 그리고 방송이 끝나면 다음 방송까지 창고에 저장하기 위해 분해하고 다음 방송 때까지 보관하고, 다시 설치해야 하는 과정들을 반복하게 된다[13].

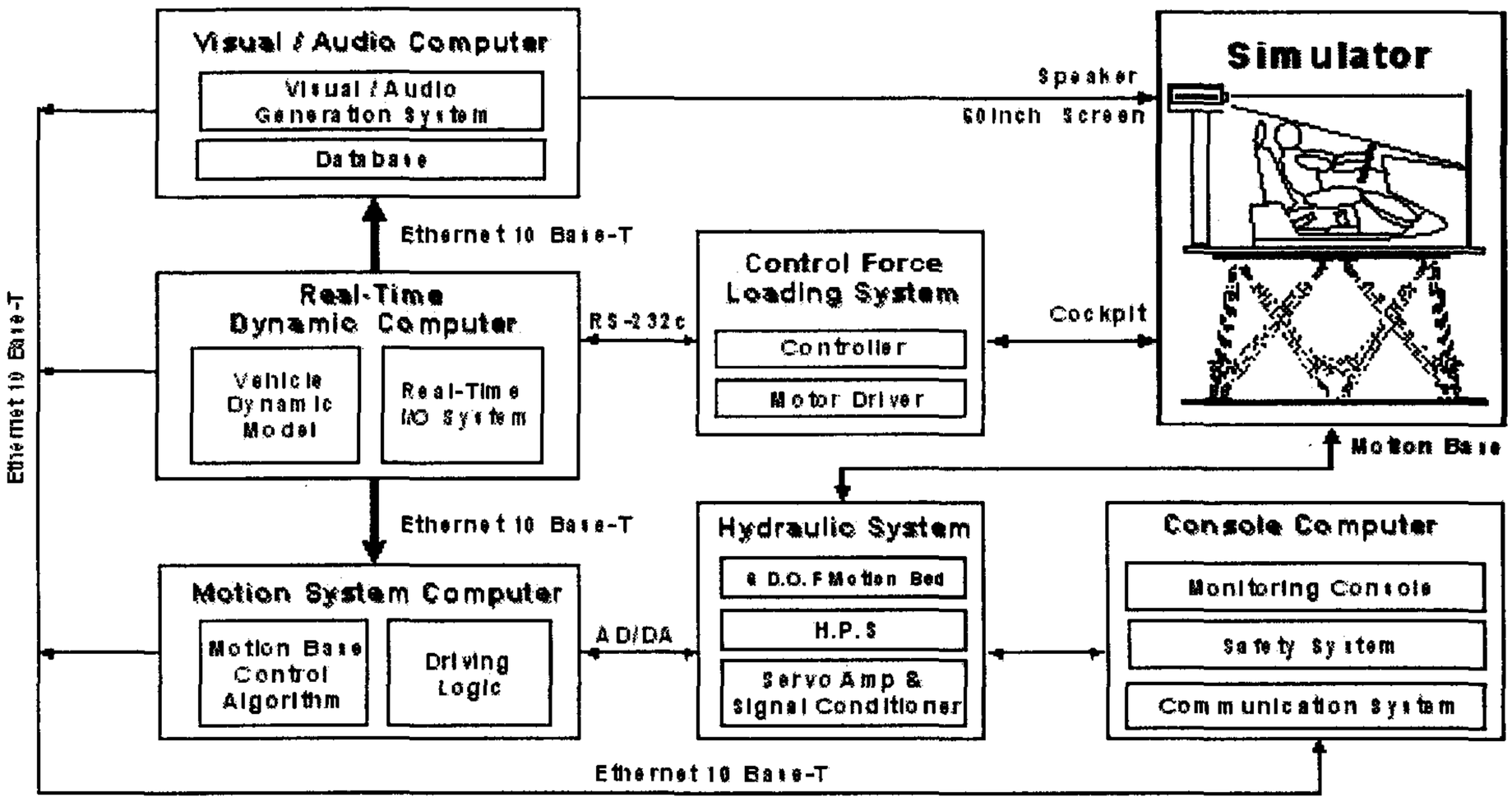
반면, 가상세트는 이러한 세트를 컴퓨터에서 디자인 하게 되고, 그 디자인이 가상세트를 구성하는 기반으로 활용되고, 이렇게 창출된 가상세트는 컴퓨터 디스크 메모리에 저장된다. 또한, 연출자가 원하면 방송 때마다 세트 디자인을 마음대로 언제나 바꿀 수 있기 때문에 가상 스튜디오 기술의 발전은 방송매체에 일대 혁신을 가져 올 첨단기술 분야이다.

본 연구에서 개발한 가상 스튜디오는 크게 3가지 부분으로 나뉘어져 있다. 우선 전체 프로그램에서 공통적으로 사용되고, 각종 그래픽 효과를 만들어 낼 수 있는 클래스 라이브러리 커널이 있고, 다음은 방송하는 과정 일체를 관리하는 운영 프로그램과 유저 인터페이스를 이용하여 저작을 보다 쉽게 할 수 있도록 구성된 저작 프로그램으로 구성된다. 여기서 개발된 운영 프로그램과 저작 프로그램의 많은 부분은 커널 클래스 라이브러리로 만들어져 있으며, 원하는 경우 사용자는 이 클래스 라이브러리를 확장할 수 있고, 이를 이용하여 다른 운영 프로그램이나 저작 프로그램을 개발할 수 있다.

2. 국민대의 실시간 차량 시뮬레이터 개발

최근 국민대학교는 국내 최초로 실시간 차량 시뮬레이터의 개발을 완료하여 1998년도부터 본격적인 시험운영에 들어 간 것으로 알려져 있다. 실시간 차량 시뮬레이터는 PC를 기반으로 LAN 시스템에 의해 기능 호스트들이 상호 유기적으로 연결되어 있는 분산 시스템의 형태를 취하고 있다[14]. 아래의 <그림 2>는 실시간 차량 시뮬레이터의

기능적 구조를 나타낸 것으로 시뮬레이터의 구성과 전체 시스템의 개념을 보여주고 있다.



<그림 2> 실시간 차량 시뮬레이터의 기능적 구조

먼저 주행 시나리오에 의한 운전자의 반응이 조향 휠 및 감가속 페달 등의 조작으로 표현되어 제어 힘 로딩 시스템의 센서가 이를 검출하고 신호처리를 거친 후 실시간 차량 시뮬레이션 컴퓨터에 입력된다. 차량 시뮬레이션 컴퓨터는 실시간으로 차량의 동력학적 거동을 해석하여 운전자의 입력에 따른 차량의 운동을 예측한다.

시각 및 음향 컴퓨터는 주행 상황에 맞는 주행환경을 재현하는 컴퓨터 그래픽 이미지 및 소음을 생성하고 프로젝터 및 스피커를 통해 재현한다. 유압 시스템으로 구동되는 6 자유도 Stewart 플랫폼으로 구성된 운동 시스템에서는 예측된 차량의 운동을 운동 플랫폼의 운동범위 내에서 재현이 가능한 샤시의 운동으로 축소 변환시키고, 제어 알고리즘은 이를 토대로 운동 플랫폼을 구동하기 위한 명령을 생성하고 실행하여 현실감 있는 운동 큐를 재현한다.

제어 힘 로딩 시스템은 운전자의 운전입력을 검출하는 기능과 더불어 운전자에게 현실감 있는 운전조작 장치의 반력 및 반토크를 피드백하는 기능을 갖는다. 시스템의 안전을 고려하고, 전체 시스템의 운전 상황을 효율적으로 파악하기 위하여 제어 상태 감시 시스템을 구성하고, 시스템의 오동작 및 위험 상황에 적극적으로 대처하기 위한 안전 시스템을 구성하였다.

한편, 차량의 운전석을 축소한 간단한 형태의 Cockpit은 실제 차량과 동일한 운전대, 계기판, 브레이크 및 엑셀레이터 페달로 이루어 진다. 각 서브시스템의 각종 데이터는 UDP/IP를 이용한 이더넷 및 RS-232C를 이용하여 통신한다.

제 2 절 국외 기술개발 현황

미국에서는 여러 회사가 고 품질 입체 표시기(Head Mounted Display, Stereo LCD Shutter Glass, BOOM등)가 개발 시판하고 있으며, 입체 음향기 역시 성공적으로 시판되고 있다. 물론, 가상현실의 가장 기본이 되는 고성능 그래픽 컴퓨터의 본산지도 역시 미국이다. 특히, 미국의 컴퓨터 기술의 저력은 기반기술이 워낙 광범위하고, 단단하다는데 있다. 즉, 어떠한 과제를 도출하고 성취하는데 필요한 일은 그 과제의 요소 부분을 잘 알고 있는 엔지니어들을 모아 조합을 하는 일이 주된 수행 작업이지 그 부분 자체를 개발하는데 큰 노력과 시간을 소요할 필요가 없다는 점이다. 그러므로 전략적인 목표가 설정되면 구현기간이 1년에서 1년 반정도면 어느 정도의 결과물을 산출하는 상황이다.

영국은 미국과 비슷한 시기에 가상현실 분야를 개척한 나라로서 다양한 가상현실 시스템용 주변기기(운동판, 데이타 장갑 등)가 영국에서도 개발되고 있으며, 가상현실 시스템 구축용 소프트웨어 및 하드웨어 통합 업체인 디비젼(DIVISION)사가 자리를 잡고 있다. 또한, 가상현실 게임 분야에서는 세계에서 가장 먼저 W-industry사에 의하여 기업화를 추진하였다. 앞으로 가상현실 분야를 중점 개발 산업 분야로 영국 상공부 차원에서 지원을 하고 있다. 그러나 영국의 대부분 가상현실 기업은 우리나라의 중소기업 수준에 머물러 있는 상황이므로 자금력 있는 후발자의 추진전략 여하에 따라서는 선두자리를 유지하기 어려울 것으로 예측된다.

일본에서는 Nintendo, SEGA등 전자 게임기 회사에서 가상현실 기술을 응용한 게임기 개발에 박차를 가하고 있다. 특히, Nintendo의 경우 미국의 실리콘 그래픽스(SGI)사와 제휴 Project Reality 라는 과제로 게임기용 멀티미디어, 그래픽 가속용 Chip Set 등을 발표하였고, '95년 말에는 이를 탑재한 64bit 게임기가 출하될 예정이다. 현재 일본의 가상현실 기술은 게임 분야를 중심으로 게임용 가상현실 주변기기의 대중화를 위한 첨병의 역할을 수행하고 있다. 한편, 연구 분야에서는 감성공학, 로보틱스 등 응용분야에 가상현실 기술을 적용하는 응용 시스템 개발이 주류를 이루고 있다.

궁극적으로 본 연구에서 개발하려는 3차원 시청각 환경 제시 시스템은 가상현실 기술을 주축으로 하고, 피실험자에게 제시하려는 가상환경의 현실감을 증대시키기 위해 시뮬레이터 분야에서 핵심기술로 간주되고 있는 운동감과 제어감을 제시하는 기술을 결합시켜야 한다. 그러므로 3차원 시청각 환경 제시 시스템 개발과 관련한 외국의 기술개발 사례는 가상현실 분야와 시뮬레이터 분야를 중심으로 살펴 보기로 한다. 본 연구에서는 시뮬레이터의 원조격인 비행 시뮬레이터와 최근 자동차 시뮬레이터의 대표격인 Iowa Driving Simulator 에 대해 소프트웨어 구조 측면을 중심으로 살펴 보기로 하고, 가상현실 시스템에 대해서는 단일 사용자 가상현실 시스템 중심으로 기능적인 측면과 소프트웨어 구조적인 측면에서 간단히 살펴 보기로 한다.

1. 비행 시뮬레이터(Flight Simulator)

비행 시뮬레이션 소프트웨어는 현실감을 피 실험자에게 주기 위하여 고정된 Frame Rate에서 수행되어야 한다[15]. 이는 시각과 청각등 감각에 따라서 상이한 frame rate가 유지되어야 하는데 시각에 대해서는 60Hz가 기본이다. 이러한 Frame 갱신률이 지켜져야 하는 시스템이기 때문에 실시간 시스템으로 구성되며 배부분 멀티 프로세싱/멀티 프로세서 환경에서 아주 다양한 모듈들이 실시간 규약을 지키기 위해서 강하게 연결된 크고 복잡한 시스템이다. 일반적인 실험환경을 구성하기 위해서는 복잡한 실제 시스템의 분석을 거쳐서 검증된 S/W들이 사용되어져야 하는데 이러한 실시간 효율성 강조로 모델링 대상의 항공기 설계와 시뮬레이터 S/W사이에 관계성이 적을 수 있다.

이러한 비행 시뮬레이션의 전 부분을 한곳에서 수행시킬 수 가 없는 데 그 이유는 넓은 지역에 퍼져 있는 전문가를 연결시키는 컴퓨터 통신 시스템의 도입이 필요했으며, 또 하나의 이유는 비행 시뮬레이션에 참여하는 조직 및 회사의 중요한 정보는 비공개되어야 하기 때문이었다. 그러나 이러한 분산 시스템을 구축하기 위해서는 디버깅/테스팅/수정등이 개발 비용을 초과할 수 있다.

가. Flight Simulation의 기능적 요구 사항들

시뮬레이션은 통상 Tightly Coupled Multiprocessor상에서 구현되고 효율성과 스케쥴링 관심 사항들을 처리해야 한다. 이중에는 감각에 따라 상이한 갱신률에서 작동하는 여러 서브 시스템들에서 발생하는 모든 이벤트들의 처리가 필요하며(Temporal

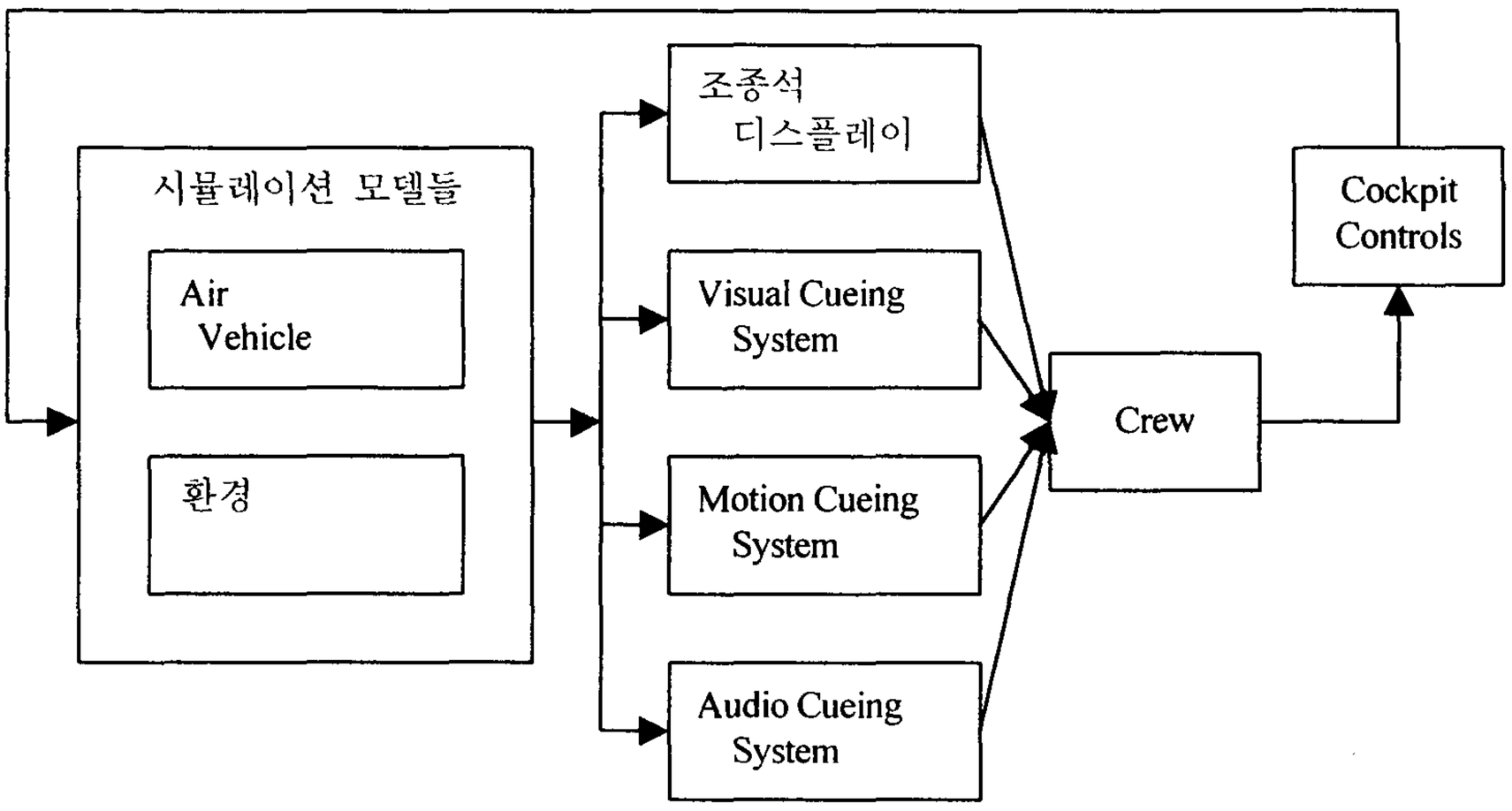
Coordination), 또한 현실 세계를 모사하여야 하기 때문에 모델의 현실감(Modeling fidelity)이 중요하게 처리되어야 한다. 또 이는 두 가지 문제로서 나누어 지는데 모델링할 실세계 Air Vehicle의 특징들을 선정하고 원하는 모델 정확도를 선정하는 문제이다. 이러한 선정 요소는 생성해야 할 S/W 서브 시스템들과 구성 요소들에 영향을 준다. 이외에도 Flight Simulator가 동작할 수 있는 실험 지원 능력을 고려해야 한다. 이는 여러가지 모드들을 포함하는데 Run, Failures, Freeze, Record,.Playback, Reposition 모드 등이 있다.

나. Flight Simulator S/W 구조

1980년대 중반까지 Flight Simulator S/W 구조는 다음과 같은 사항에 의해서 특징지어 지는데, 개발언어는 FORTRAN을 기반으로 구현되었으며, 이는 대형이고도 복잡한 시스템을 구현하는데 부적합한 언어였으며, multi-process 및 multi-processor에서 공유 메모리(Shared Memory)를 통신 근간으로 하는 시뮬레이션 시스템을 구축하였다. 이러한 S/W는 저속이면서 고가의 H/W를 효율적으로 사용하기위해서 통상 최적화되었다. 여타의 S/W 기능들을 개념적/구조적 형태의 모듈화보다 Update-Rate 및 H/W의 Load-balancing을 위해서 조각내고, 모듈로 결합되었다. 이러한 모듈들은 설계 대상으로 하는 Air vehicle의 구조와 닮아 있지 않기 때문에 해당 전문가와 Simulation 전문가들이 효율적으로 시스템을 갱신하는 것이 어려웠으며, Simulation 시스템이 제공해야 하는 비정상적 수행(malfunction)시 시스템 시뮬레이션 코드를 용이하게 삽입하기가 어려웠다.

Flight Simulator를 운영하는 모드는 Freeze, Run, Reposition모드가 있는데 Run모드에서의 수행 능력을 높이기 위해서 복잡한 코드를 생성하였으며, 비 실행모드(Freeze,Reposition)에서는 낮은 현실감을 가졌다.

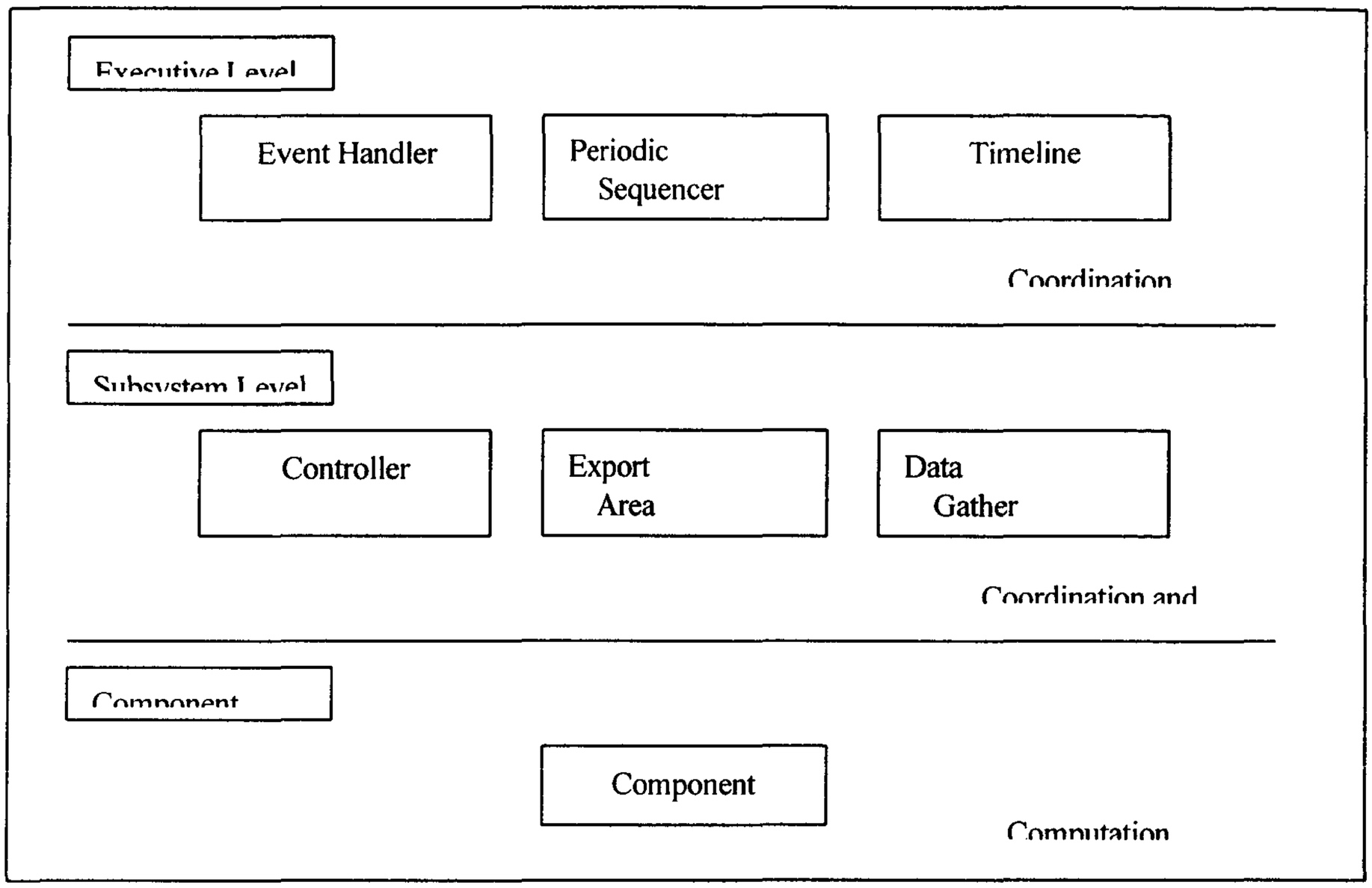
현재의 시뮬레이션 S/W를 구성하고 설계하는 기법으로 시뮬레이션의 대상 모델의 복잡성을 관리하기 위해서는 Object-oriented design과 Real time scheduling을 디자인 정책을 사용하여 복잡도를 해결하며, 설계대상 System을 구성하는 S/W 객체 유형들의 단순화(Simplicity)와 유사화(Similarity)를 통해서 최소의 객체 유형을 설계한다. 마찬가지로 부 시스템들 간의 협조 전략(Cooperation Strategy)도 마찬가지도 최소화하는 수준에서 시스템을 적절하게 부 시스템들로 구성한다.



<그림 3> Flight Simulator의 구성도

이런 방식으로 설계된 Flight Simulator는 Air Vehicle과 가상환경으로 구성되는 시뮬레이션 모델과 이를 근거로 생성되는 시각,촉감,운동감 큐를 생성하는 서브 시스템으로 구성되어져 있으며, 특히 운전자의 시각 시스템을 실제와 비슷하게 구성하기 위한 조종석 디스플레이 및 현실감 있는 제어 콘솔(Cockpit Control)을 제공한다 이외에도 운전자이외의 시뮬레이션을 제어하는 운영자 및 교육자의 제어 환경도 제공되어져야 한다

이러한 부분들을 포함하는 구조적 모델링 원리로 개발된 Flight Simulator S/W Architecture는 크게 2가지 주요 관심들의 집합으로 통상 시스템의 기능이라고 인식되는 분리된 계산 활동을 하나로 결합한 Executive 와 Computation을 처리하며 전적으로 Air Vehicle의 모델링으로 구성되는 Application으로 분리할 수 있다.



<그림 4> Air Vehicle을 위한 구조적 모델의 Level들

Excutive Level은 서브 시스템들의 실시간 스케쥴링, 프로세서들 사이에 동기화, 자료 공유, 자료 무결성 및 Event 관리로서 교육자와 관리자의 제어 Event를 처리하는 부분을 담당한다 이러한 기능들은 Periodic Sequencer, Event Handler, Timeline Synchronizer등을 통해 구현된다

실시간 수행 및 전체 스케쥴링을 책임지는 Timeline Synchronizer는 Periodic Sequencer와 Event Handler에게 Data Access, Periodic Processing ,Event Processing의 제어를 제공하고 타 프로세스들과 동기화를 유지한다 이에 의해서 Invoke되는 Periodic Sequencer는 자신의 주기적 처리명령을 통해서 서브 시스템들의 정규적인 스케쥴링을 처리한다 또한 시뮬레이션의 Run, Initialize, Freeze 모드에 대한 서브 시스템의 호출 순서를 기록한 스케쥴링 테이블을 가지고 있다 Flight Simulation 자원들의 대부분은 Periodic operation들의 계산과 주로 교육자/운영자의 제어에서 발생하는 비주기적 Event를 처리에 소모하며 이러한 부분을 Event Handler라 한다. 교육자/운영자가 발생시키는 이러한 Event들은 시뮬레이션 모드를 변경시키는데, 이러한 관점에 Event Handler는 제한적으로 시스템의 모드에 관한 정보나 Event를 처리할 반응 서브 시스템에 대한 정보를 알 필요가 있다

Subsystem Level에서의 Air Vehicle의 모델링 서브 시스템 제어기들은 계산 구성 요소들을 유기적으로 조합하여, 이러한 구성 요소들을 Air Vehicle내의 서브 시스템들을 표현하는 의미있는 집합들로 그룹화한 (Hydraulics, Air-frame Fuel braking , 엔진들 등) 구성 요소들을 보다 높은 차원의 행동 양식을 제공하기위해 각 구성요소를 Invoke 하고 관련 State Data를 전달한다.

서브 시스템 제어기들은 Event Handler에 의해서 Invoke되며 주기적 동작과 비주기적 동작을 수행한다. 서브 시스템들은 Data를 교환하기 위해 외부 교환장소(Export areas)를 사용하도록 하고, 각 서브 시스템은 정확하게 하나의 외부 교환장소에 쓸 수 있고, 다수의 외부 교환장소들로부터 읽을 수 있다. 이러한 개념은 외부 교환장소들의 액세스 책임을 개별 부속들에게 부과하지 않고 시스템 S/W의 수정 가능성과 자료 무결성을 용이하게 해준다.

Component Level에서의 구성 요소들 (reservoirs, valves, turbines, actuators등과 같은 S/W 모델들)은 Flight Simulation에서 실제 시뮬레이션 계산을 수행하는 구성요소이다. 서브시스템 제어기들에 의해 주기적 또는 비주기적 동작들을 통해 Invoke될 수 있다.

2. Iowa Driving Simulator

아이오와 대학(University of Iowa)에서는 높은 충실도(fidelity)와 현실감(realism)을 갖는 운전 경험을 표현하고, 모사차량을 운전하는 운전자에게 전체적인 감각정보(시각, 움직임, 촉각 등)를 제공하는 시뮬레이터(simulator)인 IDS(Iowa Driving Simulator)를 개발하였다[16].

IDS는 가상운전환경 시스템의 유연성(flexibility)과 재구성 능력(reconfigurability)을 유지하면서 충실도를 높이기 위하여 다음과 같은 세 가지 요소를 고려하여 운전 시뮬레이터(Driving Simulator)를 구성하였다.

운전자가 모사차량을 운전하고, 차량의 행위(BEHAVIOR)를 인식할 수 있는 감각 정보 시스템 (SENSORY CUEING SYSTEM)

6자유도를 갖는 actuator를 운용하여 운전자가 실제 차량의 움직임처럼 느낄 수 있도록 하였으며, 4 개의 채널 디스플레이 시스템으로 모사차량 돔(dome)에 가상 이미지들을 뿌려줌으로 해서 전,후방의 넓은 시야를 제공하였다. 그리고, 모사차량의 내부와 주

위에 다수의 스피커들을 설치하고 방향성 있는 오디오 정보을 제공하여 청각적인 효과를 운전자가 느낄 수 있게 하였으며 핸들과 브레이크에 포스 피드백(forced feedback)을 적용하였다.

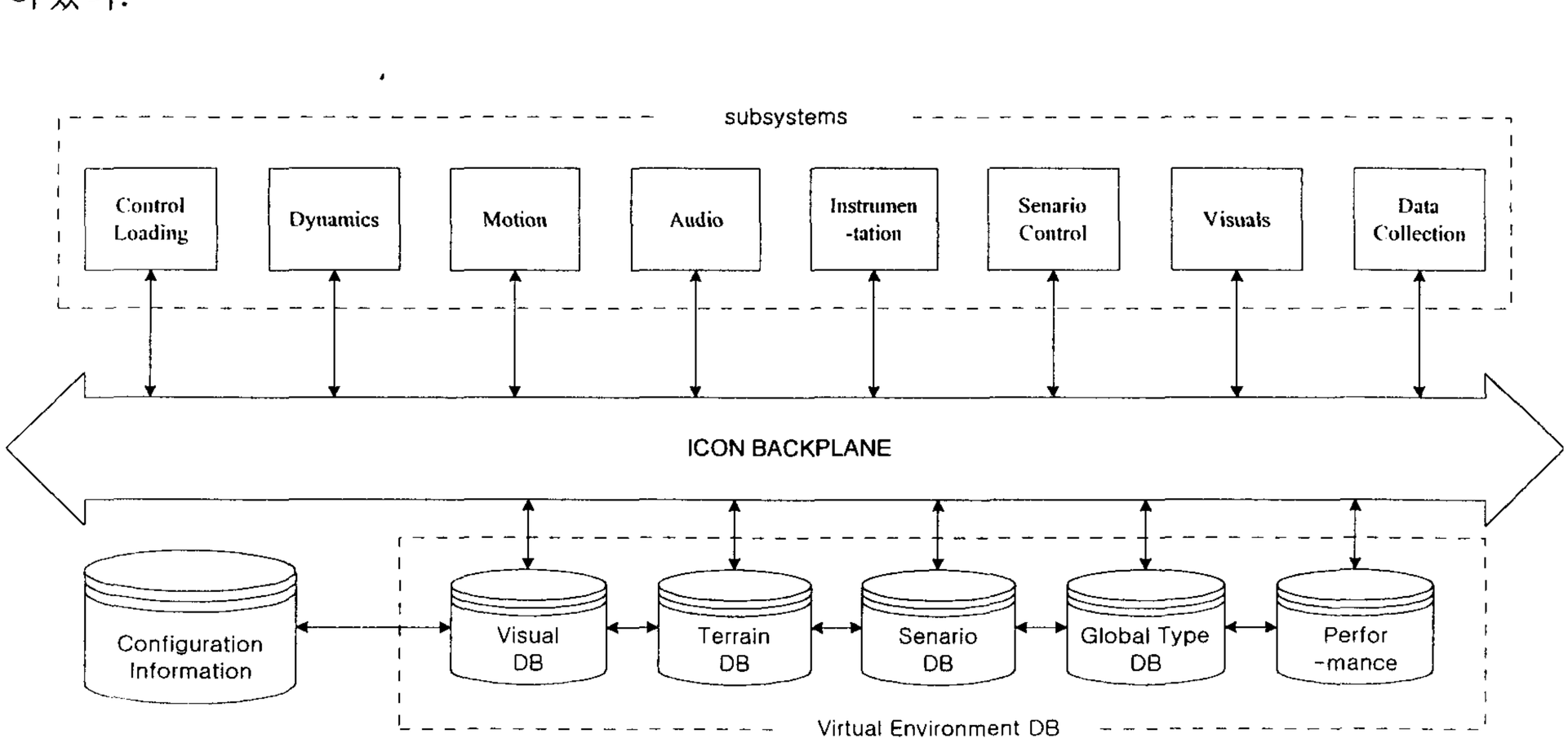
운전자의 입력에 의해 반응하는 차량 성능 모델링

운동학적 공식을 메카니즘의 조인트와 바디 구조로부터 유도하였으며, 타이어와 지표면 사이의 힘과 토크를 계산하고 이를 모델내의 휠(wheel body)에 전달하여 보다 실제감있는 모션을 생성하였다. 이들 동역학적 해석을 위해서 동 대학에서 개발한 RTRD(Real-Time Recursive Dynamics)를 이용하였고, 지표면의 정확한 표현을 위하여 지오메트리적인 요소뿐만 아니라 지표면의 재질과 같은 표면 정보도 포함하고 있다.

운전자와 상호작용 하는 외부환경

외부 세계의 정확한 표현을 위하여 근접지역의 시각정보들과, 물체, 도로, 표지판 등을 상세히 모델링 하였다.

IDS에서는 앞에서 언급된 세 가지 요소들을 제어하기 위하여 <그림 5> 과 같은 구조를 가지는 ICON(Iowa Driving Simulator Contorl)프로그램을 이용하여 가상환경을 관리하였다.



<그림 5> ICON

ICON은 크게 모듈화 된 S/W 서브시스템들과 가상환경 데이타베이스로 구성되어 있으며, 서브시스템들이 표준 인터페이스를 통하여 이들 데이터베이스를 엑세스한다.

가상환경 데이타베이스

각 서브시스템들이 가상환경에 대한 서로 다른 정보유형을 요구하기 때문에 DB를 별도로 유지하여 엑세스의 효율을 증대시켰다. 정적지형과 같은 시각 이미지 생성에 필요한 정보들을 담고있으며 차량 성능 모델링을 위하여 지표면과 자동차 타이어들 사이의 중요한 상호작용을 생성하는데 필요한 지오메트리와 표면고도, 노말, 특성자료 등의 정보 또한 가지고 있다. IDS는 지형고도와 지형 유형에 대한 질의들을 지원하기 위해 하나의 분리된 지형 DB를 사용하며 이러한 DB는 시각 DB 모델과 동시에 생성되고 자동적으로 상관관계를 가진다. 또한, 가상환경과 관련된 데이타베이스내의 VRED(Virtual Roadway Environment DB)는 도로망의 지오메트적인 요소 뿐만 아니라 지역 레이아웃을 유지하고 가상환경내에서 움직이는 모든 요소들의 정보를 가지고 있다.

서브시스템

IDS의 여러 서브시스템들은 기능적인 측면으로 분할되며, 이들간의 통신은 공용형 데이터 구조들을 통하여 이루어 진다.

운동학(Dynamic) 서브시스템은 핸들 가속기, 브레이크, 기어변속기 위치들을 감지하여 실시간으로 차량의 모션을 계산하고 모사 운전의 상태에 따라 운전자에게 포스 피드백을 제공한다. 운동(Motion) 서브시스템은 시각적 운동, 속도계나 온도계 등과 같은 계기판, 오디오 서브시스템들과 연계되어 가상환경에 대한 운전자의 인식을 제어하고, 오디오(Audio) 서브시스템은 오디오의 디지털 샘플들을 처리하여 엔진, 바람, 도로의 사운드를 재생한다. 시나리오 제어 서브시스템은 객체들의 행위들을 관리하고 가상환경의 조건들을 변경한다.

시나리오 제어 서브시스템에서 운전자의 차량과 함께 컴퓨터가 움직임을 부여하는 차량에 대한 움직임이 상태 머신(state machine)에의해 제어되는데 차량의 개수가 증가할 수록 행위에 대한 계산이 복잡해진다. 이를 해결하기 위하여 IDS에서는 차량제어를 운전자들의 의사결정과정을 모사하는 행위모델과 차량의 운동를 모사하는 물리적모델로 나누었는데, 행위모델이 물리적모델에 제어 변수들을 전달한다.

행위 모델의 초기 단계에서는 OSM(one-level state machine)을 이용하였는데 정체된 교통상황이나 교통 체증과 같은 교통현상은 자연스럽고, 반응적으로 생성이 가능하지만 차량이 많아 질수록 크기와 복잡성이 급속히 증가하여 수정이나 디버깅이 어려운 단점을 가지고 있다. 이를 해결하기 위하여 HSM(Hierarchical State Machine)과, CSM(Concurrent State Machine)들을 통합하였는데 HSM에서는 밀접한 활동들을 표현하는 상태들과 변화

들을 그룹화하여 "turn at intersection" 이라는 단일 상태 내에 캡슐화할 수 있음으로 해서 차량 행위의 프로그래밍과 수정, 디버깅을 용이하게 한다. 또, CSM은 운전 중 다수의 제약조건들과 목적들(속도제한, 빨간불에서의 정지, 충돌 회피, 위험지역 경고)을 부가함으로써 운전자가 모든 정보들을 통합하여 적절히 운전대와 가속기를 조정하게 한다. 즉, CSM으로 차량을 제어함으로써 안전한 운전에 대한 모든 요소들을 동시에 고려할 수가 있다.

3. Desktop VR: Superscape VR Toolkit

Superscape VR Toolkit은 영국의 Dimension International 사에서 IBM PC 호환 기종을 기반으로 하여 개발한 가상현실 툴킷이다[17]. 높은 수준의 상호 작용성(Interactivity)과 네트워킹을 허용하기 때문에 기존의 CAD 시스템 보다 그 이상의 기능을 제공한다. VRT는 사용하기 쉽고, 융통성 있는 프로그래밍 환경으로 Shape Editor는 가상환경을 위한 객체들의 생성을 허용하고, World Editor는 환경내 객체들과 연관이 있는 운동(Motion) 또는 행위(Behavior)를 정의하며, Visualizer는 하나의 응용을 위한 Runtime System을 형성한다.

그러나, 데스크탑(Desktop) 시스템은 고성능, 몰입형 가상환경(Virtual Environment) 시스템에서 필요로 하는 요구 사항들을 충족하지는 못한다. 현재 표준 CAD 시스템들에 의해 제공되는 사용자의 상호작용 수준 이상이 필요한 응용들을 위해서는 VRT가 매우 유용한 도구로 활용될 수 있을 것이다.

4. 분산 가상환경 시스템: dVS

영국의 Division 사가 병렬처리 시스템들을 위해 개발한 분산 가상환경 시스템(Distributed Virtual Environment System)으로 일련의 상이한 병렬 계산기 구조들에 적합하도록 설계 되었으며, Symmetric 다중 프로세서들과 단일 프로세서 시스템들의 약결합(Loosely Coupled) 네트웍들을 지원한다[18]. 디비젼은 가상환경 엑터들의 아이디어를 개발하였으며, 엑터들은 Discrete Process들의 집합이다. dVS는 통상적으로 일련의 엑터들로 구성되며, 각 엑터는 시스템의 상이한 기능들을 책임진다.

엑터는 가상환경 데이터베이스와 상호작용하는 단일 프로세스 또는 프로세스들의 집합으로 기존 다중 프로세서 병렬 시스템에서 엑터와 프로세스 사이에 차이점은 불분

명하다. 엑터는 가상환경의 한 구성요소를 모사하기 위해서 또는 실제환경에 대한 인터페이스로 작동하기 위해서 생성된다. 각 엑터는 특정 업무를 수행하기 위해 자유로이 생성되거나 삭제될 수 있으며, 한정적인 생명을 가질 수 있다.

가상환경에서 엑터들은 협업하고 동일한 글로벌 데이터들을 공유한다. dVS에서 자료의 일관성 관리는 사용자로부터 독립시켜 개발 업무를 용이하게 한다. 엑터에 의해 조작되는 객체(Element)는 하나의 추상적인 데이터 유형(Abstract Data Type)이다. 즉, 객체는 하나의 데이터 유형에 불과하며, 어떤 실제 객체도 표현하지 않는다.

하나의 주어진 객체의 인스턴스(Instance)를 'Parcel"이라 부르고, 기본적으로 여러 엑터들에 의해 공용되는 데이터의 팩케지이다. 어떤 엑터라도 하나의 Parcel을 참조할 수 있고, 또한 가질 수 있으며, 이는 자신의 Parcel Local 복사본을 가질 수 있다는 의미이다. 각 엑터는 다른 엑터들로부터 비동기적으로 운영될 수 있으며, 이는 각 Parcel이 상이한 상태에 있을 수 있다는 의미를 가진다. 엑터는 Parcel의 글로벌 상태를 변경하기 위한 요청을 개시할 수 있으며, 이는 Parcel을 잡고 있는 모든 엑터들에게 전파되어야만 한다.

한편, 비결정형 전송 메커니즘에서 모든 엑터들에게 Parcel의 갱신을 전달하는데 얼마나 오랜 시간이 소요될지 예측하는 것이 불가능하다. 그러므로 각 엑터는 자신의 시간영역에서 운영되고, 다른 엑터들과 동기화 되지 않는다. 또한, 엑터들은 환경 데이터베이스에 연결된 다른 엑터들의 존재나 위치 등을 인식하지 않는다. 엑터들은 동일한 물리적 환경 데이터베이스에 연결될 수 있고, 데이터베이스 인터페이스는 국부적으로 Parcel에 대해 배타적이며, 일반적인 모드의 엑세스들을 중재해 준다.

dVS에서 제공되는 에이젼트(Agent)라 불리우는 특수한 엑터는 원거리 가상환경 데이터베이스들 사이에 Parcel 갱신들을 중재한다. 에이젼트의 역할은 Parcel 갱신들과 엑세스들을 전달하기 위해 다른 에이젼트들과 통신시 데이터베이스 인터페이스를 사용하여 수신 에이젼트로부터 원거리 엑터들에게 정보를 전달한다. 모든 환경 데이터베이스 위치에는 단일 에이젼트가 제시되며, 임의의 데이터베이스 위상내에서 단일 에이젼트가 마스터로 지정되고, 이를 Director라고 한다. Director는 다른 에이젼트와 동일하게 동작하고, 기존 데이터베이스 위상에 조인(Join)하는 다른 데이터베이스들을 위해 잘 알려진 연결 포인트를 제공한다.

한편, Director는 다른 데이터베이스 노드들과 데이터베이스 내에 존재하는 환경들에 관한 글로벌 정보를 가지고 있다. 환경 데이터베이스는 객체(Element)들과 엑터들의

참여에 의해 생성된 Parcel들의 집합들로 구성되고, 객체(Element)들은 동적이며, 실행시간에 엑터들에 의해 생성된다.

다른 가상환경 시스템들과 다르게 사용자에게 프로그래밍 언어 수준에서 dVS 응용들을 프로그래밍 할 수 있다. 그러나, 고 수준 애니메이션 서버(AMAZE)가 가용하여 코드작성 없이 가상환경 시뮬레이션을 허용한다.

5. VPL RB2 System (Reality Built for Two)

RB2는 평가목적을 위해 매우 신속하게 가상환경을 구축할 수 있게 하는 완전한 가상환경 프로토타이핑 시스템(VE Prototyping System)이다[19]. 완전한 시스템은 다양한 실리콘 그래픽스 컴퓨터들에서 실행될 수 있는 통합된 소프트웨어 도구들의 세트로 구성된다. 어떤 의미에서 RB2 시스템은 최초의 실용적인 완전한 가상환경 시스템 이었다. RB2 시스템은 가상환경의 생성, 개발 및 조사 등을 위해 완전한 지원환경을 제공한다.

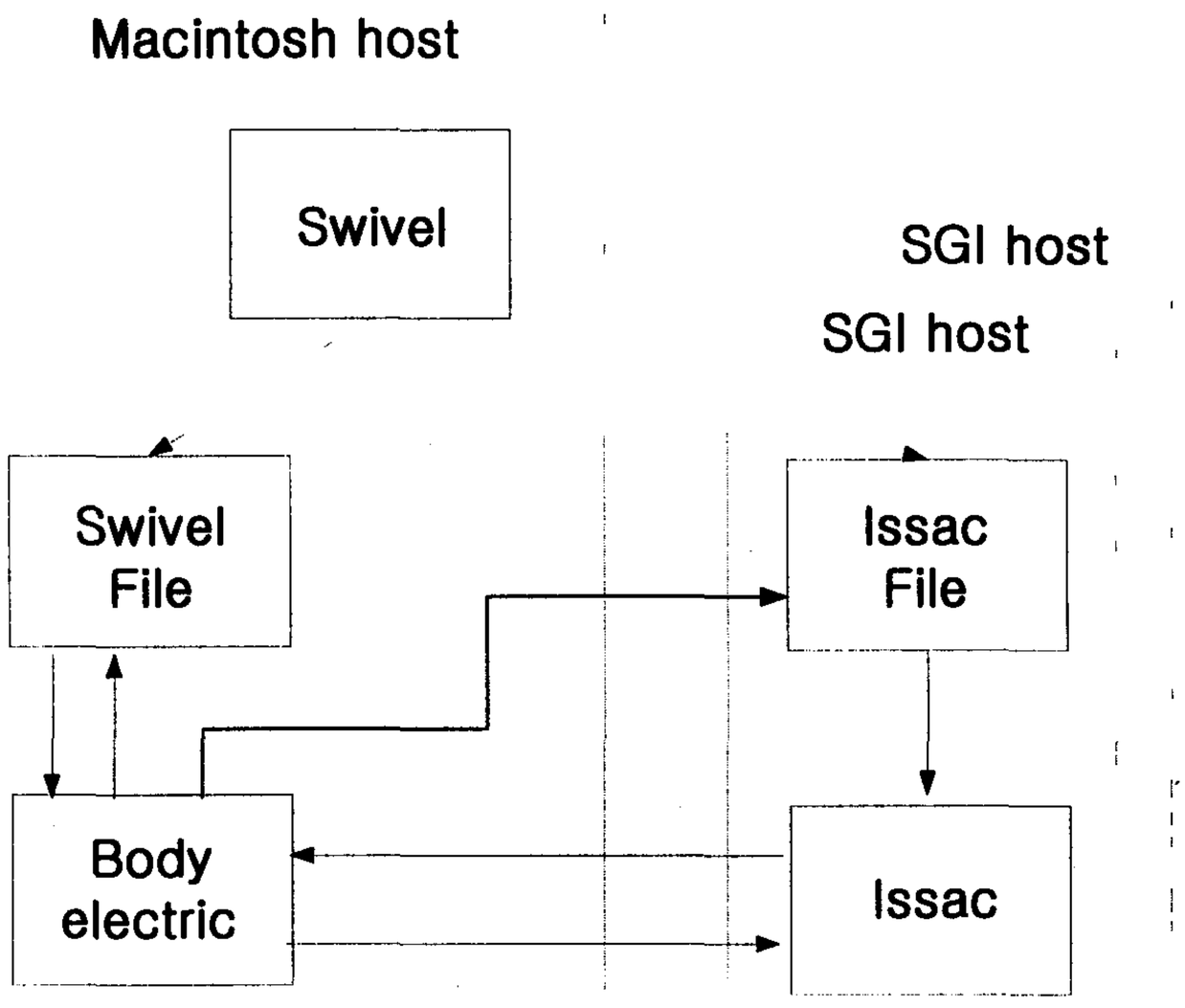
RB2 시스템 성공의 핵심은 소프트웨어 개발도구 집합을 밀접하게 통합된 성질을 가졌다는 것이다. 이는 임의의 소프트웨어 작성 없이 고도로 통합된 수준을 갖는 보다 복잡한 가상세계를 생성하는 것이 가능했다는 것이다. 사용자는 가상세계에 존재하는 객체들을 정교한 CAD-like 도구들을 사용해 생성하고, 그들의 상호작용과 행위를 고려하는 방식으로 이러한 객체들을 함께 연결(Link) 한다. RB2 시스템은 이러한 방식으로 프로그래머가 아닌 사람들이 복잡한 가상세계를 구축할 수 있도록 지원한다.

그러나, RB2 접근법은 제한된 성능을 갖는다는 단점을 가지고 있으며, 이는 갱신시간을 증가시킨다는 사실이다. Swivel 3-D Professional은 정교한 3D 모델링 프로그램으로 3D Drawing, 모델링, 그리고 매킨토시에서 사진처럼 실감있는 렌더링 오퍼레이션을 허용하며, 강력한 CAD-like Editor의 모든 특성들을 가지고 있다. 즉, 간단한 객체들을 묶어서 복잡한 객체들을 형성하는 능력을 가지고 있다는 것이다.

또한, 다양한 렌더링 모델들(wire-frame, hidden time, outline shade, shade and smooth)을 가지고 있으며, 강력한 렌더링 능력을 보유하고 있다. 그리고, 개별 Lighting 강도들을 설정할 수 있고, Fill Light 능력을 갖는 최대 8가지 Light Source들이 가용하다. 한편, 렌더링 능력을 보상하기 위한 Shadow Effects등과 같은 추가적인 특징들을 갖고 있다. 또한, RB2는 설계자에게 Material의 direct reflectance 또는 specular reflectance 변경을 허용하는 설비

들을 제공하며, 모든 유형의 Material들을 모사할 수 있다. Object Intersection들과 Edge effects의 Anti-Aliasing 등과 같은 진보된 특징들도 제공한다. 그리고, 가상환경 객체 설계자에게 움직이는 객체들을 시각화해 주는 애니메이션 기능을 제공한다.

Body Electric은 VPL RB2 시스템의 심장으로 가상환경의 객체들을 연결해 주고, 행위 및 상호작용 규칙들을 시스템에 삽입하는 것을 허용한다. 하나의 가상환경에서 일련의 입력 장치들로부터 일련의 출력 장치들로 자료의 흐름을 제어하는 Open Ended Control System이다. 사용자는 특정한 응용을 위해 요구되는 출력을 생산하기 위해 Body Electric의 특징들을 사용하는 데이터를 처리하는 데이터 플로우 네트웍(Data Flow Network)를 생성할 수 있다. Body Electric은 VPL DataGlove, VPL DataSuit, Polhemus 3Space Trackers 등과 같은 장치들로부터 입력을 받아 들이며, 입력자료는 가상환경내 객체들의 위치와 방향을 제어하기 위하여 다양한 수학적, 논리적 오퍼레이션들을 사용하여 처리될 수 있다. 처리된 또는 처리되지 않은 자료는 플레리백(Playback) 또는 사후처리를 위해 디스크에 저장될 수 있다. 그리고, 실시간 데이터는 이더넷을 통해 호스트 컴퓨터들로 전송될 수 있다. Body Electric의 데이터 플로우 언어를 사용해 사용자는 가상환경내 객체들의 행위를 기술할 수 있고, Body Electric의 자료기록과 계층적 자료모델을 가지고 인간모션을 포착할 수 있으며, 정교한 컴퓨터 그레픽스 애니메이션을 랜더링 하는데 사용할 수 있다.



<그림 6> VPL RB2 Software Configuration

<그림 6>에서 보는 바와 같이 RB2 시스템의 소프트웨어 구성(Configuration)은 Swivel, Swivel File, Body Electric(이상 매킨토시), Issac File, Issac(이상 SGI 호스트) 등이 있다. Swivel 은 소프트웨어 CAD 팩케지로 객체들을 생성하고, 가상환경을 모델링 하는데 사용한다. Body Electric은 타 객체들과의 관계성에 따라 계층적 트리를 형성하기 위하여 각 객체를 위한 포인트를 추출한다. Isaac은 포인트들과 그들의 링크들 뿐만 아니라 Swivel로부터 각 객체의 실제 형체와 칼라를 사용한다. Body Electric은 실시간으로 트리에서 포인트들의 위치와 방향을 읽고 수정하며, 수정된 내용은 Isaac에 보내지고, Body Electric으로부터 수신된 갱신 정보에 따라 각 객체를 렌더링 한다.

Body Electric의 내부에서는 각 입력장치가 정수 값을 Raw Data Array에 주며, 이 Array는 가장 최근에 획득한 자료 값을 가지고 있다. 이러한 값들의 각각을 읽음으로써 데이터 플로우 네트웍에 마사지(Massaged) 될 수 있다. 여기서 사용자는 네트웍내 데이터 마사지 모듈들을 사용함으로써 스케일링, Offsetting, 필터링(Filtering) 및 다른 많은 오퍼레이션들을 수행할 수 있다. 마사지된 값들은 트리에서 포인트들의 위치와 방향을 설정하는데 사용되며, 갱신된 포인트들은 EyePhone에 렌더링 되기 위해 Isaac에 전달되거나 상이한 컴퓨터 상에 다른 프로그램들에 의해 사용될 수 있다.

레코딩 데이터(Recording Data)는 플래이백 또는 사후처리를 위해 사용되며, Raw 데이터 레코더는 가공전 직접 Raw Data Array의 값을 기록하고, 현실세계의 연속적인 이벤트들을 포착할 수 있도록 한다. 마사지 데이터 레코더는 외부 하드웨어로부터 수신된 Raw Data 값들에 대해 데이터 플로우 네트웍의 반응을 반복할 필요가 있다. Body Electric은 이를 트리 렌더링의 연속들을 기록함으로써 달성한다. Body Electric이 마사지 데이터들을 기록할 때 마사지 데이터 레코더는 로컬 트리 좌표들로부터 입력을 취한다.

플래이백에서 데이터 플로우 네트웍은 트리로부터 단절되고, 마사지 데이터 레코더의 출력들이 트리를 애니메이션 한다.

6. Virtual Environment Operating Shell(VEOS)

VEOS는 분산 가상환경 응용들을 프로토타이핑하기 위한 소프트웨어 환경이며, VEOS의 하부 설계목표는 분산설계에서 임무들을 정의하고, 계산 자원들을 할당하기 위한 하나의 메커니즘을 생성하는데 있었다. VEOS 응용은 가능한 한 스스로 신뢰적인(Self-reliant) 차별화 되는 프로세스들로 분할될 수 있다[20]. 각 프로세스는 시스템에서 다른 프로세스들과 통신을 위한 Provision들을 가지고 분리적으로 코드화 될 수 있으며, 네트웍을 엑세스할 수 있는 임의의 UNIX 워크스테이션에서 단일 VEOS 엔티티로 구현된다.

VEOS는 프로세스간 메시지 전달과 내용 Addressable Database 엑세스를 제공한다. 이는 하드웨어 자원들이 물리적으로 밀착되어 있지 않거나, 기계 종속적인 자원들이 그들의 플랫폼 EOANS에 고립화 되는 응용들을 위해서는 이상적이다. 또한, UNIX 프로세스들에 개괄적으로 대응되는 Heavyweight Sequential 프로세스들을 사용하기 때문에 조잡한 수준(Coarse Grain)의 병렬성을 운영하는 프로토타이핑 프로그램들을 위해 이상적이다. 이러한 방식으로 VEOS는 Virtual Multiprocessor로 워크스테이션들의 네트웍을 사용하는데 사용할 수 있다.

C 프로그래머들은 XLISP로부터 엑세스 할 수 있고, 즉시 다른 VEOS 도구들과 호환성이 있는 Custom VEOS 도구들을 구축할 수 있다. LISP 프로그래머들은 다양한 하드웨어 및 소프트웨어 자원들을 사용하는 분산 프로그램들을 신속하게 설계하고, 실행할 수 있다. VEOS 운영체제가 아니며, 분산 응용들을 프로토타이핑 하기 위한 사용자 수준

의 Framework 이다. 주요 목표는 융통성과 사용의 용이성이며, 이러한 설계는 실시간 성능의 값비싼 것으로부터 나왔다.

그러나, HILT는 VEOS가 적절한 응용 구조화와 튜닝을 통해 좋은 성능을 가질 수 있다고 주장하고 있으며, VEOS는 단지 대부분 공통 UNIX 인터페이스에만 의존하기 때문에 플랫폼에 독립적이다. VEOS의 기본 사항들로는 VEOS에서 프로그래머는 제안된 가상환경의 많은 계산적인 업무들을 달성하기 위한 방법을 정의할 수 있다. VEOS는 프로그래머에게 이러한 업무들을 분산 시스템의 계산적 노드들로 덩어리화 할 수 있도록 허용하고, 이러한 덩어리들을 엔티티들이라 한다. 각 엔티티는 다음과 같은 기본(Native) 능력들-코드화된 업무기술의 해석, 엔티티간 통신, 일반화된 자료관리, 로컬 데이터 공간 상에서 일반적인 패턴 매칭-을 가지고 명확한 Unix 프로세스로 구현된다.

VEOS를 가지고 설계자는 각각이 동일 또는 상이한 UNIX 기반 기계들에 상주하는 엔티티들의 집합을 사용하는 가상환경을 구현할 수 있다. VEOS가 단지 일반적인 UNIX 인터페이스에 의존하기 때문에 하나의 가상환경 응용은 다중 하드웨어 플랫폼들을 사용할 수 있다.

VEOS 엔티티는 자료관리, 프로세스 관리, 엔티티간 통신 등과 같은 VEOS Native 능력들을 갖는 Stand-Alone 실행가능 프로그램이다. 독립적으로 상기와 같은 필요성들은 일부 기존에 존재하는 방법론들로 해결할 수 있다. 예로, LISP은 일반화된 데이터 추상화(Data Abstraction: 리스트들)과 유용한 프로그램 제어를 제공한다. Mathematica는 일반화된 사양양식과 자료와 기능의 일관성 있는 처리 등을 제공하고, Linda는 일반화된 자료관리를 제공한다. 그러나, 이러한 도구들은 공통형식을 공용하지 못할 뿐만 아니라, 이러한 시스템들의 큰 구성 요소들은 대부분의 상황에서 불필요한 것들 이다. 또한, 이러한 팩케지들을 통합하는 것은 비용적, 시간적으로 낭비가 있다. 궁극적으로 VEOS의 일반적인 형식으로 튜플(Tuple)을 선정하여 향상시키는 것으로 결정하였고, 이를 Grouple라 칭하였다. Grouple의 서브 필드들을 Element 라고 하며, 중첩(Nested)될 수 있으며, 하나의 Element도 하나의 Grouple이 될 수 있다. VEOS의 데이터 관리자인 Nancy는 표준 데이터 포멧으로 Grouple을 사용하고 있다.

VEOS 커널의 구성요소에는 크게 VEOS Grouple 관리자인 Nancy, VEOS 통신 모듈인 Talk, 그리고 VEOS 커널 제어 모듈인 Shell 등이 있다. Nancy는 "Home-Brew" 데이터베이스 관리자로 VEOS내 모든 Grouple 조작을 수행한다. 여기에는 생성, 파괴, 삽입, 복

사 등등이 포함되며, Nancy는 하나의 Grouplespace에서 자료탐색, 삽입, 대치를 위한 강력한 문법을 제공한다. 이것은 패턴매쳐로 매치/대치/실행 엔진 구현을 위한 완전한 Primitive Set을 제공한다.

VEOS의 통신 모듈인 Talk는 엔티티 통신을 위한 단지 지원된 메커니즘이다. VEOS 엔티티들은 UNIX 프로세스들을 통해 조잡한 수준(Coarse Grain)의 병렬성을 제공하기 때문에 프로세스(엔티티) 동기화 및 공용 메모리는 실질적이지 못하다. 그러므로 VEOS는 엔티티 통신 수단으로 단지 메시지 전달을 지원하며, 두가지 단순한 메시지 전달 프리미티브(Primitive: vthrow, vcatch)을 제공한다. 메시지를 다른 엔티티에게 비동기적으로 전송하고, 수신 엔티티는 또한 비동기적으로 메시지를 받기 위해 놀고 있다. 비동기식 메시지 전송은 하나의 엔티티가 메시지를 전송할 때 오퍼레이션은 신뢰적이고 수신측은 메시지를 위해 기다리고 있거나 또는 그렇지 않을 수 있다. 그리고, 엔티티 메시지의 도착을 무한정 기다릴 필요 없이 진입하는 메시지들을 항상 체크할 수 있다. 이를 또한 비 블로킹(Non-Blocking) 통신이라 부르며, Talk는 엔티티들 사이의 메시지를 평면 선형화(Flat Linearized) Grouple 형식으로 전송한다.

VEOS 커널의 제어 모듈인 Shell은 커널의 관리작업 수행하며, 초기화, 인터럽트 처리, 커널 메모리 관리 등의 일을 수행한다.

7. Minimal Reality(MR)

가상환경과 3D 사용자 인터페이스들의 다른 형태들을 개발하는 것을 지원하는 서브루틴 라이브러리이다[21]. 가상환경 시스템에서 일반적으로 사용되는 여러 장치들을 지원- Polhemus space tracker, VPL DataGlove, VPL Eyephone, Sound Synthesis Equipment 등- 한다.

또한, MR Toolkit은 다중 워크스테이션들 상에 사용자 인터페이스들의 분산, 여러 워크스테이션들 상에 자료 분산, 다양한 상호작용 기법들, 실시간 성능분석 도구들 등을 지원한다. MR Toolkit은 실리콘 그래픽스와 DEC 워크스테이션에서 C 프로그램으로부터 호출된다. 그러므로 LISP기반 VEOS Toolkit의 흥미있는 대안으로 제시되고 있다. 프로그래머가 가상환경 사용자 인터페이스를 생산하는데 필요한 기본 서비스들을 제공한다.

그러나, 가상환경 사용자 인터페이스들을 구축하기 위한 High-level 도구들을 제공하지 못하므로 프로그래머는 특정한 업무를 수행할 일부 코드를 작성해야 한다. MR Toolkit은 경험상 가상환경 응용들을 구축하기 위한 Toolkit이 가져야 하는 특징들을 다음과 같이 정의하고 있다. 첫째, 효율성(Efficiency)으로 가상환경 사용자 인터페이스의 질은 사용자에게 제시된 이미지에서 사용자의 동작을 반영하기 위한 실시간 요구에 좌우된다. 즉, 자연스럽고 만족스러운 상호작용을 위해 동작과 이미지 사이에 지연이 최소화 되어야 하며, Toolkit은 지연을 줄이기 위해 가능한 한 효율적이어야 한다.

둘째, 융통성(Flexibility)으로 가상환경은 새로운 상호작용 스타일로 새로운 분야라서 매우 빠르게 변화하고 있다. 그러므로 가상환경을 위한 Toolkit은 하드웨어 및 소프트웨어에서 변화를 수용할 수 있도록 충분히 융통적이어야 하며, 새로운 장치들의 추가 용이 및 새로운 유형의 상호작용 기법들을 제공해야 한다.

셋째, 분산 응용들 지원(Distributed Applications Support)으로 가상환경 사용자 인터페이스들의 대부분은 여러 대의 워크스테이션들 또는 컴퓨터들의 협동을 필요로 하도록 구축한다. 통상, HMD를 위한 이미지들을 생산하기 위하여 2개의 워크스테이션이 사용된다. 그러나, 프로그래머는 각 응용에서 사용자 인터페이스의 분산 측면들을 다루기 위하여 요구되는 프로그램 코드의 생산을 원치 않으므로 이는 Toolkit에서 처리되어야 한다.

넷째, 시간관리 지원(Time Management Support)으로 가상환경 사용자 인터페이스는 사용자 동작들과 HMD상의 그들의 피드백 사이에 지연이 최소가 되어야 한다.

한편, MR Toolkit은 소프트웨어의 3개 레이어(Layer)로 구성된다. 즉, Bottom 레이어는 팩케지들을 지원하는 장치들의 집합으로 구성되고, 각 팩케지는 하나의 장치를 지원하며, 팩케지는 Client/Server 짝으로 보여진다. Client 부분은 서버와 인터페이스 하기 위한 라이브러리 루틴들의 집합인 반면에 서버부분은 연속적으로 장치를 샘플링하고 필터링과 같은 저수준 오퍼레이션들을 수행한다. 서버가 연속적으로 장치를 샘플링함으로써 클라이언트 루틴들은 장치에게 샘플링을 요청할 때 포함되는 지연 없이 신속하게 가장 최신의 값을 가질 수 있다. Client/Server 모델은 시스템에 새로운 장치들의 추가를 용이하게 하고, 처리부하를 워크스테이션들에게 분산시킨다.

두번째 레이어는 장치로부터 들어 온 데이터를 처리하고, 이들을 사용자 인터페이스 프로그래머에게 더욱 편리한 형식로 전환한다. 워크스테이션과 작업공간 매핑 사이

에 자료전송과 같은 표준 서비스들을 제공한다. DataGlove와 같은 경우, 이 레이어의 루틴들은 제스쳐 인식을 수행하고, 현 글로브 상태의 시각적인 피드백을 제공한다.

세번째 툽 레이어는 프로그래머를 위한 팩케지화 된 서비스들의 집합을 제공하며, 이러한 서비스들은 평균적인 가상환경 사용자 인터페이스의 요구 사항들에 기반을 두고 있다. 사용자 인터페이스에 의해 요구되는 모든 장치들을 초기화 하는 단일 루틴이 존재한다. 그러므로, 프로그래머는 각 장치를 위한 초기화 루틴들을 기억할 필요가 없다. 또한, 이 레이어의 루틴들은 양안시현을 생성하는데 사용되는 워크스테이션들 사이의 자료 구조들과 시현 오퍼레이션들을 동기화해 준다.

MR Toolkit 응용에서 사용자 프로그램이 실행할 수 있는 3가지 역할로는 먼저 하나의 프로그램은 응용을 제어하고, 다른 프로그램들의 실행 초기화를 위한 역할을 담당한다. 하나의 응용에는 하나의 마스터 프로그램이 존재하고, 여러 개의 슬래브 프로그램이 존재할 수 있다는 것이다. 두번째는 그래픽 출력 생산에 사용되는 슬래브 프로그램은 응용의 다른 계산들에는 참여하지 않는다. 통신 프로그램은 응용에 필요한 계산들을 수행하며, 통상 마스터 프로그램과 동일한 LAN 연결된 Compute Server상에서 실행된다. 그리고, 세번째 계산 프로그램들은 그들의 입력을 마스터 프로그램으로부터 수신하고, 주기적으로 그 결과를 마스터 프로그램에게 보고한다.

MR Toolkit 구성 색션은 MR Toolkit을 초기화하고 응용을 위한 구성 파일을 처리한다. 응용의 프로그램들 사이에 공용되는 자료 구조들은 프로그램의 구성 색션에서 식별될 수 있으며, MR Toolkit은 적절한 프로그램 사이에 이러한 자료 구조들의 전송을 처리한다. MR Toolkit은 자료공용을 위해 단순한 모델을 사용한다. 즉, 마스터 프로그램은 모든 공용 자료 구조들의 복사본을 가지는 것으로 간주한다. 또한, MR Toolkit에서는 자료구조는 단지 응용의 2개 프로그램에 의해서만 공용되는 것으로 간주한다. 그러므로, 각 공용 자료구조는 마스터 프로그램에 하나의 복사본이 있고, 다른 복사본은 슬래브 또는 계산 프로그램에 있다.

MR Toolkit은 이들 프로그램중 하나가 자료를 생산하고 다른 하나가 이를 소모하는 것으로 간주한다. 즉, 프로그램들 사이에 한 방향 자료전송이 있고, 이렇게 함으로써 대부분의 자료공용 설정이 용이하며, 보다 복잡한 프로세스 구조들을 생산하는 것이 가능하다.

MR Toolkit의 계산 색션은 응용에서 프로그램에 할당된 업무를 수행한다. 슬래브 프로그램인 경우, 마스터 프로그램으로부터 들어오는 정보 패켓들을 기다리고, 다음 이미지들을 생성한다. 계산 프로그램인 경우는 다음 결과의 집합을 생산하고, 이를 마스터 프로그램으로 전송한 다음, 다음 인스트럭션을 기다린다. 프로그램의 계산 색션은 2개 부분으로 나누어 지는데 하나는 응용에 의해 사용될 환경을 설정하는 것이고, 다른 하나는 사용자와의 상호작용을 처리하는 부분이다. 사용자와 상호작용 전 작업으로는 응용에서 사용할 좌표체계 구축(기본:장비위치 방 기반 좌표체계, Work Space Mapping 팩케지를 통해 설정 가능), 사용장비 조정 및 초기화 등을 수행한다.

팩케지 수준에서는 대부분의 가상환경 프로그램들에 의해 요구되는 장치 및 서비스들 중에서 일부에 대한 고 수준 인터페이스를 제공한다. Isotrak 팩케지는 Stand-alone Polhemus Isotrak 디지타이져 사용을 지원하고, 디스플래이 팩케지는 현재 디스플레이 장치의 3가지 카테고리(HMD:EyePhone, Mono:하나의 프로그램으로 이미지 생성, No Display)를 지원한다. DataGlove 팩케지는 데이터글로브를 연결하는데 사용된다. Workspace Mapping 팩케지는 가상환경 사용자 인터페이스에서 장치 좌표들을 환경 좌표들로 매핑시키는 데 사용한다. 즉, Device 좌표는 장치자체의 좌표체계이고, Room 좌표는 장치와 사용자를 갖고 있는 룸에 기반을 둔 좌표체계로 모든 장치 좌표는 일관성 있는 하나의 좌표체계를 위해 룸좌표로 변환이 가능하다. 그리고, 환경 좌표체계는 사용자 인터페이스 내에서 사용되는 좌표이며, 룸 좌표에서 환경 좌표로의 매핑은 개별적인 사용자 인터페이스에 의존한다.

Work Space 파일은 작업 공간내 입출력 장치들의 현 구성을 기록하는데 그들의 단위에 따라 그들 좌표 체계들의 위치와 방향을 포함한다. 프로그램 인터페이스는 룸 좌표체계에서 환경 좌표체계로 매핑해 주는데 환경에서 사용될 단위를 정의하고, 룸 좌표체계에 대한 환경 좌표체계의 방향과 환경 좌표체계 내에서 룸 좌표체계의 원점 위치 등을 정의해 준다.

자료공용 팩케지는 MR Toolkit 응용을 구성하는 프로그램들 사이에 자료 전환을 위해 사용된다. 예로 계산결과를 표현하는 자료구조는 마스터 프로그램과 결과를 생성하는 계산 프로그램에 의해 사용된다. 그러므로 각 프로그램은 데이터에 대한 자신의 복사본을 가지고, 다른 프로그램의 자료에 영향을 주지 않고 조작이 가능하다. 프로그램들의 핵심지점에서 자료구조는 동기화 되는데 즉, 한 프로그램의 자료구조에서 자신 버전의 내용들을 다른 프로그램에 전송하게 된다.

계산 프로그램은 자신의 자료구조에 있는 최신 자료들을 계산의 각 시간단계에서 마스터 프로그램에게 전송한다. 이러한 과정은 모두 사용자에게 투명하고, 비동기식으로 이루어 진다. 동기식 자료 구조인 경우, 수신 프로그램이 자료 구조가 갱신될 때 제어권을 가지고, 비동기 자료 구조인 경우, 수신 프로그램은 공용 자료구조 갱신에 대해 제어권을 갖지 않는다.

사운드 팩케지는 사운드 효과를 생성하고, 사운드 생성에 단순한 장치독립 인터페이스를 제공한다. 사운드 에디터는 사운드 팩케지에 대한 프로그램 인터페이스로 사운드 서버에 연결하고, 사전 계산된 사운드들을 플래이하며, 사운드 샘플들을 생산한다.

8. WorldToolKit(WTK) Sense8 Corporation

Sense8 사에서 개발한 가상환경 응용 개발을 위한 툴킷으로 기본 레벨에서 보면 C 언어로 쓰여진 400개 이상의 함수들의 집합이다[22]. 중요한 특징은 하드웨어 독립이고, 486 그래픽스 플랫폼에서 SGI 까지 다양한 플랫폼에서 수행이 가능하다. WTK-기반 시스템의 구성 요소들로는 호스트 컴퓨터, WTK 라이브러리, C 컴파일러, 3-D 모델링 팩케지, 이미지 포착 하드웨어/소프트웨어, Paint Box 와 같은 비트맵 에디팅 소프트웨어, 하드웨어 가속기 그래픽스 보오드들 및 메모리 관리 시스템 등이 있다.

WTK는 객체지향 Naming 방식의 구조로 이는 프로그래밍 수준에서 일어나는 것에 대한 가시성(Visibility) 유지가 더욱 용이하게 해 준다. 주요 클래스들로 'universe', 'object', 'polygon', 'vertex', 'path', 'sensor', 'viewpoint', 'light source', 'portal', 'animation' 등이 있으며, 이는 WTK가 객체지향 언어의 상속과 동적 바인딩을 사용하지 않음을 의미한다.

'Universe'는 최상위 클래스로 가상환경내 단지 하나의 'universe'만 Active되고, 'sensors', 'lights', 'animation sequences', 'portals', 'viewpoints', 'graphical objects', 'serial ports', 기타 등을 포함하는 객체들로부터 구축된다. 객체들은 가상환경의 구성 요소들이며, 가상환경은 여러 개의 유니버스를 포함할 수 있다. 'Universe'는 가상환경의 모든 객체들을 포함하고, WTK내 'Simulation Manager'에 의해 유지된다.

Simulation Manager는 가상환경 내의 프로세스들의 실행을 제어하기 때문에 WTK에서 가장 중요한 부분이다. 시뮬레이션 루프는 한번 또는 여러 번 수행될 수 있으며, 프로그래머는 특수한 액션 함수들을 통해 가상환경 내에 이벤트들을 제어할 수 있다. 시뮬

레이션 루프 동안에 개별 객체들은 실행될 임무 기능(함수)들을 가질 수 있다. 한편, Universe 객체로는 다음과 같은 것이 있다.

먼저 Geometric 객체는 Universe의 기본 요소들로 정적 객체들과 동적인 행위를 보이는 동적 객체들이 있다. 계층적 구조로 조직될 수 있고 타 객체와 상호작용이 가능하며, WTK는 객체의 행위를 제어하기 위해 함수적인 임무를 할당할 수 있다. (기타, 택스쳐 적용 가능, 지형객체 생성 가능, LOD 가능, 행위함수 할당 가능)

Sensor 객체는 객체의 위치와 방향 제어를 위한 외부장치에 적용하며, 감각 입력은 Universe에서 객체의 행위 측면을 수정하는데 사용되고, 관측자의 뷰 포인트 제어에도 사용된다. 각 입력장치 각각은 WTK 라이브러리내 특수한 루틴에 의해 지원되며, WTK는 실시간 성능 보장을 위해 센서 장치들을 직접적으로 비쥬얼 이미지에 연결한다.

Lights는 WTK는 가상환경의 배경을 위한 Ambient Light와 위치 및 방향 제어가 가능한 다수의 Directed Light Source들이 있다.

뷰 포인트 객체는 사용자에게 보려는 Geometry들을 정의하도록 함으로써 임의의 위치, 방향 및 뷰잉 각도 등으로부터 가상환경을 볼 수 있도록 한다. 하나의 Universe 내에 다수의 상이한 뷰 포인트들이 유지될 수 있으나, 단지 한번에 하나의 뷰 포인트만 활성화 된다. 뷰 포인트들을 센서 객체들에 붙여줌으로써 뷰 포인트를 센서의 이동에 따라 동적으로 변경할 수 있다.

Universe 에서 객체들 사이의 상호작용을 위해 충돌탐지를 통해 교차점들을 찾는다. 그런데 이는 전체 툴킷의 성능에 상당한 영향을 미칠 수 있다. 또한, 객체들의 동적인 행위와 3D 조작 업무를 다루는데 필요한 완전한 수학 함수들의 집합을 WTK는 제공한다.

다른 Universe로의 진입은 Portals을 통해 이루어 지며, WTK에서 모든 오퍼레이션들과 동적인 행위는 하나의 로컬환경(Universe)으로 제한된다. 인접한 Universe들은 함께 'Join'될 수 있고, 상이한 Universe들 사이의 이동은 Portal들을 통해 달성된다. Portal은 특별한 Polygon들에게 할당되며, 사용자의 뷰 포인트가 설정된 Polygon을 횡단하면, 인접 Universe로 진입하게 된다. 각 Universe는 하나의 분리된 엔티티로 간주되고, 자신의 객체들에 대해 상이한 규칙들이나 동적인 행위를 부여할 수 있다. 또한, 여러 개의 작은 Universe들로 하나의 큰 가상환경을 만들 수 있다.

실시간 프로그래밍 환경에서 프로그램의 전체 또는 부분의 반복회수를 판단할 수 있는 것은 매우 중요하다. 이는 최대의 성능을 달성하기 위한 응용코드의 튜닝을 가능하게 한다. 또한, WTK는 변경이나 이동하지 않은 화면은 해상도를 높이는 Adaptive Display Resolution Scheme을 제공한다.

