제 2 단계
최종보고서

CG/VR 기술 개발

tudies on the Development of
CG/VR Technology

렌더링 시스템 개발

Development of
Rendering System

연 구 기 관
서 강 대 학 교

과 학 기 술 부

제 출 문

과학기술부 장관 귀하

본 보고서를 "CG/VR 기술개발 사업" (제1세부과제 "렌더링 시스템 개발에 관한 연구")의 2단계 최종 보고서로 제출합니다.

1998. 9. 15

주관연구기관명 : 한국첨단영상정보연구조합

세부과제 수행기관명 : 서 강 대 학 교

총괄연구책임자 : 임 인 성

연 구 원 : 박 상 훈, 이 래 경, 서 혜 경,
어 수 만, 정 원 영, 한 훈,
박 민 식, 임 권 혁, 정 효 성,
구 기 범, 박 지 현, 이 승 원,
조 성 희

위탁연구기관명 : 고 려 대 학 교

위탁연구책임자 : 이 희 웅

여 백

요 약 문

Ⅰ. 제목

렌더링 시스템 개발

Ⅱ. 연구개발의 목적 및 필요성

컴퓨터 그래픽스 기술의 발전은 컴퓨터 관련 분야 뿐만 아니라 광고나 영화와 같은 상업분야, 컴퓨터를 이용한 예술이나 교육과 같은 여러 분야에 걸쳐 많은 영향을 미치고 있다. 외국에서는 오랜기간 동안 컴퓨터 그래픽스의 기반 기술을 개발하여 고도의 관련 기술이 축적되어 있으며, 그 개발 속도가 빠른 속도로 가속화되고 있다. 이를 바탕으로 애니메이션 소프트웨어, CAD 소프트웨어 등이 개발되어 널리 사용되고 있으며, 최근에는 그러한 기술력을 바탕으로 가상 현실, 3차원 게임등과 같은 고부가 가치를 갖는 첨단 영상 관련 소프트웨어를 개발하고 있다. 반면에 국내에서는 컴퓨터 그래픽스 관련 소프트웨어의 개발을 위한 기술 축적이 부족한 실정이다. 특히, 컴퓨터 그래픽스 분야의 기반 기술중의 하나인 그래픽스 렌더링 툴킷 기술은 모든 그래픽스 소프트웨어 제작에 있어 중요한 핵심 기술로서 현실감있는 영상을 생성하기 위한 렌더링 기술과 일반 사용자들이 그러한 렌더링 기능을 쉽게 사용할 수 있도록 하여 주는 사용자 인터페이스 기술을 포함한다. 이 부분에 대한 세계적 기술은 빠른 속도로 발전하고 있지만, 국내에서는 주로 외국의 기술 도입에 의존하고 있는 형편이다. 따라서 방송, 광고, 영화, 게임, 가상 현실 등의 첨단 영상 소프트웨어 개발시 현실감 있는 영상을 생성하거나 사용자와의 효과적인 상호작용을 위한 기술을 개발하기 위해서는 렌더링 관련 기반 기술의 확충이 절

대적으로 요구된다. 이러한 시점에서 고급의 영상 생성기술의 개발을 위해서, 더 나아가서 관련 분야의 국제 경쟁력을 갖기 위해서는 그래픽스 렌더링 시스템을 자체적으로 개발하고 인력을 양성하므로써 컴퓨터 그래픽스의 기반 기술을 확충하는 것이 절대적으로 필요하다. 본 세부과제는 고급 렌더링 시스템의 개발 및 렌더링 관련 첨단 연구 수행을 통한 컴퓨터 그래픽스 관련 요소기술의 축적 및 첨단 영상분야의 고급 인력 양성을 최종 목표로 한다. 본 연구에서는 지난 1단계 2차년도에 설계한 고급 렌더링 시스템 프로토타입의 아키텍처의 기본 골격을 바탕으로 하여 하나의 시스템으로서 손색이 없는 고급 렌더링 시스템을 개발하려 한다. 2단계 2년간의 연구를 통하여 다양한 환경에서 유용하게 사용할 수 있는 고급 렌더링 시스템을 개발할 뿐만 아니라, 동시에 고급 렌더링 관련 첨단 연구를 병행함으로써 시스템 개발 및 고급 렌더링 기술을 축적하고 고급 인력을 양성하며, 향후 고급 에니메이션 및 가상 공간 시스템 등의 첨단 영상 분야의 소프트웨어 개발시 필요한 원천기술을 확보하려 한다. 또한 2단계 연구에서는 급변하는 소프트웨어 개발 환경에 적응하기 위하여 가상 환경 공유 및 실시간 렌더링 기법에 관한 연구를 추가하여 이에 대한 기술력을 축적하려 한다.

III. 연구개발의 내용 및 범위

2단계 연구개발의 내용 및 범위는 다음과 같이 요약할 수 있다.

1차년도 (`96)	▷ 고급 렌더링 시스템 프로토타입 구축에 관한 연구 · 미세다각형을 사용하는 Z-버퍼 방법에 기반을 둔 고급 렌더링 모듈의 개발 -렌더링 모듈 기능 요구 분석 및 설계 -기본 렌더링 기능의 구현 및 렌더링 시스템 GUI와의 연결 -고급 렌더링 기능의 구현 · 분산 및 병렬 렌더링 기법의 도입 -분산 렌더링 기법 관련 자료 조사 -통신 툴킷 PVM 및 영상공간분할 방법에 기반을 둔 분산 렌더링 기법 구현 · 렌더링 시스템 GUI 기능 개선 및 타 시스템과의 인터페이스 개발 -시스템 사용을 통한 평가 및 미비점 보완 -GUI 기능 보완 및 확장 -RIB 파일 입출력 모듈 구현 · 렌더링 시스템으로의 가상환경 개념의 도입 -기존의 분산 가상 환경 시스템 및 3D scene 공유 기법 분석 -상호 작용과 통신을 위한 자료 구조 분석 및 기존 구조 확장 -렌더링 시스템상에서의 통신 모델의 설계 및 구현 ▷ 렌더링 관련 첨단 기법 개발에 관한 연구 · 실시간 렌더링을 위한 영상기반 렌더링 기법 연구 -자료 조사 및 영상기반 렌더링 모델 연구 -영상 자료 압축을 통한 렌더링 기법 연구

2차년도 (`97)	▷ 고급 렌더링 시스템 프로토타입 구축에 관한 연구 · Z-버퍼 모듈에 대한 고급 기법 추가 -고급 렌더링 기능의 구현 -사용자에게 편의를 제공하는 렌더링 파라메터 설정 기법 연구 -Z-버퍼 모듈의 성능 향상 · 분산 및 병렬 렌더링 기법의 완성 -병렬 컴퓨터 Cray T3E으로의 렌더링 모듈 porting 및 병렬화 · 가상환경에서의 3D scene 공유를 통한 협력작업 기능의 완성 -3D scene 공유 및 협력작업 기법 연구 (일관성 유지, 다중 서버 구조 등) -가상환경 공유 관련 코드의 네트워크 툴킷화 -네트워크 툴킷을 이용한 렌더링 시스템의 확장 · 렌더링 시스템 평가를 통한 독립된 렌더링 시스템 프로토타입 구축 -VRML 2.0 파일 입출력 모듈 구현 -GUI 기능 보완 및 확장 -시스템 사용을 통한 평가 및 미비점 보완 -렌더링 시스템 소개를 위한 tutorial 제작 ▷ 렌더링 관련 첨단 기법 개발에 관한 연구 · 실시간 렌더링을 위한 영상기반 렌더링 기법 연구 -영상 자료 압축 기법 연구 · 방대한 볼륨 데이터의 효과적인 렌더링 기법 연구 -3D 웨이블릿을 이용한 방대한 의료 데이터의 압축 기법 연구 -압축기법을 이용한 볼륨 렌더링 기법 연구 -압축기법의 응용에 관한 연구

Ⅳ. 연구개발결과

지난 2년간의 연구수행 결과 다음과 같은 결과물을 산출하였다.

1차년도 ('96)	▷ 고급 렌더링 시스템 프로토타입 구축에 관한 연구 · Reyes 방법에 기반을 둔 렌더링 모듈 개발 -기능: geometric transformations, hidden-surface elimination, lighting and shading, rasterization, anti-aliasing based on stochastic samping, texture mapping -Graphics primitives: polygonal models, quadrics, parametric patches (일부) -광선 추적법 모듈과의 호환성 유지 -RenderMan Shading Language 기능 지원 -렌더링 시스템 GUI와의 연결 · 통신망상의 워크스테이션을 위한 분산 Z-버퍼 렌더링 기법 개발 · GUI 기능 개선 및 확장 -navigation 기능, texture 설정 기능, 광원 설정 기능, 기타 · RIB 파일 입출력 기능 개발 · 다중 사용자간의 3D scene 공유를 위한 기법 개발 및 렌더링 시스템으로의 적용 ▷ 렌더링 관련 첨단 기법 개발에 관한 연구 · 구를 기반으로 하는 영상기반렌더링 기법 개발 · 2D 웨이블릿을 이용한 light fields의 압축 기법 개발

2차년도 ('97)	▷ 고급 렌더링 시스템 프로토타입 구축에 관한 연구 • Z-버퍼 모듈에 대한 고급 기법 추가 -shadow, radiosity -렌더링 옵션 역분배 기법 개발 -Z-버퍼 모듈에 대한 최적화를 위한 성능 향상 • MIRAGE: 다중 사용자간의 협력작업을 지원하는 다중서버 구조 네트워크 툴킷 개발 • MIRAGE를 이용한 렌더링 시스템의 다중 사용자 시스템으로의 확장 • 병렬 컴퓨터 Cray T3E 상에서의 병렬 렌더링 기법 구현 • GUI 기능 개선 및 확장 -다중사용자 처리 기능, 기타 • VRML 2.0 파일 입출력 기능 추가 • 사용예 생성을 통한 렌더링 시스템의 간단한 tutorial 제작 ▷ 렌더링 관련 첨단 기법 개발에 관한 연구 • 3D 웨이블릿을 이용한 light fields의 압축 기법 개발 • 빠른 복원을 지원하는 3D 웨이블릿을 이용한 3D 의료 데이터의 압축 기법 개발 • 압축된 데이터로부터의 빠른 렌더링 기법 개발 • Virtual human body navigation system 개발으로의 응용

본 연구에서는 일반적인 상황을 가정한 렌더링 시스템의 프로토타입을 개발하였다. 본 과제의 수행을 통하여 축적한 요소기술들은 추후 가상현실 소프트웨어, 3D CAD 시스템, 애니메이션 소프트웨어, 실시간 3D 의료영상 소프트웨어 등 관련 소프트웨어 제작시 렌더링 모듈 개발에 유용하게 쓰일 수가 있을 것이다. 예를 들어 VRML 기반 가상환경 시스템 제작에 있어 본 연구에서 개발한 프로토타입 시스템의 코드를 적절히 사용하므로써, 적은 비용으로 렌더링 관련 부분을 개발할 수 있을 것이다. 또한 본 연구에서 개발한 웨이블릿을 이용한 방대한 의료 데이터의 압축 기법은 빠른 런-타임 복원을 지원하는 새로운 개념의 압축 기법으로서, 가상의 3차원 인체 네비게이션 시스템등 의료분야의 소프트웨어 개발에 유용하게 쓰일 수 있을 것이다. 3단계 연구가 수행될 경우 연구 목표로 설정될 구체적인 상황에 맞는 렌더링 소프트웨어로 발전될 수 있을 것이다. 렌더링 관련 기술은 컴퓨터 그래픽스 분야에서 가장 핵심적인 기술중의 하나로서 과거에는 주로 외국의 시스템을 도입하여 사용하는 형편이었다. 본 연구에서는 이러한 고급 렌더링 기법을 실제로 구현하고 또한 GUI를 개발하여 봄으로써, 이와 관련하여 상당한 요소 기술을 축적할 수 있었다. 또한 기존의 기법의 개발뿐만 아니라 렌더링 관련 연구를 수행하므로써 상당한 역량을 축적할 수 있었으며, 고급 인력을 양성할 수 있었다. 추후 한 단계 발전된 수준의 개발 및 연구를 통하여 세계적인 개발 역량을 보유할 수 있게 될 것이며, 이는 첨단 영상 관련 소프트웨어 제작 분야의 발전에 기여를 할 수 있을 것이다.

여 백

S U M M A R Y

I. Title

Development of Rendering System.

II. Objectives and Significance of the Research

Advances in Computer Graphics technologies have greatly affected a variety of fields, such as entertainment, commercials, arts and education, as well as computer-related areas. As the results of active efforts for research and development, high technologies in computer graphics have been accumulated in the leading countries. Based on such technologies, they have produced commercial products such as animation software and CAD/CAM software, and have recently developed high value-added products for virtual reality, 3D games, etc. On the other hand, while the technologies are being developed rapidly worldwide, it is true that they, in Korea, rely on foreign technologies at high costs. This research, in its second stage ('96.12 to '98.9), is the first part of the three sub-projects of the six-year long project "The Development of Technologies for CG/VR" of STEP 2000. In particular, this research focuses on rendering toolkits techniques, one of the most important fundamentals in computer graphics, that include such techniques as photo-realistic rendering and user-friendly interfaces. In this research, we aim at two parallel objectives: development of a prototype of advanced rendering system, and advanced research in rendering technologies. During the period of one year and nine months, we develop a general-purpose rendering system with effectively designed user-interface. As well, we carry out advanced research related to graphics rendering. These research efforts will let us acquire high technologies in graphics software production

and rendering techniques, produce competent experts, and apply such techniques to building graphics software, like advanced animation and virtual reality software.

III. Scopes and Contents of the Research

The scopes and contents of this research are summarized in the following table.

1st year ('96)	▷ Research on development of an advance rendering system prototype • Development of a rendering library based on Z-buffer and micropolygons -Requirement analysis and design of rendering module -Implementation of fundamental functions and connection to GUI -Implementation of advanced functions • Introduction of distributed and parallel rendering methods -Literature search on distributed rendering -Implementation of distributed rendering using the network toolkit PVM and image-space partition • Improvement of GUI functions and development of data I/O interfaces -Performance test -Improvement and extension of GUI functions -Implementation of an I/O module for the RIB file format • Introduction of concepts of virtual environment into the rendering system -Analysis of previous virtual environment systems and 3D scene sharing techniques -Analysis of data structures for mutual interaction and communication, and extension of the rendering system structure -Design and implementation of a network model on the rendering system ▷ Research on advance techniques for rendering • Research on image-based rendering for real-time rendering -Literature search on image-based rendering models -Research on image data compression

2nd year (`97)	▷ Research on development of an advance rendering system prototype • Addition of advance functions to the Z-buffer rendering module -Implementation of advance rendering functions -Research on user-friendly rendering parameter selection techniques -Performance improvement of the Z-buffer module -Porting and parallelization of the rendering module onto the parallel computer Cray T3E • Implementation of CSCW through 3D scene sharing in virtual environments -Research on 3D scene sharing and collaboration techniques (consistency, multi-server structure, etc) -Building network toolkits for virtual environment sharing -Extension of the rendering system using the network toolkit • Building a rendering system prototype and performance test -Implementation of an I/O module for the VRML 2.0 file format -Extension of GUI functions -Test and debugging through the use of system -Production of an introductory tutorial ▷ Research on advance techniques for rendering • Research on image-based rendering for real-time rendering -Design of image data compression techniques • Research on effective rendering of very large volume data -Compression of very large volume data using 3D wavelets -Development of volume rendering techniques using compression schemes -Research on applications of compression techniques

IV. Research Results

1st year ('96)	▷ Research on development of an advance rendering system prototype • Development of a rendering module based on the Reyes architecture -Functions: geometric transformations, hidden-surface elimination, lighting and shading, rasterization, anti-aliasing based on stochastic sampling, texture mapping -Graphics primitives: polygonal models, quadrics, parametric patches -Compatibility with the ray-tracing module -Support of RenderMan Shading Language -Connection to the rendering system GUI • Development of distributed Z-buffer rendering on networked workstations • Improvement and extension of GUI functions -Functions: navigation, texture parameter selection, light source selection, etc. • Development of an I/O module for the RIB file format • Development of new schemes for multi-user 3D scene sharing and application to the rendering system ▷ Research on advance techniques for rendering • Development of an image-based rendering method based on spheres • Development of a compression technique for light fields using 2D wavelets

2nd year (`97)	▷ Research on development of an advance rendering system prototype • Addition of advanced functions to the Z-buffer module -Shadow, radiosity -Development of a reverse selection technique for rendering parameters -Optimization and performance enhancement of the Z-buffer module • Development of MIRAGE: a multi-server-based network toolkit for multi-user collaboration • Building a multi-user rendering system using MIRAGE • Implementation of parallel rendering on the parallel computer Cray T3E • Improvement and extension of GUI functions -Functions: multi-user interaction, etc • Development of an I/O module for the VRML 2.0 file format • Production of an introductory tutorial for the rendering system using examples ▷ Research on advance techniques for rendering • Development of a compression technique for light fields using 3D wavelets • Development of a compression technique for 3D medical data using 3D wavelets supporting fast reconstruction • Development of a fast rendering method for compressed data • Application to development of a virtual human body navigation system

V. Future Plans

In this research, we have built a general-purpose rendering system prototype. The know-how, acquired through the research, will be very useful in building a rendering module for a variety of software, such as virtual reality software, 3D CAD/CAM software, animation software, and real-time 3D medical imaging systems. For example, when they develop a VRML-based virtual environment system, they can produce software at the less expense by using the related codes, generated in the research. Also, the wavelet-based compression technique supporting fast reconstruction for very large medical data can be effectively used in building a 3D virtual human body navigation system. When the third-stage project goes on, we plan to select a specific application area and apply our research results for effective software development. Rendering techniques, one of the most important parts, have been usually imported from foreign countries. In this research, we have made a great deal of efforts in implementing various rendering-related techniques ourselves, and could accumulate a fair amount of know-how on rendering and GUI technologies. In the future, we plan to continue our research and development efforts, and it will let us contribute to the growth of graphics-software production industry.

C O N T E N T S

목 차

제 1 장 서론

컴퓨터 그래픽스 기술의 발전은 컴퓨터 관련 분야 뿐 만 아니라 광고나 영화와 같은 상업분야, 컴퓨터를 이용한 예술이나 교육과 같은 여러 분야에 걸쳐 많은 영향을 미치고 있다. 외국에서는 오랜기간 동안 컴퓨터 그래픽스의 기반 기술을 개발하여 고도의 관련 기술을 축적하여 왔으며, 그 개발 속도가 빠른 속도로 가속화되고 있다. 이를 바탕으로 애니메이션 소프트웨어, CAD 소프트웨어 등이 개발되어 널리 사용되고 있으며, 최근에는 그러한 기술력을 바탕으로 가상 현실, 3차원 게임등과 같은 고부가 가치를 갖는 첨단 영상 관련 소프트웨어를 개발하고 있다. 반면에 국내에서는 컴퓨터 그래픽스 관련 소프트웨어의 개발을 위한 기술 축적이 비교적 부족한 실정이다. 특히, 컴퓨터 그래픽스 분야의 기반 기술중의 하나인 그래픽스 렌더링 툴킷 기술은 모든 그래픽스 소프트웨어 제작에 있어 중요한 핵심 기술로서 현실감있는 영상을 생성하기 위한 렌더링 기술과 일반 사용자들이 쉽게 그러한 렌더링 기능을 쉽게 사용할 수 있도록 하여주는 사용자 인터페이스 기술을 포함한다. 이 부분에 대한 세계적 기술은 빠른 속도로 발전하고 있지만, 국내에서는 주로 외국의 기술 도입에 의존하고 있는 형편이다. 따라서 방송, 광고, 영화, 게임, 가상 현실 등의 첨단 영상 소프트웨어 개발시 현실감 있는 영상을 생성하거나 사용자와의 상호작용을 위한 기술을 개발하기 위해서는 렌더링 관련 기반 기술의 확충이 절대적으로 요구된다. 이러한 시점에서 고급의 영상 생성기술의 개발을 위해서, 더 나아가서 관련 분야의 국제 경쟁력을 갖기 위해서는 그래픽스 렌더링 시스템을 자체적으로 개발하고 인력을 양성하므로써 컴퓨터 그래픽스의 기반 기술을 확충하는 것이 절대적으로 필요하다.

본 세부과제에서는 고급 렌더링 시스템의 개발 및 렌더링 관련 첨단 연

구 수행을 통한 컴퓨터 그래픽스 관련 요소기술의 축적 및 첨단 영상분야의 고급 인력 양성을 최종 목표로 연구 및 개발을 하여왔다. 본 연구에서는 지난 1단계 2차년도에 설계한 고급 렌더링 시스템 프로토타입의 아키텍쳐의 기본 골격을 바탕으로 하여 하나의 시스템으로서 손색이 없는 고급 렌더링 시스템을 개발하여 왔다. 2단계 2년간의 연구를 통하여 다양한 환경에서 유용하게 사용할 수 있는 고급 렌더링 시스템을 개발할 뿐만 아니라, 동시에 고급 렌더링 관련 첨단 연구을 병행함으로써 시스템 개발 및 고급 렌더링 기술 축적하고 고급 인력을 양성하며, 향후 고급 에니메이션 및 가상 공간 시스템 등의 첨단 영상 분야의 소프트웨어 개발시 필요한 원천기술을 확보하려 하였다.

특히 2단계 연구에서는 급변하는 소프트웨어 개발 환경에 적응하기 위하여 가상 환경 공유 및 실시간 렌더링 기법에 관한 연구를 추가하여 이에 대한 기술력을 축적하여 왔다. 가상 환경이란 사용자들이 가상의 세상에 몰입하여 마치 실제의 세상에서 행하는 바와 같이 그 안에 존재하는 물체나 다른 사람들과의 상호 작용 및 공동의 작업을 통하여 원하는 목표를 달성케 하여 주는 실시간 시뮬레이션 시스템을 말한다. 전 세계적으로 가상 환경 구축과 관련하여 활발한 연구가 진행되어 왔으며 현재 건축, 게임, 오락, 교육, 의료, 군사 등을 비롯한 다양한 분야에서 이러한 가상 환경 시스템을 구현하고 사용하고 있다. 고속의 통신망의 출현에 따른 통신망 속도의 향상은 통신망을 통하여 연결된 여러 대의 컴퓨터에 접속한 다수의 사용자들이 참여하여 상호 작용하는 분산 처리에 기반을 둔 효과적인 다중 사용자 가상 환경 시스템의 구현을 가능케 하여 주고 있다. 따라서 이러한 소프트웨어 개발 환경의 변화에 대비하기 위하여 가상 환경 공유에 관한 연구를 수행하고 이를 다중 사용자가 통신망을 통하여 공유하고 있는 렌더링 시스템의 장면을 동시에 편집할 수 있도록 응용하였

다. 또한 3D 게임등과 같이 실시간 렌더링 기술이 절대적으로 필요한 분야에 적용하기 위하여 영상 기반 렌더링 기법에 관한 연구를 새로이 추가하여 수행하여 왔다.

2단계 2년간(1996. 12. 4. ~ 1998. 9. 3.)의 연구개발의 최종목표, 내용 및 범위는 다음과 같이 요약할 수 있다.

<table>
<tr><td colspan="2">연구개발 최종 목표</td></tr>
<tr><td colspan="2">▷고급 렌더링 시스템 프로토타입의 개발
▷첨단 렌더링 연구를 통한 기술 확보
▷첨단 영상 소프트웨어 개발을 위한 가상 환경 공유 및 실시간 렌더링 기술의 연구
▷첨단 영상 분야의 고급 인력 양성</td></tr>
<tr><td colspan="2">연구개발 내용 및 범위</td></tr>
<tr><td>1차년도
(`96)</td><td>▷ 고급 렌더링 시스템 프로토타입 구축에 관한 연구
· 미세다각형을 사용하는 Z-버퍼 방법에 기반을 둔 고급 렌더링 모듈의 개발
· 분산 및 병렬 렌더링 기법의 도입
· 렌더링 시스템 GUI 기능 개선 및 타 시스템과의 인터페이스 개발
· 렌더링 시스템으로의 가상환경 개념의 도입
▷ 렌더링 관련 첨단 기법 개발에 관한 연구
· 실시간 렌더링을 위한 영상기반 렌더링 기법 연구</td></tr>
<tr><td>2차년도
(`97)</td><td>▷ 고급 렌더링 시스템 프로토타입 구축에 관한 연구
· Z-버퍼 모듈에 대한 고급 기법 추가
· 분산 및 병렬 렌더링 기법의 완성
· 가상환경에서의 3D scene 공유를 통한 협력작업 기능의 완성
· 렌더링 시스템 평가를 통한 독립된 렌더링 시스템 프로토타입 구축
▷ 렌더링 관련 첨단 기법 개발에 관한 연구
· 실시간 렌더링을 위한 영상기반 렌더링 기법 연구
· 방대한 볼륨 데이터의 효과적인 렌더링 기법 연구</td></tr>
</table>

여 백

제 2 장 국내외 기술개발 현황

미국등지의 외국에서는 지난 수십년간의 연구개발 노력의 결과로서 상당한 수준의 컴퓨터 그래픽스 렌더링 및 사용자 인터페이스 기술을 보유하고 있다. 이러한 기술력은 MAYA, SoftImage 3D, RenderMan, 3D Studio Max등과 같이 널리 쓰이고 있는 통합 애니메이션 소프트웨어 뿐만 아니라 AutoCAD, FormZ와 같은 CAD/CAM 소프트웨어등 여러 응용 소프트웨어들에서 사실적인 영상을 생성하기 위한 렌러링 툴킷을 제작하는데 핵심적인 역할을 하고 있다. 이러한 고가의 소프트웨어들은 외국은 물론 국내에서도 방송, 광고, 영화, 건축등 다양한 분야에서 널리 이용되고 있다. 과거에는 그러한 소프트웨어들을 사용하려면 고성능의 그래픽스 가속기를 부착한 고가의 워크스테이션이나 메인 프래임급의 컴퓨터가 필요하였다. 한편 PC의 성능향상 및 PC용 그래픽스 가속기 가격의 급격한 하락으로 인하여, 이러한 스프트웨어들을 사용한 작업이 점차 보편화 되고 있다. 이는 관련 소프트웨어 시장이 소수의 전문가들에서 쉽게 PC를 접할수 있는 방대한 규모의 일반 사용자들으로 옮겨감을 의미하며, 실리콘 그래픽스 및 마이크로 소프트등과 같은 관련 기업들이 이에 대하여 막대한 투자를 하고 있다.

국내에서는 위와 같은 렌더링 및 렌더링 관련 사용자 인터페이스의 개발이 비교적 미미한 상태였고, 주로 고가의 외국 기술 및 소프트웨어를 구입하여 사용하는 형편이었다. 이의 결과로서 첨단 영상 기술의 외국에 대한 종속이라는 결과를 낳게 되었는데, 대부분의 경우 외국에서 개발한 소프트웨어를 사용하여 프로그램을 개발하는 단계에 있으므로 그래픽스 라이브러리 또는 렌더링 기술들의 축적이 많이 되어 있지 않은 실정이다. 이러한 상황에서 고가의 로열티를 지불하여 기술 도입을 한다 하더라도

핵심 기술은 이전받을 수가 없을 뿐만 아니라, 기술의 급속한 발전으로 구입된 기술을 완전히 습득하기 전에 낙후된 기술로 전락할 가능성이 농후하기 때문에 이를 충분히 활용할 수 있는지가 불확실 하다. 또한 대부분의 고급 그래픽스 기술들은 외국의 경우에서도 현재 개발중에 있는 것이 대부분이므로 이러한 기술의 도입은 현실의 여건으로는 매우 어렵다. 따라서 렌더링 및 그래픽스 사용자 환경 구축에 대한 지속적인 투자를 통한 독자적인 기술의 개발 및 확보는 필수적이라 할 수가 있다. 다행히도 컴퓨터 그래픽스의 중요성을 인식하여 과학기술부에서는 1994년부터 특정연구개발사업의 하나로서 STEP2000과제를 통하여 컴퓨터 그래픽스 분야의 기술개발에 투자를 하여왔다. 특히 최근 몇 년동안 관련분야의 연구진이 많이 확충되어 국내에서도 컴퓨터 그래픽스 관련 연구가 활발하게 진행되리라 예상된다.

제 3 장 연구개발 수행 내용 및 결과

지난 2년간 수행한 연구개발의 내용 및 범위는 다음과 같이 요약할 수 있다.

1차년도 (`96)	▷ 고급 렌더링 시스템 프로토타입 구축에 관한 연구 · 미세다각형을 사용하는 Z-버퍼 방법에 기반을 둔 고급 렌더링 모듈의 개발 -렌더링 모듈 기능 요구 분석 및 설계 -기본 렌더링 기능의 구현 및 렌더링 시스템 GUI와의 연결 -고급 렌더링 기능의 구현 · 분산 및 병렬 렌더링 기법의 도입 -분산 렌더링 기법 관련 자료 조사 -통신 툴킷 PVM 및 영상공간분할 방법에 기반을 둔 분산 렌더링 기법 구현 · 렌더링 시스템 GUI 기능 개선 및 타 시스템과의 인터페이스 개발 -시스템 사용을 통한 평가 및 미비점 보완 -GUI 기능 보완 및 확장 -RIB 파일 입출력 모듈 구현 · 렌더링 시스템으로의 가상환경 개념의 도입 -기존의 분산 가상 환경 시스템 및 3D scene 공유 기법 분석 -상호 작용과 통신을 위한 자료 구조 분석 및 기존 구조 확장 -렌더링 시스템상에서의 통신 모델의 설계 및 구현 ▷ 렌더링 관련 첨단 기법 개발에 관한 연구 · 실시간 렌더링을 위한 영상기반 렌더링 기법 연구 -자료 조사 및 영상기반 렌더링 모델 연구 -영상 자료 압축을 통한 렌더링 기법 연구

2차년도 (`97)	▷ 고급 렌더링 시스템 프로토타입 구축에 관한 연구 · Z-버퍼 모듈에 대한 고급 기법 추가 -고급 렌더링 기능의 구현 -사용자에게 편의를 제공하는 렌더링 파라메터 설정 기법 연구 -Z-버퍼 모듈의 성능 향상 · 분산 및 병렬 렌더링 기법의 완성 -병렬 컴퓨터 Cray T3E으로의 렌더링 모듈 porting 및 병렬화 · 가상환경에서의 3D scene 공유를 통한 협력작업 기능의 완성 -3D scene 공유 및 협력작업 기법 연구 (일관성 유지, 다중 서버 구조 등) -가상환경 공유 관련 코드의 네트워크 툴킷화 -네트워크 툴킷을 이용한 렌더링 시스템의 확장 · 렌더링 시스템 평가를 통한 독립된 렌더링 시스템 프로토타입 구축 -VRML 2.0 파일 입출력 모듈 구현 -GUI 기능 보완 및 확장 -시스템 사용을 통한 평가 및 미비점 보완 -렌더링 시스템 소개를 위한 tutorial 제작 ▷ 렌더링 관련 첨단 기법 개발에 관한 연구 · 실시간 렌더링을 위한 영상기반 렌더링 기법 연구 -영상 자료 압축 기법 연구 · 방대한 볼륨 데이터의 효과적인 렌더링 기법 연구 -3D 웨이블릿을 이용한 방대한 의료 데이터의 압축 기법 연구 -압축기법을 이용한 볼륨 렌더링 기법 연구 -압축기법의 응용에 관한 연구

지난 2년간 수행한 연구개발의 결과는 다음과 같이 요약할 수 있다.