9. Software System MultiGen

MultiGen은 소프트웨어 시스템 사에서 개발한 비쥬얼 시스템 데이터베이스를 생성하고 에디팅하는데 사용되는 상호작용 도구이다[23]. 현재 다양한 플랫폼을 위한 버전들이 존재하고, 사용자 관점에서 보면 매우 강력한 3D 모델링 도구이다. MultiGen의 장점 중에 하나는 우선 사용자가 기본적인 3D 모델링 팩케지를 구매한 다음, 전체 팩케지 구성을 특정 응용에 적합하도록 필요한 확장 도구들을 구매할 수 있다는 것이다. MultiGen은 아마 세계에서 최상의 3D 모델링 팩케지들 중에 하나로 간주되고 있으며, 비행 시뮬레이션 분야에서는 표준으로 간주되고 있다. 비록 값은 비싸지만 정교한 모델들을 쉽게 생성할 수 있는 장점이 있다. 또한, MultiGen은 가상환경에서 사용될 복잡한 모델들을 생성하는데 소요되는 시간을 쉽게 개관해 볼 수 있다.

한편, Digital Terrain Elevation Data(DTED) Option은 DMA의 DTED를 MultiGen 파일에 통합할 수 있도록 허용한다. DTED의 Elevation Spot Height Data를 MultiGen 내부형식 즉, Polygonal Data Base로 변환해 준다. 이때 Spot Data를 지형에 맞게 트라이앵글들로 변환해 주는 Delaunay 알고리즘과 1개, 2개 또는 4개 Polygon들로 구성된 정사각 지형을 생성해 주는 Polymesh 알고리즘을 사용한다.

또한, Digital Feature Analysis Data(DFAD) Conversion Option은 DFAD를 모델링 도구로 Importing 가능하도록 해 준다. 이는 설계자에게 실세계 데이터를 가상환경에 통합할 수 있도록 허용한다. DFAD는 도로, 빌딩, 강, 숲 등을 포함하기 때문에 이따금 Cultural Data라고 한다. MultiGen은 지형 데이터와 DFAD를 자동적으로 매핑해 줌으로 실제 지형을 생성하는데 매우 효율적이다.

Texture Option는 택스쳐를 이용하여 적은 수의 Polygon으로 현실감 증대하고, 높은 Throughput Rate 달성할 수 있도록 한다. 최대 16개의 택스쳐 빠레트를 하나의 데이터베이스 파일에 할당할 수 있다. 각 빠레트는 256 x 256 택셀Telxel)들을 가지며, 최대 64개

의 택스쳐 패턴들을 가질 수 있다. 하나의 택스쳐 패턴을 하나의 Polygon에 부여하기 위해서는 다수의 택샐을 다수의 픽셀에 매핑하는 것이 필요하다.

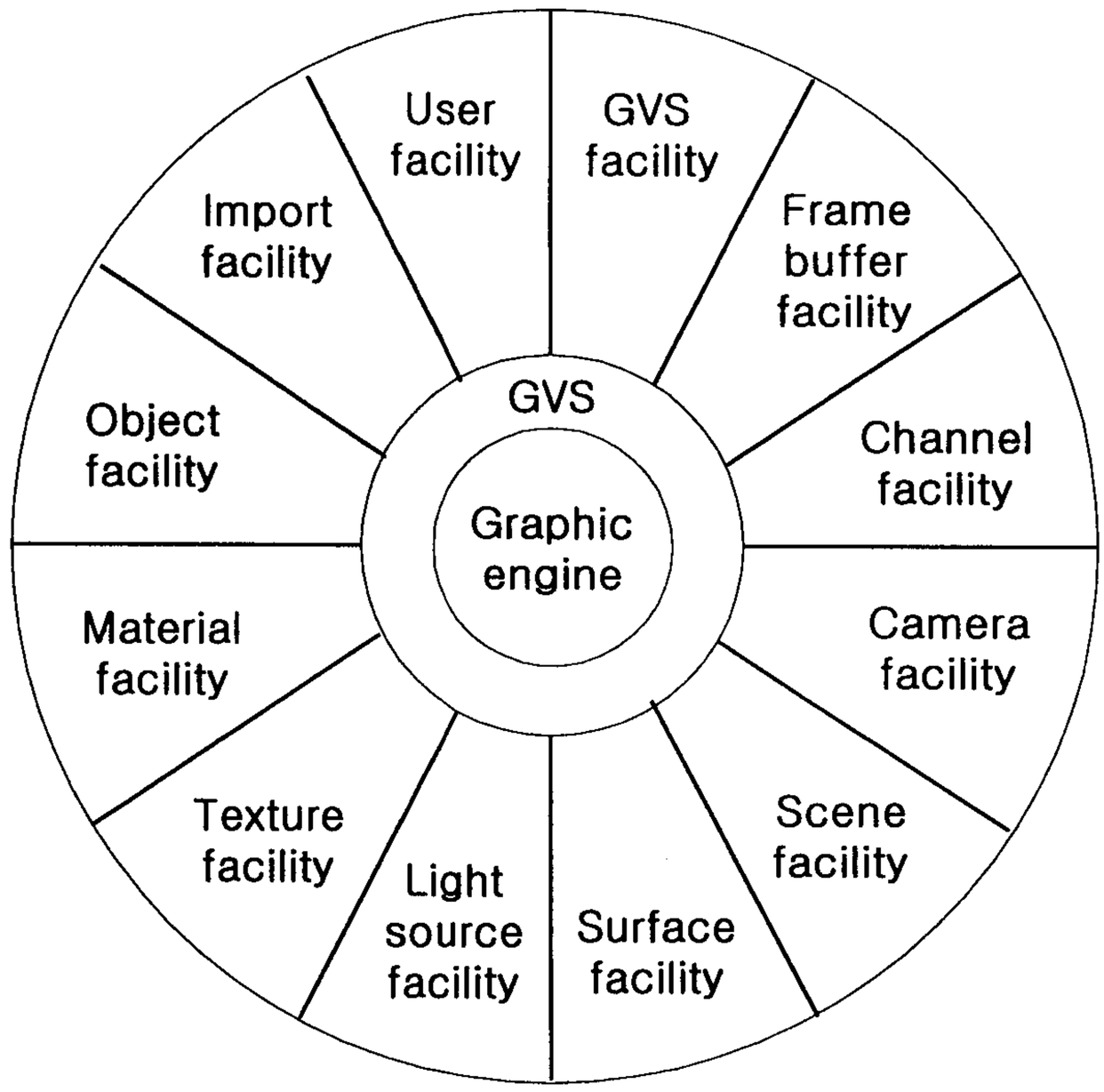
10. Gemini Technology Corporation Generic Visual System(GVS)

통합된 가상환경 시현을 위해서는 Discrete 3D 객체들을 하나의 대화형 비쥬얼 디스플레이를 형성하기 위해 이들을 통합하는 것이 필요하다[23]. 하나의 방법은 3D 데이터 집합을 디스플레이 리스트로 변환하고, 이를 표준 그래픽스 환경에서 사용한다.

가장 효과적인 접근법은 가상 객체들을 생성하기 위한 3D 모델링 팩케지를 운용하고, 이들을 가상세계로 통합하는 수단을 고안하는 것이다. GVS는 최소 수준의 코딩으로 실시간 비쥬얼 모사환경을 생산하는 도구이다. GVS는 분산 컴퓨팅 시스템에 연결된 하나의 비쥬얼 시스템을 제공하기 위해 높은 수준의 도구들을 운용하는데 중점을 두었으며, 다음과 같은 몇가지 중요한 용어들이 있다.

Camera는 관찰자(Viewer)의 위치, 방향 및 각도 등을 제어하기 위한 GVS 자원이다. Channel은 디스플래이 Twist, 디스플레이 Shear, 뷰포인트, Near/Far Clipping Plane, 현재 관찰 카메라 등과 같은 지오매트리 특징들의 디스플래이를 제어한다. 또한, 관찰 거리에 따라 LOD(Level Of Details) 모델들 사이에 자동적인 전환을 제공한다.

GVS 커널은 호스트 플랫폼에 위치한 다수 개의 설비(Facility)들로 분할되며, 각 설비는 기본적으로 하나의 S/W 조각으로 하나의 GVS 자원을 관리한다. 각 설비는 기본적으로 DB, 초기화 루틴들, 조작 루티들, 처리루틴, Shutdown 루틴 등을 가진다. <그림 7>은 GVS의 설비 구조를 도식화한 것이다.



<그림 7> GVS의 설비구조

GVS 설비는 커널로 GVS 응용의 높은 수준의 실행 제어를 제공하며, 정교한 오퍼레이션을 위해서 손쉽게 메인 루틴들에 통합될 수 있다.

사용자 설비는 프로그래머에게 GVS를 엑세스할 수 있도록 하며, 이는 자원들 생성을 위한 코드 다음에 특별한 초기화 코드의 삽입을 가능하게 한다.

객체 설비는 자동화된 화면 관리 기능들(LOD, 자동 객체 컬링)과 객체의 운동 방정식 할당 등을 가능하도록 한다.

Material 설비는 MultiGen에서 GVS로 Import된 객체들의 Material 특징들 재확보 하고, GVS를 통해 생성된 Drawing Primitive들에 대해 Material 정의들을 적용할 수 있다. 주요 정의들에는 Ambient Colour, Diffuse Colour, Emissive Colour, Specular Colour, Reflectivity Coefficient, Transparency Effects emdd 있다.

또한 , 택스쳐에 대해 Material 설비와 동일한 기능을 수행하는 택스쳐 설비, 타 모델링 팩케지에서 생성된 모델들을 GVS 환경으로 Import 해 주는 Import 설비가 있다.

Scene 설비는 사용자에 의해 제어될 Scene들을 생성하며, 각 Scene은 사용자가 정의한 방식대로 함께 가져오는 객체들의 집합이다. Scene 설비는 FOV Culling, 디스플래이된 FOV에 Drawing 오퍼레이션들을 제한하는 등과 같은 저 수준 업무들을 책임진다.

그리고, Light Source 설비는 Light source의 칼라, 강도, 유형(Local , Infinite), 위치/방향, Ambient 칼라 등을 제어하며, Camera 설비는 디스플래이 할 것을 제어하는 방법을 제공하고, 다수의 카메라를 지원하며, 각 카메라는 눈의 위치와 방향을 제어한다.

Channel 설비는 프로그래머에게 특정 GVS 카메라를 하나의 채널에 할당하는 것을 허용하고, 이미지에 Display Twist Angle들 또는 Off Axis Projection Connection들을 적용할 수 있게 한다. 이러한 기능을 통해 상이한 Binocular Overlaps과 Off Axis Projection System 처리가 가능하다.

원래 GVS는 비쥬얼 시뮬레이션 응용들을 위해 개발 되었으며, 다양한 시뮬레이션 응용분야에서 개발시간을 상당히 절약해 주었다. 이는 복잡한 시뮬레이션 환경을 구축하는데 필요한 대부분의 특징들을 GVS 커널에 포함되어 있기 때문이었다. 만일 GVS가 정교한 3D 모델링 팩케지(예:MultiGen)와 결합이 된다면, 대부분의 비쥬얼 시뮬레이션 응용들을 커버할 수 있는 강력한 Tool Set이 될 것이다.

제 3 장 연구개발 수행내용 및 결과

제 1 절 3차원 시각 환경 제시기 개발

1. 3차원 시각 환경 제시기의 주요 연구 개발 내용

● **Simulation Micro-Kernel** : 실험 참여자에게 현실감 있는 가상환경을 제공하기 위해서 요구되는 필수적인 기본 기능들인 시각환경 DB관리, 실시간 3D 렌더링, device I/O, dynamics simulation등을 제공하여 감성공학에서 요구되는 다양한 형태의 실험 요구에 효율적으로 대처하기 위한 기본 커널을 구현하였다.

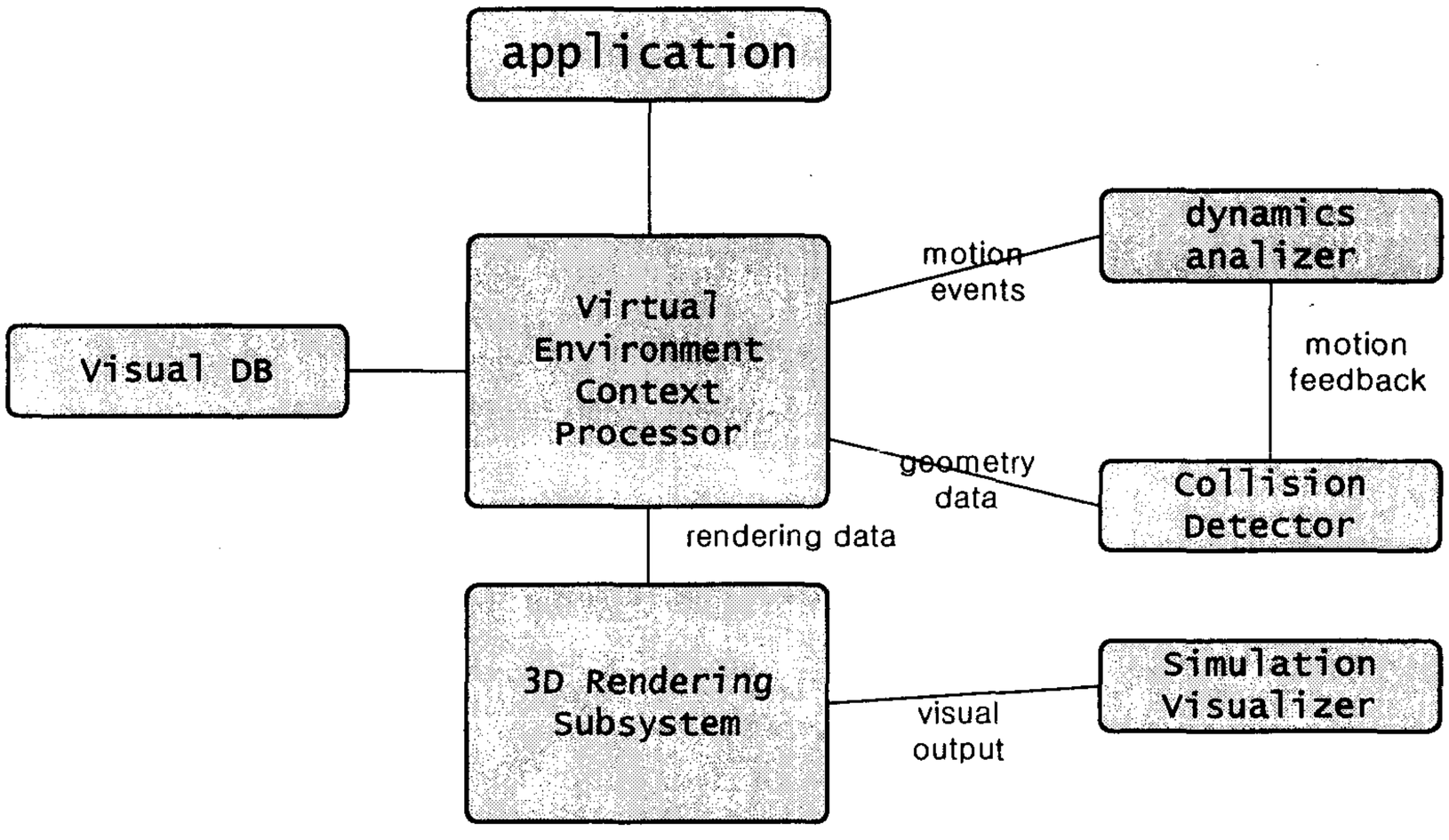
● **Simulation Network Server** : 3차원 시각환경제시기의 기능확장은 다양한 실험 요구에 부응하는 외부 기능 모듈들의 실행시간에 연결에 의해 이루어진다, 이를 위해 simulation server는 가상환경의 여러 공유 객체의 정보들을 유지, 관리하는 서비스를 제공하고 server에 접속하는 기능 모듈들은 이를 바탕으로 실험의 목적에 따른 다양한 종류의 입출력 장치들을 제공하게 된다. 따라서 자연스러운 사용자 인터페이스를 통해 실험 참여자에게 현실에 최대한 가까운 가상환경을 제공하므로써, 인간의 감성 요소 수집과 분석을 지원한다.

● **Simulation Authoring / Control Station** : 감성공학을 위한 범용의 가상환경 저작/운영 환경으로써 실험 계획자가 요구하는 조건에 알맞은 모의 환경을 신속하게 설계 구축하는 저작 기능과 실험 제어 기능, 실행시의 기능확장을 위한 스크립트 처리기 등을 제공하여 다양한 감성 모델의 신속한 검증을 돕게 된다.

● **External Module Interface API** : 외부 기능 모듈 개발자에게는 모의 환경에 존재하는 객체들에 대한 일관성 있는 접근과 통신 기능 제어를 위한 추상화 도구를 제공하므로써 전체 가상환경의 현실감을 높이기 위한 기능 모듈의 편리한 추가와 확장을 지원한다.

2. Simulation Micro-Kernel

3차원 시각환경 제시기의 기본 기능을 정의한 커널로 <그림 8>에서 처럼 가상환경을 기술하는 Context 처리, 시각 데이터베이스 관리, 충돌 검사, 역학 해석, 실시간 3차원 렌더링과 시각 제시기등으로 다양한 실험환경 시뮬레이션을 위한 응용 프로그램의 요구에 반응할 수 있다.



<그림 8> Logical View of the Simulation Kernel Architecture

가. Virtual Environment Context -- VRML

가상환경을 이루는 추상적 모델인 엔티티의 인스턴스들이 환경의 제시단계에서 구체화되기 위해서는 Visual, Audio, Sensor, Behavior등 각각의 단위 제시기가 인식가능한 입출력의 구체적 명세규칙이 필요하다.

특히 이러한 명세 규칙은 실험환경에 관여하는 모든 외부 기능 모듈이 공통적으로 인식가능한 표준화된 규약을 가질 경우 더욱 효과적이며, 실험환경의 저작 단계에서도 저작자에게 유연한 저작 환경을 제공해 줄 수 있다.

본 연구에서는 이러한 가상환경의 Context로써 풍부한 표현력과 표준으로 인한 호환성, 유연한 확장과 조합성등의 이유로 VRML 2.0[24],[25]을 사용한다.

나. Visual DB

현실감 있는 3차원 시각 환경의 효율적인 시각화를 위해서는 <표 2>과 같이 여러 CAD또는 CG등의 3D modeling 도구들로부터 생성된 visual data들을 입수하여 실시간으로 렌더링하는 기능이 필수적으로 요구되는 데, 신속하고 신뢰성 있는 실험 환경의 구축을 위해 기존의 다양한 시청각 자원을 운영하고 관리해 줄 수 있는 데이터베이스 관리가 필요하다.

또한 단순한 자료의 변환을 떠나 실험환경의 저작단계에서 이들을 관리하고 접근하기 위해서는 아래의 예에서 처럼 Inline이라는 가상환경 context를 사용한다.

```
Inline {
        url [ "import-filename", ... ]
}
```

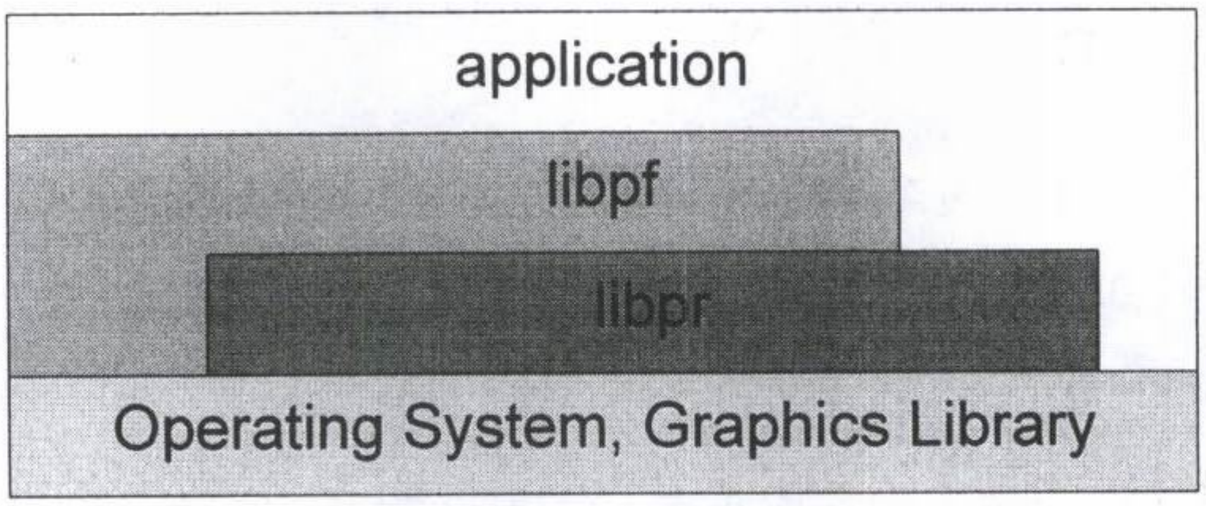
<표 2> Importable File Formats

File extension	Format description
DXF	Autodesk's AutoCAD
3DS	Autodesk's 3D Studio
OBJ	Alias/Wavefront
FLT	Paradim's MultiGen/OpenFlight
IV	Inventor
DWB	Coryphaeus's Designer's Workbench
PFB	SGI's Performer Binary

다. 3D Rendering Subsystem – IRIS Performer

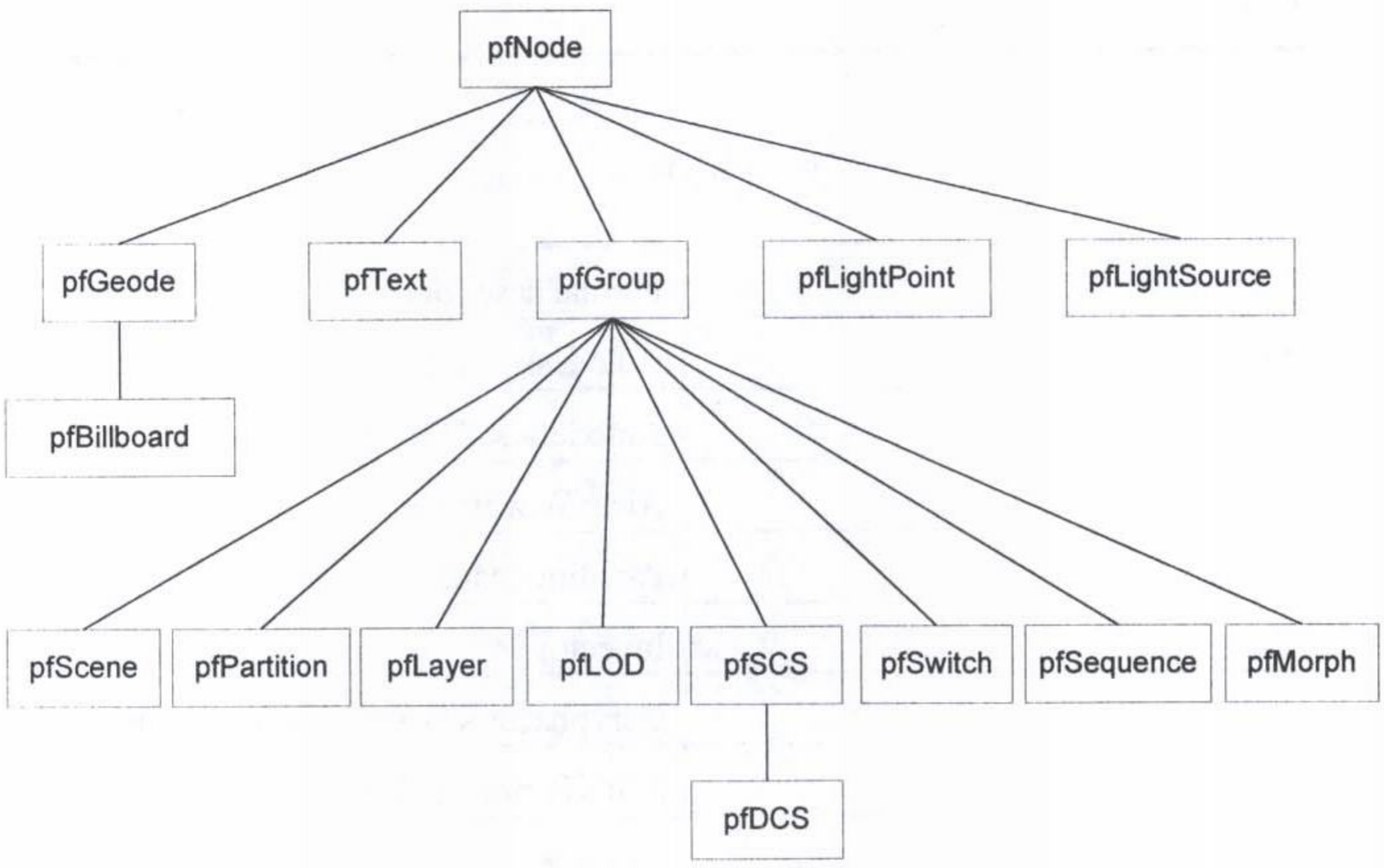
시각 데이터베이스들로 구성된 가상환경의 시각 요소들을 실시간으로 시각화하기 위해서 시뮬레이션 마이크로 커널은 3차원 렌더링을 위한 서브시스템으로 **IRIS** Performer[26],[27]를 사용하였다. 렌더링 서브시스템을 별도의 기능 단위로 분리한 것은 마이크로 커널의 하드웨어 의존도를 최대한 낮추는 데에도 도움이 되는데, 이것은 향후 3차원 시각제시기를 다중 플랫폼에서 운영가능하게 할 수 도 있기 때문이다.

IRIS Performer는 <그림 9>와 같이 하드웨어에 대해 여러 단계의 레이어를 두고 기능 구현에 접근하기 때문에 운영 하드웨어에 따라 최적의 성능을 유지하고, 실시간 렌더링을 위한 다양한 기능들을 제공하고 있다.



<그림 9> Performer Library Layering

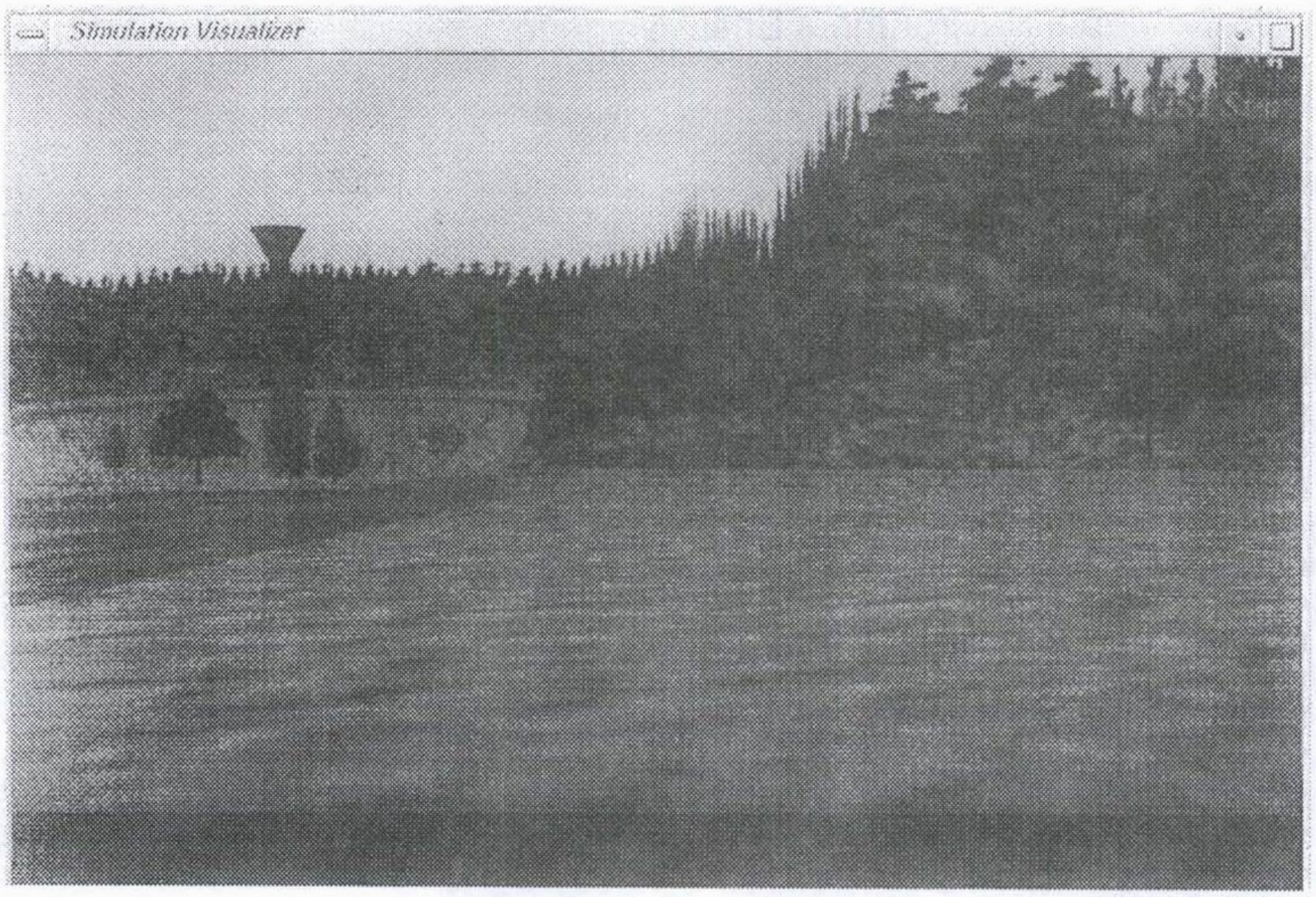
또한 가상 환경을 빠르게 시각화하기 위해 시각 객체들을 구성하는 Scene Graph의 단위 객체로써 <그림 10>처럼 기능 별로 정의한 다양한 노드를 제공한다.



<그림 10> Performer Node Hierarchy

라. Simulation Visualizer

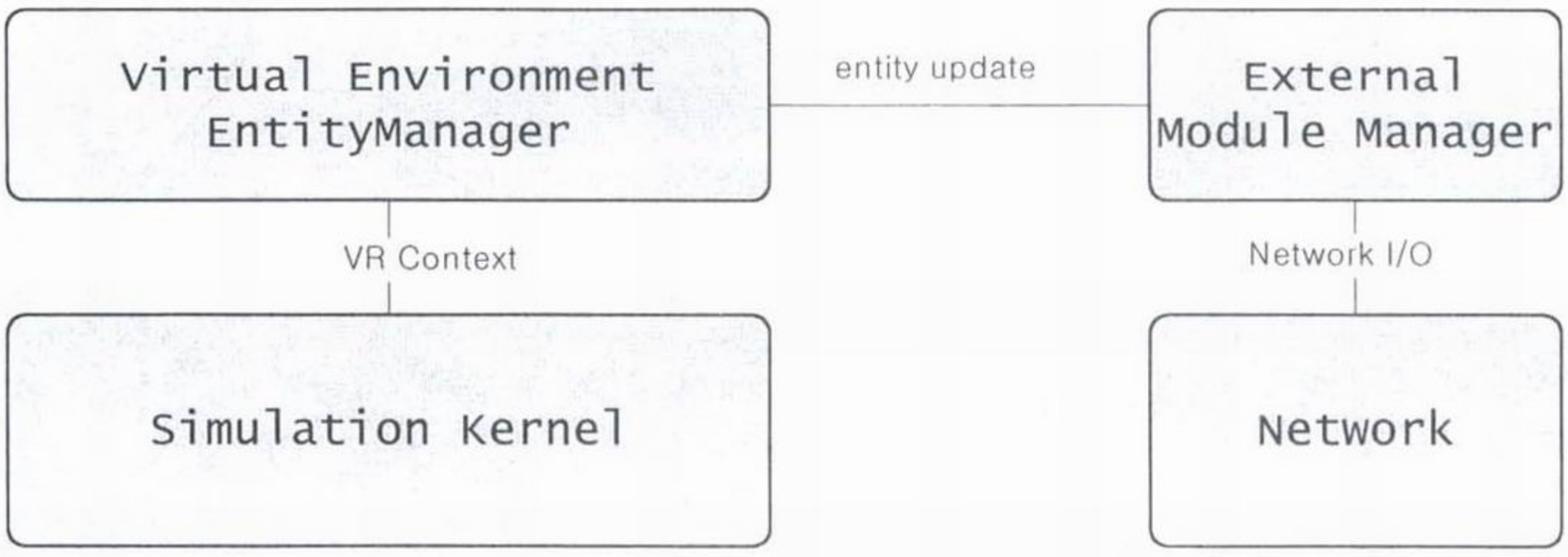
가상 현실 기술은 피실험자의 반응에 대한 시청각 요소의 처리가 실시간으로 디스플레이 되야 한다. 따라서, 시뮬레이션의 제시기와 데이터베이스 관리자는 실시간의 처리를 위하여 그래픽 파이프라인, 상세도, 효율적인 데이터 구조를 이용한다. 또한 모든 상태변수의 참조와 조작이 초당 30 프레임 내에 가능하도록 내부 데이터를 구성하여야 한다.



<그림 11> Simulation Visualizer

3. Simulation Network Server

3차원 시각 제시를 위한 기본 기능들을 제공하는 마이크로 커널은 실험 환경의 요구에 따라 다양한 기능을 제공하는 외부 모듈들과 실행시간에 연결되어 그 기능을 확장하게 되는 데, 이 때 시뮬레이션 서버는 <그림 12>에서 처럼 각 모듈의 접속과 유지, 데이터 공유, 통신등의 종합적인 서비스를 담당한다.



<그림 12> Logical View of the Simulation Server Architecture

가. Virtual Environment Entity Manager

엔티티는 가상환경을 이루는 추상화된 모델들의 기본단위로써 실험환경의 요구에 따라 새로 정의하거나, 기존에 정의된 형태에 필요한 기능적 특성을 추가하므로써 재정의 가능하다.

실험환경의 저작단계에서는 다른 엔티티들과 주고 받을 속성정보를 저장하고 있는 구조체의 형태로 가상환경내에 정의되어 있다가 실험이 진행되는 동안 환경내에서 그 인스턴스를 생성하여 환경과 반응하게 된다.

또한 엔티티들은 외부 기능 모듈들과 Simulation Kernel간의 정보공유의 기본단위가 되어, 적절한 통신 규약에 따르는 기능 모듈이 상이한 환경과 기능에도 불구하고 가상환경을 공유할 수 있게 해준다.

나. External Module Manager

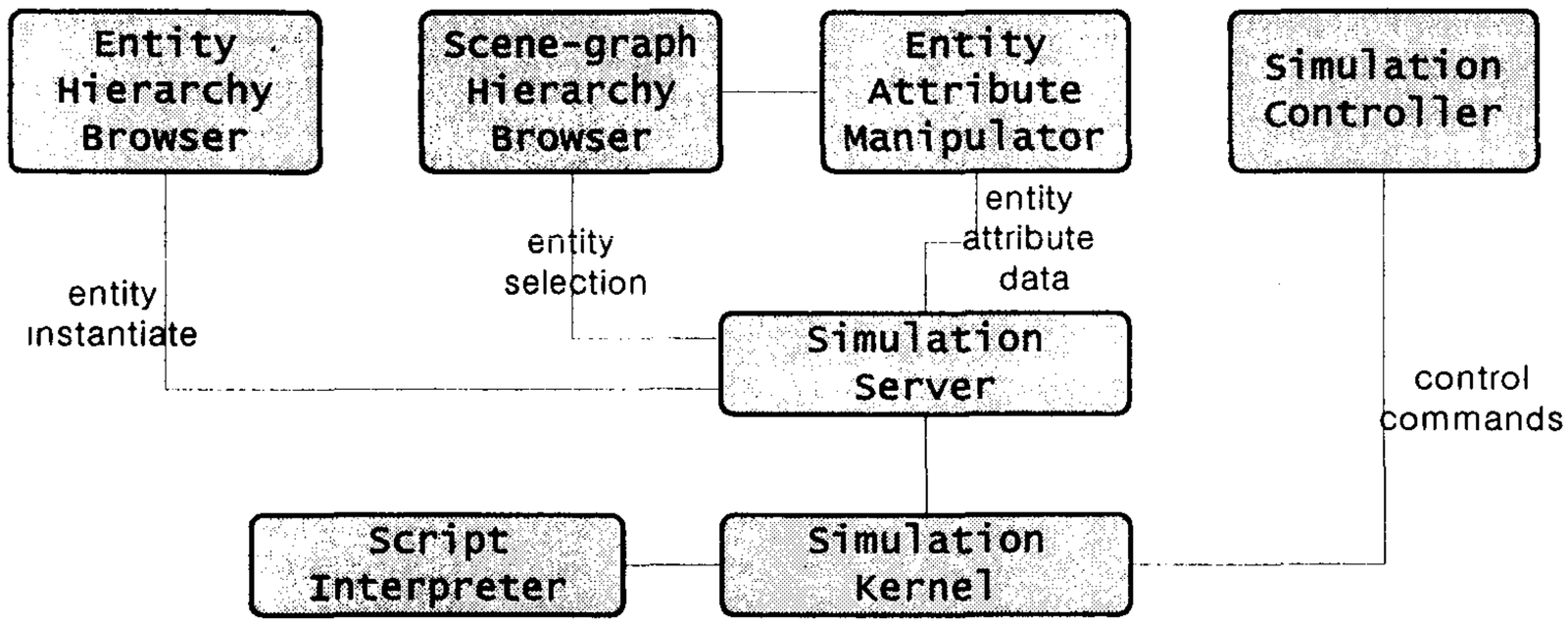
외부 기능 모듈들은 실행시에 서버에 접속하여 가상 환경에 존재하는 객체들에 접근하여 그 정보를 참조하거나 갱신하게 되는 데, 이들이 서버에 접속하여 기능을 수행하고 접속을 종료할 때까지의 다음과 같은 서비스를 제공한다.

- Registration of External Module
- Management of Entity Name/ID
- Notification of Entity Instance Creation
- Monitoring Entity Attribute Modification

4. SACS(Simulation Authoring / Control Station)

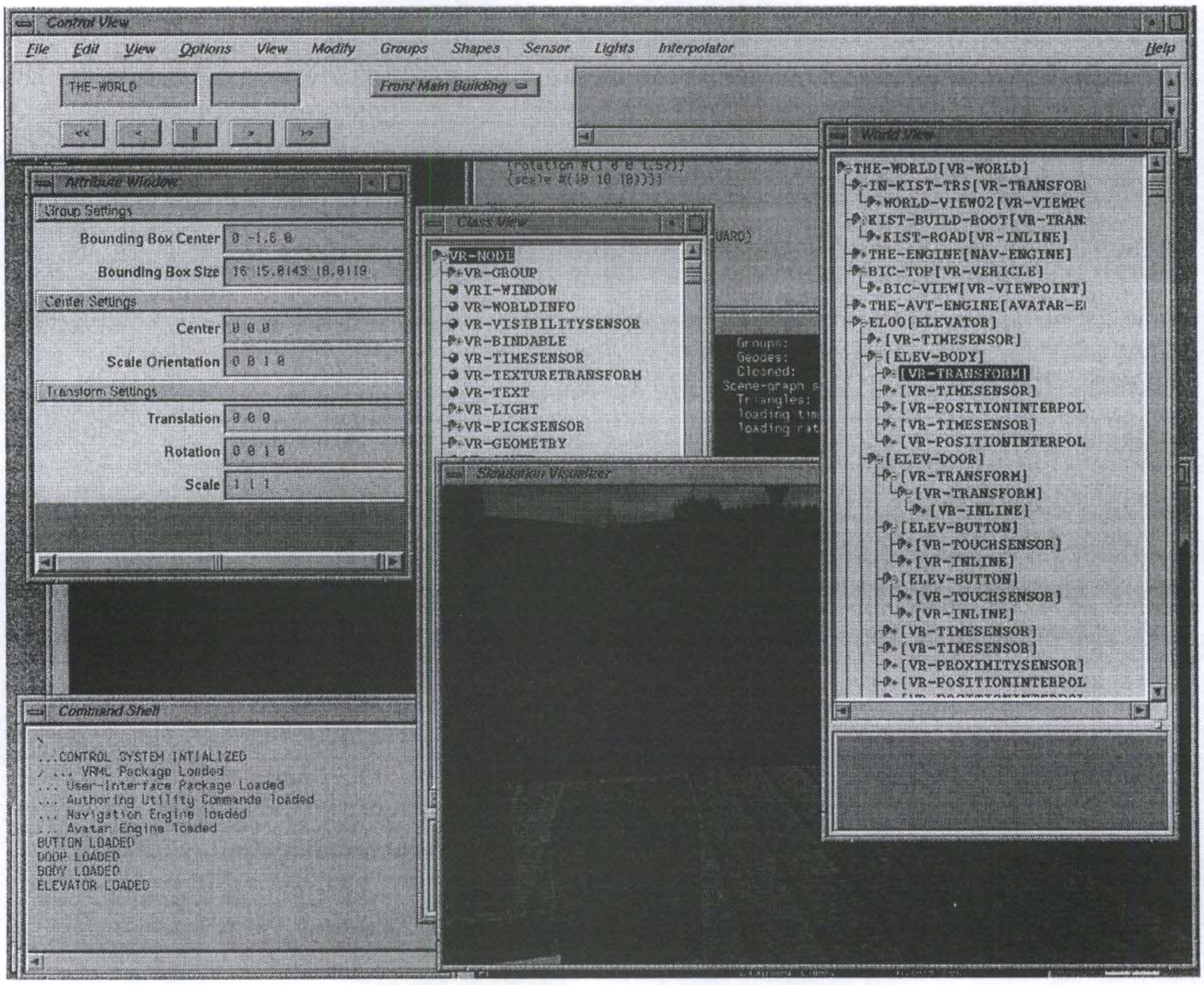
감성공학 실험의 지원도구로써 SACS는 실험계획자에게 제공되며 <그림 13>과 같은 주요 기능 단위들이 유기적으로 상호반응하면서 실행된다.

저작단계에서 가상환경 엔티티들의 형태 정보와 속성정보, 연결관계 등을 시각화해주며 이를 실험 서버에 등록하고 운영단계에서 실험 커널에 전달하여 실험을 통제하는 통합 지원도구이다.



<그림 13> Component Relationship of the SACS

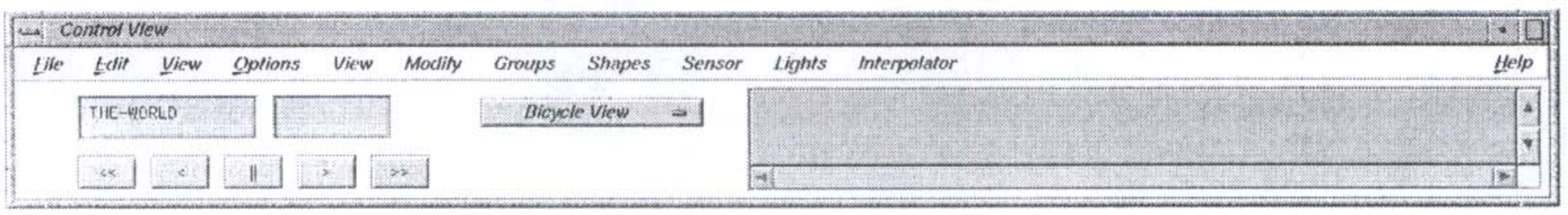
감성공학적 실험환경에 대한 저작단계에서 프로그래밍을 최소화하고 효율을 높이기 위하여 <그림 14>와 같은 비쥬얼 개발 환경을 제공한다. 다양한 제어를 확인하며 저작할 수 있는 도구를 제공하고 있으며 동시에 여러 개의 도구를 통해서 통합 환경을 저작하고 제시할 수 있도록 하였다. 통합 환경 저작/제시기는 전체적으로 시스템을 운영하는 주 조작기를 통해서 조절이 된다.



<그림 14> Snapshot of Simulation Authoring/Control Station

가. Simulation Controller

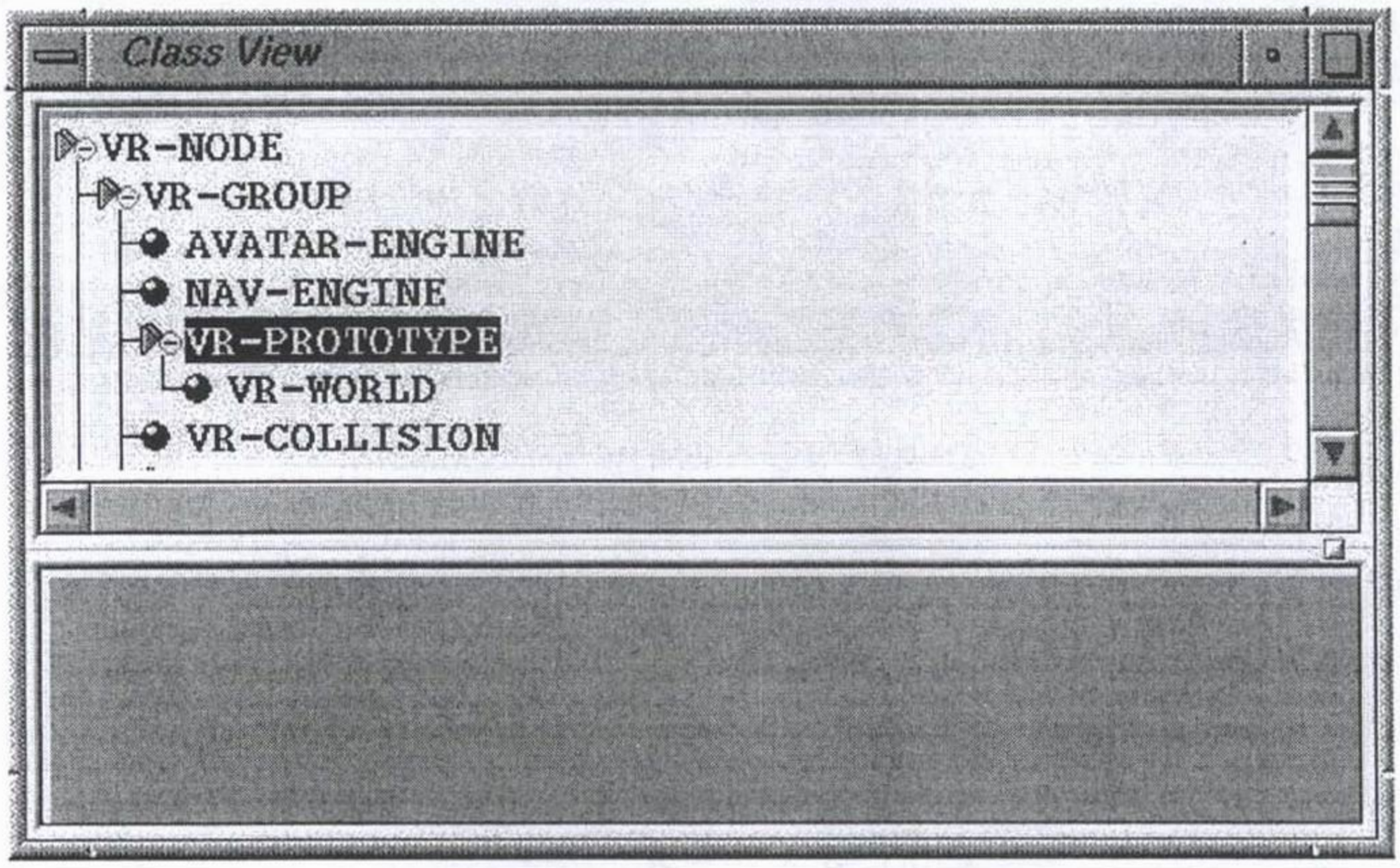
시뮬레이션 제어기는 감성 모델을 검증하기 위하여 실험자가 요구하는 실험환경의 저작과 운영과정 전반에 걸쳐 실험 환경을 조절하는 제어 기능을 제공한다. <그림 15>은 SACS의 전체 운영을 책임지는 실험환경 제어기이다.



<그림 15> Simulation Controller of SACS

나. Entity Hierarchy Browser

3차원의 환경을 저작하려는 사용자는 SACS에서 제공하는 기존의 클래스를 사용할 수 있다. 3차원 환경을 구축하는데 필요한 클래스는 3차원 환경을 구성하는데 기본적으로 필요한 클래스와 특정한 환경을 저작할 수 있는 클래스로 나뉘어 진다. <그림 16>은 현재 가상환경을 구성하고 있는 클래스들의 상속 관계를 찾아볼 수 있는 브라우저이다.



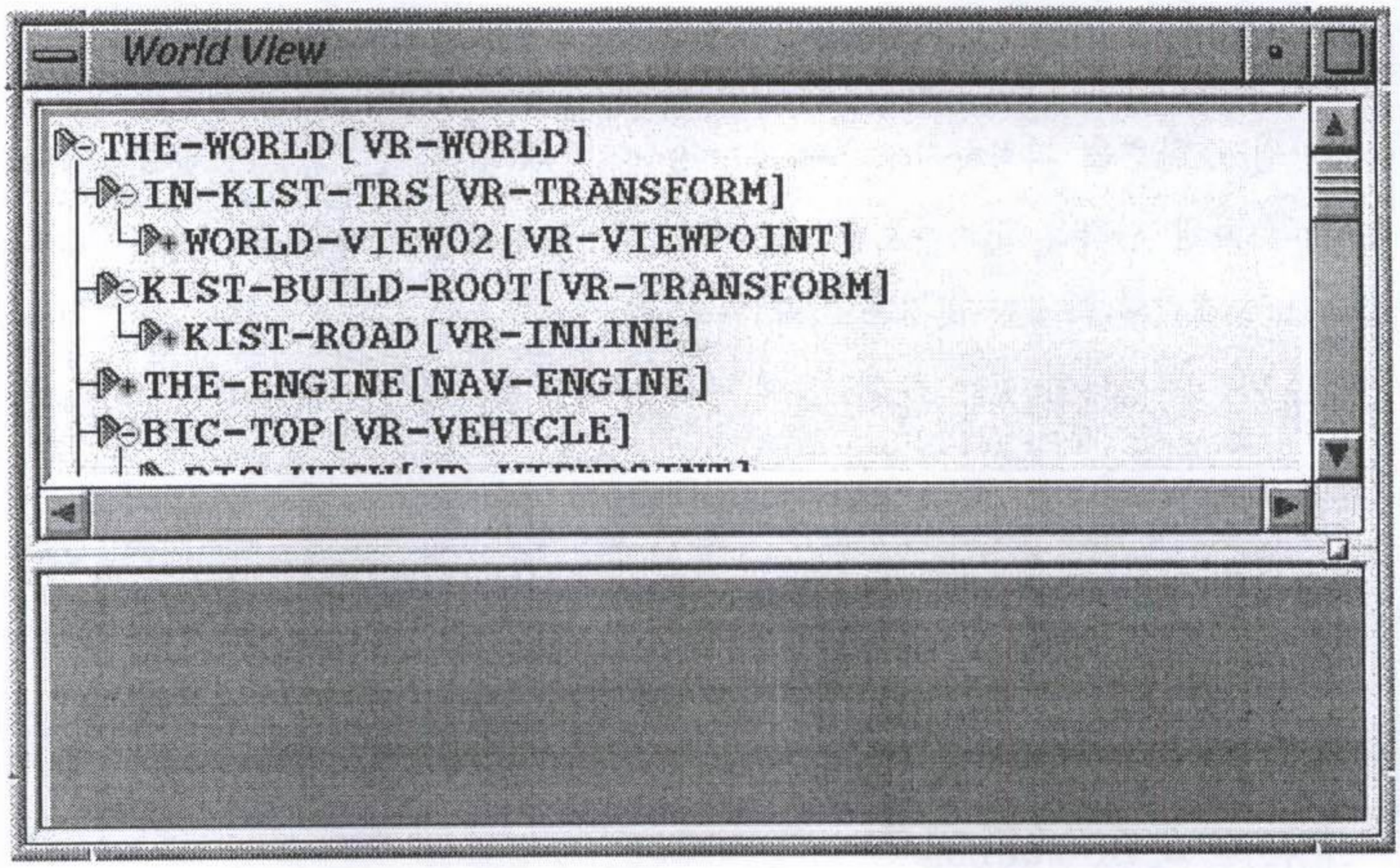
<그림 16> Entity Hierarchy Browser

다. Scene-Graph Hierarchy Browser

<그림 17>은 저작자가 특정 엔티티를 지정하여 그 속성을 변경하고자 할 때 가상 환경 내의 엔티티들의 연관 관계를 시각화 해주는 도구이다.

가상 환경의 엔티티들은 일반적으로 수평적 연관 관계뿐 아니라 상하의 종속 관계를 가지고 구성되기도 하는 데, 이러한 엔티티들의 연관관계를 scene-graph라 부르고, 브라우저에서는 그들을 위에서 아래로 나열하는 형태의 리스트를 사용해 표시해 준다.

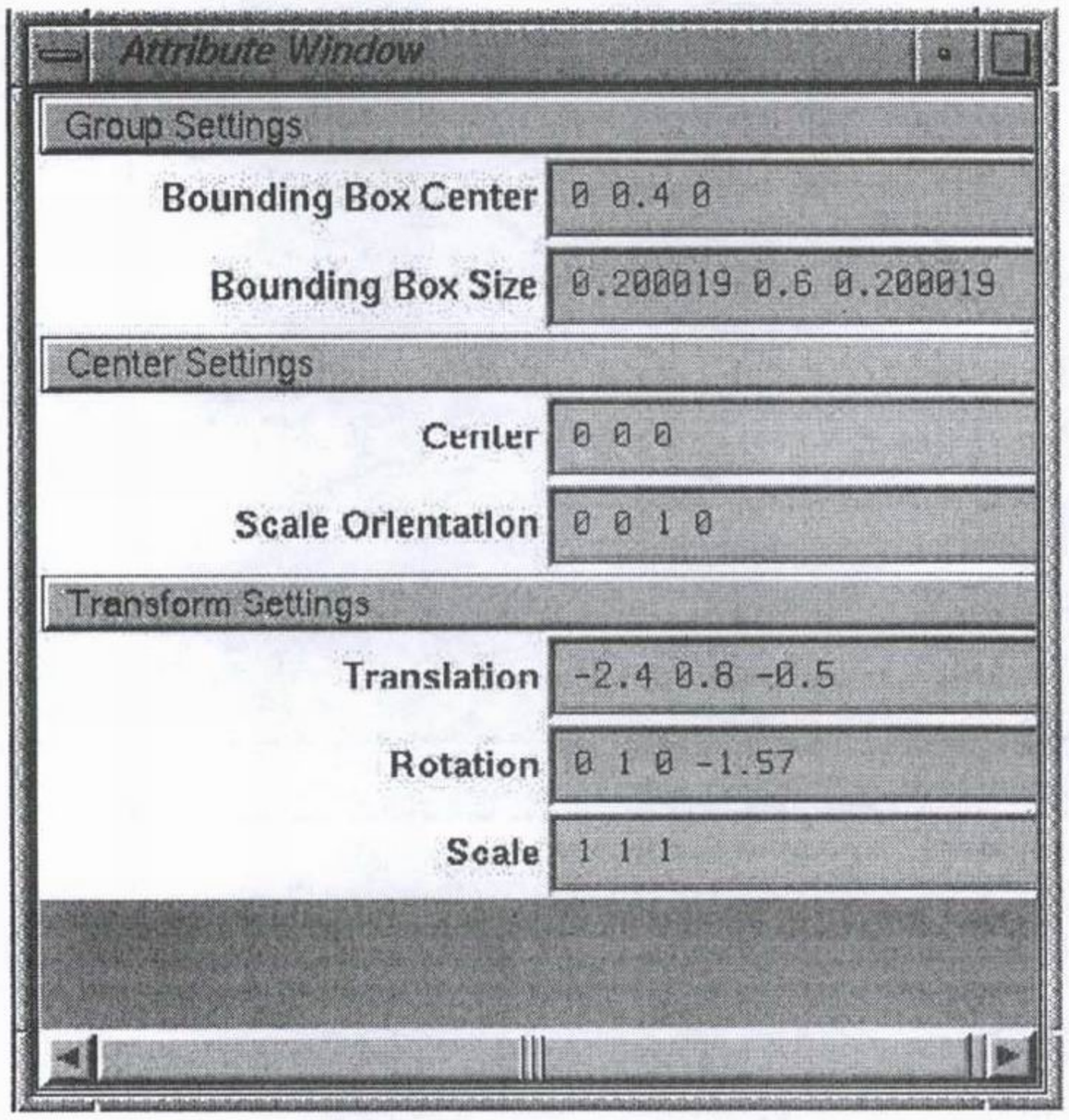
이 때, 원하는 엔티티를 선택하면 그 엔티티의 정보가 속성 편집기로 전달되고 반대로 시각제시기의 편집창에서 직접 엔티티를 선택하면 그 내용이 브라우저의 현재 아이템으로 설정된다.



<그림 17> Scene Graph Hierarchy Browser

라. Entity Attribute Manipulator

가상환경 속에 존재하는 엔티티들의 속성을 참조하고 편집하기 위한 도구로써 상위 엔티티들로부터의 상속 관계에 따라 <그림 18>와 같은 편집창을 다르게 볼 수 있어, 여러 단계의 속성정보 접근을 가능하게 해준다. 특히 이들 편집 기능은 기본으로 제공되는 기능외에도 이들을 조합하는 스크립트를 통해 확장할 수 있어, 실험환경의 특성에 따라 저작자에게 더 편리한 환경을 제공해 줄 수 있다.



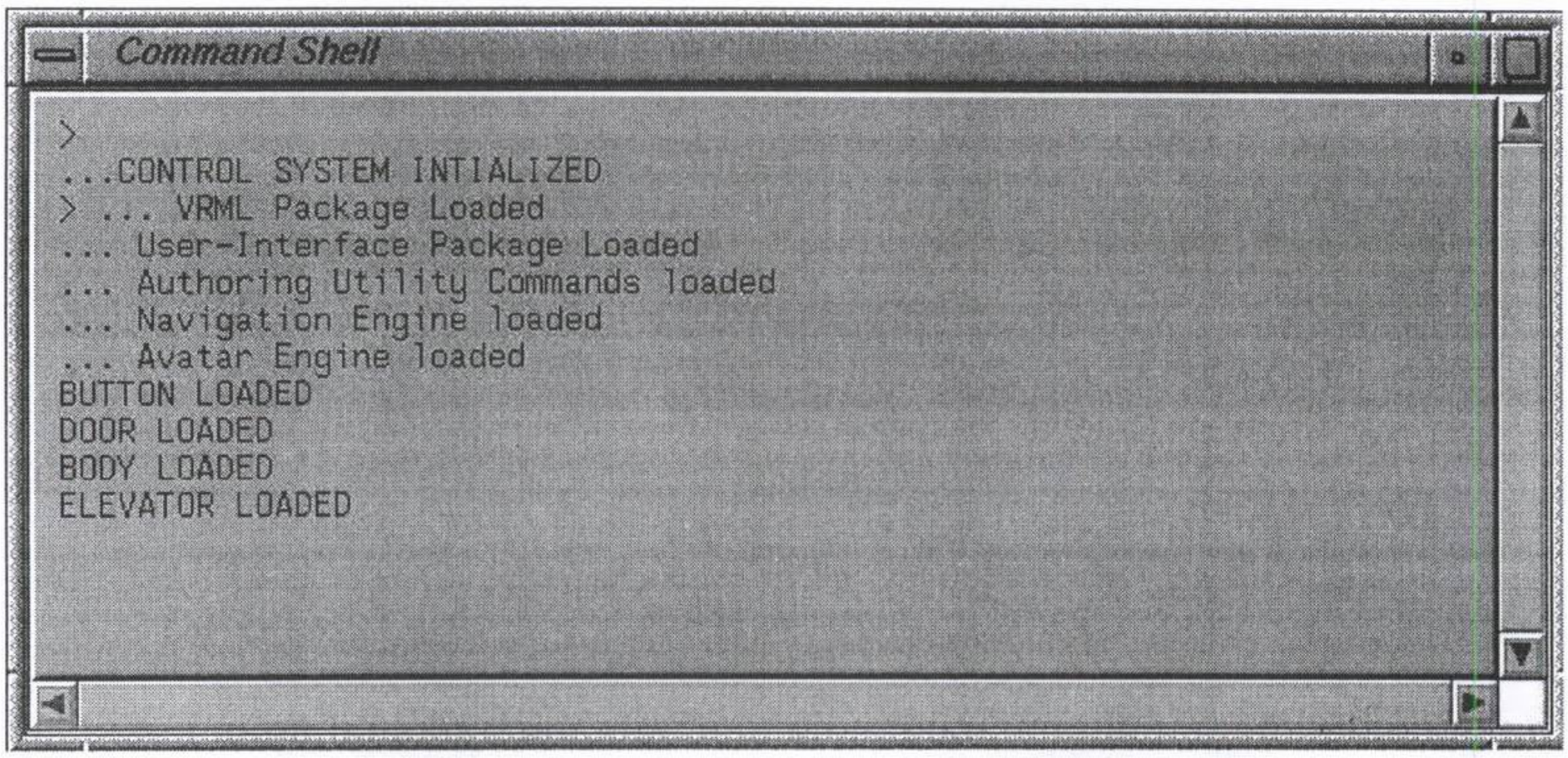
<그림 18> Entity Attribute Manipulator

ㅁ. Virtual Environment Authoring Script(SOOL)

가상 환경속에 존재하는 여러 객체들간의 연결 관계를 기술하고 기능 모듈의 실행 상태를 지정해 주는 등 Simulation Kernel이 저작단계에서 외부 환경과 대화하는 도구로 [28] 스크립트 언어 SOOL(Simplified Object Oriented Language)을 제공한다.

SOOL은 기호 연산과 리스트 연산을 주요 기능으로 다루는 Scheme[29]에서 기본 구조를 차용하고, 단순화된 객체 지향 구조[30]를 추가한 것으로 Simulation Kernel의 여러 계층에서 폭넓은 기능 확장을 지원한다.

<그림 19>은 시뮬레이션의 저작과 조절의 기반이 되는 스크립트 언어를 사용하기 위한 명령어 셸(Command Shell) 창이다. 다양한 실험환경에 대한 효과적인 구축을 가능하도록 하였으며 이전에 개발된 환경 데이터에 대한 수정과 재사용을 용이하도록 하였다.



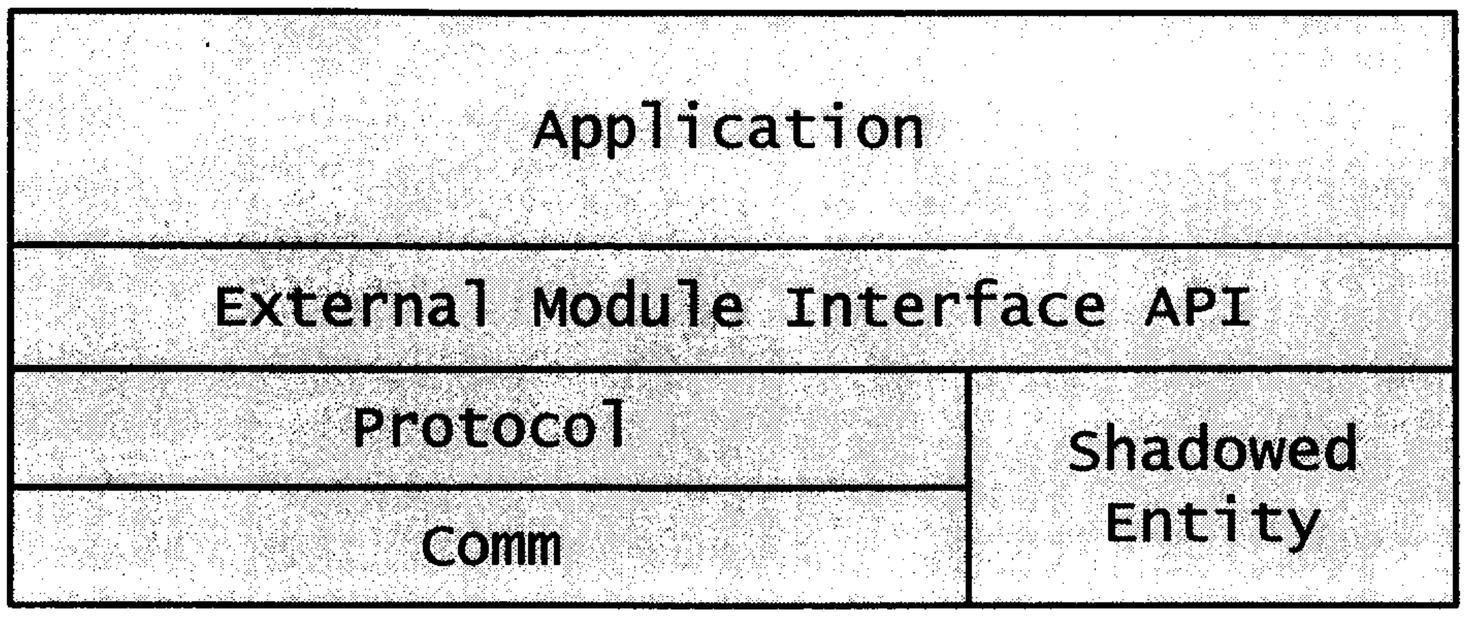
<그림 19> Script Command Shell

바. 개발환경

SACS의 하드웨어 기반은 별도의 전용 시스템을 이용하지 않고 실리콘 그래픽스 워크스테이션과 IBM-PC를 사용한다. 소프트웨어 개발 환경은 IRIS Performer, X-Windows, Motif , C/C++ 이다. 3차원 데이터를 모델링하는 도구로는 MultiGen, Wavefront, 3D Studio을 이용하였다.

5. EMI(External Module Interface) API

감성공학 실험을 위해서는 다양한 실험 목적에 부합하는 기능을 제공하기 단위 모듈을 외부 기능 모듈이라 정의하고 이를 위해 simulation server에 접속하여 원하는 엔티티들의 정보를 참조하고 변경하는 기능들을 손쉽게 프로그램할 수 있는 API(Application Programming Interface)를 제공한다. 이 API는 기본적으로 C++로 작성되어 이식성을 보장할 수 있고, <그림 20>과 같은 레이어를 통해 네트웤등의 하부 기능들도 대신 처리해 줄 수 있어 기능 모듈 제작자는 응용 프로그램 본래의 기능 구현에 전념할 수 있다.



<그림 20> External Module API Layering

가. Shadowed Entity (Run-time Entity)

저작자가 실험 저작 단계에서 구성한 엔티티들은 실험 서버에 등록되어 있는 데, 이들은 각종 응용 프로그램의 실행환경 차이로 인해 직접 공유될 수 없다. 또한 세부적인 속성의 차이들도 최대한 은닉해 두는 것이 기능 모듈의 개발단계에서 복잡도를 낮추는 데 도움이 된다.

따라서 각 모듈들은 자신이 제공하려는 기능적 의도에 따라 참조하려는 엔티티들의 허상을 만들어 응용 프로그램 내부에 위치시키고 이들을 대상으로 속성의 참조와 변경을 가하게 된다. API내부에서는 이들 허상 엔티티들을 주시하고 있다가 그들이 변경되었을 때 실험 서버와의 통신을 통해 그들의 상태를 일치시킨다.

나. Protocol

외부 모듈이 서버와 접속될 때 서버의 상태와 환경을 인지해 내고 모듈 자신의 등록 정보를 전달하는 기능을 자동으로 처리해 준다. 특히 서버와의 통신 경로, 방식등을 지정하는 데에 매우 중요한 역할을 수행하며, 기능 모듈이 수행하는 기능의 중요도, 엔티티 참조 빈도, 실행환경에 따라 앞으로도 다양한 프로토콜을 제공할 계획이다.

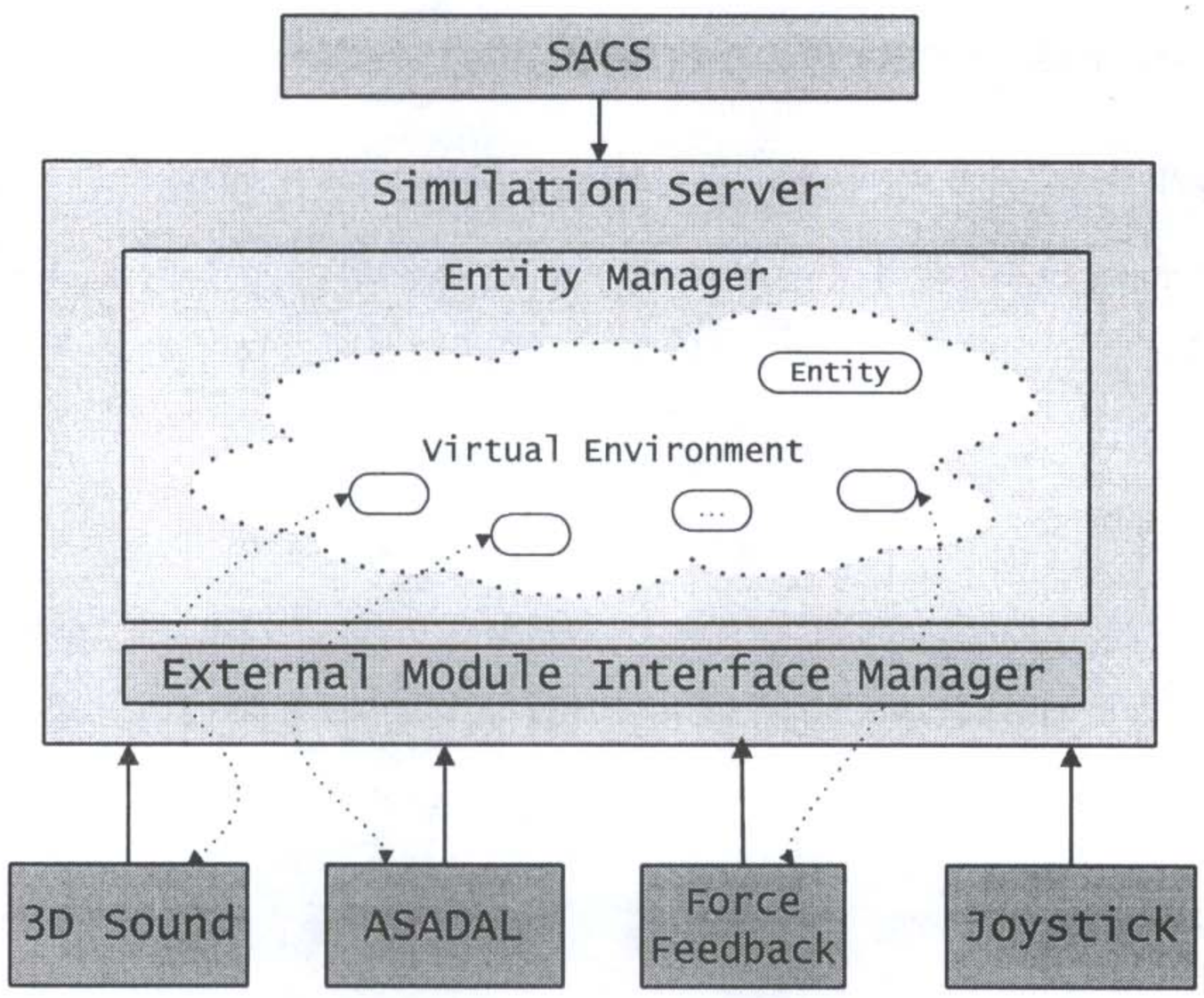
다. Comm

가상환경의 현실감을 높이기 위해서는 다양한 입출력 디바이스를 사용한 사용자 인터페이스가 필요하게 되는 데, 이런 기능은 대부분 외부 기능 모듈을 정의해서 제공하

게 된다. 이를 위해 입출력을 위한 저수준 함수들을 추상화한 통신 구조를 API에 포함시켜 모듈 개발자로 하여금 입출력 제어의 세부 사항으로부터 자유롭게 해 줄수 있다.

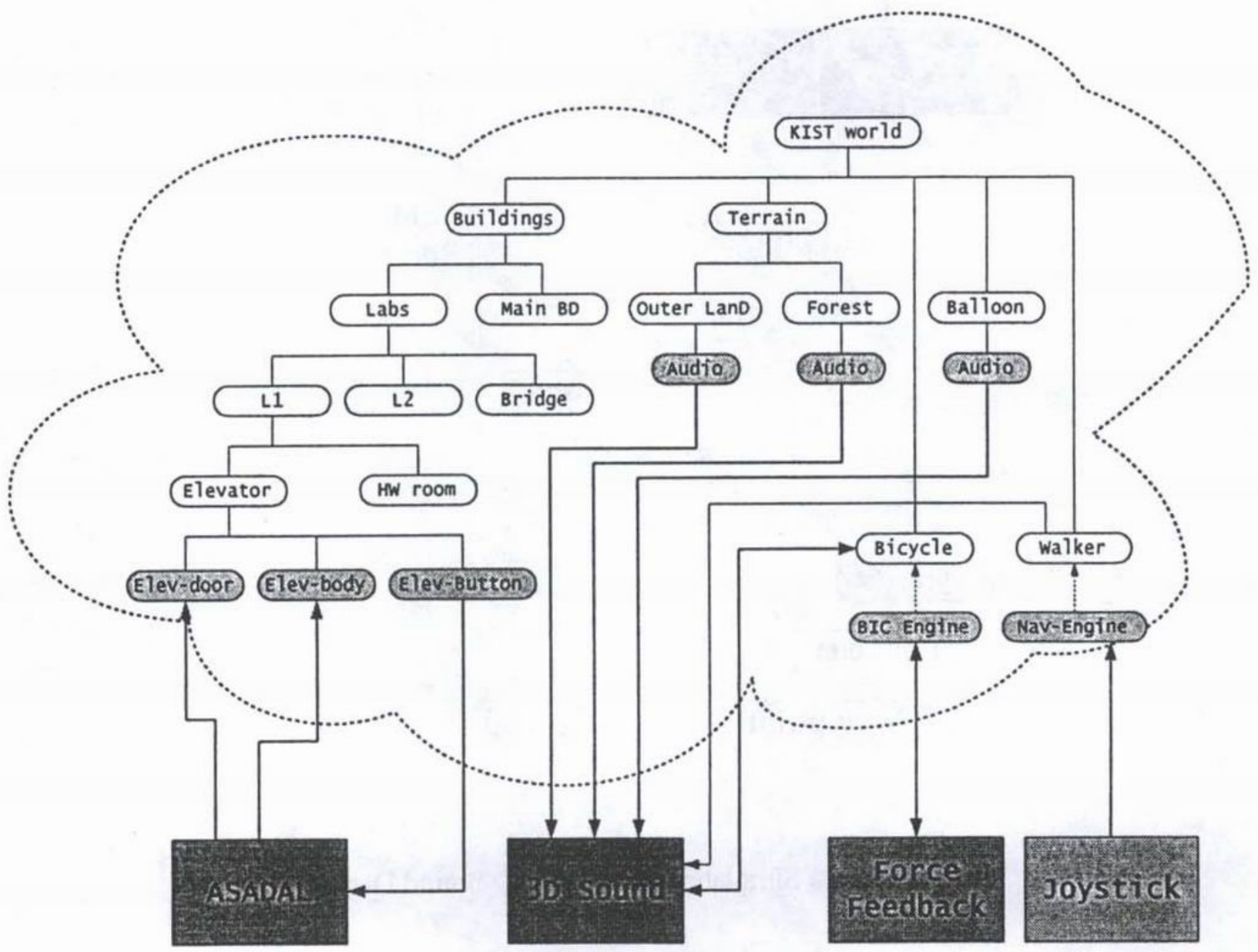
6. 3차원 시각환경제시기 실험 운영사례 –"네비게이션 장치의 성능 평가"

"네비게이션 장치의 성능 평가" 실험의 경우는 <그림 21>과 같이 자전거의 force-feedback, 3차원 청각환경제시기, 가상환경의 객체 행위 저작기, 조이스틱 인터페이스등의 기능 모듈을 정의하여 수행되었다.