1차년도 ('96)	▷ 고급 렌더링 시스템 프로토타입 구축에 관한 연구 • Reyes 방법에 기반을 둔 렌더링 모듈 개발 -기능: geometric transformations, hidden-surface elimination, lighting and shading, rasterization, anti-aliasing based on stochastic samping, texture mapping -Graphics primitives: polygonal models, quadrics, parametric patches (일부) -광선 추적법 모듈과의 호환성 유지 -RenderMan Shading Language 기능 지원 -렌더링 시스템 GUI와의 연결 • 통신망상의 워크스테이션을 위한 분산 Z-버퍼 렌더링 기법 개발 • GUI 기능 개선 및 확장 -navigation 기능, texture 설정 기능, 광원 설정 기능, 기타 • RIB 파일 입출력 기능 개발 • 다중 사용자간의 3D scene 공유를 위한 기법 개발 및 렌더링 시스템으로의 적용 ▷ 렌더링 관련 첨단 기법 개발에 관한 연구 • 구를 기반으로 하는 영상기반렌더링 기법 개발 • 2D 웨이블릿을 이용한 light fields의 압축 기법 개발

2차년도 ('97)	▷ 고급 렌더링 시스템 프로토타입 구축에 관한 연구 • Z-버퍼 모듈에 대한 고급 기법 추가 -shadow, radiosity -렌더링 옵션 역분배 기법 개발 -Z-버퍼 모듈에 대한 최적화를 위한 성능 향상 • MIRAGE: 다중 사용자간의 협력작업을 지원하는 다중서버 구조 네트워크 툴킷 개발 • MIRAGE를 이용한 렌더링 시스템의 다중 사용자 시스템으로의 확장 • 병렬 컴퓨터 Cray T3E 상에서의 병렬 렌더링 기법 구현 • GUI 기능 개선 및 확장 -다중사용자 처리 기능, 기타 • VRML 2.0 파일 입출력 기능 추가 • 사용예 생성을 통한 렌더링 시스템의 간단한 tutorial 제작 ▷ 렌더링 관련 첨단 기법 개발에 관한 연구 • 3D 웨이블릿을 이용한 light fields의 압축 기법 개발 • 빠른 복원을 지원하는 3D 웨이블릿을 이용한 3D 의료 데이터의 압축 기법 개발 • 압축된 데이터로부터의 빠른 렌더링 기법 개발 • Virtual human body navigation system 개발으로의 응용

제 1 절 Reyes 구조에 기반을 둔 Z 버퍼 렌더링 모듈의 개발

1. Reyes구조 알고리즘

Reyes는 1981년 Loren Carpenter가 제작한 렌더링 시스템으로서, 빠른 속도로 고화질의 고해상도 이미지를 만들어 내기 위해 고안되었다. 실제로 Reyes 구조에 기반을 둔 Z 버퍼 렌더러는 광선추적법(Ray Tracing)에 비해 속도면에서 월등한 성능을 보여주며 화질 또한 광선추적법에 견줄 만 하다. 하지만 그 방법상의 한계로 인해 은면제거(hidden surface removal)와 그림자 효과(shadow effect), 그리고 실제와 같은 빛의 반사와 굴절에 의한 효과 등을 직관적인 방법으로 지원하지 못하기 때문에 텍스쳐 맵핑(texture mapping)[1-15], 환경 맵핑(environment mapping)[1-16] 등의 여러 가지 맵핑 기법을 이용하여 시뮬레이션 한다.

Reyes구조 알고리즘(그림 1.1)은 각각의 모델링 객체들을 적절한 형태로 읽어서 카메라 공간에서 경계상자(bounding box)를 이용하여 해당 모델이 렌더링 대상인지, 카메라의 시야 밖에 위치하기 때문에 렌더링 과정에서 제외시킬 것인지 여부를 결정한다. 그 다음 렌더링 대상이 되는 객체에 대해 다이싱 검사(dicing test)를 수행한다. 이것은 해당 물체를 구성하는 다각형들이 다이싱을 적용할 수 있을 만큼 충분히 작은가를 검사하는 과정이다. 일정크기 이상의 다각형은 다이싱 과정에서 많은 메모리를 필요로 하기 때문에 다이싱 검사는 필수적이다. 이 때 다이싱이 불가능한 객체에 대해서는 스플리팅을 해준다. 스플리팅(splitting)이란 하나의 다각형을 그 형태와 속성이 변하지 않는 범위에서 여러 개의 다각형으로 쪼개는 과정이다. 스플리팅 된 다각형들은 다시 바운딩 상자 검사 단계부터 시작하여 계속해서 렌더링 대상인지의 검사와 다이싱 검사

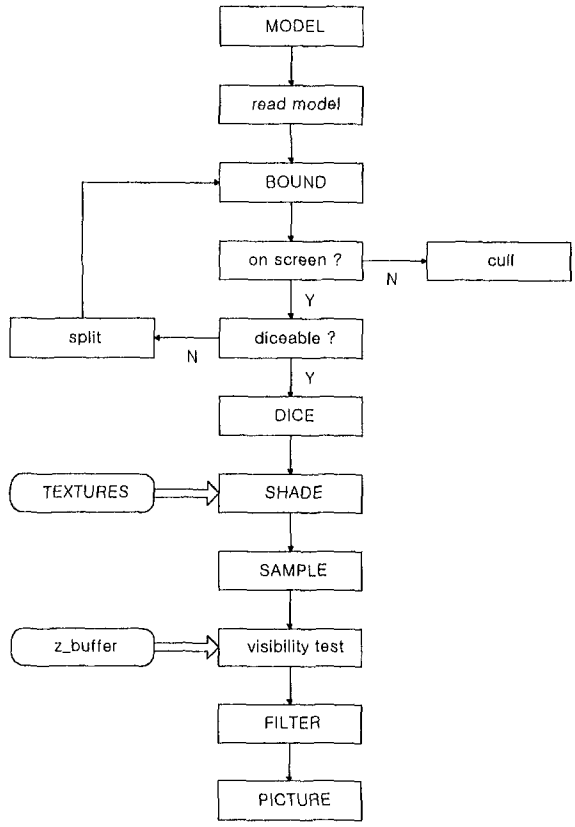


그림 1.1 Reyes 구조 알고리즘의 순서도

를 수행하게 된다. 즉 이들 세 개의 프로세스는 재귀적인 방법으로 구현될 수 있다.

세 단계의 과정을 통과한 객체들에 대해서는 다이싱 과정을 수행하게 된다. 다이싱(dicing)이란 다각형을 미세 다각형(micropolygon)으로 나누는 작업이다. 미세 다각형으로 나누어진 후 해당 미세 다각형의 격자점에서 세이딩이 일어나고 샘플링되어 색상 값이 결정되기 때문에 다이싱은 Reyes구조 알고리즘에서 가장 핵심적인 과정이다. 이 과정에서 생성된 미세 다각형의 수에 따라 이후의 세이딩과 샘플링 등의 작업에 필요한 시간과 최종 이미지의 화질이 많은 영향을 받는다.

이렇게 미세 다각형으로 나누어진 후 미세 다각형의 각 격자점에 대해 세이

딩이 수행된다. 세이딩(shading)이란 해당 물체의 픽셀에 대한 색상 값을 얻어주는 작업이다[1-17]. 이 때에는 현재 렌더링 시킬 장면에 포함된 모든 광원과 물체의 특성, 텍스쳐 등을 고려해서 색상 값을 결정한다. 미세 다각형이 많을수록 세이딩 되는 점의 수가 많아지기 때문에 더 긴 렌더링 시간을 필요로 하게 되지만 이미지의 질은 좋아진다.

모든 격자점에 대한 세이딩 작업이 끝나면 각 미세 다각형에 대해 스크린 공간(screen space)에서의 바운딩 상자를 만들고 해당 픽셀(pixel)의 각 샘플링 점에 대한 샘플링 작업이 수행된다. 샘플링(sampling)이란 이미지의 각 픽셀에 위치한 샘플링 점에서의 색상 값을 얻어오는 작업을 말한다. 미세 다각형의 각 격자점이 실제로 각 픽셀의 샘플링 점과 일치하지 않기 때문에 2차 선형 보간법(bilinear interpolation)[1-3]을 사용하여 픽셀의 샘플링 점에서의 색상 값을 미세 다각형의 각 격자점의 색상 값으로부터 얻어오게 된다.

보통 하나의 픽셀에 대해서 가로 세로 두 개씩 4개의 샘플링 점을 할당하는데 이 때문에 픽셀의 최종 색상 값을 얻기 위해서는 필터링 작업을 거쳐야 한다. 필터링(filtering)이란 1개 이상의 색상 값을 특정한 필터 함수(filtering function)에 통과시켜 하나의 색상 값으로 묶어내는 작업인데 보통 가우시안 필터 함수(Gaussian filter function)를 사용한다. 그러나 샘플링된 색상 값이 모두 필터링 과정에 필요한 것은 아니다. 일단 샘플링된 값이라 하더라도 그 부분이 다른 물체에 의해 가려져 있어 화면상에서 보이지 않는다면 필터링 과정을 거칠 필요가 없다. 이러한 비교 작업을 가시성 검사(visibility test)라 한다. 가시성 검사는 해당 샘플링 점의 Z값, 즉 카메라로부터 떨어진 거리 값과 Z 버퍼 내에서의 해당 픽셀의 Z값을 비교함으로써 이루어진다[1-4]. 샘플링된 점이 Z 버퍼에 들어있는 픽셀보다 카메라로부터 더 멀리 떨어져 있으면 해당 샘플링 점의 모든 정보는 필요가 없게 되며 필터링 과정을 거치지 않는다. 이렇게 해서 필터링 과정까지 살아남은 샘플링 점의 Z값은 전체 Z 버퍼내의 해

당 픽셀의 Z값으로 갱신되어 이후의 렌더링 작업을 위한 자료로 남게 된다.

스플리팅과 다이싱은 Reyes구조 방법에서 가장 중요한 부분이다. 이 두 과정은 물체의 데이터 형태마다 적용 방법이 다른데 다각형의 경우 더욱 프로그램하기 어렵다. 왜냐하면 다각형 데이터는 일정한 형태를 취하지 않으며 사용되는 점의 개수와 나열된 방향과 모양을 알 수 없기 때문이다. 이러한 다각형에 대한 스플리팅과 다이싱은 실제 프로그램 방법에 따라서 다양한 방법으로 구현될 수 있다.

스플리팅은 다각형이 다이싱하기에 너무 크거나 입실론 평면(epsilon plane)에 걸려 있는 경우에 수행된다. 입실론 평면은 이론적으로 카메라 공간에서 양의 Z축 방향으로 원점보다 조금 앞에 있는 평면이다. 하나의 다각형을 스플리팅하기 위해서는 일단 다각형의 중심을 구하고 이 중심과 각 변의 중점을 연결한다. 이러한 스플리팅은 물체 데이터의 위상(topology)을 보존하기 위해 카메라 공간에서 행해진다(그림 1.2).

다이싱은 스플리팅보다 복잡한 과정을 갖는다. 다이싱의 목적은 세이딩을 일반적으로 적용시킬 미세 다각형의 생성인데, 일반적인 경우 해당 다각형의 바운딩 상자를 여러 개의 미세 다각형으로 나누는 작업으로 대치된다. 하지만 이 경우 세이딩 시에 해당 세이딩 점이 다각형에 포함되는지의 여부를 확인해야 하는 과정을 거치는 등의 문제가 있다. 따라서 본 구현에서는 바운딩 상자가 아닌 다각형 자체를 나누는 방법을 이용하였다. 이 방법은 경우에 따라서 위의 방법보다 많은 시간 비용을 요구하지만 더 좋은 결과 이미지를 만들어낼 수 있다(그림 1.3).

그림 1.2에서 보다시피 먼저 다각형의 각 변을 선형 보간법(linear interpolation)을 이용하여 나눈 뒤 나머지 중간 부분의 미세 다각형 격자점을 선형 보간법으로 구성하는 알고리즘을 이용하였다. 그림에서 보이는 검정 사각형 격자는 각 변을 선형 보간법으로 나눈 모양을 보여주고 있다. 여기서 사용

된 격자(grid)의 수는 실제 스크린 공간에서 물체가 갖는 크기를 추측하여 결정한다. 이 것은 나중에 렌더링된 이미지의 질을 결정짓는 가장 큰 요소로 작용한다. 만일 Nyquist 샘플링을 만족시키고자 할 경우 격자의 수는 스크린 공간의 바운딩 상자가 갖는 픽셀의 수의 4배로 결정된다. 이러한 격자의 크기는 실제 사용자가 이미지의 화질과 렌더링 시간의 기회 비용(trade off)을 고려하여 결정하게 된다[1-1].

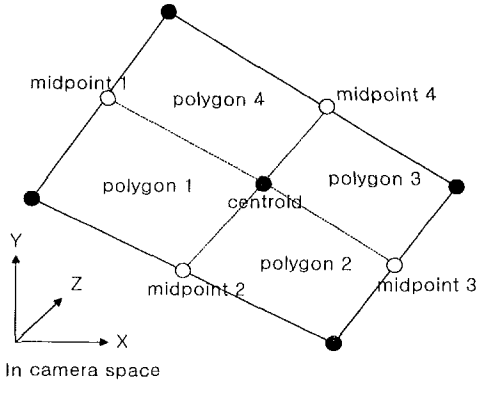


그림 1.2 스플리팅 과정

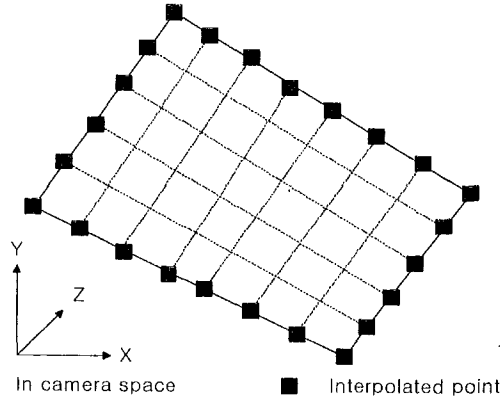


그림 1.3 다이싱 과정

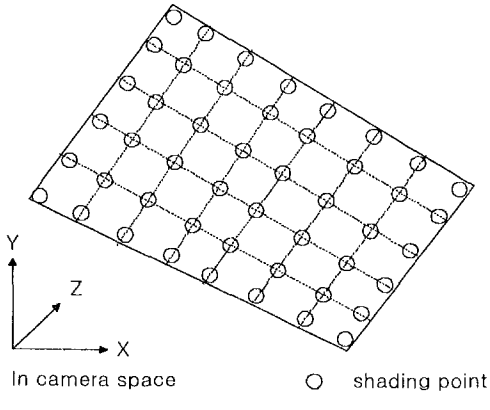


그림 1.4 미세 다각형의 세이딩

2. 세이딩, 샘플링 및 합성

세이딩은 미세 다각형의 격자점에서 이루어진다. 모든 미세 다각형의 격자점이 다각형에 포함되어 있으므로 순차적으로 세이딩을 수행하면 된다. 이로써 위에서 설명한 바와 같이 바운딩 상자를 이용한 다이싱 방법에 비해 세이딩 질면에서 더 나은 성능을 발휘할 수 있게 된다. 물론 다이싱 과정은 다소 오래 걸리겠지만 세이딩 단계에서는 해당 격자점이 다각형 안쪽에 위치하는가의 여부를 검사할 필요가 없게 된다[1-3].

그림 1.4는 미세 다각형의 세이딩을 보여주고 있다. 그림에서 원은 세이딩 점을 나타낸다. 그림 1.4에서 보다시피 다각형의 각 변 위에 위치한 세이딩 점은 약간씩 다각형의 안쪽으로 치우쳐져 있는데 이것은 실제 세이딩 시에 부동 소수점 연산의 오류 발생 가능성을 줄이기 위함이다. 실제로 각 변에 위치한 세이딩 점을 그대로 세이딩에 이용하면 부동 소수점 연산의 수치적인 오류로 인해 해당 점의 법선벡터나 기타 세이딩에 필요한 파라미터들의 계산에서 오류가 발생할 수 있으며 세이딩 결과가 잘못될 가능성이 커진다. 본 구현에서는 각 변에 위치한 세이딩 점을 다음 세이딩 점의 위치에 20% 가깝도록 설정했다. 또한 물체 데이터가 너무 작거나 물체가 멀리 떨어져 있어 스크린 공간에서 큰 영역을 차지하지 못한다면 다이싱 결과 물체가 하나의 미세 다각형도 갖지 못하게 되는 문제가 발생할 수 있다. 본 구현에서는 하나의 다각형이 최소 2 × 2의 미세 다각형을 가지게끔 하여 이 문제를 근원적으로 해결하였다.

Reyes구조 방법에서는 샘플링 점을 만들기 위해 스토캐스틱 샘플링 기법을 사용한다. 스토캐스틱 방법은 샘플링될 영역을 일정한 크기로 나눈 다음 각 영역의 중심에서 임의로 조금 벗어난 위치를 샘플링 점으로 정하는 방법이다. 조금 벗어난 위치를 결정하기 위해서는 지터(jitter)를 사용한다[1-5]. 지터는 임의로 만들어진 값으로 의미 없이 무작위로 선정된다. 그러나 샘플링 점이 극

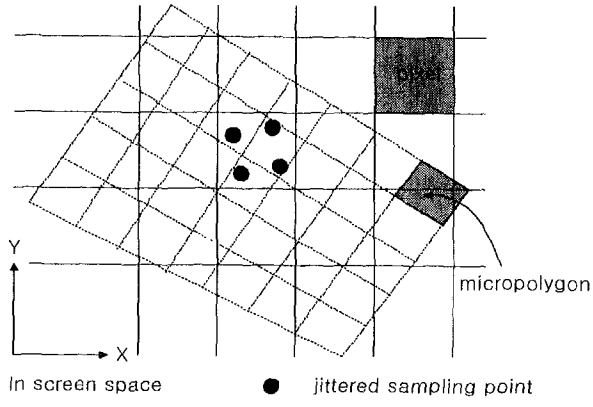


그림 1.5 샘플링

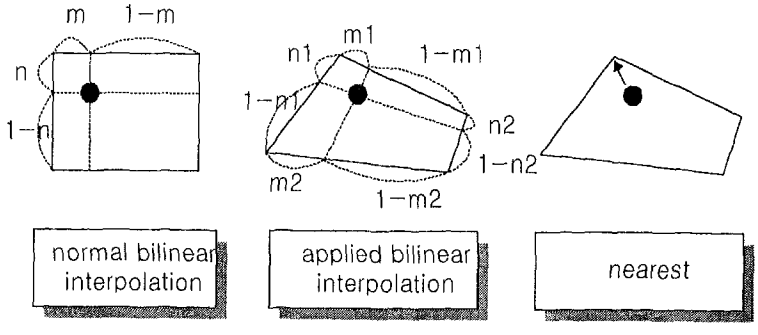


그림 1.6 샘플링 방법

단적으로 치우치는 경우를 막기 위해서 그 범위에 제한을 둔다[1-3]. 그림 1.5는 미세 다각형이 스크린 공간에서 샘플링되는 과정을 나타낸다.

셰이딩된 미세 다각형들은 하나씩 샘플링된다. 미세 다각형은 스크린 공간으로 바운딩되고 이 바운딩 상자 안에 들어오는 샘플링 점(sampling point)들에 대해서 다음의 방법 등을 이용하여 색상 값과 투명도, 그리고 Z값을 얻는다.

샘플링 작업에서 흔히 사용하는 알고리즘에는 2차 선형 보간법이 있지만 여기에서는 기본적인 공식을 그대로 적용할 수 없다. 왜냐하면 2차 선형 보간법을 이용하려면 샘플링 점의 위치가 사각형 내에서 가로 세로로 각각 얼마만큼의 비율로 떨어져 있는지를 알아야 하는데 이 경우에는 각 샘플링 점의 스크린

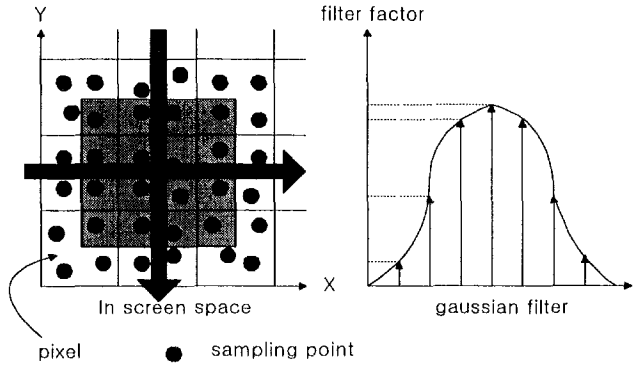


그림 1.7 가우시안 필터 함수를 이용한 이미지 합성

공간상에서의 위치정보만을 알 수 있으며 일반적으로 2차 선형 보간법을 적용시킬 미세다각형이 직사각형의 형태가 아니기 때문이다. 따라서 본 구현에서는 이 2차 선형 보간법을 응용하여 문제를 해결하였다. 그림 1.6에 응용된 2차 선형 보간법을 나타내었다.

제일 왼쪽 그림은 일반적인 2차 선형 보간법을 나타낸다. 중간 그림은 이를 응용하여 만든 보간법으로써 가로, 세로 변들의 기울기로 평균치를 구해서 만들어진 직선과 양 두 변의 교점을 구한 다음 m1, m2, n1, n2 등의 값을 구해낸다. 이렇게 함으로써 현재 점이 다각형 내에서 어느 정도의 비율로 각 꼭지점에 대해 떨어져 있는지를 알 수 있다. 오른쪽 그림은 이러한 2차 선형 보간법을 이용하지 않고 가장 가까이에 위치한 꼭지점의 데이터를 그대로 사용하는 방법을 나타낸 것으로 시간 비용 측면에서 가장 유리한 방법이지만 미세 다각형의 조밀도에 따라서 이미지의 화질을 많이 떨어뜨릴 위험이 있는 방법이다.

이렇게 샘플링된 색상 값은 최종적으로 필터 함수를 사용하여 합성된다. 필터 함수에는 삼각형(triangle) 함수, 상자(box) 함수, 캣멀롬(Catmull-Rom) 함수, 가우시안(Gaussian) 함수 등이 있다[1]. 본 구현에서는 가우시안 필터 함수를 이용하여 색상을 합성한다. 그림 1.7은 샘플링 점들이 어떤 식으로 합성될 것인지를 보여준다.

그림 1.7에서는 한 픽셀당 샘플링 점의 수를 2 × 2로 설정한 예를 보이고 있다. 한 픽셀이 4개의 샘플링 점을 가질 경우에는 이웃하는 픽셀의 샘플링 점까지 모두 16개의 샘플링 점을 가우시안 필터 함수를 이용하여 합성하게 된다.

$$filter\ factor = e^{-2*distance}$$

위의 식은 본 구현에서 사용한 가우시안 필터 함수의 식이다. 가우시안 필터 요소(filter factor)는 픽셀의 중심으로부터 샘플링 점까지의 거리가 늘어남에 따라 값이 줄어드는 형태를 취한다.

3. 투명도 처리

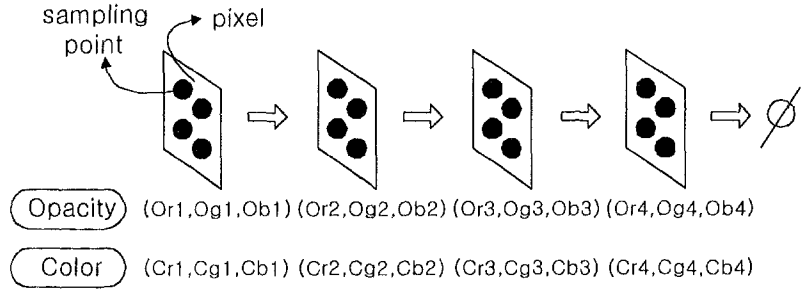


그림 1.8 투명도 처리를 위한 자료구조

Reyes구조 알고리즘은 최대한 간단한 렌더링만을 구현하고 있기 때문에 여러 가지 추가적인 기능의 지원을 위해서는 그 구조의 변경이 불가피하다. 투명도(opacity)를 처리하기 위해서는 각 샘플링 점에서의 Z값과 Z 버퍼의 해당픽셀의 Z 값을 비교하여 카메라 공간에서 해당 물체 앞에 다른 물체가 존재한다 할지라도 무조건 그 값을 버려서는 안된다. 바로 앞의 물체가 투명한 물체일 경우 최종 색상 값에는 뒷물체의 색상 값도 참조되어야 하기 때문이다[1-9]. 또한 렌더링 과정에 들어오는 물체의 앞뒤 순서는 임의적이므로, 투명도를 처리하기 위해서는 유연한 메모리 관리가 필요하다.

본 구현에서는 연결 리스트(linked list)를 이용하여 임의의 순서로 들어오는 투명성을 지닌 샘플링 점들을 관리한다. 해당 샘플링 점에 하나 이상의 투명성을 지닌 데이터가 들어오게 되면 그 샘플링 점에 대한 연결 리스트가 활성화 된다. 리스트가 활성화되면 그 다음부터 중간 중간에 들어오는 데이터들을 메모리의 동적 할당을 통해 관리하게 된다. 그림 1.8은 이러한 연결 리스트의 구조를 나타낸다.

그림 1.8에서 나타낸 바와 같이 연결 리스트의 각 노드에는 투명도와 색상 값이 OPACITY, RED, GREEN, BLUE의 형태로 저장되어 있다. 예를 들어 RED에 대한 최종 색상 값을 구하기 위해서는 다음과 같은 계산이 필요하다[1-1, 1-10].

Last_R = (A * Cr1) + (B * Cr2) + (C * Cr3) + (D * Cr4)

A = Or1 * (1.0)

B = Or2 * (1.0 - (A))

C = Or3 * (1.0 - (A + B))

D = Or4 * (1.0 - (A + B + C))

따라서 하나의 샘플링 점에 대한 최종 색상 값을 구하기 위해서는 위와 같은 계산을 R, G, B 채널에 대해 수행해야 한다.

4. 2차 곡면의 지원

광선 추적법에서는 알고리즘의 특성으로 인해 2차 곡면 데이터를 쉽게 처리할 수 있다. 하지만 Reyes구조에 기반을 둔 Z 버퍼 렌더러에서는 광선 추적법과는 달리 2차 곡면 쉽게 처리할 수 있는 메커니즘이 제공되지 않는다. Reyes 구조 알고리즘의 핵심 이론이 바로 각 물체를 미세 다각형으로 잘게 쪼개는 것

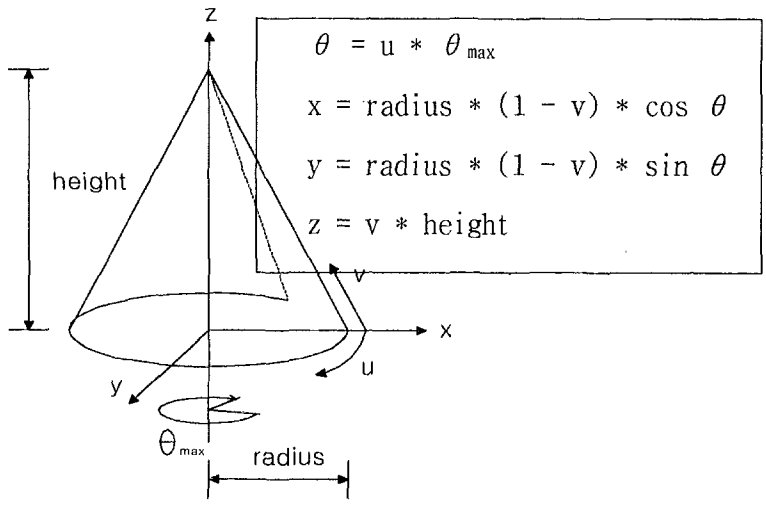


그림 1.9 원뿔 2차 곡면의 구성

이라 하였는데, 이것이 바로 그 근본 원인이다. 수식으로 제공된 2차 곡면을 미세 다각형으로 직접 쪼개는 작업은 비효율적이기 쉽다. 따라서 본 구현에서는 우선 2차 곡면을 다각형화하여 렌더링의 나머지 과정은 일반 다각형과 같은 형식으로 처리하도록 했다[1-3,1-6].

예를 들어, 원뿔을 3차원 곡면으로 나타내기 위한 수식은 그림의 오른쪽 상자에서 보이는 바와 같다. 이 때 u와 v를 0에서 1사이로 적절히 이동시키면서 생성되는 점들을 연결하여 삼각형으로 다각형화시킨다. 그리고 0에서 1사이의 값을 얼마나 조밀하게 이동시키는 가에 따라서 해당 2차 곡면을 얼마나 정확하게 다각형화시킬 것인가를 결정할 수 있다. 그러나 이 조밀도가 높을수록 많은 수의 다각형이 생성되기 때문에 최종 이미지의 화질은 좋아지겠지만 렌더링 속도는 느려지게 된다.

5. 그림자 Z 버퍼의 구현

최종 이미지를 더욱 실사와 같게 하려면 그림자 효과(shadow effect)의 지원은 필수적이다. 광선 추적법에서는 자연스럽게 그림자 효과를 얻어낼 수 있지

만[1-13], Reyes구조 알고리즘을 이용할 경우 그림자 효과는 부가적인 알고리즘을 이용해야 구현이 가능하다. Reyes구조 알고리즘에서는 그림자 효과의 지원을 위해 보통 그림자 다각형(shadow polygon) 방법, 그림자 체적(shadow volume) 방법, 그리고 그림자 Z 버퍼(shadow Z buffer) 방법 중 하나를 이용한다[1-5]. 본 구현에서는 그림자 Z 버퍼 방법을 이용했다.

그림자 Z 버퍼 방법은 먼저 각 광원에 대해 하나씩의 버퍼를 할당하고 각 다각형 데이터에 대해 광원으로부터의 거리를 저장한 후, 렌더링을 할 때 각 광원에 대해 해당 Z 버퍼를 참조하여 실제 렌더링 중인 샘플링 점에 그림자가 드리우는지를 확인하는 방법이다. 이 방법을 이용하기 위해서는 각 광원에 대해 하나씩의 그림자 Z 버퍼를 구성해야 한다. 그림자 Z버퍼는 광원의 종류에 따라서 그 구성 형태가 달라진다. 예를 들어, 방향 광원(directional light source)일 경우에는 빛이 전경으로부터 무한히 멀리 떨어져 있는 것으로 가정하기 때문에 그림자 Z 버퍼는 하나의 2차원 버퍼만으로도 충분하다. 하지만 나머지 점 광원(point light source)이나 스팟 광원(spot light source) 또는 면적 광원(area light source)의 경우는 사방으로 빛이 발산되므로 광원 주위로 상자나 구 형태의 그림자 Z 버퍼를 드리우는 형태로 버퍼가 구성되어야만 한다. 실제 본 구현에서는 이들 광원에 대해 정육면체를 광원 주위에 드리운 형태로 그림자 Z 버퍼를 구현하였다.

그림 1.10에서 보듯이, 각 다각형 데이터는 광원쪽으로 투사되는데 이 때 정육면체의 어느 면에 교점이 위치하는 지를 확인한 후 교점을 계산한다[1-5]. 그림자 Z 버퍼에는 이들 교점들을 이용해서 해당 위치에 광원으로부터 다각형이 떨어진 거리가 저장된다. 하나의 다각형이 여러 면의 그림자 Z 버퍼에 투사되면 면들의 교점은 2차 선형 보간법을 이용하여 구한다.

그림자 Z 버퍼는 실제 구현에서 다양한 크기를 가질 수 있다. 일반적으로 최종 이미지의 가로 세로 픽셀 개수와 같은 크기로 버퍼를 할당받는데, 버퍼의

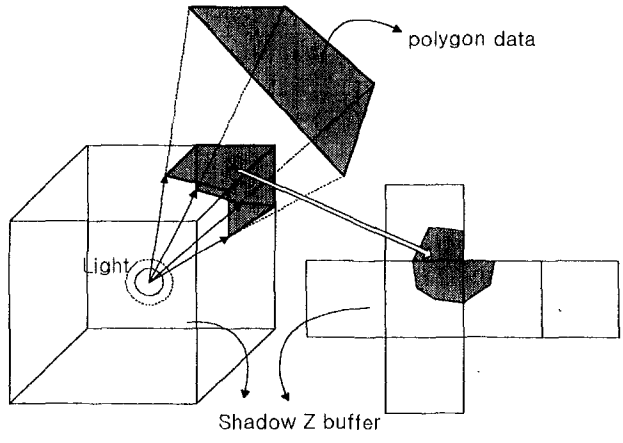


그림 1.10 그림자 Z 버퍼

해상도를 높일수록 그림자의 화질이 좋아지게 된다. 하지만 메모리의 효율적인 사용과 렌더링 시간의 문제로 인해 대부분 최종 이미지의 가로 세로 픽셀 개수의 두 배 정도의 크기로 할당한다.

Z 값은 항상 광원으로부터 가장 가까운 다각형 데이터와의 거리로 갱신되는데 이 때 계산된 하나의 Z값을 그대로 저장하면 실제 렌더링 작업에서 그림자 Z 버퍼의 값을 샘플링할 때 Moir패턴이 생길 수 있다[1-7]. Moir패턴이란 실제 그림자가 드리우지 않는 부분에 자기 자신의 데이터가 스스로 자신에게 그림자를 지게 하는 데서 생기는 알리아싱 현상을 말한다.

그림 1.11에서 보면 광원에서 가장 가까운 다각형 표면이 위에 위치하고 그 밑에 두 번째로 가까운 다각형이 존재한다. Moir 패턴은 엄밀히 말해서 그림자 Z 버퍼의 해상도가 실제 세이딩이 일어나는 미세 다각형의 해상도보다 낮기 때문에 일어나는 현상이다. 즉 그림자 Z 버퍼에서 하나의 샘플링 점은 실제로 한 개 이상의 세이딩 점을 내포하고 있는 것이다. 이 때문에 원칙적으로는 달라야 할 여러 개의 세이딩 점에서의 광원까지의 거리가 같아지게 되어 그중 광원까지의 거리가 그림자 Z 버퍼 내에서의 해당 Z 값보다 먼 샘플링 점에 그림자가 드리우게 되는 것이다. 이러한 현상을 막기 위한 방법에는 일단 그림자 Z 버퍼

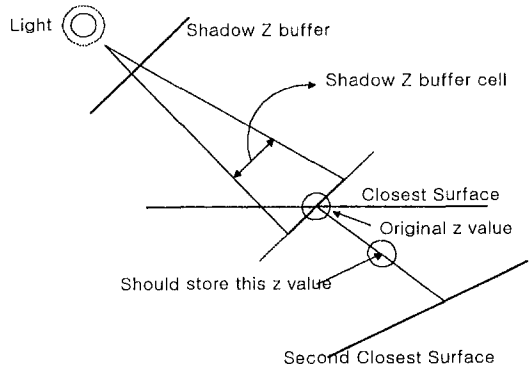


그림 1.11 Moire 패턴의 생성 이유

는 그대로 만든 후, 각 샘플링 점에서 그림자 Z 버퍼 내에서의 해당 Z 값과 비교를 할 때 인위적으로 비교적 작은 수를 더한 후에 비교를 하는 방법이 있다. 하지만 이 방법은 시간 비용은 적지만 모든 상황에서 정확한 계산이 이루어진다는 보장이 없다. 따라서 본 구현에서는 위 그림에서 보였듯이 두 번째로 가까운 다각형까지의 거리와 가장 가까운 다각형까지의 거리의 평균값을 그림자 Z 버퍼에 저장하는 방식으로 이 문제를 해결하였다.

6. 그림자 Z 버퍼의 샘플링 및 필터링

이렇게 만들어진 그림자 Z 버퍼는 최종적으로 샘플링과 필터링을 통해 그림자 효과의 지원에 이용된다. 그림자 Z 버퍼의 크기가 커질수록 알리아싱이 줄어들긴 하나, 보통의 경우 그림자의 알리아싱을 막을 수 있을 만큼의 충분한 크기로 만들기 어렵다[1-14]. 시간비용은 대체로 그림자 Z 버퍼의 크기 증가비율만큼 커지지는 않지만 메모리 요구량은 심각하게 늘어난다. 점 광원의 경우 그림자 Z 버퍼의 크기를 512 × 512로 하고 하나의 픽셀당 4 바이트 크기를 갖는 부동 소수점 데이터 타입을 사용할 경우 한 면당 1 Mbyte의 저장공간을 필요로 하므로 전체 6 Mbyte의 공간이 필요하다. 그리고 Moir 패턴 문제를 근본

적으로 제거하기 위해서는 여벌의 메모리 공간이 필요하므로 하나의 점 광원에 대해 전체 12 Mbyte의 메모리 공간이 소요된다. 하지만 그 크기를 1024 × 1024로 늘릴 경우 소요되는 메모리의 양은 48 Mbyte로 늘어나게 된다. 이와 같이 그림자 효과의 알리아싱 제거를 위해 그림자 Z 버퍼의 크기를 늘리는 방법은 메모리의 과다 사용으로 인해 그다지 효과적이지 못하다. 실제로 그림자 Z 버퍼의 크기는 최종 이미지 해상도의 가로 세로 2배씩 정도의 크기가 최종 이미지에 미치는 효과나 메모리의 사용 등을 고려할 때 가장 최적이다.

그림자 효과를 나타냄에 있어서 알리아싱을 줄이기 위해서는 특별한 필터링 방법을 사용하여야 한다. 기존의 텍스쳐 지원에 사용되는 필터링 방법으로는 하나의 샘플링 단위에 대해 그림자가 드리우는지 또는 아닌지에 대한 정보만을 알 수 있을 뿐이므로, 그림자가 지는 경계부분에서는 심한 알리아싱이 일어날 수밖에 없다. 따라서 근접 확률 필터링(percentage closer filtering) 방법을 사용하여 하나의 샘플링 단위에 대해 어느 정도의 확률로 그림자가 지는지를 알아낸다[1-8].

필터링을 하기 위해서는 샘플링이 선행되어야 한다. 그리고 그림자 Z 버퍼에서 샘플링을 하기 위해서는 먼저 물체 데이터의 미세 다각형 수준에서의 일정 영역을 그림자 Z 버퍼 내로 맵핑시키는 작업이 필요하다. 그림 1.12의 A는 이 과정을 설명하고 있다. 그림에 나타내었듯이 실제 세이딩이 일어나는 각 미세 다각형의 격자점들이 그림자 Z 버퍼 내로 맵핑 된다. 이 때 알리아싱을 최소한으로 줄이려면 일정 크기의 영역을 그림자 Z 버퍼로 맵핑시켜서 보다 많은 샘플링을 해야만 한다. 기본적으로 하나의 격자점은 해당 점 주위로 하나의 미세 다각형 크기만큼의 영역에 대한 모든 책임을 지게끔 되어있다. 즉 각 미세 다각형의 격자점들은 자기 주변의 영역에 대한 모든 세이딩과 투명도, 그리고 그림자 효과에 따른 처리 등을 책임져야 한다. 이러한 관점에서 하나의 미세 다각형 격자점은 자신을 포함한 주변의 영역을 그림자 Z 버퍼로 맵핑하여 샘플링

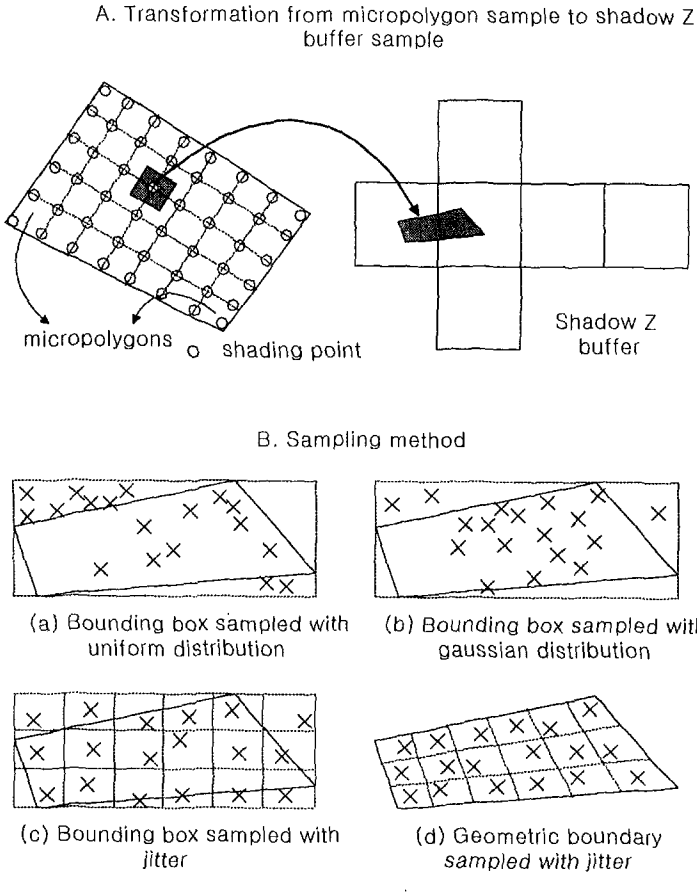


그림 1.12 그림자 Z 버퍼의 샘플링

영역을 만들게 된다. 맵핑시켜야 할 주변 영역의 크기는 달리 정해줄 수 있다. 단지 하나의 미세 다각형 크기 만큼일 경우에는 그림자 주변에서 부드러운 그림자 효과를 낼 수 없게 된다. 따라서 대부분 가로 세로 2배씩 정도의 영역을 그림자 Z 버퍼로 맵핑시키게 된다.

그림 1.12의 B는 샘플링의 방법을 나타낸다. 위의 방법으로 만들어진 샘플링 영역에 대해 샘플들을 어떤 식으로 가져오는가에 대한 방법에는 대체로 4가지가 있으며 다음과 같다.

(a) 해당 샘플링 영역의 바운딩 상자에 대해 일정 개수의 샘플들을 임의로 고르는 방법

(b) 해당 샘플링 영역의 바운딩 상자에 대해 가우시안과 같은 특정 분산 함수를 이용하여 샘플들을 고르는 방법

(c) 바운딩 상자를 몇 개의 영역으로 잘게 나눈 다음 해당 영역에서 지터(jitter)를 이용하여 샘플들을 고르는 방법

(d) 샘플링 영역을 내부적으로 몇 개의 영역으로 잘게 나눈 다음 해당 영역에서 지터(jitter)를 이용하여 샘플들을 고르는 방법

본 구현에서는 (c)방법을 이용하여 샘플링 작업을 수행한다. (c)방법은 (a), (b)에 비해 보다 정확한 값을 얻을 수 있으며 (d)방법에 비해서는 보다 적은 시간 비용을 사용하게 된다. 먼저 바운딩 상자를 구해서 논리적으로 일정한 수의 영역으로 나눈 다음 임의의 지터를 가해서 샘플링을 한다. 이 때 샘플링 점의 개수는 대개 1에서 16사이이다.

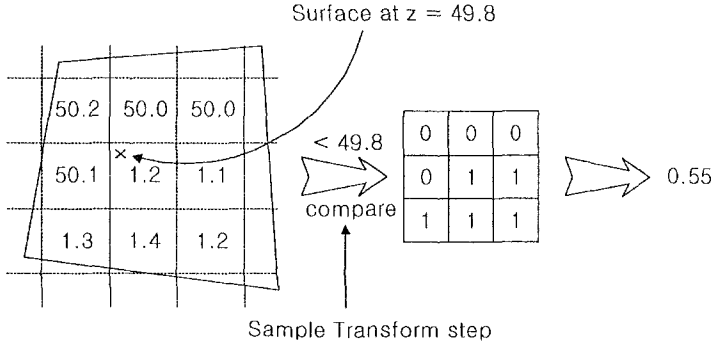


그림 1.13 근접 확률 필터링

이렇게 샘플링된 값들은 필터링 작업을 통해 최종적으로 그림자가 해당 미세 다각형에 얼마나 걸치는지 확인한다. 이 때 사용되는 필터링 방법이 근접 확률 필터링 방법이다[1-8]. 그림 1.13은 근접 확률 필터링 방법을 보여준다.

종래의 텍스쳐 맵(texture map) 필터링에서와 같이 샘플링 점을 하나로 하여 그 점의 Z 값과 비교하여 미세 다각형의 격자점에서 가져온 Z값이 크면 그림자가 지고 그렇지 않으면 그림자가 지지 않는 식으로 처리하면 그림자의 가장자

리에서 심한 알리아싱이 일어날 수밖에 없다. 하지만 모든 샘플링 점에서의 Z 값과 현재 미세 다각형의 격자점에서 가져온 Z값을 비교하여 확률 테이블을 그림 1.12와 같이 구성한 다음 얼마만큼의 확률로 그림자가 지는 지를 구함으로써 그림자의 가장자리에서 알리아싱을 제거할 수 있게 된다. 그림의 제일 오른쪽에 표시된 0.55 값은 최종적으로 그림자가 지는 확률을 나타낸다.

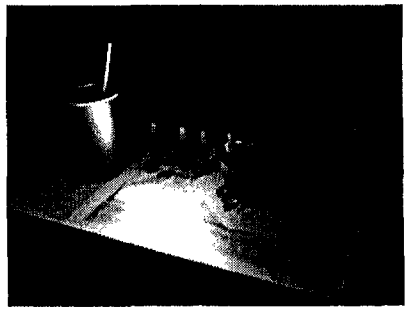


그림 1.15 Z 버퍼 렌더링 한
영상 (예제 1)

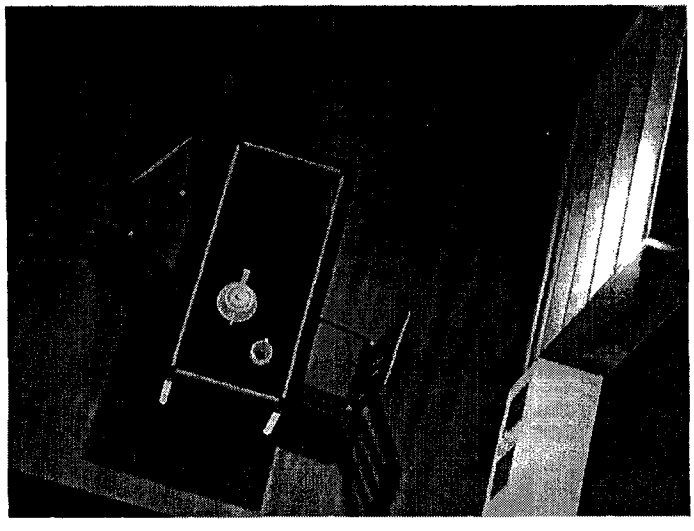


그림 1.14 Z 버퍼 렌더링 한 영상 (예제 2)

그림 1.14와 그림 1.15는 Z 버퍼 렌더링 한 영상의 예를 보여주고 있다.

7. 분산 렌더링

가. 분산 렌더링

고화질의 영상을 생성하는 일은 상당한 양의 부동 소수점 계산을 요구한다. 특히 여러 프레임으로 구성된 영화를 만들기 위해서는 방대한 양의 계산을 수행하여야 하기 때문에 대부분의 상용 소프트웨어는 네트워크에 연결된 다수의 워크스테이션을 묶어 렌더링 계산을 할 수 있는 기능을 제공한다. 예를 들어, 영화 Toy Story의 경우 1526x922해상도를 갖는 11만장의 프레임을 렌더링 하기 위하여 각 화소당 평균 약 50만번의 부동소수점 연산을 수행해야 했고, 실제로 294대의 SUN SPARCstation으로 구성된 "Sun-farm"이라 불린 워크스테이션 그룹에 의하여 렌더링이 이루어 졌다[1-21].

본 시스템에도 보다 빠른 렌더링 계산을 위해 분산처리 기능이 구현되었다[1-22]. 현재는 하나의 영상을 렌더링하기 위하여 필요한 계산을 작은 크기의 태스크로 나누어 사용 가능한 워크스테이션에서 계산을 하도록 구현되어 있다. 태스크로 분할하는 방법은 크게 물체 공간에서 기하 물체를 분할하는 방법과 이미지를 분할하는 두 가지 방법이 있다[1-24]. 본 연구에서는 이미지 분할 방법을 선택하였는데 각 태스크를 32×32, 64×64, 128×128, 256×256 등과 같은 적절한 크기로 나눌 수 있도록 하였다. 분산 렌더링 기능의 개발을 위하여 이기종의 컴퓨터들을 연결하여 병렬 프로그램을 개발할 수 있는 환경을 제공하는 응용 툴킷인 PVM 3.2 (Parallel Virtual Machine)[1-23]를 사용하였으며, 전형적인 마스터-슬레이브 구조를 이용하였다. 마스터 프로세서에서는 전체 이미지 영역을 여러 개의 태스크 영역으로 분할한 후 각 테스크 영역 정보와 물체와 그래픽스 상태를 사용 가능한 슬레이브 프로세서에 보내고 슬레이브 프로세서에서 보내온 결과를 합성하여 이미지를 생성한다. 슬레이브 프로세서에서는 마스터 프로세서에서 받은 데이터를 유지하면서 정해진 스크린 영역을 렌더

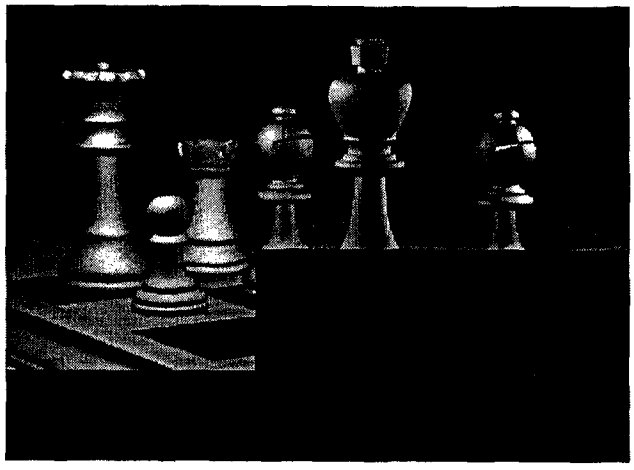


그림 1.16 분산환경에서 렌더링되고 있는 부분 영상

링하고 결과를 보낸다. 그림 1.16은 분산환경에서 렌더링 되고 있는 영상을 보여주고 있다.

나. 분산처리의 실험

표 1.1 분산처리 환경

기종	indigo2	indy	indy	indigo2	ULTRA1
CPU	R4000	R4400	R4400	R4400	UltraSPARC
CPU clock	100Mhz	150Mhz	200Mhz	200Mhz	167Mhz
memory	64MB	32MB	96MB	128MB	128MB
SPECint92	59	91.7	119	119	250
SPECfp92	61	97.5	131	131	350

분산 렌더링을 위해 사용된 기계들의 내역이 표 1.1에 수록되었다. 렌더링에 사용된 입력 데이터는 노말 벡터(normal vector)와 텍스쳐 좌표를 가지는 14,084개의 삼각형과 2개의 광원과 4개의 세이딩 모델을 사용한다. 이 실험이 갖는 특성은 하나의 이미지를 만드는 것이 목적이 아니라 애니메이션을 위한 방대한 양의 이미지를 만드는데 있다. 분산 처리의 성능을 비교하기 위해서 5개의 기계로 이루어진 그룹의 결과(그림 1.18)를 중간 성능을 갖는 하나의 기

계에서 만들어진 결과(그림 1.17)와 비교하였다.

실험내용은 5대의 기계를 하나의 그룹으로 하였을 때 640×480과 1280×1024의 해상도에서 처리해야 하는 이미지를 32×32, 64×64, 128×128, 256×256의 영역 크기로 변화시키면서 Nyquist 샘플링을 만족시킬 때와 그렇지 않을 때로 구분하여 렌더링 시간을 조사하였다.

실험 결과를 살펴보면 분산 처리를 통하여 1대의 기계에서 보다 약 3.3배의 빠른 렌더링을 수행하였다. Nyquist 샘플링을 할 경우 실험에서 영역의 크기가 작아질수록 Nyquist를 맞추지 않았을 때와 비슷한 시간 비용을 보인다. 이것은 전체 렌더링 시간 중 Nyquist 샘플링에 의한 영향보다 영역의 크기에 따른 영

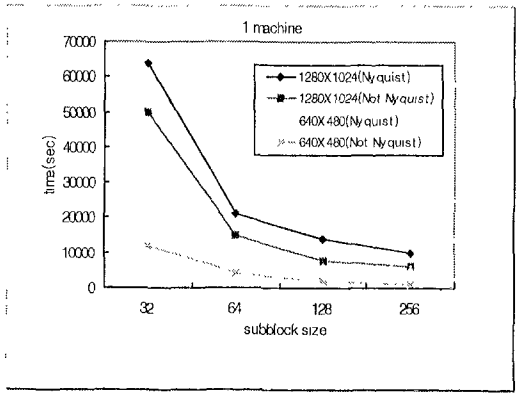


그림 1.17 기계 1대에서의 실험 결과

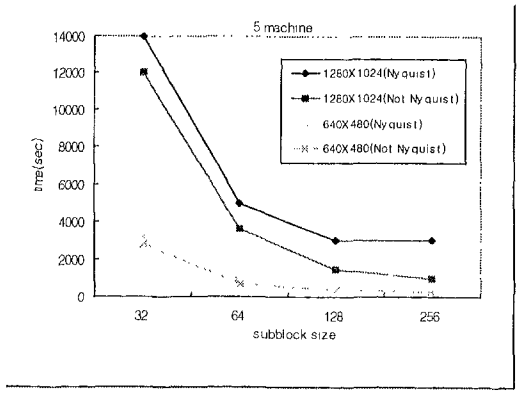


그림 1.18 기계 5대에서의 실험 결과

향이 더 크다는 것을 의미한다. 영역의 크기가 작아 많은 영역이 생성되면 하나의 물체가 여러 영역에 겹치게 된다. 그러면 하나의 물체에 대해서 중복된 계산을 수행하게 된다. 그렇다고 영역의 크기를 너무 크게 잡는다면 분산 처리의 효과는 줄어들게 된다. 영역의 수를 사용 가능한 기계 수로 결정하고 이미지를 기계 수만큼 만드는 방법을 생각할 수 있다. 이렇게 하면 기계의 성능에 따른 이미지 분할을 해야 한다. 성능에 따른 분할에 필요한 전처리 비용과 공통된 이미지 영역을 사용할 수 없는 단점을 갖는다. 그러므로 최적의 영역의 크기는 그룹내의 기계의 수와 이미지의 크기에 따라서 결정된다.

8. 병렬 렌더링

병렬 렌더링 시스템은 Reyes구조를 기반으로 하는 Z 버퍼 방법의 렌더러를 병렬 렌더러의 엔진으로 사용하였다. 이 병렬 렌더러는 마스터 슬레이브 구조에 기반을 두었다. 마스터는 전처리와 후처리, 그리고 슬레이브들의 동적 부하 조절을 담당하고 네트워크가 교착 상태(deadlock)에 빠지지 않도록 한다. 슬레이브는 마스터로부터 데이터를 받은 후에 실제적인 렌더링을 하고 결과로서 계산되어진 프레임 버퍼(frame buffer)를 마스터에게 보낸다. 또한 슬레이브는 자신에게 할당된 영역의 렌더링이 모두 끝난 후에는 마스터에게 아직 일이 끝나지 않은 이미지 영역을 할당받아 렌더링을 수행한다. 마스터에서 행하여지는 전처리로서 입력 데이터의 해석, 곡면 데이터의 다각형화, 정적 부하 조절이 행하여지며, 후 처리에서는 각 슬레이브에서 계산된 프레임 버퍼를 하나의 프레임 버퍼로 합하여 파일로 출력하는 일을 하게 된다. 그림 1.19와 그림 1.20은 마스터와 슬레이브의 전체적인 알고리즘의 순서도이다.

가. 전처리

전처리는 마스터 프로그램이 슬레이브 프로그램에게 렌더링할 데이터를 나누어주기 위한 준비 과정이다. 우선 데이터를 파일로부터 읽어 들여 분석하고 슬레이브 프로그램에서 받아들이기 쉬운 형태로 변환한다. 즉 여러 가지 형식으로 되어있는 데이터 파일을 읽어 들여 하나의 통합된 데이터 형식, 즉 다각형 형식으로 변환한다. 그리고 각 다각형들에게 렌더링시에 필요한 옵션 및 그래픽 상태 등을 설정하게 된다. 그렇게 함으로서 슬레이브 프로그램에서의 렌더링을 간단하게 만들어 준다.

나. 데이터의 분석

본 시스템에서는 RIB(RenderMan Interface Bytestream)형식의 파일을 읽어들인다. 이 RIB 프로토콜은 아스키(ASCII)와 이진(binary)형태를 모두 지원하고 아스키 형태일 때는 다음과 같은 형태의 RIB binding형식을 갖는다.

```
RIB BINDING
Request parameter1 parameter2 ... parameterN
```

본 시스템은 이러한 RIB파일을 읽어 들이면서 렌더링에 필요한 여러 매개 변수(parameter)를 처리한다. 이러한 매개 변수는 몇 개의 슬레이브 프로세스의 개수, 수학적으로 정의된 물체를 다각형화 시킬 때 어느 정도의 정확성을 가지고 다각형화 시킬 것인지 등을 결정한다.

RIB화일에는 다각형 데이터 외에도 수학적으로 정의된 데이터가 포함되어 있을 수 있다. 이렇게 수학적으로 정의되어 있는 물체는 렌더링하는데 많은 시간을 필요로 하기 때문에 이러한 수학적 데이터는 다각형으로 변환한다.

파일로 읽어 들이는 데이터는 물체 데이터 뿐만이 아니다. 파일로부터 카메라의 위치와 방향, 광원의 개수, 광원들의 위치와 종류 및 색깔, 렌더링 시의

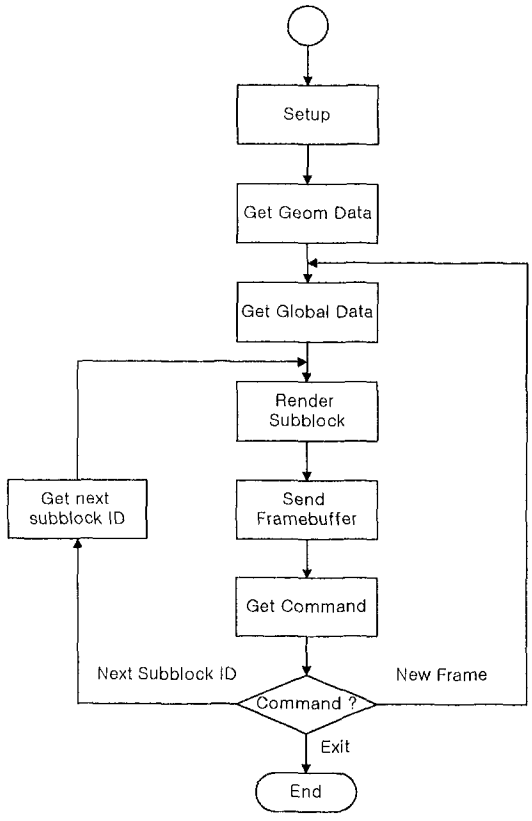


그림 1.19 슬레이브 프로그램의 순서도

여러 가지 필요한 옵션 등도 파일에서 같이 읽어 들인다.

다. 정적 부하 조절

본 프로그램은 여러 장의 이미지를 연속적으로 렌더링 하여 애니메이션을 제작하기 위한 목적으로 제작되었다. 애니메이션에서의 일련의 프레임들은 동일한 데이터를 가지고 구성하게 된다. 이는 마치 어느 일정한 배경에서 배우가 연기를 하는 일반적인 영화의 촬영과 같다고 볼 수 있다. 그러므로 애니메이션이 렌더링 될 때는 일련의 프레임들이 모두 같은 데이터를 가지고 있고 화면이 바뀔 때마다 카메라의 위치나 배우가 되는 데이터들의 변환 행렬들만이 바뀌게 된다. 그러므로 본 프로그램에서는 일련의 프레임들을 렌더링 하여 애니메이션

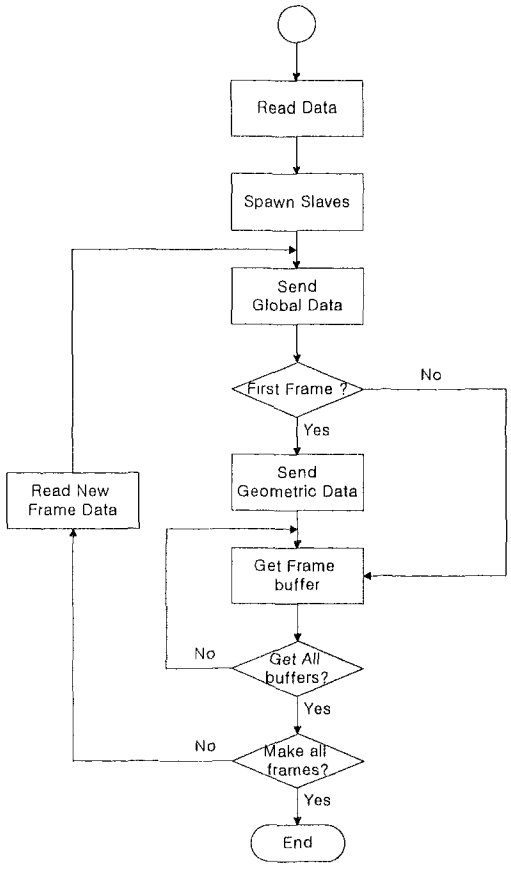


그림 1.20 마스터 프로그램의 순서도

을 구성할 때 필요한 모든 물체 데이터를 미리 보내어 놓고, 프레임이 바뀔 때 그 프레임에 필요한 부수적인 데이터들 만을 다시 보내어 계산할 이미지의 데이터를 구성하게 된다. 그러므로 전처리 과정에서의 정적 부하 조절은 이렇게 구성된 이미지를 여러 개의 영역으로 나누어 슬레이브 프로세스에게 이미지 영역의 계정을 할당하게 되는 것이다.

라. 렌더링

마스터 프로그램은 전처리 과정이 끝난 후 렌더링 과정으로 들어가게 되는데 이때 마스터 프로그램은 사용 가능한 프로세스 단위(Processing Element)를 할

당받아 각각의 PE에 슬레이브 프로그램을 수행시키고 필요한 데이터를 모두 전송한 후 슬레이브 프로세서로부터 전해져 오는 계산된 프레임 버퍼를 합성한다.

마. 동적 부하 조절

본 시스템은 동적 부하 조절을 하는데 있어서 특별한 부담을 주지 않는다. 우선 계산이 끝난 슬레이브는 자신이 계산한 프레임 버퍼와 함께 자신의 계정을 마스터에게 알려주기 때문에 마스터 프로그램은 어떤 슬레이브가 계산이 끝냈는지를 알고 있으며, 계산이 아직 끝나지 않은 영역과 아직 슬레이브에게 할당되지 않은 이미지 영역에 대한 정보 또한 알고 있다. 마스터 프로그램은 쉬는 슬레이브가 없도록 하기 위해 계산이 끝난 슬레이브에게 다음에 계산할 이미지의 계정을 전달한다.

마스터 프로그램이 한가지 더 처리해야 할 문제는 한 프레임의 계산이 끝났을 때이다. 이 때 마스터 프로그램은 새로운 데이터를 각 슬레이브에게 다시 나누어 주고 슬레이브에게 계산할 이미지 영역의 계정을 전달하게 된다.

바. 후처리

마스터 프로그램은 후처리 과정으로서 슬레이브 프로그램이 계산하여 보내어 오는 프레임 버퍼의 조각을 하나의 전체 이미지로 합성하여 프레임을 완성한다. 한 프레임에 대한 후처리가 끝난 후에 마스터 프로그램은 다음 프레임의 계산을 위한 처리과정으로 들어가게 된다.

사. 슬레이브 프로세스

슬레이브 프로그램은 자신에게 할당된 그래픽 데이터를 Reyes 방법으로 렌더링하여 계산된 프레임 버퍼를 마스터에게 보내주는 일을 한다. 슬레이브 프로

그램은 자신이 관리해야 할 여러 가지 변수들을 초기화하고 애니메이션에 사용될 모든 데이터를 마스터로부터 전송 받게 된다. 그 다음 폴리곤 데이터들을 임의의 영역으로 분할한다. 그리고 나서 마스터 프로그램으로부터 렌더링할 영역을 할당 받아서 Reyes방법으로 렌더링 한다. 렌더링이 끝난 후, 마스터에게 계산된 프레임 버퍼를 전송하고 다시 마스터에게 렌더링할 영역을 할당 받는다.

하나의 프레임을 모두 렌더링 했다는 메시지를 받으면 슬레이브 프로그램은 마스터로부터 변화가 있는 데이터를 전송 받아서 다음 프레임을 렌더링하기 시작한다. 모든 프레임을 렌더링 했을 때 슬레이브 프로그램은 종료한다.

아. 실험과 결과 분석

실험은 각 PE(Processing Element)가 128 Mbyte의 분산 메모리와 900MFLOPS의 계산 능력, 450 MHz의 Clock속도를 갖는 알파칩 프로세서를 가지고 있으며, 각 프로세싱 단위는 3차원 토러스(Torus) 구조로 구성되어있는 CRAY T3E MIMD 병렬 컴퓨터에서 실행 시켰다. 본 시스템은 각 PE간의 통신을 위해 PVM (Parallel Virtual Machine) 라이브러리를 사용했다.