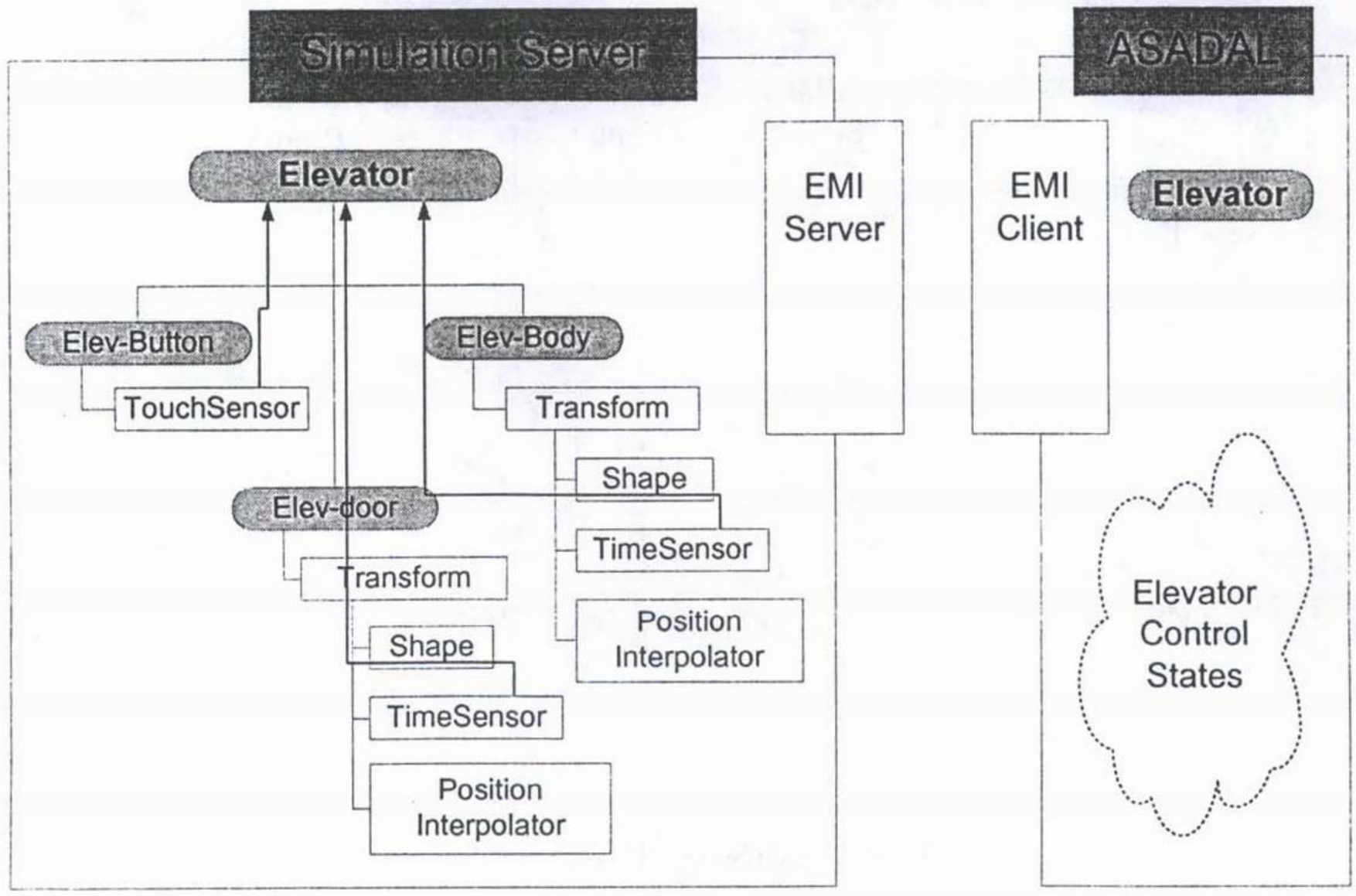


<그림 21> Run-time configuration of Bicycle Navigation Experiment

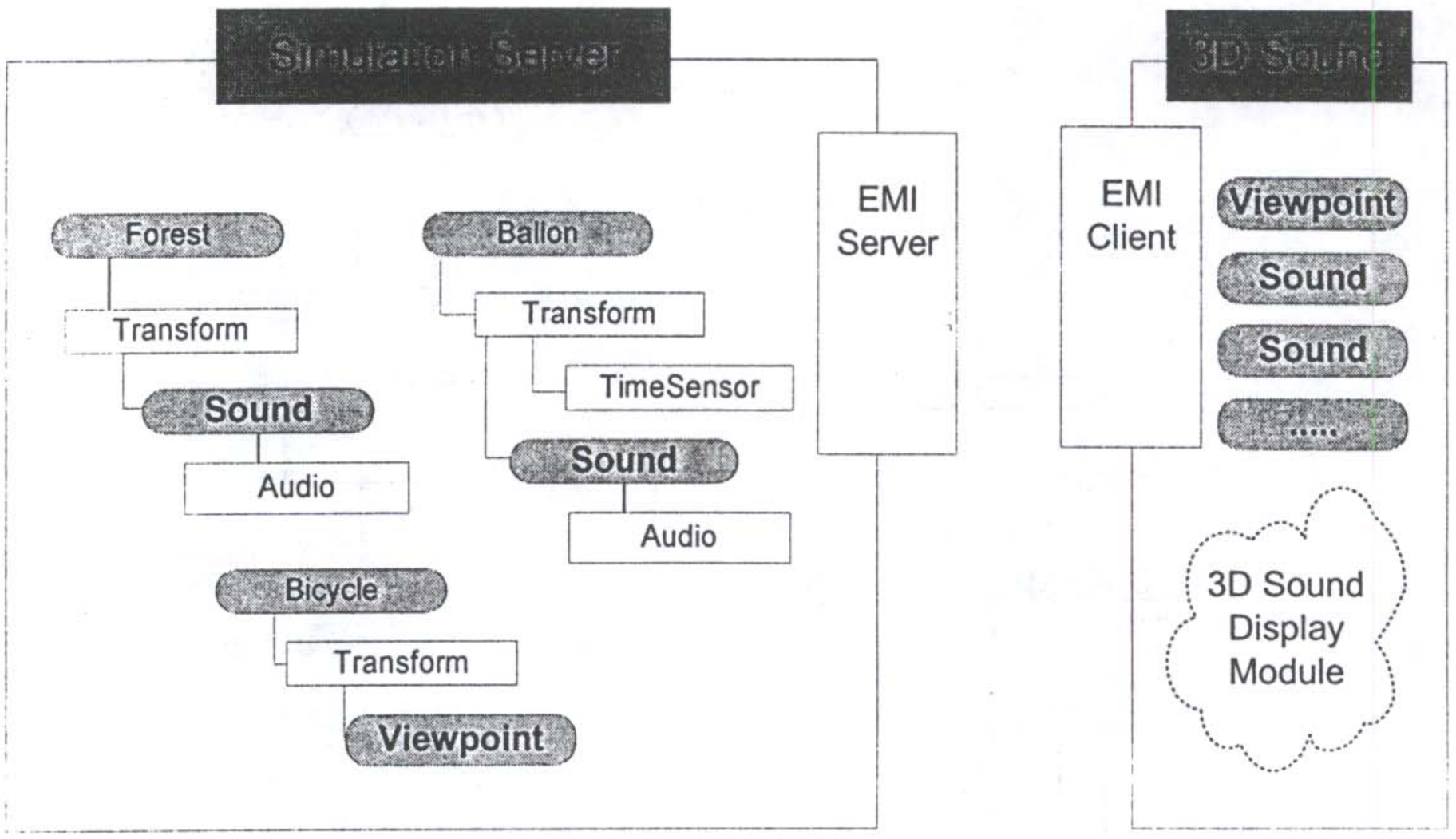
실험에 사용된 엔티티들은 <그림 22>와 같은 계층구조를 가지고 실험서버내에 존재하게 되고, 각 기능 모듈은 실행시에 서버에 연결되어 각자가 필요로 하는 엔티티들의 정보를 추출한 후 EMI API를 사용해 엔티티를 공유하게 된다. 가상 환경의 실제 엔티티들과 각 모듈이 서버로부터 생성해 내는 허상 엔티티들은 <그림 23>, <그림 24>, <그림 25>와 같다.



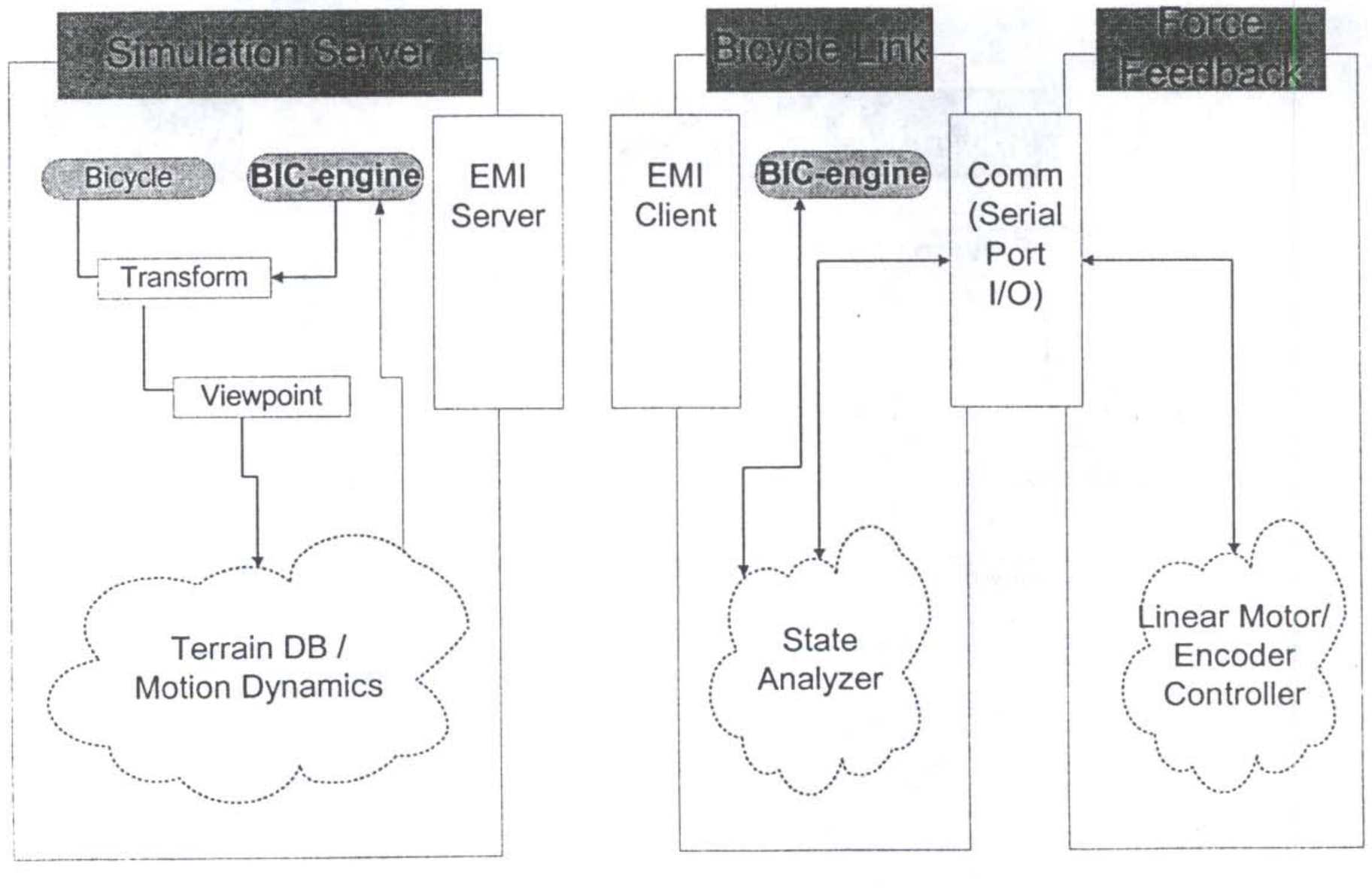
<그림 22> Entity hierarchy & External Module Configuration



<그림 23> Simulation Server와 ASADAL



<그림 24> Simulation Server와 3D Sound Display Module

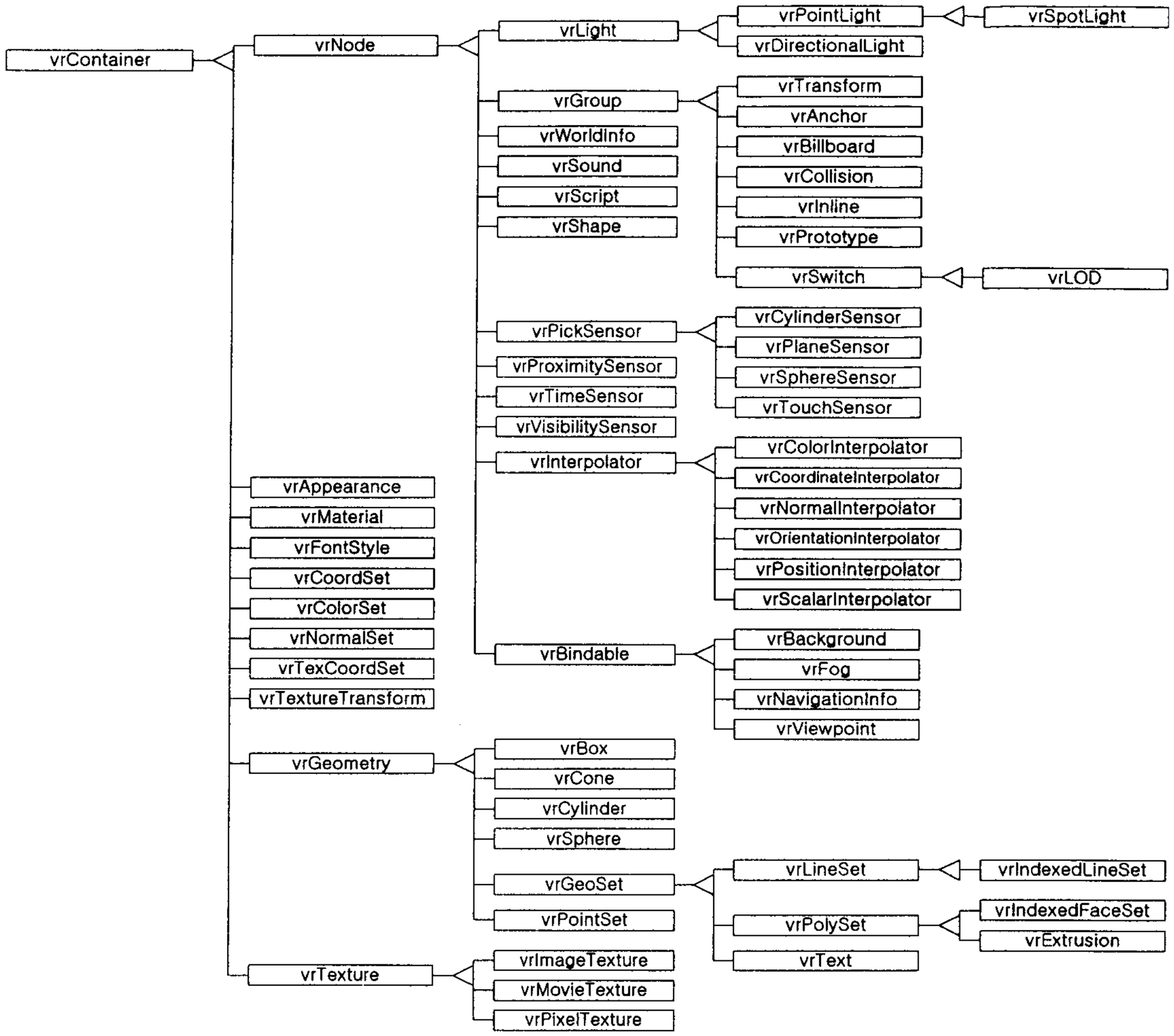


<그림 25> Simulation Server와 Force Feedback Device Module

7. API Reference I (Virtual Environment Context API)

VRML화일을 분석하여, 노드를 구성하고 노드를 지원할 PERFORMER & OPENGL Binding을 제공하는 라이브러리, 다음과 같이 구성되어 있다.

- Internal Representation Class
- Field Class
- World Description & Viewing Information
- Grouping Nodes
- Special Groups
- Common Nodes
- Sensors
- Geometry
- Geometric Properties
- Appearance
- Binable Nodes
- Loading Interface Class



<그림 26> class heirarchy of VRML

가. Internal Representation Class

(1) vrMemory

메모리 공간을 획득/반환 서비스를 제공하는 기반 클래스

(가) Class Definition for vrMemory

```
vrMemory class <- NONE
        void *      operator new ( size_t );
        void        operator delete( void * );
```

(나) Main Features of the Methods in vrMemory

operator new ()
메모리 공간 할당

operator delete ()
메모리 공간 반환

(2) vrObject

객체의 레퍼런스를 기록하고 생존 기간을 유지해주는 기반 클래스

(가) Class Definition for vrObject

```
vrObject class <- vrMemory
        void        ref();
        void        unref();
        unsigned int            getRefCount();
        void        setFlag(unsigned int f);
        void        clearFlag(unsigned int f);
        int         getFlag(unsigned int f);
```

(나) Main Features of the Methods in vrObject

ref()

객체의 레퍼런스 카운트를 증가시킨다.

unref()

객체의 레퍼런스 카운트를 감소시키고 레퍼런스하는 객체가 없다면 객체를 제거한다.

gefRefCount()

객체의 레퍼런스 카운트를 반환한다.

setFlag() clearFlag() getFlag()

객체의 Flag를 설정하고, 지우며, 반환한다.

(3) vrContainer

객체의 이름과 상속객체의 특정 정보 및 VRML의 각 노드의 필드및 Event Route에 관련된 자료 저장소를 가지고 있는 기반 클래스

(가) Class Definition for vrContainer

```
vrContainer <- vrObject
        virtual void setName(const char *);
        char *                  getName();
        void                    setUserData(void * data);
        void *                  getUserData();
        void                    setUpdate(short id, int d);
        void                    clearUpdateAll();
        int                             isUpdated(short id);
        int                             touchUpdate(short id);
        void                    sendEvent(short id);
        void                    sendEvent(class vrField *);
        void                    setUpdateBound(int = VR_TRUE);
        virtual vrContainer * instantiate() = 0;
        vrField *   lookupField(int id);
        vrField *   lookupField(const char * name);
        vrField *   addField(vrField *);
        int                     getFieldCount();
        vrField *   getField(int id);
        vrField *   getField(const char * fname);
        vrField *   getFieldIndex(int i);
        vrField *   getDefEventIn(const char * fname);
        vrField *   getDefEventOut(const char * fname);
```

```
class vrFieldInfo *         getFieldInfo(int id);
vrFieldInfo *               getFieldInfo(const char *);
int                         getNumFieldInfos();
vrFieldInfo *               getExFieldInfo(int id);
vrFieldInfo *               getExFieldInfo(const char *);
void                        setExFieldInfos(vrFieldInfoList * fil);
vrFieldInfoList *           getExFieldInfos();
void                        addParent(vrContainer * cont);
int                         removeParent(vrContainer * cont);
vrContainer *               getParent(int);
int                         getNumParents();
void                        validate();
virtual void realize() = 0;
virtual void draw() { ; }

enum CloneEnum {
            CLONE_SELF,
            CLONE_NODES,
            CLONE_ALL
};

vrContainer *               clone();
virtual vrContainer *       clone(CloneEnum what);
void                        copy(vrContainer * src);
#ifdef      PERFORMER
virtual pfNode *            get_pfNode() const { return NULL; }
#endif      //PERFORMER
```

(나) Main Features of the Methods in vrContainer

setName() getName()

객체의 이름을 설정 및 반환한다.

setUserData() getUserData()

상속 객체가 필요로 하는 공간을 객체에 삽입하고 반환할 수 있는 일반화된 멤버 함수

setUpdate() touchUpdate() clearUpdateAll() isUpdated()

객체가 렌더링시에 변환해야 될 부분이 무엇인지 변환된 부분을 설정하고 초기화하고 변환되었는지 질의하는 함수

sendEvent()

ROUTE 구문에 의해 연결되어 있는 Event-Out Field를 다른 노드의 Event-In Field로 전달하는 함수

setUpdateBound()

VRML의 Scene Graph상의 상위 노드로 전이되며 VR_UPDATE_BOUND flag를 설정하는 함수

instantiate()

Container를 상속받는 클래스들이 객체를 만들기위한 가상 함수

lookupField()

객체의 필드를 참조해주는 함수

addField(vrField *)

객체의 필드를 추가하는 함수

getFieldCount()

객체의 필드 개수를 구하는 함수

getField()

객체의 필드를 참조해주는 함수, 필드가 존재하지 않고 유효한 필드인 경우에는 VRML의 표준 필드값으로 필드를 채워서 반환해준다.

getFieldInfo()

필드를 생성할 수 있도록 각 필드에 대한 정보를 가지고 있는 객체를 반환 해주는 함수

getFieldIndex()

객체가 가지고 있는 필드 배열을 인덱스를 가지고 참조해주는 함수

getDefEventIn() getDefEventOut()

VRML화일의 ROUTE구문을 처리하기위해서 In/Out의 필드를 getField함수를 통하여 참조해 주는 함수

getNumFieldInfos()

VRML 노드의 필드정의가 몇개가 있는지 반환해주는 함수

setExFieldInfos() getExFieldInfos()

추가의 필드를 처리하기 위해 필드의 리스트를 설정하고 반환해주는 함수

getExFieldInfo()

추가의 필드의 리스트중에서 특정 필드를 반환해주는 함수

getParent()

노드의 부모 노드를 반환해주는 함수

getNumParents()

노드의 부모 노드의 개수를 반환해주는 함수

addParent() removeParent()

노드의 부모 노드를 추가/삭제해주는 함수

validate()

변경된 필드를 dirtyFields 배열에 넣어주는 함수

realize()

노드를 PERFORMER 구현으로 실제 적용하는 가상 함수

draw()

노드를 OPENGL로 그려주는 함수

clone()

현 노드의 사본을 만들어주는 함수

copy()

노드의 내용을 복사해주는 함수

get_pfNode()

그래픽 랜더러중에 PERFORMER를 이용할 경우 pfNode를 얻는 함수

(4) vrNode

VRML의 노드의 활성화 관리와 노드가 차지하는 공간를 관리하는 클래스

(가) Class Definition for vrNode

```
vrNode <- vrContainer
        virtual void setActivate(int a);
        virtual int   getActivate();
        void                    setSphereBound( vrSphereBound *, int );
        int                                 getSphereBound( vrSphereBound * );
        const vrSphereBound *getSphereBound();
```

(나) Main Features of the Methods in vrNode

setActivate() getActivate()

노드가 활성화되어 있는지 설정/반환하는 함수

setSphereBound() getSphereBound()

노드가 차지하는 공간을 설정/반환하는 함수

나. Field Class

Field의 기본 타입

```
typedef unsigned char vrBool;
typedef float                   vrFloat;
typedef float                   vrTime;
typedef int             vrInt;
typedef unsigned shortvrFlag;
typedef unsigned char vrUByte;
```

(1) vrType

한 노드안의 FieldInfo들을 관리하기 위한 클래스

(가) Class Definition for vrType

```
vrType class <- NONE
        vrType(vrType * ancestor, const char *name);
        const char *        getName();
        vrType *            getAncestor();
        vrBool              isDerivedFrom(vrType *);
        void                addFieldInfo(vrFieldInfo * f);
        vrFieldInfo *       getFieldInfo(short i);
        vrFieldInfo *       getFieldInfo(const char * fname);
        int                 getNumFieldInfos();
        vrFieldInfo *       getFieldInfoIndex(int i);
        static vrType *     getType(const char *name);
        static int          getNumTypes();
```

(나) Main Features of the Methods in vrType

getAncestor()
상위 타입을 반환한다.

isDerivedFrom()
지정된 타입으로 부터 유도된 타입인지 유무를 반환해준다.

addFieldInfo()
필드 인포 객체를 삽입한다.

getFieldInfo()
필드 인포 객체를 반환한다.

getNumFieldInfos()
필드 인포 객체가 몇개나 있는지 반환한다.

getType()
필드의 타입을 반환한다.

(2) vrArrays

배열 클래스로서 다음과 같은 배열 클래스가 있다.

```
vrIntArray
vrFloatArray
vrVec2fArray
vrVec3fArray
vrVec4fArray
vrRotationArray
vrStringArray
vrRefArray
```

(가) Class Definition for vrArray

```
vrArray class <- NONE
        void *      operator new(size_t);
        void        operator delete(void *);
        void        setCount(int n);
        void        setSize(int n);
        int                 getCount();
        int                 getSize();
```

(나) Main Features of the Methods in vrArray

operator new() operator delete()
배열을 위한 공간을 할당.해제한다.

setCount() getCount()
배열 원소의 개수를 설정/반환한다.

setSize() getSize()
배열의 크기를 설정/반환한다.

(3) vrField

VRML의 노드를 처리하는 클래스

하위 클래스

vrSFBool
vrSFInt
vrSFFloat
vrSFString
vrSFNode
vrSFVec2f
vrSFVec3f
vrSFRotation

(가) Class Definition for vrField

```
vrField <- NONE
        enum EventClassEnum {
                VR_FIELD,
                VR_EXPOSED_FIELD,
                VR_EVENT_IN,
                VR_EVENT_OUT,
        };
        virtual void            copy(vrField * f) = 0;
        void                    setDirty(int d);
        vrBool                  isDirty() const;
        void                    setUserMode(vrUByte mode);
        vrUByte                 getUserMode();
        void                    setEventClass(EventClassEnum e);
        EventClassEnum          getEventClass();
        vrContainer *           getParent();
        short                   getId();
        const char *            getName();
```

```
void                    validate();
void                    connect(vrField * dst);
void                    disconnect(vrField * dst);
void                    setInput(vrField * f);
vrFieldList *           getInputs();
vrFieldList *           getOutputs();
friend class vrContainer;
friend class vrPrototype;
friend class vrFieldInfo;
friend class vrExFieldInfo;
static void             setValidateCallback(vrFieldCallback);
```

(나) Main Features of the Methods in vrField

copy()

필드를 복사하는 함수

setDirty()

필드의 변경 유무를 설정하는 함수

isDirty()

필드가 변경되어 있는지 반환하는 함수

setUserMode() getUserMode()

사용자의 모드를 설정하고 반환하는 함수

setEventClass() getEventClass()

앞으로 일어날 이벤트의 종류를 설정하고 반환하는 함수

getParent()

부모 필드를 반환하는 함수

getId()

필드의 Id를 반환하는 함수

getName()

필드의 이름을 반환하는 함수

validate()

필드를 유효화한다.

connect() disconnect()

필드간을 연결/연결 해제한다.

setInput()

단일 입력 필드를 설정한다.

getInputs() getOutputs()

복수 입력 필드 리스트를 설정/반환한다.

static void
setValidateCallback(vrFieldCallback);

유효화 루틴을 설정하는 함수

(4) vrMField

복수개의 필드를 가지는 클래스

하위 클래스

vrMFInt class <- vrMField
vrMFFloat
vrMFString
vrMFNode
vrMFVec2f
vrMFVec3f

vrMFRotation

(가) Class Definition for vrMField

```
vrMField class <- vrField
        vrMField(vrContainer * p, short id);
        void                    setCount(int n)
        int                     getCount()
        void                    setSize(int n)
        int                     getSize()
```

(나) Main Features of the Methods in vrMField

vrMField()
컨테이너 노드에 특정 형태의 vrMField를 생성하는 함수

setCount() getCount()
개수를 설정/반환하는 함수

setSize() getSize()
크기를 설정/반환하는 함수

(5) vrFieldInfo

Event가 발생할 경우에 노드의 필드를 접근하는 방법을 제공하는 클래스

하위 클래스

vrSFBoolInfo
vrSFIntInfo
vrSFFloatInfo
vrSFFloatInfo
vrSFStringInfo
vrSFVec2fInfo
vrSFVec3fInfo
vrSFVec3fInfo
vrSFRotationInfo
vrSFNodeInfo
vrMFIntInfo
vrMFFloatInfo
vrMFStringInfo
vrMFVec2fInfo
vrMFVec3fInfo
vrMFVec3fInfo
vrMFRotationInfo
vrMFNodeInfo

(가) Class Definition for vrFieldInfo class

```
vrFieldInfo <- None
        vrFieldInfo(const char * name, short id, char vrContainer::*offset)
        const char *            getName() getId()
        char vrContainer::* getOffset();
        vrType *                getFieldType() ;
        void                    setDirty(vrContainer * parent, int d);
        virtual void *          getStorage(vrContainer * cnt)
        virtual vrField *       instantiate(vrContainer *p) = 0;
```

```
void            addAlias(const char * alias);
int             write(ostream&, vrContainer *);
int             read(istream&, vrContainer *);

friend class vrContainer;
friend class vrField;
```

(나) Main Features of the Methods in vrFieldInfo class

vrFieldInfo()

필드를 설명하는 이름, ID 및 객체 내부의 필드 위치를 이용하여 vrFieldInfo 를 구성하는 구성자.

getName() getId()

필드의 이름과 ID를 반환하는 함수

getOffset()

객체 내부의 필드 위치를 반환하는 함수

getFieldType()

필드의 형을 반환하는 함수

setDirty()

변경 유무를 설정하는 함수

getStorage()

객체의 필드위치를 반환하는 함수

instantiate()

vrContainer 객체안의 필드를 접근하는 vrField 객체를 생성해주는 함수

addAlias()

가명을 추가하는 함수

write()

필드를 쓰는 함수

read()

필드를 읽는 함수

(6) vrName

이름을 처리하는 클래스

(가) Class Definition for vrName

```
vrName class <- NONE
                vrName();
                vrName(const char * s);
                vrName(const vrString & s);
                vrName(const vrName & n);
                const char *            getString();
                int                     getLength();
                int             operator !();
friend int      operator ==(const vrName & n, const char * s)
friend int      operator ==(const char * s, const vrName & n)
friend int      operator ==(const vrName & n1, const vrName & n2)
friend int      operator !=(const vrName & n, const char * s)
friend int      operator !=(const char * s, const vrName & n)
```

(나) Main Features of the Methods in vrName

vrName()

이름을 위한 공간을 vrNameEntry를 이용해서 만든는 함수

getString() getLength()

스트링/길이를 반환

(7) vrNameEntry

이름 클래스를 문자열 테이블로 구성하는 클래스
내부 구현은 해쉬 테이블 구조로 되어 있다.

(가) Class Definition for vrNameEntry

```
vrNameEntry class <- NONE
        int                 isEmpty();
        int                 isEqual(const char * s);
        friend class vrName;
```

(나) Main Features of the Methods in vrNameEntry

isEmpty()

빈 문자열인지 여부를 반환하는 함수

isEqual()

같은 문자열인지 반환하는 함수

다. World Description & Viewing Information

(1) vrPrototype

VRML의 사용자 정의 노드 및 외부 기술 파일을 Import하기 위한 클래스

(가) Class Definition for vrPrototype

```
vrPrototype class <- vrGroup
        enum {
                FIRST = vrGroup::LAST - 1,
                LAST
        };

        virtual void            realize();
        virtual void            draw();
        virtual void            addNode(vrContainer *);
        void                    setRefFieldCount(short);
        short                   getRefFieldCount();
        void                    addRefField(const char *, vrField *);
        vrField *               getRefField(const char *);
        void                    setProtoInfo(class vrProtoInfo *);
        virtual vrContainer *   instantiate();

        friend class vrProtoInfo;
```

(나) Main Features of the Methods in vrPrototype

addNode()

프로토타입에 포함되어 있는 노드를 삽입기위한 함수

setRefFieldCount() getRefFieldCount()

레퍼런스 필드 카운트를 설정/반환하는 함수

addRefField() getRefField()

특정 이름의 레퍼런스 필드를 설정/반환하는 함수

setProtoInfo()

프로토타입을 생성할 vrProtoType을 설정하는 함수

(2) vrProtoInfo

Proto 노드를 생성하기 정보를 관리하는 클래스

(가) Class Definition for vrProtoInfo

```
vrProtoInfo <- vrGroup
        enum {
                FIRST = vrGroup::LAST - 1,
                LAST
        };
        virtual void            realize();
        virtual void            draw();
        static vrProtoInfo *    lookupProtoInfo(const char * name);
        virtual vrContainer *   instantiate();
```

(나) Main Features of the Methods in vrProtoInfo

lookupProtoInfo()

VRML의 노드를 생성할 때 정의되어 있는 ProtoInfo가 존재하는지 검사하는 함수

instantiate()

vrProtoType 노드를 생성하고 VRML화일에 기술되어있는 ROUTE 필드를 연결해주는 함수

(3) vrWorld

브라우저가 관리하는 가상 세계에 대한 정보를 관리하는 클래스

(가) Class Definition for vrWorld

```
vrWorld class <- vrPrototype
        enum {
        FIRST = vrPrototype::LAST - 1,
                WORLDINFO,
                NAVIGATIONINFO,
                AVATARPOSITION,
                AVATARROTATION,
                AVATARORIENTATION,
                LAST
        };
        virtual void            realize();
        virtual void            draw();
        virtual void            addNode(vrContainer *);
        void                    addViewpoint(vrViewpoint *);
        void                    setWorldInfo(vrWorldInfo *);
        void                    setNavigationInfo(vrNavigationInfo *);
        vrWorldInfo *           getWorldInfo();
        vrNavigationInfo *      getNavigationInfo();
        void                    setAvatarPosition(float, float, float);
        void                    setAvatarRotation(float, float, float, float);
```

(나) Main Features of the Methods in vrWorld

addNode()

자식 노드를 추가한다.

- 일반적인 VRML NODE
- 관찰자 노드
- WorldInfo 노드
- vrNavigationInfo 노드

addViewpoint()

관찰자의 시점 노드를 추가한다.

setWorldInfo() getWorldInfo()

vrWorldInfo 객체를 설정 반환한다.

setNavigationInfo() getNavigationInfo()

vrNavigationInfo 객체를 설정 반환한다.

setAvatarPosition() setAvatarRotation()

아바타의 위치/회전을 설정한다.

(4) vrViewer

Scene Graph를 관리하며 Visual 렌더링을 전담하는 클래스

(가) Class Definition for vrViewer

```
vrViewer class <- vrObject
        void            clear();
        void            realize( vrWorld * );
        void            makeView();
        void            swapBuffers();
        void            setDisplayWindow( Display *, Window );
        void            setViewport( int, int, int, int );
        void            setViewCoord( vrVec3f &, vrVec3f & );
        void            setViewMatrix( vrMatrix & );
        vrMatrix *      getViewMatrix();
        void            setFOV(vrFloat);
        void            setNearFar(vrFloat, vrFloat);
        vrGroup *       getScene() const { return _root; }
        vrGroup *       getRoot() const { return _userScene; }
        vrTrackball *   getTrackball() const { return _trackball; }
        void            setMouse( vrMouse & );
        void            resetSim();
        void            continueSim();
        void            pauseSim();
        vrCamera *              getCamera();
        void            setWorld(vrWorld *);
        vrWorld *       getWorld();
```

(나) Main Features of the Methods in vrViewer

clear()

OPENGL의 Depth, Color Frame Buffer내용을 지운다.

realize()

전체 Scene-Graph를 순회하며 그리는 함수

1. realize()함수로 순회
2. PERFORMER로 랜더링할 경우 realize()함수안의 PERFORMER 함수 호출
3. 카메라를 적용한다.
4. OPENGL로 랜더링할 경우 draw()함수로 재순회

5. 버퍼를 스왑하여 결과가 화면에 나타나도록 한다.

makeView()

OPENGL 함수 호출로 PERSPECTIVE View를 설정

setDisplayWindow()

화면을 출력할 Window를 설정하는 함수

setViewport()

OPENGL 함수 호출로 출력 영역을 지정하는 함수

setViewCoord()

좌표축을 설정하는 함수

setViewMatrix() getViewMatrix()

뷰잉 매트릭스를 설정/반환하는 함수

setFOV()

FOV값을 설정하는 함수

setNearFar()

앞/뒤의 클립핑 평면의 거리를 설정한다.

getScene()

Scene Graph의 Root 노드를 반환한다.

getRoot()

사용자 Scene Graph의 Root 노드를 반환한다.

getTrackball()

트랙볼 인터페이스 객체를 반환한다.

setMouse()

트랙볼 인터페이스에 마우스를 연결한다.

resetSim() pauseSim() continueSim()

시뮬레이션을 초기화하고 중지하고 재실행한다.

getCamera()

카메라 객체를 반환한다.

setWorld() getWorld()

vrWorld객체를 설정/반환 한다.

(5) vrCamera

시점, 시야각 및 클리핑 평면을 처리하는 클래스

(가) Class Definition for vrCamera

```
vrCamera class <- vrObject
        void            setPosition( vrFloat, vrFloat, vrFloat );
        void            setPosition( const vrVec3f & pos );
        void            getPosition( vrFloat *, vrFloat *, vrFloat * );
        void            getPosition( vrVec3f & pos );
        void            setOrientation( vrFloat, vrFloat, vrFloat, vrFloat );
        void            setOrientation( const vrRotation & ori );
        void            getOrientation( vrFloat *, vrFloat *, vrFloat *,
                vrFloat * );
        void            getOrientation( vrRotation & ori );
        void            setNearClip( vrFloat );
        vrFloat         getNearClip();
        void            setFarClip( vrFloat );
        vrFloat         getFarClip();
        void            setAspectMode( vrEnum );
        vrEnum          getAspectMode();
        void            getViewMat(vrMatrix & m) { m.copy(_viewMat); }
        void            setViewpoint( vrViewpoint * );
        void            realize();
```

```
void        apply();
#ifdef      PERFORMER
void                set_pfChannel(pfChannel * chan);
pfChannel *         get_pfChannel();
#endif      //PERFORMER
```

(나) Main Features of the Methods in vrCamera

setPosition() getPosition()

카메라의 위치를 설정/반환하는 함수

setOrientation() getOrientation()

카메라의 방향을 설정/반환하는 함수

setNearClip() getNearClip() setFarClip() getFarClip()

클리핑 평면을 설정/반환하는 함수

setAspectMode() getAspectMode()

수직/수평비를 설정/반환하는 함수

getViewMat()

계산된 메트릭스를 반환하는 함수

setViewpoint()

vrViewpoint를 설정하는 함수

apply()

입력된 파라메타를 기반으로 Viewing 매트릭스를 계산하는 함수

set_pfChannel() get_pfChannel()

PERFORMER 구현시에 pfChannel을 설정/반환하는 함수

라. Grouping Nodes

(1) vrGroup

Scene Graph를 구성하기 위해 Graph상의 노드관리 및 노드의 바운딩 박스 관리와 렌더링을 지원하는 클래스

(가) Class Definition for vrGroup

```
vrGroup class <- vrContainer
        virtual void addChild(vrNode *);
        virtual vrNode *        removeChild(int);
        virtual vrNode *        removeChild(vrNode *);
        virtual vrNode *        getChild(int);
        virtual int             getNumChildren();
        void                    realizeChildren();
        virtual void            realize();
        virtual void            setName(const char *);
        void                    setBBoxSize(const vrVec3f & bboxSize);
        void                    getBBoxSize(vrVec3f & bboxSize);
        void                    setBBoxSize(vrFloat v0, vrFloat v1, vrFloat v2);
        void                    getBBoxSize(vrFloat *v0, vrFloat *v1, vrFloat *v2);
        void                    setBBoxCenter(const vrVec3f & bboxCenter);
        void                    getBBoxCenter(vrVec3f & bboxCenter);
        void                    setBBoxCenter(vrFloat v0, vrFloat v1, vrFloat v2);
        void                    getBBoxCenter(vrFloat *v0, vrFloat *v1, vrFloat
*v2);
        void                    calcChildrenBound(vrSphereBound &
    sphereBound);
```

```
void                    calcChildrenBound(vrVec3f & center, vrVec3f &
                        size);
virtual vrContainer *   instantiate();
virtual vrContainer *   clone(vrNode::CloneEnum what);
virtual void            draw();
virtual void            drawGrid();
virtual void            drawBBox();
#ifdef      PERFORMER
void                    set_pfGroup(pfGroup *grp);
pfGroup *               get_pfGroup() const { return _pfgroup; }
virtual pfNode *        get_pfNode() const { return _pfgroup; }
static void             getModelMatrix(pfTraverser *, void *);
#endif      //PERFORMER
friend void __vrRealize(vrContainer *);
```

(나) Main Features of the Methods in vrGroup

addChild() removeChild() getChild()

자식 노드를 삽입/삭제/ 반환한다.

getNumChildren()

자식 노드의 개수를 반환한다.

realizeChildren()

자식 노드를 실제 적용하고 변경된 크기를 재계산해야 할 필요가 있을 경우 바운딩 박스를 재계산한다.

realize()

그룹 노드를 적용한다.

setName()

그룹 노드의 이름을 설정한다.

setBBoxSize() getBBoxSize()

바운딩 박스를 설정/반환한다.

setBBoxCenter() getBBoxCenter()

바운딩 박스의 중앙값을 설정/반환한다.

calcChildrenBound()

자식 노드의 바운딩 박스/구면을 계산한다.

instantiate()

새로운 그룹 노드 개체를 생성한다.

clone()

자신과 같은 새로운 노드 개체를 생성한다.

draw()

자신을 그리는 함수

drawGrid();

지역 좌표계에서 Y평면상에 X축과 Z축상에 그리드를 그린다.

drawBBox();

바운딩 박스를 그린다.

set_pfGroup() get_pfGroup()

PERFORMER구현상에서 pfGroup을 설정/반환한다.

get_pfNode()

PERFORMER구현상에서 pfNode를 반환한다.

getModelMatrix()

PERFORMER구현상에서 모델의 메트릭스를 반환한다.

(2) vrArchor

VRML 파일의 Anchor노드를 지원하기 위한 클래스
URL 링크정보와 외부 세계에 대한 기술등을 기록한다.

(가) Class Definition for vrArchor

```
vrArchor <- vrGroup
        enum {
                FIRST = vrGroup::LAST - 1,
                DESCRIPTION,
                URL,
                PARAMETER,
                LAST
        };
        virtual void            realize();
        virtual void            draw();
        virtual vrContainer *   instantiate();
```

(나) Main Features of the Methods in vrArchor

realize()

그룹 노드를 적용한다.

draw()

랜더링 함수

instantiate()

새로운 그룹 노드 개체를 생성한다.

(3) vrBillboard

빌보드 노드를 구성하기 위한 클래스

(가) Class Definition for vrBillboard

```
vrBillboard <- vrGroup
        enum {
                FIRST = vrGroup::LAST - 1,
                AXISOFROTATION,
                LAST
        };
        void                    setAxisOfRotation(vrVec3f &);
        vrVec3f &               getAxisOfRotation();
        virtual void            realize();
        virtual void            draw();
        virtual vrContainer *   instantiate();
        #ifdef      PERFORMER
        vrBillboard(pfGroup *);
        #endif      //PERFORMER
```

(나) Main Features of the Methods in vrBillboard

setAxisOfRotation() getAxisOfRotation()

빌보드 노드의 회전의 중심축을 설정/반환한다.

realize()

그룹 노드를 적용한다.

draw()

빌보드 노드를 그린다.

instantiate()

새로운 그룹 노드 개체를 생성한다.

vrBillboard(pfGroup *);

PERFORMER 구현시에 pfGroup 구조체로부터 빌보드 노드를 구성한 구성자

(4) vrCollision

VRML의 Collision노드를 지원하는 클래스
노드의 충돌 검사를 한다.

(가) Class Definition for vrCollision

```
vrCollision class <- vrGroup
        enum {
                FIRST = vrGroup::LAST - 1,
                COLLIDE,
                PROXY,
                COLLIDETIME,
                GRAVITY,
                LAST
        };
        void                    setCollide(vrBool);
        vrBool                  getCollide() const;
        void                    setCollideTime(vrTime);
        vrTime                  getCollideTime() const;
        void                    setProxy(vrNode *);
        vrNode *                getProxy();
        virtual void            realize();
        virtual void            draw();
        virtual vrContainer *   instantiate();
```

(나) Main Features of the Methods in vrCollision

setCollide() getCollide()

하위 노드에 대한 충돌 검사를 설정/반환한다.

setCollideTime() getCollideTime()

충돌 시간을 설정/반환한다.

setProxy() getProxy()

충돌 검사의 과부하를 줄이기 위해 보다 간단한 노드를 기술하도록 해주는 프록시의 설정 및 반환해주는 함수

realize()

PERFORMER 구현으로 Intersection Mask를 조정해주는 함수

(5) vrTransform

VRML의 Transform노드를 지원하기 위한 클래스

(가) Class Definition for vrTransform

```
vrTransform class <- vrGroup
        #ifdef      PERFORMER
        vrTransform(pfGroup *);
        #endif      //PERFORMER

        enum {
        FIRST = vrGroup::LAST - 1,
        CENTER,
        TRANSLATION,
        ROTATION,
        SCALE,
        SCALEORIENTATION,
LAST
};

void                setTranslation( float, float, float );
void                setRotation( float, float, float, float );
void                setEuler( float, float, float);
void                setScale( float, float, float );
void                setCenter( float, float, float );

virtual void        realize();
virtual void        draw();
virtual vrContainer *   instantiate();

virtual void        setMatrix( vrMatrix & );
virtual vrMatrix *      getMatPtr() { return & _matrix; }
void                pushWorldMatPtr();

#ifdef      PERFORMER
virtual pfNode *        get_pfNode() const { return _pfgroup; }
virtual void            set_pfDCS(pfDCS *dcs) { set_pfGroup(dcs); }
virtual pfDCS *         get_pfDCS() const   { return (pfDCS*)_pfgroup; }
#endif      //PERFORMER
```

(나) Main Features of the Methods in vrTransform

setTranslation()

이동 변환을 노드에 적용한다.

setRotation()

회전 변환을 노드에 적용한다.

setEuler()

오일러 변환을 노드에 적용한다.

setScale()

크기 변환을 노드에 적용한다.

setCenter()

중심점을 설정한다.

realize()

변환 메트릭스를 계산하고 PERFORMER의 노드 메트릭스에게 복사하는 함수
변경된 메트릭스에 의해서 크기를 재계산한다.

draw()

OPENGL의 매트리스를 변환하는 함수

get_pfNode()

PERFORMER의 pfNode 구조체를 반환하는 함수

set_pfDCS() get_pfDCS()

PERFORMER의 pfDCS를 설정/반환하는 함수

마. Special Groups

(1) vrInline

외부 VRML 화일을 포함시키는 노드

(가) Class Definition for vrInline

```
vrInline class <- vrGroup
        enum {
                FIRST = vrGroup::LAST - 1,
                URL,
                LAST
        };
        virtual void realize();
        virtual void draw();
        virtual vrContainer * instantiate();
```

(나) Main Features of the Methods in vrInline

realize()

기술된 VRML화일을 적재하여 Scene Graph에 포함시킨다.

(2) vrLOD

Level of Detail을 구현하는 클래스

(가) Class Definition for vrLOD

```
vrLOD class <- vrSwitch
        enum {
                FIRST = vrSwitch::LAST - 1,
                LEVEL,
                CENTER,
                RANGE,
                LAST
        };
        virtual void            realize();
        virtual void            draw();
        virtual vrContainer *   instantiate();
        #ifdef      PERFORMER
        virtual void            set_pfLOD(pfLOD *lod) ;
        virtual pfLOD *         get_pfLOD();
        #endif      //PERFORMER
```

(나) Main Features of the Methods in vrLOD

set_pfLOD() get_pfLOD()

PERFOMER의 pfLOD 객체를 설정/반환하는 함수

realize()

PERFORMER의 pfLOD클래스를 이용하여 LOD의 정보를 설정하는 함수

(3) vrSwitch

VRML의 Switch를 구현하는 클래스

(가) Class Definition for vrSwitch

```
vrSwitch class <- vrGroup
        enum {
                FIRST = vrGroup::LAST - 1,
                CHOICE,
                WHICHCHOICE,
                LAST
};
void                    setWhichChoice(int);
int                     getWhichChoice();
virtual void            realize();
virtual void            draw();
virtual vrContainer *   instantiate();

#ifdef      PERFORMER
virtual void            set_pfSwitch(pfSwitch *swt);
virtual pfSwitch *      get_pfSwitch();
#endif      //PERFORMER
```

(나) Main Features of the Methods in vrSwitch

setWhichChoice() getWhichChoice()

Switch할 노드를 설정/반환한다.

realize()

PERFORMER 구현으로 선택이 변경되면 pfSwitch노드의 멤버로 다른 노드를 선택한다.

draw()

vrGroup::draw()를 호출하여 랜더링한다.

바. Common Nodes

(1) vrAudioClip

VRML 화일의 AudioClip을 재생하기 위한 클래스

(가) Class Definition for vrAudioClip

```
vrAudioClip class <- vrContainer
vrBool                  getLoop();
vrFloat                 getPitch();
vrFloat                 getStartTime();
vrFloat                 getSttopTime();
vrString *              getDescription();
vrStringArray *         getUrl();
virtual void realize();
virtual vrContainer * instantiate();
```

(나) Main Features of the Methods in vrAudioClip

getLoop()

오디오 클립이 반복인지 반환한다.

getPitch()

오디오 클립의 피치를 반환한다.

getStartTime() getStopTime()

오디오 클립의 시작 시각과 끝 시각을 반환한다.

getDescription()

오디오 클립의 비고를 반환한다.

getUrl()

URL 스트링 테이블을 반환한다.

realize()

그룹 노드를 적용한다.

instantiate()

새로운 그룹 노드 개체를 생성한다.

(2) vrLight

Light노드를 구현하기 위한 클래스

(가) Class Definition for vrLight

```
vrLight class <- vrNode
        enum {
                FIRST = vrNode::LAST - 1,
                AMBIENTINTENSITY,
                COLOR,
                INTENSITY,
                ON,
                LAST
        };

        virtual void realize();
        virtual void draw();

        #ifdef      PERFORMER
        virtual pfNode *          get_pfNode();
        virtual pfLightSource * get_pfLightSource() ;
        #endif      //PERFORMER
```

(나) Main Features of the Methods in vrLight

realize()

노드의 정보를 이용하여 PERFORMER 구현의 pfLightSource을 설정

draw()

OPENGL의 LIGHT를 설정하여 LIGHT를 적용하는 함수

(3) vrDirectionalLight

VMRL의 지향 조명을 지원하는 클래스

(가) Class Definition for vrDirectionalLight

```
vrDirectionalLight class <- vrLight
        enum {
                FIRST = vrLight::LAST - 1,
                DIRECTION,
                LAST
        };

        virtual void realize();
        virtual void draw();
        virtual vrContainer * instantiate();
```

(나) Main Features of the Methods in vrDirectionalLight

realize()

지향 조명을 적용하는 함수

draw()

OPENGL함수를 이용하여 조명을 설정하는 함수

instantiate()

새로운 vrDirectionalLight 객체를 생성한다.

(4) vrPointLight

VRML의 점 조명을 지원하는 클래스

(가) Class Definition for vrPointLight

```
vrPointLight class <- vrLight
        enum {
                FIRST = vrLight::LAST - 1,
                ATTENUATION,
                LOCATION,
                RADIUS,
                LAST
        };
        virtual void realize();
        virtual void draw();
        virtual vrContainer * instantiate();
```

(나) Main Features of the Methods in vrPointLight

realize()

점 조명을 적용하는 함수

draw()

OPENGL함수를 이용하여 조명을 설정하는 함수

instantiate()

새로운 vrPointLight 객체를 생성한다.

(5) vrSpotLight

VRML의 스폿 조명을 지원하는 클래스

(가) Class Definition for vrSpotLight

```
vrSpotLight class <- vrPointLight
        enum {
                FIRST = vrPointLight::LAST - 1,
                BEAMWIDTH,
                CUTOFFANGLE,
                DIRECTION,
                LAST
        };

        virtual void            realize();
        virtual void            draw();
        virtual vrContainer *   instantiate();
```

(나) Main Features of the Methods in vrSpotLight

realize()

PERFORMER 구현으로 스폿의 Beam Width와 Cut Off Angle를 설정한다.

draw()

OPENGL 구현으로 Cut Off Angle를 설정한다.

instantiate()

새로운 vrSpotLight 객체를 생성한다.

(6) vrScript

VRML 화일 내부의 JAVA Script를 지원하기 위한 클래스

(가) Class Definition for vrScript

```
vrScript class <- vrNode
        enum {
                FIRST = vrNode::LAST - 1,
                URL,
                DIRECTOUTPUT,
                MUSTEVALUATE,
                LAST
        };
        virtual void realize();
        virtual vrContainer * instantiate();
```

(나) Main Features of the Methods in vrScript

(7) vrShape

VRML의 Shape 노드를 지원하기 위한 클래스

(가) Class Definition for vrShape

```
vrShape class <- vrNode
        enum {
        FIRST = vrNode::LAST - 1,
```

```
APPEARANCE,
GEOMETRY,
LAST
};
virtual void            setName(const char *);
vrAppearance *          getAppearance();
vrGeometry *            getGeometry();
virtual void            realize();
virtual void            draw();
virtual vrContainer *   instantiate();
#ifdef      PERFORMER
virtual pfNode *        get_pfNode() const   { return _geode; }
#endif      //PERFORMER
```

(나) Main Features of the Methods in vrShape

setName()

Shape의 이름을 설정하는 함수

getAppearance()

Shape의 외관 객체를 설정하는 함수

getGeometry()

Geometry 노드를 반환하는 함수

realize()

APPEARANCE, GEOMETRY 필드를 갱신하고,
객체가 그려질 스피어 바운드를 재계산하고
PERFORMER의 pfGeoSet 클래스를 초기화한다.

draw()

vrGeometry클래스의 draw함수를 호출한다.

(8) vrSound

VRML의 Sound노드를 지원하기 위한 클래스

(가) Class Definition for vrSound

```
vrSound class <- vrNode
        enum {
                FIRST = vrNode::LAST - 1,
                DIRECTION,
                LOCATION,
                INTENSITY,
                MAXFRONT,
                MINFRONT,
                MAXBACK,
                MINBACK,
                PRIORITY,
                SOURCE,
                SPATIALIZE,
                #ifdef                  USE_EXT_3DSOUND
                // for external sound   module
                POSITION,               // for world coord
                ORIENTATION,            // for world coord (vector form)
                ROTATION,               // for world coord (axial rotation)
                VOLUME,                 // for external sound module
                PLAY,                   // for external sound module
                PLAYTIMES,              // for external sound module
                SOURCEURL,              // for external sound module
                #endif                  //USE_EXT_3DSOUND
                LAST,
```

```
};
virtual void realize();
virtual vrContainer * instantiate();
vrVec3f &               getDirection();
vrVec3f &               getLocation();
vrFloat                 getMaxFront();
vrFloat                 getMinFront();
vrFloat                 getMaxBack();
vrFloat                 getMinBack();
vrFloat                 getPriority();
vrFloat                 getIntensity();
vrAudioClip *           getSource();
static void             updateSoundList(vrComm *, int = 0);
void                    updateInWorld(vrMatrix *);
#ifdef      PERFORMER
void                    set_gsNode(gsNode *dcs);
gsNode *                get_gsNode();
virtual pfNode *        get_pfNode();
#endif      //PERFORMER
```

(나) Main Features of the Methods in vrSound

getDirection() getLocation()

사운드 에미터의 방향과 위치를 반환한다.

getMaxFront() getMinFront() getMaxBack() getMinBack()

getPriority()

사운드의 우선권을 반환한다.

getIntensity()

사운드의 강도를 반환한다.

updateSoundList()

외부 모듈과 통신을 통해서 사운드를 랜더링하는 함수

updateInWorld()

월드 좌표계의 변화를 사운드 소스에 변경하여 외부 모듈 Update가 필요한지 계산하는 함수

(9) vrWorldInfo

VRML WorldInfo 노드를 지원하기 위한 클래스

(가) Class Definition for vrWorldInfo

```
vrWorldInfo : public vrContainer
        enum {
                FIRST       = vrContainer::LAST-1,
                INFO,
                TITLE,
                LAST
        };
        virtual void realize();
        virtual vrContainer * instantiate();
```

(나) Main Features of the Methods in vrWorldInfo

realize()

INFO, TITLE 필드를 갱신한다.

사. Sensors

(1) vrCylinderSensor

VMRL의 CylinerSensor를 지원하기 위한 클래스

(가) Class Definition for vrCylinderSensor

```
vrCylinderSensor class <- vrContainer
enum {
        FIRST      = vrContainer::LAST-1,
        ENABLED,
        AUTOOFFSET,
        OFFSET,
        DISKANGLE,
        MAXANGLE,
        MINANGLE,
        ROTATION_CHANGED,          // eventOut
        TRACKPOINT_CHANGED,        // eventOut
        ISACTIVE,             // eventOut
        LAST
};

vrRotation &          rotation_changed();
vrVec3f &             trackPoint_changed();
int                   isActive();
virtual void          realize();
virtual vrContainer * instantiate();
```

(나) Main Features of the Methods in vrCylinderSensor

rotation_changed()
변경된 회전각을 반환하는 함수

trackPoint_changed()
입력 장치의 위치 정보를 반환하는 함수

isActive()
활성화 되어 있는지 여부를 반환하는 함수

realize()
CylinderSenser를 적용하는 함수

instantiate()
새로운 CylinderSenser 객체를 생성한다.

(2) vrPlaneSensor

VRML의 PlaneSensor를 지원하는 클래스

(가) Class Definition for vrPlaneSensor

```
vrPlaneSensor class <- vrContainer
        enum {
                FIRST      = vrContainer::LAST-1,
                ENABLED,
                AUTOOFFSET,
```

```
                OFFSET,
                MAXPOSITION,
                MINPOSITION,
                TRANSLATION_CHANGED,      // eventOut
                TRACKPOINT_CHANGED,       // eventOut
                ISACTIVE,                 // eventOut
                LAST
};
vrVec3f &               translation_changed();
vrVec3f &               trackPoint_changed();
int                     isActive();
virtual void            realize();
virtual vrContainer *   instantiate();
```

(나) Main Features of the Methods in vrPlaneSensor

translation_changed()

변경된 이동량을 반환하는 함수

trackPoint_changed()

입력 장치의 위치 정보를 반환하는 함수

isActive()

활성화 되어 있는지 여부를 반환하는 함수

realize()

PlaneSenser를 적용하는 함수

instantiate()

새로운 PlaneSensor 객체를 생성한다.

(3) vrProximitySensor

VRML의 ProximitySensor를 지원하는 클래스

(가) Class Definition for vrProximitySensor

```
vrProximitySensor class <- vrNode
        enum {
                FIRST       = vrNode::LAST-1,
                ENABLED,
                CENTER,
                SIZE,
                ENTERTIME,
                EXITTIME,
                POSITION_CHANGED,           // eventOut
                ORIENTATION_CHANGED,        // eventOut
                ISACTIVE,                   // eventOut
                LAST
        };
        vrVec3f &               position_changed();
        vrRotation &            orientation_changed();
        int                     isActive();
        int                     isRegion(vrVec3f&, vrVec3f&,    vrVec3f&);
        virtual void            realize();
        virtual vrContainer *   instantiate();
```

(나) Main Features of the Methods in vrProximitySensor

position_changed()

변경된 위치 정보를 반환하는 함수

orientation_changed()

변경된 지향 정보를 반환하는 함수

isActive()

활성화 되어 있는지 여부를 반환하는 함수

isRegion()

vPos 위치 벡터가 Center와 Size로 구성된 입방체안의 공간에 있는 여부를 반환하는 함수

(4) vrSphereSensor

VRML의 SphereSensor를 지원하는 클래스

(가) Class Definition for vrSphereSensor

```
vrSphereSensor class <- vrContainer
        enum {
                FIRST      = vrContainer::LAST-1,
                ENABLED,
                AUTOOFFSET,
                OFFSET,
                ROTATION_CHANGED,          // eventOut
                TRACKPOINT_CHANGED,        // eventOut
                ISACTIVE,                  // eventOut
                LAST
        };
        vrRotation &          rotation_changed();
        vrVec3f &             trackPoint_changed();
        int                   isActive();
        virtual void          realize();
        virtual vrContainer * instantiate();
```

(나) Main Features of the Methods in vrSphereSensor

rotation_changed()

변경된 회전각을 반환하는 함수

trackPoint_changed()

변경된 이동량을 반환하는 함수

isActive()

활성화 되어 있는지 여부를 반환하는 함수

(5) vrTimeSensor

VRML의 SphereSensor를 지원하는 클래스

(가) Class Definition for vrTimeSensor

```
vrTimeSensor class <- vrNode
        enum {
                FIRST = vrNode::LAST - 1,
                CYCLEINTERVAL,
                ENABLED,
                LOOP,
                STARTTIME,
                STOPTIME,
```

```
                FRACTION,                       // eventOut
                CYCLETIME,                      // eventOut
                ISACTIVE,           // eventOut
                TIME,                           // eventOut
                LAST
        };
        vrFloat             getFractionChanged();
        int                 getIsActive();
        vrTime              getTime();
        void                setCycleInterval(vrTime);
        void                setStartTime(vrTime);
        void                setStopTime(vrTime);
        virtual void        realize();
        virtual vrContainer *   instantiate();
```

(나) Main Features of the Methods in vrTimeSensor

getFractionChanged()

시간이 변경되었는지 여부를 반환하는 함수

getIsActive()

활성화 되어 있는지 여부를 반환하는 함수

getTime()

현 시간을 반환하는 함수

setCycleInterval()

반복 구간을 설정하는 함수

setStartTime() setStopTime()

시작 시간과 끝시간을 설정하는 함수

(6) vrTouchSensor

VRML의 vrTouchSensor를 지원하는 함수

(가) Class Definition for vrTouchSensor

```
vrTouchSensor class <- vrContainer
        enum {
                FIRST       = vrContainer::LAST-1,
                ENABLED,
                HITNORMAL_CHANGED,          // eventOut
                HITPOINT_CHANGED,           // eventOut
                HITTEXCOORD_CHANGED,        // eventOut
                ISOVER,                     // eventOut
                ISACTIVE,           // eventOut
                TOUCHTIME,                  // eventOut
                LAST
        };
        vrVec3f &               hitNormal_changed();
        vrVec3f &               hitPoint_changed();
        vrVec2f &               hitTexCoord_changed();
        int                     isActive();
        int                     isOver();
        vrTime                  touchTime();
        void                    setIsOver(vrBool);
        virtual void            realize();
        virtual vrContainer *   instantiate();
```

(나) Main Features of the Methods in vrTouchSensor

hitNormal_changed()
입력장치의 지점의 노멀값이 변경된 경우 호출하는 함수
hitPoint_changed()
입력장치의 지점의 위치값이 변경된 경우 호출하는 함수
hitTexCoord_changed()
입력장치의 지점의 텍스처 위치값이 변경된 경우 호출하는 함수
isActive()
활성화 되어 있는지 여부를 반환하는 함수
isOver()
입력장치의 지점이 노드위에 존재하는지 여부를 반환하는 함수
touchTime()
입력장치가 노드를 touch한 시간을 반환하는 함수
setIsOver()
입력장치의 지점이 노드위에 존재하는지 여부를 설정하는 함수
realize()
각 필드를 갱신하는 함수

(7) vrVisibilitySensor

VRML의 VisibilitySensor노드를 지원하기 위한 클래스

(가) Class Definition for vrVisibilitySensor

```
vrVisibilitySensor class <- vrContainer
        enum {
        FIRST      = vrContainer::LAST-1,
        ENABLED,
        CENTER,
        SIZE,
        ENTERTIME,
        EXITTIME,
        ISACTIVE,          // eventOut
        LAST
};
int                 isActive();
virtual void        realize();
virtual vrContainer *   instantiate();
```

(나) Main Features of the Methods in vrVisibilitySensor

isActive()
활성화 되어 있는지 여부를 반환하는 함수

아. Geometry

(1) vrGeometry

Geometry를 관리 하는 클래스

vrGeometry 의 하위 클래스
vrBox

vrCone
vrCylinder
vrSphere

(가) Class Definition for vrGeometry

```
vrGeometry class <- vrContainer
        void                    setBoxBound(const vrBoxBound & boxBound);
        void                    getBoxBound(vrBoxBound & boxBound);
        virtual void            draw() = 0;
        virtual void            drawBound();
        #ifdef      PERFORMER
        pfGeoSet *              get_pfGeoSet();
        #endif      //PERFORMER
```

(나) Main Features of the Methods in vrGeometry

draw()

노드를 그린다.

get_pfGeoSet()

PERFORMER 구현시에 pfGeoSet 구조체를 반환한다.

(2) vrBox

VRML의 BOX 노드를 구현하기위한 클래스

(가) Class Definition for vrBox

```
vrBox class <- vrGeometry
        enum {
                FIRST = vrGeometry::LAST - 1,
                SIZE,
                LAST
        };
        virtual void realize();
        virtual void draw();
        virtual vrContainer * instantiate();
```

(나) Main Features of the Methods in vrBox

realize()

바운드 박스의 크기를 계산하고, SIZE 필드를 touchUpdate 함수를 통하여 변경하는 함수
PERFOMER 구현시에 _geoset을 구성한다.

instantiate()

새로운 박스 노드를 생성한다.

(3) vrCone

VRML의 CONE 노드를 구현하기위한 클래스

(가) Class Definition for vrCone

```
vrCone class <- vrGeometry
        enum {
                FIRST = vrGeometry::LAST-1,
                BOTTOMRADIUS,
```

```
                HEIGHT,
                SIDE,
                BOTTOM,
                LAST
        };
        virtual void realize();
        virtual void draw();
        virtual vrContainer * instantiate();
```

(나) Main Features of the Methods in vrCone

realize()

바운드 박스의 크기를 계산하고, 각 필드의 Event를 발생시키고, 크기에 관련된 정보를 이용하여 메트릭스를 구성한다.

draw()

OPENGL의 함수를 호출하여 Cone을 그린다.

instantiate()

새로운 vrCone 노드를 생성한다.

(4) vrCylinder

VRML의 Cylinder노드를 구현하기위한 클래스

(가) Class Definition for vrCylinder

```
vrCylinder class <- vrGeometry
        enum {
                FIRST = vrGeometry::LAST - 1,
                RADIUS,
                HEIGHT,
                SIDE,
                TOP,
                BOTTOM,
                LAST
        };
        virtual void            realize();
        virtual void            draw();
        virtual vrContainer *   instantiate();
```

(나) Main Features of the Methods in vrCylinder

realize()

바운드 박스의 크기를 계산하고, 각 필드의 Event를 발생시키고, 크기에 관련된 정보를 이용하여 메트릭스를 구성한다.

draw()

OPENGL의 함수를 호출하여 Cylinder를 그린다.

instantiate()

새로운 vrCylinder 노드를 생성한다.

(5) vrSphere

VRML의 Sphere 노드를 지원하기 위한 클래스

(가) Class Definition for vrSphere

```
vrSphere class <- vrGeometry
        enum {
                FIRST = vrGeometry::LAST - 1,
                RADIUS,
                LAST
        };
        virtual void            realize();
        virtual void            draw();
        virtual vrContainer *   instantiate();
```

(나) Main Features of the Methods in vrSphere

realize()

PERFORMER의 pfdNewSphere()로만들어진 구를 크기 조절한다.

draw()

OPENGL 구현으로 구를 그린다.

(다) vrPointSet

VRML의 PointSet을 지원하는 클래스

① **Class Definition for vrPointSet**

```
vrPointSet class <- vrGeometry
        enum {
                FIRST = vrGeometry::LAST-1,
                COORD,
                COLOR,
                LAST
        };

        virtual void            realize();
        virtual void            draw();
        virtual vrContainer *   instantiate();
        virtual void            calcBound(vrBoxBound & boxBound);
```

② **Main Features of the Methods in vrPointSet**

realize()

COORD, COLOR를 갱신한다.

(6) vrPolySet

VRML의 ElevationGrid, Extrusion, IndexedFaceSet을 지원하기 위한 상위 클래스
vrPolyset 의 하위 클래스

vrElevationGrid
vrExtrusion
vrIndexedFaceSet

① **Class Definition for vrPolySet**

```
vrPolySet class <- vrGeoSet
        enum {
                FIRST = vrGeoSet::LAST-1,
                LAST
```

```
};
virtual void        realize();
virtual void        calcBound(vrBoxBound & boxBound);
```

② **Main Features of the Methods in vrPolySet**

realize()

vrGeoSet::realize()를 호출하여 집합 정보를 구성한다.

calcBound(vrBoxBound & boxBound)

바운드 박스의 크기를 계산한다.

(7) vrElevationGrid

VRML의 ElevationGrid를 지원하기 위한 클래스

(가) Class Definition for vrElevationGrid

```
vrElevationGrid class <- vrPolySet
        enum {
                FIRST = vrPolySet::LAST-1,
                COLORPERVERTEX,
                NORMALPERVERTEX,
                CCW,
                SOLID,
                CREASEANGLE,
                HEIGHT,
                XDIMENSION,
                XSPACING,
                ZDIMENSION,
                ZSPACING,
                LAST
        };

        void                setCCW(vrBool ccw);
        vrBool              getCCW();
        void                setSolid(vrBool solid);
        vrBool              getSolid();
        void                setCreaseAngle(vrFloat creaseAngle);
        vrFloat             getCreaseAngle();
        void                setColorPerVertex(vrBool colorPerVertex);
        vrBool              getColorPerVertex();
        void                setNormalPerVertex(vrBool normalPerVertex);
        vrBool              getNormalPerVertex();

        virtual void        realize();
        virtual void        draw();
        virtual vrContainer * instantiate();
```

(나) Main Features of the Methods in vrElevationGrid

setCCW() getCCW()

반시계 방향 여부를 설정/반환하는 함수

setSolid() getSolid()

솔리드 여부를 설정/반환하는 함수

setCreaseAngle() getCreaseAngle()

증분 각도를 설정/반환하는 함수

setColorPerVertex() getColorPerVertex()

정점당 컬러를 설정할 것인지 여부를 설정/반환하는 함수

setNormalPerVertex() getNormalPerVertex()

정점당 노멀 벡터를 절성할 것인지 여부를 설정/반환하는 함수

realize()

각 필드의 이벤트를 설정하고 바운딩 박스의 크기를 계산한다.

draw()

OPENGL의 함수를 이용해서 그림을 그린다.

instantiate()

새로운 vrElevationGrid 객체를 반환하는 함수

(8) vrExtrusion

VMRL의 Extrusion 노드를 지원하기 위한 클래스

(가) Class Definition for vrExtrusion

```
vrExtrusion class <- vrPolySet
        enum {
                FIRST = vrPolySet::LAST-1,
                CCW,
                CONVEX,
                SOLID,
                CREASEANGLE,
                BEGINCAP,
                ENDCAP,
                CROSSSECTION,
                ORIENTATION,
                SCALE,
                SPLINE,
                LAST
        };

        void                    setCCW(vrBool ccw);
        vrBool                  getCCW();
        void                    setSolid(vrBool solid);
        vrBool                  getSolid();
        void                    setConvex(vrBool convex);
        vrBool                  getConvex();
        void                    setCreaseAngle(vrFloat creaseAngle);
        vrFloat                 getCreaseAngle();
        void                    setBeginCap(vrBool beginCap);
        vrBool                  getBeginCap();
        void                    setEndCap(vrBool endCap);
        vrBool                  getEndCap();

        virtual void            realize();
        virtual void            draw();
        virtual vrContainer *   instantiate();
```

(나) Main Features of the Methods in vrExtrusion

setCCW() getCCW()

반시계 방향 여부를 설정/반환하는 함수

setSolid() getSolid()

솔리드 여부를 설정/반환하는 함수

setConvex() getConvex()

볼록 다각형인지 여부를 설정/반환하는 함수

setCreaseAngle() getCreaseAngle()
증분 각도를 설정/반환하는 함수
setBeginCap() getBeginCap()
시작위치의 Cap여부를 설정/반환하는 함수
setEndCap() getEndCap()
끝위치의 Cap여부를 설정/반환하는 함수

realize()
각 필드의 이벤트를 설정하고 바운딩 박스의 크기를 계산한다.
draw()
OPENGL의 함수를 이용해서 그림을 그린다.
instantiate()
새로운 vrElevationGrid 객체를 반환하는 함수

(9) vrIndexedFaceSet

VRML의 IndexedFaceSet노드를 지원하기 위한 클래스
정점 집합과 인덱스 집합을 유지하여 면을 구성

(가) Class Definition for vrIndexedFaceSet

```
vrIndexedFaceSet class <- vrPolySet
        vrIndexedFaceSet();
        virtual ~vrIndexedFaceSet();
        enum {
                FIRST = vrPolySet::LAST-1,
                COLORPERVERTEX,
                NORMALPERVERTEX,
                CCW,
                CONVEX,
                SOLID,
                CREASEANGLE,
                LAST
};

void            setCCW(vrBool ccw);
vrBool          getCCW();
void            setSolid(vrBool solid);
vrBool          getSolid();
void            setConvex(vrBool convex);
vrBool          getConvex();
void            setCreaseAngle(vrFloat creaseAngle);
vrFloat         getCreaseAngle();
void            setColorPerVertex(vrBool colorPerVertex);
vrBool          getColorPerVertex();
void            setNormalPerVertex(vrBool normalPerVertex);
vrBool          getNormalPerVertex();

virtual void    realize();
virtual void    draw();
virtual vrContainer *   instantiate();
```

(나) Main Features of the Methods in vrIndexedFaceSet

setCCW() getCCW()
반시계 방향 여부를 설정/반환하는 함수

setSolid() getSolid()
솔리드 여부를 설정/반환하는 함수
setCreaseAngle() getCreaseAngle()
증분 각도를 설정/반환하는 함수
setColorPerVertex() getColorPerVertex()
정점당 컬러를 설정할 것인지 여부를 설정/반환하는 함수
setNormalPerVertex() getNormalPerVertex()
정점당 노멀 벡터를 절성할 것인지 여부를 설정/반환하는 함수
realize()
각 필드의 이벤트를 설정하고 바운딩 박스의 크기를 계산한다.
draw()
OPENGL의 함수를 이용해서 그림을 그린다.
instantiate()
새로운 vrIndexedFaceSet 객체를 반환하는 함수

(10) vrLineSet

라인 집합을 구성하는 함수

(가) Class Definition for vrLineSet

```
vrLineSet class <- vrGeoSet
        enum {
                FIRST = vrGeoSet::LAST-1,
                WIDTH,
                LAST
        };
        virtual void        realize();
        virtual void        calcBound(vrBoxBound & box);
```

(나) Main Features of the Methods in vrLineSet

realize()
라인의 배열을 기술된 값으로 초기화하는 함수
calcBound(vrBoxBound & box)
라인 집합의 바운드 박스를 구하는 함수

(11) vrIndexedLineSet

VRML의 IndexedLineSet을 지원하기 위한 클래스
정점 집합과 인덱스 집합을 유지하여 라인을 구성한다.

(가) Class Definition for vrIndexedLineSet

```
vrIndexedLineSet class <- vrLineSet
        enum {
                FIRST = vrLineSet::LAST-1,
                COORDINDEX,
                COLORINDEX,
                COLORPERVERTEX,
                LAST
        };
        virtual void        realize();
```

```
virtual void        draw();
virtual vrContainer *   instantiate();
```

(나) Main Features of the Methods in vrIndexedLineSet

realize()

각 필드의 이벤트를 갱신하는 함수

draw ()

OPENGL의 함수를 이용하여 선분을 그리는 함수

(12) vrText

VRML의 Text노드를 지원하는 클래스

(가) Class Definition for vrText

```
vrText class <- vrGeoSet
        enum {
                FIRST = vrGeoSet::LAST - 1,
                STRING,
                FONTSTYLE,
                LENGTH,
                MAXEXTENT,
                LAST
        };

        virtual void            calcBound(vrBoxBound & box);
        virtual void            realize();
        virtual void            draw();
        virtual vrContainer * I   nstantiate();
```

(나) Main Features of the Methods in vrText

calcBound()

구현 되어 있지 않음

realize()

FONTSTYLE, LENGTH, MAXEXTENT 필드를 갱신하고 vrGeoSet::realize()를 호출한다.

자. Geometric Properties

(1) vrGeoSet

지오메트리 집합을 지원하는 클래스

다음과 같은 집합을 처리한다.

COORD	좌표값
NORMAL	노멀값
COLOR	색상값
TEXCOORD	텍스처 좌표값
COORDINDEX	좌표 인덱스

NORMALINDEX 노멀 인덱스
COLORINDEX 색상 인덱스
TEXCOORDINDEX 텍스처 인덱스

(가) Class Definition for vrGeoSet

```
vrGeoSet class <- vrGeometry
        enum {
                FIRST = vrGeometry::LAST-1,
                COORD,
                NORMAL,
                COLOR,
                TEXCOORD,
                COORDINDEX,
                NORMALINDEX,
                COLORINDEX,
                TEXCOORDINDEX,
                LAST
        };
        virtual void        realize();
        virtual void        calcBound(vrBoxBound & boxBound) = 0;
```

(나) Main Features of the Methods in vrGeoSet

realize()

각 값의 개수를 알아내고 값을 추가하여 초기화한다.