본 연구에서 관심을 가지는 것은 하나의 이미지를 분산 처리하여 빠르게 렌더링하는 것이 아니라 여러 장의 이미지를 렌더링하여 하나의 애니메이션을 구성하는데 있어서의 렌더러의 구성이다. 그러므로 하나의 장면을 렌더링하는 것보다 여러 장의 이미지를 렌더링할 때 본 프로그램이 속도향상을 더욱 많이 하게 된다. 그러므로 본 연구에서는 위의 데이터를 가지고 10장의 이미지를 가지는 애니메이션을 렌더링 하였다. 본 연구에서는 하나의 프로세싱 단위를 가지고 애니메이션을 제작하였을 때 걸리는 시간과 여러 개의 프로세싱 단위를 사용하여 애니메이션을 제작하였을 때 걸리는 시간을 비교하여 어느 정도까지의 속도향상을 얻어낼 수 있는지를 보여준다. 또한 이미지의 크기에 대한 이미지

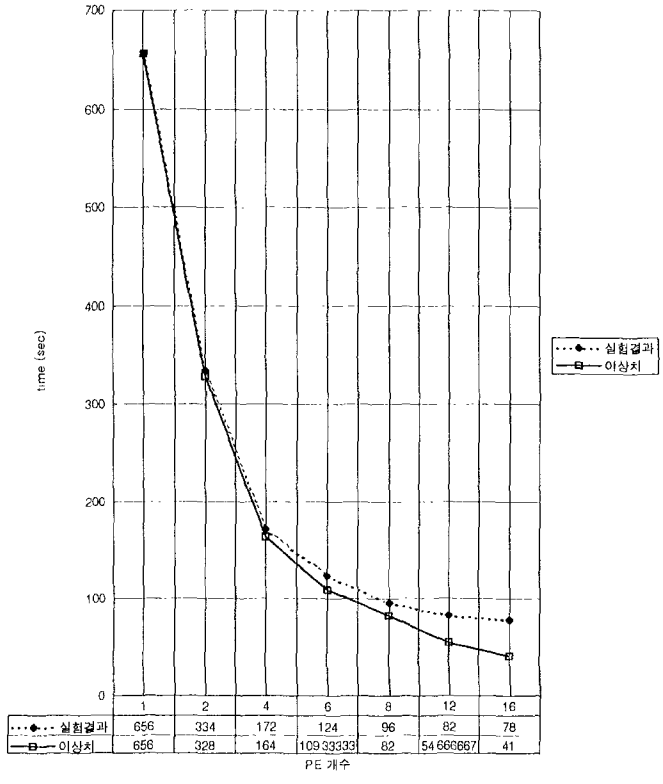


그림 1.21 PE수에 따른 렌더링 시간

영역 분할의 크기를 조사하여 분할된 영역의 크기와 속도향상의 관계를 보여준다.

본 시스템에서 사용한 입력 데이터는 노말 벡터(normal vector)를 갖는 16,739개의 삼각형과 2개의 광원을 사용하였으며 640×480의 이미지 크기를 갖는다.

실험 내용은 2개, 4개, 8개의 프로세싱 단위를 하나의 그룹으로 하였을 때 640×480의 해상도에서 32×32, 64×64, 128×128, 256×256의 영역 크기를 변화시키면서 렌더링 시간을 조사하였다. 분산 처리의 성능을 비교하기 위해서 하나의 PE에서의 결과와 비교를 하였다. 실험 결과를 통하여 다음과 같은 사실을 알 수 있다. 분산 처리를 통하여 하나의 프로세싱 단위를 사용하여 렌더링

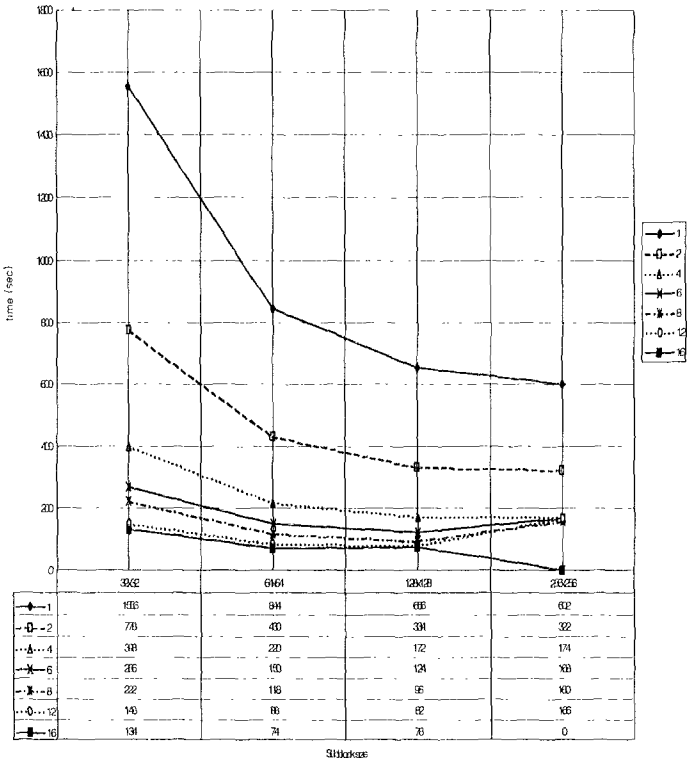
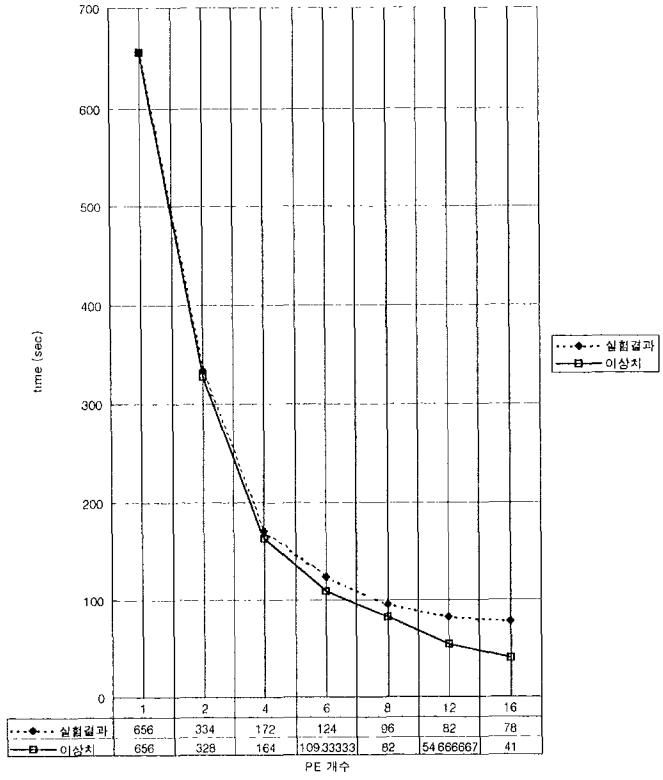


그림 1.22 이미지 영역의 크기에 따른 렌더링 시간

하였을 때 보다 8개의 프로세싱 단위를 사용하였을 때 656초 : 96초로 8개의 프로세싱 단위를 사용하였을 때 약 6.8배의 빠른 렌더링을 수행하였다. 그리고 하나의 프로세싱 단위를 사용하여 계산했을 때와 여러 개의 프로세싱 단위를 사용했을 때 영역의 크기에 따른 렌더링 시간의 변화는 비슷하게 이루어진다 (그림 1.21). 이것은 이미지 영역의 개수가 프로세서 개수에 비해 약 2배정도 많을 때 가장 많은 속도 향상을 보여주게 된다. 여러 개의 프로세싱 단위를 사용하여 렌더링 했을 때 걸리는 시간의 하나의 프로세싱 단위를 사용하여 렌더링 했을 때 보다 선형적인 속도 향상을 얻는 것이 이상적이지만 전처리 및 후처리 과정에서의 PE간의 통신과 이미지의 계산 시에 일어나는 중복된 계산 때문에 실제 선형적인 속도 향상을 기대하기는 어렵다. 그림 1.22는 하나의 프로



	1	2	4	6	8	12	16
실험결과	656	334	172	124	96	82	78
이상치	656	328	164	109.33333	82	54.666667	41

그림 1.23 애니메이션 제작에 걸리는 시간

세싱 단위를 사용하여 렌더링을 하였을 때와 2개, 4개, 8개의 프로세싱 단위를 사용하여 이미지 영역을 128×128로 나누어서 렌더링 하였을 때의 시간을 비교한 그래프이다.

같은 수의 프로세싱 단위를 이용하여 렌더링을 하였을 때 이미지 영역의 크기에 따른 렌더링 시간의 차이는 이미지 영역의 크기가 작으면 작을수록 프로세싱 단위가 일을 하지 않고 쉬는 시간이 줄어드는 대신에 하나의 물체가 여러 이미지 영역에 걸쳐서 존재 할 수 있기 때문에 여러 개의 이미지 영역이 같은 물체에 대한 계산을 중복하여 할 가능성이 많이 때문에 렌더링 시간이 늘어날 수 있다. 이미지 영역들이 중복된 계산을 피하기 위해서 이미지 영역을 크게 하면 그에 따라서 동적 부하 조절에 문제가 생겨서 계산을 하지 않고 다음 계

그림 1.24 house.rib을 병렬 렌더링 한 영상 (a)

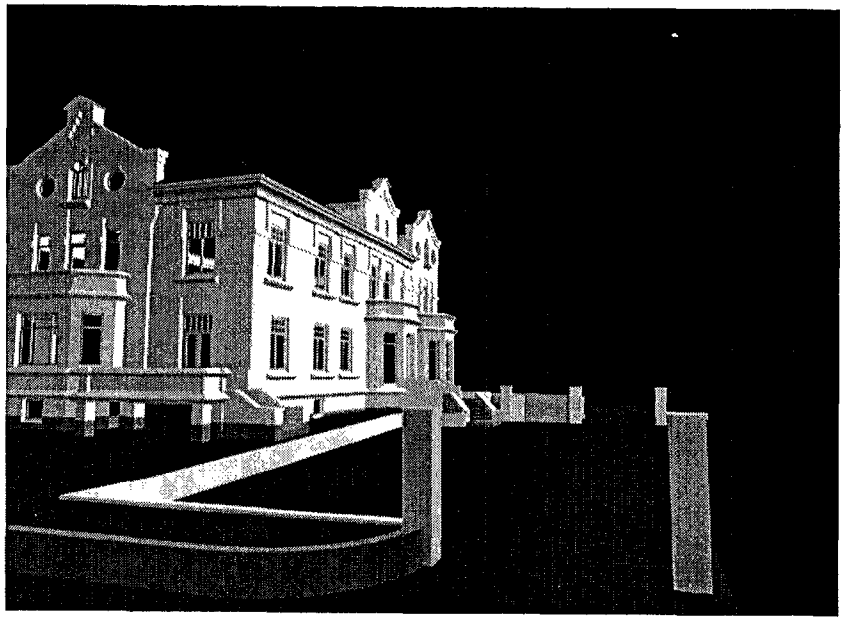


그림 1.25 house.rib를 병렬 렌더링 한 영상 (b)

산할 데이터를 받기 위해서 기다리고 있는 프로세싱 단위들이 많은 시간을 렌더링 하지 않고 기다려야 한다. 그러므로 각 이미지의 크기나 프로세싱 단위의

개수에 따른 적당한 이미지 영역의 크기를 정하여야 한다.

그림 1.23은 하나의 애니메이션을 제작할 때 하나의 프로세싱 단위를 사용했을 때와 4개의 프로세싱 단위를 사용하여 10개의 프레임을 갖는 화면을 구성할 때의 렌더링 시간을 나타내는 그래프이다. 본 연구에서 개발된 프로그램은 하나의 이미지의 렌더링을 할 때 보다는 여러 장의 애니메이션을 제작할 때 보다 좋은 속도 향상을 나타낼 수 있다. 본 프로그램은 물체 데이터를 슬레이브 프로그램에게 모두 보내줌으로써 애니메이션을 렌더링할 때 이전 프레임에 비해 변경된 데이터만을 보내주게 된다. 애니메이션을 렌더링할 때에는 처음 프레임을 렌더링할 때보다 적은 양의 데이터가 마스터로부터 슬레이브에게 전달이 됨으로써 많은 양의 프레임을 렌더링 할 때 보다 많은 렌더링 시간을 줄일 수 있다.

그림 1.24와 그림 1.25는 house.rib 파일에 대한 병렬 렌더링 한 영상의 예를 보여준다.

9. 렌더링 옵션 역분배 기법과 최적화를 위한 성능향상

컴퓨터 그래픽스에서 렌더링은 고정된 값으로 계속해서 사용할 수 있는 성질의 것이 아니라, 가변하는 전경을 계속해서 렌더링하여 이미지를 만들어야만 이 하나의 영화가 완성되는 것이다. 이러한 이유로 렌더링의 이론은 계속해서 속도의 향상을 목적으로 발달되어 왔다. 그런데 여기서 한가지 주목할 만한 사실은 컴퓨터 그래픽스가 여러 복잡한 기법들을 동원하고 있기 때문에 최종 사용자가 알아야만 하는 이론적인 부분이 너무나 많다는 것이다. 예를 들어 전경을 배치하고 렌더링을 시작해야할 부분에서 사용자들은 때때로 너무나 많은 옵션들을 만나게 된다. 물론 사용자들을 위한 편의 옵션도 존재하지만 대부분은 그저 복잡한 옵션들이기도 하다[1-11,1-12].

본 연구에서는 여기에 착안점을 두어 각 옵션들이 서로 어떤 연관성이 있으며 어떤 식으로 하나의 통합된 옵션을 만들어 낼 수 있는지에 대해 연구하였다. 즉 최종 사용자가 Reyes 구조 알고리즘에 대한 전문적인 지식이 없는 상태에서 최대한 단순화된 옵션, 즉 예를 들어 총 렌더링 소요 시간을 렌더러에 지정해 주었을 때 본 논문에서 제시한 결과치를 이용하여 렌더러는 같은 시간 비용을 이용하여 최종 이미지의 화질을 극대화시킬 수 있다.

본 연구에서는 우선 렌더링을 시작할 때 설정해 주어야 할 옵션들 중에서 시간 비용과 최종 이미지의 화질에 가장 많은 영향을 미치는 3개의 옵션을 대상으로 연구하였다[1-11,1-12].

그 첫째로는 미세 다각형의 조밀도이다. 미세 다각형의 조밀도란 다각형을 세이딩의 기본 단위인 미세 다각형으로 얼마나 잘게 쪼갤 것인지를 의미하며 실제로 모든 다른 옵션들 보다 렌더링 시간 비용과 최종 이미지의 화질에 가장 큰 영향을 미치는 옵션이다. 또한 이 옵션은 다른 많은 옵션들과 연관성을 지니고 있다. 둘째 옵션으로는 하나의 화소에 대한 샘플링 점의 개수를 꼽을 수 있다. 하나의 화소에 대해 얼마만큼의 샘플링을 수행할 것인지를 결정하는 옵션이다. 이 옵션 또한 이미지의 화질에 많은 영향을 미치는 옵션이다. 세 번째 옵션으로는 하나의 세이딩에 따르는 그림자 Z 버퍼 샘플링 점의 개수를 들 수 있다. 실제로 그림자 Z 버퍼(Shadow Z buffer) 방법으로 그림자 효과를 구현할 때에는 심한 알리아싱이 생길 수밖에 없다. 이는 이러한 알리아싱을 얼마만큼의 샘플링을 하여 최소화시킬 것인지에 대한 옵션이다.

이렇게 위에서 언급한 세 개의 옵션들이 실제로 각각 어떤 식으로 최종이미지의 화질과 전체 렌더링 시간비용에 영향을 미치며 가중치 테이블을 어떤 식으로 구성할 지에 대해서 먼저 알아보고 난 후 실제로 가중치 테이블을 구성하고 이 테이블이 보편타당한지를 실험할 것이다. 그리고 이 실험 결과를 이용하여 각 옵션들이 어떤 상호 연관성을 갖고 있는지를 알아보는 방식으로 연구를

진행하였다. 이런 방식을 사용함으로써 같은 정도의 이미지 화질을 보장하면서도 최소한의 시간 비용을 이용하여 렌더링 시킬 수 있는 하나의 통합된 옵션, 즉 시간 옵션을 얻어낼 수 있었다.

가. 렌더링 옵션

본 연구에서는 미세 다각형의 조밀도, 하나의 픽셀에 대한 샘플링 정의 개수, 그림자 Z 버퍼 샘플링 점의 개수등 비교 가능한 3개의 옵션을 선정하였는데, 이들은 Reyes구조에 기반한 렌더러에서 최종 이미지의 화질과 렌더링 시간에 많은 영향을 미친다.

(1) 미세 다각형의 조밀도

다이싱을 할 때 가장 중요한 파라미터는 다각형을 얼마나 많은 미세 다각형으로 쪼갤 것인가 하는 것이다. 앞에서 설명한 바와 같이 다이싱 작업은 본 렌더러에서 가장 핵심적인 부분이다. Nyquist 샘플링은 샘플링 시에 하나의 샘플링 점이 픽셀의 4분의 1의 크기를 갖게 하는데, 실제로 이 조건을 만족시키려면 다이싱 과정에서 해당 다각형이 최종 이미지에서 차지하는 픽셀 수의 4배로 미세 다각형을 생성해야 한다. 이렇게 함으로써, 이미지의 알리아싱을 줄일 수 있는 이론적인 조건을 충족시킬 수 있다.

하지만 실제 렌더링에 적용시켜보면 시간 비용은 다이싱으로 쪼개지는 미세 다각형의 수에 비례해서 증가하게 된다. 다이싱으로 쪼개진 미세 다각형의 격자점에서 세이딩과 그림자 Z 버퍼의 샘플링 및 필터링이 일어나며 샘플링 또한 미세 다각형을 기본으로 하기 때문이다. 그리고 세이딩이 미세 다각형의 격자점에서만 일어나기 때문에 미세 다각형의 조밀도가 최종 이미지의 화질에 미치는 영향도 매우 크다.

본 연구에서는 옵션의 범위를 0.5와 2.0사이로 제한시켰다. 이는 하나의 다각형을 스크린 공간에서 바운딩시켰을 때 바운딩 상자의 가로, 세로 크기의 얼

마만큼의 비율로 미세 다각형의 수를 결정할 것인가 하는 것이다. 즉 1.0이란 미세 다각형의 개수가 다각형의 스크린 공간상에서의 픽셀 크기와 같은 수로 결정되는 것을 의미하고, 2.0은 Nyquist 샘플링 조건에 맞춘 최대의 값이 된다. 실제로 0.5 이하의 값은 최종 이미지의 화질을 고려할 때 별 의미가 없는 값이라 할 수 있고 2.0 이상의 값은 소요되는 시간 비용과 최종 이미지의 화질을 고려할 때 그 의미가 상실되는 값이라 할 수 있다. 본 연구에서는 이 범위 내의 값 중에서 0.5, 1.0, 1.5, 2.0등 4개의 값을 선택하여 각 옵션별 렌더링 시간과 화질에 미치는 영향을 실험하였으며 가중치 테이블을 구성하기 위한 실험에서는 이를 더 세분화하여 0.75, 1.0, 1.25, 1.5, 1.75, 2.0 이렇게 6개의 값을 이용했다.

(2) 하나의 픽셀에 대한 샘플링 점의 개수

샘플링을 할 때 중요하게 고려해야 할 옵션 중 하나는 바로 한 픽셀에 대한 샘플링 점의 개수이다. 앞에서 설명한 바와 같이 Nyquist 샘플링 조건을 만족시키려면 하나의 픽셀에 대해 최소 4개의 샘플링을 하여야 한다. 그러나 이것은 어디까지나 이론적인 값이며 샘플링을 1개만 해도 기본적인 렌더링 기능은 할 수 있다. 즉 샘플링 점의 개수의 많고 적음은 최종 이미지의 화질과 렌더링 시간 비용에 영향을 미치는 옵션이 된다. 보통 2 × 2의 샘플링을 지원하나 특별한 이미지에 대해서는 1 × 1 또는 3 × 3의 샘플링을 요구할 수도 있다.

샘플링 점의 개수는 샘플링의 다음 작업인 필터링에도 상당한 영향을 미친다. 이를 그림 1.26에 나타내었다.

렌더링에 적용시킬 수 있는 픽셀당 샘플링 점의 개수는 통상 1, 4, 9 인데 이는 1 × 1, 2 × 2, 3 × 3을 의미한다. 1보다 작은 수는 생각할 수 없으며 3 × 3 이상은 Nyquist 샘플링 조건 이상으로써 최종 이미지의 화질과 렌더링 시간 비용을 고려할 때 별 장점을 지니지 못한다. 위 그림에는 이들 세 가지의 샘플링 점 개수 형태를 나타내었는데, 보는 바와 같이 필터링 영역이 서로 다

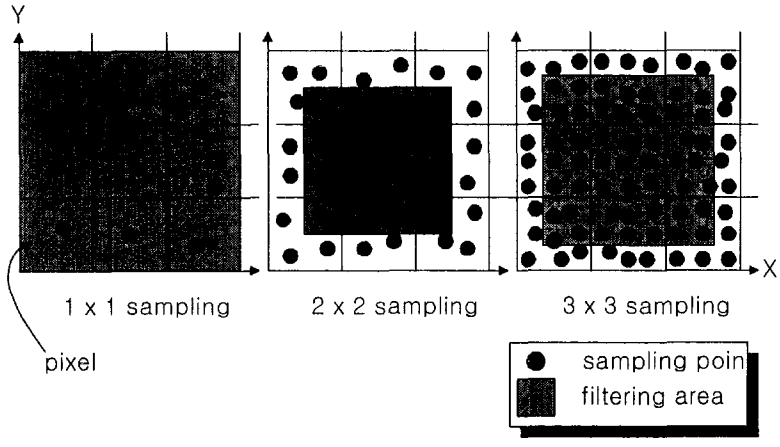


그림 1.26 샘플링 점의 개수와 필터링 영역

르다. 이는 샘플링 점의 개수가 짝수가 되지 못한데 인한 것으로 필터링 함수를 적절히 사용함으로써 서로간에 생기는 필터링 영역의 불일치를 해소할 수 있다.

위와 같은 이유로 본 연구에서는 이들 세 가지의 샘플링 점에 대한 옵션으로 실험하였으며 그 표기를 순서대로 하나의 축으로만 샘플링되는 점의 개수인 1, 2, 3으로 하였다.

(3) 하나의 세이딩에 따르는 그림자 Z 버퍼 샘플링 점의 개수

하나의 세이딩 점은 한 개 이상의 그림자 Z버퍼 샘플링 점을 가진다. 이는 그림자 효과의 알리아싱을 줄이기 위한 방법인데 보통 1, 4 또는 16 개의 샘플링 점을 사용한다. 근접 확률 필터링을 위한 샘플링 점의 개수는 최종 이미지의 화질과 시간 비용 두 측면 모두에 영향을 미치므로 본 연구에서는 이를 세 번째의 옵션으로 정했다.

샘플링 점의 개수가 적을 경우 렌더링 시간은 단축되지만 그림자의 알리아싱 때문에 최종 이미지의 화질이 떨어지게 되고 샘플링 점의 개수가 늘어날 경우 렌더링 시간은 길어지지만 그림자의 알리아싱이 줄어들기 때문에 이미지의 화질은 좋아진다.

본 연구에서는 이 옵션의 설정 값을 1, 4, 16 으로 조절하여 실험하였는데 이는 각각 1 × 1, 2 × 2, 4 × 4의 해상도를 의미한다. 그리고 표기는 순서대로 하나의 축으로만 샘플링되는 점의 개수인 1, 2, 4로 하였다.

나. 가중치 테이블 설계 및 상호 연관성 분석

본 연구에서는 각 옵션이 렌더링 시간과 화질에 미치는 영향을 조사해 봄으로써 개별적인 옵션이 각각 어느 정도의 영향력을 가지고 있는지를 알아보고자 한다. 이 데이터는 각 옵션들의 특성을 분석하는데 사용됨과 동시에 각 옵션들의 상호 연관성을 밝히는데 있어 참고자료로 사용될 수 있다. 그리고 옵션들이 서로 연관된 복합적인 실험을 거쳐 가중치 테이블을 구성하고 일률적인 선형 가중치 옵션들을 사용한 결과와 비교하여 그 효용 가치를 분석할 것이다. 그리고 가중치 테이블을 분석하여 각 옵션들이 서로 어떠한 연관성을 지니고 있는지를 파악해본다.

(1) 실험 환경 및 특성

본 실험에 사용된 기계는 Sun사의 UltraSPARC 기종이며 CPU는 167MHz 클럭을 사용하고 메모리는 128MB가 장착되어 있다. 운영체제는 Solaris 2.5를 사용하고 있다.

각 옵션별 렌더링 시간과 화질 효율성 비교와 가중치 테이블 설계의 실험을 위하여 각각 Room과 Museum, 그리고 Sphere라 명명한 세 개의 실험 데이터를 사용하였는데 이들은 광원의 개수에 따라 해당 이름 뒤에 광원의 개수를 의미하는 숫자를 붙여 구체적으로 표기하였다. 이들 데이터는 모두 RenderMan 규격을 따르는 rib 형식으로 작성되어진 파일이다. 다음 표 1.2는 각 데이터의 특성을 정리한 것이다.

최종 이미지의 화질을 비교하기 위해서 PSNR(Peak-Signal-to-Noise-Ratio)

표 1.2 실험 데이터 환경

	Room1	Room2	Museum1	Museum2	Sphere1	Sphere2
최종 이미지 해상도	640x480	640x480	640x480	640x480	640x480	640x480
광원 개수	1	2	1	2	1	2
다각형 데이터 개수	14521	14521	7283	7283	2701	2701

공식을 이용하였다. 이 공식은 신호(signal)값을 노이즈(noise)의 비율로 나타내는데 있어 각 이미지의 최고값을 이용하는 방법인데 실제 공식은 다음과 같다.

$$PSNR = \frac{xsize \times ysize \times peak^2}{\sum_{x=0}^{xsize-1} \sum_{y=0}^{ysize-1} \left(\hat{f}(x,y) - f(x,y)\right)^2}$$

공식에서 xsize와 ysize는 각각 이미지의 가로와 세로 크기이며 peak는 이미지 색의 최대값이다. 그리고 $\hat{f}(x,y)$는 비교할 이미지의 픽셀 값이며 $f(x,y)$는 비교 원본 이미지의 픽셀 값이다. PSNR의 값은 보통 dB단위로 나타내는데 그 공식은 다음과 같다.

$$PSNR(dB) = 10\log_{10} PSNR$$

PSNR을 dB단위로 나타내었을 때의 그 값은 0보다 크며 같은 이미지를 비교하였을 때에는 무한대(Inf)값을 나타낸다. PSNR은 사람의 눈으로는 구분하지 못할 만큼의 차이점도 수치로 구분해 내는 장점을 지니고 있지만 완전한 객관성을 지니고 있지 못하다는 단점도 있다. 원본과 비교 대상 이미지의 픽셀만을 일대일 비교하여 그 차이값을 누적시켜나가는 방법을 사용함으로써 전체 이미지의 조화성을 무시하였기 때문이다. 사람의 눈에는 전체적인 이미지의 윤곽이나 조화가 상응하는 픽셀간의 차이보다 더 크게 느껴진다. 또 다른 단점은 비

교 대상이 되는 두 이미지의 특성에 너무 의존적이라는 점이다. 즉 비슷한 형식의 옵션 배열을 이용하여 렌더링된 두 이미지는 실제 사람의 눈으로 느끼는 차이에 비해 훨씬 적은 차이값을 보여준다.

(2) 각 옵션별 렌더링 시간과 화질 효율성 비교

그림 1.27,그림 1.28 그리고 그림 1.29는 각 옵션이 전체 렌더링 시간과 최종 이미지의 화질에 미치는 영향을 알아보기 위해 다른 옵션들은 특정 값에 고정시킨 상태에서 해당 옵션의 값만을 변화시키며 시간과 화질을 각각 초단위와 dB 단위로 측정해 본 결과이다.

타 옵션이 변화할 때 각 옵션 별로 고정시킨 특정 값은 미세 다각형의 조밀도의 경우 1.0이며 픽셀당 샘플링 점의 개수는 2, 그리고 하나의 셰이딩에 따르는 그림자 Z 버퍼 샘플링 점의 개수는 2이다. 각 옵션별로 왼쪽 그래프는 Room1 데이터의 결과이며 오른쪽 그래프는 Museum2 데이터의 결과이다.

모든 옵션을 최상과 최하로 주었을 때의 시간과 화질 결과에서 각각 최저의 시간과 화질, 그리고 최고의 시간과 화질을 얻어서 그 범위를 이용하여 그래프를 나타내었는데 실험 결과에서 알 수 있듯이 Room1 데이터와 Museum2 데이터는 각 옵션의 변화에 대해 서로 거의 비슷한 양상을 보여 주었다. 이것은 곧 대부분의 데이터가 모든 옵션의 변화에 대한 렌더링 시간과 화질의 변화가 비슷한 양상 또는 상수 배의 비율로 증가하거나 감소하는 형태를 보여줄 것이라는 예측을 할 수 있게 한다.

미세 다각형의 조밀도 옵션의 경우 Room1 데이터와 Museum2 데이터 각각의 시간 비용에 대한 이미지 화질의 변화 양상은 다소 다르다. 그 이유는 Museum2 데이터의 경우 광원이 두 개인 것에서 찾을 수 있다. 광원이 많아지면 셰이딩 시간이 증가함과 동시에 그림자 생성을 위한 그림자 Z 버퍼 작업 또한 비례하여 증가된다. 그리고 광원에 의한 그림자의 알리아싱이 이미지의 화질에도 영

향을 미치게 된다.

샘플링 점의 개수 옵션의 경우는 두 데이터의 시간 비용과 화질 양상이 거의 비슷한 것으로 나타났다. 이것은 이 옵션이 별다른 외부 환경의 영향을 거의 받지 않는다는 것을 보여준다.

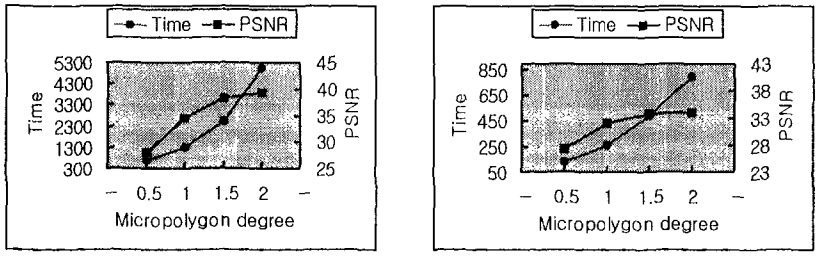


그림 1.27 미세 다각형의 조밀도 옵션

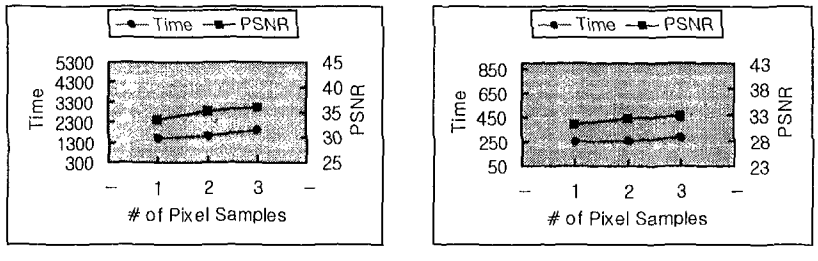


그림 1.28 픽셀당 샘플링 점의 개수 옵션

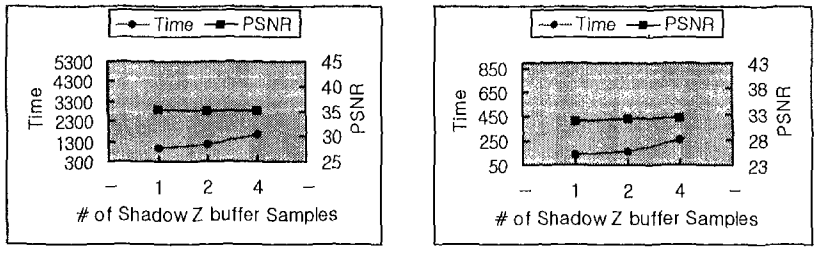


그림 1.29 하나의 세이딩에 따르는 그림자 Z 버퍼 샘플링 점의 개수

그림 1.30 미세 다각형의 조밀도 옵션에 따른 이미지 화질의 변화

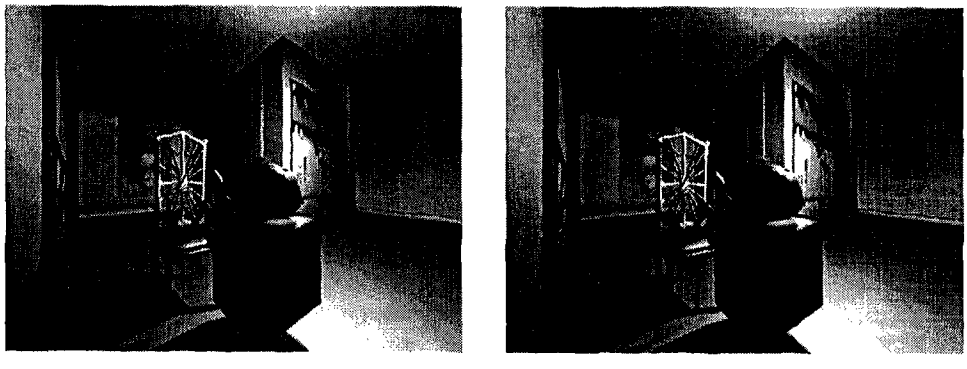


그림 1.31 픽셀당 샘플링 점의 개수 옵션에 따른 이미지 화질의 변화

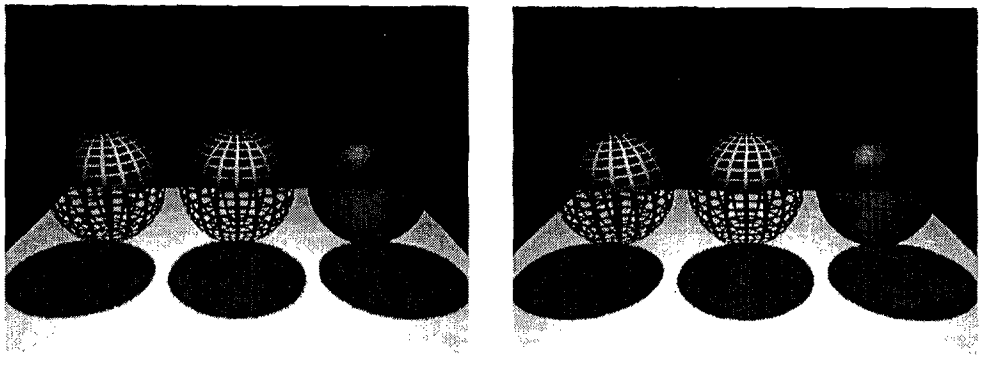


그림 1.32 그림자 Z 버퍼 샘플링 점의 개수 옵션에 따른 이미지 화질의 변화

그림자 Z 버퍼 샘플링 점의 개수 옵션의 경우는 두 데이터에 대한 화질 양상은 거의 비슷한 추세를 나타내나 시간 비용 측면에서는 Museum2 데이터가 훨씬 높은 증가 추세를 보인다. 이 또한 광원의 개수 차이가 그 이유라 할 수 있다.

그림 1.30, 그림 1.31 그리고 그림 1.32는 각 옵션의 변화에 따라 최종 이미지의 화질이 어떻게 변하는 지를 보여준다. 그림 1.30은 미세 다각형의 조밀도 옵션에 따른 Room1 이미지의 변화를 나타내고, 그림 1.31은 픽셀당 샘플링 점의 개수 옵션에 따른 Museum1 이미지의 변화를, 그리고 그림 1.32는 그림자 Z 버퍼 샘플링 점의 개수 옵션에 따른 Sphere1 이미지의 변화를 나타낸다. 왼쪽 그림은 옵션을 낮게 설정했을 때의 결과이고 오른쪽 그림은 옵션을 높게 설정하였을 때의 결과이다.

(3) 가중치 테이블 설계

본 연구에서 정의한 가중치 테이블이란 사용자가 지정해 줄 수 있는 렌더링 옵션을 단 하나로 제한하였을 때, 즉 하나의 옵션만을 조정함으로써 전체 렌더링 시간과 최종 이미지 화질을 결정하게 하였을 때 그 옵션의 정도에 따라 해당 구간에서 낼 수 있는 최상의 이미지 화질을 보장하며 옵션이 변함에 따라 렌더링 시간이 선형적(linear)으로 변하게 함으로써 사용자가 전체 예상 시간을 예측 가능하게 해 주는 세부 옵션들의 배열을 구하기 위한 테이블이다. 따라서 본 연구에서 설계한 가중치 테이블을 이용하면 사용자가 입력한 하나의 파라미터를 가중치 테이블에 대입하여 해당 파라미터 값에 가장 알맞은 세부 옵션들의 배열을 구해낼 수 있으며 그 결과, 선형적인 렌더링 시간과 함께 그 시간동안만에 계산해낼 수 있는 최적의 이미지 화질을 얻어낼 수 있게 된다.

가중치 테이블을 설계하기에 앞서 생각해 보아야 할 점이 있다. 외부 파라미터의 영향이 그것인데, 설명한 광원의 개수는 이미지의 전체 렌더링 시간에 많은 영향을 미친다. 즉 광원의 개수는 그 많고 적음에 따라 각 시간 구간에서 최고의 화질을 얻기 위한 옵션들의 배열 상태를 다르게 한다. 따라서 본 연구에서는 몇 가지 실험을 거쳐 광원의 수가 옵션들의 배열에 어떠한 영향이 있는지 파악하고, 각 광원의 수에 따라 서로 다른 가중치 테이블을 설계해 보았다.

0.75	1	1	286.0	0.00	33.30	352.7	0.00	33.13	66.67
0.75	1	2	313.7	0.01	31.74	413.7	0.02	31.67	100.00
0.75	1	4	436.0	0.06	31.98	661.0	0.08	31.84	225.00
0.75	2	1	330.3	0.02	34.19	399.0	0.01	33.96	68.67
0.75	2	2	360.0	0.03	32.23	462.3	0.03	32.08	102.33
0.75	2	4	485.0	0.08	32.52	709.7	0.09	32.28	224.67
0.75	3	1	482.3	0.07	35.15	581.3	0.06	34.69	99.00
0.75	3	2	524.7	0.09	32.83	640.7	0.07	32.57	116.00
0.75	3	4	650.0	0.14	33.15	892.7	0.13	32.80	242.67
1.00	1	1	452.3	0.06	34.13	545.0	0.05	33.93	92.67
1.00	1	2	495.3	0.08	34.12	652.7	0.07	34.10	157.33
1.00	1	4	704.0	0.16	34.32	1070.3	0.18	34.29	366.33
1.00	2	1	490.7	0.08	35.68	598.3	0.06	35.34	107.67
1.00	2	2	544.7	0.10	35.73	706.3	0.09	35.63	161.67
1.00	2	4	754.3	0.18	36.03	1125.0	0.19	35.88	370.67
1.00	3	1	660.3	0.14	36.51	788.7	0.11	35.88	128.33
1.00	3	2	719.7	0.16	36.47	900.3	0.14	36.14	180.67
1.00	3	4	926.0	0.24	36.81	1320.3	0.24	36.42	394.33
1.25	1	1	658.7	0.14	34.51	793.7	0.11	34.28	135.00
1.25	1	2	732.7	0.17	35.51	962.3	0.15	35.63	229.67
1.25	1	4	1050.7	0.29	35.70	1602.3	0.31	35.80	551.67
1.25	2	1	714.3	0.16	36.33	862.7	0.13	35.93	148.33
1.25	2	2	795.7	0.19	38.21	1027.7	0.17	38.26	232.00
1.25	2	4	1114.3	0.31	38.50	1667.7	0.33	38.51	553.33
1.25	3	1	896.3	0.23	37.34	1065.0	0.18	36.61	168.67
1.25	3	2	1019.0	0.28	39.63	1233.0	0.22	39.31	214.00
1.25	3	4	1374.3	0.41	40.02	1894.3	0.39	39.63	520.00
1.50	1	1	973.0	0.26	34.64	1103.7	0.19	34.42	130.67
1.50	1	2	1199.7	0.35	36.31	1344.0	0.25	36.51	144.33
1.50	1	4	1683.3	0.53	36.49	2247.7	0.47	36.68	564.33
1.50	2	1	1145.7	0.33	36.56	1184.0	0.21	36.16	38.33
1.50	2	2	1269.0	0.37	39.99	1417.3	0.27	40.19	148.33
1.50	2	4	1751.7	0.56	40.25	2324.7	0.49	40.46	573.00
1.50	3	1	1343.0	0.40	37.69	1395.7	0.26	36.96	52.67
1.50	3	2	1257.0	0.37	42.58	1666.0	0.33	42.17	409.00
1.50	3	4	1707.3	0.54	43.08	2524.3	0.54	42.64	817.00
1.75	1	1	1167.0	0.33	34.65	1541.3	0.30	34.43	374.33
1.75	1	2	1334.7	0.40	36.61	1817.7	0.37	36.89	483.00
1.75	1	4	1955.3	0.63	36.76	3041.0	0.67	37.04	1085.67
1.75	2	1	1255.7	0.37	36.69	1589.3	0.31	36.24	333.67
1.75	2	2	1415.0	0.43	40.62	1902.0	0.39	41.15	487.00
1.75	2	4	2034.0	0.66	40.85	3120.0	0.69	41.41	1086.00
1.75	3	1	1472.3	0.45	37.91	1816.3	0.37	37.12	344.00
1.75	3	2	1632.7	0.51	44.32	2129.0	0.44	44.15	496.33
1.75	3	4	2265.3	0.75	44.92	3389.3	0.76	44.77	1124.00
2.00	1	1	1530.0	0.47	34.90	1893.7	0.38	34.70	363.67
2.00	1	2	1752.7	0.56	37.23	2264.7	0.48	37.57	512.00
2.00	1	4	2583.0	0.87	37.36	3880.0	0.88	37.73	1297.00
2.00	2	1	1642.0	0.52	37.24	1938.3	0.40	36.83	296.33
2.00	2	2	1854.3	0.60	42.86	2354.3	0.50	43.55	500.00
2.00	2	4	2687.3	0.91	42.99	3969.7	0.90	43.81	1282.33
2.00	3	1	1884.0	0.61	38.76	2194.0	0.46	37.96	310.00
2.00	3	2	2055.7	0.67	56.92	2607.7	0.56	55.90	552.00
2.00	3	4	2919.7	1.00	Infinite	4356.7	1.00	Infinite	1437.00

표 1.3 가중치 테이블을 위한 실험 결과

가중치 테이블을 작성하기 위해서 먼저 세 가지 옵션들을 각각의 범위 내에서 변화시켜가며 그 변화 추이에 따른 전체 렌더링 시간과 최종 이미지와 화질을 구해보았다. 이것은 이들 세 가지 옵션들이 상호 작용하여 만들어낸 각기 다른 배열 상태에 따라 전체 렌더링 시간과 최종 이미지의 화질이 어떻게 변화

하는 지를 구해내기 위한 작업으로 가중치 테이블 설계의 첫 번째 단계가 된다. 표 1.3에서 첫 번째 옵션은 미세 다각형의 조밀도(Opt1)이고 두 번째 옵션은 샘플링 점의 개수(Opt2), 그리고 세 번째 옵션은 그림자 Z 버퍼 샘플링 점의 개수(Opt3)이다.

L1 Time은 Room1 데이터와 Museum1, 그리고 Sphere1 데이터의 렌더링 시간 비용의 평균을 나타낸다. 이에 대한 최종 이미지 화질 결과의 평균을 L1 PSNR로 나타내었다. 그리고 Room2, Museum2, Sphere2 데이터에 대한 시간과 화질 결과치 평균은 각각 L2 Time과 L2 PSNR로 표시하였다.

그리고 두 종류의 결과 중 시간비용은 그 전체 범위를 참고하여 정규화(normalize)시켰으며 이를 Normalize로 나타내었다. 즉 0.0에서 1.0 사이의 값으로 시간 범위를 재조정하여 따로 보인 것이다. 이것은 광원 개수의 변화에 따른 시간 비용의 변화 추이를 보다 쉽게 알아보기 위한 것이다. 표 1.3의 제일 오른쪽에 표시된 TimeDiff는 광원의 개수가 다른 두 가지 종류의 데이터 집합의 평균 시간 비용의 차이를 표시한 것이다. 즉 이 값은 광원의 개수가 두 개 이상인 데이터의 실험 결과 값을 이들 두 데이터의 결과 값으로부터 추측해 내기 위해 사용된다.

광원의 개수가 달라지게 되면 같은 조합의 옵션 상태에서도 요구되는 시간 비용 비율이 달라지게 되므로 광원의 개수가 서로 다른 이들 두 데이터 집합의 결과 값을 이용하여 실험되지 않은 다른 많은 데이터들의 실험 결과 값을 유도해 낼 수 있다. 실제로 첫 번째 데이터 집합(L1)은 한 개의 광원을 가지고 두 번째 데이터 집합(D2)은 두 개의 광원을 가지고 있는데 표의 결과 값을 살펴보면 각 옵션의 변화에 따라 두 데이터 집합는 서로 비슷한 양상의 이미지 화질 값(PSNR)을 나타내게 된다. 이것은 곧 대부분의 이미지가 서로 같은 옵션 배열로 렌더링 되었을 때 이미지의 화질은 광원을 포함한 여러 가지 파라미터들의 영향을 별로 받지 않고 비슷한 양상을 보인다는 것을 의미한다. 이것은 곧 위

0.75	1	1	286.0	0.00	33.30	0.75	1	1	352.7	0.00	33.13
0.75	1	2	313.7	0.01	31.74	0.75	2	1	399.0	0.01	33.96
0.75	2	1	330.3	0.02	34.19	0.75	1	2	413.7	0.02	31.67
0.75	2	2	360.0	0.03	32.23	0.75	2	2	462.3	0.03	32.08
0.75	1	4	436.0	0.06	31.98	1.00	1	1	545.0	0.05	33.93
1.00	1	1	452.3	0.06	34.13	0.75	3	1	581.3	0.06	34.69
0.75	3	1	482.3	0.07	35.15	1.00	2	1	598.3	0.06	35.34
0.75	2	4	485.0	0.08	32.52	0.75	3	2	640.7	0.07	32.57
1.00	2	1	490.7	0.08	35.68	1.00	1	2	652.7	0.07	34.10
1.00	1	2	495.3	0.08	34.12	0.75	1	4	661.0	0.08	31.84
0.75	3	2	524.7	0.09	32.83	1.00	2	2	706.3	0.09	35.63
1.00	2	2	544.7	0.10	35.73	0.75	2	4	709.7	0.09	32.28
0.75	3	4	650.0	0.14	33.15	1.00	3	1	788.7	0.11	35.88
1.25	1	1	658.7	0.14	34.51	1.25	1	1	793.7	0.11	34.28
1.00	3	1	660.3	0.14	36.51	1.25	2	1	862.7	0.13	35.93
1.00	1	4	704.0	0.16	34.32	0.75	3	4	892.7	0.13	32.80
1.25	2	1	714.3	0.16	36.33	1.00	3	2	900.3	0.14	36.14
1.00	3	2	719.7	0.16	36.47	1.25	1	2	962.3	0.15	35.63
1.25	1	2	732.7	0.17	35.51	1.25	2	2	1027.7	0.17	38.26
1.00	2	4	754.3	0.18	36.03	1.25	3	1	1065.0	0.18	36.61
1.25	2	2	795.7	0.19	38.21	1.00	1	4	1070.3	0.18	34.29
1.25	3	1	896.3	0.23	37.34	1.50	1	1	1103.7	0.19	34.42
1.00	3	4	926.0	0.24	36.81	1.00	2	4	1125.0	0.19	35.88
1.50	1	1	973.0	0.26	34.64	1.50	2	1	1184.0	0.21	36.16
1.25	3	2	1019.0	0.28	39.63	1.25	3	2	1233.0	0.22	39.31
1.25	1	4	1050.7	0.29	35.70	1.00	3	4	1320.3	0.24	36.42
1.25	2	4	1114.3	0.31	38.50	1.50	1	2	1344.0	0.25	36.51
1.50	2	1	1145.7	0.33	36.56	1.50	3	1	1395.7	0.26	36.96
1.75	1	1	1167.0	0.33	34.65	1.50	2	2	1417.3	0.27	40.19
1.50	1	2	1199.7	0.35	36.31	1.75	1	1	1541.3	0.30	34.43
1.75	2	1	1255.7	0.37	36.69	1.75	2	1	1589.3	0.31	36.24
1.50	3	2	1257.0	0.37	42.58	1.25	1	4	1602.3	0.31	35.80
1.50	2	2	1269.0	0.37	39.99	1.50	3	2	1666.0	0.33	42.17
1.75	1	2	1334.7	0.40	36.61	1.25	2	4	1667.7	0.33	38.51
1.50	3	1	1343.0	0.40	37.69	1.75	3	1	1816.3	0.37	37.12
1.25	3	4	1374.3	0.41	40.02	1.75	1	2	1817.7	0.37	36.89
1.75	2	2	1415.0	0.43	40.62	2.00	1	1	1893.7	0.38	34.70
1.75	3	1	1472.3	0.45	37.91	1.25	3	4	1894.3	0.39	39.63
2.00	1	1	1530.0	0.47	34.90	1.75	2	2	1902.0	0.39	41.15
1.75	3	2	1632.7	0.51	44.32	2.00	2	1	1938.3	0.40	36.83
2.00	2	1	1642.0	0.52	37.24	1.75	3	2	2129.0	0.44	44.15
1.50	1	4	1683.3	0.53	36.49	2.00	3	1	2194.0	0.46	37.96
1.50	3	4	1707.3	0.54	43.08	1.50	1	4	2247.7	0.47	36.68
1.50	2	4	1751.7	0.56	40.25	2.00	1	2	2264.7	0.48	37.57
2.00	1	2	1752.7	0.56	37.23	1.50	2	4	2324.7	0.49	40.46
2.00	2	2	1854.3	0.60	42.86	2.00	2	2	2354.3	0.50	43.55
2.00	3	1	1884.0	0.61	38.76	1.50	3	4	2524.3	0.54	42.64
1.75	1	4	1955.3	0.63	36.76	2.00	3	2	2607.7	0.56	55.90
1.75	2	4	2034.0	0.66	40.85	1.75	1	4	3041.0	0.67	37.04
2.00	3	2	2055.7	0.67	56.92	1.75	2	4	3120.0	0.69	41.41
1.75	3	4	2265.3	0.75	44.92	1.75	3	4	3389.3	0.76	44.77
2.00	1	4	2583.0	0.87	37.36	2.00	1	4	3880.0	0.88	37.73
2.00	2	4	2687.3	0.91	42.99	2.00	2	4	3969.7	0.90	43.81
2.00	3	4	2919.7	1.00	Infinite	2.00	3	4	4356.7	1.00	Infinite

표 1.4 시간에 대해 정렬된 데이터

의 화질 데이터는 그대로 사용할 수 있다는 것이 되므로 광원의 개수 파라미터가 렌더링 시간에 미치는 영향만을 예상하여 재계산해줄 수 있다면 위의 데이터들을 이용하여 서로 다른 수의 광원을 가진 데이터들에 대한 가중치 테이블

0.75	1	1	419.33	0.00	33.30	0.75	1	1	419.33	0.00	33.30
0.75	1	2	513.67	0.02	31.74	0.75	2	1	467.67	0.01	34.19
0.75	1	4	886.00	0.09	31.98	0.75	1	2	513.67	0.02	31.74
0.75	2	1	467.67	0.01	34.19	0.75	2	2	564.67	0.03	32.23
0.75	2	2	564.67	0.03	32.23	1.00	1	1	637.67	0.04	34.13
0.75	2	4	934.33	0.10	32.52	0.75	3	1	680.33	0.05	35.15
0.75	3	1	680.33	0.05	35.15	1.00	2	1	706.00	0.05	35.68
0.75	3	2	756.67	0.06	32.83	0.75	3	2	756.67	0.06	32.83
0.75	3	4	1135.33	0.13	33.15	1.00	1	2	810.00	0.07	34.12
1.00	1	1	637.67	0.04	34.13	1.00	2	2	868.00	0.08	35.73
1.00	1	2	810.00	0.07	34.12	0.75	1	4	886.00	0.09	31.98
1.00	1	4	1436.67	0.19	34.32	1.00	3	1	917.00	0.09	36.51
1.00	2	1	706.00	0.05	35.68	1.25	1	1	928.67	0.09	34.51
1.00	2	2	868.00	0.08	35.73	0.75	2	4	934.33	0.10	32.52
1.00	2	4	1495.67	0.20	36.03	1.25	2	1	1011.00	0.11	36.33
1.00	3	1	917.00	0.09	36.51	1.00	3	2	1081.00	0.12	36.47
1.00	3	2	1081.00	0.12	36.47	0.75	3	4	1135.33	0.13	33.15
1.00	3	4	1714.67	0.24	36.81	1.25	1	2	1192.00	0.14	35.51
1.25	1	1	928.67	0.09	34.51	1.50	2	1	1222.33	0.15	36.56
1.25	1	2	1192.00	0.14	35.51	1.25	3	1	1233.67	0.15	37.34
1.25	1	4	2154.00	0.32	35.70	1.50	1	1	1234.33	0.15	34.64
1.25	2	1	1011.00	0.11	36.33	1.25	2	2	1259.67	0.16	38.21
1.25	2	2	1259.67	0.16	38.21	1.00	1	4	1436.67	0.19	34.32
1.25	2	4	2221.00	0.34	38.50	1.25	3	2	1447.00	0.19	39.63
1.25	3	1	1233.67	0.15	37.34	1.50	3	1	1448.33	0.19	37.69
1.25	3	2	1447.00	0.19	39.63	1.50	1	2	1488.33	0.20	36.31
1.25	3	4	2414.33	0.37	40.02	1.00	2	4	1495.67	0.20	36.03
1.50	1	1	1234.33	0.15	34.64	1.50	2	2	1565.67	0.21	39.99
1.50	1	2	1488.33	0.20	36.31	1.00	3	4	1714.67	0.24	36.81
1.50	1	4	2812.00	0.45	36.49	1.75	1	1	1915.67	0.28	34.65
1.50	2	1	1222.33	0.15	36.56	1.75	2	1	1923.00	0.28	36.69
1.50	2	2	1565.67	0.21	39.99	1.50	3	2	2075.00	0.31	42.58
1.50	2	4	2897.67	0.46	40.25	1.25	1	4	2154.00	0.32	35.70
1.50	3	1	1448.33	0.19	37.69	1.75	3	1	2160.33	0.32	37.91
1.50	3	2	2075.00	0.31	42.58	1.25	2	4	2221.00	0.34	38.50
1.50	3	4	3341.33	0.54	43.08	2.00	2	1	2234.67	0.34	37.24
1.75	1	1	1915.67	0.28	34.65	2.00	1	1	2257.33	0.34	34.90
1.75	1	2	2300.67	0.35	36.61	1.75	1	2	2300.67	0.35	36.61
1.75	1	4	4126.67	0.69	36.76	1.75	2	2	2389.00	0.37	40.62
1.75	2	1	1923.00	0.28	36.69	1.25	3	4	2414.33	0.37	40.02
1.75	2	2	2389.00	0.37	40.62	2.00	3	1	2504.00	0.39	38.76
1.75	2	4	4206.00	0.70	40.85	1.75	3	2	2625.33	0.41	44.32
1.75	3	1	2160.33	0.32	37.91	2.00	1	2	2776.67	0.44	37.23
1.75	3	2	2625.33	0.41	44.32	1.50	1	4	2812.00	0.45	36.49
1.75	3	4	4513.33	0.76	44.92	2.00	2	2	2854.33	0.45	42.86
2.00	1	1	2257.33	0.34	34.90	1.50	2	4	2897.67	0.46	40.25
2.00	1	2	2776.67	0.44	37.23	2.00	3	2	3159.67	0.51	56.92
2.00	1	4	5177.00	0.89	37.36	1.50	3	4	3341.33	0.54	43.08
2.00	2	1	2234.67	0.34	37.24	1.75	1	4	4126.67	0.69	36.76
2.00	2	2	2854.33	0.45	42.86	1.75	2	4	4206.00	0.70	40.85
2.00	2	4	5252.00	0.90	42.99	1.75	3	4	4513.33	0.76	44.92
2.00	3	1	2504.00	0.39	38.76	2.00	1	4	5177.00	0.89	37.36
2.00	3	2	3159.67	0.51	56.92	2.00	2	4	5252.00	0.90	42.99
2.00	3	4	5793.67	1.00	Infinite	2.00	3	4	5793.67	1.00	Infinite

표 1.5 광원이 3개인 경우에 대해 예측된 시간대별 정렬 데이터

을 모두 구성할 수 있을 것이다.

그럼 우선 하나와 두 개의 광원이 존재하는 데이터에 대한 가중치 테이블을

구성해 보자. 이는 위의 실험 결과를 렌더링 시간 비용으로 정렬한 후 필요로 하는 만큼의 구간을 나누어 해당 시간 구간에서 가장 높은 화질을 나타내는 옵션을 골라냄으로써 만들어질 수 있다. 즉 샘플링 구간에서 최대한 높은 화질의 옵션을 골라내서 가중치 테이블을 구성한 뒤 중간 값은 선형 보간법을 이용하여 구해내는 것이다. 표 1.4는 광원이 한 개인 경우(왼쪽 데이터)와 두 개인 경우(오른쪽 데이터)에 대해 렌더링 시간으로 정렬한 데이터이다. 이것은 렌더링 시간으로 구간을 나눈 뒤 해당 구간에서 가장 적절한 옵션 배열을 샘플링하여 가중치 테이블을 구성해 내기 위한 작업이다.

표 1.3을 보면 매 시간 비용 구간마다 최고의 화질을 갖는 옵션 배열과 떨어진 두 구간 사이를 보간해 줄 수 있는 값이 포함된 옵션 배열들에 대해 색이 칠해진 것을 볼 수 있다. 하지만 그 전 구간에 비해 화질이 높아진 옵션 배열이 없거나 해당 구간이 존재하지 않는 경우도 있다. 이 때에는 그 전 구간과 다음 구간의 값으로 적절히 보간해주어야 한다.

표 1.5는 표 1.3에서 구한 TimeDiff 값을 이용하여 광원의 개수가 3개인 경우에 해당하는 각 옵션 배열별 렌더링 시간과 이미지 화질 데이터를 재구성한 것이다. 즉 광원의 개수가 2개인 데이터들에 대한 옵션 배열별 결과치중에서 렌더링 시간에 해당 부분에 TimeDiff의 값을 더해줌으로써 광원의 개수가 3개인 경우에 대한 결과치를 예측해 낸 것이다. 실제로 실험에 사용된 이미지의 특성에 따라 이 예측은 다소 조율이 필요할 수 있으나, 이러한 간단한 실험을 통하여 일단 이러한 예측 방법이 어느 정도의 정확도를 보이는 지에 대해서 살펴볼 수 있을 것이다. 표 1.5의 왼쪽 부분은 정렬되지 않는 데이터이며 오른쪽은 표 1.4와 같이 렌더링 시간으로 데이터를 정렬한 것이다. 또한 이 데이터에는 표 1.4와 마찬가지로 구간별 최고 화질을 나타내는 옵션 배열에 대해 표시가 되어 있다.

표에서 주어진 데이터를 이용하여 최종적인 가중치 테이블을 구성하려면 중

구간	1개 광원 데이터			2개 광원 데이터			3개 광원 데이터		
	Opt 1	Opt 2	Opt 3	Opt 1	Opt 2	Opt 3	Opt 1	Opt 2	Opt 3
0.0	0.75	1	1	0.75	1	1	0.75	1	1
0.1	1.00	2	1	1.00	2	2	1.00	3	1
0.2	1.25	2	2	1.25	2	2	1.25	3	2
0.3	1.25	3	2	1.50	2	2	1.50	3	2
				1.50	3	2			
0.4	1.50	3	2				1.75	3	2
				1.75	3	2			
0.5	1.75	3	2				2.00	3	2
0.6				2.00	3	2			
0.7	2.00	3	2		②				
								③	
0.8				1.75	3	4			
		①							
0.9									
1.0	2.00	3	4	2.00	3	4	2.00	3	4

표 1.6 예비 가중치 테이블

간에 해당되는 값이 없는 부분이나 정확히 들어맞지 않는 값을 완전한 형태로 보간해 주는 작업이 필요하다. 표 1.6은 이 작업을 위한 테이블이다.

표 1.6은 전체 데이터를 10개의 시간 구간으로 나누어 정확히 해당하는 값이 있는 경우엔 해당 옵션 배열이 명시되어져 있고 그렇지 않을 경우엔 비워져 있다. 이러한 빈칸들은 적절한 보간법을 이용하여 내용을 채워야 하는데 이때에는 다음의 규칙이 있다. 보간된 옵션 배열을 이용한 렌더링에서 전체 렌더링 시간과 이미지의 화질은 전 구간과 후 구간의 시간과 화질값 범위 사이에 들어어야 하며 특히 렌더링 시간은 전후 두 구간의 중간 값에 근접할수록 좋은 보간 값이라는 것이다. 그리고 각 옵션이 해당 범위를 넘어서서는 안 된다.

표 1.6에서 짙은 블록으로 표현된 부분은 보간이 까다로운 부분이며 나머지 비워진 부분은 옵션 2 와 3이 같기 때문에 옵션 1을 적절히 보간해 주면 된다. 앞 뒤 구간의 옵션 2와 3이 다를 경우에는 이산값(discrete value)을 보간해주어야 하므로 조그마한 변화에도 렌더링 시간과 화질에 많은 변화가 생기게 된다. 따라서 이러한 경우에는 보간에 많은 신경을 써야 하는데 이중 ①의 경우는 적절한 보간값이 존재하지 않는다. 각 옵션값은 해당 범위를 넘어서서는 안되기 때문에 모두 전 구간의 배열을 그대로 사용하였다. 그리고 ②와 ③의 경우에는 전 구간의 옵션 2와 3을 적절히 변화시키면서 옵션 1의 값을 중심으로 보간해주었다. 이때 옵션 1의 변화에 따른 렌더링 시간의 변화에 초점을 맞추어 절대 다음 구간에서의 렌더링 시간을 초과하지 않도록 해야 한다. 이렇게 비어있는 값을 모두 보간해 놓은 완성된 가중치 테이블을 표 1.7에 나타내었다.

광원의 개수에 따라 최종 가중치 테이블의 구성이 다소 다른 것을 볼 수 있는데 광원이 많을수록 옵션 3의 변화에 민감해지는 것을 볼 수 있다.

(4) 실험과 결과 분석

실험은 하나의 입력을 파라미터로 받아들여 그것을 적절한 옵션들의 배열로

구간	1개 광원 데이터			2개 광원 데이터			3개 광원 데이터		
	Opt 1	Opt 2	Opt 3	Opt 1	Opt 2	Opt 3	Opt 1	Opt 2	Opt 3
0.0	0.75	1	1	0.75	1	1	0.75	1	1
0.1	1.00	2	1	1.00	2	2	1.00	3	1
0.2	1.25	2	2	1.25	2	2	1.25	3	2
0.3	1.25	3	2	1.50	2	2	1.50	3	2
0.4	1.50	3	2	1.65	3	2	1.75	3	2
0.5	1.75	3	2	1.87	3	2	2.00	3	2
0.6	1.87	3	2	2.00	3	2	1.60	3	4
0.7	2.00	3	2	1.70	3	4	1.70	3	4
0.8	2.00	3	2	1.75	3	4	1.80	3	4
0.9	2.00	3	2	1.87	3	4	1.90	3	4
1.0	2.00	3	4	2.00	3	4	2.00	3	4

표 1.7 완성된 가중치 테이블

	Chess1	Chess2	Sphere3
최종 이미지 해상도	640 x 480	640 x 480	640 x 480
광원 개수	1	2	3
다각형 데이터 개수	25231	25231	2701

표 1.8 실험 데이터 환경

바꾼 후 각각에 대해서 전체 렌더링 시간과 최종 이미지 화질을 구해보는 식으로 진행하였다. 이때 두 가지의 실험을 하는데, 하나는 하나의 입력을 받아들여 특정 옵션의 배열로 바꿀 때 일률적인 선형 가중치(linear weighting value)를 사용하는 실험이고 나머지 하나는 앞에서 구한 가중치 테이블을 이용하여 바꾼 옵션들의 배열로 실험을 하는 것이다.

실험에 사용된 데이터는 총 3개(표 1.8)인데, 광원이 각각 1개와 2개인 Chess1, Chess2 데이터와 광원이 3개인 Sphere3 데이터가 그것이다. 특히 Sphere3 데이터는 3개의 광원을 가지는데, 이는 위에서 예측된 3개 이상의 광원 데이터에 대한 가중치 테이블이 정확하게 동작하는가를 실험하는데 사용될 것이다. 그리고 각각의 데이터들이 가중치 테이블을 이용하여 렌더링하였을 때 일률적인 선형 가중치를 사용한 결과와 어떠한 차이를 보이는지를 그래프로 설명할 것이다.

Sphere 데이터는 2차 곡면으로 이루어진 데이터이므로 특별히 다각형화 작업을 하여야만 한다. 하지만 그리 많지 않은 데이터에 대해선 그 시간비용이 크지 않다. 따라서 본 실험에서는 다각형화 작업에 드는 시간 비용을 무시하였다.