(2) vrColorSet

VRML의 Color노드를 구현한 클래스

(가) Class Definition for vrColorSet

```
vrColorSet class <- vrContainer
        int                 getNumColors();
        vrVec3f *           getColor();
        vrVec3f *           color();

        virtual void realize();
        virtual vrContainer * instantiate();
```

(나) Main Features of the Methods in vrColorSet

getNumColors()

기술된 Color의 개수를 반환한다.

getColor() color()

Colordml 배열을 반환한다.

realize()

COLOR필드를 갱신하는 함수

instantiate()

vrColorSet 객체를 반환하는 함수

(3) vrCoordSet

VRML의 Coordinate노드를 지원하기 위한 클래스

(가) Class Definition for vrCoordSet

```
vrCoordSet class <- vrContainer

        enum {
                FIRST       = vrContainer::LAST-1,
                POINT,
                LAST
        };

        int                             getNumPoints();
        vrVec3f *               getPoint();
        vrVec3f *               point();

        virtual void realize();
        virtual vrContainer * instantiate();
```

(나) Main Features of the Methods in vrCoordSet

getNumPoints()
Point의 갯수를 반환한다.

getPoint() point()
Point의 배열을 반환한다.

realize()
POINT 이벤트를 설정하여 CoordSet이 적용되게 한다.

instantiate()
새로운 CoordSet 객체를 생성한다.

(4) vrNormalSet

VRML의 Normal노드를 지원하기위한 클래스

(가) Class Definition for vrNormalSet

```
vrNormalSet class <- vrContainer
        enum {
                FIRST       = vrContainer::LAST-1,
                VECTOR,
                LAST
        };

        int                             getNumVectors();
        vrVec3f *               getVector();
        vrVec3f *               vector();
        virtual void realize();
        virtual vrContainer * instantiate();
```

(나) Main Features of the Methods in vrNormalSet

getNumVectors()

벡터의 갯수를 반환한다.

getVector() vector()

벡터를 반환한다.

realize()

필드를 갱신한다.

(5) vrTexCoordSet

VRML의 TextureCoordinate 노드를 지원하는 클래스

(가) Class Definition for vrTexCoordSet

```
vrTexCoordSet class <- vrContainer
        enum {
                FIRST     = vrContainer::LAST-1,
                POINT,
                LAST
        };

        int                     getNumPoints() ;
        vrVec2f *               getPoint() ;
        vrVec2f *               point();

        virtual void realize();
        virtual vrContainer * instantiate();
```

(나) Main Features of the Methods in vrTexCoordSet

getNumPoints()

기술된 Points의 개수를 반환한다.

getPoint() point()

Point의 배열를 반환한다.

realize()

POINT 필드를 갱신하는 함수

instantiate()

vrTexCoordSet 객체를 반환하는 함수

(6) vrAppearance

VRML의 노드의 외부 형상을 기술하는 클래스이다.

(가) Class Definition for vrAppearance

```
vrAppearance class <- vrContainer
        virtual void realize();
        virtual void apply();
        virtual vrContainer * instantiate();
        #ifdef      PERFORMER
        pfGeoState *            get_pfGeoState()
        #endif      //PERFORMER
```

(나) Main Features of the Methods in vrAppearance

realize()

그룹 노드를 적용한다.

apply()

그룹 노드를 적용하기위해 재질, 텍스처, 택스처 트랜스폼을 적용한다

instantiate()

새로운 그룹 노드 개체를 생성한다.

get_pfGeoState()

PERFORMER구현상에서 pfGeoState를 반환한다.

(7) vrFontStyle

폰트의 형태를 기술하기 위한 클래스

(가) Class Definition for vrFontStyle

```
vrFontStyle class <- vrContainer
        enum {
                FIRST      = vrContainer::LAST-1,
                FAMILY,
                LANGUAGE,
                STYLE,
                JUSTIFY,
                HORIZONTAL,
                LEFTTORIGHT,
                TOPTOBOTTOM,
                SIZE,
                SPACING,
                LAST
        };
        virtual void              realize();
        virtual void              apply();
        virtual vrContainer *     instantiate();
```

(나) Main Features of the Methods in vrFontStyle

realize()

각 필드가 반영되도록 이벤트를 발생한다.

instantiate()

vrFontStyle 객체를 생성하여 반환한다.

(8) vrMaterial

재질감을 명시하는 클래스

(가) Class Definition for vrMaterial

```
vrMaterial class <- vrContainer
enum {
        FIRST      = vrContainer::LAST-1,
        AMBIENTINTENSITY,
        SHININESS,
        TRANSPARENCY,
        DIFFUSECOLOR,
        EMISSIVECOLOR,
        SPECULARCOLOR,
        LAST
};
```

```
void                    setDiffuseColor( float, float, float );
void                    setEmissiveColor( float, float, float );
void                    setSpecularColor( float, float, float );
virtual void            realize();
virtual void            apply();
virtual vrContainer *   instantiate();
#ifdef      PERFORMER
pfMaterial *get_pfMaterial() { return _pfmat; }
#endif      //PERFORMER
friend classvrAppearance;
```

(나) Main Features of the Methods in vrMaterial

setDiffuseColor() setEmissiveColor() setSpecularColor()

재질의 Diffuse, Emissive, Specular 색상을 설정하는 함수

realize()

PERFOMER의 pfMaterial 클래스를 이용하여 빛에 대한 반사 정도를 설정하는 함수

apply()

OPENGL의 함수를 이용하여 빛에 대한 반사 정도를 설정하는 함수

(9) vrTexture

VRML의 ImageTexture PixelTexture MovieTexture 노드의 상위 클래스
vrTexture의 하위 클래스

vrImageTexture
vrPixelTexture
vrMovieTexture

(가) Class Definition for vrTexture

```
vrTexture class <- vrContainer
        enum {
                FIRST       = vrContainer::LAST-1,
                REPEATS,
                REPEATT,
                LAST
        };

        virtual void            realize();
        virtual void            apply();
        virtual void            bindAppearance(vrAppearance *);
        #ifdef      PERFORMER
        pfTexture *             get_pfTexture();
        #endif      //PERFORMER
```

① **Main Features of the Methods in vrTexture**

realize()

PERFORMER의 구현으로 텍스처의 반복을 설정하는 함수

bindAppearance(vrAppearance *)

PERFORMER의 구현으로 텍스처의 유무를 설정하는 함수

(10) vrImageTexture

VRML의 ImageTexture노드를 지원하기 위한 클래스

(가) Class Definition for vrImageTexture

```
vrImageTexture class <- vrTexture
enum {
FIRST = vrTexture::LAST - 1,
URL,
LAST
};
virtual void realize();
virtual vrContainer * instantiate();
```

(나) Main Features of the Methods in

realize()
PERFORMER구현시에 텍스처 메모리를 읽어드린다.

(11) vrPixelTexture

VRML의 PixelTexture를 지원하는 클래스

(가) Class Definition for vrPixelTexture

```
vrPixelTexture class <- vrTexture
        enum {
                FIRST = vrTexture::LAST - 1,
                IMAGE,
                LAST
        };
        virtual void realize();
        virtual vrContainer * instantiate();
```

(나) Main Features of the Methods in vrPixelTexture

realize()

필드를 갱신하고 vrTexture::realize()를 호출하여 텍스춰를 구성한다

(12) vrMovieTexture

VRML의 MovieTexture를 지원하기 위한 노드

(가) Class Definition for vrMovieTexture

```
vrMovieTexture class <- vrTexture
enum {
        FIRST = vrTexture::LAST - 1,
        URL,
        LOOP,
        SPEED,
        STARTTIME,
        STOPTIME,
        LAST
};
virtual void realize();
```

```
virtual vrContainer * instantiate();
```

(나) Main Features of the Methods in vrMovieTexture

realize()

현재 지원되지 않는다.

(13) vrTextureTransform

VRML의 TextureTransform을 구현한 클래스

(가) Class Definition for vrTextureTransform

```
vrTextureTransform class <- vrContainer
        enum {
                FIRST       = vrContainer::LAST-1,
                CENTER,
                ROTATION,
                SCALE,
                TRANSLATION,
                LAST
        };
        virtual void            realize();
        virtual void            apply();
        virtual vrContainer *   instantiate();
```

(나) Main Features of the Methods in vrTextureTransform

차. Interpolators

(1) vrInterpolator

객체의 다양한 애니메이션을 위한 보간법을 제공하는 클래스

하위 클래스

vrColorInterpolator: COLOR를 보간해주는 클래스
vrCoordinateInterpolator: 좌표를 보간해주는 클래스
vrNormalInterpolator: 법선의 좌표를 보간해주는 클래스
vrOrientationInterpolator: 방향 선분을 보간해주는 클래스
vrScalarInterpolator: 값을 보간해주는 클래스

(가) Class Definition for vrInterpolator

```
vrInterpolator class <- vrNode
        int             getNumKeys();
        vrFloat *       getKey();
        vrFloat *       key();
        void            setFraction(float);
        vrFloat         getFraction();
        virtual void    realize();
        virtual void    eval() = 0;
```

(나) Main Features of the Methods in vrInterpolator

getNumKeys()
보간법을 적용하기 위한 키값의 개수를 반환한다.
getKey() key()
키값의 배열을 반환한다.
setFraction(float) getFraction()
보간하기 위한 값을 설정 반환한다.
realize()
보간을 적용한다.
eval()
보간 값을 계산한다.

카. Binadable Nodes

(1) vrBindable

랜더러가 적용할 객체인지 판단해주는 클래스

(가) Class Definition for vrBindable

```
vrBindable class <- vrContainer
        enum {
                FIRST      = vrBindable::LAST-1,
                COLOR,
                VISIBILITYRANGE,
                FOGTYPE,
                LAST
        };
        virtual void setBind(vrBool)
        virtual vrBool          getBind();
        void                    setIsBound(vrBool);
        virtual void realize();
        friend class vrCamera;
```

(나) Main Features of the Methods in vrBindable

setBind() getBind()
Bind 유무를 결정/반환하는 가상 함수
setIsBound()
Bind 유무를 설정하고 ISBOUND 필드에 이벤트를 전달하는 함수
realize()
ISBOUND 필드를 touchUpdate 함수를 통하여 변경하는 함수

(2) vrBackground

가상 세계의 배경을 기술하기 위한 클래스

(가) Class Definition for vrBackground

```
vrBackground class <- vrBindable
        enum {
                FIRST = vrBindable::LAST-1,
                GROUNDANGLE,
                GROUNDCOLOR,
                SKYANGLE,
```

```
                    SKYCOLOR,
                    BACKURL,
                    BOTTOMURL,
                    FRONTURL,
                    LEFTURL,
                    RIGHTURL,
                    TOPURL,
                    LAST
};
virtual vrBindableStack &       getBindableStack();
virtual void                    realize();
virtual vrContainer *           instantiate();
```

(나) Main Features of the Methods in vrBackground

getBindableStack()
배경을 기술하기위한 연결 스택을 반환한다.

realize()
그룹 노드를 적용한다.

instantiate()
새로운 그룹 노드 개체를 생성한다.

(3) vrFog

안개 효과를 기술하기 위한 클래스

(가) Class Definition for vrFog

```
vrFog class <- vrBindable
enum {
        FIRST       = vrBindable::LAST-1,
        COLOR,
        VISIBILITYRANGE,
        FOGTYPE,
        LAST
};
void                    setColor(float, float, float);
void                    getColor(vrVec3f &);
void                    setFogType(const char *) { ; }
vrString &              getFogType() { return _fogType; }
void                    setVisibilityRange(float v) { _visibilityRange = v; }
vrFloat                 getVisibilityRange() { return _visibilityRange; }
virtual vrBindableStack &
                        getBindableStack() { return _fogStack; }
virtual void            realize();
virtual vrContainer *   instantiate();
```

(나) Main Features of the Methods in vrFog

setColor() getColor()
안개의 색상을 지정/반환하는 함수

setFogType() getFogType()
안개의 형태를 설정/반환하는 함수
setVisibilityRange() getVisibilityRange()
안개가 시야에 미치는 영역을 설정/반환하는 함수

(4) vrNavigationInfo

브라우저에서 사용될 네비게이션 정보를 담고 있는 클래스

(가) Class Definition for vrNavigationInfo

```
vrNavigationInfo class <- vrBindable
        enum {
                FIRST       = vrBindable::LAST-1,
                AVATARSIZE,
                HEADLIGHT,
                SPEED,
                VISIBILITYLIMIT,
                TYPE,
                LAST
        };
        vrFloatArray &          getAvatarSize();
        vrBool                  getHeadLight();
        vrFloat                 getSpeed();
        vrFloat                 getVisiblityLimit();
        vrString &              getNavType(int i);
        int                     getNavTypeCount();
        virtual vrBindableStack & getBindableStack();
        virtual void            realize();
        virtual vrContainer *   instantiate();
```

(나) Main Features of the Methods in vrNavigationInfo

getAvatarSize()
아바타의 크기를 반환한다.
getHeadLight()
헤드라이트를 켜는지 여부를 반환한다.
getSpeed()
네비게이션 스피드를 반환한다.
getVisiblityLimit()
시야 한계를 반환한다.
getNavType()
네비게이션 타입을 반환한다.
getNavTypeCount()
네비게이션의 종류를 반환한다.
getBindableStack()
네비게이션 정보 스택을 반환한다.
realize()
필드를 갱신한다.

(5) vrViewpoint

관찰자의 정보를 관리하는 클래스

(가) Class Definition for vrViewpoint

```
vrViewpoint class <- vrBindable
        enum {
                FIRST       = vrBindable::LAST-1,
                FIELDOFVIEW,
                JUMP,
                POSITION,
                ORIENTATION,
                DESCRIPTION,
                LAST
        };

        virtual void            setFOV(float);
        virtual void            getFOV(float &);
        virtual void            getCoord(vrVec3f &, vrVec3f &);
        void                    setPosition(float, float, float);
        void                    setOrientation(float, float, float, float);
        void                    getPosition(vrVec3f &);
        void                    getOrientation(vrRotation &);
        const char *            getDescription();
        virtual void            setBind(vrBool);
        static vrBindableStack &        getBindableStack();
        static int              getNumStack();
        virtual int             isInStack();
        static vrViewpoint *    topStack();
        static void             pushStack(vrViewpoint *);
        static vrViewpoint *    popStack();
        virtual void            realize();
        virtual vrContainer * instantiate();
        static void             realizeTopStack();
        #ifdef      PERFORMER
        void                    getAbsMat(pfMatrix & m);
        void                    getAbsCoord(pfCoord &);
        #endif      //PERFORMER
        friend class vrCamera;
```

(나) Main Features of the Methods in vrViewpoint

setFOV() getFOV()

FOV값을 설정/반환하는 함수

getCoord()

좌표축의 위치와 좌표축의 방향을 반환하는 함수

setPosition() getPosition()

관찰자의 위치를 설정/반환하는 함수

setOrientation() getOrientation()

바라보는 방향을 설정/반환하는 함수

getDescription()

설명을 반환하는 함수

setBind()

View 스택을 이용하여 시점을 관리하는 함수

Bind된 Viewpoint는 랜더러에게 연결되어 시점 정보를 제공한다.

●TRUE로 호출할 경우

스택상의 이미 Bind되어 있는 View를 해제하고 현재의 View를 Bind한다.

- FALSE로 호출할 경우

스택상의 최근의 Bind되었던 View를 다시 Bind 하고 현재의 View를 해제한다

getBindableStack()

View Stack을 반환하는 함수

getNumStack()
Stack안의 Viewpoint객체의 개수를 반환하는 함수
isInStack()
Stack안에 Viewpoint객체가 존재하는 여부를 반환하는 함수
topStack()
Stack의 Top Viewpoint객체를 반환하는 함수
pushStack(vrViewpoint *)
Stack에 새로운 vrViewpoint 객체를 삽입하는 함수
popStack()
Top Viewpoint를 스택에서 제거하는 함수
realize()
관찰자의 위치에서 시점에 관한 매트릭스를 계산하는 함수
realizeTopStack()
Top Viewpoint객체를 realize하는 함수
getAbsCoord(pfCoord &)
관찰자의 위치와 Head,Pitch,Roll 값을 반환하는 함수

타. Loading Interface Class

(1) vrLoader

VRML화일을 파싱하고, 기술된 노드의 정보를 관리하는 클래스

관리 정보
1. 파싱 과정상에서 노드를 기술할 필드의 스택
2. 노드의 스택
3. 노드의 이름 공간 스택
4. 프로토타입 노드를 생성할 정보를 관리하는 프로토타입 인포 스택

(가) Class Definition for vrLoader

vrLoader <- NONE

void	init();
void	setRoot(vrWorld *);
vrWorld *	getRoot();
void	addDecl(vrContainer *);
void	addProto(vrProtoInfo *);
void	addRouter(const char *, const char *, const char *, const char *);
void	addNameSpace(char *, vrContainer *);
vrContainer *	lookupNameSpace(const char *);
void	pushNameSpace();
void	popNameSpace();
vrContainer *	topNode();
vrContainer *	createNode(const char *);
void	pushNode(const char *);
vrContainer *	popNode();
const char *	pushField(const char *);
void	popField();
void	setValue(vrBool);
void	setValue(vrInt);
void	setValue(vrFloat);

void	setValue(vrFloat, vrFloat);
void	setValue(vrFloat, vrFloat, vrFloat);
void	setValue(vrFloat, vrFloat, vrFloat, vrFloat);
void	setValue(const char *);
void	setValue(vrContainer *);
void	appendValue(vrInt);
void	appendValue(vrFloat);
void	appendValue(vrFloat, vrFloat);
void	appendValue(vrFloat, vrFloat, vrFloat);
void	appendValue(vrFloat, vrFloat, vrFloat, vrFloat);
void	appendValue(const char *);
void	appendValue(vrContainer *);
protoArray *	topProtoArray();
vrProtoInfo *	topProto();
void	pushProto(vrProtoInfo *);
vrProtoInfo *	popProto();
void	addInterface(int, int, const char *);
void	addReference(const char *);

(나) Main Features of the Methods in vrLoader

init()

내부 구조 초기화

setRoot() getRoot()

Root노드를 설정/반환하는 함수

addDecl()

노드안의 필드로 정의된 노드를 추가하는 함수

일반 노드에 추가될 경우와 프로토타입 노드에 추가될 경우가 있다.

addProto()

Prototype 노드를 추가하는 함수

addRouter()

노드의 필드를 연결하는 ROUTE 구문을 처리하는 함수

addNameSpace()

현재의 스택상의 맨위 이름 공간에 주어진 이름의 노드를 추가하는 함수

lookupNameSpace()

이름 공간에서 주어진 이름의 노드를 반환하는 함수

pushNameSpace() popNameSpace()

이름 공간을 push/pop하는 함수

topNode()

노드 스택상의 상위 노드를 반환하는 함수

createNode()

노드를 생성하는 함수

pushNode() popNode()

노드 스택에 노드를 push/pop하는 함수

pushField() popField()

필드 스택에 필드를 push/pop하는 함수

setValue()

기본 형과 노드에 대해서 필드 스택의 맨 위 필드의 값을 설정

appendValue()

필드 스택에 기본 형과 노드를 추가하는 함수

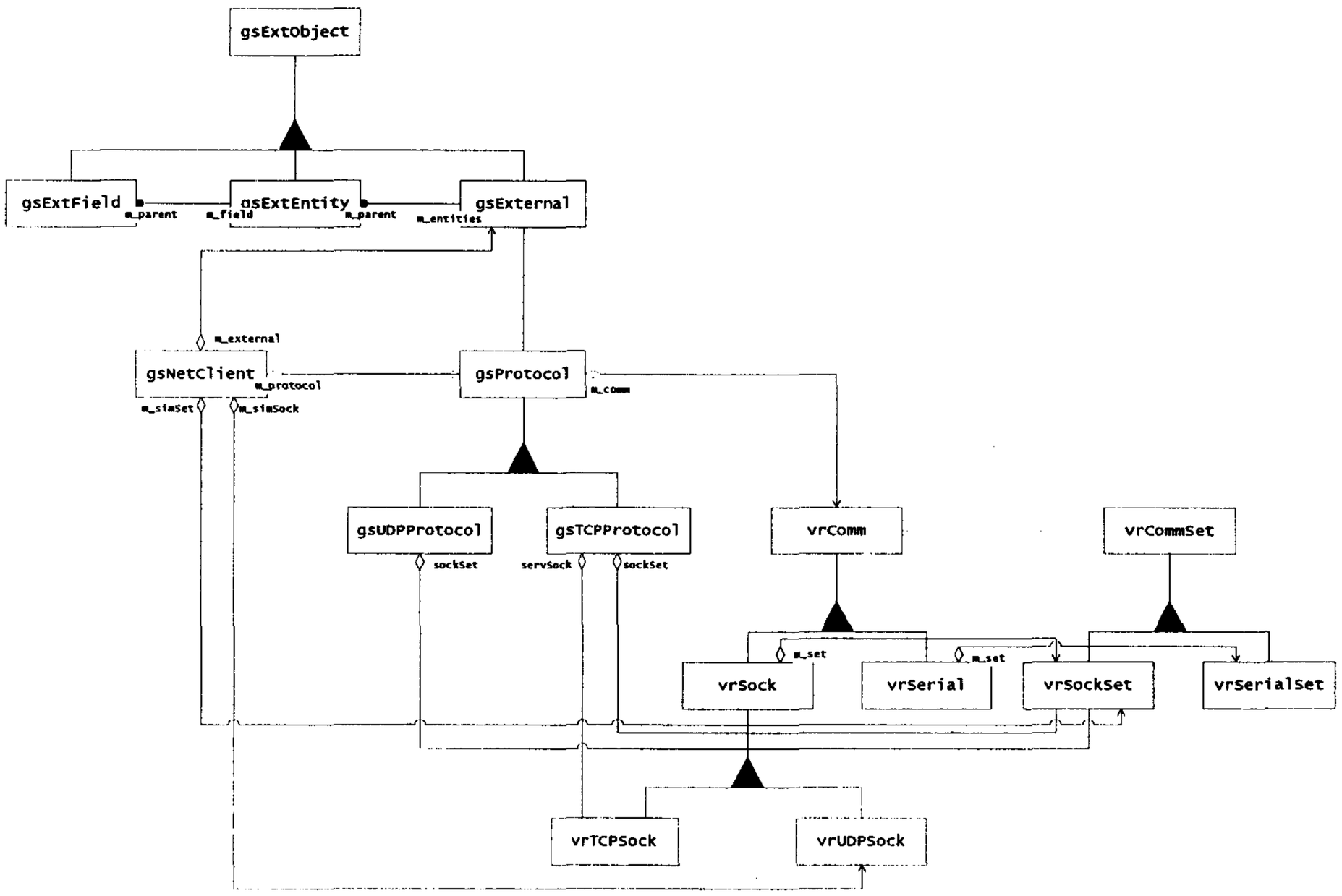
(다) vrLoadVRML()

인자로 주어지는 이름의 VRML 파일을 파싱하고 Scene Graph의 루트 노드를 반환하는 함수

8. API Reference II(External Module Interface API)

외부 모듈의 개발을 지원하기 위한 API로 다음의 네가지 구성요소들이 <그림 27>에 보여지는 연관 관계를 이루고 있다.

- Shadowed Entity
- Protocol Handler
- Basic Communication
- Socket Communication



<그림 27> class diagram of EMI API

가. Shadowed Entity

외부 장치와 연결을 위한 클래스들을 정의

(1) gsExtObject

(가) Class Declaration for gsExtObject

```
struct gsExtObject {
static void *s_arena;
public:
void *        operator new(size_t);
void          operator delete (void * mem);
static void   setArena(void * arena);
static void *getArena();
};
```

(나) Main Features of the Methods in gsExtObject

operator new(size_t)
전달인자 size_t만큼의 메모리를 할당

operator delete (void * mem)
전달인자가 가리키는 객체에 할당된 메모리를 해제

setArena(void * arena)
객체의 주소를 설정

getArena()
객체의 주소를 반환

(2) gsExtField

(가) Class Declaration for gsExtField

```
struct gsExtField : public gsExtObject {
char          m_name[50];
class gsExtEntity *       m_parent;
BYTE          m_putDirty;
BYTE          m_getDirty;
char *        m_getBuff;
char *        m_putBuff;
void *        m_userData;
public:
void          setParent(gsExtEntity * ent);
gsExtEntity *             getParent();
void          setUserData(void * data);
void *        getUserData();
void          setName(const char * name);
char *        getName();
int           setValue(const char * buff);
int           getValue(char * buff);
};
```

(나) Main Features of the Methods in gsExtField

m_name
객체의 이름

m_parent
부모 객체의 주소

m_getBuff
Serve에서 전달하는 값을 저장하는 기억공간

m_putBuff

server에 전달하는 값을 저장하는 기억공간

m_putDirty

보낼 data를 m_putBuff에 저장하고 나면 m_putDirty는 1이 되고, Serve에서 m_putBuff의 내용을 가져가면 m_putDitry는 0이된다.

m_getDirty

Serve에서 m_getBuff로 값을 전달하고 나면 m_getDirty는 1이 되고, m_getBuff에 저장된 값이 Client로 전달되면 m_getDirty는 0이 된다.

m_userData

사용자가 정의하는 임의의 data

setParent(gsExtEntity * ent)

부모 객체의 주소를 설정

getParent()

부모 객체의 주소를 반환

setUserData(void * data)

임의의 data를 설정

getUserData()

사용자 정의 data를 반환

setName(const char * name)

gsExtField의 이름을 설정

getName()

gsExtField의 이름을 반환

setValue(const char * buff)

m_putBuff에 값을 할당, m_putDirty는 1이된다.

getValue(char * buff)

전달 인자인 buff에 m_getBuff의 내용을 담아 반환한다. m_getDirty를 0으로 만들고 이전의 m_getDirty의 값을 return 값으로 반환

(3) gsExtEntity

(가) Class Declaration for gsExtEntity

```
struct gsExtEntity : public gsExtObject {
char        m_name[50];
short       m_id;
short       m_maxFields;
short       m_numFields;
void *      m_userData;
gsExtField **           m_fields;
gsExtemal *             m_parent;
gsExtEventProcessor m_eventProc;
public:
void        postNodeId();
void        setNodeId(int);
short       getNodeId();
void        setUserData(void * data);
void *      getUserData();
void        setEventProcessor(gsExtEventProcessor proc);
void        doEventProcessor(gsExtField * fld);
gsExtField *            queryField(const char * fname);
gsExtField *            lookupField(const char * fname);
char *      getName();
};
```

(나) Main Features of the Methods in gsExtEntity

m_name
객체의 이름

m_id
객체의 식별자

m_fields
일대다 연관관계에 있는 gsExtField 객체의 주소를 가리키는 포인터

m_maxFields
gsExtField 객체의 최대 개수, 기본 값은 32

m_numFields
gsExtField 객체의 개수

m_userData
사용자가 정의하는 임의의 data

m_parent
부모 객체의 주소

m_eventProc
event processor를 위한 함수 포인터
함수의 정의는 외부에서 한다.

postNodeId()
setNodeId(int id)
객체의 식별자를 할당

getNodeId()
객체의 식별자를 반환

setUserData(void * data)
사용자가 정의하는 임의의 data를 할당

getUserData()
사용자가 정의한 data를 반환

setEventProcessor(gsExtEventProcessor proc)
event processor를 할당

doEventProcessor(gsExtField * fld)
외부에서 정의된 Event Processor를 실행

queryField(const char * fname)
server 에 gsExtField 객체를 등록

lookupField(const char * fname)
전달인자의 field name과 같은 이름을 가진 gsExtField의 주소를 반환,
없을 경우는 NULL을 반환

getName()
객체의 이름(m_name)을 반환

(4) gsExternal

(가) Class Declaration for gsExternal

```
class       gsExternal : public gsExtObject {
char        m_name[50];
short       m_id;
class gsProtocol *      m_protocol;
char        m_listName[50];
char        m_needListName;
int         m_numControl;
```

```
char        m_controlBuff[30][32];
gsExtEntity *       m_entities;
short       m_numEntities;
short       m_maxEntities;
int         m_buffSize;
short       m_enabled;
gsExtControlProcessor           m_controlProc;
public:
void        setProtocol(gsProtocol *);
void        setControlProcessor(gsExtControlProcessor);
char *      getName();
short       getId();
int         isEnabled();
void        setEnabled(int e);
void        close();
void        update();
gsExtEntity *           makeEntity(const char * name);
gsExtEntity *           lookupEntity(const char * entName);
gsExtEntity *           lookupEntityId(short i);
int         getNumEntities();
gsExtEntity *           getEntity(int n);
gsExtEntity *           queryNode(const char * nodeName);
void        queryNodeList(const char * nameList);
void        postClose();
void        postNodeName(const char * nodeName);
};
```

(나) Main Features of the Methods in gsExternal

m_name

객체의 이름

m_id

객체의 식별자

m_protocol

gsProtocol 객체의 주소

m_listName

m_needListName

m_numControl

control command의 개수

m_controlBuff

control buffer

m_entities

gsExtEntity 객체 주소를 가리키는 포인터

m_numEntities

gsExtEntity 객체의 개수

m_maxEntities

gsExtEntity 객체의 최대 개수, 기본값은 16

m_buffSize

gsExtField 객체의 m_putBuff와 m_getBuff의 size, 기본 size는 500

m_enabled

gsExternal 객체의 생성시는 1, close되면 0

m_controlProc

control process를 위한 함수 포인터
정의는 외부에서 한다

setProtocol(gsProtocol * prot)

전달인자 prot로 gsProtocol 객체의 주소를 할당

setControlProcessor(gsExtControlProcessor)

control processor를 설정

getName()

객체의 이름을 반환

gsExternal

getId()

객체의 식별자를 반환

isEnabled()

m_enabled 값을 반환

setEnabled(int e)

m_enabled 값을 설정

close()

m_enabled를 0으로 한다

update()

header의 형태에 따라 entity와 field를 갱신

makeEntity(const char * name)

전달 인자 name의 이름으로 entity를 추가하고 생성된 entity의 주소를 반환

lookupEntity(const char * entName)

entity의 이름으로 entity를 찾아 주소를 반환

lookupEntityId(short id)

entity의 식별자로 entity를 찾아 주소를 반환

getNumEntities()

entity의 개수를 반환

getEntity(int n)

entity들 중에서 전달인자가 가리키는 위치의 entity의 주소를 반환

queryNode(const char * nodeName)

전달인자인 node의 이름으로 entity를 추가하고 server에 query node로 등록

queryNodeList(const char * nameList)

server에 전달인자의 이름으로 query node list로 등록

postClose()

close() 함수와 동일

postNodeName(const char * nodeName)

entity를 추가하지 않고 server에 전달인자의 이름으로 query node로 등록

나. Protocol Handler

(1) gsProtocol

(가) Class Declaration for gsProtocol

```
class       gsProtocol {
protected:
class gsExternal *      m_external;
vrComm *  m_comm;
int         m_length;
static unsigned short   s_headerLen;
public:
virtual const char *    className();
void        setExternal(gsExternal * e);
gsExternal *            getExternal();
void        setComm(vrComm * comm);
vrComm *  getComm();
void        setLength(int len);
```

```
int         getLength();
int         getNumReady();
virtual void getProtocolInfo(char * str, int s) = 0;
virtual int  bind() = 0;
virtual int  accept() = 0;
virtual int  link() = 0;
virtual int  get(gsHeaderSpec * header, char * buff) = 0;
virtual int  put(gsHeaderSpec * header, char * buff) = 0;
};
```

(나) Main Features of the Methods in gsProtocol

m_external
gsExternal 객체의 주소

m_comm
vrComm 객체의 주소

m_length
객체의 길이

s_headerLen
header의 길이

className()
class의 이름을 반환
여기서는 "gsProtocol"을 반환

setExternal(gsExternal * e)
gsExternal 객체의 주소를 설정

getExternal()
gsExternal 객체의 주소를 반환

setComm(vrComm * comm)
vrComm 객체의 주소를 설정

getComm()
vrComm 객체의 주소를 반환

setLength(int len)
객체의 길이를 설정

getLength()
객체의 길이를 반환

getNumReady()
읽어들일 data가 있는지 없는지 검사
읽어들일 data 가 있으면 data의 byte 수를 반환. 없으면 0을 반환

getProtocolInfo(char * str, int s)
pure virtual function

bind()
pure virtual function

accept()
pure virtual function

link()
pure virtual function

get(gsHeaderSpec * header, char * buff)
pure virtual function

put(gsHeaderSpec *, char *)
pure virtual function

(2) gsTCPProtocol

(가) Class Declaration for gsTCPProtocol

```
class       gsTCPProtocol : public gsProtocol {
public:
vrTCPSock               servSock;
public:
virtual const char *    className()
virtual int   bind();
virtual int   accept();
virtual int   link();
virtual void getProtocolInfo(char * srt, int s);
virtual int   get(gsHeaderSpec * header, char * buff);
virtual int   put(gsHeaderSpec * header, char * buff);
};
```

(나) Main Features of the Methods in gsTCPProtocol

servSock

server를 가리키는 TCP socket

className()

클래스의 이름을 반환
gsTCPProtocol

bind()

socket을 open하고 socket과 port를 연결
host의 이름과 port 번호를 출력

accept()

client의 connection을 accept.
성공하면 1을, 실패하면 0을 반환
listen 할 때 대기 connection 수는 5

link()

server에 connect
성공하면 1을, 실패하면 0을 반환

getProtocolInfo(char * str , int s)

host의 이름과 port 번호를 전달인자 str에 담아서 반환

get(gsHeaderSpec * header, char * buff)

전달인자 header와 buff에 data를 읽어들인다.
성공하면 header의 크기와 buff의 크기를 합한 값을 반환
실패하면 0을 반환

put(gsHeaderSpec * header , char * buff)

전달인자 header와 buff의 data를 전송
전달되는 data의 총 byte 수를 반환
실패하면 error code를 반환

(3) gsUDPProtocol

(가) Class Declaration for gsUDPProtocol

```
class       gsUDPProtocol : public gsProtocol {
public:
virtual const char *    className();
```

```
virtual void getProtocolInfo(char * str, int s);
virtual int  bind();
virtual int  accept();
virtual int  link();
virtual int  get(gsHeaderSpec * header, char * buff);
virtual int  put(gsHeaderSpec * header, char * buff);
};
```

(나) Main Features of the Methods in gsUDPProtocol

className()

클래스의 이름을 반환
gsUDPProtocol

getProtocolInfo(char * str, int s)

전달인자 str에 host의 이름과 port 번호를 담아서 반환

bind()

UDP socket을 open하고 socket과 port를 연결
host의 이름과 port 번호를 출력

accept()

server의 socket과 client의 socket이 연결되었는지 확인

link()

server가 client로부터 연결요청을 받아들인다.

get(gsHeaderSpec * header, char * buff)

전달인자 header와 buff에 data를 읽어들인다.
성공하면 header의 크기와 buff의 크기를 합한 값을 반환
실패하면 0을 반환

put(gsHeaderSpec * header , char * buff)

전달인자 header와 buff의 data를 전송
전달되는 data의 총 byte 수를 반환
실패하면 error code를 반환

(4) gsNetClient

(가) Class Declaration for gsNetClient

```
class       gsNetClient {
public:
gsNetClient(const char * name , gsProtocol * prot);
int         isSimServerAlive() const;
void        setSimServerHostname(const char * servHost);
int         lookupSimulationServer(const char * args);
void        bindProtocol();
gsExternal *            acceptProtocol();
gsProtocol *            getProtocol();
};
```

(나) Main Features of the Methods in gsNetClient

gsNetClient(const char * name, gsProtocol * prot)

모듈의 이름과 protocol로 객체 생성

isSimServerAlive()

simulation을 위한 server가 살아 있는지를 검사

살아있으면 1을 없으면 0을 반환

setSimServerHostname(const char * servHost)

server의 host 이름을 설정
이미 존재하고 있으면 경고 메시지를 출력하고 기존의 socket을 없애고 다시 만든다.

lookupSimulationServer(const char * args)

server를 살펴본다.
server에 socket이 설정되지 않았거나 살아있지 않으면 0을 반환
server의 socket이 정상적으로 살아있으면 1을 반환

bindProtocol()

protocol을 bind
protocol의 bind()를 참조

acceptProtocol()

protocol이 accept 되어있으면 external을 생성하고 그 주소값을 반환
protocol이 accept 되어있지 않으면 NULL을 반환

getProtocol()

protocol의 주소를 반환

다. Basic Communication

(1) vrCommSet

(가) Class Declaration for vrCommSet

```
class     vrCommSet {
public:
virtual void setHost(const char *host);
};
```

(나) Main Features of the Methods in vrCommSet

setHost(const char * host)

virtual function
IP address를 할당

(2) vrComm

(가) Class Declaration for vrComm

```
class     vrComm {
protected:
int       m_fd;
char *    m_buff;
int       m_buffSize;
public:
vrComm();
enum vrCommMode {VR_DEFAULT, VR_NONBLOCK, VR_OWNER, VR_ASYNC};
int       checkFd();
virtual int   open() = 0;
int       close();
virtual int   flush();
virtual int   checkReady();
int       isOpen();
void      setOpen(int o);
int       setFileDes(int fd);
```

```
int         getFileDes();
int         setBuffSize(int s);
int         getBuffSize() const;
int         setMode(int mode, int val);
int         setMode(vrCommMode mode);
virtual int put(const char *buff, short len);
virtual int get(char *buff, short len);
virtual int putByte(char c);
virtual int getByte();
virtual int getString(char * buff, int n);
virtual void print() = 0;
};
```

(나) Main Features of the Methods in vrComm

m_fd

file descriptor

m_buff

data를 저장하는 장소

m_buffSize;

m_buff의 크기, 기본값은 1024

m_isOpen

file descriptor의 상태를 나타낸다.
0은 close, 1은 open 상태

checkFd()

file descriptor를 검사.
file descriptor가 없으면 error message를 출력.
file descriptor가 close 상태면 0을, open 상태면 1을 return.

open()

pure virtual function.
file descriptor를 open.

close()

file descriptor를 닫는다.

flush()

checkReady()

읽어올 data가 있는지를 검사
읽어올 data가 있으면 data의 byte 수를 반환
data가 없거나 descriptor가 없으면 0을 반환

isOpen()

file descriptor가 open이면 1을 close이면 0을 반환.

setOpen(int o)

file descriptor의 상태를 설정.

setFileDes(int fd)

file descriptor를 설정

getFileDes()

file descriptor를 반환

setBuffSize(int size)

m_buff의 크기를 설정, 기존의 m_buff를 삭제하고 전달인자의 size만큼 m_buff에 메모리를 다시 할당.

getBuffSize()

m_buff의 크기를 반환

setMode(int mode, int val)

file contorl 모드를 설정.

setMode(vrCommMode mode)

file contorl 모드를 설정.

put(const char * buff, short len)

buff에 data를 담아서 생성된 file descriptor에 전달, len은 buff의 길이
전달된 data의 총 byte 수를 반환, 모두 전달되고 나면 0을 반환
실패하면 -1을 반환

get(char * buff, short len)

생성된 file descriptor로부터 buff에 data를 담아 가져온다.
len은 user가 정의하는 buff의 크기, 길이를 정의하지 않으면 m_buffSize만큼 설정
읽어들인 byte 수를 반환, 모두 읽어들이면 0을 반환
실패하면 -1을 반환.

putByte(char c)

생성된 file descriptor에 1 byte를 전달
성공하면 1 실패하면 0을 반환

getByte()

생성된 file scriptor로 부터 1 byte 를 읽어들인다.
성공하면 읽은 1byte data를, 실패하면 -1을 반환

getString(char * buff , int n)

file descriptor에서 전달인자 n byte 만큼의 data를 읽어들인다.

print()

pure virtual function

라. Socket Communication

(1) vrSockSet

(가) Class Declaration for vrSockSet

```
class       vrSockSet : public vrCommSet {
public:
vrSockSet();
vrSockSet(const char *, ushort port);
vrSockSet(vrSockSet&);
static void  getSelfHost(char *, int);
void         setHost(const char *);
void         setPort(ushort port);
char *       getHost() const;
ushort       getPort() const;
sockaddr *   getSockAddr();
int          getLength();
void         print();
};
```

(나) Main Features of the Methods in vrSockSet

vrSockSet(const char * host, ushort port)

host와 port로 객체 생성

vrSockSet(vrSockSet&)

복사 생성자

getSelfHost(char * buff , int s)

buff에 local host의 이름을 담아서 반환, s는 buff의 길이

setHost(const char * host)

host 를 입력받아 그에 대한 IP address를 할당, address family는 AF_INET으로 한다.

setPort(ushort port)

port를 설정

getHost()

network address 대한 host의 정보를 추출하여 이로 부터 host의 이름을 반환

getPort()

port의 번호를 반환

getSockAddr()

socket address를 반환

getLength()

socket address의 크기를 반환

print()

host의 이름과 port의 번호를 출력

(2) vrSock

(가) Class Declaration for vrSock

```
class       vrSock : public vrComm {
public:
vrSock();
vrSock(vrSockSet * set);
void        setSetting(vrSockSet * set);
vrSockSet *             getSetting();
virtual int   bind();
virtual int   shutdown(int);
virtual int   checkReady();
virtual void print();
};
```

(나) Main Features of the Methods in vrSock

vrSock(vrSockSet * set)

vrSockSet 클래스로 socket 생성

setSetting(vrSockSet * set)

vrSockSet 객체의 주소를 할당

getSetting()

vrSockSet 클래스 객체의 주소를 반환

bind()

생성된 socket과 port를 연결.
name space에 존재하는 socket에 이름을 할당

shutdown(int action)

전달인자인 action 값에 따라 socket을 shutdown한다.
action :
0 : data를 읽지않는다.
1 : data를 더 이상 보내지 않는다.
2 : data를 읽거나 보내지 않는다.

checkReady()

읽어올 data가 있는지를 검사
읽어올 data가 있으면 data의 byte 수를 반환

print()

data가 없거나 descriptor가 없으면 0을 반환

socket의 file descriptor와 host이름과 port의 번호를 출력

(3) vrTCPSock

(가) Class Declaration for vrTCPSock

```
class       vrTCPSock : public vrSock {
public:
    vrTCPSock(vrSockSet *);
    virtual int   open();
    virtual int   listen(int);
    virtual int   accept(vrTCPSock &);
    virtual int   connect();
    virtual int   put(const char *, short = -1);
    virtual int   get(char *, short = -1);
};
```

(나) Main Features of the Methods in vrTCPSock

vrTCPSock(vrSockSet * set)

vrSockSet으로 객체 생성

open()

TCP용 socket을 생성
domain은 internet, socket type은 stream socket으로 설정
file descriptor를 반환

listen(int num)

server에서만 사용.
connect되지 않은 server의 socket에 client에서 요청된 connection을 listening
전달인자 num은 server가 accept하는 동안 대기하는 connection의 최대수
성공하면 0을, 실패하면 error code들을 반환

connect()

client에서만 사용
특정 server에 socket의 연결을 요청
성공하면 0을, 실패하면 error code들을 반환

accept(vrTCPSock & client)

server에서만 사용, client에서 요청된 connection을 accept.
server의 socket과 동일한 property를 가진 새로운 socket을 생성
client에 새로운 socket의 file descriptor를 설정
새로운 socket의 file descriptor를 반환

put(const char * buff, short len)

connect된 socket에 data를 보낸다.
전달인자 buff는 data buffer의 주소, len은 buff의 크기, 기본값은 1024
성공하면 보내진 data의 전체 byte 수를, 실패하면 error code들을 반환

get(char * buff, short len)

connect된 socket에서 data를 읽어들인다.
전달인자 buff는 data를 읽어오는 data buffer의 주소, len은 buff의 크기, 기본값은 1024
읽어들인 data의 byte 수를 반환, 성공적으로 끝나면 0을 반환, 실패하면 error code들을 반환

(4) vrUDPSock

(가) Class Declaration for vrUDPSock

```
class       vrUDPSock : public vrSock {
public:
vrUDPSock(vrSockSet *);
virtual int   open();
virtual int   put(const char *, short = -1);
virtual int   get(char *, short = -1);
};
```

(나) Main Features of the Methods in vrUDPSock

vrUDPSock(vrSockSet * set)

vrSockSet 클래스로 객체 생성

open()

UDP용 socket을 생성
domain은 Internet, socket type은 datagram socket
file descriptor를 반환

put(const char * buff, short len)

data를 전송할 socket address에 data를 전달.
전달인자 buff는 전송할 data buffer의 주소, len은 data의 크기, 기본값은 1024
전송하는 data의 전체 byte 수를 반환, 실패하면 error code들을 반환

get(char * buff , short len)

socket address로 부터 data를 읽어 들인다.
전달인자 buff는 전송 받을 data buffer의 주소, len은data의 크기, 기본값은 1024
전송 받는 data의 byte 수를 반환, 성공적으로 끝나면 0을 반환
실패하면 error code들을 반환

제 2 절 3차원 사운드 제시 시스템 개발

1. 국내외 기술개발 현황

가. 관련 연구

(1) HRTF(Head Related Transfer Function)

MIT대학의 Media 연구실에서는 인간이 3차원 공간 상의 음을 인식하는 과정을 실험을 통하여 데이터베이스로 구축해 놓았다[31].

인간의 외이(outer ears)의 모양은 음원이 고막에 도달하기 전에 음원의 스펙트럼과 타이밍을 변형시킨다. 이러한 귀바퀴(pinnae)에 의한 필터링 과정을 HRTF(Head related transfer function)라고 부른다. HRTF는 귀바퀴와 음원사이의 상대적인 위치에 따라 달라지게 된다. HRTF는 각각의 위치마다 고유하기 때문에 앞과 뒤, 위와 아래에 위치하는 음원을 구별할 수 있게 해준다. HRTF의 측정은 실험자의 실험용 인형(dummy head)의 귀통로(ear canal)에 마이크로폰을 놓고 음원의 방위각(azimuth)과 고도(elevation)를 변경시키면서 녹음하여 이루어진다. 이와같이 구축된 HRTF 데이타는 가상 오디오 시스템에서 참여자와 음원간의 상대적인 위치를 입력받아 원음의 소리를 변형시키는 방식으로 응용된다.

HRTF 방식의 단점은 귀바퀴의 모양이 각 개인마다 다르게 때문에 특정 대상을 통해 측정된 데이터는 정확도가 떨어진다는 점이다. 이상적인 HRTF 데이터는 각각 개인마다 측정하여 되어야 하나, 개인별로 HRTF 데이터를 측정하여 사용한다는 것은 매우 비효율적이고 거의 불가능하다고 볼 수 있다. 대부분은 사람들은 HRTF 측정데이타를 통하여 적절히 음원의 공간정보를 파악한다는 것은 실험을 통하여 확인되었다[3].

(2) NPSNET – 3D Sound Server

NPSNET(Naval Postgraduate School Networked Vehicle Simulator)에서 3차원 사운드 시스템을 구성하기 위해서 구현된 하드웨어 시스템이다[32]. MIDI를 기반으로하며 스피커와 같은 개방형 사운드 장비와 DSP장비를 별도로 사용한다. 육면체 사운드 모델(Sound Cube Model)을 제안하여 각 꼭지점에 8 개를 스피커를 배치하고, 분배 공식에 의해서 각 스피커의 볼륨을 조절하는 방식을 사용한다. 단점은 개방형 시스템이 가지는 공통된 특징으로 청취자의 위치가 고정되어 있어야 하며, 음파의 반향, 간섭의 영향으로 사운드가 변질될 가능성이 있다.

(3) SRS (Sound Retrieval System)

SRS 연구소에서 사람의 청각구조를 바탕으로 개발한 3차원 사운드 기술이다. 소리의 주파수에 따라 사람의 귀바퀴를 통해 들어오는 시간의 차이를 분석하여 소리의 위치를 인식하게 되는 점에 착안하여 개발된 기술로서 두개의 스피커를 이용하여 3차원 사운드 효과를 구현하였다. 출력회로의 별도장치를 통해 주파수 대역별로 재생시간에 차이를 두어 인위적으로 3차원 효과를 느끼도록 하는 방식이다. 현재 전자제품이나 스

피커에서 주로 사용되고 있으며, 단점으로 2차원의 소리를 3차원으로 처리하는 과정에서 음의 왜곡이 생기거나 장시간 청취할 경우 두통을 유발할 수도 있다[33].

(4) QSound

QSound 기술은 좌우로 구분되어진 2차원 스테레오 방식의 웨이브 데이터에 QSound라는 고유의 알고리즘을 적용하여 3차원 효과를 얻는 기술이다. 일반 스테레오의 경우 양쪽 소리의 단순 분리에 의해 공간적인 느낌을 받기 어려우나 QSound 알고리즘을 적용하면 반대편 소리의 데이터를 포함하게 되어 소리의 변화에 상호작용하게 된다. 이에 따라 소리의 이동을 입체적으로 표현할 수 있다. 이 기법은 현실감나는 3차원 사운드를 구현할 수 있지만 웨이브 데이터에 한정되며 적용하기 위해 실시간 처리를 수행할 있는 별도 ADSP 프로세서가 필요하다. 또한 응용 프로그램에서 이를 적용하여 개발되어야 하기 때문에 보편화되는데 어려움이 있다[33].

(5) VRML 2.0 의 사운드 노드

VRML 2.0에서 제공되는 사운드 노드[34]는 필드에서 공간상의 음원에 대한 속성을 기술하여 3차원 사운드를 생성할 수 있다. VRML 2.0의 사운드 노드에서 제공되는 필드를 통하여 공간상의 음원의 위치, 음의 발산방향, 발산 영역과 음의 세기, 반복횟수, 파일의 URL을 명시할 수 있다. VRML은 웹 환경에서 적은 리소스를 사용하면서 3차원 사운드의 효과를 내는 데 효과적이다.

현재 VRML 2.0의 사운드 노드에서 반향(reverberation), 도플러 효과(Doppler shift)등은 제공되고 있지 않다. 또한 VRML의 사운드 효과는 VRML브라우저를 통하여서만 재생이 가능하기 때문에 일반적인 타 시스템의 백-엔드(back-end) 모듈로 사용되기에는 무리가 따른다[36].

<그림 28>은 VRML 2.0의 사운드 노드의 예이다.

```
Sound {
        Source AudioClip {
        Description “sound example”
        Loop FALSE
        Pitch 1.0
        StartTime 0
        StopTime 0
        url [“song.wav”]
}
intensity 1
priority 0
```

```
                location 0 0 0
                direction 0 0 0
                minFront 1
                maxFront 10
                minBack 0.5
                maxBack 5
                spatialize TRUE
}
```

<그림 28> VRML 2.0의 사운드 노드

나. 연구결과가 국내외 기술개발현황에서 차지하는 위치

기존의 3차원 사운드 관련 시스템은 실시간 처리 개념이 부족하거나 실험적인 수준에 머무르는 데 비해, 본 연구를 통하여 개발된 사운드 렌더링 및 모델링 시스템은 이벤트-드리븐(event-driven)방식의 가상현실 시스템과 쉽게 통합될 수 있도록 설계되었다. 사운드 시스템을 제어하기 위한 API를 통하여 가상현실시스템에서 발생되는 이벤트 정보를 전달이 용이하다. 본 시스템 개발에 사용된 소프트웨어 방식에 의한 3차원 사운드 렌더링 시스템은 특별한 하드웨어 사양을 요구하지 않는다.

실제 활용이 가능한 3차원 사운드 모델링 및 렌더링 시스템의 개발

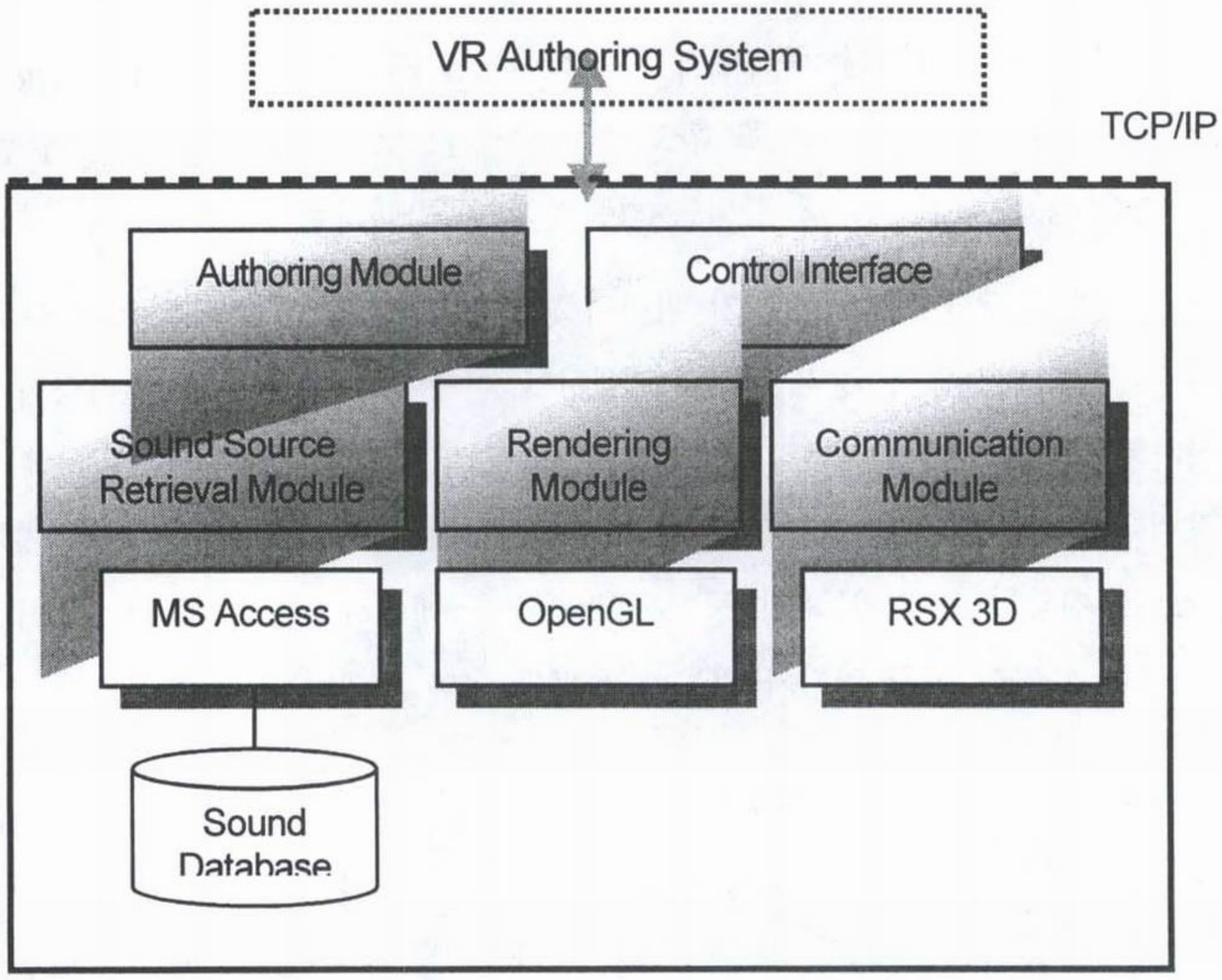
청각환경 제어를 위한 API를 통하여 타 시스템의 백엔드(back-end) 모듈로 활용 가능

특별한 하드웨어 사양을 요구하지 않고 소프트웨어적으로 3차원 사운드 효과 제공

다수의 사용자를 통한 실험을 통하여 3차원 음향 효과 확인

2. 연구개발 수행 내용 및 결과

본 과제에서 개발한 전체시스템의 구성요소는 저작모듈, 제어 인터페이스, 사운드 렌더링 모듈, 외부 통신 모듈, 사운드 소스 검색 모듈로 이루어진다. 사운드 렌더링 라이브러리로 Intel RSX 3D가 사용되었으며, 그래픽 디스플레이 부분은 OpenGL을 이용하였다. 음원 관리를 위한 DBMS으로는 Microsoft Access를 채택하였다. <그림 29>은 본 과제에서 개발한 전체 시스템의 구성도이다.



<그림 29> 본 과제에서 개발한 전체 시스템 구성도

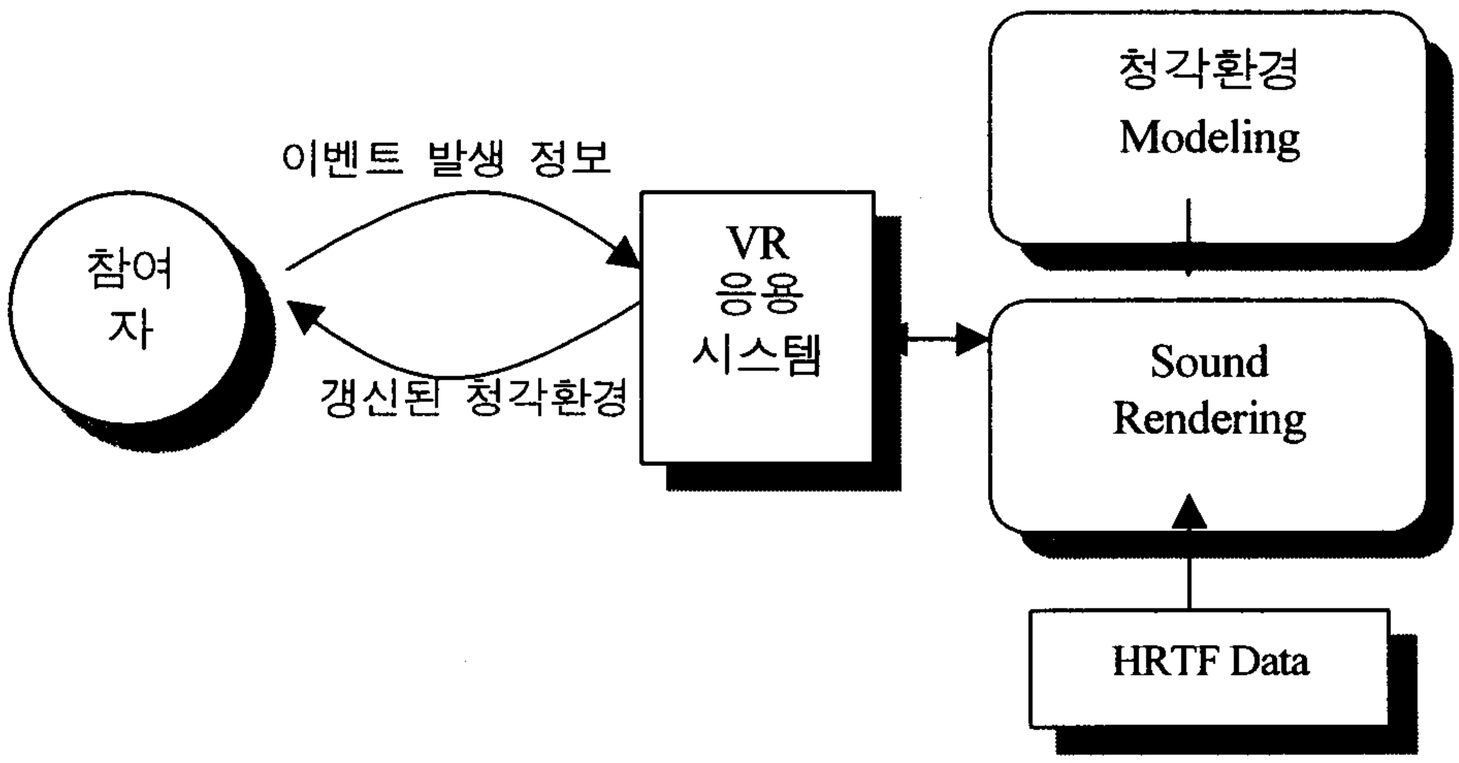
가. 3차원 사운드의 개요

3차원 사운드는 실세계에 존재하는 음을 인간이 공간적인 특성과 함께 인식하는 내용을 가상으로 재현한 소리로 정의할 수 있다. 3차원 사운드를 재현하기 위하여 음원과 청취자와의 위상관계가 기본 요소로 사용되며 다수의 음원이 존재하거나 청취자와 음원이 동시에 이동하는 경우에 대한 처리도 필요하다. 일반적으로 3차원 사운드는 시각 객체와 함께 통합되어 참여자에게 제시될 때 가상환경의 몰입감을 극대화 해준다. 또한 시각환경은 참여자의 FOV(Field of View)에 대해서만 제시해주는 데 비하여 청각환경은 FOV밖에서 발생하는 이벤트 정보를 제공해줄 수 있다.

시점의 이동에 따라 변화되는 환경에 대한 새로운 이미지를 생성하는 과정을 그래픽 렌더링(graphic rendering)이라고 하며, 이와 유사하게 청각환경에서 청취자와 응시방향과 음원과의 위상관계를 통해 새로운 소리를 재생하는 과정을 사운드 렌더링(sound rendering) 이라고 부른다.

3차원 사운드 렌더링은 인간이 실세계에서 공간상의 음원을 인식하는 내용을 실험을 통하여 구축된 데이터, 즉 HRTF 측정데이터를 사용하여 이루어진다 HRTF 데이터베이스로부터 필요한 변환 정보를 참조하여 실시간으로 변환(convolution), 합성(mixing) 등의 과정을 통해 청취자가 듣게 되는 최종적인 음을 생성한다

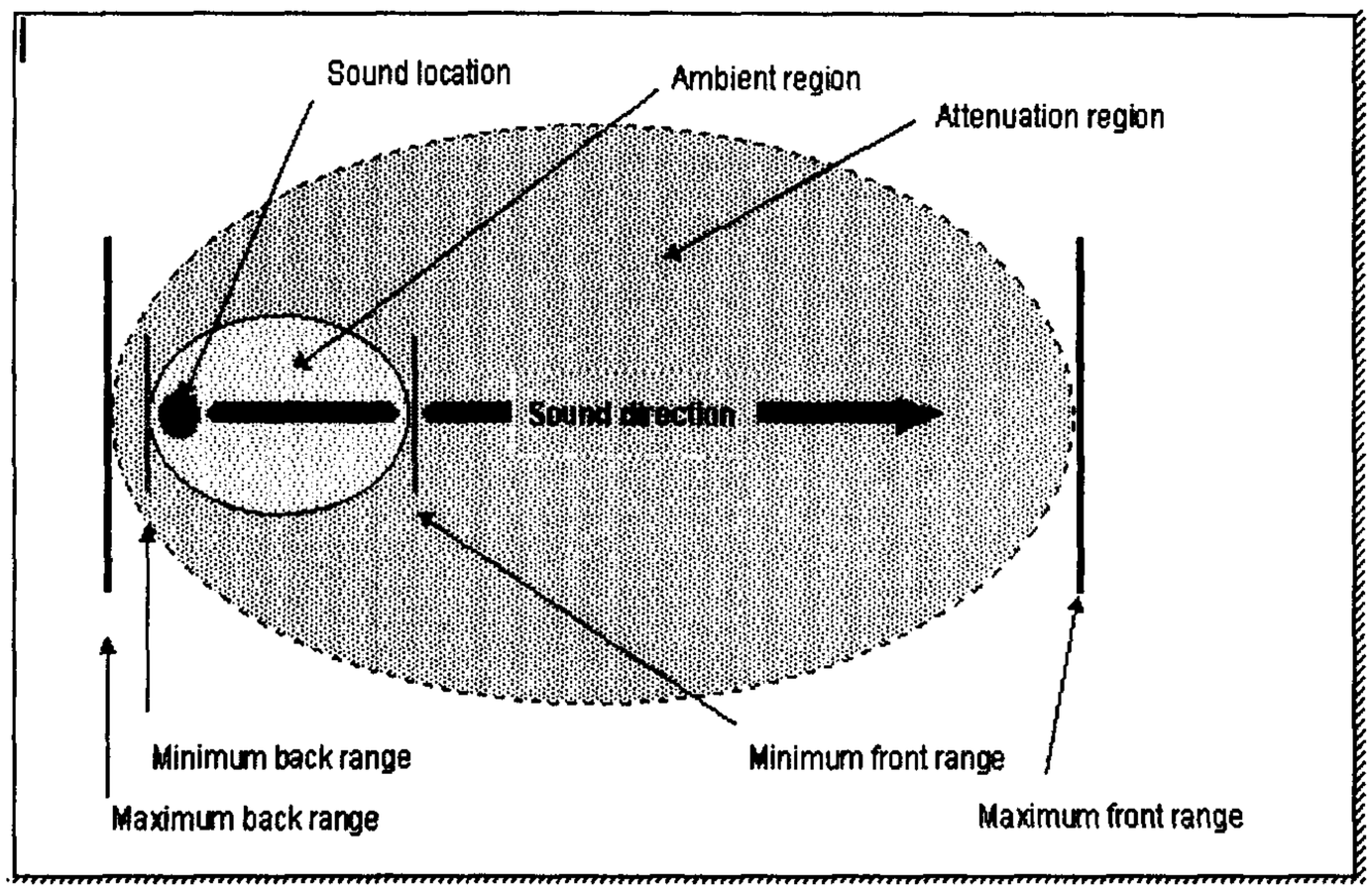
3차원 청각환경을 재현하기 위하여 <그림 30>와 같은 처리과정이 필요하다 샘플링된 음원을 이용하여 청각환경을 모델링이 이루어지고, 사운드 렌더링의 입력정보로 청취자로부터 발생된 이벤트 발생정보, 음원 모델링 데이터(청각 데이터베이스), HRTF 측정데이타가 사용된다 렌더링된 결과는 청각 데이터베이스를 갱신하여 청취자에게 피드백으로 전달된다 위의 과정 중 모델링과정을 제외하고는 실시간으로 처리되며 시스템의 성능에 따라 동시 처리가 가능한 음원의 수와 프레임 율이 결정된다



<그림 30> 3차원 사운드 재생 과정

나 청각환경 모델링 시스템

음원 모델은 공간 상에 존재하는 음원의 특성을 정의하기 위한 모델로 음원의 발산 위치, 발산 방향, 발산 영역으로 구성된다 <그림 31>. 발산 영역은 음의 세기가 일정한 고정 영역(ambient region)과 음의 세기 및 특성에 변화가 있는 감쇄영역(attenuation region)으로 구분된다 발산 방향을 기준으로 감쇄영역까지의 최대거리를 전방최대거리(maximum front)라 하고 발산 반대 방향으로 감쇄영역까지 최대 거리를 후방최대거리(maximum back)이라고 부른다. 고정영역에 대해서는 전방최소거리(minimum front)와 후방최소거리(minimum back)가 존재한다.



<그림 31> 3차원 음원 모델

청각환경 모델링은 시간적인 속성만을 갖는 음원 소스(sound source)에 공간적인 속성을 부여하는 과정이라고 할 수 있다. 발산 방향, 영역 등의 속성을 설정하여 다수의 음원 모델을 정의한다. 본 시스템에서는 청각환경 모델링 인터페이스와 모델링된 환경에 대한 그래픽 디스플레이 기능을 제공한다. 그래픽 디스플레이는 저작내용과 수행과정을 확인하기 위해 사용된다. 모델링된 환경을 청각환경 재생모듈을 통해 사용자와 능동적으로(interactive) 조작과정을 통하여 음원의 추가 및 모델링 내용에 대한 수정이 가능하다.

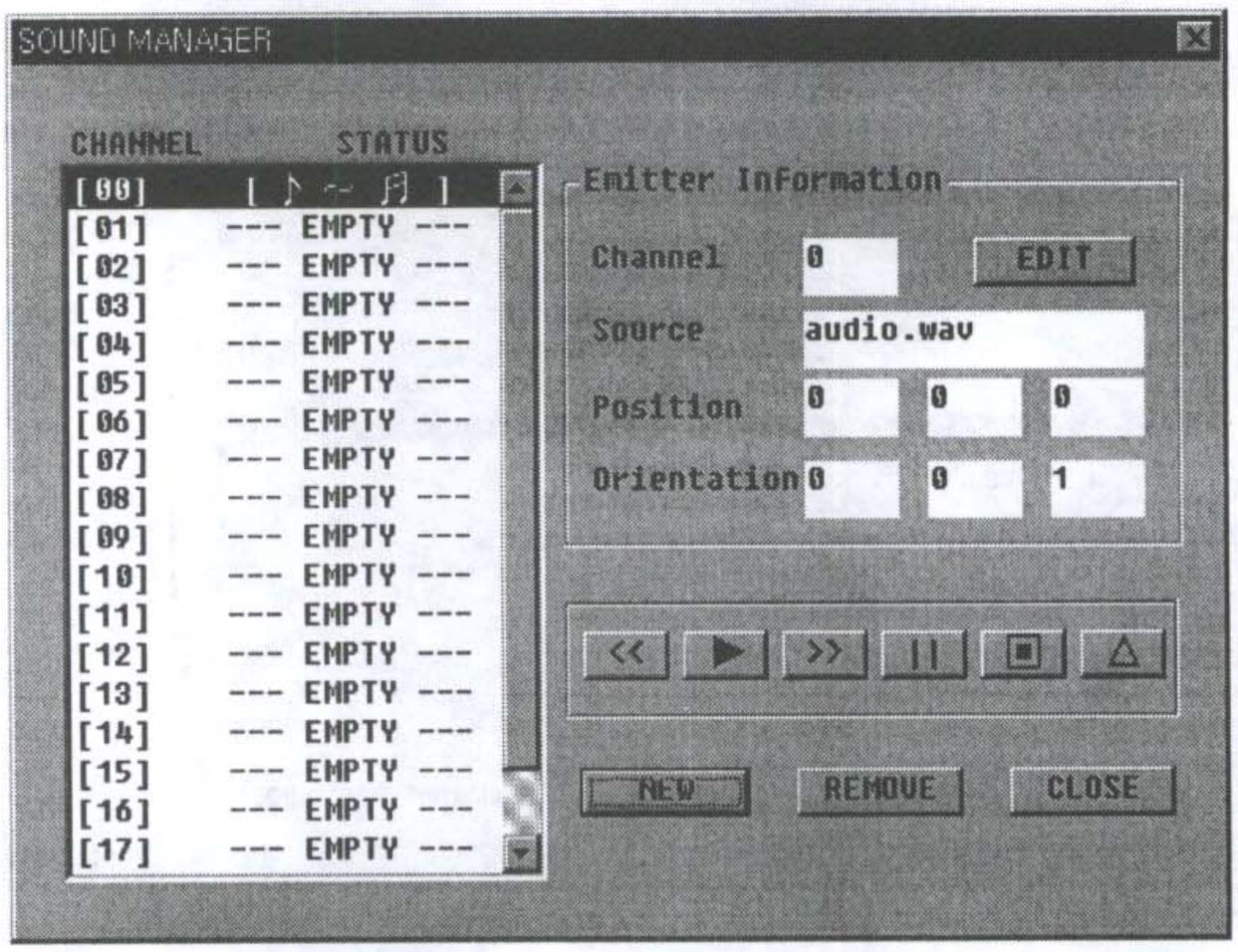
(1) 개발된 모델링 시스템의 구성

청각환경 모델링 시스템의 구성요소로 사운드 환경 관리자, 음원 속성 편집기, 반향효과 제어기, 모델링 환경뷰어를 개발하였다. 저작의 효율성을 높이기 위하여 그래픽 방식에 의한 청각환경 모델링 개념을 도입하였다[37].

(가) 사운드 모델 관리자

사운드 에미터 모델 생성, 편집, 삭제기능

20 개의 음원채널에 대한 동시 생성 가능



<그림 32> 사운드 모델 관리자

(나) 음원 속성 편집기

사운드 에미터 모델을 생성하는 편집도구

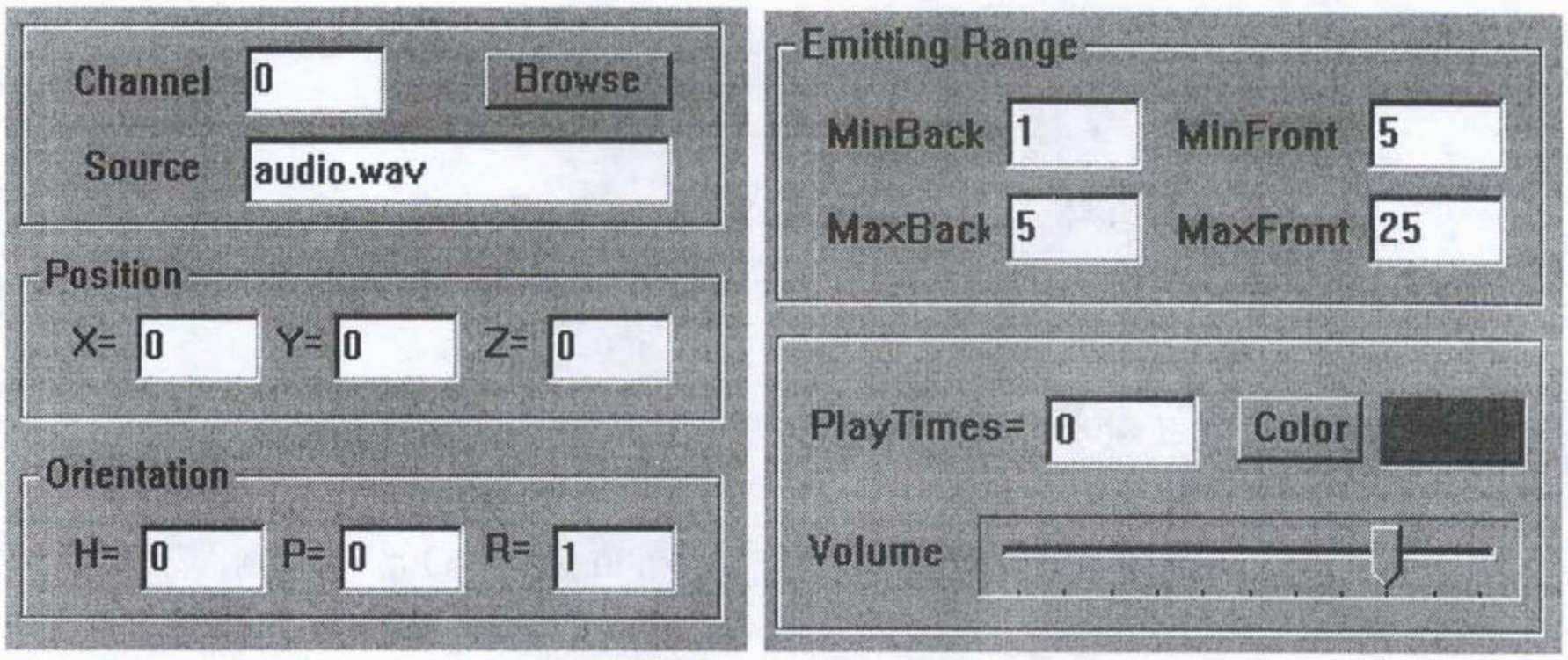
음원 소스 브라우징, 음원 URL, 발산방향, 발산영역, 음의 세기, 재생반복 횟수 등 속성 편집

그래픽 환경에서의 음원 식별을 위한 색상 설정기능

[표2]는 사운드 모델의 속성을 정리한 것이다.

<표 3> Sound Emitter Model Parameter

Parameter	설 명
Px, Py, Pz	사운드 에미터의 3차원 공간 상의 좌표 (right-handed coordinate-system)
Ox, Oy, Oz	음의 발산 방향 벡터
Max Front, Max Back	음의 세기가 감쇄되는 영역 (attenuation region)
MinFront, MinBack	음의 세기가 고정되는 영역(ambient region)
Intensity	음의 세기
Play times	음원의 재생 횟수(0 인 경우 무한 Loop)
Source file	Wave, Midi 형식의 사운드 소스 파일의 URL



<그림 33> 사운드 에미터 속성 편집기

(다) 반향효과 제어기

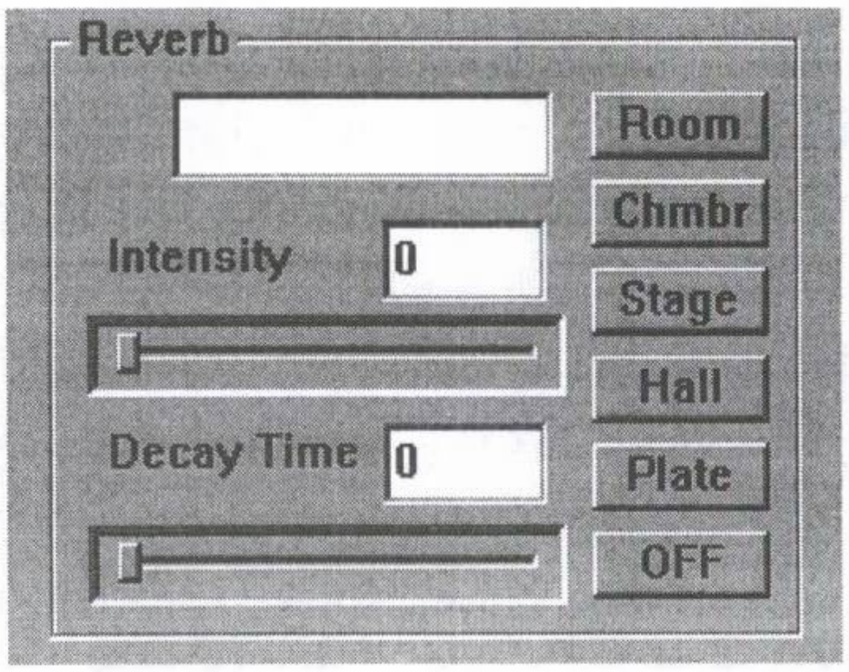
반향효과를 위한 매개변수 설정 및 PRESET기능

5개의 반향 PRESET : Room, Chamber, Stage, Hall, Plate

반향되는 음원의 Intensity, DecayTime 속성값 조절

<표 4> 반향 제어 매개변수

Parameter	기 능
Intensity	반향되는 음의 세기(0 ~ 1)
DecayTime	반향음이 소멸되는 시간(0 ~ 10)



<그림 34> 반향 효과 설정 패널

(라) 모델링 환경 뷰어

모델링된 청각 정보를 그래픽 환경에 디스플레이 기능(OpenGL 그래픽 라이브러리 사용)

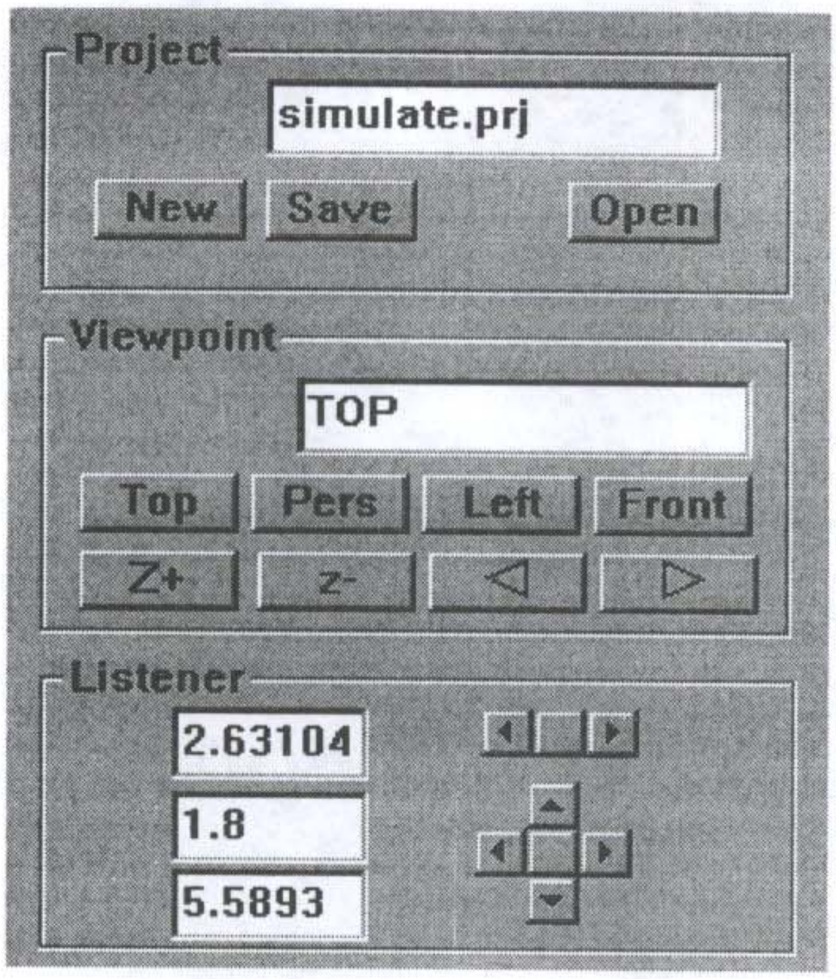
사용자 조작에 따라 변화된 화면 갱신

다중 뷰포인트(Viewpoint) 제공, Zoom-In, Zoom-Out, Rotation 기능

(마) 모델링 환경의 저장 및 로딩 기능

모델링된 환경을 Project 파일로 저장 및 로딩하는 기능 제공

Project 파일은 자체 정의한 포맷을 사용하였다.



<그림 35> 청각환경 조작 패널

다. 청각환경 렌더링 시스템

청각환경 렌더링은 청취자의 위치, 방향 정보와 음원의 모델링과 정보를 통해 청각환경을 갱신을 실시간으로 처리해주는 과정이다. 즉 청취자가 이동하거나 방향을 전환하는 경우 들려오는 음의 속성은 달라지게 되고, 이 과정은 허용오차 이내로 처리되어야 현실감을 유지할 수 있다. 청각 렌더링은 보통 HRTF 측정데이타를 기반으로 이루어지며 계산량이 많기 때문에 음원의 수가 많은 경우나 저속의 컴퓨터상에서는 음이 끊기는 현상에 발생할 수 있고 실감나는 환경을 제공해주기 어렵다.

본 과제에서는 HRTF 데이타를 직접 처리하는 수준이 아니라 HRTF 데이타를 기반으로 공개된 렌더링 라이브러리를 사용하였다. Intel 사에서 제공되는 RSX(Realistic Sound Experience)[6]는 청각환경을 제어하기 위한 API를 제공하고 있으며, 본 연구에서는 사용자의 입력 정보를 받아들여 라이브러리의 API를 호출하는 인터페이스 모듈을 개발하였다.

(1) 렌더링 시스템의 수행과정

렌더링 시스템은 런타임에 수행되며, 청취자의 주행에 따라 변화되는 청각환경을 실시간으로 재현해 준다. 수행 도중 모델링된 음원속성의 변경이 가능하다.

(가) 음원 모델링 데이터를 통한 청각환경 초기화

3차원 청각환경의 구성을 위하여 환경 파라미터의 설정 및 초기화가 이루어진다. 특히 RSX 3D 에서 제공하는 클래스 객체의 초기화 작업을 수행한다.

(나) 사운드 에미터 생성

모델링 도구를 이용하여 사운드 에미터의 속성 값을 설정하고 청각 데이터베이스를 구축한다. 데이터베이스의 에미터 모델을 참조하여 음을 생성한다.

(다) 사용자 주행 인터페이스

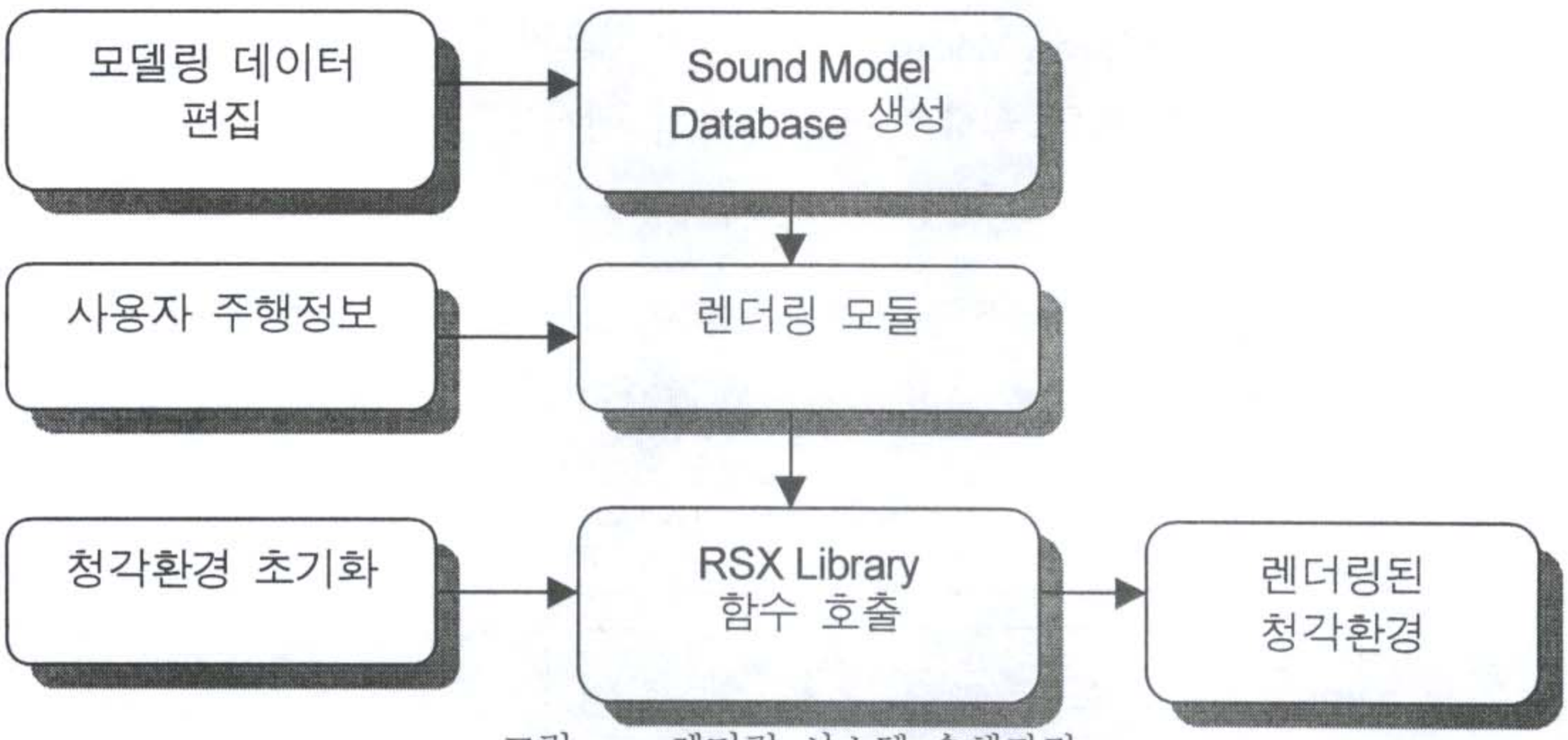
모델링된 음원이 재생되는 모드에서 청취자의 위치 변경, 방향 변경이 가능하다. 사용자가 주행한 정보는 다시 렌더링 시스템으로 보내져 새롭게 청각 환경을 갱신한다.

<표 5> Listener Parameter

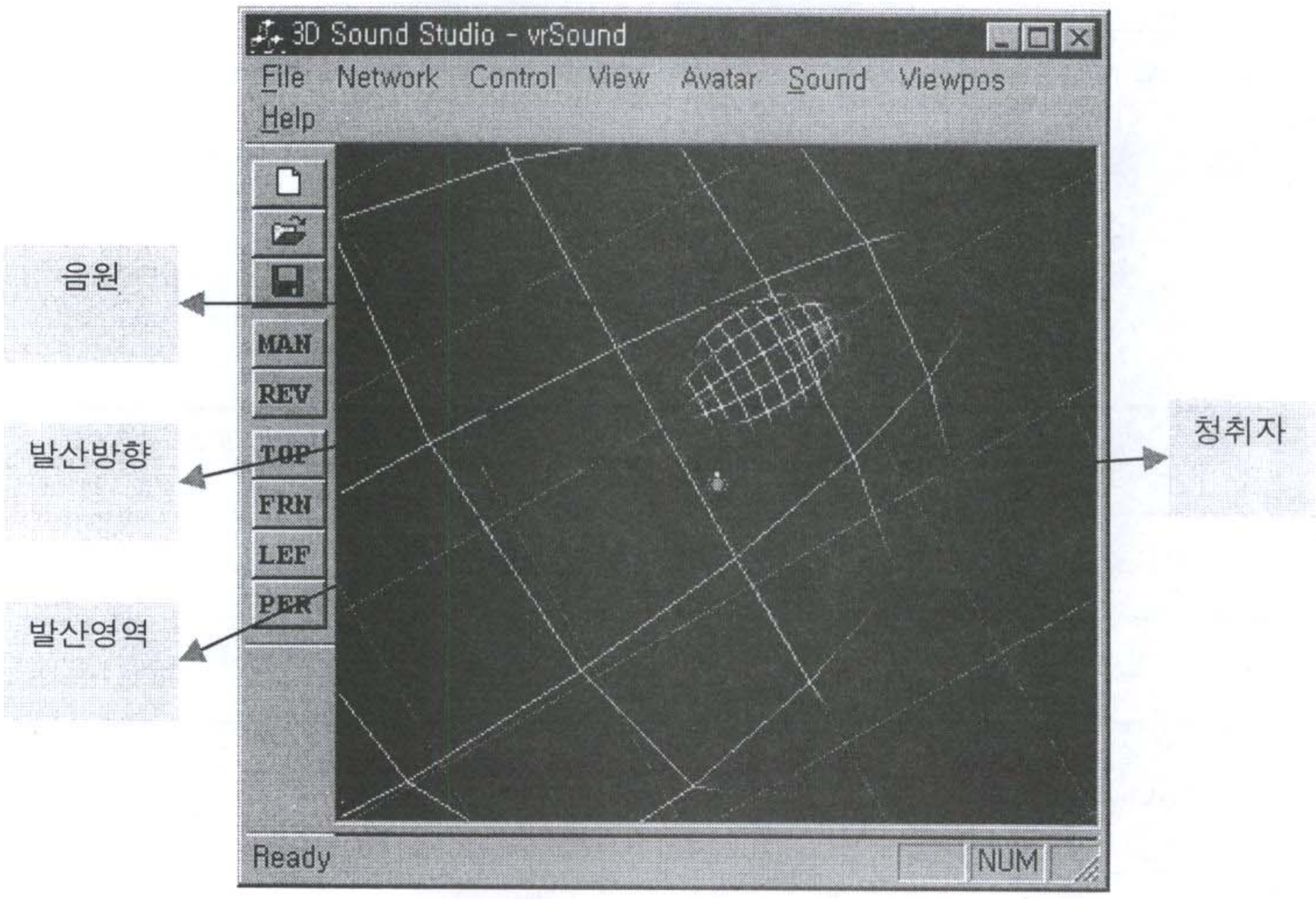
Parameter	기 능
Px, Py, Pz	청취자의 위치좌표
Ox, Oy, Oz	청취자의 응시방향에 대한 벡터

(라) 렌더링 모듈

청각환경을 제어할 수 있는 고수준(high-level)의 함수를 구현하였으며 환경에 대한 제어를 용이하게 하였으며, 각 함수의 기능은 하부의 RSX 3D 라이브러리 함수 호출을 통하여 이루어진다. 사용자 정보 및 사운드 에미터 모델 정보를 바탕으로 렌더링을 수행한다.



<그림 36> 렌더링 시스템 수행과정



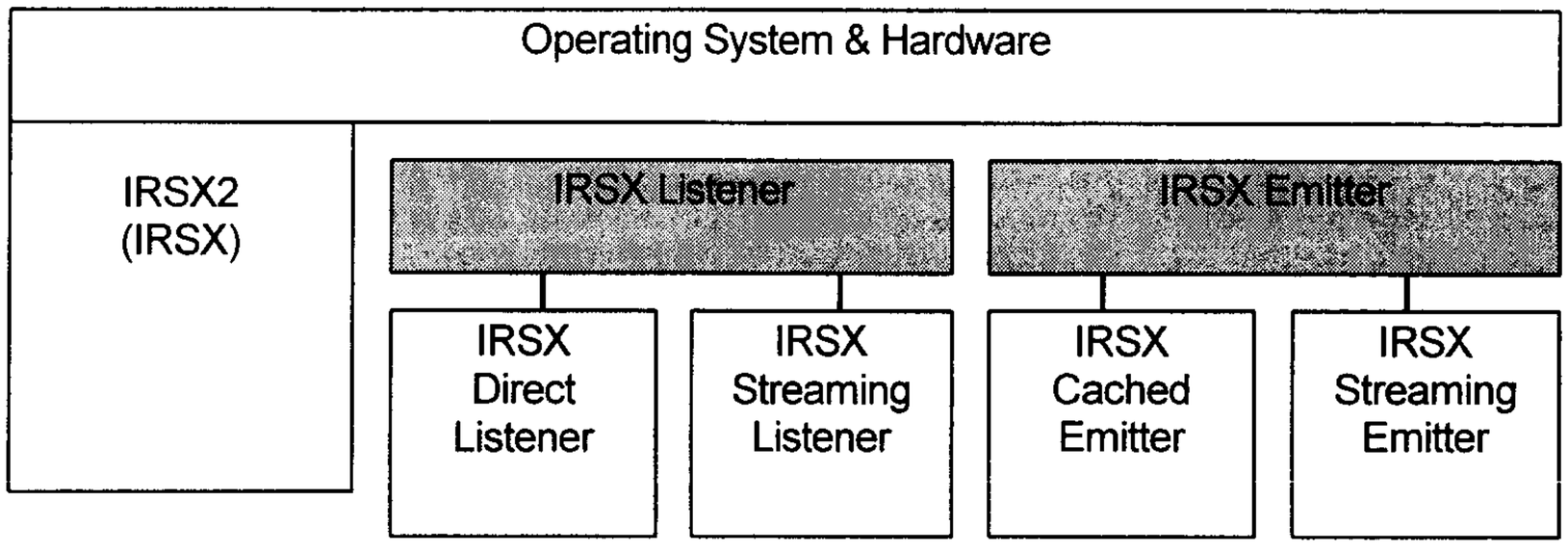
<그림 37> 사운드 렌더링 수행화면

(마) Intel RSX 3D Library

Intel사에서 제공하는 RSX 3D 라이브러리는 현재 웹 상의 가상환경 표준기술언어인 VRML 2.0의 사운드 노드를 처리하는 모듈로 사용되고 있으며, 3차원 청각환경을 제어하는데 필요한 저수준(low-level)의 API를 제공하고 있다. 본 과제에서 개발된 렌더링 시스템은 모델링 환경을 바탕으로 하부에서 RSX 3D API를 호출하여 구현되었다.

Emitter Model과 Listener Model을 정의할 수 있으며 실행시간에 정의 모델의 속성을 변경시킬 수 있다 청취자의 속성은 Listener Model에서, Sound Emitter는 Emitter Model에서 담당하며 Sound Emitter 복수로 생성이 가능하다

다음은 RSX 3D에서 제공되는 클래스와 인터페이스이다 짙은색으로 처리된 부분이 인터페이스 부분이며, OS를 제외한 나머지 부분이 클래스이다



<그림 38> RSX 3D의 클래스와 인터페이스

다음 표에 RSX 3D의 클래스와 인터페이스의 주요 기능이다

<표 6> RSX 3D의 클래스와 인터페이스의 기능

클래스 / 인터페이스	주 요 기 능
IRSX2(IRSX)	3차원 오디오 환경을 관리하는 메인 인터페이스
IRSXEmitter	Sound Emitter 의 공통 서비스를 제공하는 abstract class
IRSXCached Emitter	Sound source에 대한 URL, 메모리, 리소스 를 사용하는 high-level emitter interface (position, oritation, emittting model 정의)
IRSXStreaming Emitter	실시간 오디오 버퍼를 처리하는 low-level emitter interface
IRSXListener	Direct listener, streaming listener 의 공통 서비스를 제공하는 abstract class
IRSX Direct Listener	출력장치로부터 direct listener에게 연결/해제하는 도구 제공 high-level 인터페이스 (청취자의 위치, 방향)
IRSX Streaming Listener	Streaming listener에게 동기적인 오디오 출력 생성을 요청하는 도구 제공 low-level interface

라. 음원 DB 및 검색도구

(1) 3차원 음향 저작 시스템을 위한 음원 DB 설계

VR응용 시스템 저작시 필요한 음원의 신속하고 용이한 확보를 통해 저작의 효율성을 높이기 위하여 음원 DB의 개발을 추진되었다. 음원 DB를 구축하기 위하여 음원 및 음원속성관리를 위한 스키마를 정의하였으며, DBMS는 관계형 데이터베이스인 Microsoft Access 97을 이용하였다.

아래의 표는 음원 DB의 정의된 스키마 필드이다. ContentType과 Description을 제외하고는 음원 소스로부터 추출이 가능하다.

<표 7> 음원 Database Schema

Field	ID	FileName	FilePath	Format	FileSize
Data type	Long	String	string	string	Long
Data range Or example	0 ~	“bird.wav”	\animal\bird	“wav” or “mid”	35 KB
Field	**Sampling Rate**	**Sampling Resolution**	**RunningTime**	**ContentType**	**Description**
Data type	Float	Integer	Integer	String	String
Data range Or example	11.02kHz ~ 44.1 kHz	8 bits ~ 16 bits	25 sec	“bird”, “car”	“sparrow”, “car horn”

(2) 음원 검색 도구의 개발

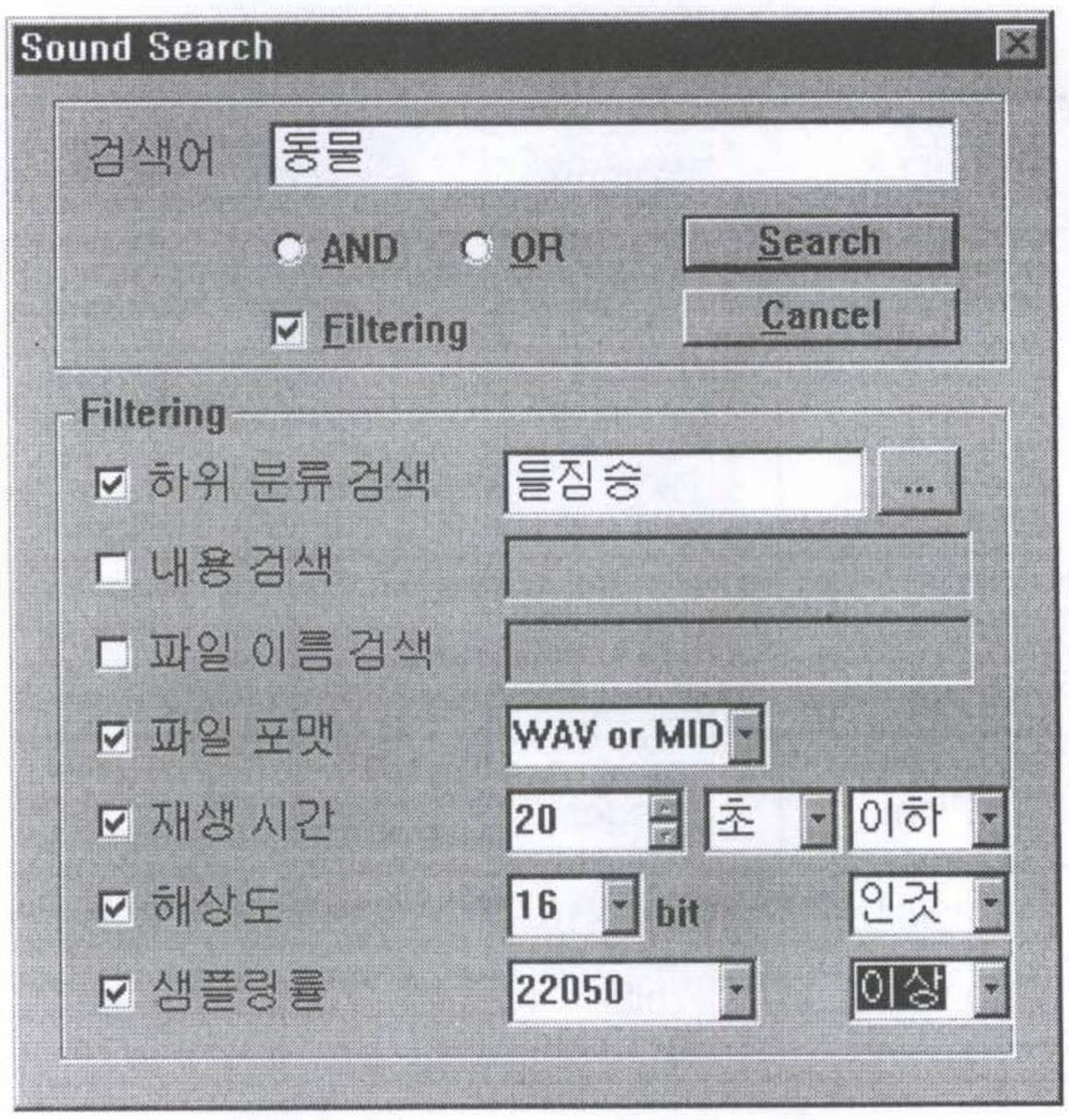
저작에 이용할 음원을 효율적으로 검색하기 위하여 기존의 텍스트 검색 등 관련 검색 기법을 활용하여 검색 도구를 개발한다.

<표 8> 음원 검색 도구의 기능

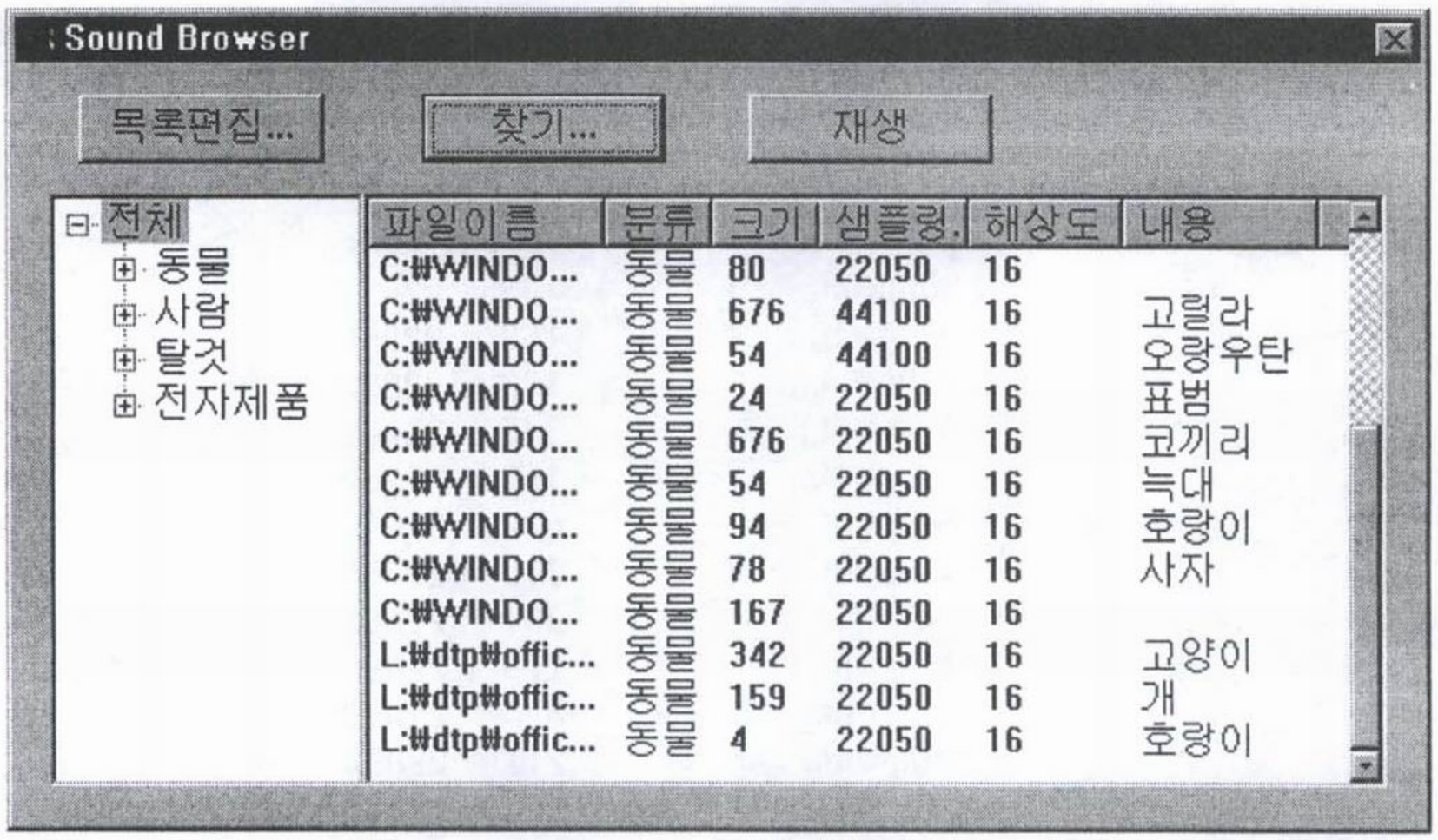
주요 기능	내 용

텍스트 불리언 검색	- AND, OR 연산자 지원
검색 필터링 옵션 기능	- 검색에 사용될 음원 속성 필드를 선택하여 검색 데이터의 양을 조절 - 분류목록을 통한 검색 영역지정, 재생 시간별 검색이 가능

<그림 39>와 <그림 40>는 각각 텍스트 검색화면의 예와 검색결과이다.



<그림 39> 텍스트 검색 화면의 예



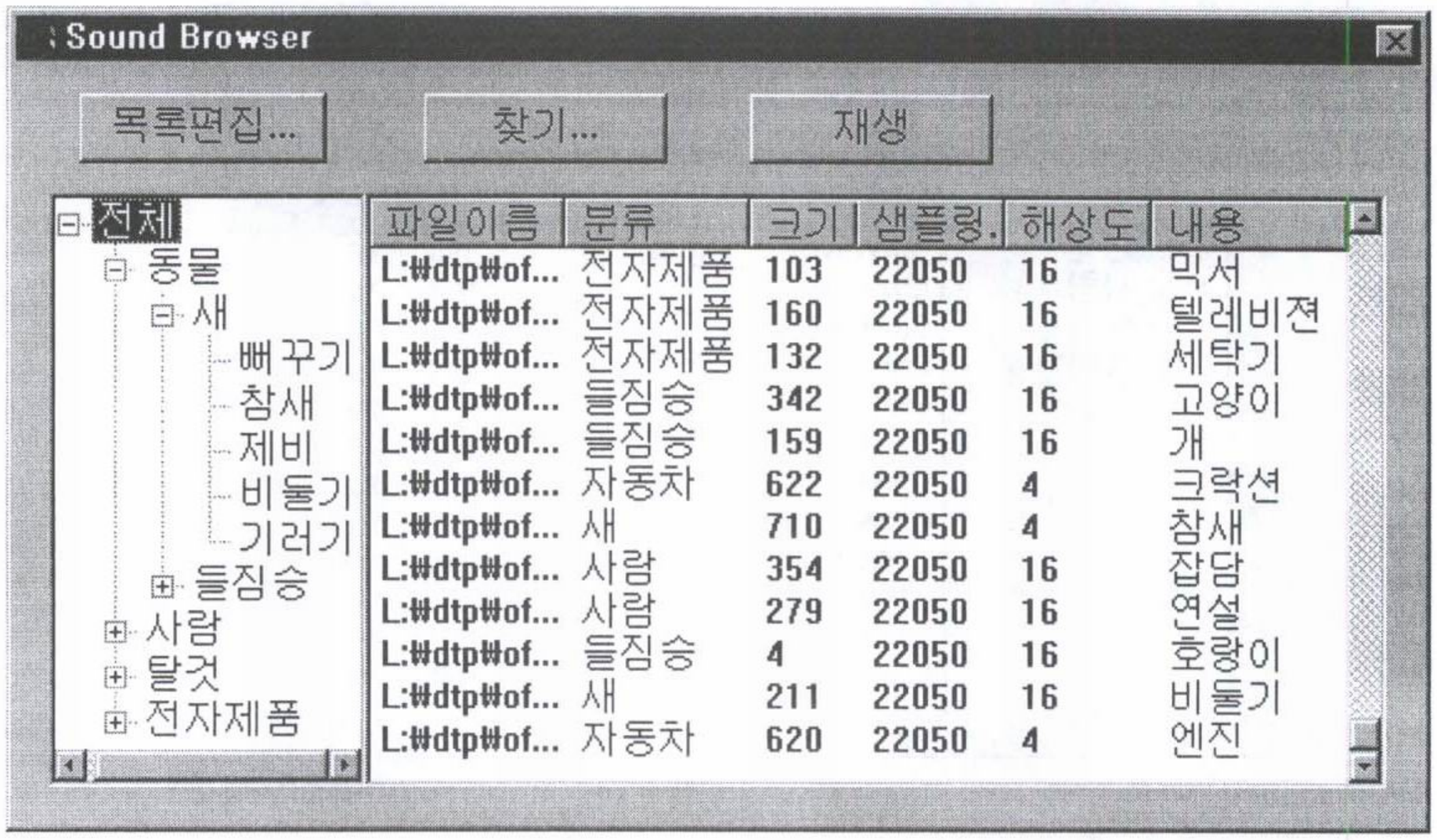
<그림 40> 검색 결과

(3) 음원 브라우징 도구의 개발

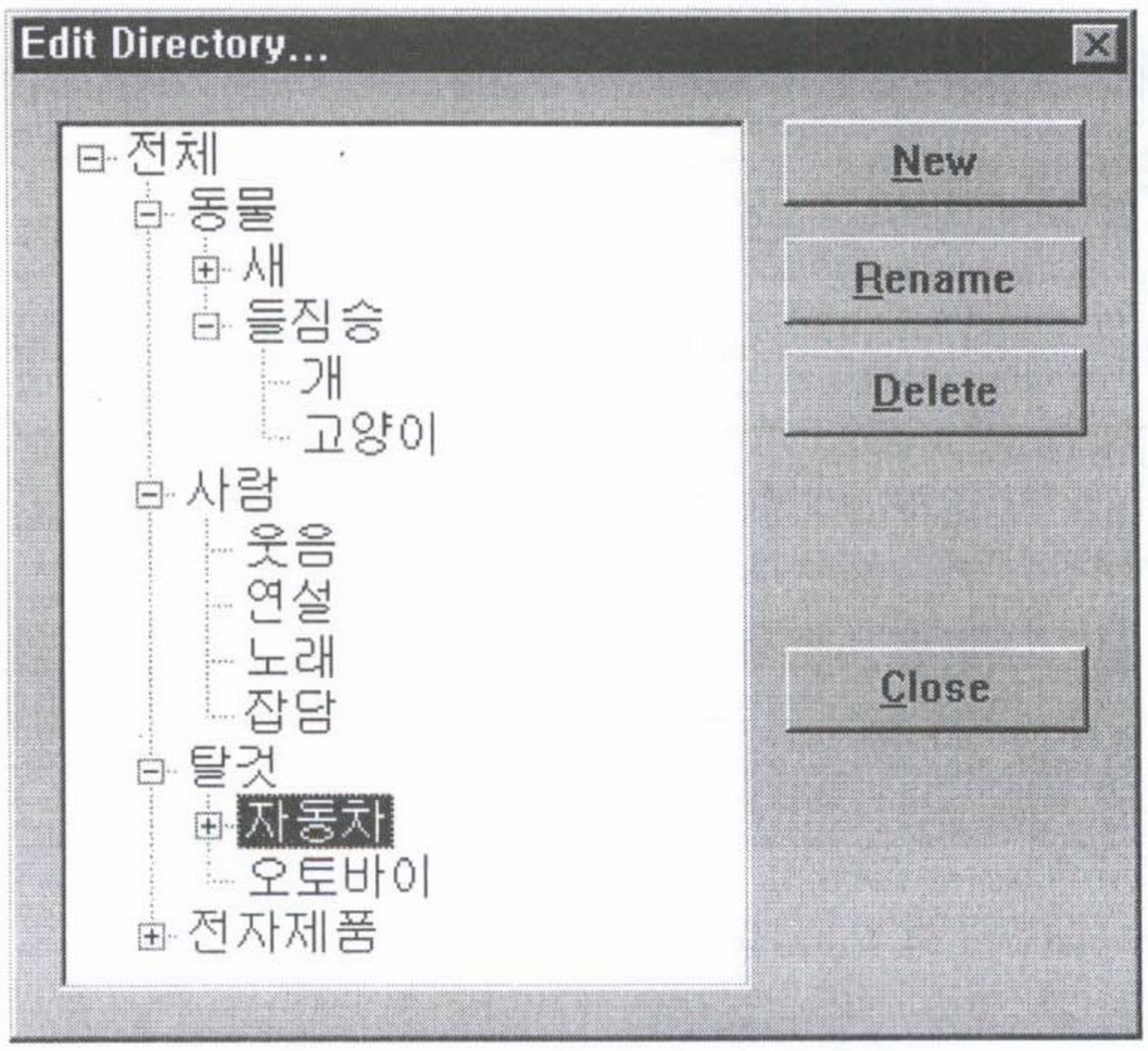
검색된 음원이 직접 사용하기 부적절할 경우, 음원 필터링작업, 노이즈 제거 및 생성 등 작업을 수행할 편집 도구를 개발한다.

<표 9> 음원 브라우징 도구의 기능

주요 기능	내 용
계층적 분류 목록을 통한 음원탐색	예) 탈것-자동차, 동물-새, 사람-걷는 소리 등
분류 목록 편집 및 저장 기능	2단계 계층 목록 편집, 저장
음원 속성정보 출력 및 재생기능	Preplay 기능
음원 클립 추가 및 편집 기능	외부편집기 호출을 통한 음원 편집



<그림 41> 브라우징 과정의 예



<그림 42> 음원 분류 목록 편집

마. 시스템 통합

(1) 시각시스템과의 통합

본 과제에서 개발된 사운드 서버 시스템은 VR 응용시스템과 동기화되기 위하여 클라이언트-서버 방식으로 이벤트 메시지 통신을 수행한다. 이벤트 메시지를 통하여 VR 환경에서 시시각각 변화하는 객체의 위상정보를 수신한다. 이벤트 메시지에 의한 통신은 청각환경 제어뿐만 아니라 시각환경과의 동기화를 위하여 필요하다. 이벤트 메시지는 참여자의 위치, 방향 정보, 사운드의 위상 및 모델 정보, 반향효과 제어정보로 구분되어 정의되었다. 통신환경은 LAN을 기반으로 TCP-IP상에서 소켓통신에 의하여 수행된다.

동기화를 위하여 메시지 수신에 대한 응답을 보내는 Handshaking 방식을 채택하였으며 정교한 동기화를 위해서는 VR 저작 시스템에서 제어 메시지 전송 시 타임클럭(time clock)을 이용하여 전송하는 방식도 함께 사용하였다. 즉 제어 메시지 폭주로 인해 동기화가 되지 않는 경우를 방지하기 위하여 클라이언트 쪽에서 메시지를 전송할 때 서버의 처리 용량을 고려하여 전송을 늦추는 방식이다.

(2) 청각환경 제어 API

VR응용 시스템으로부터 통신 채널을 통하여 청각환경 제어를 위하여 이벤트 발생 정보를 메시지로 변환하여 전송한다. 전송하는 이벤트 메시지는 아래 표의 각 필드를 조합하여 구성된다. Object Type은 크게 Emitter, Avatar, Reverb로 이루어지고, 복수의 Emitter를 구별하기 위한 ID가 부여되고, 반향(Reverb)효과는 전체 청각환경에 적용된다.

<표 10> 이벤트 메시지 포맷

Object type	Operation	Parameter
EMITTER	ID	Integer
	POSITION	Float X, Y, Z
	ORIENTATION	Float X, Y, Z
	SOURCE	File URL
	MINFRONT	Float number
	MINBACK	Float number
	MAXFRONT	Float number
	MAXBACK	Float number
	PLAYTIMES	Integer number
	VOLUME	Float number(0 ~ 1)

	PLAY STOP	No parameter No parameter
AVATAR	POSITION ORIENTATION	Float X, Y, Z Float X, Y, Z
REVERB	OFF ROOM STAGE CHAMBER HALL PLATE	No parameter “ “ “ “ “

제 3 절스테레오스코픽 3차원 시각 파라미터 구현 기술

1. 입체시와 심도 인식

가. 깊이 인식에 영향을 미치는 요인

스테레오스코픽 디스플레이에서 깊이 인식에 영향을 주는 요인은 parallax의 크기, parallax의 방향(crossed, uncrossed), 디스플레이되는 물체의 크기, 관측자의 스크린으로부터의 거리등으로 크게 나누어 질 수 있다. 스테레오스코픽 디스플레이는 물체들간의 상대적인 깊이를 일으키는 요인을 제공하며 이를 통해 입체감을 느끼게 되는데 깊이인식 문제에서 가장 큰 관심사가 되는 것은 느껴지는 깊이가 정확한가 하는 것이다. 이러한 관심은 양안 시차(binocular disparity) 자체가 불분명한 깊이 인지 요인이 되기 때문이다. 예를 들어 같은 disparity값이 관측거리에 따라 서로 다른 깊이를 일으킬 수 가 있다. 따라서 정확한 깊이 인식을 위해서는 관측거리를 고려하여 시차 정보가 재 설정 되어야 하며 이와 같이 관측거리에 대해 보정된 disparity를 가지며 정확한 깊이를 나타내는 것을 depth constancy라고 한다.

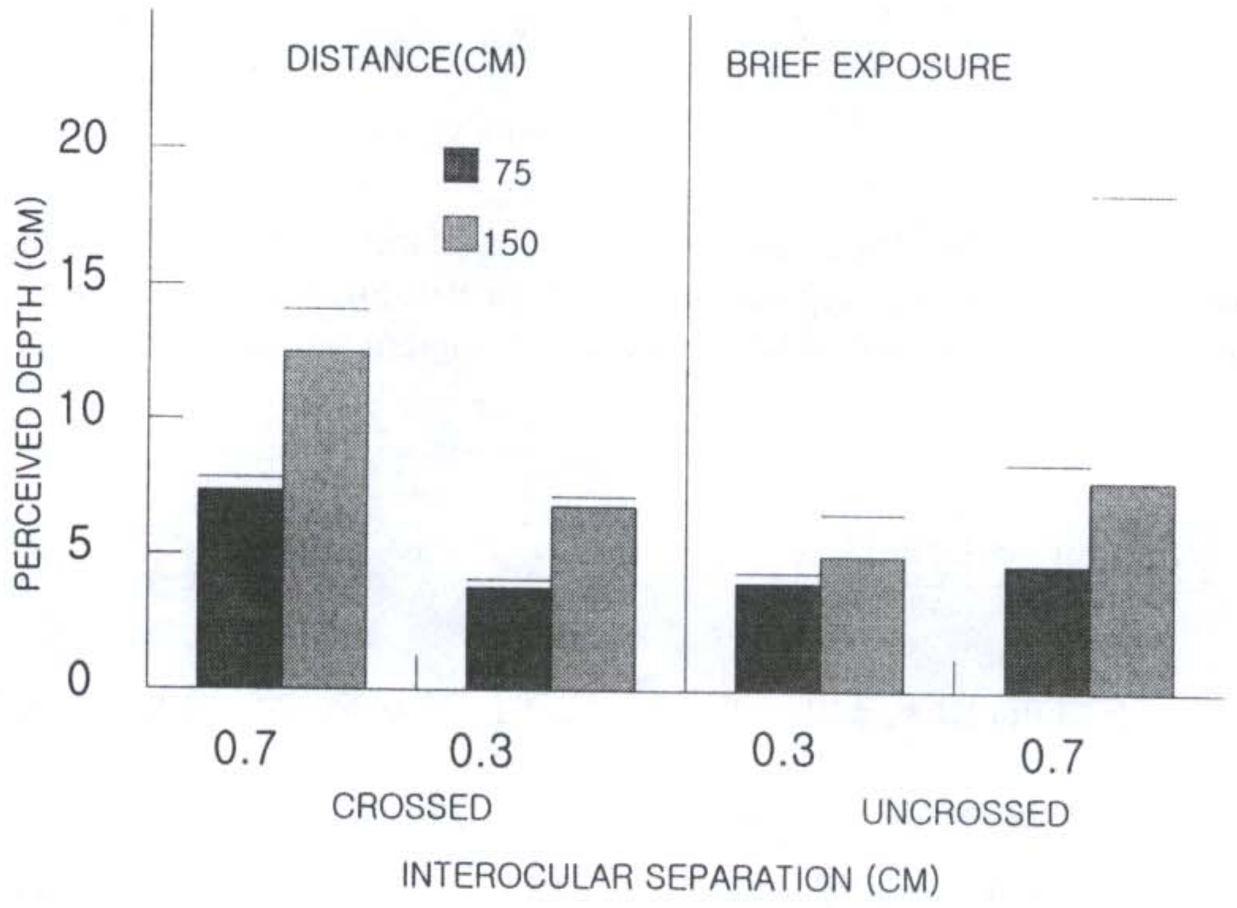
<그림 43>는 스크린상에서의 수평 parallax를 0.3과 0.7로 하고 두 가지의 관측거리(75, 150cm)로 하였을 경우의 실제 인식되는 깊이의 측정치(막대)와 기하학적으로 예상되는 깊이(가로선)을 비교한 것이다. 또한 노출시간을 무한대 노출과 160ms로 구분하였다. 도표를 통해 관측거리 및 시차의 증가에 따라 깊이가 증가하며 crossed disparity의 경우 uncrossed보다 약간 증가량이 큼을 알 수 있다.

Uncrossed disparity의 경우 특히 시차와 관측거리가 클 때 예상깊이에 크게 못미친다.

보여지는 물체 즉 자극의 크기는 기하학적으로는 깊이인식에 영향을 준다고 할 수 없으나 특히 uncrossed disparity의 경우 실제적으로는 깊이에 변화를 준다. <그림 44>과 <그림 45>를 통해 큰 물체 일 때 uncrossed disparity에서 시차에 따른 깊이의 증가가 크며, 작은 물체일 경우 crossed disparity에서 더큰 증가를 보인다. 또 stimuli가 크고, contrast가 작을 때(0.50이하) 오히려 예측된 깊이보다 더 깊이감을 느끼는 경우도 있다.

Crossed direction에서 인식되는 깊이는 예상되는 깊이와 일치하며, 관측거리의 증가에 대해서도 적절히 변화함을 알 수 있다. 그러나 uncrossed direction의 경우 예상된 깊이보다 적은 수치를 나타내고 특히 stimulus가 작고, 관측거리가 길고, parallax가 클때 이러한 현상이 더욱 심해진다.

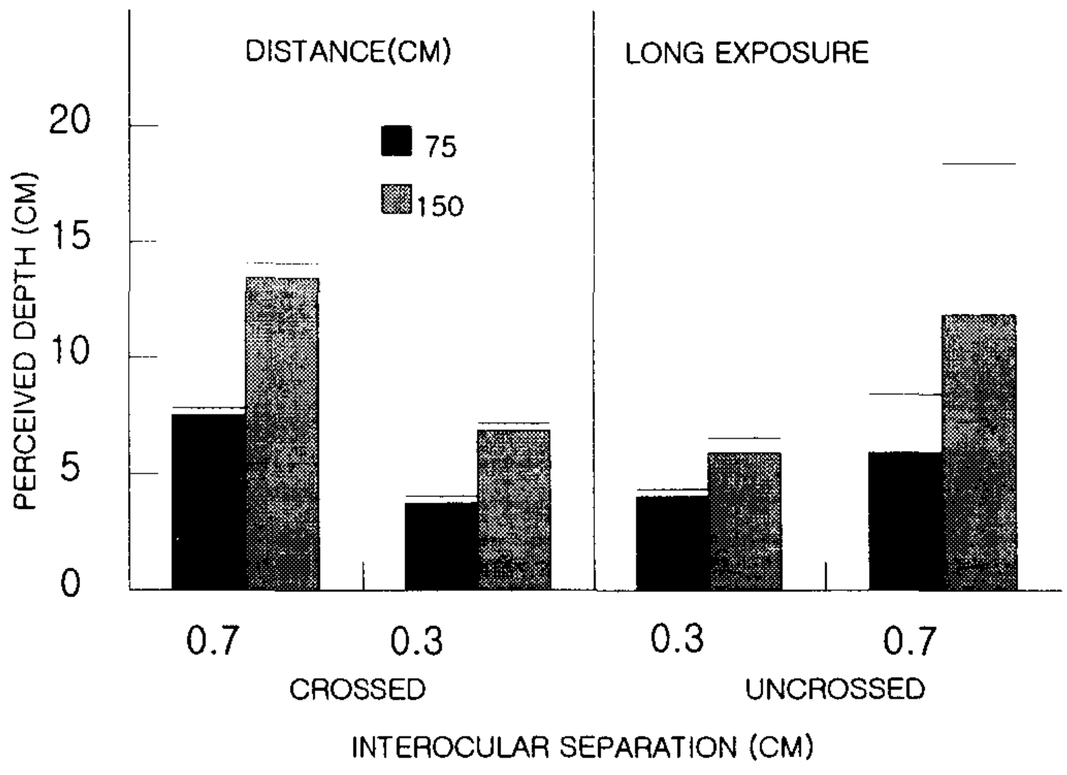
Uncrossed depth의 경우 기하학적인 예측에 일치하지 않더라도 양안 시차에 의해 생긴 눈의 수렴각에 의해 근거하여 변한다고 말할 수 있으나 수렴각을 깊이를 예측하기 위해 사용하는 것은 옳지 않다. 깊이(depth)는 두 물체사이의 공간적 간격을 말한다. Streoscopic depth의 지각은 눈의 수렴위치에 관계없이 상대적인 깊이를 제공하는 disparity로부터 직접적으로 느껴지며 안구의 움직임을 배제한 짧은 노출로부터 정확히 구분될 수 있다.



<그림 43> Brief(160ms) stimulus exposure.

거리(distance)는 관측자와 물체사이의 공간적 간격을 뜻한다. 거리의 지각은 다양한 요인으로부터 이루어지는데 안구의 수렴각의 변화를 포함한다. 이러한 변화는 두 개의 거리사이의 차로써 나타내어지는 깊이정보를 간접적으로 제공한다. 눈의 수렴위치에 의해서도 시차가 발생하지만 이로부터 평가된 거리는 시차로부터 직접적으로 발생한 것이 아닌 자극으로부터 감응된 신경에 의한 것이다.

스테레오스코피에서의 깊이 인식은 눈의 수렴각의 변화에 근거한 깊이 인식과 매우 다른 정보와 작용을 수반하므로 수렴각의 범주에서만 입체시를 다루는 것은 적절치 못하다. 이는 수렴각이 입체시에서 아무런 작용도 하지 않음을 뜻하지는 않는다. 실제로 수렴위치는 두 가지 방법으로 시차에 영향을 주는데, 하나는 horopter의 위치에 영향을 주므로 합성될 시각의 시차를 결정하는 것이고, 또 하나는 수렴각이 시차의 크기조절과 깊이 일관성을 위해 독자적인 거리정보를 제공하는 것이다. 결론적으로 눈의 수렴위치가 입체시에서의 깊이인식의 작용원리는 아니라고 할 수 있다.

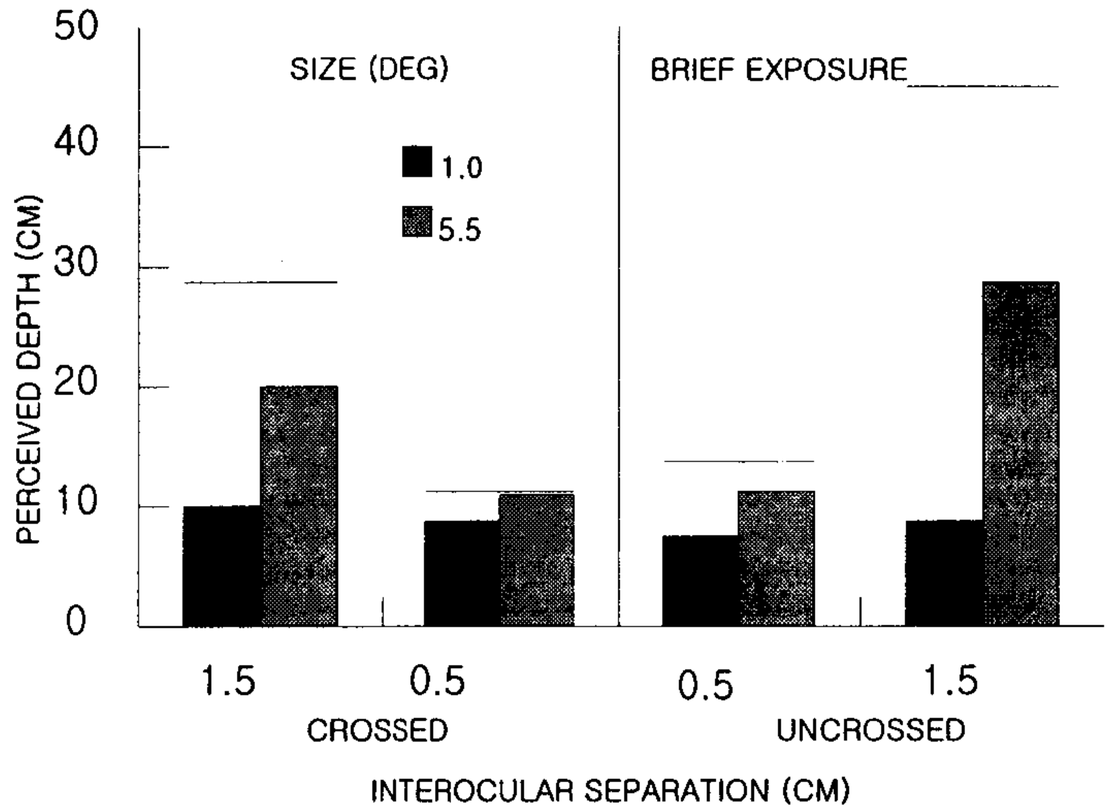


<그림 44> Perceived depth obtained under the crossed and uncrossed separation conditions and two viewing distances. Interocular separation = separation between the half-images of the stereoscopic stimulus. Unlimited stimulus exposure. Horizontal line above each histogram depicts depth predicted by geometry.

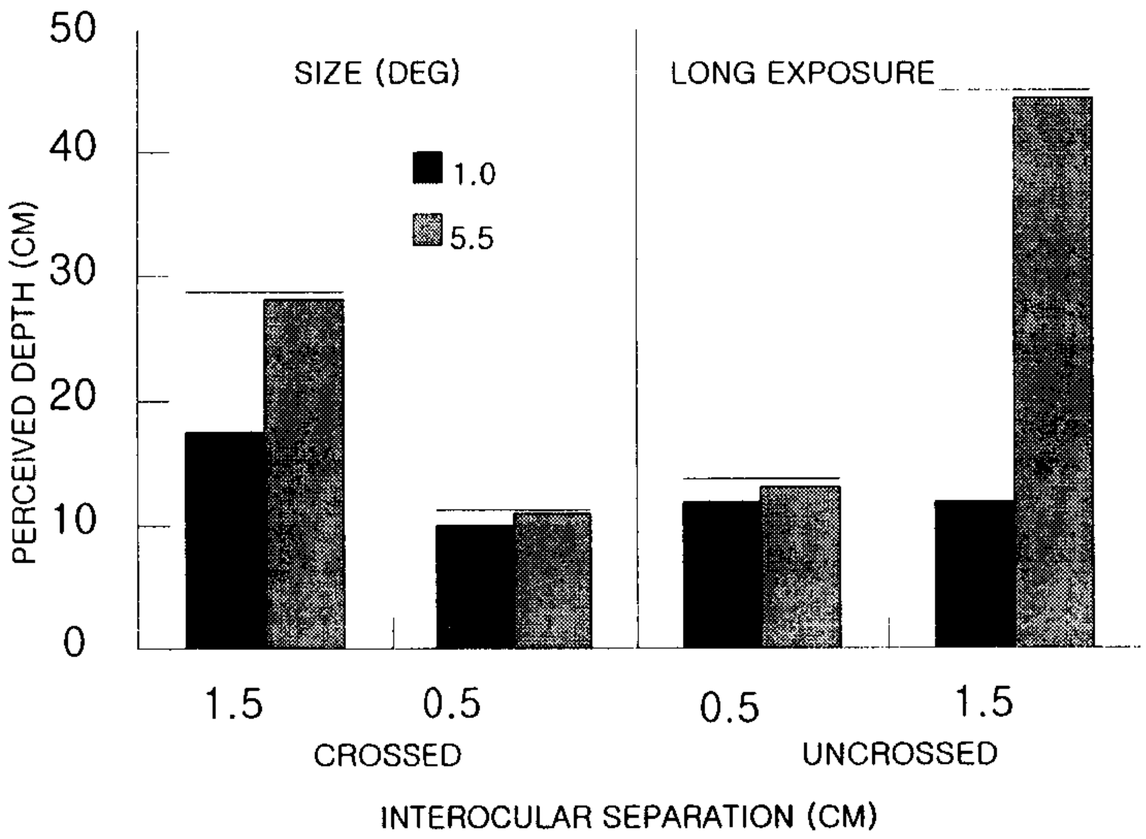
나. 합성의 한계

스테레오스코픽 디스플레이의 효과적인 사용은 그 용도에 따라 다르겠지만 대체로 양안 합성 한계(binocular fusion limits)의 설정과 깊이 인식의 정확도에 의해 좌우된다고 할 수 있다. 삼차원 물체의 표현에 있어서 이차원 디스플레이(2D)나 원근 디스플레이(2.5D)를 사용하는 것보다 실제적인 양안시를 제공하는 것이 입체적인 관계를 나타내는

데 유용하다. 양안시를 통해 인간이 받아들일 수 있는 데이터의 영역은 대개 경험적으로 얻어진다. 이러한 데이터는 순차적인 영상을 표시하는 스테레오스코픽 디스플레이에서 몇 가지 요소를 고려하여 얻어지는데 현재의 구현에서 가장 문제가 되는 것은 첫째, accommodation과 binocular convergence의 불일치로 고정된 초점거리에 대해 양안시차를 직접적으로 다루기 때문에 발생되는 현상이다. 두 번째는 ghost image인데 이는 왼쪽과 오른쪽 화면사이의 crosstalk에 의한 것이다. 이러한 기술 의존적인 문제들은 관측자가 두 개의 분리된 상을 한 개로 합성하는데 있어서 직접적인 영향을 미친다. 또한 망막에의 노출시간(exposure duration)도 영향을 주는데 표 (2-2-2)에서는 200-ms와 2-s가 사용되었다. 200-ms의 경우 상의 합성(fusion)에는 충분하나 안구의 움직임은 작용하지않는 시간이다. 이에 반해 2-s의 경우는 상의 합성에 안구의 움직임까지 적용되는 시간이다.



<그림 45> Brief(160ms) stimulus exposure.



<그림 46> Perceived depth obtained under the crossed and uncrossed separation conditions and two stimulus sizes(the size given in the legend is for one dimension of the stimulus only, height or width). Interocular separation = separation between the half-images of the stereoscopic stimulus. Long stimulus exposure. Horizontal line above each histogram depicts depth predicted by geometry.

CRT를 기반으로 하는 순차영상 디스플레이에서는 (1)셔터가 닫혔거나 열렸을 경우의 투과 광량 (2)형광물질의 지속시간 (3)스크린상에서의 상의 상하위치 등이 양눈의 화면사이의 양안 cross talk의 양을 결정한다. 첫 번째 요소의 경우 반대편 눈의 화면이 새는 양이 직접적으로 ghost image의 intensity와 연관된다. 이러한 상호관계는 대부분의 CRT에서 사용되는 P22형광물질의 적, 녹, 청색의 형광이 서로 다른 시간 비율로 소멸되기 때문에 발생한다. 적색과 청색의 경우 실제적으로 녹색 형광물질에 비해 소멸되는 시간이 짧다. 아래 도표는 적색과 상대적으로 녹색 요소를 많이 포함하고 있는 백색을 통

해 비교되었다. 마지막으로 스크린상의 수직위치는 스크린의 위치에 따라 남아 있는 형광 잔상의 양이 다르기 때문이다. 스크린의 윗쪽에서는 raster scan이 아래쪽까지 이루어지는 동안 형광시간이 끝나게 되나 아래쪽의 경우 현재의 화면이 스캔 되자마자 좌우 영상의 교환이 이루어지므로 형광잔상이 많이 남아 있게 된다. 이러한 양안 cross talk의 양은 올바른 눈의 상의 휘도(luminance)와 원치않는 ghost image의 휘도의 비로 나타내는데 이는 표 2-2-1과 같다.

Vertical Screen Position	Red Image	White Image
Top	61.3 : 1	17.0 : 1
Middle	50.8 : 1	14.4 : 1
Bottom	41.1 : 1	11.0 : 1

200-ms Stimulus Duration(Min of Arc)						2-s Stimulus Duration(Degrees)					
Crossed			Uncrossed			Crossed			Uncrossed		
Red	White	Mean	Red	White	Mean	Red	White	Mean	Red	White	Mean
29.6	28.9	29.3	26.2	25.6	25.9	6.28	3.90	5.09	2.04	1.36	1.70
26.9	26.1	26.5	23.5	23.3	23.4	6.37	3.48	4.93	1.87	1.26	1.57
28.1	23.1	25.6	24.3	22.4	23.4	5.91	3.61	4.76	1.93	0.97	1.45
28.2	26.0	27.1	24.7	23.8	24.3	6.19	3.66	4.93	1.95	1.20	1.57

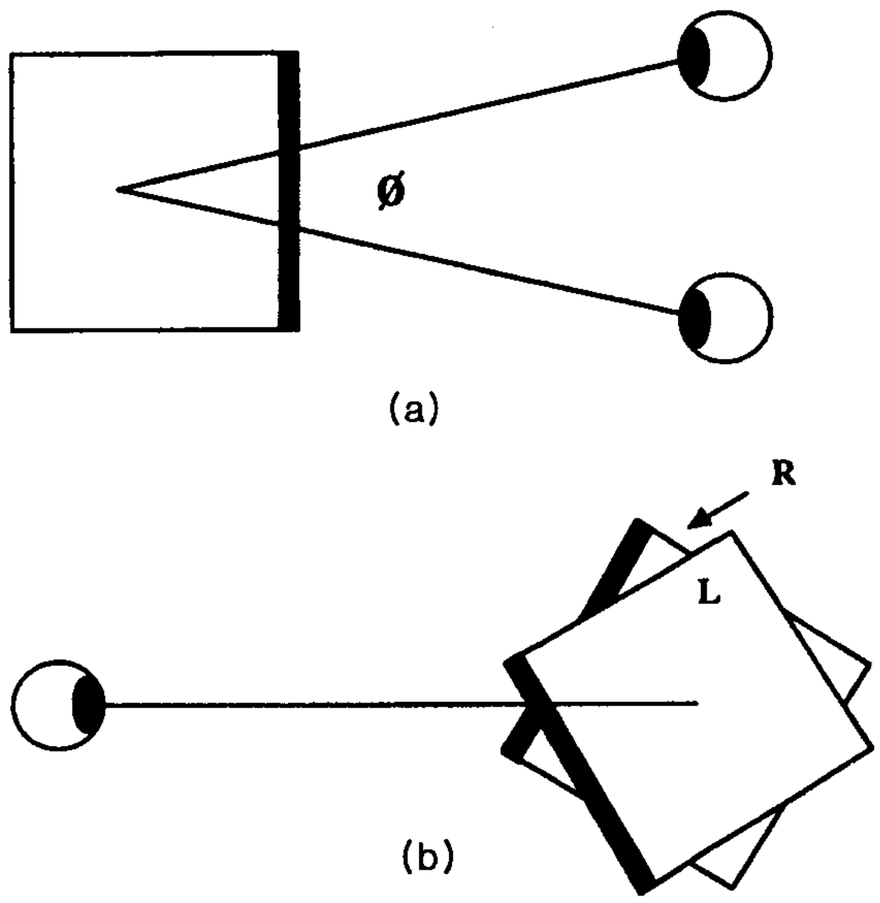
위의 표를 통해 짧은 노출시간일 때 양안 시차 한계(diplopia threshold)는 crossed disparity의 경우 대략 24 min arc, uncrossed disparity의 경우는 27 min arc이고 2초의 노출시간을 주었을 경우 각각 4.93, 1.57 deg로 vergence responses가 보다 넓은 fusion의 범위를 가져옴을 알 수 있다.

2. 입체 영상의 계산

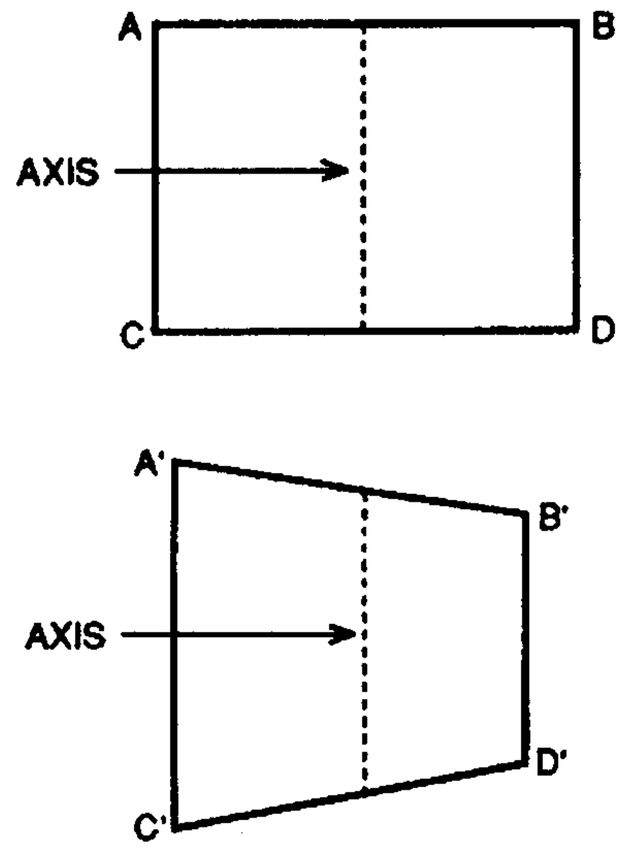
가. 원근 투영의 회전

3-1-1과 같이 3차원적인 영상의 특정한 부분에 관측자가 초점을 맺고 있다고 생각하면 시계의 축은 안쪽으로 약간 기울어져 있으며 응시하는점 P에서 각 phi 를 이루며 만난다. 두 개의 영상을 생성하기 위해서는 <그림 47>(b)에서와 같이 점 P에 대해서 각

phi over 2만큼 왼쪽 영상에 대해서는 오른쪽으로, 오른쪽 영상에 대해서는 왼쪽으로 회전하는 방법을 사용한다.



<그림 47> Stereo views from rotations (a) Right- and left-eye views of a rectangular box. (b) Views created by rotatıng box with respect to a single viewpoint.



<그림 48> Rotation produces distortion.

입체 영상의 쌍을 만들어내기 위한 이러한 방법은 왼쪽과 오른쪽의 원근시야 사이에 <그림 48>와 같이 수직 시차(vertical parallax)를 일으킨다. 이러한 수직 시차의 양은 스크린의 가장자리에서 최대가 되며 중앙부에서는 사라진다. 수직 시차를 줄이기위해 phi 를 감소시키면 깊이효과를 일으키는 수평 시차까지 줄어든다. 또한 투영중심과 회전

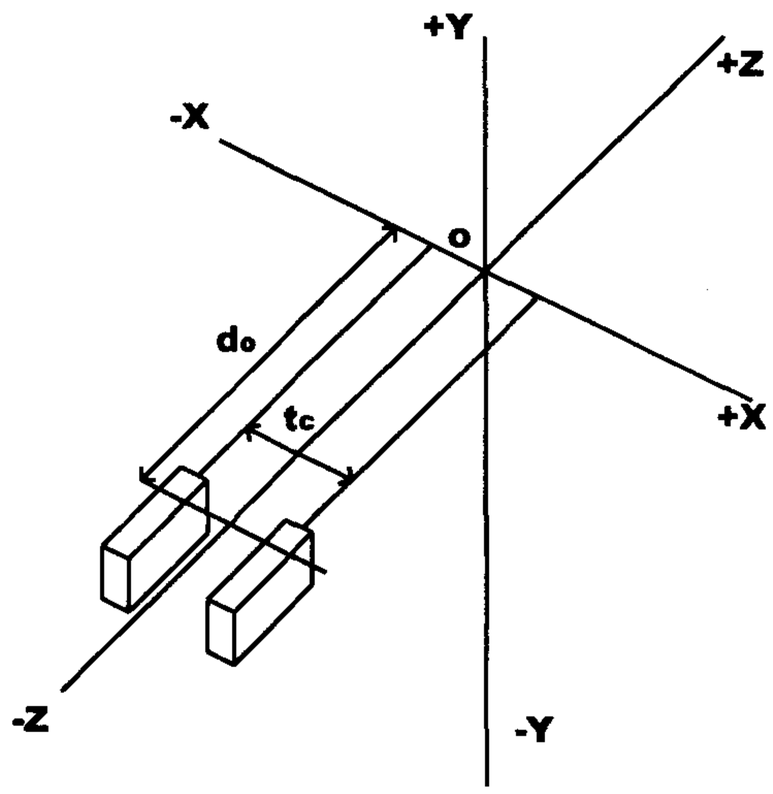
축간의 거리가 R 일 때 R over 2을 반경으로 하는 반원통형의 면이 stereo window가 되므로 평면의 CRT 스크린에 투영되었을 때 깊이관계에서의 왜곡이 발생한다

나. 평행 렌즈축에의한 방법

입체 영상의 계산은 두 개의 단일 영상을 서로 다른 viewpoint에서 그리는 것이다. 과거에는 두 개의 상을 만드는데 물체를 회전시켜 몇 도 기울이는 방법을 사용했는데, 이 방법은 왼쪽과 오른쪽 눈에 대응되는 한 점에 대해 수직방향으로의 시차를 일으킨다. 이러한 수직 시차의 합성(fusion)은 관측자로 하여금 불쾌감을 일으킬 수 있다.

따라서 이를 피하기 위해 수평으로 이동된 두 개의 view를 이용하고 ZPS(zero parallax setting)를 만들기 위해 결과적인 상도 수평적으로 이동하는 방법(HIT - horizontal image translation)을 이용한다.

<그림 49>과 같이 수평으로된 두 개의 카메라를 가지고 있고 이를 한 개의 TV에 표시한다고 하면, 카메라가 tc만큼 떨어져 있기 때문에 정확히 겹쳐지지는 않는 두 개의 상을 볼 수 있다. 두 대의 카메라는 평행한 렌즈축을 가지고 있기 때문에 물체의 서로 다른 부분을 보게 된다. 이 두 개의 상을 상의 한 부분이 겹쳐질 수 있도록 수평이동하면 겹쳐지는 한 부분의 시차는 0이 되며, 스크린상에 나타나게 된다.



<그림 49> Distortion-free images. The camera axes(or the axes of the centers of perspective) must be parallel.

이것을 "convergence"라고 부르는데 이 용어는 생리학에서의 합성을 위한 안구의 회전작용과 쉽게 혼동되므로 HIT로 구분하여 부른다. 상이 수평이동하여 상의 일부가

완전히 겹쳐졌다면 겹쳐진 부분은 ZPS에 있게 된다. 물체의 중심에 ZPS가 있다고 생각하면 물체의 일부는 스크린의 앞쪽에 나머지부분은 스크린의 뒤쪽에서 보이게 된다.

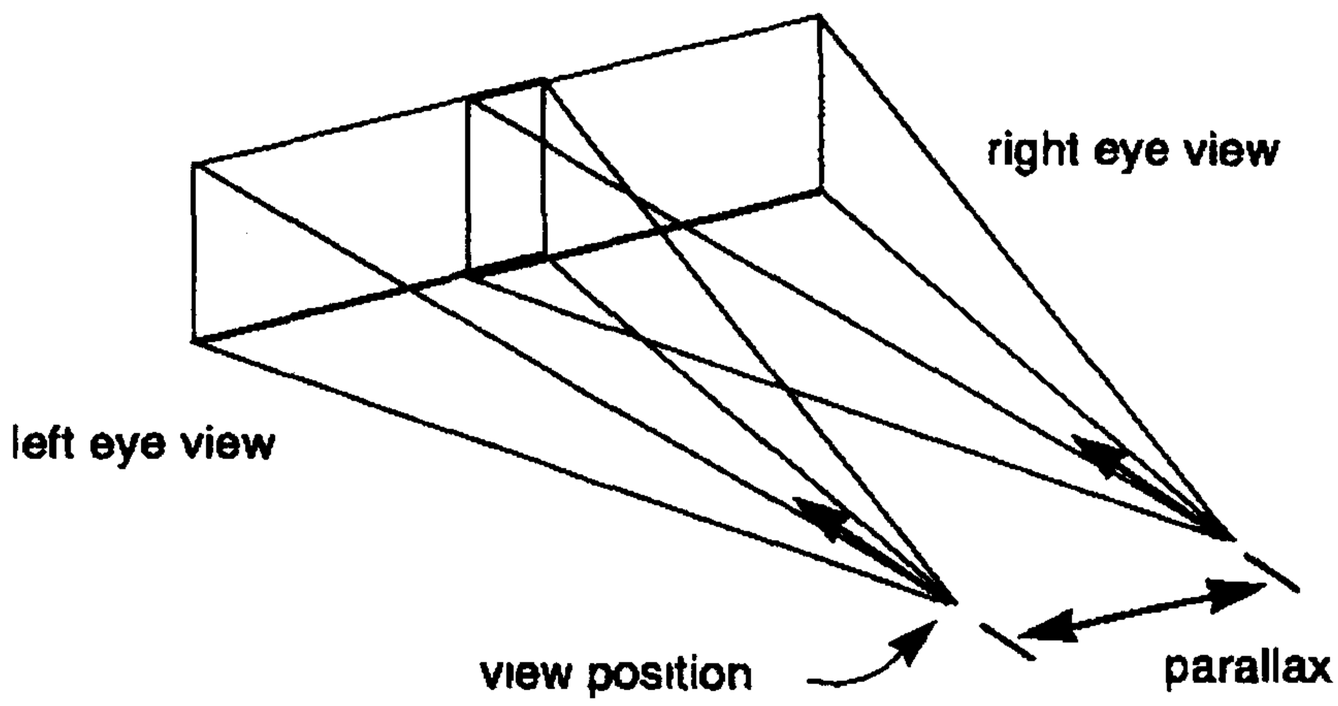
HIT가 사용되더라도 다른 조건(angle of view of lenses, distance of object from the lenses, tc)이 고정된다면 전체적인 깊이에는 아무런 변화가 없다. 다만 관측자는 시차의 변화에 따라 눈의 수렴위치를 다시 바꿀 뿐이다. 이러한 작용이 끝난 후에는 다시 이전과 같은 깊이로 상이 보이게 된다. ZPS에서는 breakdown이 발생하지 않으므로 되도록 이면 물체의 대부분이 ZPS의 근처에 있도록 하기 위해서 물체의 중심에 ZPS를 두도록 HIT하는 것이 좋다. 또 이를 통해 시차의 편중을 감소시킬 수 있으므로 ghosting 효과를 줄일 수 있고 입체감을 높이기 위해 합성 한계(fusion limit)내의 더 큰 시차를 충분히 활용할 수 있다.

깊이 효과를 증대시키기 위해서는 넓은 viewing angle을 사용하고 tc를 작게 사용하는 것이 좋으며 다음과 같은 관계를 가진다. 물체의 가까운 부분과 먼 부분의 상대적인 위치가 과장되기 때문인데 이러한 원근의 과장은 깊이 효과를 크게 한다.