(가) 실험 결과

그림 1.33, 그림 1.34 그리고 그림 1.35그래프는 0에서 1사이의 사용자 입력값 하나만을 받아들여 해당 값에 대해 두 번의 렌더링, 즉 선형 가중치를 적용한 렌더링과 본 연구에서 설계한 가중치 테이블을 이용한 렌더링을 한 후 그

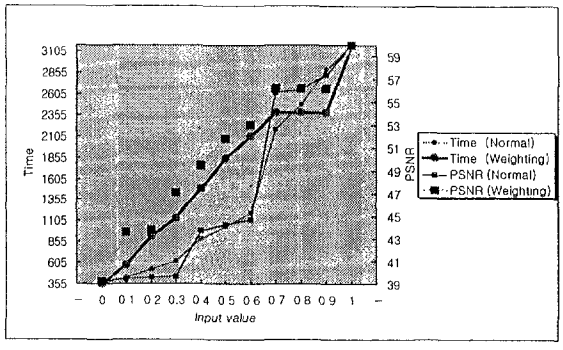


그림 1.33 Chess1 데이터의 실험 결과

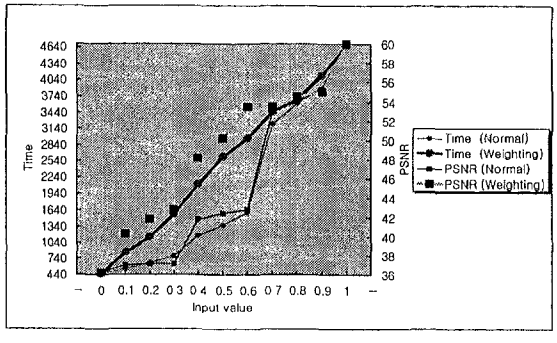


그림 1.34 Chess2 데이터의 실험 결과

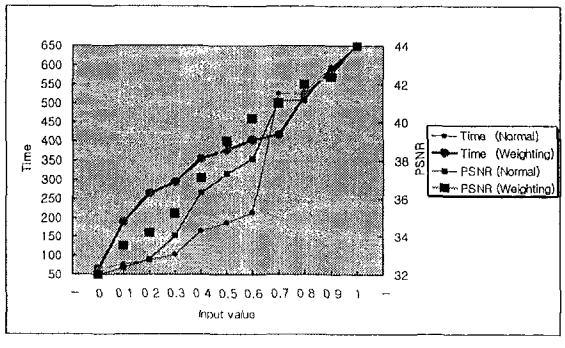


그림 1.35 Sphere3 데이터의 실험결과

결과를 렌더링 시간 비용에 대한 그래프로 나타낸 것이고, 그림 1.36, 그림 1.37 그리고 그림 1.38은 이에 대한 최종 영상을 나타낸 것이다.

가로축은 0에서 1까지 모두 11개의 구간으로 나누어져 있으며 각 가로축마다 모두 4개의 값을 가지는데, 선형 가중치를 적용하였을 때의 렌더링 시간과 이

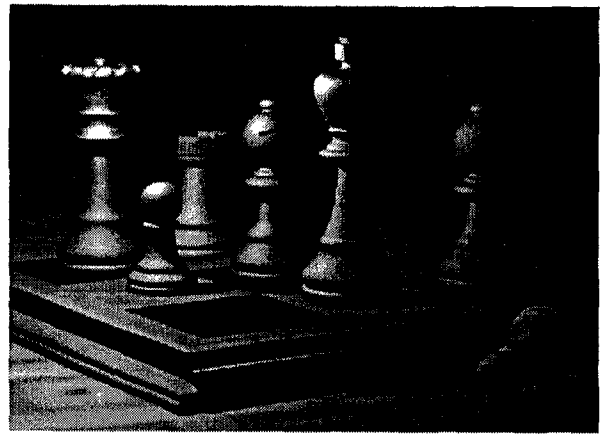


그림 1.36 Chess1 데이터의 최종 이미지

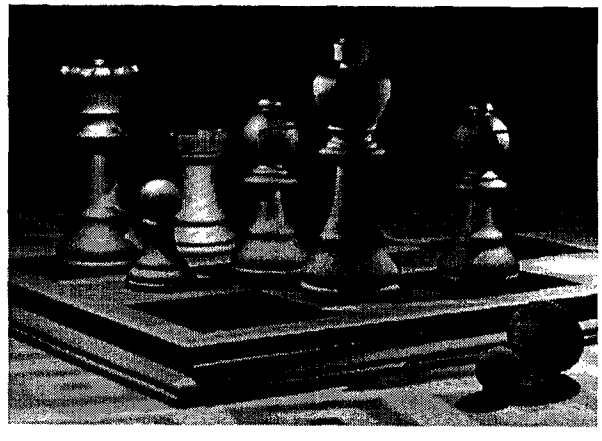


그림 1.37 Chess2 데이터의 최종 이미지

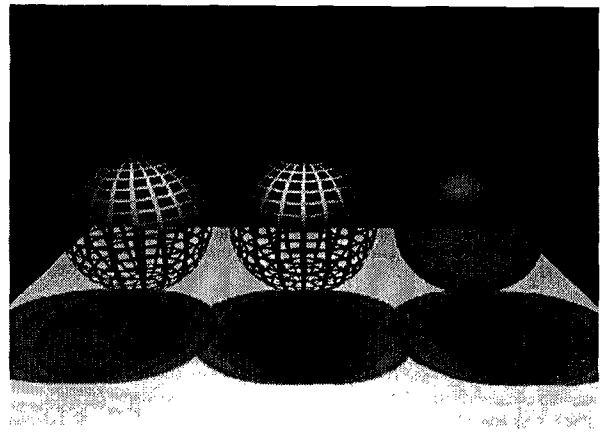


그림 1.38 Sphere3 데이터의 최종 이미지

미지 화질, 그리고 가중치 테이블을 이용하였을 때의 렌더링 시간과 이미지 화질이 그것이다. 범례에 표시한대로 두 벌의 시간 비용과 이미지 화질이 그래프에 나타나 있는데 얇은 선이 일률적인 선형 가중치에 대한 결과(Normal)이며 굵은 선으로 표현된 그래프가 가중치 테이블을 이용한 결과(Weighting)이다.

(나) 결과의 비교 분석

실험에 사용된 데이터에 선형적인 가중치를 적용하여 렌더링된 결과는 그 렌더링 시간과 화질이 예측할 수 없을 정도로 심한 변화를 보인다. 하지만 본 연구에서 제시한 가중치 테이블을 이용하여 렌더링한 결과는 대체로 예측 가능한 선형적인 증가 추세를 보여준다. 그리고 각 구간마다 해당 시간을 최대한 효율적으로 이용하여 최대의 화질 얻어내는 방향으로 그래프가 진행되는 것을 볼 수 있다.

여기서 주목할 결과는 세 번째 그래프인 그림 1.35의 Sphere3 데이터에 대한 결과치이다. Sphere3 데이터에 대한 실험은 전혀 사전 예비 측정 없이 이루어진 것으로써 단지 앞의 두 가지 데이터의 결과를 이용하여 예측된 가중치 테이블을 이용하였을 뿐이다. 하지만 비교적 선형적인 렌더링 시간 증가 추세를 보여주고 있으며 화질 또한 만족할만하다. 하지만 앞의 결과치들에 비해 화질 변화의 경향이 다소 예측하기 어려우며 작은 변화를 많이 일으키는 것을 볼 수 있다.

(5) 각 옵션들의 상호 연관성 분석

옵션들 중에는 서로 연관이 있는 것들이 존재한다. 이 것은 옵션들끼리 서로 영향을 미치는 관계를 의미하며 이 연관성을 분석해 보는 것은 옵션들의 특성을 보다 완벽히 이해하여 보다 활용성이 높은 가중치 테이블의 구성을 가능케 한다.

표 1.7의 가중치 테이블을 위한 실험 결과를 살펴보면 데이터 값들이 일정한

형태로 변하는 것을 알 수 있다. 즉 옵션2(픽셀당 샘플링 점의 개수)가 증가할수록 광원에 대한 영향은 줄어들고 옵션3(하나의 세이딩에 따르는 그림자 Z 버퍼 샘플링 점의 개수)이 증가할수록 광원에 대한 영향은 늘어남을 알 수 있다. 그리고 옵션1(미세 다각형의 조밀도)에 대해서는 옵션 3의 높고 낮음에 따라 값이 상대적으로 변함을 알 수 있다. 즉 옵션 3이 증가할 경우 옵션 1의 광원에 대한 영향은 증가하게 된다. 이것은 또 옵션 2와도 관련이 있는데 샘플링 점의 개수(옵션 2)가 증가할수록 미세 다각형의 조밀도(옵션 1)의 광원에 대한 영향이 증가함을 알 수 있다.

그림자를 만들기 위한 계산은 각 미세 다각형에 대해 그림자 Z 버퍼 샘플링 점의 개수만큼 수행된다. 따라서 광원의 개수가 늘어나서 그림자 계산이 많아지면 질수록 수행 시간이 늘어나게 되며 이는 미세 다각형의 개수에 비례하고 그림자 Z 버퍼 샘플링 점의 개수에 비례한다. 따라서 미세 다각형의 개수와 그림자 Z 버퍼 샘플링 점의 개수는 그림자 계산에 따르는 시간 비용측면에서 서로 연관성을 가지게 되는 것이다.

샘플링 점의 개수는 이론적으로 여타 파라미터와 옵션에 대해 큰 영향을 받지 않는다. 단지 미세 다각형의 개수가 증가할수록 샘플링 점의 개수의 변화에 따라 샘플링 작업 시에 해당 미세 다각형 속에 어떠한 샘플링점이 속해 있는지를 비교, 검사해보아야 하므로 시간 비용에 차이를 보이게 되는 것이다. 즉 미세 다각형의 개수와 샘플링 점의 개수 사이에는 약간의 상호 연관성이 있다 하겠다.

또한 본 연구에서의 실험에는 옵션으로 선택하지 않았지만 그림자 Z 버퍼의 크기 옵션 또한 몇몇 파라미터에 대한 영향을 받는다. 그림자 Z 버퍼의 크기는 그림자 Z 버퍼의 샘플링 작업과는 시간 비용 측면에서 무관하며 단지 렌더링하여야할 전체 다각형의 수가 증가함에 따라 조금씩의 시간 비용 증가 추세를 보이게 된다. 왜냐하면 그림자 Z 버퍼의 크기가 클수록 버퍼의 생성 시에 개별

다각형에 대해 처리하여야 할 버퍼내의 영역이 커지기 때문이다.

(6) 향후과제

본 연구에서는 몇 가지의 데이터에 대해 각 옵션별로 몇 가지 측면에서 실험을 해봄으로써 새로운 가중치 테이블을 설계하였다. 실제로 이 테이블을 이용한 렌더링은 일반 선형적인 가중치 부여에 의한 렌더링보다 훨씬 나은 성능을 보여주었다. 선형적인 시간 비용 증가에 따라 렌더링 시간의 예측을 가능하게 했으며 각 구간별로 최대한의 화질을 보이도록 하였다. 하지만 아직 몇 가지 문제점을 지니고 있는데, 가중치 테이블을 이용한 렌더링에서 생각해볼 수 있는 문제는 대략 다음 네 가지로 요약된다.

첫째, 몇 개의 샘플 데이터를 이용한 실험 결과를 이용하여 만든 가중치 테이블로 대부분의 데이터들을 효과적으로 처리할 수 있을 것인가에 대한 문제를 생각해 볼 수 있다. 이 문제는 사실 중요한 문제이긴 하나 실제로 본 연구에 언급되지 않는 여러 가지 실험들을 통해 볼 때 대부분의 데이터들은 서로 비슷한 시간 비용과 이미지 화질의 비율을 나타냄을 알 수 있었다. 본 연구에서는 3개의 데이터를 하나의 집합으로 하여 실험하였지만 실제로 보다 많은 데이터로 실험하는 것이 바람직할 것이며 그로써 보다 효율적인 결과를 얻을 수 있을 것이다. 하지만 대부분의 데이터를 위한 가중치 테이블은 각각의 파라미터에 대한 많은 연관성을 생각하지 않을 수 없고, 결국엔 엄청난 양의 테이블로 서로 얽혀버릴 수 있다. 따라서 본 연구에서는 모든 데이터에 대해 보다 정확한 값을 돌려주는 결과보다는 보편적인 데이터 구조에 알맞은 형태로 가중치 테이블을 구성하였다 할 수 있겠다.

둘째, 선형 보간법을 이용한 중간값 계산이 까다로울 수 있으며 경우에 따라선 잘못된 결과를 되돌릴 수도 있다. 이것은 각 옵션들의 이산적인 특성과 결과가 테이블의 형태로 만들어졌다는 점 때문이다.

셋째, 가중치 테이블 구성을 위한 실험에서 제한된 수의 옵션 샘플을 이용함

으로써 각 시간 비용 구간에서 재샘플할 데이터들의 수적인 한계가 생길 수 있다. 즉 이로 인해, 제한된 폭의 결과 값만을 얻어낼 수밖에 없다.

넷째, 가중치 테이블을 하나의 함수로 표현하기가 거의 불가능하다는 것이다.

이들 문제는 대부분 실험 데이터의 제한적인 요소와 옵션의 샘플의 수적인 제한이 그 원인이다. 따라서 보다 정확한 결과를 얻기 위해서는 보다 다양한 데이터들에 대한 실험이 필요하며 보다 다양한 측면에서 옵션들의 상호 연관성을 연구하여야 할 것이다.

제 2 절 렌더링 시스템 기능의 개선, 확장 및 사용예

1. 사용자 인터페이스

본 렌더링 시스템은 그래픽적으로 합성된 이미지를 만드는데 있어서 많은 장점을 가진 RenderMan Interface[2-1]의 다양한 기능들을, 사용자가 쉽게 사용할 수 있도록 돕는 편리하고도 강력한 사용자 인터페이스를 제공한다. 만약 사용자 인터페이스의 도움 없이 기존의 RenderMan Interface를 사용하여 이미지를 만들고자 하면 사용자는 이 RenderMan Interface를 지원하는 렌더러를 위한 입력 파일을 직접 만들어야 한다. 이 입력 파일은 사용자가 만들고자 하는 이미지를 위한 전경에 대한 묘사이다. 즉, 카메라의 위치와 방향, 광원에 대한 설정, 전경 내에 배치된 물체들에 대한 모든 묘사들이 이 입력 파일 내에 들어가게 된다.

그러나 자신이 원하는 전경을 그대로 묘사하는 입력 파일을 만들기란 그리 쉬운 일이 아니다. 여기서 입력 파일 내에는 물체나 광원의 위치와 그들의 속성을 지정하기 위해 여러 수치 값들이나 지정어들이 들어가야 한다. 그러나 이런 값들은 그리 직관적이지 않다. 예를 들어 카메라의 방향을 임의의 물체를 중심으로 놓고 또 다른 물체까지 보이도록 하고 싶을 때 정확히 카메라를 배치하기 위해서 사용자는 3차원적인 변환의 계산과 상상력등을 동원해야 한다. 그리고 자신이 계산한 변환이나 속성들이 알맞게 되었는지를 알아보기 위해서는 렌더러를 통해서 최종적인 이미지를 만들어 볼 수밖에 없다. 그런데 최종적인 이미지를 만드는 과정은 매우 많은 시간을 요구할 수도 있기 때문에 이런 과정은 문제가 많다. 만약 최종 이미지가 맘에 들지 않는다면 사용자는 입력 파일을 다시 수정하고 다

시 렌더링하는 과정을 반복해야 하기 때문이다. 이러한 문제점 때문에 고급 렌더링 시스템에 있어서 편리한 사용자 인터페이스는 매우 필수적인 요소이다. 본 렌더링 시스템에서 제공하는 사용자 인터페이스를 통해, 사용자는 자신이 원하는 전경을 화면을 통해 보면서 실시간으로 전경을 직접 구성하고 편집할 수 있다. 그리고 화면을 통해 보이는 전경 그대로가 최종 이미지 그대로의 구도이기 때문에 사용자는 최종 이미지에 대한 구상을 쉽게 할 수 있다. 또한 본 렌더링 시스템은 RenderMan Interface의 가장 큰 장점 중의 하나인 셰이더를 효율적으로 관리, 이용할 수 있는 인터페이스까지 지원한다.

본 렌더링 시스템의 인터페이스는 크게 다음과 같은 3가지 구조로 나뉘어져 있다. 이는 처음에 본 렌더링 시스템이 실행되면서 생성되는 윈도우에 나타나는 3가지 메뉴이기도 하다.

- Scene Manipulation

사용자가 만들고자 하는 이미지를 위한 전경을 구성할 수 있도록 도와주는 기능들이 있으며 본 렌더링 시스템의 가장 주요한 기능이다. 여기서는 새로운 전경을 생성하거나 파일로 저장된 기존의 전경을 읽어들여서 편집할 수 있으며 편집된 전경을 다시 저장할 수 있다. 또한 사용자가 원하는 전경을 이루기 위한 여러 형태의 기하학적 물체 데이터를 이용할 수 있다.

- Image Viewer

본 고급 렌더링 시스템에서 생성되는 이미지 파일의 포맷은 TIF이다. 이 TIF 이미지 파일 포맷은 널리 알려진 것이기 때문에 이 파일을 볼 수 있는 여러 그래픽 도구들이 개발되었으나 여기서는 사용자의 편리를 도모하기 위해 이 기능을 본 렌더링 시스템에 추가 시켰다. 여기서는 단순히 이미지를 보는 것이 아니라 좀 더 관심있는 부분을 선택해서 자세히 보는

기능도 제공한다.

- Shader Database

기본적으로 본 렌더링 시스템은 RenderMan Interaface에서 도입된 개념인 세이더를 지원한다. 사용자는 이 세이더를 이용해서 자신이 원하는 물체의 특성을 표현할 수 있다. 그러나 이 세이더는 매우 다양하고 융통성이 높기 때문에 이를 관리하고 효과적으로 사용하기 위해서는 세이더를 위한 데이터 베이스가 필수적이다. 여기서 제공하는 데이터 베이스는 사용자가 쓰고 있는 세이더의 관리와 새로운 세이더의 등록과 삭제등의 기능들을 지원한다. 사용할 수 있는 세이더 타입으로는 6가지, surface, deformation, volumn, light, displacement imager shader가 있다.

기본적으로 본 렌더링 시스템은 실리콘 그래픽스 워크스테이션 호환 기종을 위해 제작되었다. 화면을 통해 보이는 전경은 OpenGL 그래픽 라이브러리를 이용하였고 창의 생성과 각 종 기능들을 제공하기 위해서 MOTIF 라이브러리를 이용하였다.

사용자가 전경 조작창을 누르면 초기 창이 생성되고 맨 위의 메뉴바가 있다. 전경 조작을 위한 기능들과 자주 사용될 것으로 예상되는 기능들은 옆의 사이드 메뉴로 끌어 내었다. 또한 맨 밑에는 정보 안내 영역이 있어서 이 창의 기능들을 이용하는 동안에 수시로 여러 가지 유용한 정보들을 제공한다. 먼저 맨 위의 메뉴 바에는 File, Item, Option, Environment, Etc등의 메뉴들이 보이는데 이 메뉴들은 다음에서 설명하는 바와 같이 여러 서브 메뉴로 구성되어 있다.

가. File 메뉴

(1) New

새로운 scene를 만들기 위하여 모든 item들을 없애고, 왼편의 모든 인

터페이스를 초기화한다.

(2) Load

이미 저장된 다른 scene을 읽어온다. 이때 읽어올 수 있는 파일은 SERT의 고유 format인 *.scn과 *.asc이다. *.scn은 binary file이고, *.asc는 ascii file이다. load를 한 뒤에는 반드시 close 버튼을 눌러야 Load 창이 닫힌다.

(3) Save

현재의 scene를 저장한다. 이 때, 저장할 수 있는 file format은 *.scn과 *.asc가 있다.

(4) Save As

다른 이름 혹은 다른 포맷으로 저장하기.

(5) Quit

모든 작업을 종료하고 SERT의 Scene Manipulation을 닫는다. 이때, 저장을 하지 않았더라도 절대로 물어보는 일이 없으므로 주의하여야 한다.

나. Item 메뉴

여기서는 사용자가 작업하는 도중에 선택한 물체 목록에 대한 여러 조작을 가능하게 해주는 기능들이 들어있다. 그러나 여기에 있는 기능들은 물체 목록 단위에 대한 조작이므로 더 세부적으로 물체 목록을 구성하는 물체에 대한 조작은 사이드 메뉴에 있는 Item Manipulation 버튼을 이용하여야 한다.

(1) Insert

현재의 scene에 새로운 item을 삽입한다. 삽입할 item은 SERT 고유의 *.itm file이어야 한다. *.itm은 한 item 만을 저장한 file이다. 삽입한 item은 현재 scene의 원점에 삽입된다.

(2) Label

현재 선택되어 있는 item의 이름(Label)을 바꾼다. 같은 이름의 item이 있을 경우, SERT에서 자체적으로 바꾸어주는 system이 없어서 같은 이름의 item이 여러개 있을 수 있으므로 복잡한 scene에서 작업할 때에는 혼동되기 쉽다. 따라서 이 Label 기능을 이용하여 직접 바꾸어 주어야 한다.

(3) Copy

현재 선택되어 있는 item을 복사한다. 복사하면 그냥 현재의 위치에 있으므로 복사가 되었는지를 확인하려면 선택되어 있는 item을 움직여서 확인하거나 Item - Select by Name을 실행시켜서 같은 이름의 item이 여러개 인가를 확인하면 된다.

(4) Delete

현재 선택되어 있는 item을 삭제한다.

(5) Focus

현재 선택되어 있는 item으로 focus를 맞춘다. 즉, 화면의 정 중앙에 선택된 item이 오게 한다.

(6) Area Light

현재 선택된 item을 Area Light로 만든다. 일단 Area Light로 만들면 다시 item으로 복구를 할 수 없다. Area Light로 만들면 Item - Select by Name에서 해당되는 item이 없어지고 Light가 추가되어 있는 것을 볼 수 있다. 생성된 Area Light는 Environment - Light에서 속성을 수정할 수 있다.

(7) Set Position

선택된 item의 위치, rotate, scale 등의 값을 변경한다.

(가) New Position

선택된 item를 설정된 새로운 위치로 옮긴다.

(나) Add Position

New Position의 위치에서 설정된 값만큼 이동시킨다.(offset 개념)

(다) Add Rotation

선택된 item을 u, v, n의 방향으로 회전시킨다. (정확한 숫자 값만큼 회전)

(라) Add Scale

선택된 item을 설정된 값만큼 scaling한다.

(8) Select by Name : 각 item의 이름으로 Item을 선택한다.

다. Option 메뉴

(1) Camera

Camera에 관련된 여러 가지 option을 제어한다. 여기서 바뀌는 값에 따라 Camera에 잡히는 scene의 크기, 방향, 위치 등이 바뀌어 Rendering되는 image에 영향을 준다.

(가) Resoultion

Rendering되는 Image의 수평, 수직 해상도를 결정한다.

(나) Pixel Aspect Ratio

각 pixel의 Aspect Ratio를 결정한다.

(다) Frame Aspect Ratio

Image의 전체 frame의 Aspect Ratio를 결정한다.

(라) Crop Window

실제로 Rendering될 영역을 결정한다. 전체 영역 중 일부분만을 렌더링 할 수 있다.

(마) Screen Window

그려질 영역의 스크린 좌표계를 설정한다.

(바) Camera Projection

화면으로의 투영방법을 결정한다. 평행 투영과 원근 투영이 있다.

(사) Clipping Plane

절단평면의 위치를 결정한다. 이 값을 잘 조절함으로서 관심밖의 멀리 있는 물체들을 고려 대상에서 제외함으로서 효율적인 렌더링을 할 수 있다.

(아) Depth of Field

초점 거리와 렌즈의 특성 등을 정의한다. 이 기능을 잘 이용하면 관심 있는 물체만 또렷하고 다른 물체들은 흐리게 나타나게 하는 효과 등을 연출할 수 있다.

(2) Display

카메라에 의해 만들어진 실세계의 영상을 디스플레이하는 과정에서 필요한 여러 인수들이 있다.

(가) Filtering

필터 함수와 주위의 참조 화소의 수를 결정한다.

(나) Pixel Variance

실제 화소값에서의 추정된 계산값을 지정한다.

(다) Sampling Rates

수직, 수평의 효과적인 샘플링 비율을 지정한다.

(라) Exposure

개인(gain)과 감마값을 결정한다.

(마) Color Quantizer

색과 불투명도의 양자를 조절

(바) Depth Quantizer

깊이 정보의 양자를 조절

(사) Display Type

출력을 파일로 할 것인가 화면으로 할 것인가를 지정

(아) Display Mode

GRB, GRBA, Z, RGBZ 형식 중에서 하나를 지정

라. Light 메뉴

이 기능은 Environment 메뉴안에 있는 Light 기능을 선택함으로서 이용할 수 있다. 최종 이미지의 사실감은 광원의 효율적인 사용과도 매우 밀접한 관련이 있다. 이는 광원이 물체의 색과 그림자등, 현실감을 주는 요소들에 매우 큰 영향을 미치기 때문이다. 본 렌더링 시스템은 이런 요구를 만족시키기 위해 다양한 형태의 광원과 기능들을 지원한다.

(1) Default Light

(가) Create

두 가지 형태의 광원을 지원한다. 하나는 표준 광원, 또 하나는 광원 셰이더를 부여하는 특수한 형태의 광원이다. 그림 2.1의 윈도우에서 Create버튼을 선택하면 그림 2.3과 같은 Control Panel이 뜬다. 이것은 표준 광원으로 4가지의 광원 (ambientlight, distantlight, pointlight, spotlight)이 생성 가능하다. 광원들의 모습이 그림 2.2에 나타나 있다. 어떤 광원을 생성하는가에 따라 사용 가능한 옵션들이 달라진다.

1) Ambient Light

전경 전체에 일정한 값의 밝기를 주는 광원

2) Point Light

한 점에서 사방으로 퍼지는 광원

3) Distant Light

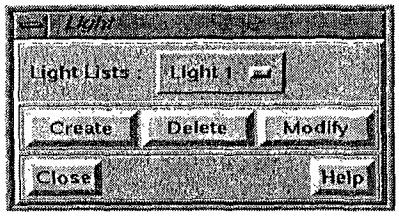


그림 2.1 light가 생성된 경우의 윈도우

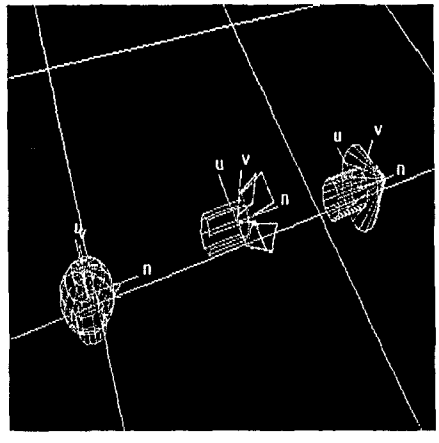


그림 2.2 광원들 (왼쪽부터 pointlight, spotlight, distantlight)

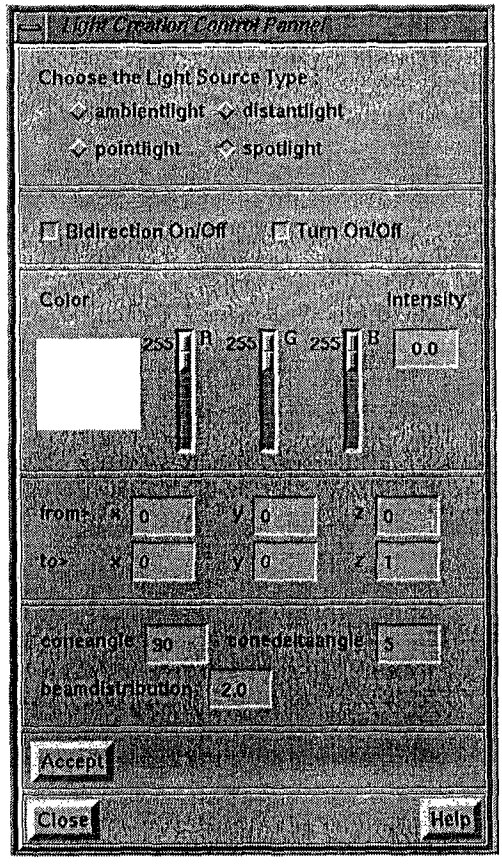


그림 2.3 Light create control panel

광원이 위치를 가지지 않고 일정한 방향으로 평행으로 뻗어오는 빛의 경우로 광원이 물체로부터 아주 멀리 떨어져 있는 경우가 이에 해당한다.

4) Spot Light

무대 조명같이 특정 부분에 강한 빛을 비추고 주위로 나갈수록 급격히 어두어지는 빛

그림 2.3의 Light create control panel의 파라메터들에 대한 설명은 아래에 기술한다.

● Bidirection On/Off - distantlight의 경우 빛의 방향이 단방향인지 양방향인지 결정한다.

● Turn On/Off - 생성된 광원을 켜거나 끈다.

● Color - 광원의 색을 조정한다.

● Intensity - 광원의 세기를 조정한다.

● from, to - 광원의 위치와 방향을 결정한다.

● coneangle, conedeltaangle, beamdistribution - spotlight에서 빛이 나아가는 각도, 반영정도, 쇠퇴정도를 조절해준다.

(나) Delete

현재 전경내에 존재하는 광원들 중에 하나를 선택하여 삭제하는 기능을 제공한다.

(다) Modify

현재 전경 내에 존재하는 광원들 중에 하나를 선택하여 광원의 색이나 세기등을 수정할 수 있는 기능을 제공한다.

(2) Light Shader의 생성, 삭제, 수정

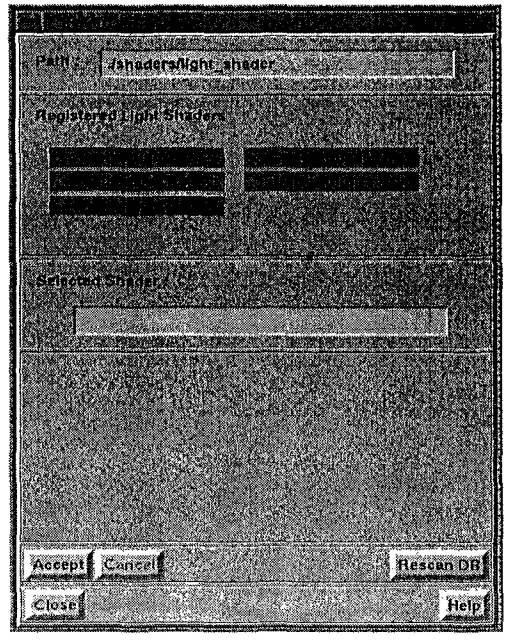


그림 2.4 Shader creation window

위에서 설명한 default light이외에도 user가 shading language로 작성한 light shader를 이용할 수 있다.

Environment 메뉴의 Light에서 shader를 선택, light의 생성, 삭제, 수정과정과 동일하다. 그림 2.4는 Shader 선택 윈도우이다. 여기에서 Registered light shaders중에서 한 shader를 클릭하면 selected shader에 그 shader가 나타나게되고 그 아래쪽에는 각 parameter들이 나타나 값을 수정 할 수 있다.

마. Etc 메뉴

여기서는 전경을 조작하고 이동하는데 도움이 되는 여러 세부 기능들을 제공한다.

(1) background색 바꾸기

Environment의 background로 가면 색을 조절할 수 있는 윈도우가 하나 뜬다. 여기서 지정한 RGB값이 전경조작창의 색이 된다(그림 2.5).

(2) 조작속도 조절

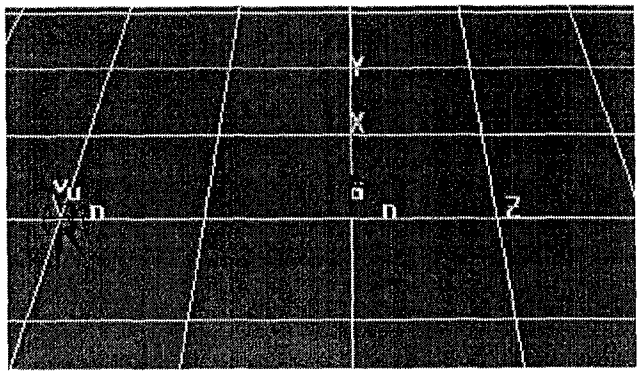


그림 2.5 푸른색으로 바뀐 전경 조작창

마우스의 dragging에 따라 전경조작창의 움직임이 너무 민감할 경우 조작속도를 조절할 수 있다. Etc의 moving speed로 가면 오른쪽과 같은 윈도우가 하나 뜨고 여기에서 moving factor 값을 작게 해 주면 움직임이 둔화되고 값을 크게 해 주면 움직임이 민감해 진다.

(3) Global scale값 조절

Etc의 Set Global Scale로 가면 그림 2.6과 같은 윈도우가 뜬다. 여기서 new scale ration의 x, y, z값을 조절해 줌에 따라 해당 축에 대한 scale값이 결정된다. 그림 2.7은 간단한 예제를 보여주고 있다.