다. WorldToolKit 에서의 응용

(1) World coordinate에서의 parallax를 통한 screen parallax의 근사적 계산

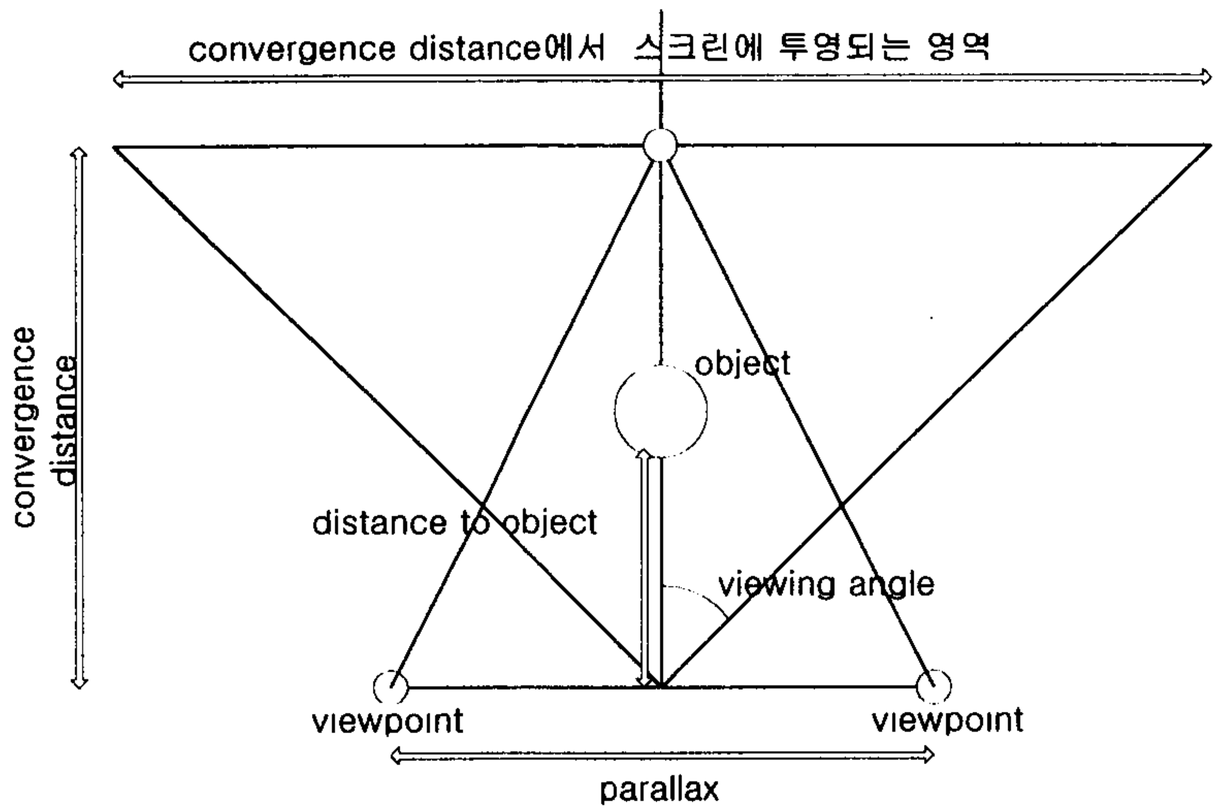
WorldToolKit에서 사용되는 parallax는 <그림 50>에서와 같이 왼쪽과 오른쪽의 viewpoint사이의 3D 가상세계에서의 간격이며 이 world coordinate에서의 거리 단위를 사용한다. 즉 시차가 0이면 두 viewpoint의 위치가 같은 경우이다.



<그림 50> Stereoscopic viewing

Convergence는 왼쪽과 오른쪽의 영상의 수평 변위이며 픽셀단위로 조절된다. 이에 반해 convergence distance는 parallax와 같이 world coordinate에서의 값으로 주어지며 물체가 스크린상에 나타나는 위치 즉, ZPS가 되는 거리이다. 그러나 convergence를 convergence 거리를 통해 조절하는 경우 parallel lens axis algorithm이 아닌 비대칭 투영법(asymmetric projection)을 이용한다. 비대칭 투영은 convergence 거리를 바꿈으로써 디스플레이장치의 전후에 상을 옮길 수 있기 때문에 매우 유용하다.

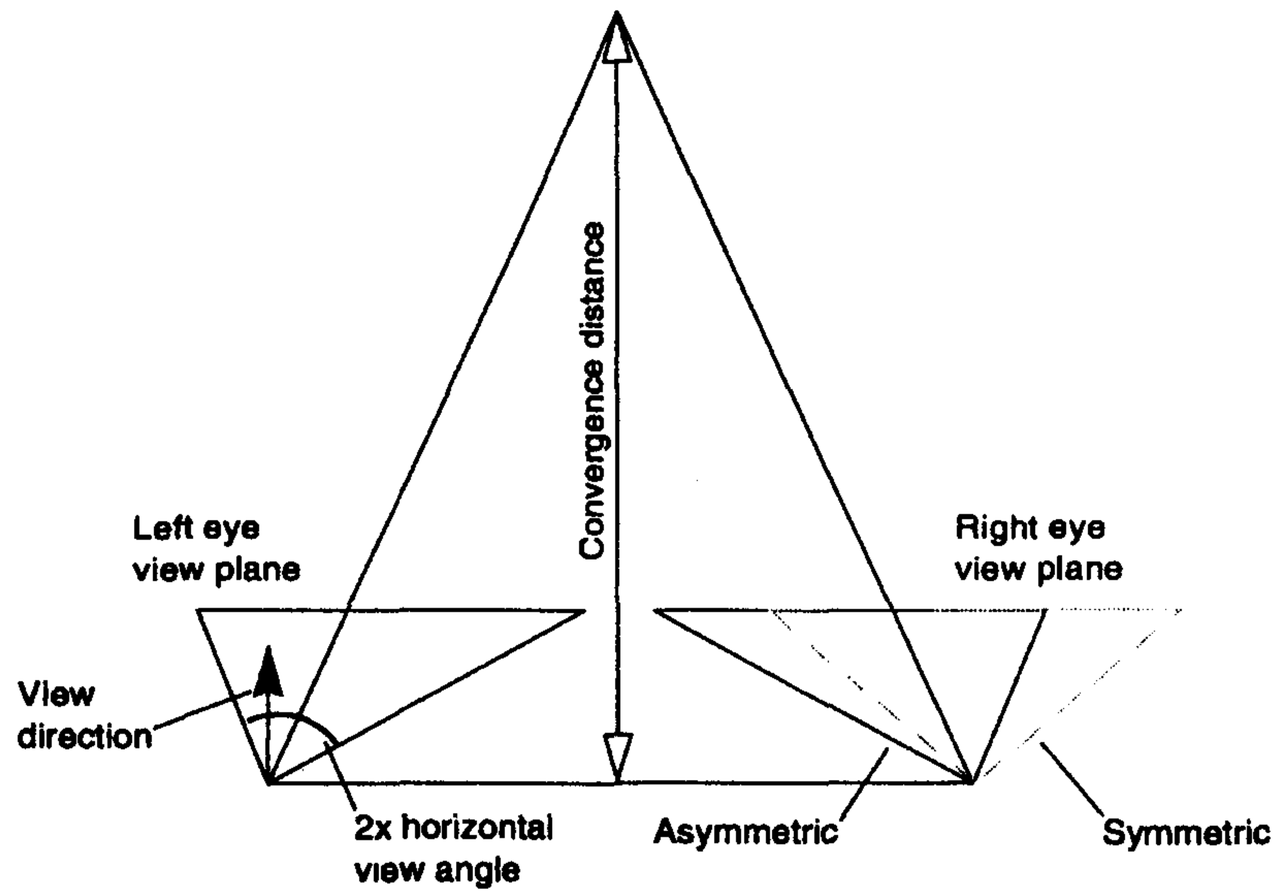
실제로 스크린에 보여지는 parallax의 값은 스크린의 크기에 따라서 달라지며 다음과 같은 근사적인 관계를 통해 world coordinate에서의 parallax를 viewpoint로부터의 거리 d_obj 에 있는 물체에 대한 screen parallax로 바꾸어 생각할 수 있다(<그림 51>).



<그림 51> World coordinate에서의 parallax를 통한 screen parallax의 근사.

(2) ZPS의 위치 설정

ZPS(zero parallax setting)은 왼쪽과 오른쪽의 대응되는 점이 parallax를 가지지 않는 지점으로 parallel lens algorithm의 경우 두 단안 영상의 수평 이동으로 설정할 수 있고 비대칭 투영의 경우 convergence distance의 조절로써 바꾸어질 수 있다(<그림 52>). 이를 통해 같은 크기의 parallax를 이용해서도 제공하는 깊이는 같으나 거리에 있어서만 차이를 가지는 입체영상을 만들어 낼 수 있다. 이러한 과정은 입체영상에서 ZPS를 어떤 위치에 놓는가에 따라 결정되는데 ZPS가 물체의 앞에 있을 경우 상은 스크린의 안쪽에 생기게 되며 뒤에 있을 경우 스크린의 밖으로 돌출되어 보인다.



<그림 52> Top view of stereoscopic viewing pyramid with asymmetric projection.

거리의 지각은 안구의 수렴각의 변화를 통해 주로 이루어지는데 이는 경험적인 정보만을 제공할 뿐이므로 ZPS의 위치가 바뀌더라도 전체적으로 지각되는 영상에 있어서는 차이가 없다고 할 수 있다. 또한 적절한 ZPS의 위치는 positive와 negative parallax의 분산을 가져오므로 crosstalk를 감소 시킬 수 있다.

실제로 Convergence distance의 조절을 통해 crosstalk를 감소 시킬 수 있었으며 보다 큰 parallax를 사용할 수 있으므로 깊이효과 증대를 가져왔다. 그러나 풍경 또는 건물 내부등과 같이 가까운 물체가 모니터 가장자리의 수직한 부분(vertical screen surround)에 의해 잘려지는 경우 비록 negative parallax를 가지더라도 CRT space에 있는 것으로 인식되며 오히려 깊이감의 저하를 가져온다. 따라서 화면의 보여지는 물체의 위치 관계를 파악하여 ZPS면이 이들의 중심부에 놓이도록 convergence distance를 조절하는 것이 좋으나 단일 또는 몇 개로 무리를 이루는 물체가 아닌 경우 가장 가까운 물체에 ZPS를 놓아 전체적으로 positive parallax를 가지도록 하거나 관측자가 관심을 두는 부분이 화면의 중심부임을 고려하여 약간의 negative parallax를 사용 할 수도 있다. Convergence distance는 viewpoint의 위치 변화에 따라 실시간으로 물체의 위치를 파악한 뒤 얻어지는데 수행속도는 저하된다. <그림 53>에 실제적인 응용예를 보였다.

라. 깊이의 측정

(1) 입체영상의 최적화

어떤 거리에 있는 물체를 응시할 경우 안구는 특정한 수렴각으로 회전하게 되는데 이러한 작용을 convergence라고 하며 수정체의 두께를 조절하여 이 물체위에 초점을 맺는 과정을 accommodation이라고 한다. 그러나 스테레오스코픽 디스플레이의 경우 parallax의 방향과 값에 따라 안구는 임의의 위치에 수렴하게 되나 초점은 항상 스크린위에 맺게 되며 이는 일상적인 생활에서 발생하는 accommodation과 convergence의 관계(A/C relationship)와는 다른 형태를 가지게 된다. 이러한 A/C의 관계가 크게 벗어나게 되면 관측자는 두 개의 영상을 하나로 합성하지 못하며 눈에 불쾌감을 느끼게 된다. 따라서 A/C의 관계를 유지하기 위해서는 parallax의 방향 및 크기의 적절한 조절이 필요하다.

스테레오스코픽 디스플레이의 효과적인 사용은 대체로 양안 합성 한계(binocular fusion limits)와 깊이 인식의 정확도에 의해 좌우 된다고 할 수 있다. 양안시를 통해 인간이 받아들일 수 있는 데이터의 영역은 대개 경험적으로 얻어지며 이러한 영역은 A/C의 불일치와 ghost ımage에 의해서 결정된다. 합성 한계의 양은 대개 각도로 나타내어진 parallax의 크기로 표시하는데 일반적으로 1.5o정도이다. 보고에 의하면 positive parallax의 경우 합성 한계는 1.57o, negative parallax의 경우는 4.93o인데 입체영상 제작자들은 negative parallax도 1.5o근처의 합성 한계치를 사용하고 순간적으로 돌출한 뒤 사라지거나 화면속으로 들어가는 물체의 경우에만 이 값의 2-3배정도를 사용할 것을 권고한다.

깊이(depth)는 두 물체사이의 공간적 간격을 뜻하는데 깊이의 지각은 눈의 수렴위치에 관계없이 상대적인 깊이를 제공하는 망막시차로부터 직접적으로 느껴지며 안구의 움직임을 배제한 짧은 노출로부터 구분될 수 있다. 거리(distance)는 관측자와 물체사이의 공간적 간격을 뜻한다. 거리의 지각은 다양한 요인으로부터 이루어지는며 안구의 수렴각의 변화를 포함한다. 이러한 변화는 두 개의 거리사이의 차로써 나타내어지는 깊이정보를 간접적으로 제공한다.

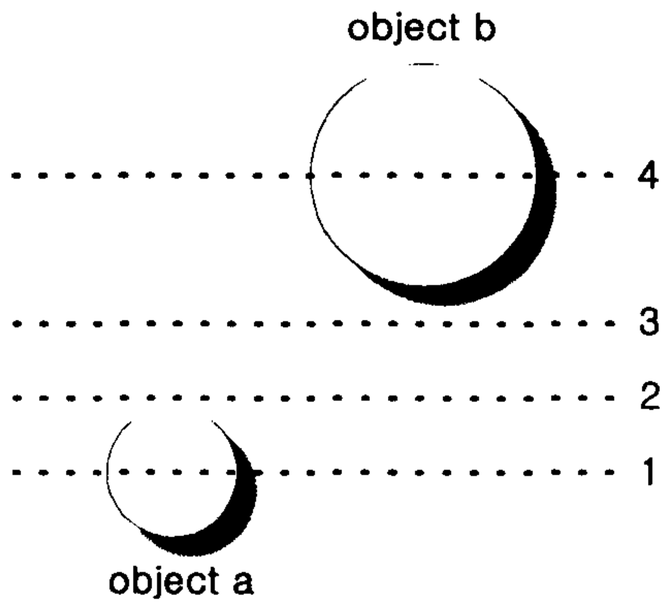
스테레오스코픽 디스플레이에서의 깊이 인식에 영향을 주는 요인은 parallax의 크기, parallax의 방향(negative, positive), 표시되는 물체의 크기, 관측자의 스크린으로부터의 거리 등으로 크게 나누어 질 수 있다. Parallax에 의해서 유발되는 입체감은 물체들간의 상대적인 깊이이며 깊이 인식에서의 가장 큰 관심사는 인지되는 깊이의 정확성인데 관

측거리에 따라 느껴지는 깊이가 달라지므로 관측거리에 따라 parallax의 크기를 재 설정하여야 하며 이와 같이 관측거리에 관계없이 정확한 깊이를 나타내는 것을 depth constancy라고 한다.

ZPS(zero parallax setting)는 물체중 screen parallax를 가지지 않는 곳인데 화면의 바로 위에 상이 보이게 되며 이를 중심으로 parallax를 가지는 부분은 화면 안쪽과 바깥쪽으로 나뉘어져 나타난다. 물체공간에서의 ZPS의 위치를 조절함으로써 전체적인 parallax의 양을 바꾸어 화면에 대한 상의 위치를 임의로 옮길 수 있으며 이를 이용하여 parallax 방향의 편중을 막고 crosstalk의 감소를 통해 깊이감을 높일 수 있다.

(2) 깊이의 측정

서로 다른 공간적 위치 및 parallax를 가지는 두 개의 물체를 test stimuli로 사용하였고 ZPS의 위치에 따른 물체간의 깊이 변화를 관찰 하였다. 두 물체는 크기에 의한 깊이 효과를 배제하기 위하여 화면상에서 동일한 크기 및 색상을 가지도록 하였다. 물체는 동일한 크기로 화면상에서 보여지기 위해 서로 다른 크기로 모델링 되었으며 화면상에서의 크기는 관측거리 70cm에서 4.50× 10.36 deg이며 화면의 크기 325× 240mm의 17inch 모니터를 사용하였다.



<그림 53>사용된 ZPS의 위치

측정 방법은 verbal method와 probe method를 사용하였다. Verbal method는 피실험자가 화면을 기준으로한 물체의 위치를 관측거리의 백분율로 말하는 방법이며 probe

method는 피실험자가 probe의 위치를 움직여 물체와 동일한 깊이에 놓이도록 조절하도록 한 뒤 화면으로부터의 거리를 측정하는 방법이다.

ZPS의 위치는 <그림 53> 과 같이 4가지로 나누어 측정하였다.

① 물체는 모두 화면의 안쪽에 보인다. 이때 물체 b는 합성 한계를 1.57o로 할 때 최대의 positive parallax를 가진다.

② 최대의 negative와 positive parallax의 크기는 같다.

③ 물체 a, b의 중간위치.

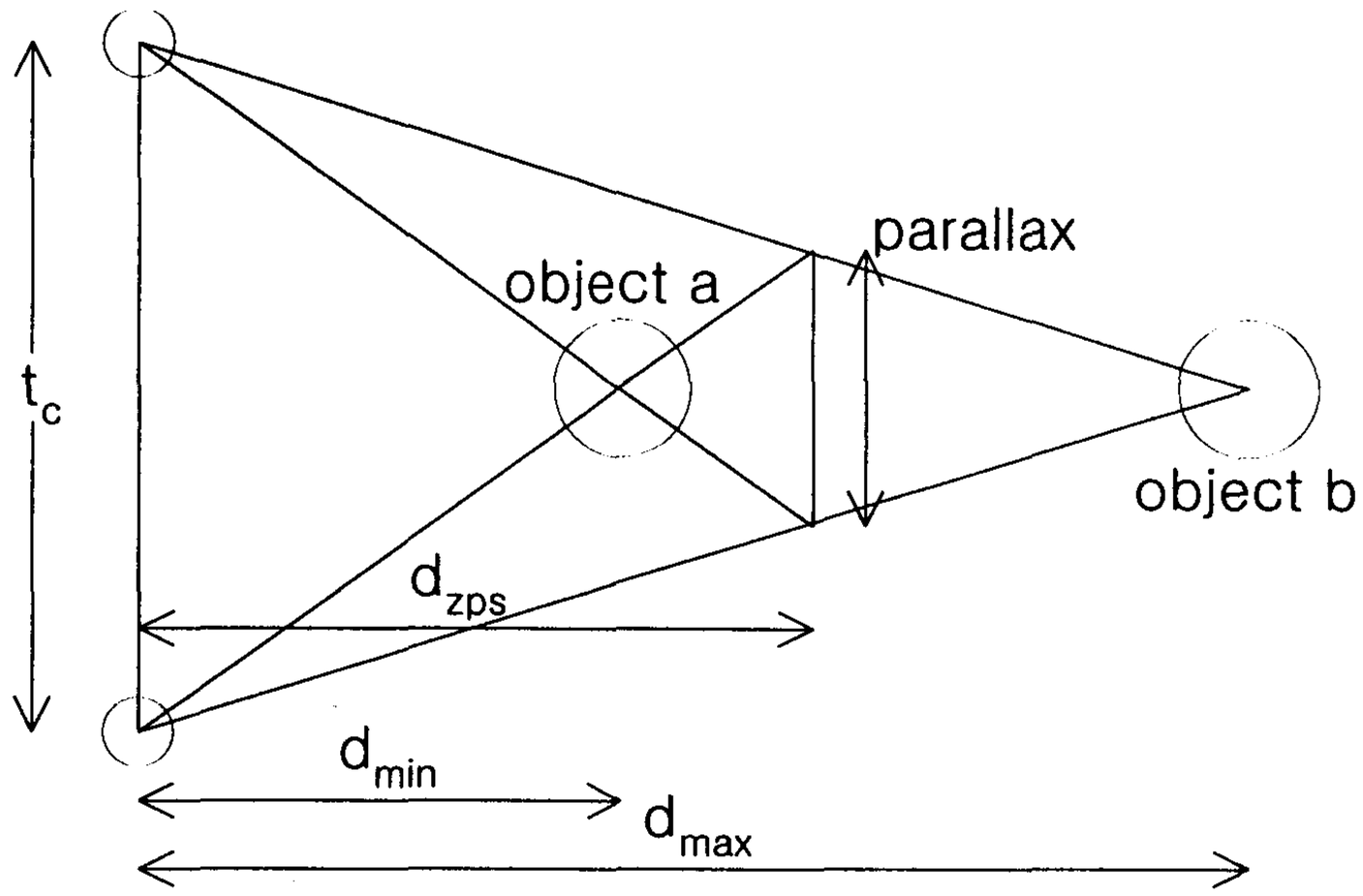
④ 전체적으로 negative parallax를 가지며 물체 a에서 최대치가 된다.

3. 결과 및 논의

Verbal method의 경우 피 실험자에 따른 편차가 심했으며 negative parallax의 경우 기하학적으로 예상되는 깊이보다 더 큰 값을 가지는 overconstancy가 관측되었다. 상대적으로 probe method는 보고된 수치와 대체로 일치하는 결과를 보였으며 피 실험자간의 편차가 작았다. 특히 positive parallax의 경우 보고된 수치에 못미치는 결과를 나타내었는데 이는 합성 한계의 값을 사용했으므로 합성에 어려움을 가졌기 때문으로 생각된다. 반면 negative parallax의 경우 대체로 일치하는 결과를 보였는데 역시 1.57o를 기준으로 하였으므로 한계치 4.93o보다 작은 값의 사용으로 상의 합성이 수월했음을 나타낸다(<그림 54>).

Verbal method의 경우 피 실험자에 따른 편차가 심했으며 negative parallax의 경우 기하학적으로 예상되는 깊이보다 더 큰 값

ZPS의 이동에 따른 물체 a와 b의 상대적인 깊이는 기하학적인 계산에의한 예상치는 위치에 따라 매우 큰 차이를 보이며 깊이감에 있어서의 왜곡을 일으킬 것으로 생각되었으나 실험적인 수치는 ①②③④각각 14.92, 13.58, 13.92, 13.15로 전체적으로 negative parallax를 가질 경우 약간 줄어드는 경향을 보이나 매우 적은 양이므로 ZPS의 변화는 깊이에 큰 영향을 주지 않는 것으로 생각된다.



<그림 54> ZPS의 위치 계산

물체의 중심위치에 ZPS를 두었을 때 netative parallax의 경우 화면상에서 12mm, positive parallax의 경우 6mm정도의 값을 지니며 관측된 깊이는 각각 9.00cm와 4.92cm로 parallax 및 깊이의 편중이 있음을 알 수 있다.

Negative와 positive의 최대 parallax가 같아지는 위치에 ZPS를 놓는 경우 8mm정도의 screen parallax를 가지며 이에 대해 기하학적으로 예상되는 깊이는 각각 7.67과 9.82cm가 되는데 실험적으로 각각 6.25와 7.33cm의 값을 얻었다. 따라서 parallax의 크기를 같게 할 경우 parallax 뿐만아니라 깊이 또한 viewer space와 CRT space로 적절히 분산됨을 알 수 있다.

ZPS의 위치를 바꿈으로써 기하학적인 작도에 의해서는 영상안에서의 물체간의 깊이감 및 물체와 관측자 사이의 거리감의 왜곡이 있을 수 있으나 실제적으로 인지되는 깊이에 있어서는 이에 의한 영향이 없음을 실험적으로 확인하였다.

3차원 영상기술을 구현하는 기술은 감성공학평가기술 특히 3차원 시각환경구현기술에 중요하다. 3차원 영상을 구현하는 방법에는 크게 스테레오스코피 기술을 사용하는 방법과 홀로그램을 사용하는 기술로 나눌 수 있다. 그러나 궁극적으로는 홀로그래피 기술과 스테레오스코픽 기술을 혼합한 홀로그래픽 스테레오스코피 기술이 필요할 것으로 판단된다. 이러한 목적을 달성하기 위해서는 무엇보다도 스테레오스코피 기술에 대한 보다 구체적인 연구가 필요한데, 그 중에서도 인간의 시각에 영향을 미치는 시각파라미

터에 대한 구체적인 평가와 보완작업이 필요하다. 따라서 본 연구에서는 스테레오스코픽 3차원 영상의 시각파라미터를 평가하고 보완하는 연구를 단계적으로 수행하였다.

본 연구는 CRT를 기반으로 하는 스테레오스코픽 시스템에서 인지되는 깊이에 영향을 미치는 시각 요소들을 분석하고 parallax의 분배 및 인식되는 깊이와 합성 한계에 큰 영향을 미치는 ZPS의 위치를 적절히 선정하기 위한 연구를 하였다. 실험에 의한 통계적 결과를 통해 ZPS의 위치 변화가 물체간의 상대적인 깊이에 미치는 영향이 매우 작음을 볼 수 있으며 가장 타당성 있는 ZPS의 위치는 negative와 positive parallax의 최대값의 크기가 같아지는 지점으로 선택해야함을 알 수 있었다.. 여기서 얻은 ZPS의 위치는 두 개의 viewpoint의 축이 교차하는 지점까지의 거리인 convergence distance와 같으며 VR software에서 물체의 최소거리와 최대거리를 파악한 후 실시간적으로 쉽게 응용될 수 있다.

Head-Up Display는 항공기 또는 자동차에 응용이 되는 장치이나 가상세계를 실제 외부영상과 연결하기 위해 see-through HMD에 응용하기 위한 개발이 계속 되고 있다. 따라서 본 연구에서는 HUD 또는 HMD에 응용될 수 있는 홀로그래픽 광 결합기를 설계, 제작하고 평가하였다.

이러한 연구를 바탕으로 스테레오스코픽 3차원 영상의 거리감을 실감있게 구현하는 작업이 후속으로 연구되어야 할 것이다. 이들 바탕위에 홀로그램기술을 결합하여 홀로그래픽 스테레오그램 기술을 개발하면 감성평가에 필요한 가상의 3차원 영상과 공간을 실현할 수 있다고 판단된다.

제 4 절 Driving Simulator의 평가수법 개발

1. 국내외 기술개발 현황

Simulator는 산업의 여러 분야에서, 특히 사고 발생 위험이나 경제적으로 피해기 막대한 분야에서 주로 개발되어 작업자의 훈련이나 사용 시스템의 특성분석에 폭 넓게 활용되어 왔다. 현재에는 컴퓨터의 발전과 소프트웨어의 기술발전에 따라 응용의 영역을 넓히고 있으며 특히 자동차 관련 기술개발 분야에 이용되기 시작하면서 신차의 개발

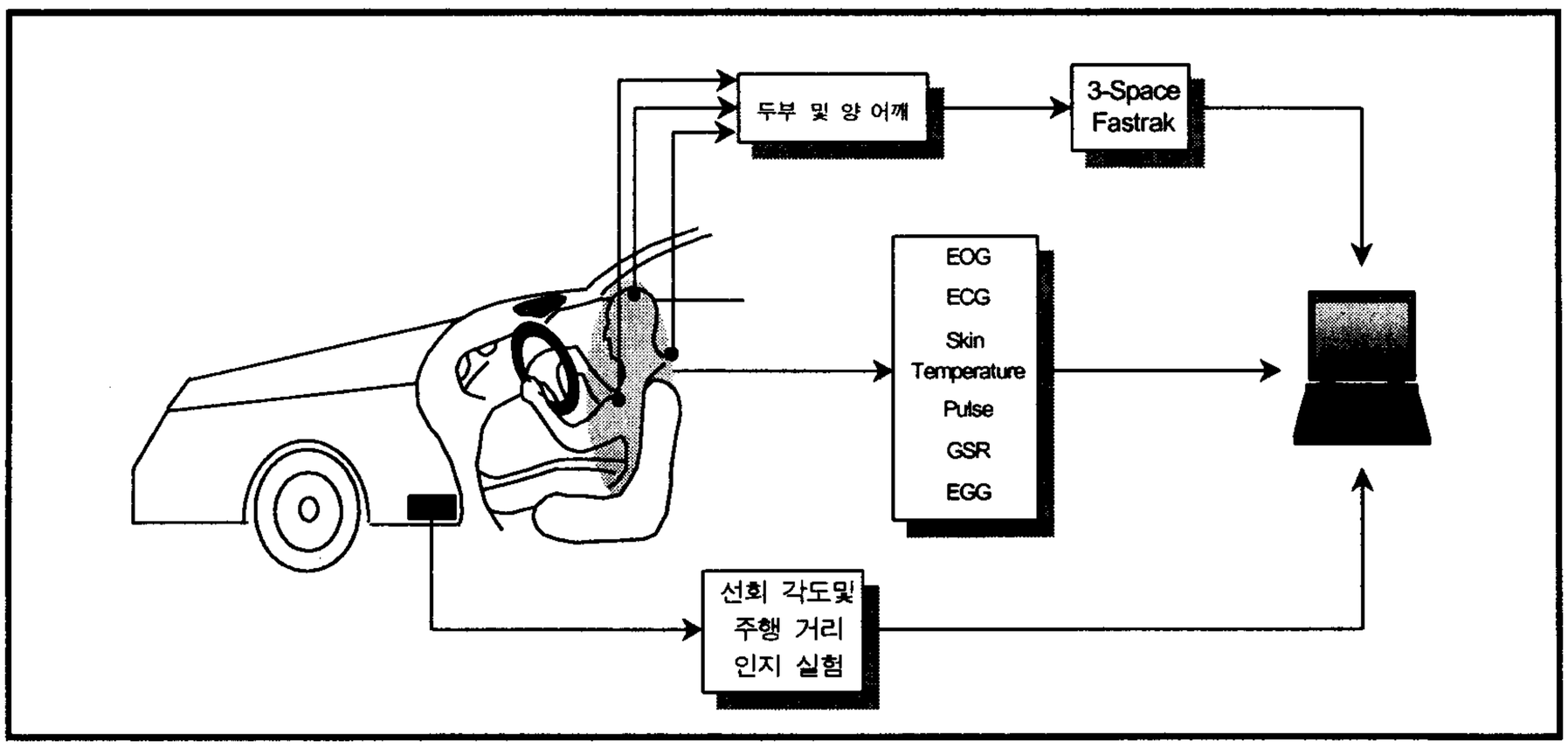
등에 유용하게 쓰이고 있다. 복잡한 차량 시스템 때문에 1980년대에 개발되기 시작한 Driving Simulator 는 오늘날 다양한 Driving Simulator들이 개발되었다[38]. 이에 따라 자동차 회사에서는 신차의 설계시 인간공학 및 감성공학 기술 응용의 중요성이 널리 확산됨에 따라 그 이용이 더욱 확대될 것으로 전망된다. 국내에는 항공산업계에서 조종사 훈련용으로 Flight Simulator를 도입하거나 일부 개발되어 사용하고 있지만 자동차 설계분야에 이용될 수 있는 수준의 Driving Simulator는 개발이 진행되고 있는 상태이다[39]. 하지만 아직 개발된 Simulator가 실제로 어느 정도의 현실감을 나타내는지를 판가름할 수 있는 방법은 감성공학적인 측면에서는 아직 발표되어 있지 않은 상황이다.

2. 연구개발 수행내용 및 결과

가. 연구개발 수행내용

(1) 실험장치의 구성

<그림 55>은 실차 및 Driving Simulator에서 주행 중에 운전자가 받는 일련의 자극들에 대한 반응으로 일어나는 각종 생체신호 및 운전자의 거동을 측정하기 위한 실험장치의 개략도이다. 실험장치는 생체신호 측정 장치와 운전자 거동측정 장치, 가속도 측정 장치로 구성되어 있다. 또한 운전자가 느끼는 주행거리 및 선회각 인지 실험을 실제 차량에 탑재하여 주행을 하면서 실험을 할 수 있게 구성하였다.



<그림 55> 생체신호 측정장치 시스템의 구성

(2) 휴대형 생체 신호 측정 장치

자동차 주행 중에는 여러 가지 주변의 자극에 신체가 노출됨으로 인하여 인간의 생리 상태가 변하게 된다. 이런 자극들은 운전자에게 차량의 상태와 위험을 경보하는 역할도 하지만 인간에게 불쾌한 자극은 자칫 멀미 등의 원인이 되기 때문에 인간의 생리 상태에 대한 중요한 정보를 제공하는 생체신호(ECG, EOG, EMG, RESP, Skin Temp., EGG) 등을 측정함으로써 주행 중에 일어나는 각종 생리 상태를 정량적으로 판단할 수 있다. 이런 생체 신호를 휴대형 생체 신호 처리 장치를 자동차 안에 구축하여 자동차가 실제

주행하면서 운전자가 받는 각종 자극에 대한 생리 상태를 측정했다. 운전자의 생체신호는 운전자의 운행에 지장이 없는 범위에서 다음과 같은 전극 부착 위치를 선정했다.

(가) 전극 부착 위치

- ECG : 왼쪽 가슴 부위

- EOG : 운전자 눈의 깜빡임 측정과 안구의 좌우 움직임측정을 위해 수평, 수직 방향으로 전극 부착

- EMG : 스티어링 조작과 긴장도 측정을 위해 왼쪽 팔 위에 전극을 부착

- RESP : 호흡센서를 코에 부착

- EGG : 운전자의 복부에 전극을 부착

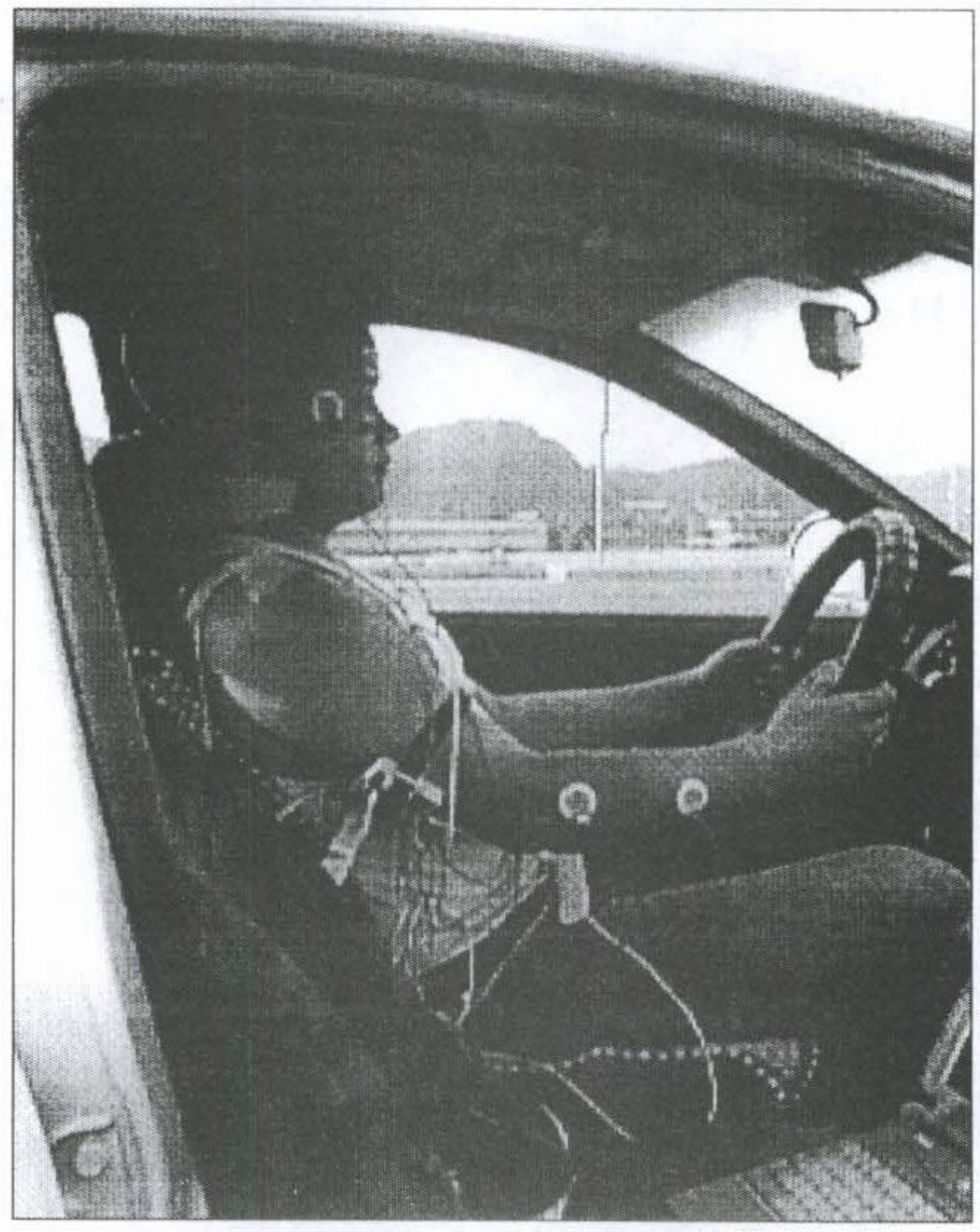
(나) 시스템 구축 계측기 사양

- 휴대용 생체 계측기 ·6채널,

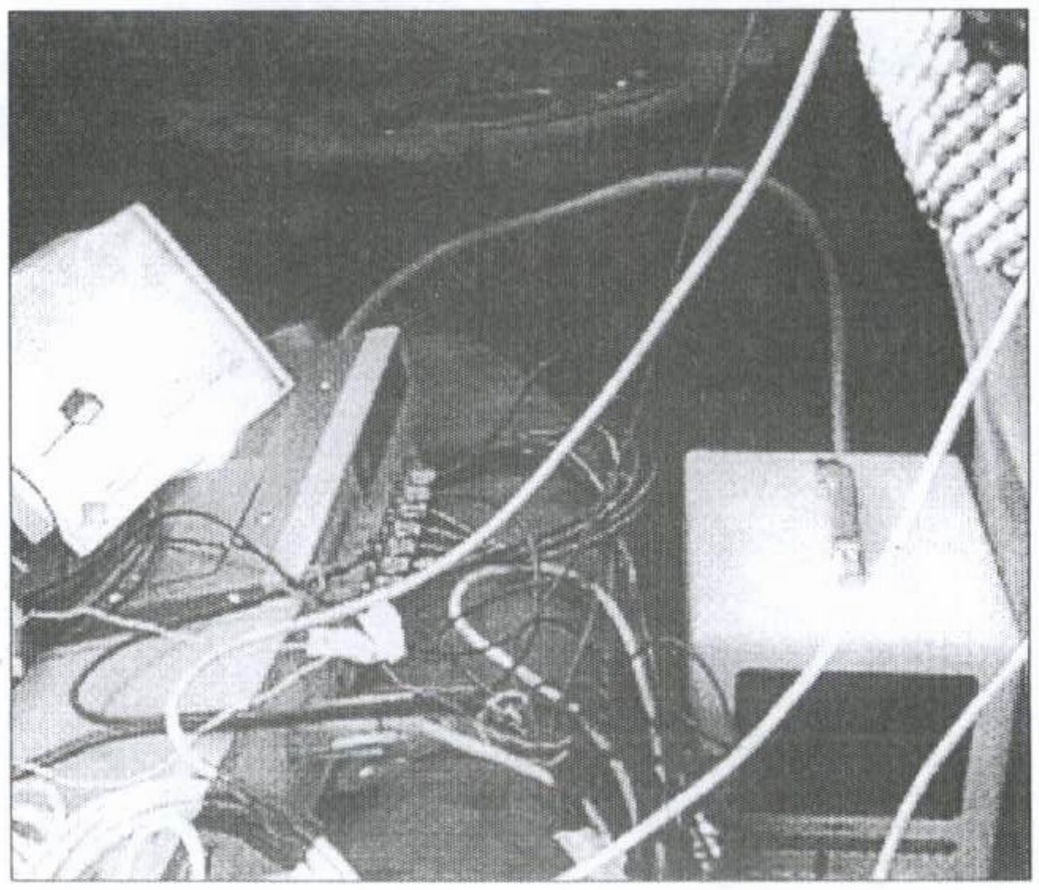
- 무선 측정 범위 약 30m

- 데이터레코더 ·DAT 방식, 8채널

<그림 56>는 운전자 생체 신호를 위한 전극과 거동 측정 장치를 부착하고 운전하는 그림이며 <그림 57>은 각종 장치들이 차량의 뒷좌석에 설치되어 있는 모습이다.



<그림 56> 운전자 생체신호 측정모습



<그림 57> 차량에 탑재된 계측장비 및 거동측정

(다) 신체 거동 측정 장치

자동차의 주행 시 항상 변화하고 있는 가속도 및 선회에 의하여 피동적인 신체의 움직임이 발생함으로 운전자의 머리 및 양 어깨의 동요를 측정함으로써 신체의 움직임을 정량적으로 측정할 수 있게 된다. 즉, 운전자는 자동차가 가속될 때와 등속 주행 시, 그리고 감속될 때 앞, 뒤로 움직이게 됨으로 실제 차량에서 받는 가감속의 정도와 Simulator에서 받는 가감속의 정도를 각각의 움직임의 차이로 간접적으로 유추해 볼 수 있게 된다. 머리와 어깨의 움직임은 자기 3차원 위치 센서인 3-SPACE FASTRAK을 이용하여 측정했다. 운전자의 거동측정 장치도 자동차 내에 직접 설치되며 컴퓨터로 데이터를 수집, 분석한다.

(라) 3-SPACE FASTRAK의 사양

- 측정정도 : 0.08㎝, 0.15°
- 측정범위 . 최대 305㎝
- Interface : RS232C

(3) 실차 주행시의 가속도의 변화의 측정

자동차 운전 중에 일어나는 가속도의 자극은 주로 내이에 존재하는 전정감각에 의하여 감지된다고 알려져 있으며 Driving Simulator에서도 운동판을 이용하여 신체 전체를 움직이도록 함으로써 인간에게 가속도를 부여하게 된다. 이와 같이 운동판에 의한 가속도의 제시를 위해 실제 자동차의 데이터를 수집하여 이를 Driving Simulator에 이용할 수 있다[40].

(가) 사용센서 사양

- KYOWA사, 1축, 측정범위 2G

(4) 주행거리 및 선회각도 인지실험

자동차의 운전 중에 일어나는 가속도 및 각가속도의 자극은 주로 내이에 존재하는 전정감각에 의하여 감지되고 있다고 알려져 있다. 따라서, Simulator에서도 인간에게 가속도 및 각가속도의 감각을 제시하기 위해서는 전정감각기를 자극할 수 있는 장치가 필요하게 되며, 현재 대부분의 simulator에서는 운동판(motion base)을 통하여 신체 전체를 움직임을 부여하여 인간에게 가속도 및 각가속도를 제시하고 있다.

이와 같이 운동판에 의한 가속도 및 각가속도의 제시에 의한 운동의 정보가 인간에게 어느 정도 정확하게 전달되는가를 정량적으로 평가하기 위하여, 실차에서 부여한 가속도 및 각가속도 운동에 의한 주행거리 및 선회각도의 인지와 운동판을 이용하여 가상적으로 제시한 (각)가속도의 인지를 비교 분석할 필요성이 있다. 실차를 이용해서 조수석에 피험자를 앉힌 후 눈을 가린 상태에서 직선주행 및 선회주행을 실시하여 피험자가 인지하는 주행거리 및 선회각을 응답하게 한 후, 실제로 주행한 거리 및 선회 각도를 비교하도록 한다.

나. 실험방법

(1) 운전자 생리신호 및 거동측정 실험

휴대형 생체 신호 계측 장치와 신체 거동 측정 장치, 그리고 가속도 센서를 운전자와 자동차에 설치하여 실제 도로에서 실험했다.

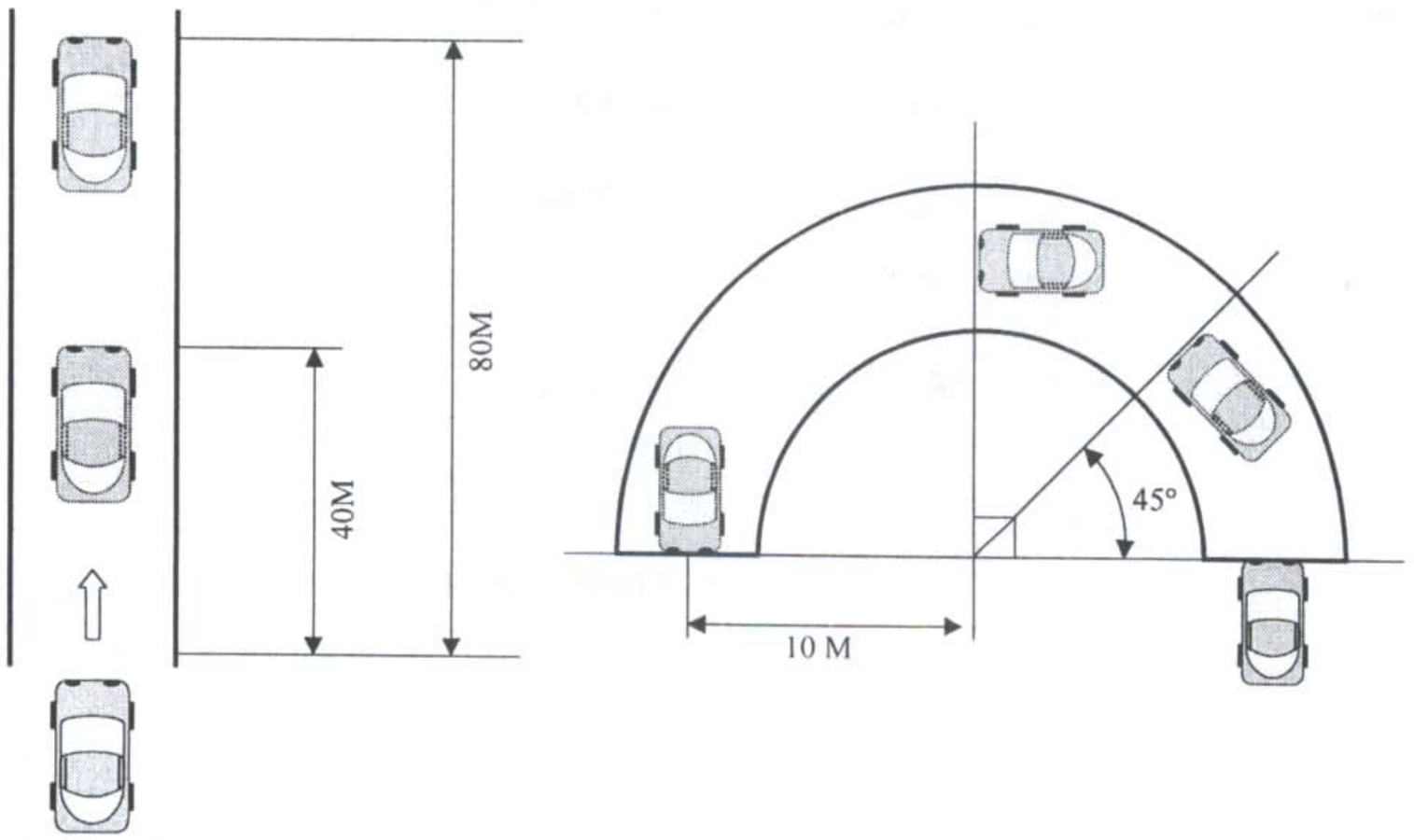
주행환경은 전주~논산간 고속도로와 전주~남원, 진안 간 국도에서 실험을 진행했다. 실험 차량은 아반떼, 액셀, 액센트로 3명의 운전자를 상대로 실험을 했다. 실험방법은 상술한 장치들을 차량 뒷좌석에 설치하고 생체신호를 측정하기 위한 전극을 운전에 방해되지 않은 범위에서 부착하였다. 가속도 센서는 자동차 실내의 중심부인 운전자 오른쪽에 있는 콘솔박스에 부착했다. <그림 58>는 전방의 상황을 알 수 있게 실험 중에 비디오 카메라로 전방을 녹화하여 커브나 신호등 등의 영향을 고려했다.

실험방법은 주행 전 5분간 생체 신호를 측정하고 20분간 주행 중에 생체 신호와 가속도를 측정한 다음 주행 후 다시 5분간 생체 신호를 측정하였다.

<그림 58> 비디오 카메라 촬영

가속도의 측정은 기어 변속 없이 자동차를 가속시켜 센서를 통해 얻어진 데이터를 자동차 Simulator에서 얻어진 데이터와 비교 분석을 위해 여러 조건의 데이터를 저장했다. 주행형식은10초간 정지상태에서 60㎞/h로 가속하고 10초간 등속 주행, 그리고 10초간 60㎞/h에서 정지하는 형식으로 실험을 진행했다.

(2) 거리 인식 및 선회각도 인식실험



<그림 59> 거리 및 선회각도 인지실험

- 직선 주행 패턴

<그림 59>와 같이 직선주행은 40M와 80M의 거리를 인지하는 실험을 하였다. 차량의 속도는 최대 15km/h이하로 하고 운행중 기어의 변속은 하지 않았다. 먼저 실험에 앞서 조수석에 앉은 피험자는 눈을 뜬 상태에서 2번의 예비 운행을 갖은 뒤 본격적인 실험을 진행하였다. 눈을 감은 상태에서 각각의 거리를 인지하였을 때 준비된 부저로 신호를 보내, 그 위치를 표시하였다.

- 선회 주행 패턴

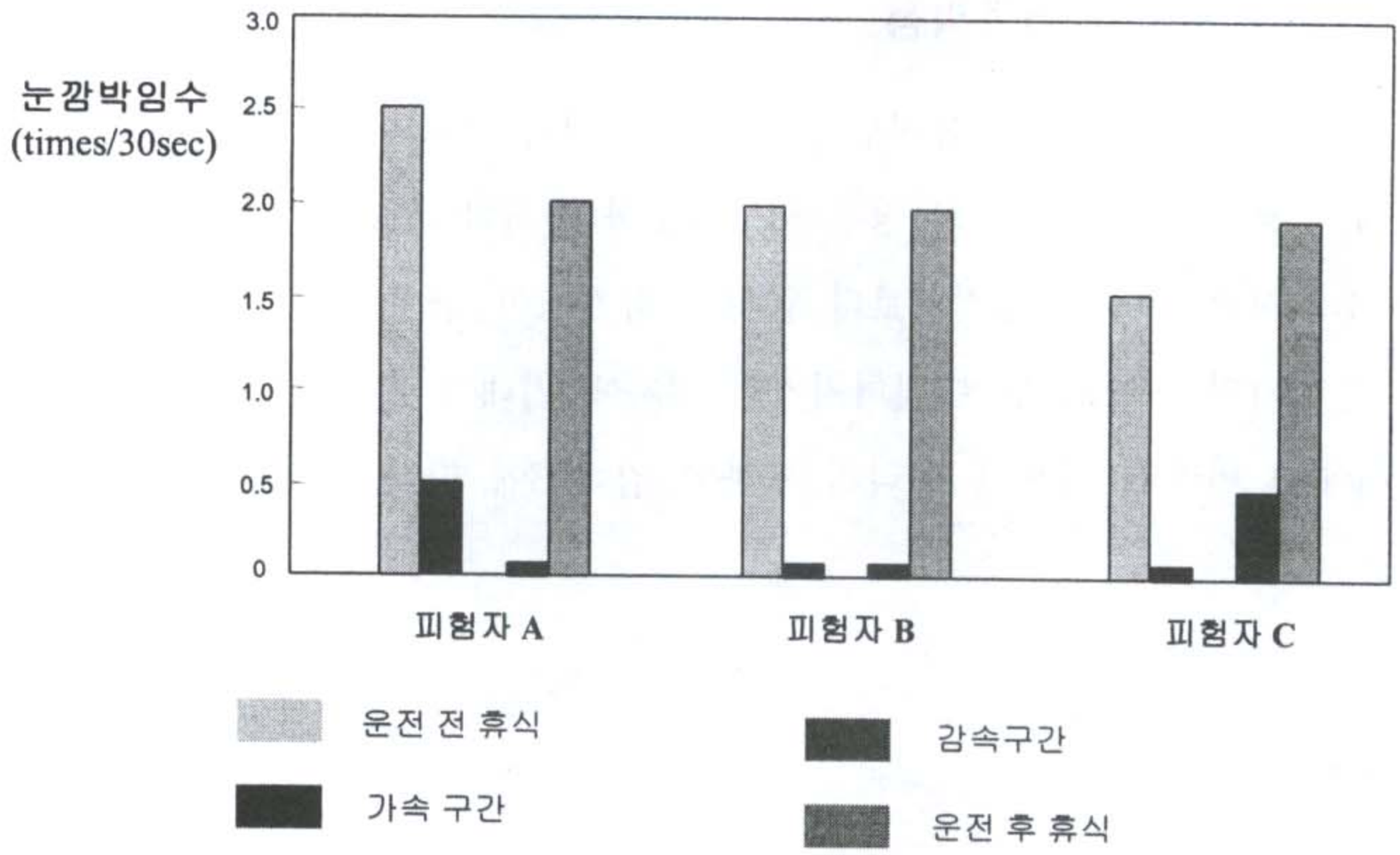
직선 주행에서와 마찬가지로 선회주행에서도 그림과 같은 반지름 10M의 선회구간을 피험자는 눈을 뜬 상태에서 2번 예비 주행을 실시하였다. 예비 주행을 마치고 피험자는 폐안한 상태에서 45°, 90°, 135°, 180°의 인지각도를 응답하게 했다.

다. 실험결과

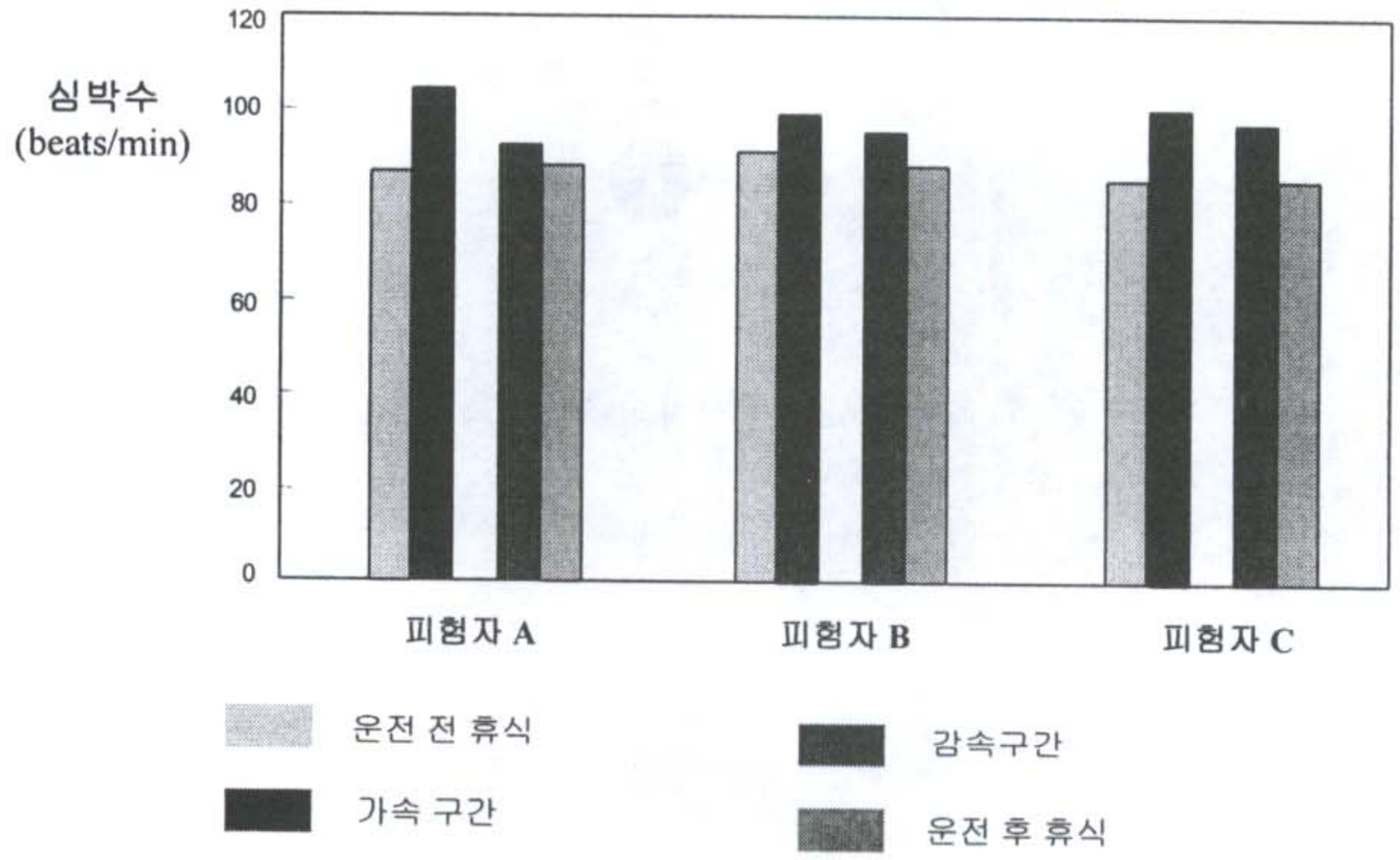
(1) 운전자 생리신호

자동차에서 계측된 각각의 신호들은 실험실에서 처리되어 실제 차량의 상태에 따른 운전자의 신호와 차후에 Simulator에서 받는 자극에 반응한 운전자의 생체 신호와 비교하여 Driving simulator를 평가할 수 있는 파라메터를 유도한다 . 본 연구에서는 EOG, ECG신호의 유효성을 평가했는데 자동차 전방 상태에 따라 운전자의 깜박임이 변화한다[41]는 실험결과를 유도했다. <그림 60>은 운행 중 2분간의 깜빡거림의 변화를 보여주는 그림으로 운전자가 신호등을 발견한 뒤 현저하게 깜빡거림이 저하한다는 것을 알 수 있다.

운전 중 긴장 상태에 따라 HR(Heart Rate)이 변화한다는 연구결과가 알려져 있다[42]. 본 연구에서도 심박수의 변화를 측정했다. <그림 61>은 운전 전, 가속구간, 감속구간, 운전 후의 심박수의 변화를 나타내는 그래프로 운전전후의 심박수보다 가속이나 감속구간에서 심박수가 증가함을 볼 수 있다. 이는 가속이나 감속구간에서 운전자가 보다 긴장되어 있다는 것을 보여준다. 다른 생체 신호(EMG, RESP)등은 뚜렷한 변화를 보여주지 않아 Driving Simulator의 평가 지표로 EOG와 ECG로 운전자의 생리 상태의 변화를 알 수가 있었다.



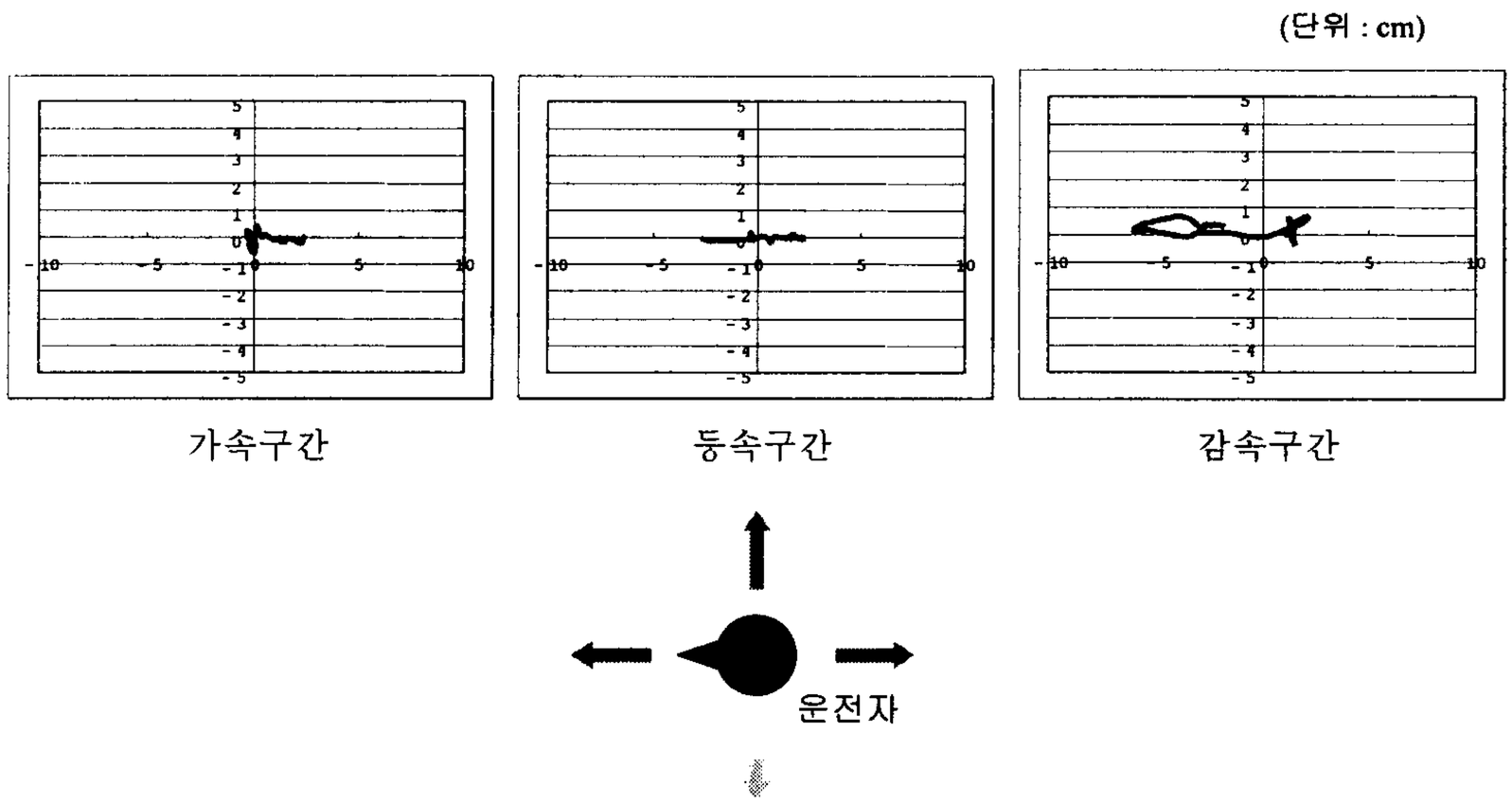
<그림 60> 운전 전후의 눈깜박임수의 변화



<그림 61> 운전 전후의 심박수의 변화

(2) 운전자 거동측정

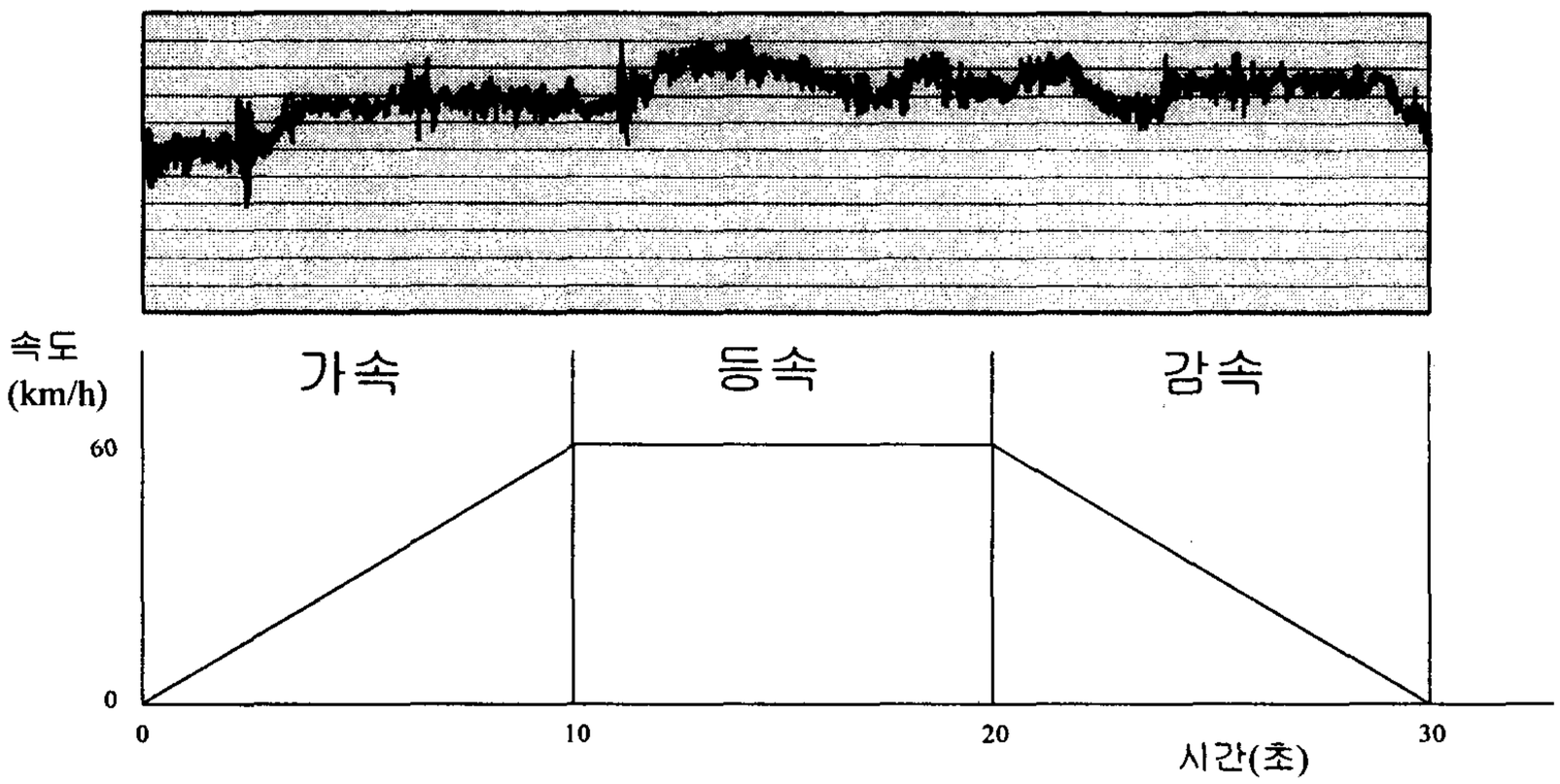
<그림 62>은 머리의 움직임을 측정한 것이다. 가속이나 등속주행시보다 감속할 때 움직임이 많아진다. 양어깨의 경우에도 비슷한 결과가 유도되었다. 실험은 직선 도로에서 실시하여 좌우의 움직임보다 앞뒤의 움직임이 상대적으로 많이 나타났다. 가속시에는 운전자의 머리가 뒤로 젖혀지지만 의자에 기대게 됨으로 움직임은 한계가 있지만 감속시에는 머리가 앞으로 쏠리기 때문에 감속도에 따라서 앞으로의 움직임은 많아진다.



<그림 62> 운전자의 거동 측정 결과

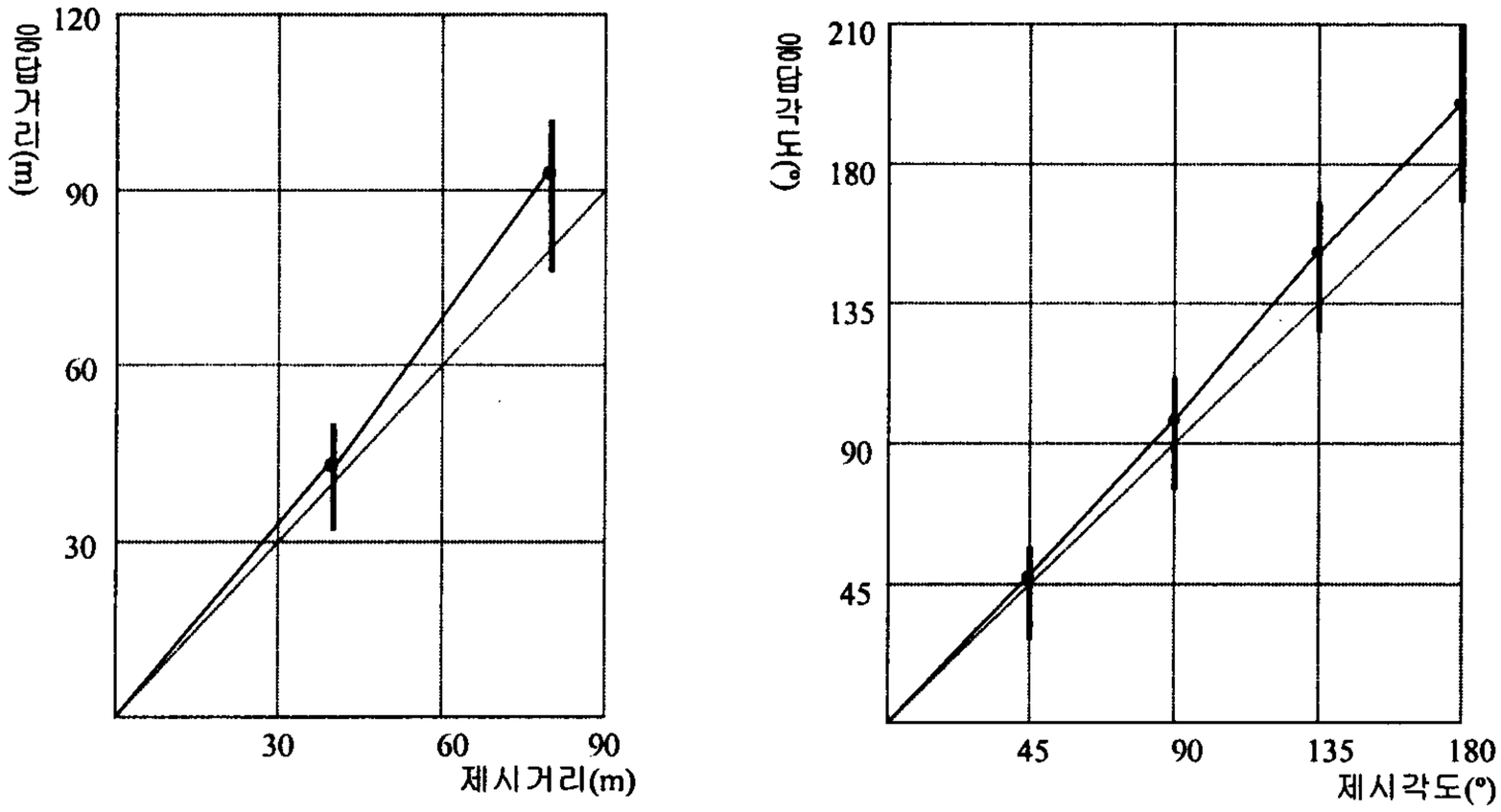
(3) 가속도 측정

가속도 센서를 이용하여 자동차의 가속도를 측정하였다. <그림 63>은 실측한 가속도를 나타내고 있다. 이 가속도 신호는 후에 Driving Simulator에 직접 적용할 수 있는데 운동판의 움직임에 대한 입력신호로 실제와 비슷한 운동감을 느끼게 할 수 있을 것이다. 본 실험에서는 아직 1축으로 전후의 가속도만을 측정하였지만 후에 전후, 좌우, 상하 3축으로 가속도를 측정하면 더욱 개선된 입력자료로 활용할 수 있다.



<그림 63> 자동차 가속도의 변화

(4) 거리 및 선회각도 인지실험



<그림 64> 주행거리 및 선회각도 인지실험 결과

<그림 64>는 주행거리 인지실험과 선회각도 인지실험 결과를 나타내는 그래프이다. 남자 대학생 10명을 대상으로 주행거리와 선회각도 인지실험을 한 결과 평균적으로 큰 오차없이 응답하였으나 인지한 거리나 선회각도보다 실제로는 더 멀리 주행하게 되었다. 실험 전에 도로상에 거리 및 선회각도를 표시하고 눈을 뜬 상태에서 예비 실험을 갖았기 때문에 큰 오차는 발생하지 않은 것으로 생각된다. 하지만 각 개인의 차이는, 주행거리 인지실험에서는 약 20~30m, 선회각도는 30~40°의 차이를 보여주었다. 이후에 Simulator 상에서 이와 같은 실험을 실시할 때에는 디스프레이상에 거리 및 선회각도를 인지할 수 있게 해주는 구조물, 예를 들면, 가로등, 가로수 등을 꼭 설치를 해주어야 한다는 결론을 얻었다.

라. Bike Simulator의 개발

인간이 자세 균형 제어를 하는 데에는 시각, 전정감각, 체성감각 등이 필요하다. 마찬가지로 자전거를 타기 위해서도 이들 세 가지 감각이 필수적이다. 즉, 자전거는 자세 균형 제어에 필요한 감각들의 통합적 상호 작용면에서 효과적이다. 여기에 가상 현실 기술을 사용하여 단조로움을 피할 수 있게 한다. 가상 현실 기술은 인간의 오감에 신호를 전달하여 실제는 존재하지 않는 가상 환경을 만들어 인간이 그 가상 환경과 상호작

용을 하게 함으로써 인간의 통합된 감각 기관을 자극하는데 효과적인 방법이다[43]. 즉, 가상환경 속에 있는 피험자로 하여금 실제 환경에 들어와 있는 듯한 느낌을 갖게 함으로써 가상 공간에 몰두하게 하여 가상 공간의 화면에 따라 감각 기관이 작용하여 실제 환경에서 일어날 수 있는 반응을 일어나게 하는 것이다.

<그림 65>은 제작된 Bike Simulator의 구성도이다. 먼저 가상 공간의 설계 및 제작은 3D Studio와 월드툴킷(WTK)등을 사용하여 가상 공간의 설계 및 제작을 하였다. 이러한 프로그램등을 이용하여 Bike 시뮬레이터를 위한 가상 도로를 만들어 실험에 이용한다.

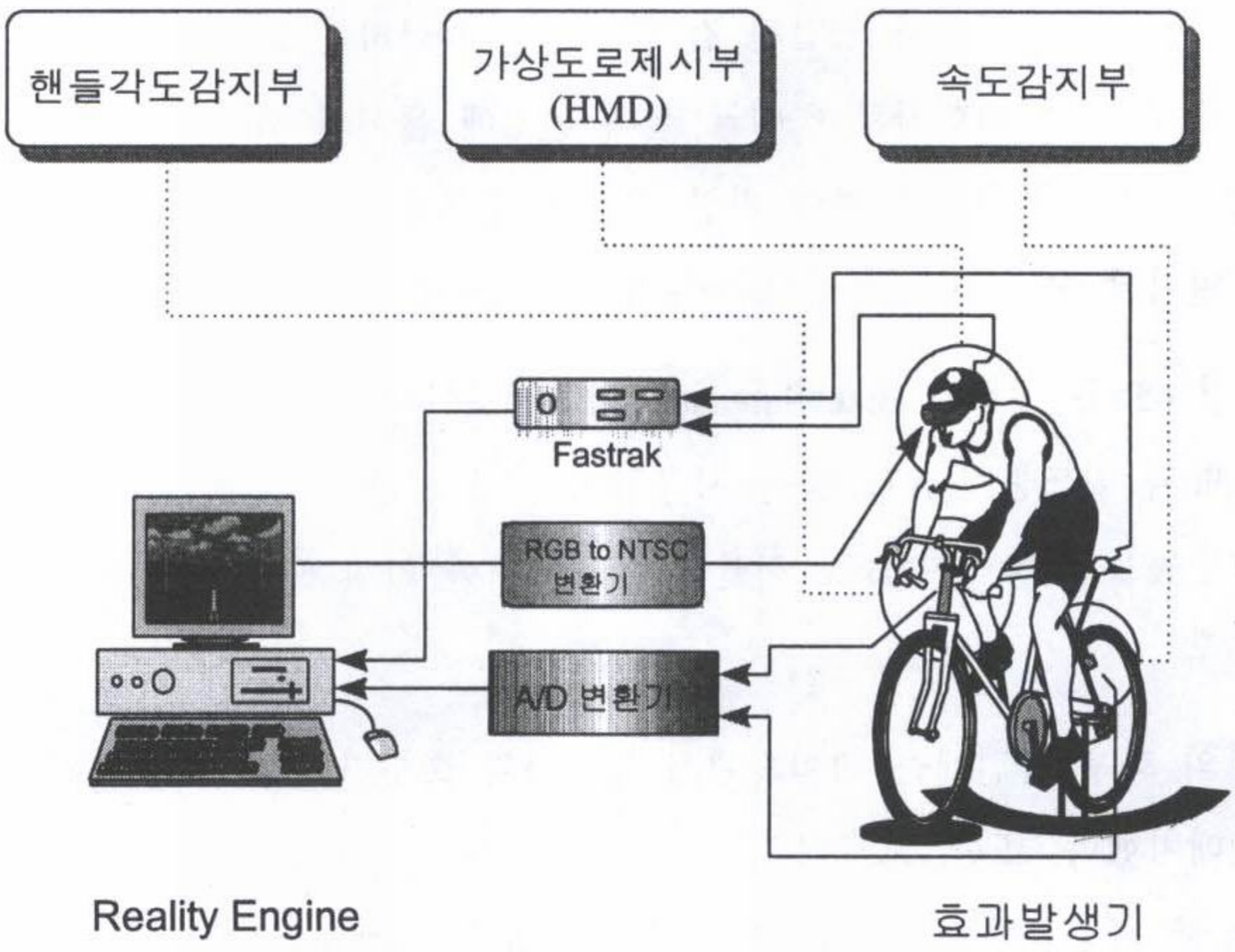
도로의 좌우 양편에는 피험자에게 몰입감과 속도감을 보다 효과적으로 제공해 줄 가로등을 배치했다. 또한, 도로의 중앙에는 노란색으로 도로의 중앙선을 표시하여 피험자가 가상 공간을 주행할 때, 따라가야 할 지표로써 사용하였다.

여기서 자전거의 핸들각과 속도는 A/D 컨버터를 통해서 컴퓨터로 입력된다.