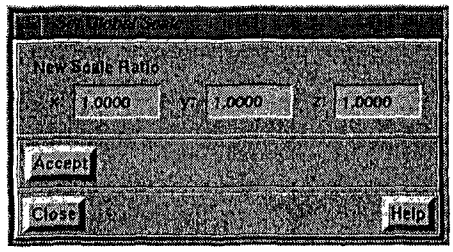


그림 2.6 Global scale factor 설정

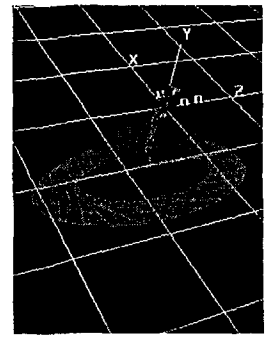


그림 2.7 x=y=1, z=2인 경우

(4) scene mode change

전경 조작창에 대한 각종 mode를 조정한다. Etc의 scene mode change로 들어가면 그림 2.8의 윈도우가 뜨고, check box를 on/off 시키느냐에 따라 여러 mode를 조정한다. item local rotate on/off의 경우 off가 default로 되어있는데 이것은 scene모드에서 scene전체를 rotate시킬 경

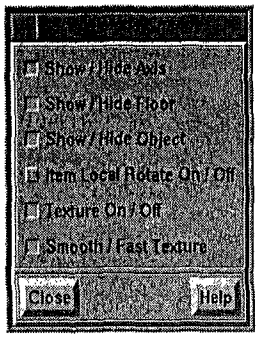


그림 2.8 Scene Mode change

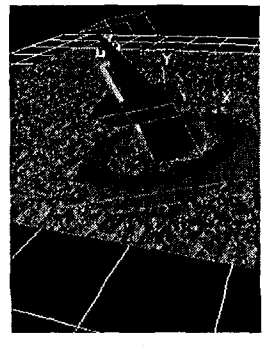


그림 2.9 Item local rotate on

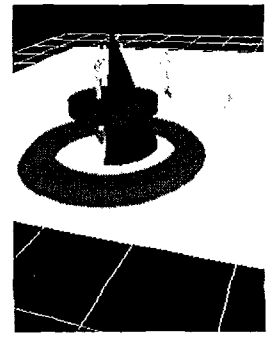


그림 2.10 Texture off

우 object역시 scene과 같이 rotate되지만 on으로 바꿀 경우 활성화된 object만 rotate된다(object모드에서 해당 object만 rotate시키는 효과)(그림 2.9). 그림 2.10은 텍스쳐가 off된 상태를 보여준다.

바. 사이드 메뉴

전경조작창 오른쪽에 보면 그림 2.11과 같은 전경 조작 도구들을 볼 수 있다. 이 도구들은 전경조작창에 여러 가지 조작을 해준다.

(1) Draw Mode

scene을 wireframe이나 shaded로 볼 수 있게 해주는 radio button으로 되어있다(그림 2.12, 그림 2.13).

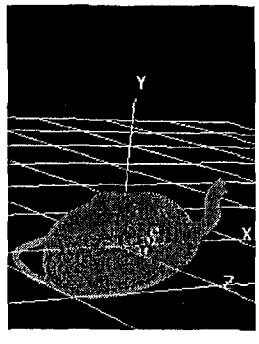


그림 2.12 wireframe

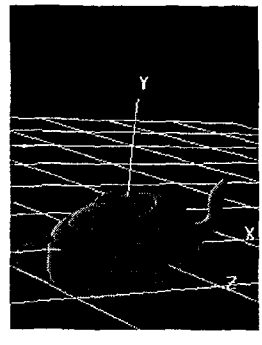


그림 2.13 shaded

(2) View Mode

scene을 perspective나 parallel로 볼 수 있게 해주는 radio button으로 되어 있다.

(3) camera/eye

scene을 볼 수 있는 도구에는 camera와 eye두개가 있으며 각각은 camera모양과 안경 모양의 object로 존재한다. camera/eye radio button중에 선택되는 view가 전경조작창에 나타나게 된다.

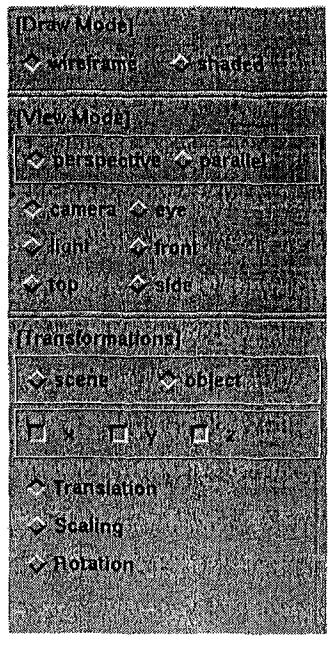


그림 2.11 사이드 메뉴

(4) front/top/side

scene을 front/top/side에서 볼 수 있다.

(5) Transformation

scene transformation / object transformation 의 두 가지가 있다.

(가) scene transformation

scene전체에 대해 translation, scaling, rotation을 가한다. default로 x,y,z축이 모두 on되어 있는데 off시키는 축에 대해서는 해당 transform이 일어나지 않는다.(rotation의 경우 x,y,z가 아닌 u,v,n이 나타나고 on/off에 대한 결과는 x,y,z에 대한 결과와 동일하다) anchored는, rotation이 eye중심으로 rotate하는 것과는 달리 select된 object를 축으로 rotate된다.

(나) object transformation

한 object에 대해 translation, scaling, rotation을 가한다.

(6) Render

전경조작도구 아래있는 Rendring button을 누르면 그림 2.14와 같은 윈도우가 뜬다. 렌더링 방법으로 raytracing과 z-buffer method가 이용가능하다.

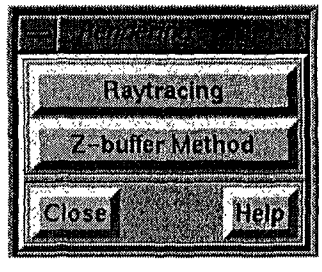


그림 2.14 Render 선택 윈도우

(가) Raytracing

raytracing으로 rendering한다. option은 그림 2.15와 같다.

(나) Z-buffer method

Z-buffer로 rendering한다. option은 그림 2.16과 같다.

OK를 클릭하면 렌더링 결과가 출력되는 윈도우가 뜬다. Rendering이 모두 끝난후 image를 RGB형식으로 save시킬 수 있다. Quick Save는 현재

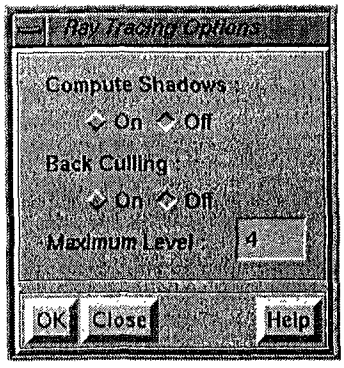


그림 2.15 RayTracing 설정 윈도우

filename(전경의 이름)으로 save한다.

(7) Item Manipulation

이 버튼은 현재 하나의 물체 목록이 선택된 상태에서만 유효해 진다. 이 버튼을 통해 물체 목록 조작창을 열어서 물체 목록에 대한 좀 더 세부적인 조작이 가능하다. 물체 목록 조작창의 기능들에 대한 자세한 설명은 뒤에서 하겠다.

(가) Attribute

1) Opacity & Color

현재 선택된 물체의 색과 불투명도를 지정한다.

2) Surface Shader

표면 세이더를 지정한다. 이 때에는 세이더 데이터베이스를 통해 등록된 세이더만 사용이 가능하며 세이더에 넘겨주는 여러가지 인수값들을 변경하여 사용자가 원하는 다양한 효과를 낼 수 있다.

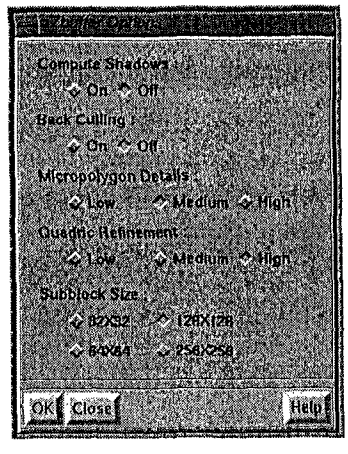


그림 2.16 Z-Buffer 설정 윈도우

3) Volume Shader

선택된 물체에 체적 세이더를 지정한다. 이것 역시 등록된 세이더만 가능하며 인수값을 변경할 수 있다.

4) Texture

선택된 물체에 2차원적인 텍스쳐를 입혀 사실감을 높이고 싶을 때에 사용한다.

5) Approximation

이 기능은 다각형 모델이 아닌 2차 함수의 회전체나 패치들에 대해서만 유효하다. 본 렌더링 시스템은 이런 형태의 물체들을 다각형 모델로 근사화하여 전경에 보여주는데, 근사화하는 지수를 설정한다. 이 수치가 높을수록 본래의 모습에 가까워 지지만 다각형이 매우 많아져서 작업에 불편함을 초래할 수도 있다.

(나) Object

1) Label

현재 선택된 물체에 이름을 부여한다.

2) Delete

현재 선택된 물체를 아이템으로부터 제거한다.

3) Set Position

현재 선택된 물체의 위치를 값을 지정하여 조정할 수 있다.

4) Select by Name

현재 보고 있는 물체 목록에 포함된 물체들을 이름을 보고 선택할 수 있다.

5) Attach Item

현재 선택된 물체의 계층구조 아래에 새로운 물체를 삽입한다.

(8) Scene Manipulation

원하는 화면을 rendering 할 수 있게 한다. 화면과 카메라, 라이트 소스, 텍스쳐 등등, 주된 작업이 이루어 진다.

(가) RIB renderer

RIB (renderman-inferface-bytestream) file을 하나 혹은 여러 장을 한꺼번에 rendering할 수 있게 한다.

1) single RIB rendering

RIB 파일을 render 할 때 사용. 정적인 image를 rendering 할 때 주로

사용이 된다.

2) multiple RIB rendering

animation이나 여러장의 RIB file들을 동시에 처리해야하는 경우에 사용된다.

(나) Image Viewer

파일형식으로 저장된 여러 종류의 image를 보여주는 기능을 가진다. 창에서 Zoom in과Zoom out이 가능하고 부이미지를 다시 불러올 수도 있다.

(9) Shader database : 여러 type으로 지정되어있는 shader들을 등록(register)하거나 삭제하여 원하는 방식으로 shading을 할 수 있게 한다. 사용할 수 있는 shader type으로는 6가지, surface, deformation, volumn, light, displacement imager shader가 있다.

2. 타시스템과의 인터페이스

가. RIB 파일의 렌더링

본 렌더링 시스템은 RIB (renderman-inferface-bytestream) 파일에 대한 렌더링을 제공한다. 두 가지 모드에서 제공하는데 single RIB rendering과 multiple RIB rendering이다. single RIB rendering은 정적인 image를 렌더링하기 위한 하나의 RIB 파일에 대한 렌더링에서 주로 사용이 되고, multiple RIB rendering은 animation이나 여러장의 RIB file들을 일괄처리하여 처리해야하는 경우에 사용된다.

그림 2.17과 그림 2.18은 RIB 파일을 렌더링 한 예제를 보여준다. RIB 파일의 간단한 편집을 통하여 이미지의 해상도와 카메라의 위치등을 수정할 수 있다.

나. VRML 2.0에서 SERT으로의 파일 전환

그림 2.17 RIB 파일을 렌더링
시스템을 통해 광선추적법으로
렌더링한 예제1

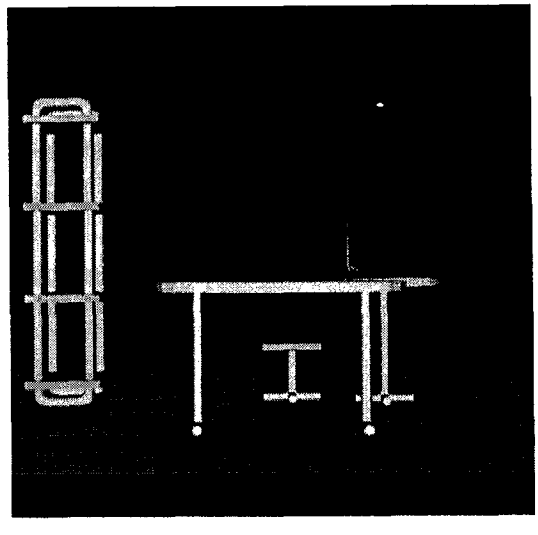


그림 2.18 RIB 파일을 렌더링 시스템을
통해 광선추적법으로 렌더링한 예제2

본 렌더링 시스템은 삼차원 시스템의 스크립트 표준으로 자리잡은 VRML 2.0 파일을 본 렌더링 시스템의 아스키 파일로 전환하는 전환 프로그램을 갖추고 있다. 삼차원 시스템 스크립트 파일은 기하학적인 모델의 정보, 모델의 변환 정보, 모델간의 계층 구조 정보, 모델의 속성(Material) 정

보, 라이트와 카메라의 위치, 방향 및 속성 정보, 그리고 VRML 2.0의 경우 사용자와의 상호작용을 위한 처리와 애니메이션 정보까지 포함하는 복잡한 구조를 갖는다. 각 렌더링 시스템에서 사용자 인터페이스와 렌더러를 만족하는 정보들을 위한 효율적인 자료 구조를 설계하는 일은 시스템의 일관성과 속도 측면에서 볼 때 핵심적인 작업이다. 그러므로 각 삼차원 시스템의 데이터의 전환에서는 효율성이 고려되어야 되며 각 시스템의 특수한 사항들에 대한 전환중의 손실을 최소화 혹은 에뮬레이션이 되어는 융통성을 가져야 한다. 본 시스템의 전환시스템은 위에서 언급된 효율성과 융통성을 만족시키는 프로그램이다.

(1) 렌더링 시스템의 파일 구조

본 렌더링 시스템의 개략적인 파일 구조가 그림 2.19와 같다. 각 부분은 시작 태그와 종료 태그로 그 블록을 명시적으로 표기한다. 각각의 태그는 다음과 같다.

(가) 물체 목록

물체 목록이라는 것은 전경 조작창을 통해서 편집될 수 있는 기본 단위를

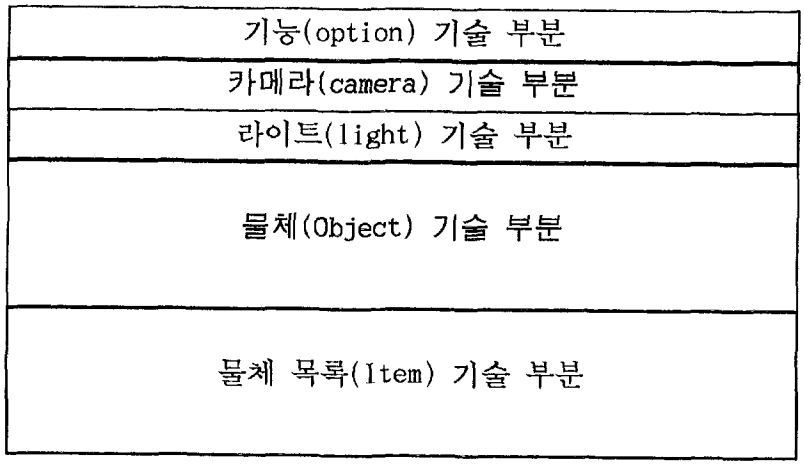


그림 2.19 본 렌더링 시스템 파일 구조

말한다. 예를 들어서 사용자가 시골 풍경을 전경으로 편집하고자 한다면 나무, 소, 울타리 등의 모든 대상들이 각각 물체 목록이 될 수 있는데,

표 2.1 파일 구조의 태그들

	시작 태그	종료 태그
기능 기술 부분	OptionBegin	OptionEnd
카메라 기술 부분	CameraBegin	CameraEnd
라이트 기술 부분	LightBegin	LightEnd
물체 기술 부분	ObjectBegin	ObjectEnd
물체 목록 기술 부분	ItemBegin	ItemEnd

각 물체 목록들에 대한 물체의 모양, 색과 표면의 성질 등과 같은 특징들을 저장할 수 있는 자료 구조이다. 본 렌더링 시스템에서는 하나의 물체 목록을 표현하기 위한 자료 구조는 다음과 같다.

```
struct _Item  {
  GRString itemname;
  GRGeomEdit geomedit;
  GRShort objnum;
  GRBoolean selected;

  GRObject *rootobj;
  struct _Item *lnext;
  struct _Item *rnext;
};
```

물체 목록에 할당되는 이름을 저장하기 위한 공간인 itemname과 물체 목록을 위한 여러 가지 기하학적인 정보를 표현하기 위한 공간인 geomedit가 있다. 또한 하나의 물체 목록은 계층구조적인 모델을 지원하기 때문에 여러 물체들의 조합으로서 구성될 수 있다. 이러한 계층구조적인 모델을 지원하기 위해서 계층구조내의 물체들의 수를 저장하기 위한 objnum과, 최상위 레벨의 물체에 접근하기 위한 포인터가 rootobj가 있으며, 효율적인 물체 목록의 연결 리스트를 구성하기 위한 포인터들 lnext, rnext들이 있다. 보는 바와 같이 하나의 물체 목록은 단지 각 물체들을

묶어서 하나의 개념으로 보여지게 하는 역할만을 하는 것을 알 수 있다. 이에 대한 자세한 개념도는 그림 2.20과 같다.

그림 2.20에서와 같이 만약 여러 개의 물체 목록이 전경내에 존재한다면, 본 렌더링 시스템은 이들을 하나의 연결 리스트로서 관리한다. 이것은 전경 내에서 이러한 물체 목록들이 자주 추가되고 삭제되는 특징을 가지기 때문이다. 또한 임의의 순간적인 접근이 필요한 기능들이 없으므로 이러한 연결 리스트로 표현하는 것은 매우 효율적이다. 만약 사용자가 사용자 인터페이스의 Delete를 통해 물체 목록을 삭제하면 내부적으로 이 연결 리스트 상에서 해당하는 노드가 삭제되는 것이다. Copy의 경우에는 똑같은 노드가 하나 더 생성되어 연결 리스트에 추가되는 것으로 보면 된다.

또 하나 중요한 것은 하나의 편집 단위인 물체 목록을 전경 내에 제대로 표현하기 위해서는 그 물체 목록에 대한 기하학적인 위치 정보를 충분히 저장하고 있어야 한다는 것이다. 본 렌더링 시스템에서 기하학적인 위치 정보를 저장하기 위한 자료 구조는 다음과 같이 정의되어 있다.

```
struct _GeomEdit  {
  GRPoint3D origin;
  GRVect3D n;
  GRVect3D v;
  GRVect3D u;
```

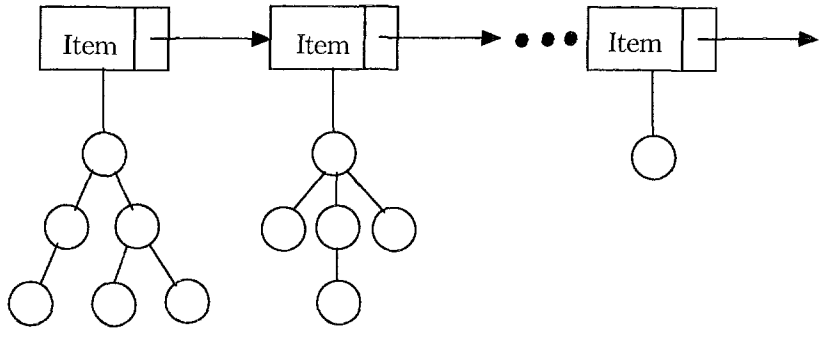


그림 2.20 물체 목록의 연결 리스트

```
  GRMatrix xform;
};
```

여기서 origin은 물체 목록의 중심이 현재 월드 상의 어느 곳에 위치하는지를 나타낸다. 즉 월드 상에서의 x, y, z 값을 차례로 저장하고 있다. 그리고 u, v, n은 현재 이 물체 목록의 지역적인 좌표축을 나타내고 있는 벡터형 자료로서 물체 목록의 회전 변환 시에 중요한 정보이다. 이러한 정보들은 사용자가 물체 목록에 적용시킨 기하학적인 변환들이 결과적으로 어떻게 작용되었는지를 나타낸다. 그리고 4×4 행렬의 자료형인 xform은 모델링 좌표계 내의 물체 목록을 월드 좌표계 상으로 변환시켜 주는 변환 행렬이다. 지금까지 이 물체 목록에 영향을 미쳐온 변환 행렬들을 xform의 왼쪽에 곱하여서 결과로 나온 새로운 행렬을 xform에 저장함으로서 최종적인 xform을 얻을 수 있다.

(나) 물체

하나의 물체 목록은 하나 이상의 물체들로 구성되어 있다고 했는데 이제이 물체라는 개념에 대해 자세히 살펴보겠다. 물체는 삼차원 기하학 정보를 표현하는 가장 작은 단위로서 더 이상 나뉘어 질 수 없다. 예를 들어 자동차라는 물체 목록 내에서는 타이어, 문, 유리창 등이 하나의 물체로서 여겨질 수 있으며 이들 간의 적절한 조합에 의해 하나의 자동차로서 보여지는 것이다. 본 고급 렌더링 시스템에서 하나의 물체를 표현하기 위해 정의한 자료 구조는 다음과 같다.

```
struct _Object  {
  GRString objname;
  GRColor3D color, opacity;
  unsigned long flags;
  GRMap texture;
  GRShadingInterpolationType shadinginterpolationtype;
```

```
    GRFloat shadingrate;
    GRShader surface;
    GRShader atmosphere;
    GRShader interior;
    GRShader exterior;
    GRShader deformation;
    GRShader displacement;
    GRGeomEdit geomedit;
    GRGeomDB *geodbptr;
    GRInt childnum;
    struct _Object *parent;
    struct _Object *child;
    struct _Object *lsibling;
    struct _Object *rsibling;
};
```

물체는 본 고급 렌더링 시스템에서 세부적인 조작과 편집을 지원하는 가장 작은 단위이기 때문에 여러 속성들을 저장하기 위한 공간들이 필요하다. 물체의 이름과 색, 불투명도, 텍스쳐, 세이더 등에 대한 정보들을 저장하기 위한 구조체들이 선언되어 있다. 또한 이런 물체들이 모여서 하나의 물체 목록을 이룰 수 있도록 하는 각 종 포인터 정보들과 자신의 기하학적인 위치 정보를 위한 geomedit도 볼 수 있다.

하나의 물체 목록은 여러 물체들이 계층 구조적인 트리 구조를 이루며

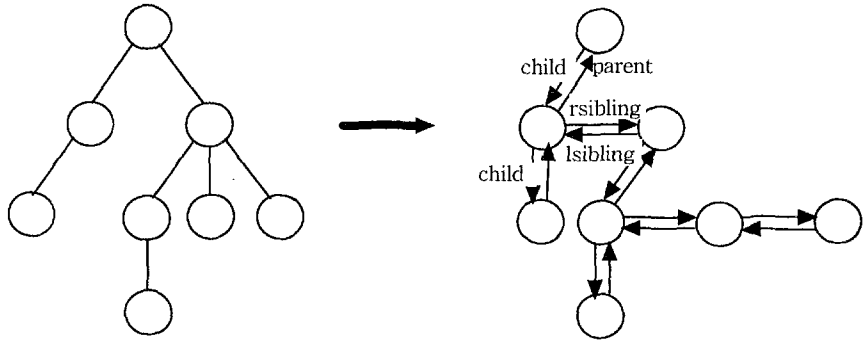


그림 2.21 실제 트리와 내부 저장 형태와의 관계

구성된다. 따라서 각 물체들은 트리를 구성하고 이 트리를 시스템이 접근하기 위한 여러 정보들을 담고 있어야만 한다. 여기서 유지되는 트리는 임의의 자식 노드를 가질 수 있기 때문에 그림 2.21과 같이 하나의 자식과 형제 노드를 가르키는 방법으로 트리를 유지했다. 즉, 왼쪽의 개념적인 트리의 형태는 실제 내부적으로 오른쪽과 같은 형식으로 저장되어 있다.

위의 자료 구조에서 눈여겨 볼 것은 물체의 형태에 대한 기하학적인 정보, 즉 이 물체를 형성하는 다각형이나, 2차 함수의 회전체, 패치들에 대한 정보들이 없다는 것이다. 이것은 이런 기하학적인 형태 정보를 따로 데이터 베이스화 해서 그 형태 정보를 가진 곳을 가리키는 geomdbptr이라는 포인터만을 유지하기 때문이다.

(다) 기하학적인 형태 정보를 위한 데이터베이스

위에서 언급한 바와 같이 본 렌더링 시스템은 하나의 물체내에 기하학적인 형태 정보까지 일일이 다 저장하는 것이 아니라 그림 2.22에서 보는 바와 같이 기하학적인 형태 정보를 데이터 베이스화 해 놓고 해당하는 정보를 가르키는 포인터만 가지고 있는다. 여기서 구성하는 데이터 베이스는 단순한 연결 리스트로서 다양한 형식의 기하학적인 형태 정보를 저장

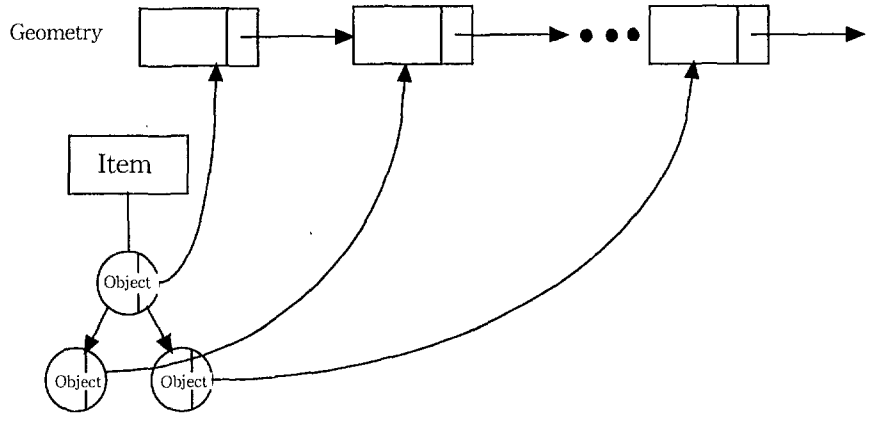


그림 2.22 물체 목록과 기하학적인 형태 정보의 관계

한다. 즉 연결 리스트 내의 하나의 노드가 하나의 기하학적인 형태 정보를 표현한다. 데이터 베이스 내의 노드 또한 사용자의 작업 도중에 자주 추가되고 삭제되는 특성을 가지므로 이런 연결 리스트 형태의 표현이 적합하다.

본 렌더링 시스템에서 도입한 개념인 기하학적인 형태 정보 데이터 베이스를 사용하면 물체 데이터를 관리하는데 있어서 메모리를 매우 효율적으로 사용할 수 있다. 만약 사용자가 현재 전경 내에 존재하는 물체를 하나 더 복제하고 싶다면 사용자는 Copy 메뉴를 통해 새로 복제된 물체가 전경 내에 추가되는 것을 볼 수 있을 것이다. 그러나 내부적으로는 그 물체 목록을 나타내기 위한 적은 양의 메모리만 더 사용되고 상대적으로 매우 많은 메모리를 요구하는 기하학적인 형태 정보를 저장하기 위한 메모리가 더 필요하지는 않다. 이것은 복제되는 물체 목록의 기하학적인 형태 정보 데이터 베이스에 대한 포인터만 복사하면 되기 때문이다.

예를 들어 수 천 개의 다각형으로 이루어진 컴퓨터라는 물체가 수 십대 놓여 있는 전산실을 표현하고 싶다면 컴퓨터의 기하학적인 형태 정보를 저장하기 위한 하나의 저장 공간만이 요구되고 나머지 수 십 개의 컴퓨터를 위한 수 십 개의 물체 목록들은 이 형태 정보를 고유하게 되므로 같은 물체의 수가 늘어나는 것에 따른 메모리 문제는 방지하게 된다.

본 렌더링 시스템에서는 다양한 형식의 기하학적인 형태 정보를 지원한다. 가장 일반적인 다각형 모델에서부터 2차 함수의 회전 물체, 패치에 이르기까지 매우 다양하다. 이런 다양한 형태의 정보를 하나의 노드에 저장하기 위해, 데이터 베이스 내의 한 노드는 다음과 같은 공용체 정보를 포함하는 기하학적 형태 정보를 유지한다.

```
struct _Geometry  {
  GRPrimitiveType tag;
  GRInt listnum;
```

```
    union  {
      GRSphere sphere;
      GRCone cone;
      GRDisk disk;
      GRCylinder cylinder;
      GRHyperboloid hyperboloid;
      GRParaboloid paraboloid;
      GRTorus torus;
      GRPolygons *polygons;
      GRPolyhedron polyhedron;
      GRPatches patches;
      GRPatchMesh patchmesh;
      GRNuPatches *nupatches;
    } geodata;
};
```

여기서 공용체 내의 포함되는 기하학적 형태 정보들을 살펴보면 다음과 같다.

1) sphere

반지름과 회전각, 위 아래의 절단 길이 등에 의해 구를 표현한다. 구의 중심은 모델링 좌표계의 원점이 된다.

2) cone

반지름과 회전각, 높이 등으로 원뿔을 표현한다. 원뿔의 바닥의 중심이 모델링 좌표계의 원점이 된다.

3) disk

반지름과 회전각으로서 원반을 표현하는데 높이를 주어 원점으로부터의 거리를 지정할 수 있다.

4) cylinder

반지름과 회전각, 위 아래 높이 등으로 원통을 표현한다. 물론 모델링 좌표계의 원점을 기준으로 한다.

5) hyperboloid

반지름과 위 아래 절단 길이 등을 이용하여 포물선의 회전체를 표현한다.

6) paraboloid

두 점이 주어지면 모델링 좌표계의 y축에 대한 회전으로 만들어지는 표면을 표현한다.

7) torus

대반지름, 소반지름, 회전각 등을 이용하여 도우넛 모양의 물체를 표현한다.

8) polygons

다각형의 리스트로 이루어진 물체에 대한 표현으로서 아마도 대부부느 이 경우에 해당될 것이다. 한 다각형의 꼭지점들의 위치와, 법선 벡터, 텍스처 좌표 등을 연결 리스트로서 저장한다.

9) polyhedron

다각형을 이루는 요소들이 리스트로 주어지는 것이 아니라 하나의 배열로서 존재하고 그에 대한 인덱스를 참조하는 형태이다. 이 표현 형태를 쓰면 중복되는 꼭지점에 대한 낭비를 줄일 수 있다.

10) patches

패치들의 리스트로 이루어진 물체에 대한 표현이다. 패치는 곡면을 부드럽게 표현할 수 있는 장점이 있으나 그에 따른 정보의 양도 많아지게 된다.

11) patchmesh

패치를 이루는 제어점들이 한 번에 배열로서 주어지는 표현이다.

12) nupatches

NURB 곡면의 리스트로 이루어진 물체에 대한 표현이다.

(라) 카메라와 광원

본 렌더링 시스템의 사용자 인터페이스는 전경 내의 물체 목록 뿐만 아니라 카메라와 광원에 대한 조작도 가능하다. 이들은 객체 변환 모드에서 선택하여 직접 조작할 수도 있지만 카메라나 광원이 시점이 되어서 전경 변환 모드에서의 움직임을 통해 조작이 가능하다.

(2) VRML 2.0

VRML[2-2]은 Virtual Reality Modeling Language의 약자로 Internet상의 가상적인 3차원 물체를 설계할 때 사용되는 스크립트이다. Internet이란 가상 공간에 자신의 방, 건물, 도시, 산, 혹은 혹성까지도 만들 수 있으며 또한 가상의 가구, 자동차, 사람, 우주선 따위로 가상의 공간을 채우고 네비게시션 프로그램을 통해서 그 공간을 여행할 수 있다. VRML이란 상상력을 구체화시키는 이러한 요구를 만족시키는 언어이다. VRML 2.0은 VRML 1.0에 추가적인 몇 가지 기능이 첨가된 것으로 1.0 버전에서는 정지된 공간만을 표현한데 반해서 2.0 버전에서는 공간 상에서 물체의 애니메이션을 표현할 수 있고 마우스나 기타 입력 장치를 통해서 가상 공간에 사용자의 인터액티브한 요구에 반응하는 방법을 표현할 수 있다. 또 공간에 물체나 인터넷 사이트 링크 뿐 만 아니라 사운드나 동영상 등도 링크할 수 있으며 그 외에도 기존의 모델링 스크립트로 기능도 진보되었다.

(가) VRML 2.0의 구조

VRML은 Open Inventor의 부분 집합에 몇 가지 인터넷과 관련된 항목이 추가되어진 언어를 선언한 것으로 VRML 2.0은 그 외에 인터액티브 반응, 애니메이션 등이 추가된 것이다. VRML 파일은 VRML 공간을 기술한 스크립트 파일이며 직접 문서 편집기로 작성할 수도 있고 VRML을 전문적으로 생성하는 비주얼 툴을 사용할 수 도 있다. VRML 파일 이름 다음에는 .wrl이라고 표기해서 파일의 종류를 명시적으로 표기하는데 "dot world"라고 발

음한다. VRML 파일은 이 파일이 VRML의 파일임을 명시하는 헤더가 존재하며 그 다음에는 다음과 같은 항목들을 이용하여 VRML의 내용 부분을 구성한다.