또한 3차원 위치 측정기인 Fastrak의 입력을 통해 머리의 회전에 따라 배경 이미지를 변하도록 하여 피험자의 시선 방향에 맞는 이미지를 보여준다. 이렇게 해서 자전거를 타고 가다가 주위를 둘러볼 수 있도록 하여 보다 현실감 있는 가상 공간을 제작하였다.

자전거의 설계 및 제작은 피험자가 실제 자전거를 타는 것과 같은 효과를 제시하기 위해서 실제 자전거를 이용하고 자전거가 좌우로 움직일 수 있는 장치를 하였다.

피험자는 실제 자전거를 타는 것처럼 BIKE를 타고 주행하면 된다. 즉, 자전거의 페달을 굴러서 주행속도를 결정하고, 핸들 조작을 통해 주행 방향을 결정하면 되는 것이다. 이런 동작을 하기 위해 BIKE는 각도센서, 속도센서, 그리고 자전거의 좌우 기울임을 주기 위한 구조로 설계 및 제작을 하였다.



<그림 65> Bike Simulator의 구성도

제 5 절 가상환경 구축을 위한 행위명세 및 시뮬레이션에 관한 연구

1. 국내외 기술개발 현황

그래픽스와 가상 현실에서의 모델링은 크게 형태 모델링과 행위 모델링으로 나누어 연구되어 왔다. 형태 모델링은 객체의 모양이나 형태 정보의 표현과 처리를 다루며, 사용자가 보는 이미지들을 만들어내는데 이용된다. 다각형(Polygons), 곡선과 표면(Curves and Surfaces), Fractrals, Particle systems 등으로 각 객체의 형태를 모델링하고 그것들을 계층적으로 구성하는 방법이 많이 사용되고 있다. 행위 모델링은 객체의 움직임이나 행위를 기술한다. Keyframe을 이용하는 행위 모델링 방법은 운동학(Kinemattics)나 역학(Dynamics)을 이용하여 Keyframe들의 움직임을 계산하고 그 사이의 Frame들은 Interpolation을 통해 만들어 낸다. 운동학에서는 변환(Transformation)이나 중요 인자(Key Parameter)에 의해 움직임을 기술하고 역학에서는 힘(Force)이나 토크(Torque)나 다른 물리적 성질들을 고려한 미분 방정식을 계산하여 움직임을 기술한다. Reynolds의 ASAS[44, Perlin의 Improv[45]나 Thalman의 CINEMIRA 등은 스크립트(Script)를 이용하는 프로시듀럴 프로그래밍을 통하여 움직임을 기술한다. Gobetti의 VB2[46] 구조는 시스템의 상태와 행위를 서로 관계가

있는 객체들의 네트워크로서 표현하는데, 객체들의 여러 가지 관계를 계층적인 제한 조건(Hierarchical Constraint)에 의해 기술하고 시스템의 변화에 대해 제한 조건들을 만족시킨다. 하지만 이러한 시스템들은 객체가 가지는 행위나 기능들을 형태와 명확히 분리해 놓지 않았으며 행위 기술 방법이 낮은 단계에서 이루어지므로 때문에 그들이 복잡해 지면 다루기 어렵다.

그 반면, 소프트웨어 공학 분야에서도 시스템의 행위와 기능 모델링과 그것의 시뮬레이션을 통한, 가상 현실에서도 중요한 실시간 성질의 보장에 중점을 둔 연구를 해왔다. Harel은 그래피컬한 Computer-Aided Software Engineering(CASE) 도구인 STATEMATE[8]를 만들었는데, 이 도구는 대형의 복잡한 실시간 시스템에 대한 명세, 분석, 시뮬레이션, 설계, 다큐먼트 생성, 코드 생성 등을 한다. ASADAL은 실시간 시스템을 위한 그래피컬한 CASE 도구이다[47]. 이 도구는 쉽게 단계적으로 시스템에 접근할 수 있는 방법을 제공하고 있으며 특히 시뮬레이션 및 증명을 통해 시스템에 대한 유효성 검사, 검증 등을 한다. 하지만, 대부분 소프트웨어나 시스템의 내부에 대해 관심이 있을 뿐, 외부 객체들에 대한 모델링 방법에 대해서는 많은 연구가 되어 있지 않고 게다가 형태나 외부 모습까지 고려한 것은 거의 없는 실정이다.

2. 연구개발 수행 내용 및 결과

가. 연구내용

(1) 시스템 모델링

(가) 개념

형태로의 확장을 하지 않은 ASADAL에서 주요 시스템 모델링 철학은 단계적인(Incremental) 개발 방법의 철학에 기반하며, 그에 적합한 명세 방법과 시뮬레이션 도구를 가지고 있다. 우리는 형태 설계에도 이러한 개념을 확장하고자 한다. 단계적인 개발은 프로그램을 단계적으로 개발하고 유효성 검사할 수 있게 하며, Spiral, Evolutionary-prototypıng, Staged-delivery[49] 등의 많은 생명 주기 모델의 기반이 되고 있다.

객체들의 형태적 성질들(Physical Properties), 3차원 공간 내에서의 배치(Configuration), 물리적 법칙에 반응하는 행위(Reactive Behavior to Physical Laws)등 형태에 관련된 속성들은 행위, 기능과 동시에 단계적으로 명세된다. 형태 명세가 제어되는 객체

들의 모양과 물리적인 관계들 뿐 아니라 환경 객체들도 포함함에 주의해야 한다. 기본적인 생각은 각 차원(즉, 형태, 행위, 기능)의 명세들이 서로서로에게 영향을 준다는 것이다.

우리의 목표는 명세를 여러 상세화 단계에서 시뮬레이션하면서 제어되는 객체와 환경 객체들을 가시화함으로써 개발중인 시스템을 분석, 제어 시스템과 제어되는 시스템, 관련된 환경 객체들간의 상호 작용들을 관찰하는 것이다. 이렇게 하기 위해 제어 시스템 명세의 시뮬레이션은 외부 환경에 대한 시뮬레이션과 동시에 이루어져야 한다.

ASADAL/PROTO의 실시간 시스템에 대한 새로운 모델링 방법은 다음과 같은 세 가지 모델링 차원을 다룬다.

첫째, 행위 명세는 사건(Event)에 대응하는 시스템의 상태 변화를 보여주며, 상태에 따라 적절한 기능들을 활성화(Activation)시킨다.

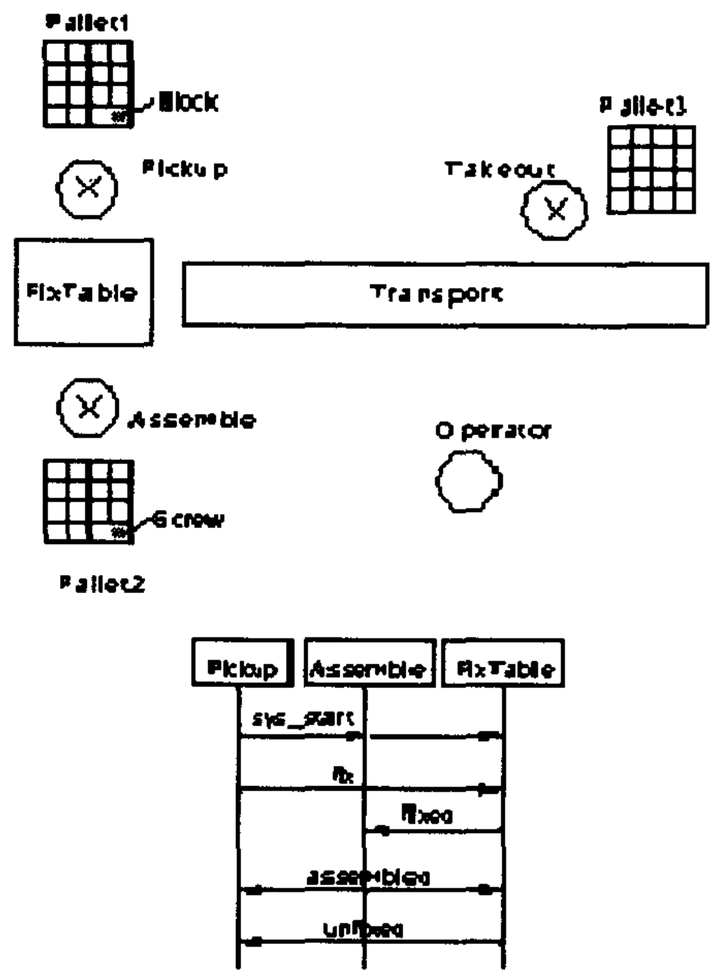
둘째, 기능 명세는 외부로부터의 입력에 대하여 데이터와 제어 신호의 계산 알고리즘들을 가진다.

셋째, 형태 명세는 객체들의 여러 공간적 성질들을 정의한다. 행위와 기능 명세가 모든 객체에게 주어지는 반면, 형태 명세는 물리적 성질을 갖는 제어되는 시스템(Controlled Device)이나 환경 객체(Environmental Object)에 주어진다.

행위와 기능 모델들은 각각 ASADAL[47]의 명세 방법에서 제공하는 TES(Time-Enriched Statechart)와 DFD(Data Flow Diagram)의 정형적 명세 방법을 사용하여 만들어진다. DFD는 프로세스들을 명세하는데, 이들을 여러 개의 부분 프로세스와 그들의 입력/출력관계로 분할함으로써 시스템의 기능을 밝혀낸다. DFD는 그것을 제어하는 TES를 가질 수 있는데, TES는 어떻게 프로세스가 행동하는지와 시스템의 상태에 따라서 그 프로세스를 언제 실행할지를 명세한다. DFD와 TES를 이용한 정형적 명세 방법의 자세한 설명은 [47]을 참조하라. 형태는 VOS를 사용하여 표현한다. VOS의 주요 목적은 물리적 객체의 형태와 공간상의 정적(Static) 및 동적(Dynamic) 배치를 기술하는 것이다.

VOS는 제어되는 시스템이나 실세계 환경 객체들 사이의 물리적 관계나 제약 조건들도 명세한다.

(나) 모델링 과정과 시뮬레이션에 의한 유효성 검사

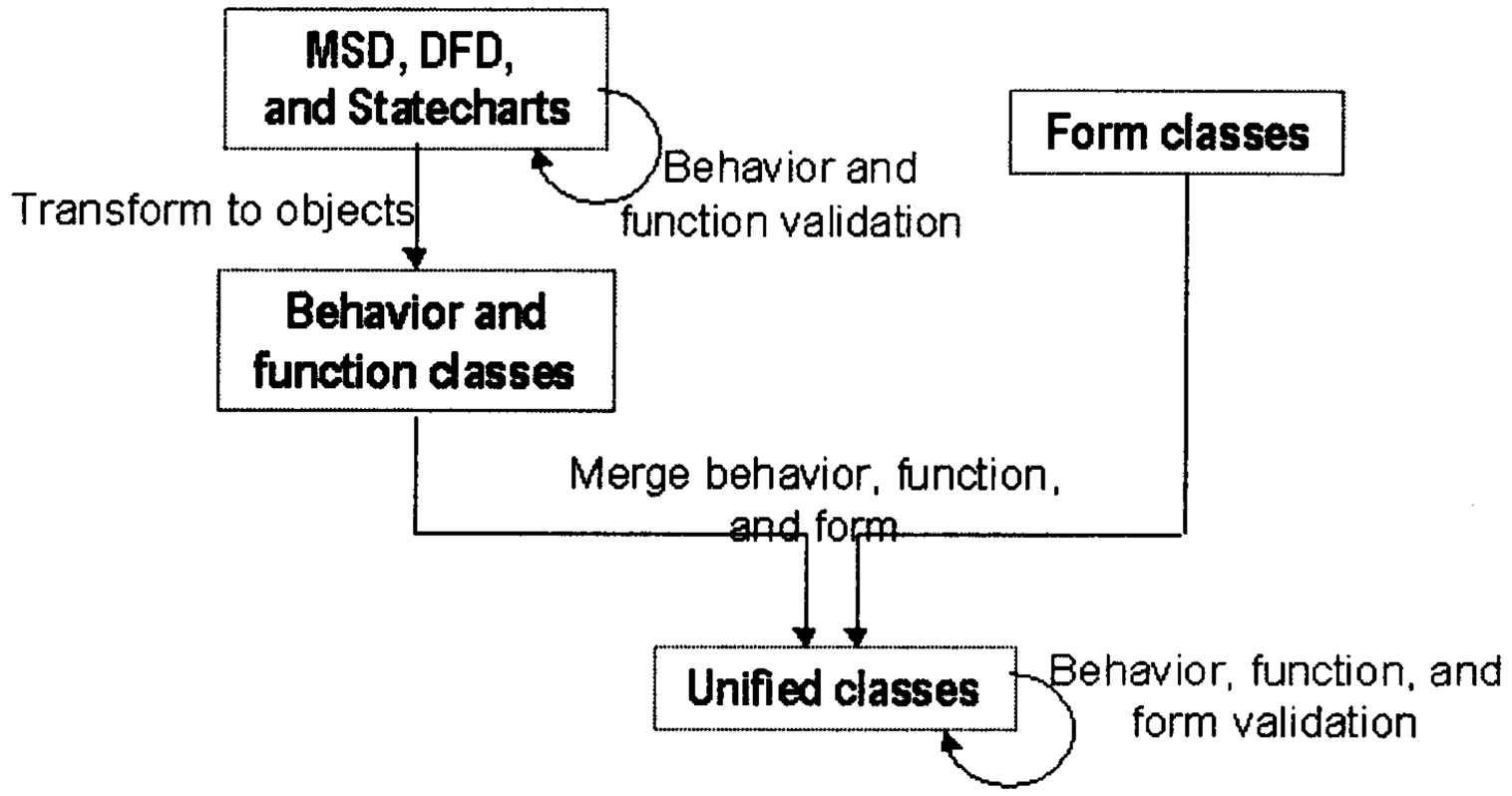


<그림 66> 한 개의 block과 두 개의 screw의 조립에 대한 생산 시스템의 설정

ASADAL에서 행위와 기능 명세들은 실제로 몇 장의 Message Sequence Diagram(MSD)을 그리는 것으로 시작된다. MSD는 시스템의 객체들 사이에 교환되는 데이타나 제어 신호들을 시간에 따라 나열함으로써 실시간 시스템의 외부 행동에 대한 전형적인 시나리오들을 기술한다(<그림 66> 참조). MSD는 처음부터 객체들을 찾을 수 있게 해 줌으로써 형태에 대한 고려와 함께 전체 모델링 과정에서 객체 지향성을 지원한다. 행위/기능 명세와 형태 설계가 처음에는 각각 서로에게 영향을 주고 받으며 진행되다가 어느 시점에서 검증의 목적으로 시뮬레이션을 효과적으로 수행하기 위해 객체 단위로 결합하므로 이들이 객체들에 대한 일치된 견해를 보이는 것이 중요하다.

<그림 67>는 단계적인 방식으로 시스템을 명세하고 유효성 검사를 하는 모델링 과정을 보여 준다. 제어 시스템과 실세계 환경을 포함한 전체 시스템의 명세는 그래피컬한 도구인 TES와 DFD를 가지고 만든다. 실세계에 존재하는 객체들에 대한 형태의 명세는 VOS를 사용하여 병렬적으로 수행한다. 객체 지향적 환경을 만들기 위해서 TES를 클래스들의 병렬적인 행위들과 그와 연결된 DFD의 기능들로 나눈다. 이렇게 만든 클래스들 중 로봇같은 것들은 형태를 가질 것이고 제어 소프트웨어와 같은 클래스들은 형태를 갖지 않을 것이다. 그래서 로봇과 같이 행위, 기능, 형태를 모두 가진 클래스들은 행위와 기능 클래스들과 그 형태를 기술한 VOS 클래스들로부터 다중(Multiple) 상속함으로써 만

들어진다. 실세계 객체들은 이러한 형태, 행위, 기능 클래스들로부터 설정 인자값(위치, 색깔)들을 새로 할당하고 객체화(Instantiation)함으로써 만들어지고, 형태가 없는 객체들은 행위와 기능 클래스들로부터 객체화된다.



<그림 67> 단계적인 명세와 유효성 검사 과정

두 개의 시뮬레이션 루프(Loop)가 수행되는데, 하나는 실시간 제어 시스템의 행위와 기능 명세의 시뮬레이션이고, 다른 하나는 형태를 가진 제어되는 시스템 또는 실제 환경에 존재하는 외부 객체들의 물리적 시뮬레이션이며 가시화와 동시에 이루어지며 상호 작용하는 시스템의 행위가 ASADAL[47]을 사용하여 관찰되고 분석될 수 있다. 이러한 명세와 시뮬레이션에 의한 유효성 검사 과정은 단계적인 방식으로 계속될 수 있다.

(2) 명세 방법

이 절에서는 간단한 로봇을 이용한 생산 시스템을 예제로 사용하여 명세 방법에 대해 설명한다. 우선, 중요한 시스템과 환경 객체를 인지하는데 이용되는 방법과 사용되는 요소들에 대해 설명하고, 그 후에 객체에 대한 상세화를 단계적인 방법으로 서로 다른 차원에서 수행하고 나중에 이들을 합치는데 그 과정에 이용되는 명세 방법을 설명한다.

(가) 시스템 명세

처음에 시스템이 수행해야 하는 "두 부품을 조립하라"라는 기능에 대한 요구 사항으로부터 시작하여, Block을 하나 가져와서 고정시키고 두 개의 Screw를 가져와서 조립

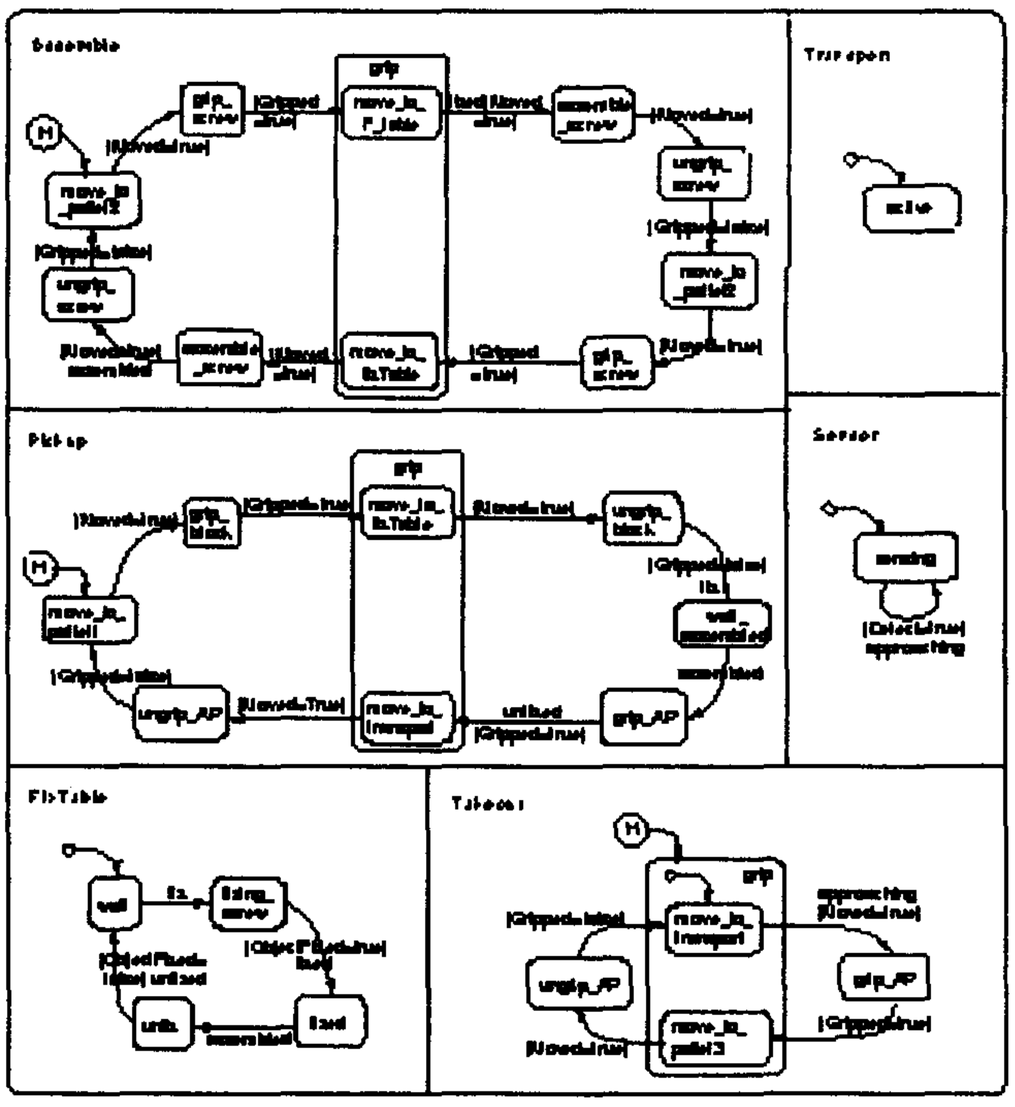
한 다음 그것을 저장할 곳으로 수송하고 내려 놓는 세분화된 기능들로 명세한다. 시스템이 그러한 기능을 수행하기 위해서 어떻게 행동해야 할지에 대하여 고려함으로써 <그림 67>과 같은 초기의 설정을 만든다.

그 과정에서 객체는 컴퓨터에 의해 제어되어 활동적으로 일을 수행하는 로봇과 같은 객체와 활동적이지 않은 Screw와 같은 환경 객체로 구분된다.

실시간 시스템의 외부 행위에 대한 전형적인 시나리오를 기술하기 위해 사용되는 ASADAL에서 소개된 MSD를 시스템의 목적을 완수하기 위해 필요한 시스템 객체와 환경 객체를 더 명확하게 하기 위해 사용한다. <그림 68>의 아래에 있는 MSD는 전형적인 시나리오를 가지고 객체 사이의 제어 신호의 흐름을 보여 준다. 예를 들어 <그림 68>의 MSD는 만약 Block을 *FixTable*에 옮긴 뒤에 *Pickup*이 *FixTable*에 *Fix*신호를 보내면, *FixTable*은 Block을 고정한 후에 *Fixed*신호로 반응한다는 것을 보인다.

시스템을 구성하는 객체들과 그들 사이에 흐르는 데이터 등으로 시스템의 설정을 명확히 한 후 각 객체에 대한 상세한 행위, 기능, 형태 명세를 시작할 수 있다. 그럼 다음에서 행위과 기능 명세에 대해 먼저 설명한다.

(나) 행위와 기능 명세



<그림 68> 생산 시스템 예제의 TES명세

이제 ASADAL의 TES와 DFD를 사용하여 MSD의 각 객체의 기능 구조와 행위를 상세히 명세한다. 먼저, 각 객체의 행위를 따로 명세하지 않고 전체 시스템의 행위를 객체들의 동시 다발적인(Concurrent) 상태들로 명세한다. <그림 68>의 TES는 예제 시스템의 행위를 보여준다. 예를 들면, <그림 68>의 윗부분에서 보이는 *Assemble*의 행위는 여러 가지 사건들과 환경 조건들에 의해서 다른 객체들과 동기화되어서 *Pallet*과 *FixTable*사이에서 움직이는 상태, Screw를 잡거나 놓는 상태, 조립 작업을 수행하는 상태 등을 가진다. 제어 시스템/제어되는 시스템과 환경 객체들 사이의 관계, 그리고 환경 객체들 사이의 관계는 물리적이므로 간접적인 환경 객체들의 행위는 TES를 사용하여 명세하지 않고, VOS에서 제약 조건으로서 표현될 것이다.

일단 행위 명세가 TES와 DFD를 사용하여 만들어지면, 명세의 논리적인 일치(Consistency)와 옳음(Correctness), 시간에 맞는 행위에 대한 분석을 할 수 있다. ASADAL의 Subsystem인 ASADAL/SIM은 DFD와 TES 명세들을 실행하고 데이타 흐름에 대한 통계학적 분석, 도달 가능성 분석, 비결정성에 대한 분석 등을 수행한다. 이러한 분석 결과는 형태 명세를 수행하기 전에 행위와 기능 명세가 가진 문제점들을 발견하고 바르게 고치는 데 사용될 수 있다.

나중에 행위와 기능 명세들을 VOS라는 객체 지향적인 형태 명세와 결합하기 위해 TES와 DFD도 <그림 69>의 왼쪽 윗부분에서 보이는 것처럼 객체 지향적으로 변환해야 한다. 변환된 객체는 TES에 명세된 그것의 행위를 가지고, 어떤 일이 실행되어야 할 때 DFD에 있던 적당한 함수들을 부르며, 상태 전이에 의해 사건이 발생할 때 채널을 통해 신호를 보낸다.

형태가 없는 객체들은 행위와 기능 명세만으로 충분할 것이나 어떤 객체들은 행위와 기능뿐만 아니라 형태를 가진다. 이러한 객체들에 대해 형태는 VOS를 사용하여 명세하며 VOS는 객체들의 물리적인 특성들과 공간적인 행위들을 보인다.

(다) VOS

형태 명세는 형태를 가진 객체들의 공간적인 성질과 설정 정보를 기술하며, 모양, 표면정보(Material), 위치, 움직임 등의 성질(Attribute)들을 포함한다.

형태 명세는 행위와 기능 모델링과는 독립적으로 할 수 있는데, 점진적인 방식으로 상세화해 나간다. 예를 들면, 처음에는 시스템 요구 사항을 분석하고 난 후에 떠오르는 초기의 대략의 모양과 부피, 설정 정보들을 생각함으로써 시작할 수 있다. 이후의 명

세는 행위와 기능 명세가 상세해짐에 따라 점진적으로 상세화되며 그에 따라 행태 명세의 기본적인행위들로 표현되었던 행위나 기능들을 행위나 기능 명세에서 재정의하여 사용할 수도 있다. 기본 행위와 기능은 사용자가 제어에 신경쓸 필요없는 하위 단계의 행위와 기능을 말한다.

VOS는 크게 형태 모델링, 장면 그래프 구성, 규칙과 제약 조건의 기술에 의해 만들어진다. 형태 모델링은 형태를 가진 객체들의 형태를 만들고 3차원 가상 공간 상에 보여 준다. 객체는 여러 하위 객체(Subobject)들로 구성되며 그것의 구조를 장면 그래프(Scene Graph)로 나타낸다. 여러 그래픽 라이브러리에서 많이 사용되고 있는 장면 그래프(Scene Graph)와 비슷한 구조를 사용하며 데이터 구조는 트리(Tree)이다. 트리상의 자식 노드(Child Node)들의 좌표 시스템은 부모 노드의 좌표 시스템에 상대적이며 부모 노드에 어떤 움직임이 적용되면 그것의 모든 자식 노드들에게도 똑같은 움직임이 적용된다. 마지막으로 명세 언어는 객체들 사이의 관계나 규칙, 제약 사항들을 기술한다.

VOS는 객체 지향적인 언어이다. 예를 들어, Manipulator의 VOS는 Joint와 Link의 VOS를 조합함으로써 만들어진다.

① 규칙과 제약 조건 기술

가상 객체 명세 언어는 가상 세계 객체들 사이의 관계에 관한 규칙들이나 제약 조건들을 기술하기 위한 언어이다. 이러한 언어를 제공함으로써 물리학에 기반한 시뮬레이션(Physically-based Simulation)을 하지 않고 상위 단계의 추상화를 통해 사용자가 생각하기 쉽고 실제와 비슷한 시뮬레이션을 수행할 수 있다.

이 명세 언어에서 객체는 그것을 이루는 점들의 집합으로 정의하며, 객체들이 움직일 수 있는가를 기준으로 정적 객체(Static Object)와 동적 객체(Dynamic Object)로 나누었고, 객체를 움직일 수 있는 원동력이 있는가를 기준으로 능동적 객체(Active Object)와 수동적 객체(Passive Object)로 나누었다. 동적 객체는 움직임을 명세하기 위해 속도, 가속도, 각속도, 각가속도 등을 성질로 가지고, 능동적 객체는 수동적 객체들을 수동적인 자식으로 등록하여 제어한다. 객체들 사이의 관계들도 그러한 관계에 있는 객체들의 순서쌍으로 이루어지는 집합이며, 장면 그래프상의 부모와 자식 관계(ParentChild)와 능동적 객체와 그것에 의해 원동력을 가지는 수동적인 객체 사이의 관계(Active-parentPassivechild)등이 기본적으로 제공되는 집합이다. 그리고, 점과 선, 면, 입체에 대한

정의 및 모든 객체의 각 축으로의 최대값(Xmax, Ymax, Zmax)과 최소값(Xmin, Ymin, Xmin)와 교집합, 합집합, 차집합, 여집합 등의 집합 연산자를 명세에 사용할 수 있다.

기본적으로 제공되는 집합들의 의미는 다음과 같다.

$A \in Point$

객체 A는 점이다.

$A \in Line$

객체 A는 선이다.

$A \in Face$

객체 A는 면이다.

$A \in Volume$

객체 A는 입체이다.

$(A, B) \in ParentChild$

트리 구조의 장면 그래프에서 A는 부모이고 B는 자식이다. 이 관계에 있을 때 부모의 움직임이 그대로 자식에게 적용된다..

$(A, B) \in AcitveparentPassivechild$

A가 능동적 객체인 부모이고 B가 수동적 객체인 자식이다. 이 관계에서는 부모가 움직이지 않아도 부모의 원동력에 의해 수동적인 자식들은 움직일 수 있다.

$(A, B) \in Attach$

A와 B는 서로 붙어 있는 관계이다. 두 객체 중 어느 한 쪽에게 움직임이 적용되면 다른 한 쪽에게 그대로 적용된다.

$(A, B) \in AncestorDescendant$

트리 구조의 장면 그래프에서 A는 조상이고 B는 후손인 관계로 이것에 관한 추론 법칙(Inference Rule)은 다음과 같이 정의될 수 있다.

$(A, B) \in AncestorDescendant \equiv (A, B) \in ParentChild$

$$(A, B) \in AncestorDescendant \equiv (A, C) \in ParentChild, (C, B) \in AncestorDescendant$$

② 사실

장면 그래프로부터 ParentChild관계에 있는 모든 사실들이 자동으로 생성된다. 예를 들어, 로봇(Manipulator)의 경우 그것의 장면 그래프로부터 다음과 같은 사실들이 생성된다.

$(r1 : j1, r1 : l1) \in ParentChild$

$(r1 : l1, r1 : j2) \in ParentChild$

$(r1 : j2, r1 : l2) \in ParentChild$

$(r1 : j2, r1 : j3) \in ParentChild$

$(r1 : j3, r1 : l3) \in ParentChild$

$(r1 : l3, r1 : g) \in ParentChild$

$(r1 : g, r1 : g : frj) \in ParentChild$

$(r1 : g, r1 : g : flj) \in ParentChild$

$(r1 : g : frj, r1 : g : fr) \in ParentChild$

$(r1 : g : flj, r1 : g : fl) \in ParentChild$

③ 규칙들(Rules)

규칙들은 도메인에 관계없이 사용될 수 있는 메타 규칙과 어떤 도메인에서만 사용되는 도메인 규칙과 사용자의 애플리케이션에서 필요한 규칙들로 세분화될 수 있다. 그러나, 사용자가 시스템에서 제공하는 메타 규칙과 도메인 규칙을 더 상세히 기술하거나 다른 의미로 사용하고자 하는 경우에는 규칙들을 재정의하여 사용할 수 있다. 사용자가 같은 이름을 가진 다른 의미의 규칙을 정의하면 그것이 더 높은 우선 순위를 가지게 된다. 아래의 규칙들은 우리가 다루고 있는 예제에서 사용되는 규칙들이다.

메타 규칙들(Meta Rules)

$$(A, B) \in Interfere \equiv (A \in SpatialObject) \wedge (B \in SpatialObject) \wedge (A \cap B \neq \phi)$$

두 객체 A와 B 사이의 부딪힘(Interference)을 검사하는 것은 물리적 시뮬레이션을 위해서 아주 유용하며, 두 객체의 교집합이 있을 때로 정의한다.

$$(A, B) \in \text{Contact} \equiv (A \in \text{SpatialObject}) \wedge (B \in \text{SpatialObject}) \wedge ((A, B) \in \text{Interfere}) \wedge (A \cap B \in \text{Volume})$$

두 객체 A와 B가 부딪혔지만 부딪혀서 생긴 교집합이 입체가 아닌 점이나 선, 또는 면인 경우를 닿았다(Contact)고 정의한다.

$$(A, B) \in \text{In} \equiv (A \in \text{SpatialObject}) \wedge (B \in \text{SpatialObject}) \wedge (A \cap B = A)$$

객체 A가 B 안에 있다는 것은 A와 B의 교집합이 A가 된다는 것으로 정의한다.

$$(A, B) \in \text{On} \equiv (A \in \text{SpatialObject}) \wedge (B \in \text{SpatialObject}) \wedge ((A, B) \in \text{Contact}) \wedge (A.\text{Zmin} = B.\text{Zmin})$$

객체 A가 객체 B 위에 있다는 것은 두 물체가 닿아있고 A의 Z축으로의 최소값이 B의 Z축으로의 최대값과 같을 때로 정의한다.

$$(A \in \text{SpatialObject}) \wedge (B \in \text{DynamicObject}) \wedge ((A, B) \notin \text{AncestorDescendant}) \rightarrow (B.\text{accel} = (0, 0, -9.8))$$

동적 객체 B가 정적 객체 A의 후손이 아니라면 B는 중력의 영향을 받게 된다.

도메인 규칙들(Domain Rules)

$$(A, B) \in \text{Gripped} \equiv (A \in \text{SpatialObject}) \wedge (B \in \text{Gripper}) \wedge ((A, B.\text{LeftFinger}) \in \text{Contact}) \wedge ((A, B.\text{RightFinger}) \in \text{Contact})$$

그리퍼(Gripper)의 왼쪽 집게(B.LeftFinger)와 오른쪽 집게(B.RightFinger)가 객체 A에 닿으면 객체 A가 그리퍼 B에 의해 잡힌 상태이다.

$$(A, B) \in \text{Fixed} \equiv (A \in \text{SpatialObject}) \wedge (B \in \text{Gripper}) \wedge ((A, B) \in \text{On}) \wedge ((A, B.\text{LeftFixBar}) \in \text{Contact}) \wedge ((A, B.\text{RightFixBar}) \in \text{Contact})$$

고정 테이블(FixTable)B의 고정을 위한 왼쪽 막대 (B.LeftFixBar)와 오른쪽 막대 (B.RightFixBar)가 A와 닿으면 객체 A가 고정 테이블 B 위에 고정된 상태이다.

$$(A, B) \in \text{Assembled} \equiv (A \in \text{Screw}) \wedge (B \in \text{Block}) \wedge ((A.\text{tail}, B.\text{hole}) \in \text{In})$$

나사 A의 꼬리 부분이 블록 B의 구멍 안에 있으면 두 객체는 조립된 상태이다.

$$(A \in DynamicObject) \wedge (B \in ConveyorBelt) \wedge ((A, B) \in On) \rightarrow ((B, A) \in ParentChild) \wedge ((B, A) \in ActiveparentPassivechild)$$

컨베이어 벨트 B 위에 동적 객체 A가 있으면 B는 A의 부모인 동시에 동적 부모가 된다. 그러므로, 컨베이어 벨트가 움직이면 동적 객체가 움직일 뿐 아니라 컨베이어 벨트의 원동력에 의해 동적 객체만 움직일 수도 있다.

$$(A \in DynamicObject) \wedge (B \in Gripper) \wedge ((A, B) \in Gripped) \text{ c } ((B, A) \in ParentChild) \wedge ((B, A) \in ActiveparentPassivechild)$$

그리퍼 B에 의해 동적 객체 A가 잡히면 B는 A의 부모인 동시에 동적 부모가 된다.

애플리케이션 규칙들(Application Rules)

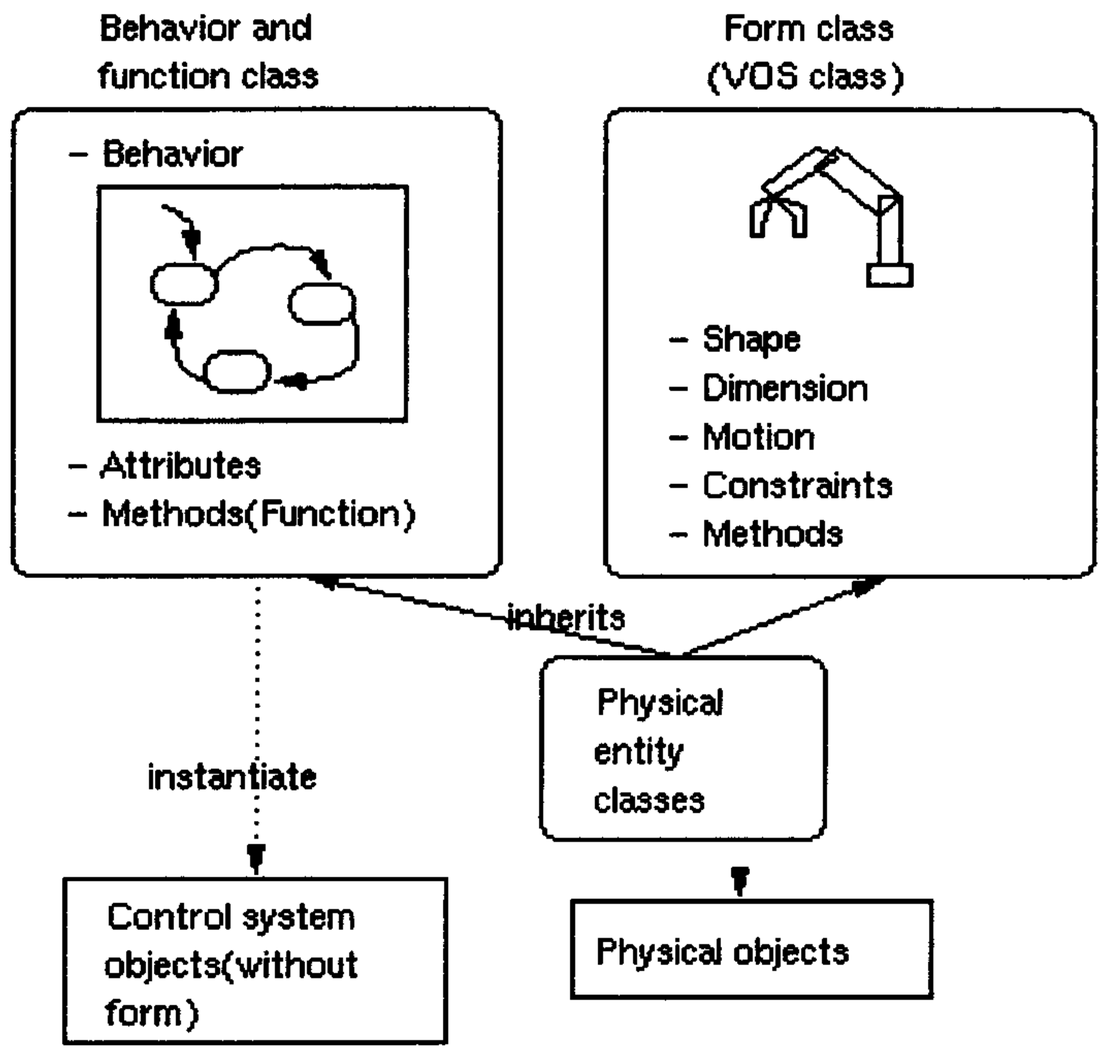
$$(A \in Screw) \wedge (B \in Block) \wedge ((A, B) \in Assembled) \in ((A, B) \in Attach)$$

나사 A와 블록 B가 조립되면 두 개는 서로 붙어 있다. 그러므로, 그리퍼가 나사를 잡고 움직여도 블록이 따라 움직이고 블록을 잡고 움직여도 나사가 따라 움직이게 된다.

④ 제약 조건들(Constraints)

위에서 설명한 규칙들은 시뮬레이션을 할 때 두 객체 사이의 관계를 집합을 사용하여 정의하고 애플리케이션 규칙을 만족시킴으로써 물리적 시뮬레이션을 대신하기 위한 것이다. 반면에 제약 조건들은 만들고자 하는 시스템이 지켜야 할 규칙들이다. 예를 들어, $(A \in Manipulator) \wedge (B \in ConveyorBelt) \rightarrow ((A, B) \notin Interfere)$가 나타내는 것은 로봇과 컨베이어 벨트는 서로 부딪혀서는 안 된다는 것을 나타낸다.

(라) 행위, 기능, 형태를 가지는 통합 클래스



<그림 69> 행위, 기능, 형태를 명세하는 객체 모델

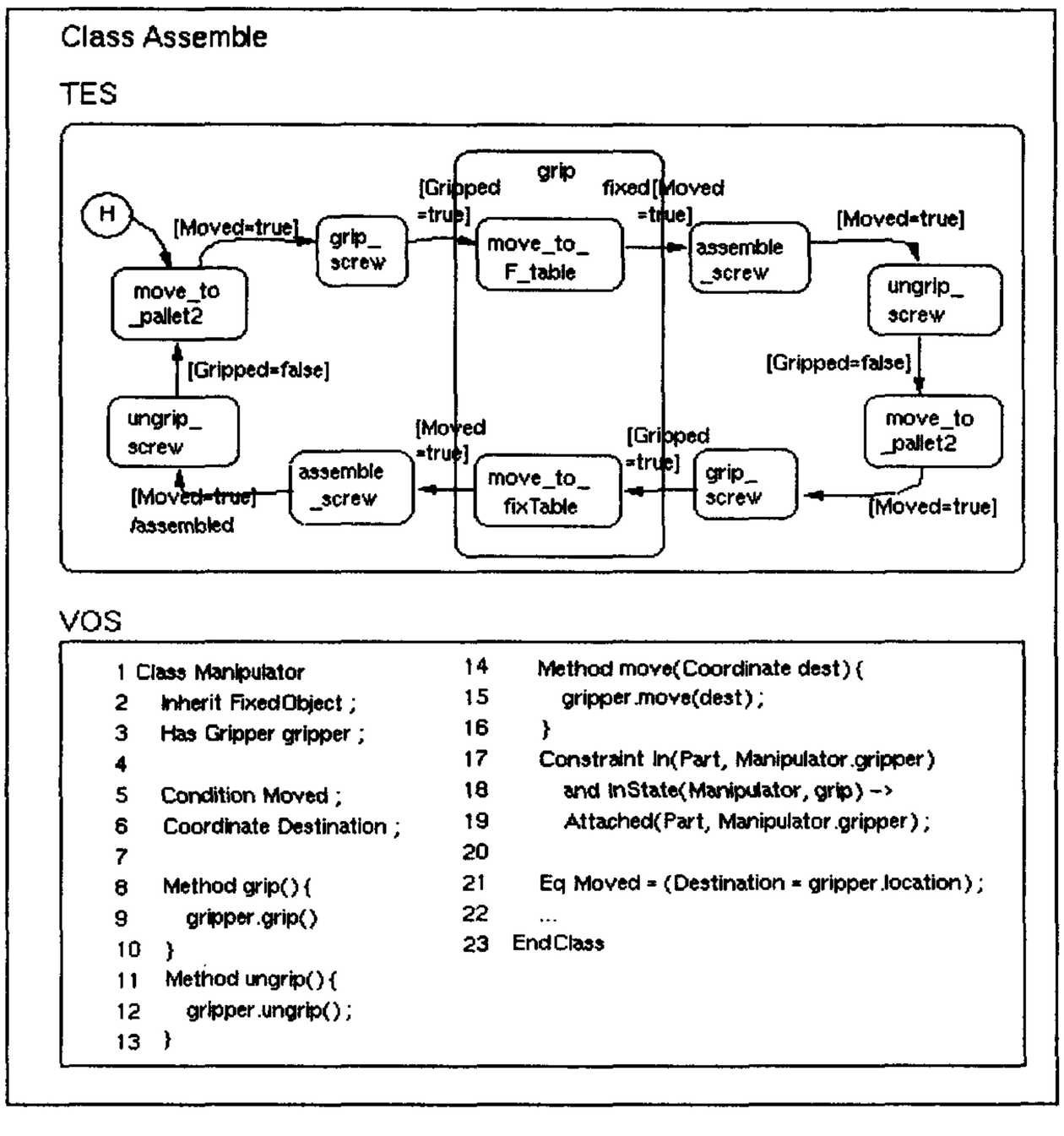
행위/기능 명세와 VOS를 만드는 것이 동시에 진행될 수 있지만, 가시화 시뮬레이션은 제어되는 시스템과 환경 객체들이 세 가지 차원의 명세를 모두 가지기를 요구한다. 그러므로, 제어되는 시스템과 환경 객체들에 대해 <그림 69>처럼 완전한 실시간 객체 표현을 만들기 위해 적절히 행위/기능 클래스와 VOS를 결합해야 한다. 우리는 이러한 통합 객체들을 행위/기능 클래스와 VOS 클래스를 다중 상속함으로써 만들 수 있다. 그러면 물리적 객체들은 새로운 통합 클래스들을 객체화함으로써 만들어지고, 제어 시스템처럼 형태가 없는 객체들은 행위/기능 클래스들로부터 객체화된다.

이런 방식으로 같은 행위를 갖지만 다른 물리적 성질을 갖거나 그 반대 경우의 클래스들을 쉽게 만들 수 있다. 예를 들어 *Assemble, Pickup, Takeout* 클래스들은 모두 팔과 End-Effector를 가지는 *Manipulator* VOS 클래스를 상속하지만 <그림 69>의 행위, 기능 명세에서 만들어진 클래스들로부터 서로 다른 행위와 기능들을 상속한다. *Manipulator* 클래스는 물리적 세계의 Manipulator의 움직임을 보이는 *move* 메쏘드를 갖는다. 객체들이 클래스의 객체화에 의해 만들어지며, 필요하다면 여러 개의 *Takeout* 클래스 객체를 만들 수 있다.

<그림 69>은 행위, 기능, 형태를 가진 *Assemble*클래스를 보여 준다. 이 클래스의 행위와 기능은 <그림 68>의 TES로부터 변환된 행위, 기능 객체 클래스로부터 상속하고,

그것의 형태 명세는 이미 정의된 *Manipulator* 클래스로부터 상속한다. *Assemble*은 VOS에 정의된 *move(Pallet2)* 메쏘드를 호출함으로써 *Pallet2*로 움직이고, *grip()*을 이용하여 Screw를 잡는다. 움직여진 Screw의 상태는 VOS에 정의된 *Moved*의 조건 변수로 나타난다. 여기서 *In, On, Above* 등은 도구에서 제공하는 공간적 관계 조건들이다.

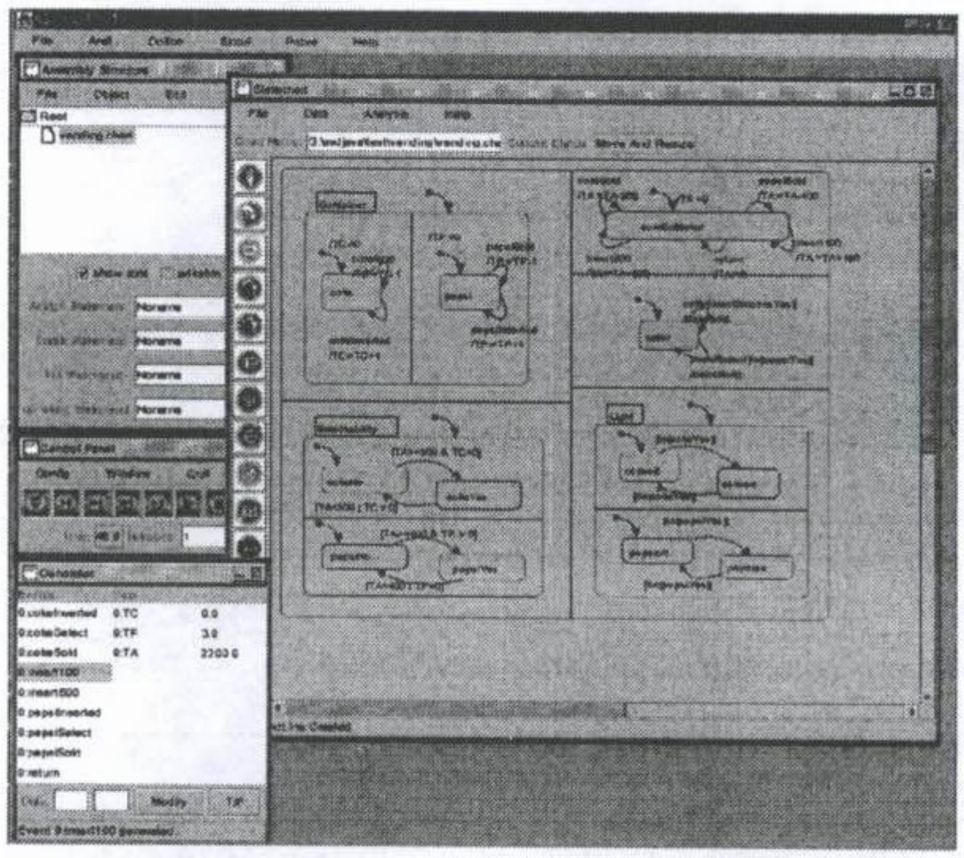
시스템과 그것의 환경에 있는 모든 클래스들이 명세되면 시스템 모델을 만들어야 한다. 객체들은 객체화되면서 초기 설정과 같은 값들이 주어진다. 예를 들면, <그림 70>에 있는 물리적 객체들은 위치, 방향 등에 대한 적절한 설정 인자 값들을 가진 *Assemble, Transport, FixTable*와 같은 통합 클래스로부터 객체화된다. 형태와 환경 객체 행위까지 고려함으로써 전체 시스템 명세의 가시화 시뮬레이션은 훨씬 더 현실적이 되고 개발 초기 단계부터 올바른 시스템 명세의 개발을 돕는다.



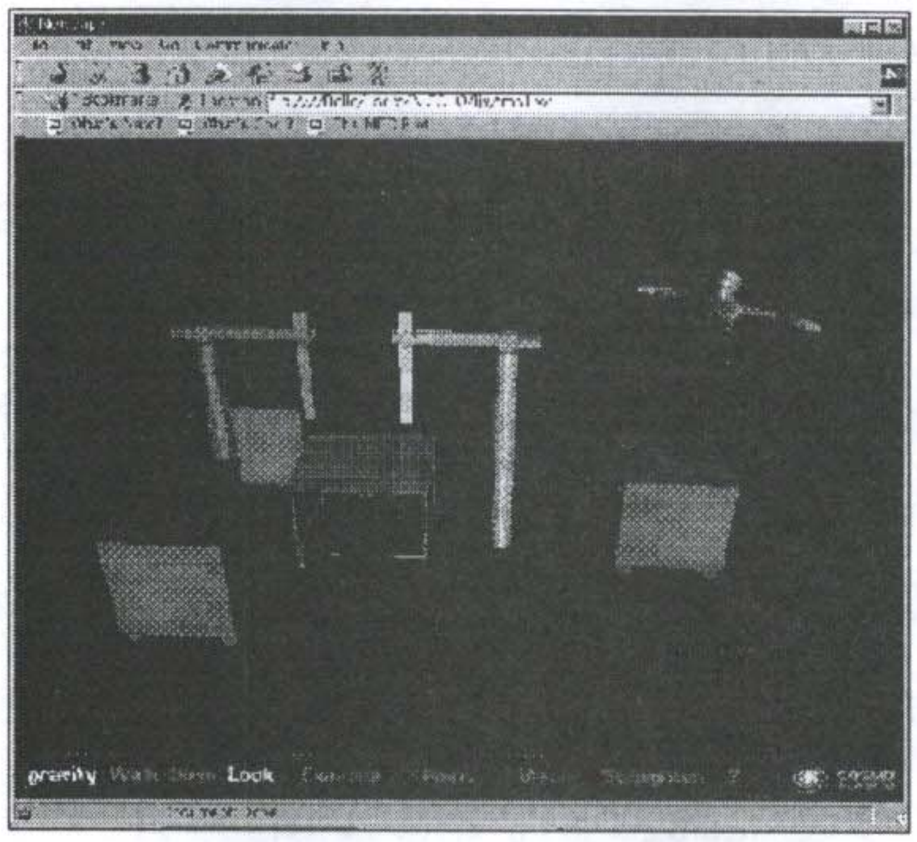
<그림 70> 물리적 클래스 Assemble

나. 연구결과

행위와 기능 명세의 시뮬레이션 환경은 완성되었다. 전체 시스템을 ASADAL[47]이라고 부르고 "http://selab.postech.ac.kr/~realtime"을 통하여 얻을 수 있다. <그림 71>은 ASADAL/SIM의 시뮬레이션 엔진과 분석 도구를 보여 준다. ASADAL/PROTO는 ASADAL/SIM에 형태 표현과 물리적 시뮬레이션을 위한 명세 방법과 도구를 더해 확장한 것이다. 전체 구현이 Java3D에 기반하고 있지만 그것의 초기 프로토타입은 VRML(Virtual Reality Markup Language)과 자바 기술[50]에 기반하여 구현되었다. 생산 공장 예제의 가시화 시뮬레이션에 대한 화면이 <그림 72>에 있다. 한 Manipulator가 두 개의 다른 Pallet으로부터 부품들을 가져와서 조립하고 다른 한 Manipulator가 완성된 물건을 세번째 Pallet에 옮겨 놓는다. 사람이 시뮬레이션의 행위에 영향을 줄 수 있는데, 한 가지 예로 사람이 테이블에 부딪히면 Pickup Manipulator는 조립을 실패하게 된다.



<그림 71> ASADAL 행위와 기능 시뮬레이터의 sanpshot



<그림 72> ASADAL/PROTO 가시화를 위한 프로토타입 환경

VR 모델은 시스템 개발 전에 시스템의 작동 모습을 미리 보면서 유효성을 검증하는 것, 비행 시뮬레이터 같이 시스템과 함께 개발되어 사용자의 훈련을 하는 것 등을 위하여 만들어진다. 따라서 많은 경우 VR 모델은 행위, 기능의 시뮬레이션과 밀접한 관련하에 연동되어야 할 필요성이 있다. 이러한 경우 VR 모델과 행위, 기능 모델은 같이 만들어져야 하며 이들 모두 처음에는 목적과 밀접하고 중요한 것에서부터 상세화해 나가면서 단계적으로 모델링을 해 나가는 방법을 제공해야 한다. "형태는 기능을 따른다"는 말이 있는데, 이것은 시스템의 기하학적, 재료적 성질들이 의미없게 있는 것이 아니고 어떤 효용 가치를 가진다는 것을 말한다[51]. 이러한 형태, 행위, 기능의 유기적인 연결과 이들의 모델링 방법은 그래픽스/VR에서도, 소프트웨어 공학에서도 거의 연구된 바가 없기 때문에 우리는 이 연구에서 이들을 통합하고 시뮬레이션과 가시화를 하는 것이 어떤 식으로 이루어질 수 있으며 어떤 의미를 지니는 것인지를 보여주었다.

ASADAL의 단계적이고 계층적인 모델링과 시뮬레이션 및 가시화 도구들은 객체지향적인 VR 시스템을 체계적으로 빠르게, 쉽게 만들고 모델의 변화나 시스템 구성의 변화에 쉽게 대처할 수 있게 해 준다.

제 6 절 VR System의 인간공학적 설계기술 개발

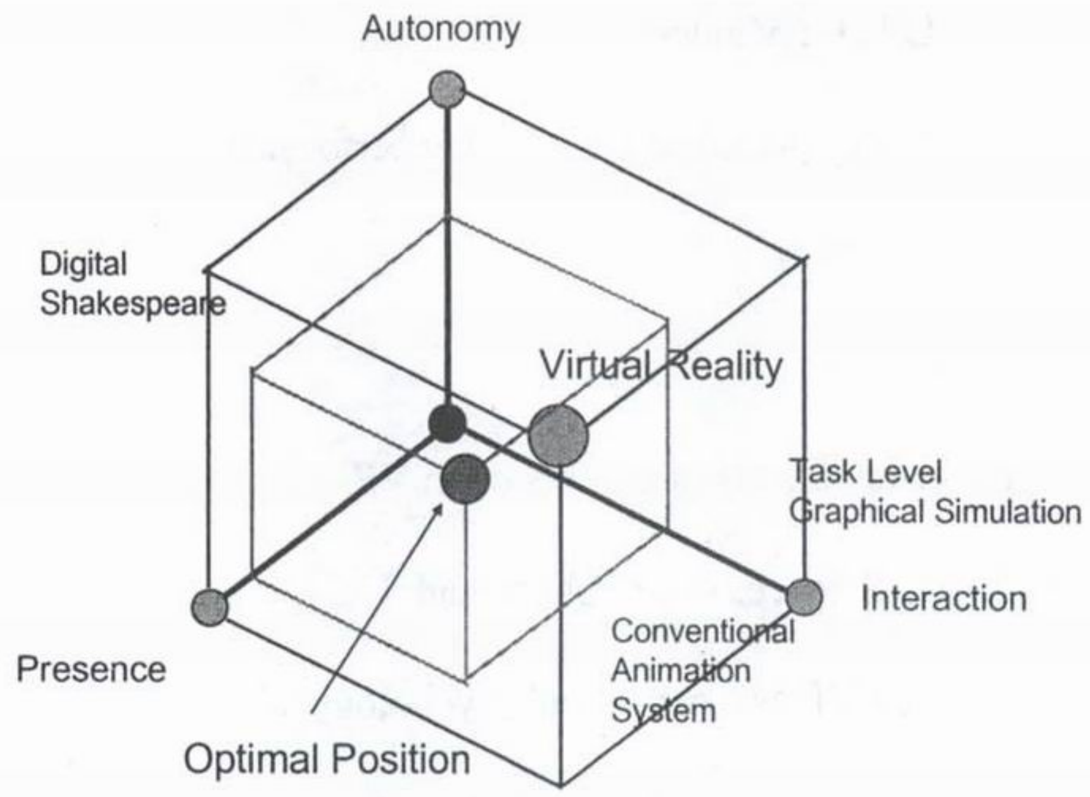
1. 국내외 기술개발 현황

VR분야의 시장은 현재 정형화 되어 있지 못하다. 현재는 VR기술의 진전에 따른 여러가지 가능성만이 무성하게 탐색될 뿐, 상업적으로 성공을 거둔 사례는 그리 많지 못한 형편이며, 군사용 훈련시뮬레이터나 비행 시뮬레이터분야등 비상업적인부분에서는 상당한 진전을 보이고 있다. 다만 VR기술이 인간생활 전반에 걸친 다양한 응용분야를 갖고 있으며 그 파급효과가 상당할 것으로 예상되기에, 세계 각국의 투자가나 과학기술자, 정부등이 미래의 VR시장에 대한 주목을 갖고 있는 것이다 [52].

인공현실감 기술은 Zeltzer가 분류한 바와 같이 가상세계에서 얼마나 임장감(Presence)을 느낄 수 있는가 하는 presence요소, 이러한 개체들을 얼마나 자유롭게 다룰 수 있는가하는 interation요소, 가상세계에 존재하는 객체들의 자율성을 부여하는 autonomy요소를 만족시킴으로써 완전한 가상환경을 만들수 있다.

<그림 73>은 Zeltzer의 AIP cube을 나타내고 있다[53]. 현재의 기술수준에서는 완전한 가상현실을 만들기위한 요소들인 presence, interaction, autonomy을 구현하기는 불가능하다. 그렇기 때문에 가상현실 시뮬레이터의 사용용도에 따라 가장중요한 요소가 무엇인지를 파악하여 요소들의 수준을 조정함으로써 최적의 인공현실감 시뮬레이터를 구현할 수 있다.

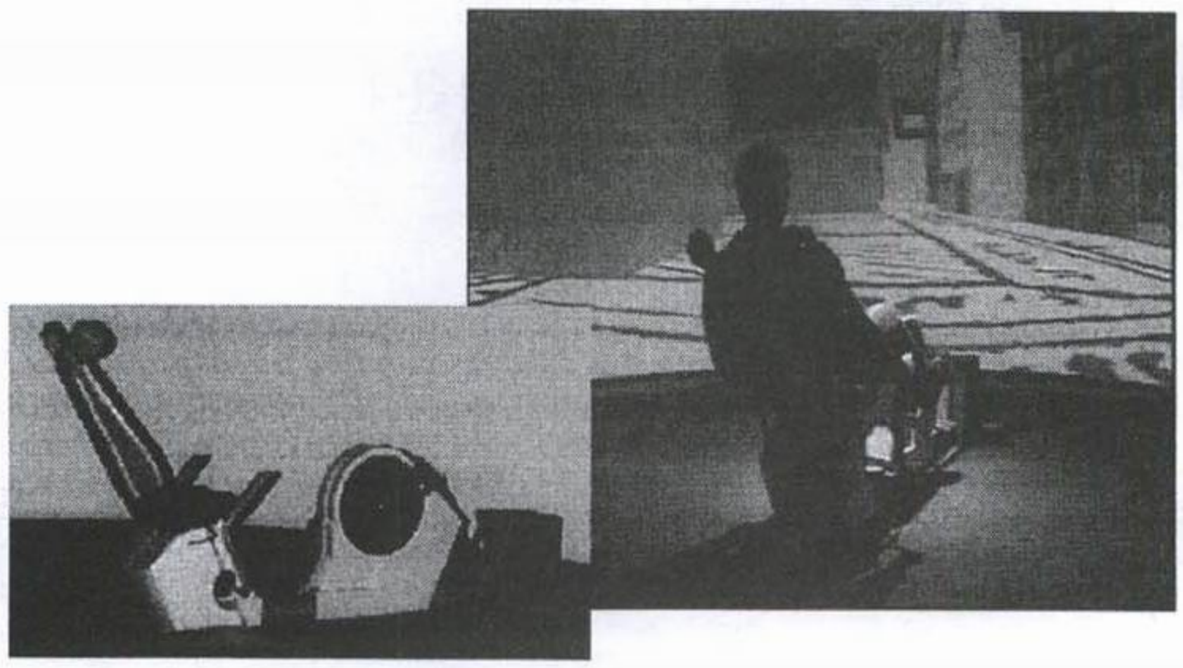
예를들어, wayfinding과 같이 목적지를 찾는 작업은 제시되는 환경의 reality이 가장 중요한 요소가 되고, 전자제품을 작동시키는 작업은 가상환경과 사용자간의 interaction부분과 물체의 autonomy가 중요한 요소가 된다. 또한 제품에 대한 인간이 느끼는 감성을 파악하는데 있어서 외형을 현실과 똑같이 표현하기 위한 높은 해상가 필요한다.



<그림 73> Zeltzer의 AIP cube

Max-Planck-Institut에서는 object recognition, depth perception, wayfinding, driving, grasping에 관한 biological study을 수행하고 있다. 이러한 과정에서 현실감을 높이기 위한 실험으로써 Speed perception, Orientation behaviour, View-based navigation, Driving behaviour와 같은 4가지에 대한 실험을 하고 있다.

<그림 74>는 위의 실험을 수행하기 위하여 구축한 가상환경 시스템을 보여주고 있다.



<그림 74> Max-Planck-Institut에서 구축한 가상환경 시스템

기타 연구기관에서는 다음과 같은 내용을 수행하고 있다.

- The perception of spatial layout in real and virtual worlds
- Perceived spatial layout of a simulated scene as a function of experience

· HF Research Lab. Of Minnesota Univ.

- Effect of visual display parameters on driving performance in a VE driving simulator

  · Northeastern Univ.

- Studies into the visual effects of immersion in VE

  · Dept. of Human Sciences of Loughborough Univ.

- A hill study : using a VE as a perceptual psychology laboratory

  · Computer science Dept. and the psychology Dept. of virginia univ.

이러한 일련의 연구들은 모두 최근에 이루어지고 있으며, 연구수준이 초기단계에 머무르고 있다.

2. 연구개발수행 내용 및 결과

본 장에서는 가상환경 시스템설계에 있어서 인간공학적인 요소를 살펴보고, 본 연구의 목적인 3차원 시청각제시기의 현실감을 평가하기 위한 여러가지 평가방법론에 대해 고찰하였다. 또한 현실감평가 실험과 이에 대한 분석을 통하여 가상환경제시기의 임장감을 향상시키기 위한 여러가지 요인을 파악하였다.

가. VR system의 인간공학적 요소

가상환경에서의 인간공학적 논점은 다음과 같이 크게 7가지 정도로 적용분야를 정할 수 있다 [54].

첫째, Multiple sensory chanels. 인간이 현실에서 느끼는 오감과 얼마나 비슷하게 인공적으로 5개의 감각에 feedback을 줄 수 있는가에 대한 연구분야로써, 작업특성에 따라 인간이 주로 의존하는 감각을 규명하는데 목적이 있다.

둘째, participant representation. 요즘 가상환경 시스템에서 논점중에 하나가 참여자 자신의 신체를 어떻게 가상환경에 표현하는가이다. 이때 인간공학은 가상의 신체를 디자인하는데 도움을 줄 수 있다.

셋째, design and fit of HMDs. 입체시(stereopsis)는 인간이 자신의 수평적으로 분리된 두눈으로 인하여, 망막상에 시차가 일어나고, 이 시차로 인한 두눈의 상을 대뇌에서 하나로 합성하여 대상물체의 입체를 인지하는 것을 말한다. 이러한 입체시를 재생하기 위해서는 현재로써 HMD가 가장 타당하다. 하지만, 사람마다 신체의 다양성 때문에 두눈 사이의 거리가 틀리다. 이를 극복하기 위해서 인체치수를 효과적으로 디자인에 적용하기 위한부분이 상당히 중요하다.

넷째, navigation and orientation. 가상환경의 큰 장점중의 하나는 장소에 구애받지 않고 여러가지 환경과 상호작용할 수 있다는 것이다. 이러한 상호작용에 있어서 얼마만큼 인간이 몰입하고 사용하기 편한가를 평가할 수 있다. 이에 대한 정량적인 평가부분이 중요하다.

다섯째, Presence and involvement. 인간이 가상환경에 보다 깊게 몰입할 수 있는 요인을 파악하여, 가상환경의 설계에 응용할 수 있어야 한다. 하지만 이러한 평가방법들은 상당히 많이 존재하며, 정량화되어있지 않다.

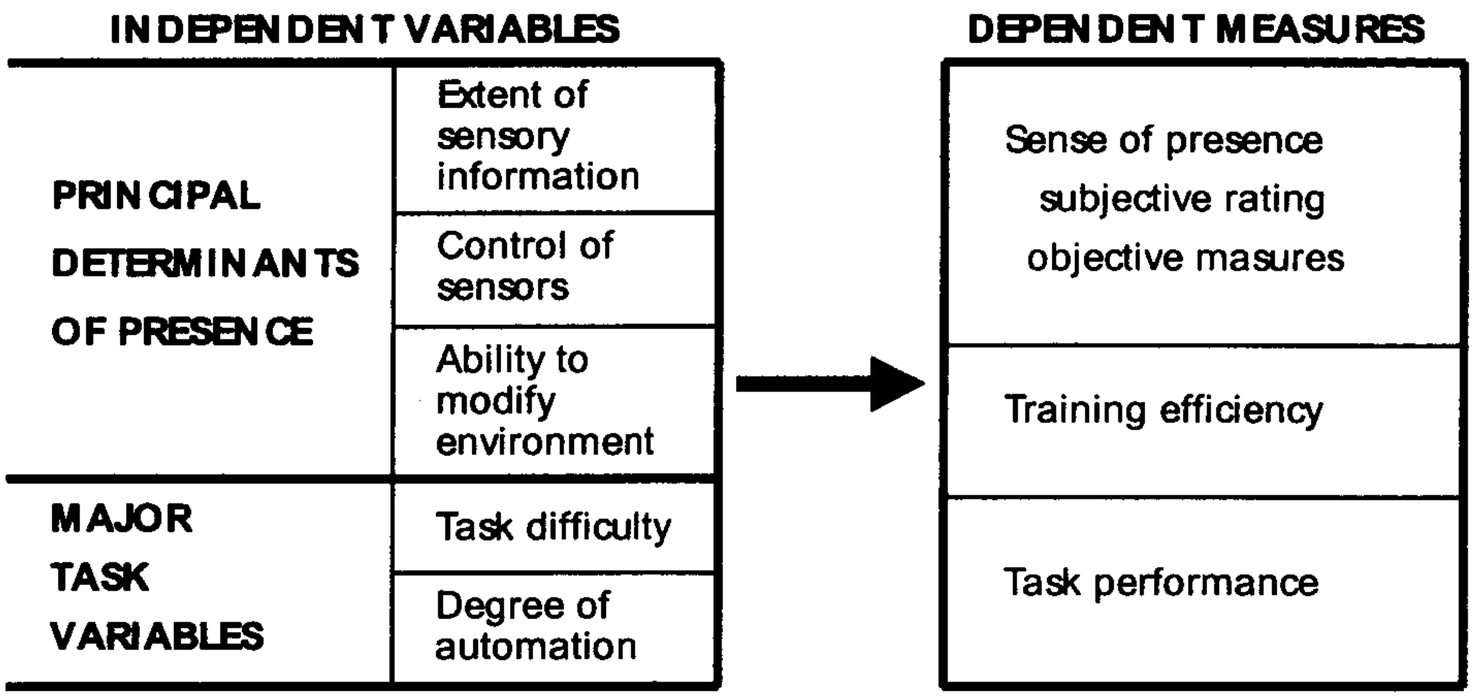
여섯째, quality of pictorial scene. 보여지는 장면의 현실감정도를 실험을 통하여 결정하여야 한다.

일곱째, stereoscopic displays. HMD의 각 display장치에 어떠한 시각에서 보여주어야지 3-D로 보여질 수 있는지는 인간의 눈의 구조를 분석함으로써 구현할 수 있다.

나. 현실감평가 방법론

현실감을 평가하는 많은 방법들이 존재하고 있지만, 현실감자체가 매우 주관적인 특성을 가지고 있기 때문에 정확하게 정량화시키는 상당히 힘들다. 결국 구축된 시스템의 특성 및 사용자의 특성에 따라 상당히 많은 영향을 받는다고 할 수 있다.

<그림 75>은 가상환경에서의 임장감, 학습성, 효율성, 수행에 대한 평가를 위해서 각각의 독립변수에 따른 종속변수를 결정하여, 이를 평가하는 방법을 보여주고 있다. 임장감의 정도는 subjective rating과 objective measures에 의해 측정된다.



<그림 75> 임장감, 학습성, 효율성, 수행의 실험측정

가상환경 시스템의 현실감을 평가하기위한 방법론은 개인의 주관적인 특성을 파악하기 위한 psychophysical measures방법이 많이 사용된다. 이러한 측정방법은 사용자들이 느끼는 부분에 대해 rating을 하는 것으로써, 항목의 설정이 상당히 중요하다.

Witmer는 가상환경 시스템의 임장감을 측정하기 위하여 임장감에 영향을 미치는 여러가지 요소들을 분류하여, Presence Questionnaire을 만들었다. 이를 이용하여 사용자들에 대한 rating을 통하여, 구축된 가상환경 시스템이 임장감에 어떻게 영향을 미치는지 판명하였다. 본 실험에서도 해상도, 시야(Field of view), 디스플레이의 frame rate, interface의 반응속도 항목에 대한 rating을 통하여 임장감에 영향을 미치는 요인을 분석하였다.

인간이 환경에서 얻게 되는 정보의 정도를 측정하기 위하여 본 실험에서는 Sketch map methode을 이용하였다. Sketch map method는 여러 인지심리학자들이나 다른 분야의 학자들에 의해 쓰이고 있는 방법으로서 상당히 간단한 방법과 직관적인 분석이 가능함으로 인해 여러 분야에서 널리 쓰이고 있는 방법이다. 개별적 인지지도(cognitive map)는 공간적 심상의 활성 정보 탐색 구조이며, 인지지도는 또한 객체들과 운동감각, 시각적, 청각적 암시들의 기억으로 구성되어진다.

이러한 인지지도에 대한 연구의 어려움은 개인별 내적지도의 외적표현에 대한 추출의 문제이다. 즉 각 인간들에게 내적으로 확립되어있는 추상적인 인지지도를 어떻게 외부로 표현하게 할것인가 하는 것이 상당히 어려움 문제로 남아있는 것이다.

Golledge는 인지지도의 평가하는데 있어서 다음의 서로다른 4가지 방법를 제시하였다[55].

- Experimenter observation of subject behavior
- Historical reconstruction
- Analysis of external representations
- Indirect judgment tasks

본 실험에서는 피실험자들이 작성한 인지지도와 실험을 주관한 관찰자에 의해서 rating을 하였다.

다. 현실감평가 실험항목

본 실험에서는 감각기관별 측정항목(시청각제시기가 응용될 분야)을 구성하고 있는 아주 기본적인 요소에 대해서 실험이 이루어졌다. 다시말해, size, slope, angle, length, area, volume 같은 현실세계를 이루고 있는 가장 기본적인 요소들에 대한 simulator와 현실과의 차이점을 찾으려 하였다.

가상환경은 length가 기본이 되어 size와 distance를 갖는 환경을 구성하기 때문에 3차원 시청각 환경 제시기의 현실감 평가 실험에서는 기본적으로 제시기가 보여주는 length를 측정하고 length를 기반으로 하는 size와 distance를 측정하는 실험을 설계하였다.

Size는 length로 이루어진 물체가 지니게 되는 2차원이나 3차원적인 크기를 말하고, length는 물체의 변이 이루고 있는 길이 단위의 정도를 가리킨다. distance는 평면적인 의미를 지니는 length라고 할 수 있으며 관찰자의 움직임에 따른 traversed distance와 움직임을 동반하지 않는 perceived distance로 나눌 수 있다 [56]. 가상환경은 length가 기본이 되어 size와 distance를 갖는 환경을 구성하기 때문에 3차원 시청각 환경 제시기의 현실감 평가 실험에서는 기본적으로 제시기가 보여주는 length를 측정하고 length를 기반으로 하는 size와 distance를 측정하는 실험을 설계하였다.

인간이 현실과 가상에서 주변환경에 대한 정보를 어떻게 받아들이는지에 대한 차이점을 조사함으로써, 현재의 제한적인 기술로서 보다 높은 수준의 현실감을 실현하기 위하여, 가상이 현실과 어떠한 점에서 비슷해져야 하는가를 고찰하였다. 이에대한 실험으로 sketch map method을 이용하여 피실험자가 작성한 cognitive map에 대한 분석을 실시하였다. Map goodness, object classes, relative object positioning 측면에서 평가하였다. 인간의 object 인식에 대한 실험으로 object search을 실시하였다 [57].

(1) 실험방법 및 계획

본 연구에서 실시한 실험은 Cognitive map, Object search, Size distance estimation 로 이루어져 있다. 실험에서 이루어지는 평가방법은 5점척도를 이용하여 분석을 실시하였다.

(가) Cognitive map

① 실험목적

Wayfinding을 위한 Virtual world의 제작에 있어서, 인간의 머리속에 world의 map이 얼마나 잘 구축되는가가 매우 중요한 부분이다. 그러므로 구축된 virtual world에서 인간이 navigation하는 동안 인간들의 머리속에 어떠한 map이 생성되는지를 평가할 필요가 있다. 이를 위해 본 실험에서는 sketch maps method를 이용하여, 현실과 가상환경에서의 차이를 살펴보았다.

② 피실험자

피 실험자 그룹은 between-subject design에 의해 모두 30명을 선발하였다. 피실험자는 공간지식을 가진 부류로써 주로 이공계열 대학생들을 중심으로 간단한 실험 및 설문을 통하여 선발하였다. 그룹간의 차이를 최소화하기 위하여, 20명에 대한 질의을 통해 random하게 구성하였다.

③ 실험계획

피실험자들은 현실, 가상과 video에서 10분간 자전거를 이용한 navigation을 실시하게 된다. 가상에서는 자전거 인터페이스을 사용자들이 익숙하게 쓰기 위하여 pretest world를 만들어 10분간 훈련을 시켰다.

(m : male, f : female)

피실험자	상 황	Navigation path
Group-A (m:6, f:4)	현 실	임 의
Group-B (m:6, f:4)	가상현실	임 의
Group-C (m:5, f:4)	비 디 오	고 정

④ 평가방법

인간의 머리속에 구축된 map을 밖으로 표현한다는 것은 개인적인 능력에 좌우되기 때문에, 매우 어렵고 여러가지 문제점을 내포하고 있다. Golledge는 환경에 대한 인지적 정보를 끄집어내기 위해아 다음과 같이 4가지 방법을 제시하였다.

- 피실험자 행동의 관찰
- 실험과정의 분석
- 작성된 map의 분석
- 간접적인 평가

본 실험에서는 피실험자 행동을 바탕으로 작성된 map에 대한 분석을 실시하였다.

피실험자가 작성한 map에 대한 평가는 다음과 같은 3가지 항목에서 5점 척도를 실시하였다. 평가를 실시한 자는 실험을 진행한 2명의 실험자들에 의해 작성되었다.

- Map goodness

  전체적인 map이 현실과 어느정도로 비슷한가?

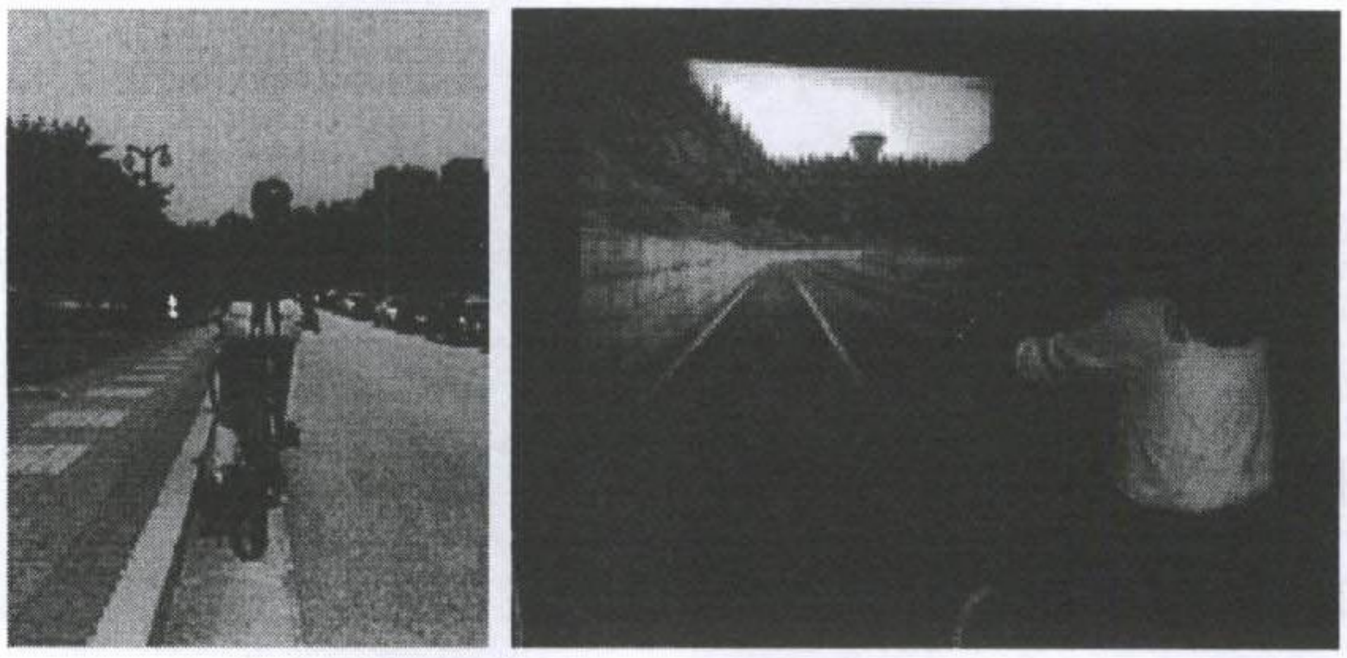
- object classes

  · map goodness의 하위 평가항목으로써 인식된 object를 어떻게 표현하였는가?

- relative object positioning

  · 하위 평가항목으로써, object간의 scale이 실제와 어느정도 비슷한가?

가상환경에서의 map goodness 평가값은 피실험자들이 설문한 시뮬레이터의 해상도(resolution), 시야(Field of view), Display부드러움(frame/sec), 자전거의 반응속도의 4가지 항목에 대한 상관관계분석을 통하여, map goodness에 영향을 미치는 인자를 도출하였다.



<그림 76> 가상과 현실에서의 실험장면

(나) Object Search

① 실험목적

이전의 경험을 바탕으로 world 상에 제시된 object을 찾아 가는 것이다. 과거에 정보가 얼마정도 남아있는지, 현실과 가상에서 똑같은 object을 얼마나 다르게 인식하는지, 목적지에 도달하는 시간/path 등을 파악함으로써, 가상세계와 현실세계와의 차이점을 규명하려하였다.

② 피실험자

피실험자 group은 cognitive map 실험을 행한 자로 한다.

③ 실험계획

현실과 가상에서 동일한 object(쓰레기통)를 선택하였다. 이 object를 가상에 현실과 똑같은 크기, 위치로 모델링하였다. 피실험자들이 제시된 object들을 어떤 경로를 따라 찾아가는지, 시간은 얼마나 걸리는지, 실수로 지나치는 경우가 있는지에 대해서 관찰한다.

(다) Size distance estimation

① 실험목적

환경내에서 object를 인식하는데 있어서, 이들에대한 size, distance의 차이가 현실과 가상에서 얼마나 존재하는지를 평가함으로써, 가상현실시스템의 현실감을 높이기 위해서 어떠한 요소를 조정해야 하는지를 파악하는 것이 목적이다.

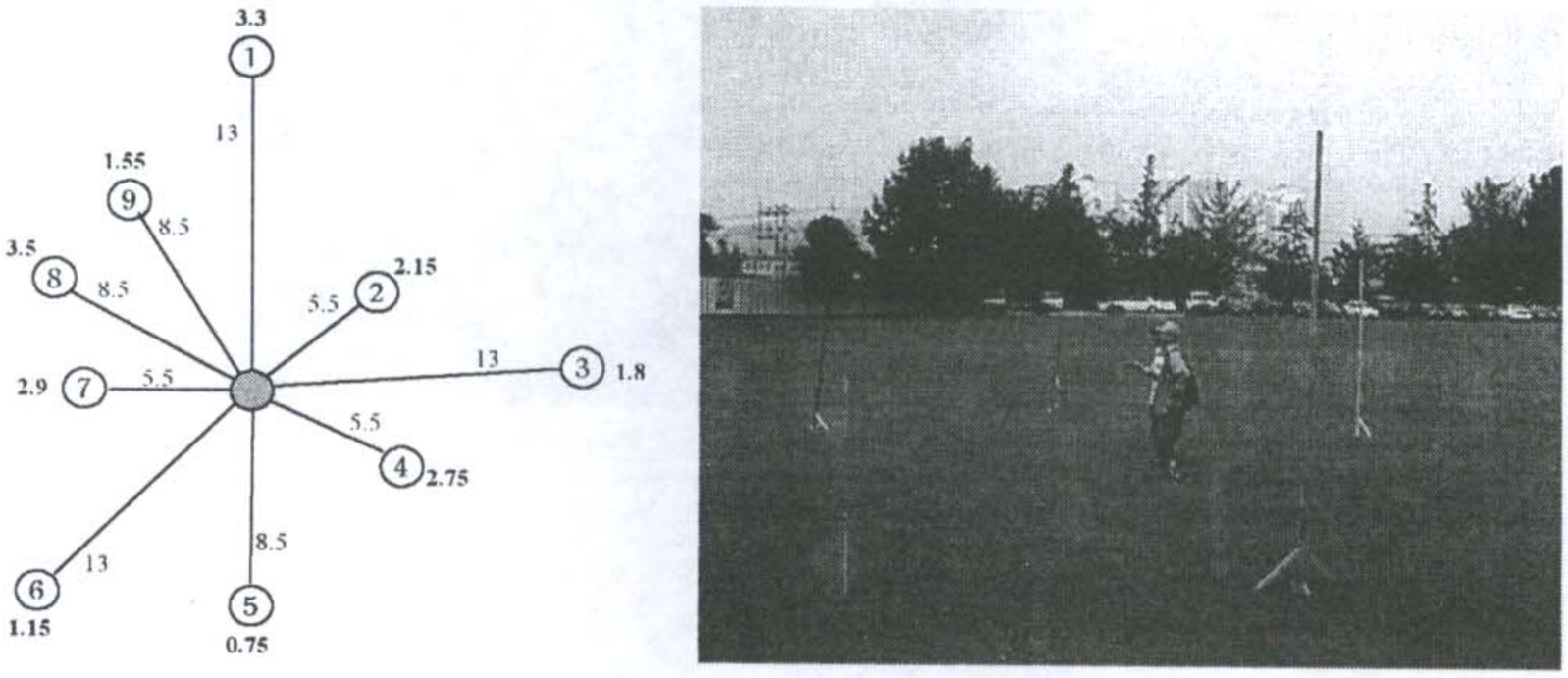
② 피실험자

피실험자들은 크기, 거리를 정확하게 측정할 수 있는 자로 선발하였다. 이를 위해 선별과정에서 간단한 object에 대한 크기와 거리를 측정하게 하여, 적합한 자를 선택하였다. 가상과 현실의 집단간의 차이를 줄이기 위하여 피실험자들을 random하게 그룹에 배치하였다.

③ 실험계획

피실험자들은 여러 종류의 크기와 거리로 이루어진 막대기 주위 및 주변환경을 5분 정도 둘러보게 한 후, 정해진 위치에서 각 막대기의 크기와 거리를 측정하게 하였다.

막대기의 크기는 0.75, 1.15, 1.55, 1.8, 2.15, 2.75, 2.9, 3.3, 3.5m로 구성하였으며, 거리는 5.5, 8.3, 13m로 하였다. 이러한 수치는 막대기 제작과 주변환경을 고려하여 선정되었다.



<그림 77> Size, distance estimation 실험환경

(2) 실험환경 및 장비

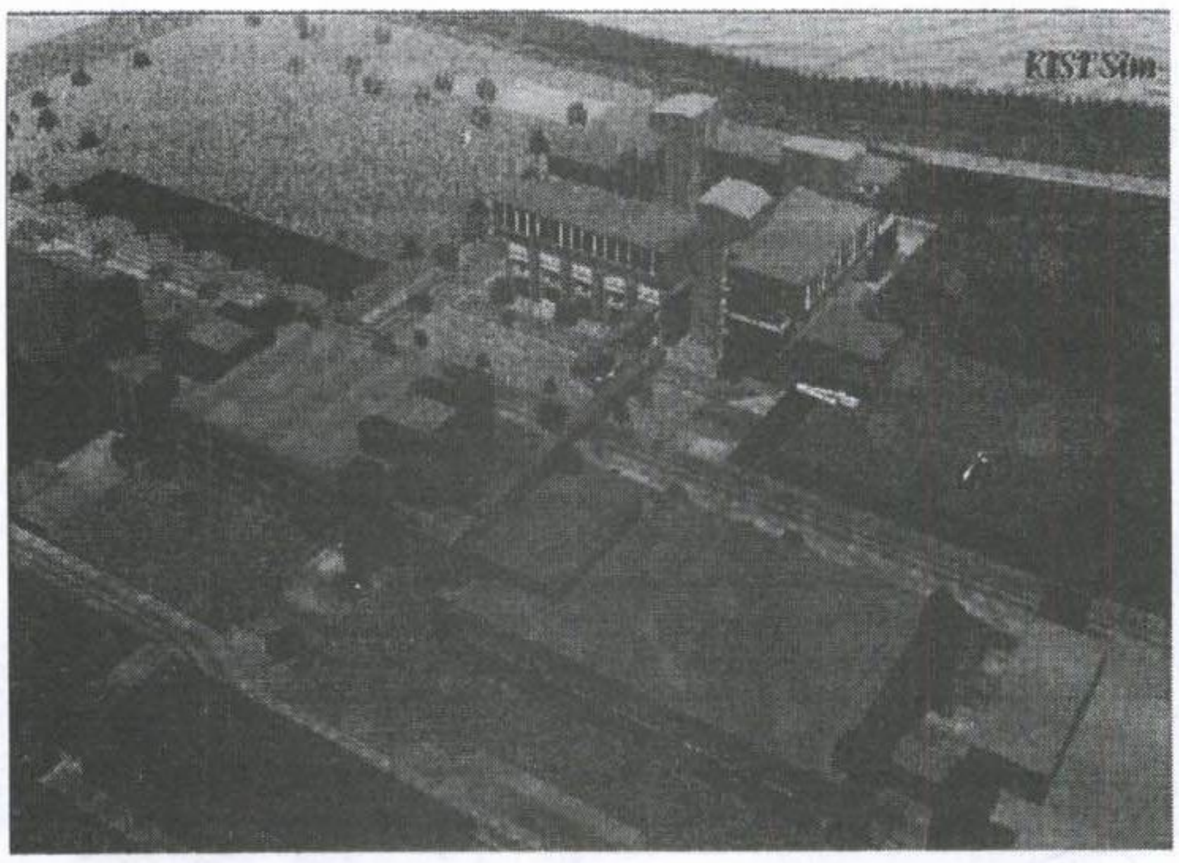
실험에서 사용된 가상환경은 감성공학의 일환으로 KIST Virtual Dream에서 개발된 3차원 시청각환경제시기를 이용하였다.

- H/W Platform: Silicon Graphics Workstation / IRIX 6.2, IBM-PC Window 95, TCP/IP.
- S/W Platform: IRIS Performer / OpenGL, X-Window, Motif API,

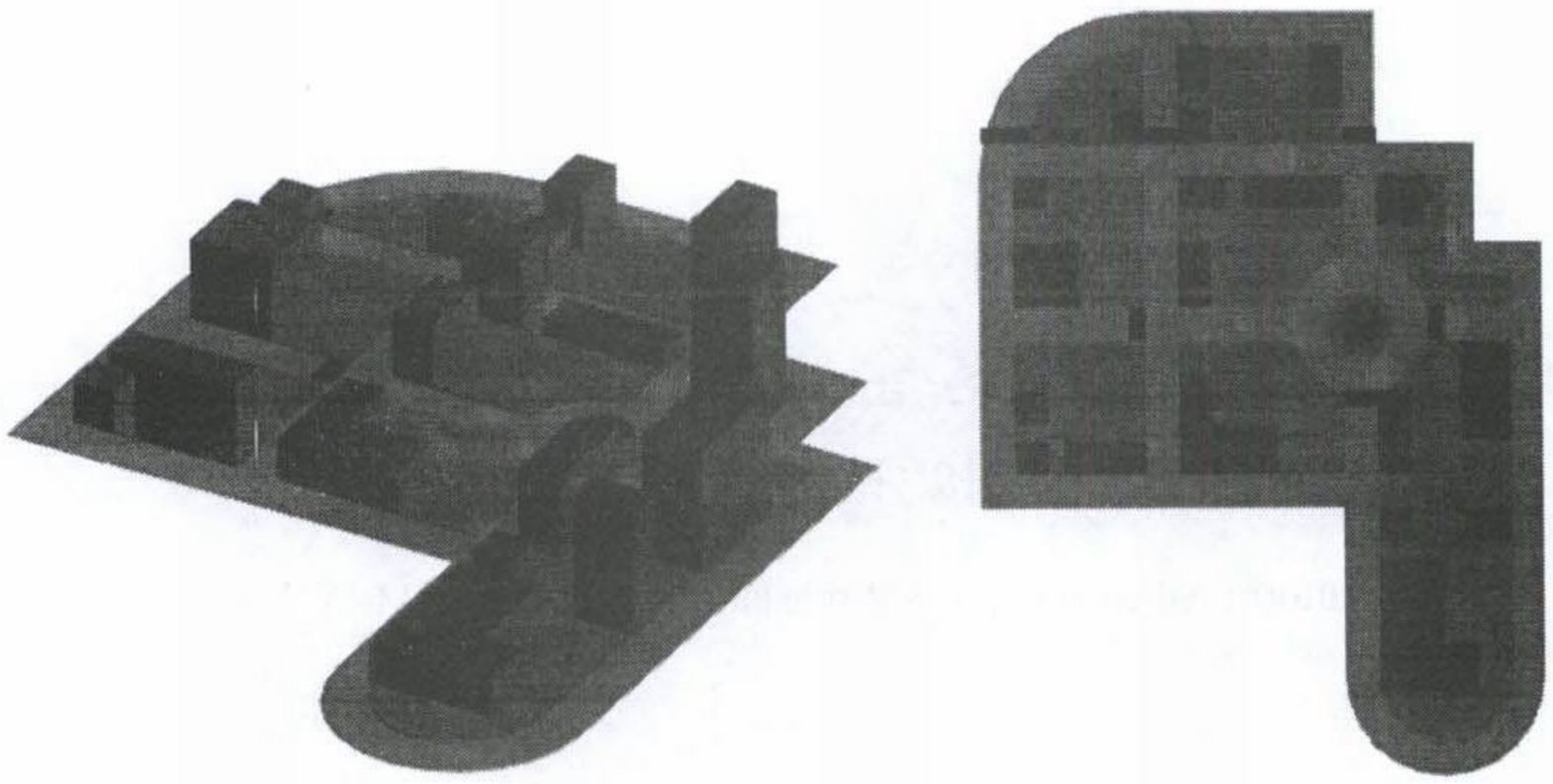
Scheme / C / C++.

- Visual Modeling Tools: MultiGen, Wavefront, 3D Studio.

<그림 78>은 구축된 가상의 KIST 환경을 보여주고 있으며, <그림 79>은 실험자가 KIST환경을 경험하기 이전에 가상환경 interface인 자전거를 숙지하기 위한 가상의 pretest 환경이다.



<그림 78> 구축된 KIST 가상 환경

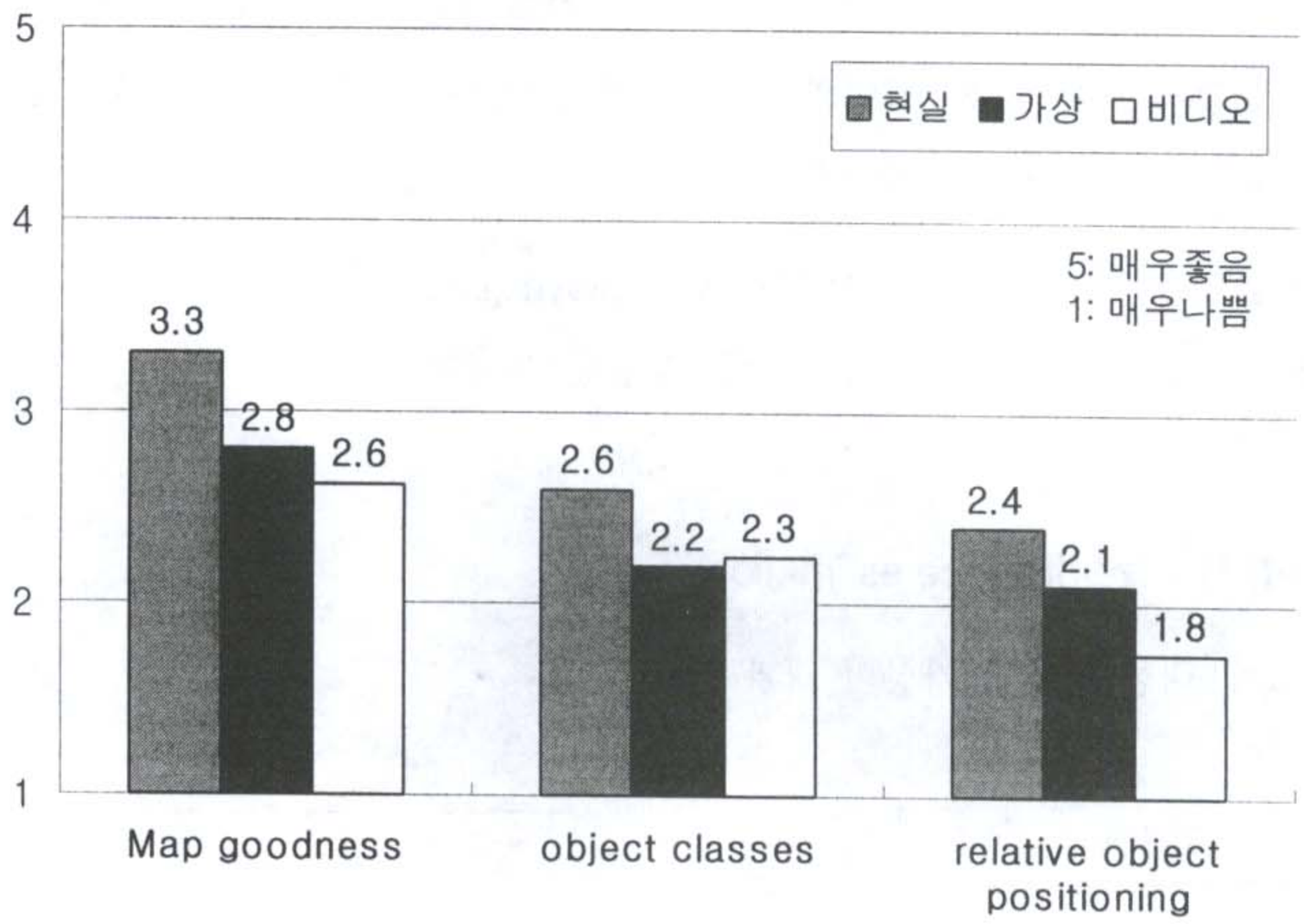


<그림 79> Pretest에 사용된 virtual world

(3) 실험결과

(가) Cognitive map

Sketch map analysis method을 이용하여, 피실험자들이 작성한 map에 대한 map goodness, object classes, relative object positioning에 대한 5점 척도를 실시하였다. 실험당시 피실험자들 계속해서 관찰한 2명의 실험자에 의해 작성하였다. 결과는 다음과 같다.



<그림 80> 작성된 map에 대한 5점척도 결과

가상환경에서는 FOV(Field of View)가 현실보다 좁기 때문에 환경에 대한 공간인지 면에서 정보를 적게 획득하여 수행이 떨어짐을 볼 수 있다. 하지만, object search 실험까지 끝난 후 다시 map을 피실험자에게 작성하도록 하였을 때, 현실과 차이점을 찾기 어려웠다. 이것은 FOV가 좁아 획득되는 정보는 적지만, 경험하는 시간이 흐름에 따라 현실과 비슷해 짐을 볼 수 있었다. 이러한 추가적인 시간을 결정할 수 있다면, 현 시스템을 이용하여 현실과 비슷한 가상의 경험을 할 수 있다.

비디오 실험은 가상에서 보여지는 환경보다 현실감이 뛰어나지만, sketch map을 평가한 결과 가상환경보다 나쁘게 나타났다. 이러한 이유로는 고정된 path를 따라 단순히

찍은 화면을 보여줌으로써 피실험자들이 원하는 방향을 볼 수 없고, 화면을 관찰함으로써 10분간의 실험동안 중반 이후부터는 집중력이 급격히 떨어져 환경을 인식하는데 어려움을 느꼈다.

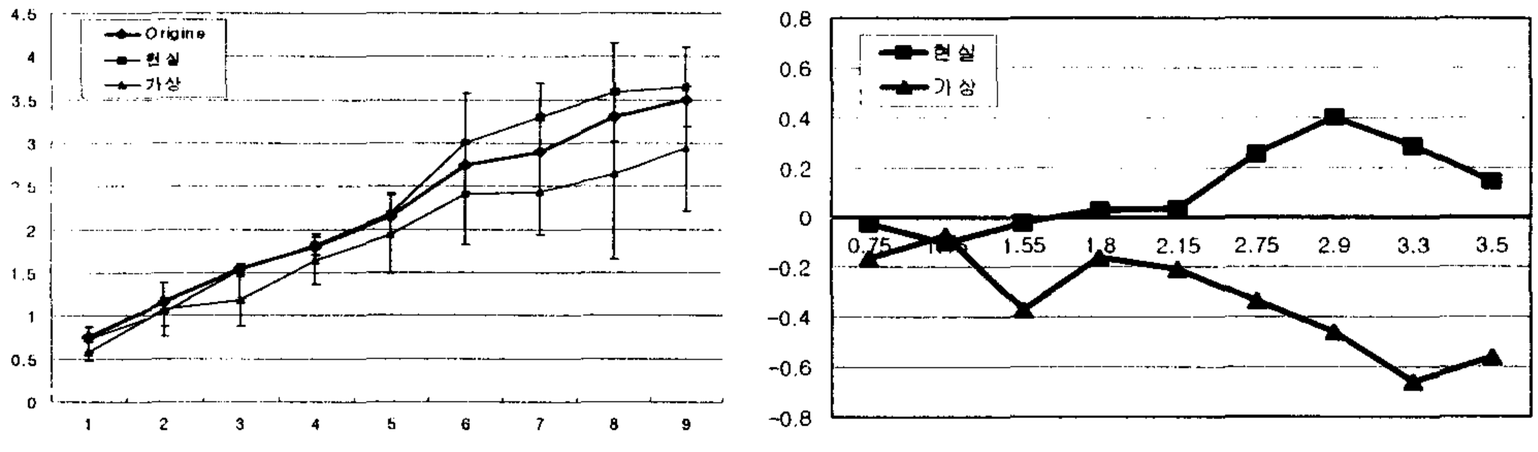
(나) Object search

현실과 가상 모두에서 피실험자들이 이전에 경험한 장소에 있는 object를 모두 찾아 냈다. object를 인지 못하고 지난 경우는 없었다. 가상의 실험에서 2명의 피실험자들이 object를 찾아 navigation을 할 때, 공간상의 거리를 잘못 인식하고, 먼 거리로 돌아가는 경우가 있었다. 또한 건물 사이의 좁은 길을 인지하지 못하고 계속해서 넓은 길로만 navigation하는 것을 관찰 할 수 있었다.

이러한 결과를 바탕으로 FOV가 좁은 wayfinging 위주의 가상환경 구축시 도로표지 시스템을 보다 자세히 만들어야 함을 도출할 수 있다.

(다) Size distance estimation

Size에 대한 현실과 가상의 결과 값은 다음과 같다.

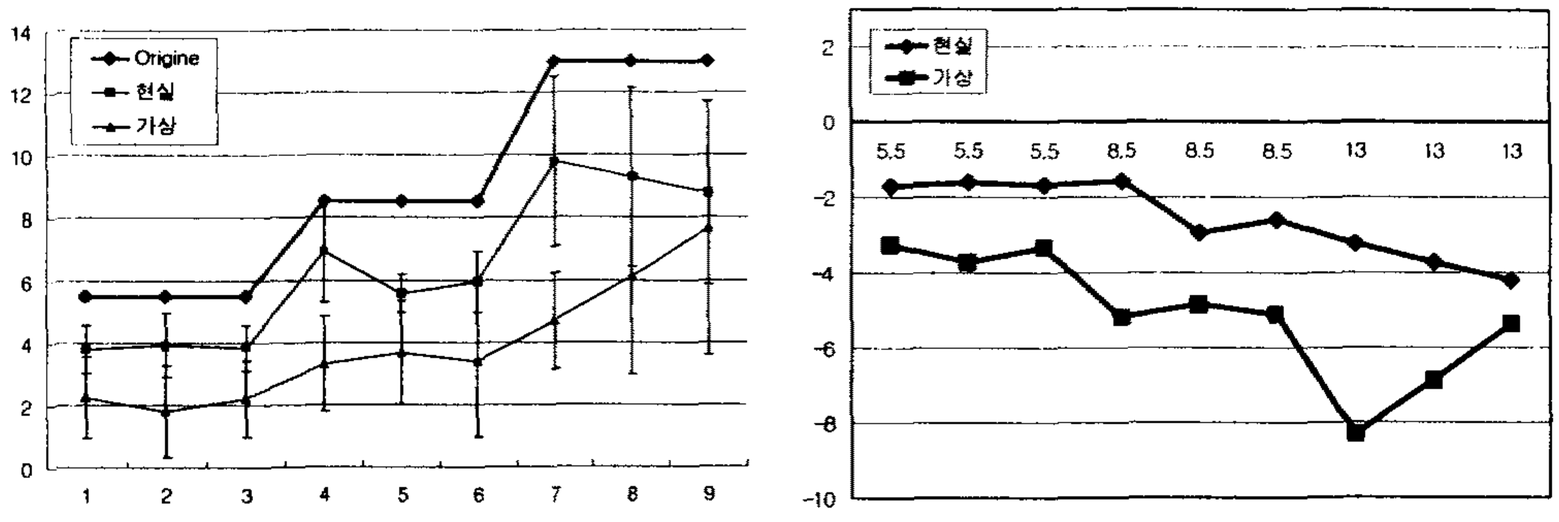


<그림 81> 현실과 가상의 size estimation

선행 연구된 실험결과에서는 실험환경에 따라 actual size에 비해 under-, over-estimation 하는 경향이 있는 것으로 나타나, 실험환경에 따라 좌우되는 것으로 조사되었다. 본 실험결과에서는 현실에서 over-estimation하는 것으로 나타났고, 가상에서는 under-estimation하는 것으로 나타났다. 이러한 결과는 실험환경적인 영향이 있는것으로 판단이

되나 그 원인을 규명할 수 없었다. 하지만 현실과 가상환경에서의 estimation차이 수준은 타 연구수행결과와 비슷하게 나타났다.

가상과 현실 모두에서 크기가 커짐에 따라 예측를 잘 하지 못했다. 반대로 2m 이하의 크기는 가상과 현실 모두에서 잘 예측하고 있었으며, 통계적인 분석을 통한 집단간의 비교에도 차이점을 찾지 못했다.



<그림 82> 현실과 가상의 distance(depth) estimation

Distance 예측에서는 가상과 현실 모두 잘 예측하지 못하고 있었으나, 가상환경에서의 예측이 현실보다 더 크게 under estimation한 이유는 피실험자 위치에서 막대까지의 거리 중 절반부분이 프로젝션의 FOV가 좁은 이유로 피실험자들이 다리를 볼 수 없었기 때문이다.

(라) 가상환경 시뮬레이터에 대한 평가

시뮬레이터 평가를 위하여 다음과 같은 항목에 대한 설문을 실시하였다.

- 시각적인 측면에서의 평가
  - 스크린의 해상도
  - 피실험자가 느끼는 시스템의 Field of view
  - 디스플레이되는 화면의 부드러움 정도
- Object들과의 상호작용에 대한 평가
  - 풀밭에 들어갔을 때의 현실과의 비교

- 벽에 부딪혔을 때 현실과의 비교
- 요철을 지나 갔을 때의 현실과의 비교

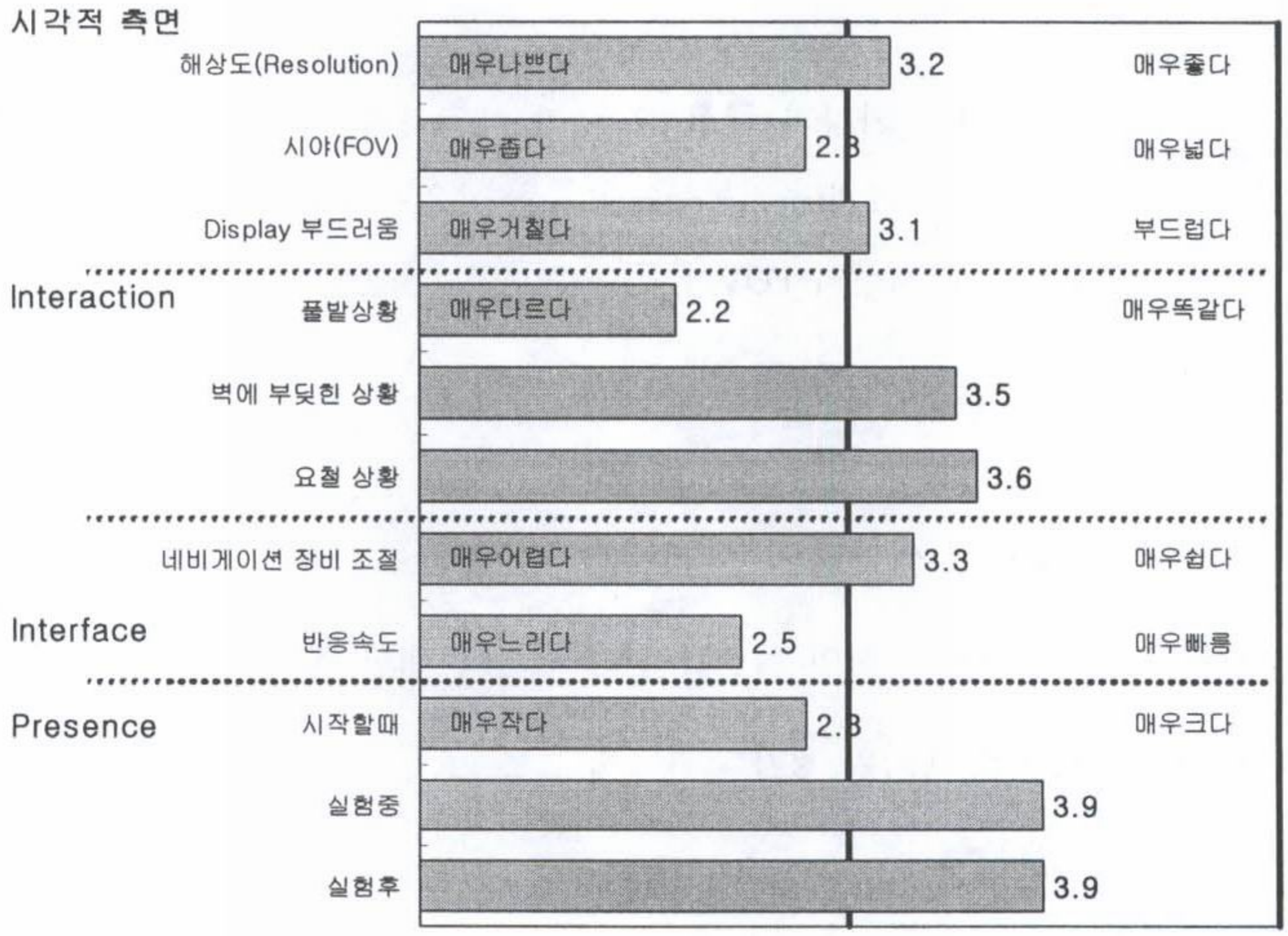
● 물리적 휴먼인터페이스에 대한 평가

- 자전거 사용 용이성
- 자전거의 반응속도

● 가상현실에 몰입된 정도

- 가상현실 실험의 시작 직후, 실험 중, 실험 후 시점에서의 몰입정도

설문결과는 다음과 같이 나타났다.

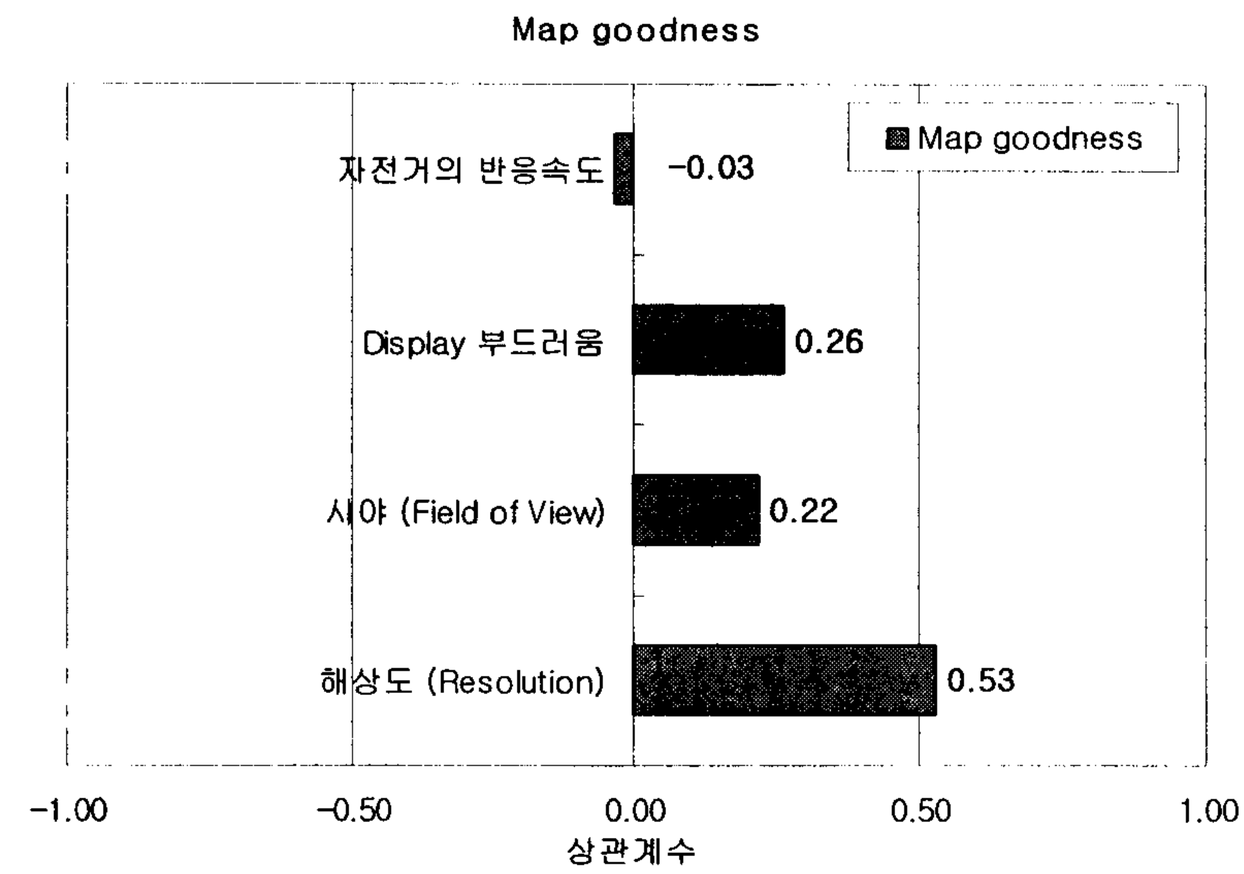


<그림 83> 시뮬레이터에 대한 설문결과

가상환경에서 벽에 부딪힐 때는 force feedback, 요철을 넘어갈 때 시각적인 피드백을 줌으로써, 피실험자들이 현실과 비슷하다고 느꼈다. 단지 한가지의 피드백만으로 사

람들은 상당히 현실에 가깝다고 느끼고 있다. 피실험자들이 simulator에서 보내는 시간이 길어질 수록 임장감이 향상되는 것을 볼 수 있다.

다음은 위의 설문의 시각적측면(해상도, 시야, display 부드러움), interface(반응속도)와 cognitive map 실험에서 얻은 Map goodness와의 상관관계를 분석하였다. 이러한 분석은 인간의 머리속에 map이 생성될 때 어떠한 요소가 많은 영향을 미치는지를 살펴보기 위해서다.



<그림 84> Map goodness scale과 측정항목간의 상관관계

위의 그래프와 같이, 해상도의 수준이 map을 생성시키는 상당히 크게 작용하고 있음을 볼 수 있다. 시야(FOV)항목은 map goodness가 좋은 피실험자일수록 본 시뮬레이터에서 느끼는 FOV를 넓게 인식하고 있었다. 또한 자전거의 반응속도는 전혀 영향을 미치고 있지 않다. 위 설문결과를 보면 피실험자들이 자전거 인터페이스의 반응속도가 매우 느린것으로 느끼고 있다. 이럴 경우 인터페이스의 속도가 실험의 큰 영향을 미치지 않는 경우에는 현실보다 빠르게 움직이게 설계할 필요가 있다. 실질적으로 피실험자와의 인터뷰에서도 현 인터페이스의 속도가 너무 느려, 단순힌 wayfinding 실험에서 지루함을 느껴 결과적으로 몰입감을 떨어트리는 것으로 나타났다.

라. 임장감(Presence)을 향상시키기 위한 요인

가상환경에서의 임장감(presence)을 향상시키기 시켜 인간을 완전히 가상세계에 몰입시키기 위해서는 시각, 청각, 촉각, 후각, 미각 등의 인간의 오감에 실제에 가까운 자극을 인공적으로 제공하여야 한다. 본 시스템에서는 시각과 청각만을 이용하여 임장감의 정도를 측정하여 보았다. 앞에 도식화한 실험결과에 의하면, 지금의 시스템에서 임장감의 정도를 보통으로 나타났다. 하지만, 본 실험 중 간단한 실험 몇가지와 피실험자와의 인터뷰를 통해서 다음과 같은 요소들을 추출할 수 있었다.

첫째, wayfinding simulator에서의 navigation 장치 속도. 시뮬레이터의 응용부분에서 시간의 중요성이 적은 경우에는 장치의 속도의 현실보다 높일 필요성이 있다. 실험에서 navigation 장치의 속도가 현실과 비슷한 수준으로 맞추어져 있었지만, 가상의 경우에는 한정된 시야에 의해 지루함을 느껴 가상환경에 몰입하는데 방해요소로써 작용을 하였다.

둘째, navigation장치의 force feedback or visual feedback. 실질적으로 가상환경에서 요철을 넘거나, 벽에 부딪히거나, 경사진 도로를 올라갈 때 등에서 단순한 visual feedback과 force feedback을 주었다. 이때 피실험자들은 feedback을 느낌으로써, 인간의 주의가 쏠려 가상환경에 더욱더 몰입함을 살펴볼 수 있었다. 하지만, 언덕을 올라가면서 force feedback을 받아 힘겹게 올라갔을 때, 피실험자들의 머리속에는 "내리막길에서는 페달을 밟지 않아도 내려가겠구나"라는 생각을 하게 된다. 이러한 것은 인간이 느끼는 양립성에서 비롯된 것이다. 이때 이러한 기대가 깨어지면 몰입감을 방해할 수 있다. 그러므로 feedback을 줄 경우 자극과 반응에 근거하여 생성시켰야 한다.

셋째, navigation을 하기 전에 적절한 event의 결정. 위의 실험결과에 의하면, 피실험자들이 가상환경을 경험하기 시작할 때 몰입되는 정도가 매우 낮았고, 시간이 지나면서 그 정도는 계속해서 올라갔다. 이러한 결과를 바탕으로 실질적으로 simulator에서 네비게이션하기 이전에 가상의 환상적인 장면을 제시 함으로써, 시작부터 몰입감을 증대시킬 필요가 있다. 예를 들어, 하늘로 높이 날아올라서 빠른 속도로 땅으로 내려오거나, 실험과 상관없는 복잡한 환경을 빠른 속도로 보여준다. 실질적으로 2명의 피실험자에 대해 이러한 예를 적용시켰을 때 다른 피실험자들보다 더 빨리 몰입함을 볼 수 있었다.

마. 결론 및 추후연구과제

본 연구에서는 현재 구축된 3차원 시청각환경의 시각적인 부분에 대한 현실감(reality)을 비교평가하여 차이점을 규명하려 하였다. 가상환경에서의 환경에 대한 필요한 만큼의 정보를 얻는데 있어서, 현 시스템의 FOV가 좁아 현실보다 많은 시간이 필요했다. 네비게이션장치의 속도가 인간의 인지수행능력에 영향을 미치지 않은 결과를 바탕으로, 장치의 속도를 증가시켜 좁은 FOV의 한계를 극복할 수 있다.object 인식에 대해서는 차이점을 발견할 수 없었다. 현실감 자체가 매우 주관적인 것으로 객관적 평가가 매우 힘들었다.

지금까지의 실험결과를 바탕으로 보다 체계적인 분석 및 타 연구과제와의 비교를 통해, 경험의 전달을 위해서는 실제의 상황과 어떠한 점에서 비슷해져야 하는가를 파악하여, 가상환경 시스템의 현실감을 경제적으로 향상 시키는 방안을 제시하며, 가상환경 시스템의 인간공학적인 평가기준을 확립하는 것이 추후연구과제이다. 또한, 현재의 시청각제시기의 현실감을 보다 높일 수 있는 World Modelling Guideline을 구축하는 것을 본 연구의 최종목적이다.

여 백

제 4 장 연구개발목표 달성도 및 대외기여도

3차원 시청각 환경 제시 시스템의 개발은 '95년 12월부터 '98년 11월까지 총 3년간 연차적으로 수행되었으며, 1차 년도에는 관련자료 수집과 외국 구축사례 조사를 통한 기초연구에 중점을 두었다. 2차 년도에는 전체 시스템의 하드웨어 및 소프트웨어 구조의 설계와 주요 기능 모듈들의 설계 및 구현에 역점을 두었고, 3차 년도에는 기존 모듈의 보완 발전 및 추가적인 기능 모듈들 개발과 시스템 통합 및 검증에 중점을 두고 수행하였다. 연차별로 주요 연구개발 목표들을 정리해 보면 다음과 같다.

- 1차 년도('95.12 - '96.11)
  - 가상현실 시스템 및 시뮬레이터 관련자료 수집과 해외 구축사례 조사
  - 시, 청각 동기화 구현 및 청각 전달함수 등에 의거한 3차원 청각 제시 시스템 조사
  - 감성측정평가 시뮬레이터의 시청각 환경 제시 시스템 설계
  - 시청각 환경 제시 시스템의 기본 운영체계 구축
  - 시청각 환경 제시 시스템 개발을 위한 기본 하드웨어 및 소프트웨어 환경 선정

- 2차 년도('96.12 – '97.11)
  - 시청각 환경 제시 시스템 개발을 위한 하드웨어 및 소프트웨어 체계 구축
  - 시청각 환경 제시 시스템의 통합 소프트에어 구조 설계
  - 시각 환경 저작/제시 시스템의 설계 및 주요 기능모듈 개발
  - 청각 환경 저작/제시 시스템의 설계 및 주요 기능모듈 개발
  - 시뮬레이터 검증을 위한 기본연구 수행 및 주거환경 대상 프로토타입 시스템 설계 및 제작

- 3차 년도('97.12 – '98.11)
  - 시각 환경 저작/제시 시스템의 추가기능 개발 및 기존 개발기능 보완
  - 청각 환경 저작/제시 시스템의 추가기능 개발 및 기존 개발기능 보완
  - 가상환경내 동적객체의 행위모사 방법 연구, 행위모사 시스템 설계 및 구현
  - 스테레오스코픽 3차원 시각 파라미터 구현기술 연구 및 HMD 캘리브레이션(Calibration)

- 가상현실 시스템의 인간 공학적인 설계방법 연구 및 개발 시스템의 현실감 평가 연구
- 시각 및 청각 환경 저작/제시 시스템의 통합 및 기능 보완, 발전
- 시청각환경 제시 시스템을 활용한 자전거 주행 데모 시스템 구축

상기와 같이 설정한 연차별 연구개발 목표하에 KIST 연구진과 위탁과제 형태로 참여한 대학 연구팀이 공동으로 다수의 기초연구와 소프트웨어 시스템을 개발하였다. 먼저, 기초연구로 12차 비행/차량 시뮬레이션 갱신 강좌와 HICS '96 참가 등을 통하여 시뮬레이터와 가상현실 시스템에 관한 자료를 수집하였고, 미국의 Wright State 대학, 워싱톤 대학 HITL 및 Amstrong 연구소 등과 같은 전문기관에 소속된 국제 시뮬레이션 연구자와 상호교류 및 협조체제를 구축하였다.

한편, 3차원 시청각 환경 제시 시스템 개발과 직접적인 연관은 없으나, 시스템 설계 및 검증과 관련한 주변 연구가 있다. 즉, 3차원 시청각 환경 제시 시스템은 가상현실 기술을 주축으로 하기 때문에 가상현실 시스템을 설계하고 구현하는데 인간 공학적인 측면에서 중점적으로 고려해야 할 요소들을 추출하고, 개발 시스템이 제시하는 가상환경의 현실감을 평가하는 방법에 대한 연구와 이를 토대로 실제 데모 시스템의 현실감을 평가하는 연구를 수행하였다. 그리고, 3차원 시청각 환경 제시 시스템이 최종적으로 활용될 분야가 감성측정평가 시뮬레이터이기 때문에 이와 관련한 기초연구로 차량 시뮬레이터 검증 방법론 및 검증 프로쉬져에 대한 연구도 아울러 수행하였다.

또한, 개발 시스템이 제시할 3차원 시청각 환경의 몰입감과 입체감을 향상시키기 위해 스태레오스코픽 3차원 시각 파라미터 구현기술에 대한 연구를 수행하였으며, 이 연구결과를 활용하여 HMD 캘리브레이션 작업을 수행하였다. 그리고, 2차 년도와 3차 년도에는 일부 구현된 기능 모듈들을 이용하여 주거환경 데모 시스템과 자전거 주행 데모 시스템을 구축하여 운용함으로써 사용자의 요구사항 수렴과 개발 모듈의 검증작업 등을 수행하였다. 또한 이러한 데모 시스템을 다양한 정보기술 전시회에 출품함으로써 개발 시스템의 홍보 및 산업체 적용분야 개척 노력도 병행하여 수행하였다.

한편, 3차원 시청각 환경 제시 시스템 개발과 직접적으로 연관하여 수행된 연구개발 내용은 크게 3차원 시각 환경 시스템, 3차원 청각 환경 시스템 및 동적객체 행위모사 시스템 등의 개발이다. 먼저 3차원 시각 환경 시스템 개발을 위해서는 개발 플렛폼으로 활용할 하드웨어 시스템과 소프트웨어 개발도구 등에 대한 조사와 선정작업을 수행하였다. 하드웨어 플렛폼으로는 SGI 머쉰을 중심으로 다수의 펜티엄급 PC가 청각 환경과 행

위모사 기능 모듈들을 수행하는 분산구조로 설정하였다. 한편, 소프트웨어 개발도구로는 IRIS Performer를 중심으로 C++ 언어를 사용하고, 장비 인터페이스 개발 및 구성 모듈간 통합을 위해 C++과 스크립팅 언어인 Scheme을 사용하는 것으로 결정하였다.

또한, 3차원 청각 환경 시스템 개발은 시각 시스템과 행위모사 시스템에 부하를 줄이기 위해 PC에 Windows 95/98을 기반으로 개발하며, 이를 위하여 여러 가지 HRTF 상업용 시스템들에 대해 기능 및 성능에 관한 사전 조사연구를 실시하였다. 이러한 과정을 통해 최종적으로 개발 지원도구로 RSX 3D 라이브러리를 선정하였다. 그리고, 청각환경 그래픽 디스플레이를 위해 OpenGL을 사용했고, 음원저작 및 편집을 위해서는 MS Access를 사용하였다.

한편, 가상환경내 동적객체의 운동 또는 행위를 모사하기 위한 동적객체 행위모사 시스템은 청각 환경 시스템과 마찬가지로 PC를 기반으로 개발하였다. 행위모사 시스템을 개발하는데 활용한 개발도구는 포항공대가 다년간의 연구노력으로 개발한 ASADAL을 사용하였다. ASADAL/PROTO는 기존의 ASADAL이 가지고 있는 행위 및 기능 명세 및 시뮬레이션 기능에 형태 명세를 추가하여 형태, 행위, 기능의 명세를 가시화와 함께 시뮬레이션할 수 있게 확장한 방법 및 이를 구현한 도구이다.

상기와 같은 노력의 결과로 본 연구과제에서는 3차원 시각 환경 저작 및 제시 시스템, 3차원 청각 환경 저작 및 제시 시스템, 그리고 동적객체의 행위모사 시스템 등을 개발하였고, 이들은 다시 LAN을 통해 3차원 시청각 환경 제시 시스템으로 통합되었다. 또한, 통합과 더불어 3차원 시청각 환경 제시 시스템이 제시하는 가상환경에 대한 현실감 평가 연구도 아울러 실시하였다. 개별적인 개발 시스템들에 대한 상세한 내용은 본 연구보고서 3장에서 세부 연구 토픽별로 상세히 소개되어 있다. 비록 각각의 개발시스템이 기능적 또는 성능적으로 완벽하다고는 말할 수 없으나, 본 연구과제에서 궁극적으로 추구하고자 하는 감성측정평가 시뮬레이터의 시청각 환경 제시 시스템을 위한 토대를 마련하였다고 평가할 수 있다. 앞으로 기능적인 확장과 성능적인 향상 노력을 기울인다면 소기의 연구목표를 달성할 수 있을 것으로 사료된다.

한편, 본 연구과제를 통해 습득한 주요 개발 기술로는 3차원 시각 환경 저작 및 렌더링 기술, 3차원 사운드 저작 및 렌더링 기술, 가상객체 행위명세 및 모사 기술, 휴먼-컴퓨터 인터페이스 기술, 모션 및 Force Feedback 기술, 가상환경과 시뮬레이션 측면에서 다양한 큐들(시각, 청각, 모션, Control Feedback 등)의 통합기술, 그리고 가상환경의 현실

감 평가기술 등이다. 이러한 기술들은 우선 감성측정평가 시뮬레이터의 개발 및 활용을 통해 보다 인간 친화적인 산업제품을 생산함으로써 국산 산업제품의 경쟁력을 향상할 수 있을 것으로 기대된다.

본 연구과제를 통해 그 동안 외국기술에 많이 의존하고 있던 가상현실 및 시뮬레이터 분야의 원천기술을 확보함으로써 이러한 기술들을 기반으로 하는 체험형 교육 및 훈련, 게임/오락, 의료, 과학, CAD 등과 같은 다양한 산업분야의 발전에 크게 기여할 것으로 예상된다. 그리고, 본 연구과제를 통해 축적된 기술과 경험을 토대로 가상현실 기술의 산업화, 이를 활용한 고 부가가치 산업 및 제품의 창출이 가능할 것으로 기대된다.

제 1 절 3차원 시각 환경 제시기 개발

본 연구과제는 3년에 걸쳐 수행되었으며, 주된 개발 목표는 VR환경을 위한 3차원 시각환경 제시 시스템 개발과 청각 환경, 행위모사 시스템 등과의 통합을 이루며, 감성측정을 위한 통합 시뮬레이터를 개발하기 위해서 운동감, 촉감, 후각, 열환경 등의 타 모의환경들과의 효율적인 통합을 고려한 시청각 구조를 마련하는 것이다. 개발된 시스템을 통하여 시각 환경을 중심으로 한 주거환경 데모 시스템을 개발하고 시연하였으며 청각 환경, 행위모사 시스템과의 통합을 이루어 자전거 주행 데모 시스템을 구축/시연 하였다.

<표 11> 시각환경 제시 기술 연차별 연구목표(계획서상의 목표)

개발 년차	연 구 개 발 목 표
1차 년도 ('96)	• 가사현실 시스템 및 시뮬레이터 관련 자료수집, 해외사례 조사 • 감성측정평가 시뮬레이터의 시청각 환경 제시 시스템 설계 • 시각 환경 제시 시스템의 기본 운영체계 구축 • 시각 환경 제시 시스템 개발을 위한 기본 H/W 및 S/W환경 선정
2차 년도 ('97)	• 시각 환경 제시 시스템의 통합 소프트웨어 구조 설계 • 시각 환경 저작/제시 시스템의 설계 및 주요 기능모듈 개발 • 주거환경을 대상으로 하는 프로토타입을 설계하고 제작
3차 년도 ('98)	• 시각 환경 저작/제시 시스템의 추가기능 개발 및 기존 기능 보완 • 시각 및 청각 환경 저작/제시 시스템의 통합 및 기능 보완, 발전

<표 12> 시각환경 제시 기술 연차별 연구결과

개발 년차	연 구 개 발 결 과
1차 년도 ('96)	• 시청각 환경 제시 시스템의 개략적 구조 설계 및 주요 기능 설립 • 시청각 환경 제시 시스템의 기본 운영체계 및 관리체계 구축 • 시각 환경 제시 시스템을 위한 제시기의 사양 및 개발 소프트웨어와 하드웨어 플랫폼 결정 및 확보 (SGI/IRIX 6.2, IRIS Performer)
2차 년도 ('97)	• 시청각 환경 제시 시스템의 통합 소프트웨어 구조 설계 완료 • 시각 환경 제시 시스템의 구조 설계 및 주요 기능모듈 개발완료 • 주거환경을 대상으로 한 데모 시스템 구축
3차 년도 ('98)	• 시각 환경 저작/제시 시스템의 기능 보완, 발전 및 개발 완료 • 시청각 환경 저작/제시 시스템의 통합, 보완, 평가 및 인증 실시 • 타 모의환경들(운동감,촉감,후각,열환경)의 효율적 통합을 고려한 소프트웨어 구조 설계 • 시청각 환경 제시 시스템을 활용한 자전거 주행 시스템 시연

개발된 3차원 시청각 환경 제시기는 그동안 외국기술에 많이 의존하고 있던 가상현실 및 시뮬레이터 분야의 원천기술을 확보함으로써 이러한 기술을 기반으로 하는 체험형 교육/훈련, 게임/오락, 의료, 과학, CAD등의 산업 분야의 발전에 기여하였다.

가상 프로토타이핑(Virtual Prototyping) 분야에서 제품이나 환경의 설계 및 생산에 소요되는 시간, 노력 및 비용 등을 현저히 절감할 수 있을 뿐만 아니라 제품의 질 향상에도 크게 기여하게 되었다. 또한, 시청각 환경 저작 시스템을 기반으로 개발된 가상 스튜디오(Virtual Studio)는 외국의 시스템을 대체하여 사용되었으며 이 분야의 기술을 세계적인 수준으로 끌어올렸다.

제 2 절 3차원 사운드 제시 시스템 개발

1. 연도별 연구목표의 달성도

본 연구과제는 1차년도에서 3차년도까지 3년에 걸쳐 수행되었으며, 주된 개발 목표는 VR환경을 위한 3차원 청각환경 제시 시스템 개발에 관한 것이다. 본 과제에서 개발된 시스템은 KIST의 VR 저작시스템과의 통합을 통하여 "감성공학 과제평가"와 "세계청소년과학 축전"에서 전시를 통하여 실시간 주행 데모를 성공적으로 수행한 바 있다.

<표 13> 청각환경 제시 기술 연차별 연구목표 및 달성도

년차	연구목표	세부연구목표	목표의 달성도
1차년도 (1996)	VR환경에서 시청각미디어 정립 및 시스템 설계	VR환경에서 시청각미디어 개념 정립	100 %
		시각미디어 동기화 제어 모듈 개발	100%
2차년도 (1997)	3차원 사운드 렌더링 및 저작 시스템의 개발	사운드 렌더링시스템의 개발	100%
		청각환경 모델링 도구 개발	100%
		통신 제어 API개발	100%
3차년도 (1998)	음원 DB 구축 및 검색 도구의 개발	음원 DB 스키마설계	100%
		음원 검색도구개발	95%

2. 관련분야의 기술발전에의 기여도

VR응용 시스템에서 3차원 사운드에 대한 중요성과 필요성이 부각되는 시점이지만 대부분은 시스템은 실험적인 수준에 머물렀다고 볼 수 있다.

본 과제에서 개발된 사운드 모델링 및 렌더링 시스템은 관련 분야에서 실질적으로 활용가능하며 하드웨어적으로 특별한 사양을 요구하지 않는 장점이 있다. 또한 청각환경 제어를 위한 API library를 제공하고 타 시스템과 통합이 용이하다고 할 수 있다.

API를 제공하여 시스템 통합이 용이하다.

특별한 하드웨어 장비를 요구하지 않고 소프트웨어적인 방식으로 실시간 사운드 렌더링이 가능하다.

그래픽 방식에 의한 사운드 모델링 개념을 사용하였다.

멀티미디어 타이틀, 게임, VR응용시스템 등 다양한 응용 시스템에 활용 될 수 있다.

제 3 절 Driving Simulator의 평가수법 개발

1. 연구개발 목표 달성도

가. 평가의 착안점

<표 14>Driving Simulator의 평가 수법 개발 프로젝트의 평가 착안점

구 분	평가의 착안점 및 척도
2차년도 (1997)	• 생리신호 측정시스템의 완성도는? • 거동측정 시스템의 완성도는? • 실차에서의 생리신호 및 거동데이터는 의미있게 측정되어졌는가?
3차년도 (1998)	• 실차에서의 운전 특성(인지) 평가는 의미있게 측정되어졌는가? • 장애자 재활훈련에 이용 가능한 Bike Simulator의 완성도는?

나. 목표의 달성도

(1) 휴대형 생리 신호 측정시스템의 구축

휴대형 생체신호 계측기를 자동차안에 구축하여 자동차가 실제 주행하면서 운전자가 받는 각종 자극에 대한 생리상태를 측정하였다.

(2) 신체 거동 측정 장치의 구축

3차원 위치 센서인 3-SPACE FASTRAK을 이용하여 운전자의 머리 및 양 어깨의 거동을 측정할 수 있었다.

(3) 실차 주행시의 가속도 변화의 측정

자동차에 가속도 센서를 부착하여 자동차에 가해지는 가속도를 측정하였다.

(4) 실차에서의 운전 특성(인지)의 측정

조수석에 동승한 피험자는 눈을 감은 상태에서 자동차가 주행한 거리와 선회한 각도를 인지하는 실험을 실시하여 탑승자의 인지특성 측정하였다.

(5) Bike Simulator의 개발

가상 현실감 기술을 이용하여 장애자의 감각을 통합적으로 자극할 수 있는 신체 자세 제어 훈련 시스템을 개발하였다.

2. 대외기여도

Driving Simulator는 주로 수학적인 모델링이 어려운 운전자의 행동이나 느낌, 반응 및 판단 등과 관계되는 기술개발에 많이 활용된다. 이러한 Driving Simulator와 실제 차량과의 현실감의 차이는 실험자와 피험자에게 중대한 오류를 범할 수 있게 된다.

본 연구에서는 Driving Simulator의 평가에 활용을 목적으로 실제 차량에서 운전자 운전특성(생리적, 거동학적)의 계측, 분석 시스템을 개발하여 Driving Simulator의 유효성 평가 지침에 활용 가능함을 시사하고, VR 시스템의 응용 기술로 장애자의 재활 훈련에 활용 가능한 Bike Simulator를 개발하여 장애자의 효과적인 재활훈련에 기여할 수 있는 시스템의 개발에 활용 가능하다.

제 4 절 가상환경 구축을 위한 행위명세 및 시뮬레이션에 관한 연구

1. 목표 달성도

가상객체의 행위, 형태, 기능을 나타내고, 객체간의 통신 등을 고려한 일반적인 객체지향 적인 중간언어 개발은 완료하였고 아사달과 가상환경 시스템(OpenInventor)을 연결한 예제도 완성하였으나, 실시간 동작을 위하여 OpenInventor이외의 Target언어로 변환하는 일과 모델 Library를 구축하는 것은 진행 중에 있다.

2. 기여도

가. 기술적 측면

형태의 효과적인 표현 방법을 습득하였고, 형태를 효과적으로 시뮬레이션하면서 가시화하는 기술 및 라이브러리도 획득하였다. 그리고, 행위, 기술, 형태가 모두 표현된 시스템을 시뮬레이션, 가시화 할 수 있는 도구를 얻게 되었다.

나. 산업경제적 측면

감성공학 연구용 가상 시스템 개발에 필요한 비용, 시간의 절감을 가져올 수 있게 되었고, 여러 다른 행위, 기능, 형태를 가지는 가상 시스템을 쉽게 만들어내어 감성공학 연구의 효율성을 극대화 할 수 있다. 그리고, 비행 시뮬레이터 등의 가상 현실 테스트베드, 원자력 발전소등의 시뮬레이션 분야에 파급 효과가 기대된다.

제 5 절 VR System의 인간공학적 설계기술 개발

본 연구의 최종목표는 VR 환경의 인간적합성 평가를 위한 인간공학적인 평가기술 개발이다. 그 세부목표는 다음과 같은 3가지로 이루어져 있다.

1. VR 시스템의 인간공학적인 요소 파악

VR을 위한 휴먼인터페이스와 관련된 인간의 시각적, 청각적 또는 촉각적인 특성 및 각 감각의 동기화에 대한 조사연구로써, 본 연구에서는 시각적인면을 중심으로 인간이 느끼는 현실감과 visual & force feedback에 대한 연구가 진행되었다.

2. 임장감(Presence)를 향상시키기 위한 요인파악

Zeltzer의 모형에 근거하여 시뮬레이션의 종류에 따라 필요한 fidelity요소들을 실험을 통하여 확인함. Wayfinding용 시뮬레이션에서의 cognitive map, object search, size estimation을 통하여 임장감을 향상시킬 수 있는 요인들을 도출하였다.

3. VR 시스템의 응용시나리오 작성

감성평가시뮬레이터의 용도에 맞는 VR 시스템에 대한 시나리오 및 시스템구성에 대한 방안을 연구. 본 연구에서는 구축된 wayfinding용 시뮬레이터에 대한 실험을 통하여, wayfinding 시뮬레이터가 보다 낳은 현실감을 사용자에게 제공하기 위한 여러가지 시나리오 가이드라인을 도출하였다.

제 5 장 연구개발 결과의 활용계획

3차원 시청각 환경 제시 시스템은 1차적으로 감성측정평가 시뮬레이터에서 필요로 하는 다양한 3차원 시청각 환경들을 저작하고, 제시하는데 활용될 것이다. 또한, 3차원 시청각 환경 제시 시스템을 가상 프로토타이핑 분야에 활용함으로써 제품이나 환경의 설계 및 생산에 소요되는 시간, 노력 및 비용 등을 현저히 절감할 수 있을 뿐만 아니라 제품의 질 향상에도 크게 기여할 수 있을 것이다.

한편, 3차원 시청각 환경 제시 시스템 개발을 통해 습득한 가상현실 기술과 시뮬레이션 관련 기술들을 앞으로 체험형 교육, 오락 및 훈련 등을 위한 각종 시뮬레이터(자동차, 항공기, 전차, 선박 등) 개발에 직접적으로 활용할 수 있을 것으로 기대된다. 또한, 3차원 시청각 환경 제시 시스템의 시청각 환경 저작 시스템을 기반으로 개발된 가상 스튜디오(Virtual Studio)는 향후 다양한 멀티미디어 응용분야에서 창의적인 응용 사례들을 구축하는데 활용할 계획이며, 공동 개발사인 문화방송에서는 선거방송, 스포츠 중계, 뉴스 보도방송 등에 이미 활용한 바 있으며, 앞으로 어린이 교육/놀이 프로그램 제작, 일기예보, 이벤트성 프로그램 제작 등에 확대하여 활용할 계획이다.

또한, 본 연구과제 수행을 통해 구축된 주거환경 설계 시스템과 자전거 주행 시스템은 개발 시스템의 산업화 적용 가능성을 보여 주었다. 실제 주거환경 설계 시스템은 동아건설과 공동으로 수행하였고, 현재 동아건설에서 모델 하우스 개발에 활발하게 활용하고 있다. 그리고 자전거 주행 시스템은 자전거를 타고 가상환경을 주행하는 시스템으로 헬스, 스포츠 및 의용공학 분야에서 많은 관심을 나타냈었다. 그러나, 개발 시스템의 이러한 산업화에 걸림돌로 작용하는 것 중에 하나가 개발 플랫폼이 워크스테이션이어서 하드웨어 구축 비용이 지나치게 높다는 것이다. 그래서 본 연구팀에서는 개발 시스템의 산업화를 촉진하기 위하여 PC 수준으로 다운싸이징하는 작업을 수행할 계획에 있다. 이러한 작업이 완료되고 나면 개발 시스템을 다양한 산업분야에 적용하는데 있어서 큰 초기비용 없이 가능하리라 생각된다.

현실감을 향상시킬 수 있는 여러가지 요인들과 가상현실 응용시나리오 작성에 있어서 고려해야할 사항들을 파악한 실험은 가상으로 구축된 가전제품이나 건축물 등의 평가시스템의 구축, 감성평가를 목적으로 구축된 가상환경 시스템의 응용시나리오 작성,

인공현실감 시스템의 현실감 및 임장감 향상과 관계된 연구 및 경제적으로 실현시킬 수 있는 연구, 인공현실감 장치 및 입출력장치의 평가 연구등에 활용될 수 있다.

한편, 본 연구과제에서 개발한 3차원 시청각 환경 제시 시스템은 오늘날 다양한 산업분야에서 적용하려는 가상현실 기술이 안고 있는 대부분이 요소 기술들을 포함하고 있기 때문에 본 연구를 통해 습득한 기술과 경험은 가상현실 기술의 적용사례 개발에도 곧 바로 활용할 수 있을 것으로 기대된다. 이러한 목적을 달성하기 위해서는 기존에 개발된 시스템의 신뢰성 향상과 기능적인 확장 및 성능적인 튜닝 작업이 필요할 것으로 사료된다. 이는 개발작업 자체를 통해서 이루어지기 보다는 기 개발된 시스템을 특정 응용분야에 적용하는 사례개발을 통해 달성되는 것이 사용자의 요구사항 수렴과 신뢰성 향상에 도움이 되리라 사료된다.

제 6 장 참고문헌

[1] 노무수, "산업환경의 변화와 감성과학", '97한국감성과학회 연차학술대회 논문집, 한국감성과학회, pp. 6-10, 1997.

[2] 한국표준과학연구원, "감성공학기술개발 연구기획 보완을 위한 연구", 과학기술처, pp. 5-47, 1996

[3] 감성공학기술개발 연구기획단, "감성공학기술개발사업 연구기획 최종보고서", 한국표준과학연구원, pp. 60-65, 1992

[4] 이남식, 박세진, "감성측정평가 시뮬레이터", 감성공학기술, 한국표준과학연구원, pp. 46-52, 1996.

[5] 고희동, "3차원 시청각 환경 제시기의 설계 및 구현", 대한전자공학회지, 1997.

[6] Klaus, Hanno, and Werner, "Prototyping Validation in Virtual Environments- Vision and First Implementation".

[7] Kerstin Forsberg, Viktoria Institute for Research on Information Technology, "The Virtual Product: An Advanced User Interface for Real Products ?"

[8] Computer Graphics World, "Collaborative Design", Sep. 1998. Vol. 21

[9] 이구형, "감성공학적 제품 개발과 생산을 위한 CAD/CAM", 한국 CAD/CAM 학회 이론/응용 연구회 워크샵, 1998

[10] Heedong Ko, Moon-Sang Kim, Hyun-Goo Park and Seung-Woo Kim, "Face sculpturing robot with recognition capability", Computer-Aided Design, pp 814-821, Nov. 1994, Vol 26, No 11

[11] 한국과학기술연구원, "인공현실감 시스템 개발", 과학기술처, 1994. 11

[12] 고희동 외, "가상현실감 시스템 운영체계에 관한 연구", 한국과학기술연구원, 1995. 4

[13] 고희동 외, "가상스튜디오 운영 및 저작 도구 개발", 한국과학기술연구원, 1998. 3

[14] 조준희, "실시간 차량 시뮬레이터의 개발", 국민대학교 대학원, 1997

[15] Iowa State University, "Driving Simulation", 1998

[16] Frank M. Cardullo, "Fundamentals of Simulation", State University of New York, 1998.1

[17] Dave Snowdon, Chris Greenhalgh, Steve Benford, Adroam Bullock and Chris Brown, "A Review of Distributed Architectures for Networked Virtual Reality", The University of Nottingham, pp21, 1996

[18] Grimsdale C., "dVS-distributed virtual environment system", Proc. Computer Graphics '91 Conference, London (ISBN 0 86353 282 9), 1991

[19] M. Teitel, "Reality built for two: A virtual reality toolkit." Computer Graphics 24(2):35-36, 1990

[20] Bricken W., "Virtual Environment Operating System: Preliminary Functional Architecture.", TR-HITL-M-90-2, Human Interface Technology Laboratory, University of Washington, Seattle, 1990

[21] Shaw C., Green M., Liang J., and Sun Y., "Decoupled simulation in virtual reality with the MR toolkit", ACM Transactions on Information Systems, 11(3), 287-317, 1993

[22] Sense 8 Co., "WorldToolKit Technical Brief, Version 1.01", 1992

[23] S. Kalawsky, "The science of virtual reality and virtual environments", Addison-Wesley, pp. 240-247, 1993

[24] Curtis Beeson, "An Object-Oriented Approach To VRML Development", Second Symposium on the Virtual Reality Modeling Language, page17-24, 1997

[25] "The Virtual Reality Modeling Language Specification", http://vrml.sgi.com/moving-worlds/spec/credits.html, August 4, 1996

[26] *IRIS Performer Programming Guide*, SGI

[27] *IRIS Performer Reference Pages*, SGI

[28] John K. Ousterhout, "Scripting: Higher Level Programming for the 21st Century", COMPUTER, IEEE Computer Society, Volume 31, Number Three, March 1998

[29] Harold Abelson and Gerald Jay Sussman with Julie Sussman, "Structure and Interpretation of Computer Programs", The MIT Press, 1995

[30] Samuel N. Kamin, "Programming Languages : An Interpreter-Based Approach", Addison-Wesley, May, 1990

[31] HRTF (Head-Related Transfer Function), available at : http://sound.media.mit.edu

[32] Russell L. Storms, "NPSNET-3D Sound Server : An Effective Use of the Auditory Channel", Master's thesis, Naval Postgraduate School, September, 1995

[33] 김승신, 유석종, 최윤철, "가상현실 환경에서 3차원 사운드 저작을 위한 사용자 인터페이스 설계 및 구현", 정보과학회 학술발표회 논문모음집, 1997

[34] Jed Hartman, Josie Wernecke, "The VRML 2.0 Handbook : building Moving Worlds on the Web", pp. 102-106, august, 1996

[35] RSX (Realistic Sound Experience), available at : http://www.intel.com

[36] "VRML and Audio", available at : http://lecaine.music.mcgill.ca/~breder/VRMLtalk.html

[37] 김승신, 유석종, 최윤철, 고희동, "가상현실 환경에서 입체음향 저작을 위한 사용자 인터페이스 설계 및 구현", 정보과학회 논문지 1998. 9

[38] J. Drosdol and F. Panik, "The Daimler-Benz Driving Simulator A Tool for Vehicle Development", SAE Paper 850334, 1985

[39] Woon-Sung Lee et al., "Development of a Driving Simulator", The 9th International Pacific Conference on Automotive Eng., Ball, Indonesia, Nov 16-21, pp. 13-18, 1997

[40] Michitaka Hirose. et al., "A Basic Study on the Presentation of the Sensation of Motion using Motion-Base", Transactions of the Virtual Reality Society of Japan, vol.1 No.1, 1996, pp. 16-22.

[41] Takashi Inoue. et al., "Development of a Stationary Driving Simulator", 日本 自動車技術論文集, vol.26 No2, April 1995, pp. 55-60

[42] Bunji Atsumi. et al., "Evaluation of Mental Work Load in Vehicle Driving by Analysis of Heart Rate Variability", Proceedings of the Human Factors and Ergonomics Society 37th Annual Meeting, Surface Transportation, vol.1, 1993, pp.574-578.

[43] S. Ellis, "What Are Virtual Environments?", IEEE Computer Graphics and Applications 14, No. 1, pp. 17-22, January 1994.

[44] C. Reynolds, Computer Animation with Scripts and Actors, Computer Graphics, Vol. 16, pages 289-296

[45] K. Perlin and A. Goldbery, Improv : A System for Scripting Interactive Behaviors in Virtual Worlds, ACM Computer Graphics, pages 205-216, 1996

[46] E. Gobetti and J. Baloguer, VB2 : A Framework for Interaction in Synthetic Worlds, Proc. UIST, pages 167-178, 1993

[47] K. C. Kang and G. I. Ko. PARTS : A Temporal Logic-Based Real-Time Software Specification and Verification Method. Proceedings of the 18th International Conference on Software Engineering `95, pages 169-176, Seattle, U.S.A., April 23-30, 1995

[48] D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, A. Shtull-Trauring and M. Trakhtenbrot. STATEMATE : A Working Environment for the Development of Complex Reactive Systems. IEEE Transactions on Software Engineering, Vol. 16, No. 4, pages 403-414, April 1990.

[49] B. W. Boehm. A Spiral Model of Software Development and Enhancement. IEEE computer, pages 61-72, May 1988.

[50] VRML Architecture Group(VAG), The Virtual Reality Modeling Language Specification Version 2.0. Aug, 1996. Available at <http://www.vrml.org/VRML2.0/FINAL/>

[51] W. Mitchell. Reasoning about Form and Function : Computability or Design. Y. Kalay (Editor), John wiley, 1987.

[52] Pimental, K. and Teixeira, K., "Virtual Reality", MCGraw Hill, 1993.

[53] Zeltzer, D., "Autonomy, interaction, and presence", Presence, 1(1), pp.127-132, 1992.

[54] Wilson J. R., "Virtual environments and ergonomics: needs and opportunities", Ergonomics, V.40 N.10, pp.1057-1077, 1997.

[55] Golledge, R. G., "Methods and Methodological issues in environmental cognition research", In Environmental knowing, (Golledge, R.G. and Moore, G.T. Eds.), Dowden, Hutchinson and Ross, Inc., Pennsylvania, pp.300-313, 1976.

[56] Witmer B. G., Singer M. J., "Measuring presence in virtual environments: A presence questionnaire", Presence, V.7 N.3, pp.225-240, 1998.

[57] Billinghurst, M. & Weghorst, S., "The Use of Sketch Maps to Measure Cognitive Maps of Virtual Environments". Proc. of the IEEE '95 Virtual Reality Annual International Symposium, 40-47, 1995.