- 주석문
- 노드
- 필드와 필드값
- 정의되는 노드 이름
- 사용되는 노드 이름

VRML은 노드의 계층적 블록 구조로 이루어진 언어이며 노드들은 필요에 따라서 프로그래머가 정의한 노드 이름의 변수로 치환되어서 후에 이 노드를 재사용 하고자 할 때 간단하게 변수로서 그 노드를 연결해 놓을 수 있다. 간단한 VRML의 예제를 살펴보자.

```
#VRML V2.0 utf8
# A brown hut
Group  {
  children [
  # Draw the hut walls
    Shape  {
      appearance DEF Brown Appearance  {
        material Material  {
          diffuseColor 0.6 0.4 0.0
        }
      }
      geometry Cylinder  {
        height 2.0
        radius 2.0
      }
    },
  # Draw the hut roof
```

```
    Transform  {
      translation 0.0 2.0 0.0
      children Shape  {
        appearance USE Brown
        geometry Cone  {
          height 2.0
          bottomRadius 2.5
        }
      }
    }
  ]
}
```

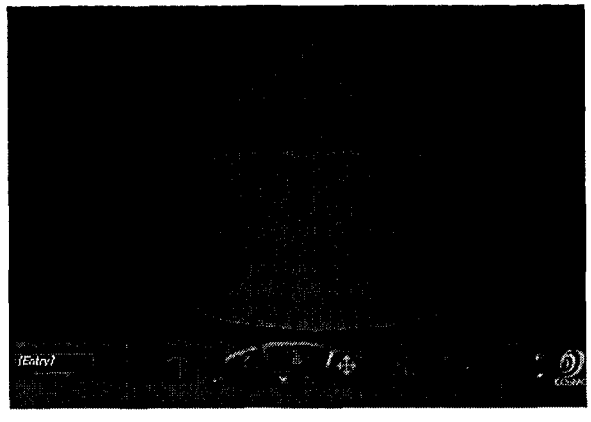


그림 2.23 예제 hut.wrl의 실행 화면

예제 소스에서 헤더는 "VRML V2.0 utf8"이고 그 외의 샵('#')으로 시작되는 문장은 모두 주석문이다. 노드의 이름은 첫 글자가 대문자인데 Group, Shape, Appearance, Material, Cylinder, Transform, Cone이 노드이다. 필드는 노드 안의 항목으로 Transform 노드의 경우 translation, children 등의 필드들을 갖는다. "DEF"라는 키워드를 사용하여 Brown이란 변수가 갈색을 표현하는 Appearance 노드를 가리키도록 선언하였고 이것은 "USE"라는 키워드에 의해서 원통의 색깔 지정 시에 사용되어 진다. 대략 볼 수 있듯이 정의된 몇 개의 블록을 조합해서 삼차원

모델의 기술 뿐 아니라 카메라, 라이트 등도 이와 같은 블록 구조로 함께 기술된다. 노드의 계층 구조에 따라서 부모 노드의 영향을 받지만 그 외의 노드와는 독립적으로 순서 따위는 중요치 않으며 노드 안의 필드도 그 순서는 상관없다. 각 노드는 기하학적 정보를 갖는 것, 변환 정보를 갖는 것, 그룹핑을 위한 것 등으로 나누어 질 수 있다.

(나) VRML 2.0의 노드와 필드

VRML 2.0은 수 십 개의 노드를 가지고 있다. 본 컨버터는 기하학 정보, 물체 속성 정보, 세이더 정보, 라이트, 카메라의 정보를 가지는 본 렌더링 시스템 파일은 VRML 2.0의 애니메이션, 인터액티브, 인터넷 혹은 멀티미디어 링크 등에 관련된 몇 개의 노드를 제외한 필요한 정보를 가지는 노드만을 분석한다. 본 컨버터가 사용하는 VRML 2.0 노드와 필드는 다음과 같다.

1) Appearance

* material 필드 - Material 노드의 포인터

2) Material

* diffuseColor 필드
* emissiveColor 필드
* transparency 필드
* ambientIntensity 필드
* specularColor 필드
* shininess 필드

3) Shape

* appearance 필드 - Appearance 노드의 포인터
* geometry 필드 - Box, Cone, Cylinder, Sphere, IndexedFaceSet 노드의 포인터

4) Box

* size 필드 - x, y, z 길이

5) Cone

* bottomRadius 필드 - 원뿔의 디스크의 반경
* height 필드 - 원뿔의 높이
* side 필드 - 원뿔의 옆 면의 존재 유무
* bottom 필드 - 원뿔의 밑 면의 존재 유무

6) Cylinder

* radius 필드 - 원통의 디스크의 반경
* height 필드 - 원통의 높이
* side 필드 - 원통의 옆면의 존재 유무
* top 필드 - 원통의 윗면의 존재 유무
* bottom 필드 - 원통의 아랫면의 존재 유무

7) Sphere

* radius 필드 - 구의 반경

8) Color

* color 필드 - R, G, B 값의 어레이(Array) 포인터

9) Coordinate

* point 필드 - x, y, z 값의 어레이(Array) 포인터

10) TextureCoordinate

* point 필드 - x, y 값의 어레이(Array) 포인터

11) Normal

* normal 필드 - x, y, z 값의 어레이(Array) 포인터

12) ImageTexture

* url 필드 - 텍스처 이미지의 파일 이름의 문자열(String)

* repeatS 필드

* repeatT 필드

13) IndexedFaceSet

* coord 필드 - Coordinate 노드의 포인터

* coordIndex 필드 - Coordinate 노드의 3차원 좌표들에 대한 인덱스 어레이

* texCoord 필드 - TextureCoordinate 노드의 포인터

* texCoordIndex 필드 - TextureCoordinate 노드의 2차원 좌표들에 대한 인덱스 어레이

* color 필드 - Color 노드의 포인터

* colorIndex 필드 - Color 노드의 RGB값들에 대한 인덱스 어레이

* normal 필드 - Normal 노드의 포인터

* normalIndex 필드 - Normal 노드의 3차원 벡터들에 대한 인덱스 어레이

14) Transform

* rotation 필드 - 임의의 축에 대한 회전

* translation 필드 - 임의의 방향으로 공간 이동

* scale 필드 - 확대, 축소

* scaleOrientation 필드 - 임의의 중심에 대한 확대, 축소

* center 필드 - 변환 중심점 지정

* children 필드 - 노드 포인터 어레이

15) Group

* children 필드 - 노드 포인터 어레이

16) Switch

* choice 필드 - 노드 포인터 어레이

* whichChoice 필드 - 선택 노드 번호, -1은 모든 노드

17) Billboard

* children 필드 - 노드 포인터 어레이

18) Collision

* children 필드 - 노드 포인터 어레이

19) PointLight

* on - 광원의 점멸, 소멸의 유무

* location 필드 - 점 광원의 위치

* attenuation 필드 - 점 광원으로 부터의 밝기의 변화 정도

* color 필드 - 점 광원의 빛의 색

* intensity 필드 - 점 광원의 빛의 강도

* ambientIntensity 필드 - Ambient 광원으로써의 빛의 강도

20) DirectionalLight

* on - 광원의 점멸, 소멸의 유무

* direction 필드 - 직사 광원의 방향

* attenuation 필드 - 직사 광원으로 부터의 밝기의 변화 정도

* color 필드 - 직사 광원의 빛의 색

* intensity 필드 - 직사 광원의 빛의 강도

* ambientIntensity 필드 - Ambient 광원으로써의 빛의 강도

21) SpotLight

* on - 광원의 점멸, 소멸의 유무

* location 필드 - 스팟 광원의 위치

* direction 필드 - 스팟 광원의 방향

* attenuation - 스팟 광원으로 부터의 밝기의 변화 정도

* color 필드 - 스팟 광원의 빛의 색

* intensity 필드 - 스팟 광원의 빛의 강도

* radius 필드 - 스팟 광원의 반지름

* ambientIntensity 필드 - Ambient 광원으로써의 빛의 강도

* beamWidth 필드

* cutOffAngle 필드

(3) VRML 2.0의 본 렌더링 시스템의 아스키 파일로의 전환

VRML 2.0 파일은 본 렌더링 시스템으로 전환할 때 발생할 수 있는 문제점과 장,단점을 설명하고 간단한 사용예를 살펴 보겠다.

(가) 전환 상의 문제점

VRML 2.0 파일(".wrl")을 본 렌더링 시스템의 아스키 파일(".asc")로 전환하는 것이 본 전환 프로그램("vrml2sert")의 목적이다. 각각의 특징적 포맷으로 인하여 완벽하게 1대 1관계의 전환은 불가능하지만 본 렌더링 시스템이 기하학적 모델과 렌더링 관련 정보는 VRML 2.0에 비해서 다양한 기능을 가지고 있기 때문에 VRML 2.0로 제작된 모델이 전환 후 변형을 일으키는 경우는 극히 드물다. 단 본 렌더링 시스템에서는 PointSet과 LineSet 같은 점으로 구성된 물체나 선으로만 구성된 물체를 표현하지 못하기 때문에 이런 물체는 전환 과정에서 생략되어 진다. 그리고 본 렌더링 시스템은 물체 목록 부분의 트리 노드들이 각각 하나의 물체를 지정해야만 하는데 VRML 2.0의 노드들의 계층 구조에서는 Group, Transform과 같은 노드들은 물체를 갖는 노드가 아니므로 본 시스템의 물체 목록의 계층 구조에 직접 대응시킬 수는 없었고 대신에 보이지 않는 가상의 물체(물체 번호 : 0, 구형 물체)를 이런 노드의 물체로 등록시켜서 본 시스템의 계층 구조로 전환하였다. 또 본 시스템의 물체 목록은 오직 1개만의 루트 노드를 가지는데 비해서 VRML 2.0은 여러 개의 루트 노드를 가질 수 있는 구조이므로 이런 여러 개의 노드를 묶을 수 있는 가상의 루트 노드

가 있다. 이런 특수한 목적을 위해서 VitualNode, Root, Unknown와 같은 특수 노드를 만들어서 본 시스템의 파일 구조로 전환을 도모하였다.

본 시스템의 경우는 물체와 그 물체를 구성하는 부품의 정도를 명확히 표현한다. 예를 들어서 의자의 경우 구성 부품을 한도를 등받이, 의자 다리, 쿠션로 지정할 수도 있지만 좀 더 자세히 나누어서 나무막대, 나사, 쿠션으로 의자를 구성할 수도 있을 것이다. 그러나 VRML 2.0의 경우는 어디까지가 가상 공간이고 어디까지가 물체 목록이며 어디까지가 구성 물체인지가 모호하기 때문에 본 시스템으로 전환하는 과정에서 어디까지를 물체 목록으로 지정하고 어디까지를 물체로 지정하는가가 가장 큰 문제였다. 결국 자동적으로 해결하는 것이 불가능하였기 때문에 전환 프로그램에 옵션을 선택하여 물체 목록과 물체의 해당 정도를 지정할 수 있도록 하였다. VRML 2.0의 Transformation의 임의의 축에 대한 회전은 Quaternion[2-3] 개념을 이용하여 행렬을 계산하였다.

본 시스템의 경우 렌더링에 관련된 기능이 강화되어 있기 때문에 렌더링에 관련된 다양한 기능 설정이 필요하다. 특히 셰이더의 경우가 그러한데 VRML 2.0의 경우에 셰이더의 개념이 없다. 때문에 전환 시에 모든 물체의 셰이더를 디폴트 셰이더인 플라스틱 셰이더로 지정하였다.

그림 2.24의 왼쪽은 SGI의 CosmoPlayer2.0으로 "sc1.wrl"을 플레이한 결과로 리얼타임 렌더링을 위하여 OpenGL Z-buffer 알고리즘을 사용한 것이고 오른쪽은 본 렌더링 시스템의 Ray Tracing을 사용한 결과이다.

(나) 전환 프로그램의 사용예

그림 2.24 VRML 2.0과 본 렌더링 시스템의 렌더링 결과

```
4 [grdegas/u/grmanet/home/cocoa/vrml > vrml2sert
Usage : vrml2sert [-option] <filename>
Options ------------------------------------------------
-clock : Generate Normal vector in clock-wise.
-countclock : Generate Normal vector in count-clock-wise.
-detail<number> : detail level(0-10)
-scene : Output to Scene file(Default)
-item : Output to Item file
-level<number> : explosion level(1-N)
5 [grdegas/u/grmanet/home/cocoa/vrml >
```

본 컨버터에 인자(Argument)를 입력하지 않으면 위와 같이 사용 방법과 가능한 옵션에 대한 설명이 출력된다. 파일 이름은 입력 파일 이름만 입력하면 되고 출력되는 파일 이름은 파일 이름 뒤에 ".asc"라는 확장자가 추가적으로 붙게 된다. 각 옵션을 살펴보면 clock과 countclock의 경우는 기하학적 정보에 법선 벡터 정보가 없을 경우에 자동적으로 법선 벡터를 생성하기 위한 다각형의 방향 설정이다. 기본 설정은 시계 방향이다.

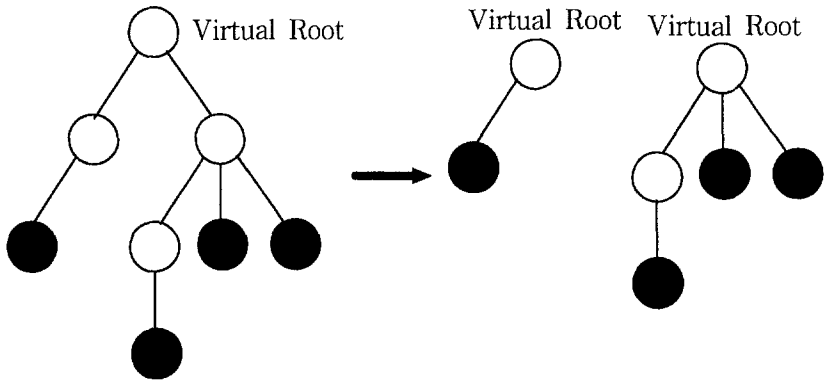


그림 2.25 VRML 2.0의 계층 구조를 level1으로 분리시킨 경우

detail은 구, 실린더, 원통, 원뿔같이 곡면으로 이루어진 물체를 다각형으로 변환할 때의 변환 정도로 그 수치가 높을수록 좀 더 부드러운 다각형을 얻을 수 있고 기본값은 5이다. 출력되는 아스키 파일은 씬 파일과 아이템 파일의 두 종류가 있는데 기본은 씬 파일이다. level은 물체와 물체 목록의 해당 정도로 자료 구조적으로 보면 씬의 계층적 구조 트리에서 루트부터 지정된 깊이까지를 제거하고 그 아래의 하나 이상의 트리를 각각 하나의 물체 목록으로 생성해서 그 트리들의 각각의 리프 노드(leaf nodes)들을 물체로 잡게 되는 것이다.

그림 2.25가 level의 수행 결과를 보여준 것이다. 기본 level값은 0으로 이 경우 계층 구조의 분리 형상이 발생하지 않는다. 진하게 칠해진 리프 노드들은 VRML 2.0의 기하학 모델을 표현하는 최종 노드들(Box, Sphere, Cylinder, Cone, IndexedFaceSet)이다. 위의 경우 각각의 좌, 우 트리는 물체 목록이 되고 각 트리의 리프 노드들은 해당 목록의 물체가 된다.

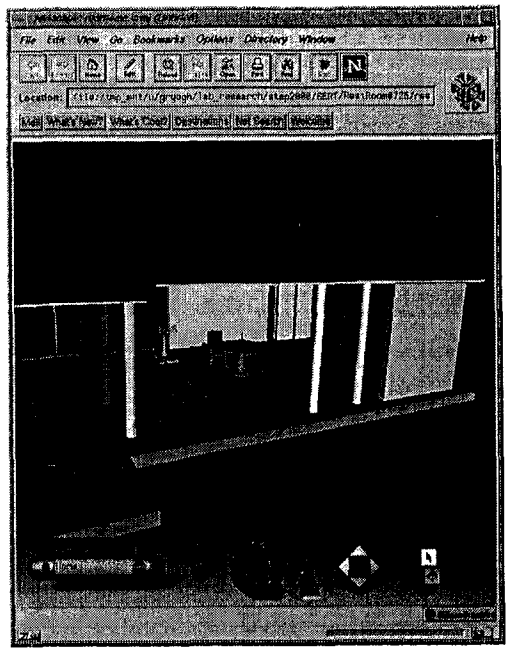
위의 예는 본 전환 프로그램에 의해서 "arch.wrl" 파일을 level1의 옵션으로 ".asc" 파일의 전경으로 전환이 성공적으로 수행된 경우의 화면 출력이다.

```
9 [grdegas/u/grmanet/home/cocoa/vrml > vrml2sert -level1 arch.wrl
Explosion Level : 1
Parsing is running now.
Parsing is over now.
Write ObjectDB.
Write LightDB.
out Point
Write ItemDB.
Explosion is processed now.....
Program is over rightly.
10 [grdegas/u/grmanet/home/cocoa/vrml >
```

(다) 전환 프로그램 개발 환경

본 전환 프로그램은 SGI 홈페이지의 VRML 2.0 Parsing source(Lex, Yacc code)를 lex와 bison으로 컴파일해서 생성된 코드를 수정, 보완해서 작성하였다. 컴퓨터 환경은 SGI Indy, IRIX 5.3 이였고 컴파일러는 g++을 사용하였다.

라. VRML 데이터의 전환과 렌더링 결과



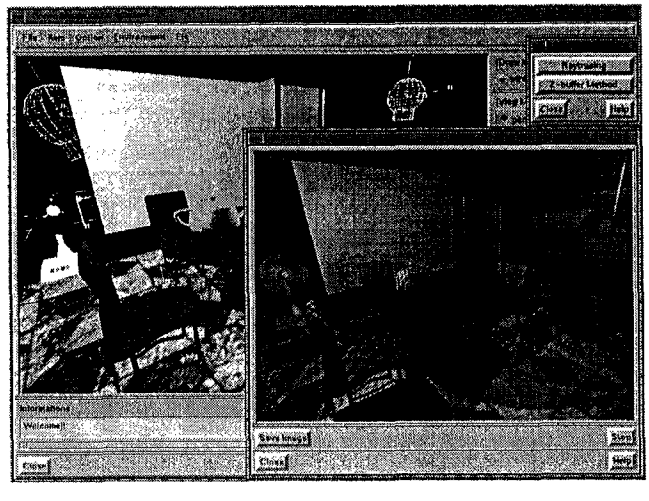
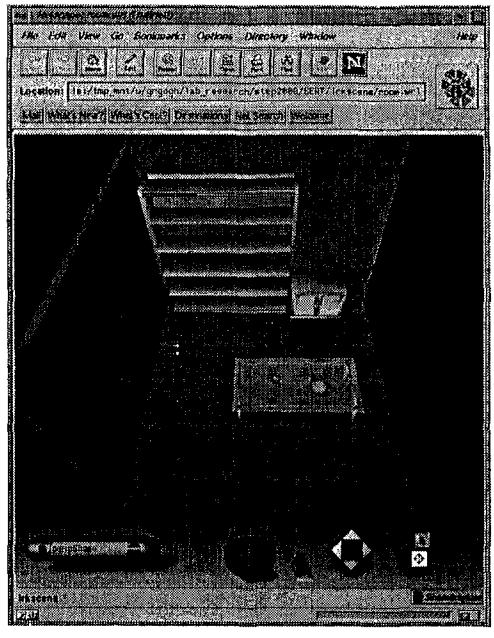


그림 2.26 Rest room의 VRML 브라우저의 렌더링 결과 (본 렌더러의 그림자, 셰이더 등을 포함한 고급 렌더링 효과를 볼 수 있다)



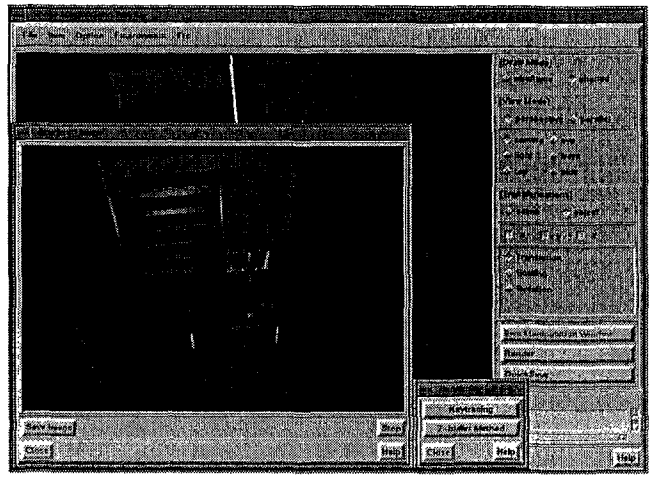
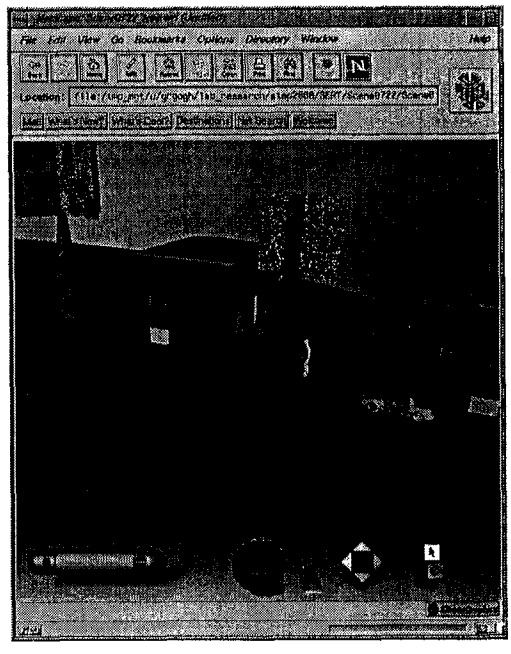


그림 2.27 Room의 VRML 브라우저의 렌더링 결과 (본 렌더러의 렌더링 전경에는 의자 두 개를 추가하였다)



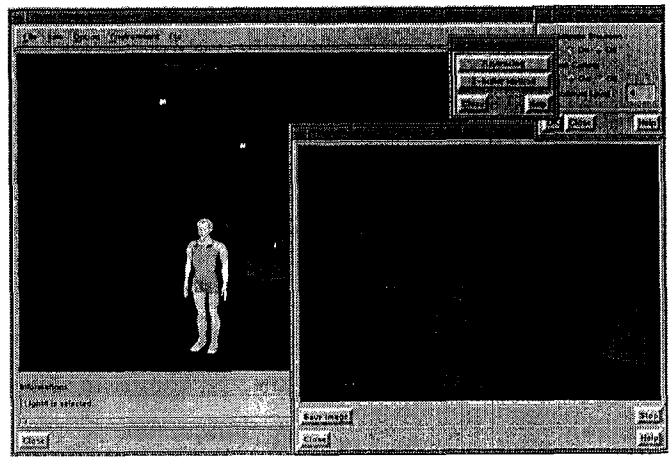


그림 2.28 Scene0722_tree_view의 VRML 브라우저의 렌더링 결과와 본 렌더러의 렌더링 결과

3. SERT의 렌더링 사용 예 (Tutorial)

가. 실행

SERT를 실행하기 위하여 준비된 아이콘이나 직접 명령어 줄(ex. %sert)을 입력한다. 그러면 4가지 메뉴를 접하게 되는데 여기서 예를 들 전경을 렌더링하기 위해 Scene Manipulator라고 적힌 메뉴를 선택한다.

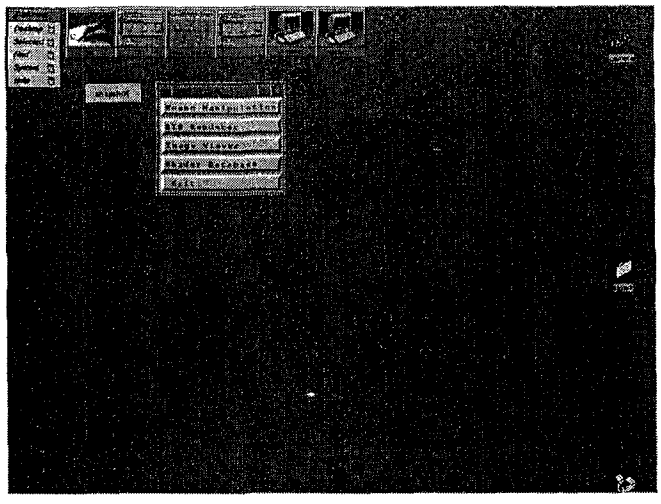


그림 2.29 Sert의 초기화면

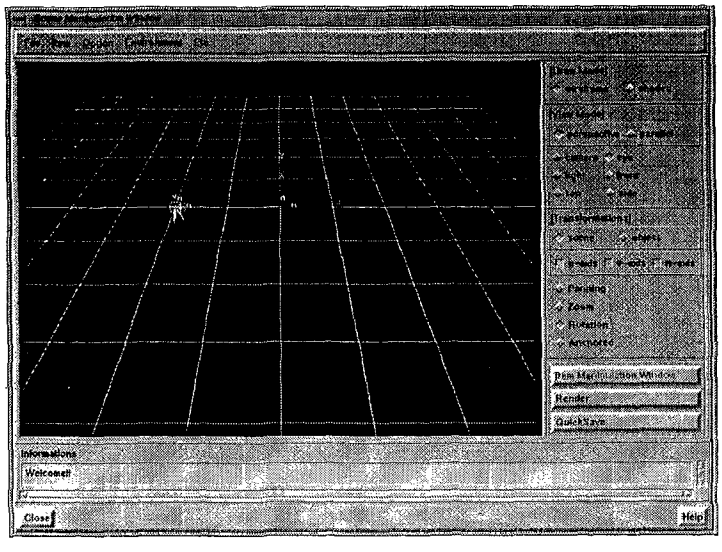


그림 2.30 Scene manipulation Window.

그러면 Scene Manipulator Window가 열리며 원하는 전경을 입력받기를 요구할 것이다.

나. 파일 열기

이제, 전경이 기술된 파일을 불러온다. 상단메뉴의 File이라고 적힌 메뉴를 선택하면 4개의 부 메뉴가 뜨는데, 가상의 방을 만들기 위해 기본 데이터가 기술되어있는 파일을 선택하기 위해 Load를 선택한다.

Load를 선택하면 File Load창과 원하는 전경이 있는 위치에 관한 Path와 Filter, File Name을 보여주는 작은 창들과 선택을 결정지어 실제로 적용시킬 수 있는 Accept 버튼, Close 버튼 등등이 있다.

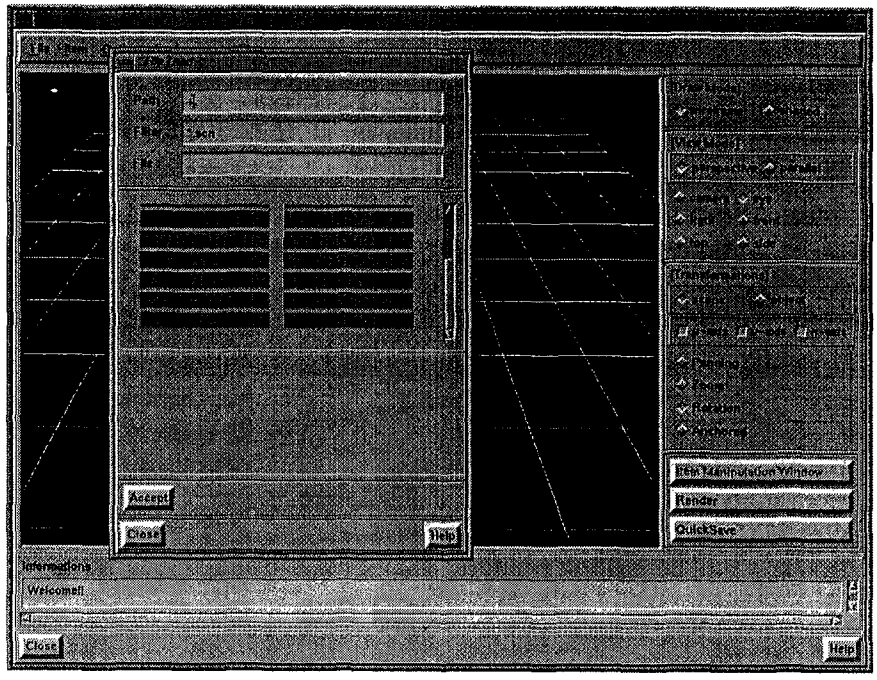


그림 2.31 File Load 창

여기서 일단 sert_data 디렉토리에 있는 sert1.scn 파일을 열기 위해 파란색 버튼으로 나타나있는 sert_data 버튼을 선택한 후, 그 디렉토리의 파일들의 내용 중 sert1.scn 버튼을 선택한 다음, Accept 버튼을 선택하면 예제로 사용할 전경이 Scene Manipulation Window에 나타날 것이다.

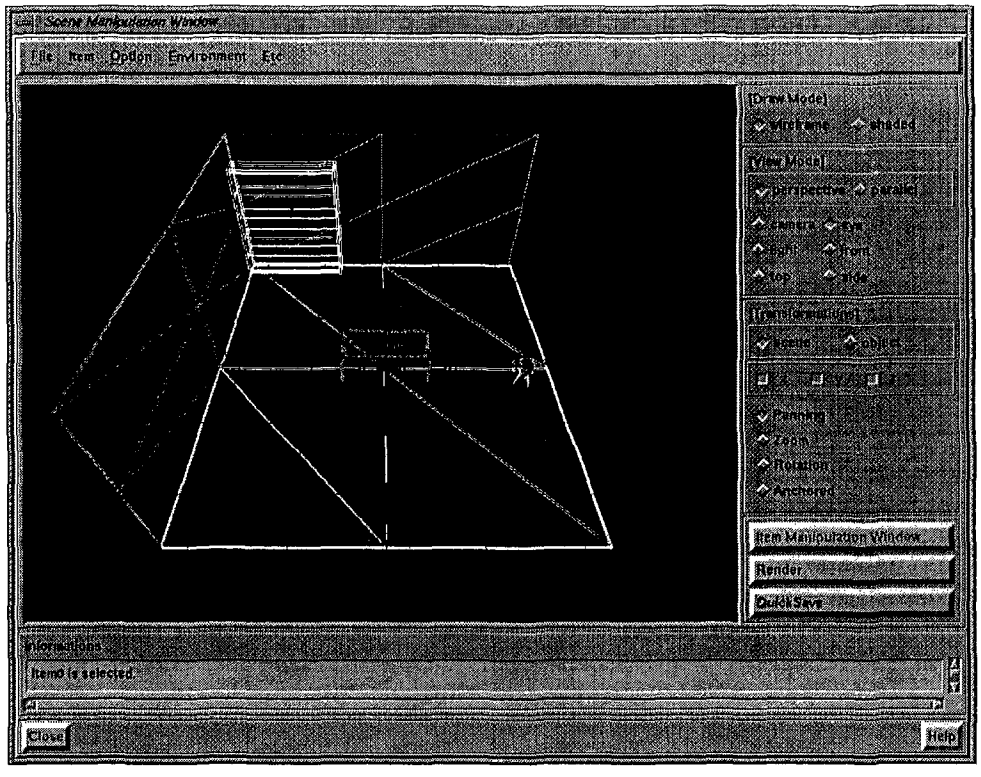


그림 2.32 불러온 예제 전경

다. 간단한 전경의 조작

지금 현재 나와있는 전경은 Wireframe 형태로 알아보기 힘들 수 있다. 보기 싫다면 오른쪽에 여러 라디오 버튼들을 이용할 수 있다. Draw Mode에 wireframe이라고 설정되어 있는 것을 shaded로 바꾸어 선택하면 색칠이 입혀진 상태로 전경이 바뀐다. (여기서 시간이 좀 걸릴 수 있지만 그것은 세이딩을 위한 계산량이 많기 때문이다.)

책상이 너무 멀리 떨어져서 보이지 않으므로 좀 가까이 가져와 보려면, 나타난 전경을 보기 좋은 상태로 끌어당기고 돌리기 위해 네 개의 버튼이 준비되어 있다. Panning과 Zoom, Rotation은 말 그대로 전경을 가까이 당겨보거나 돌려보거나 전체를 움직여 볼 수 있게 해주고 Anchored는 어느 한 점이나 물체를 선택하여 전경을 자유롭게 위치시킬 수 있도록 도와

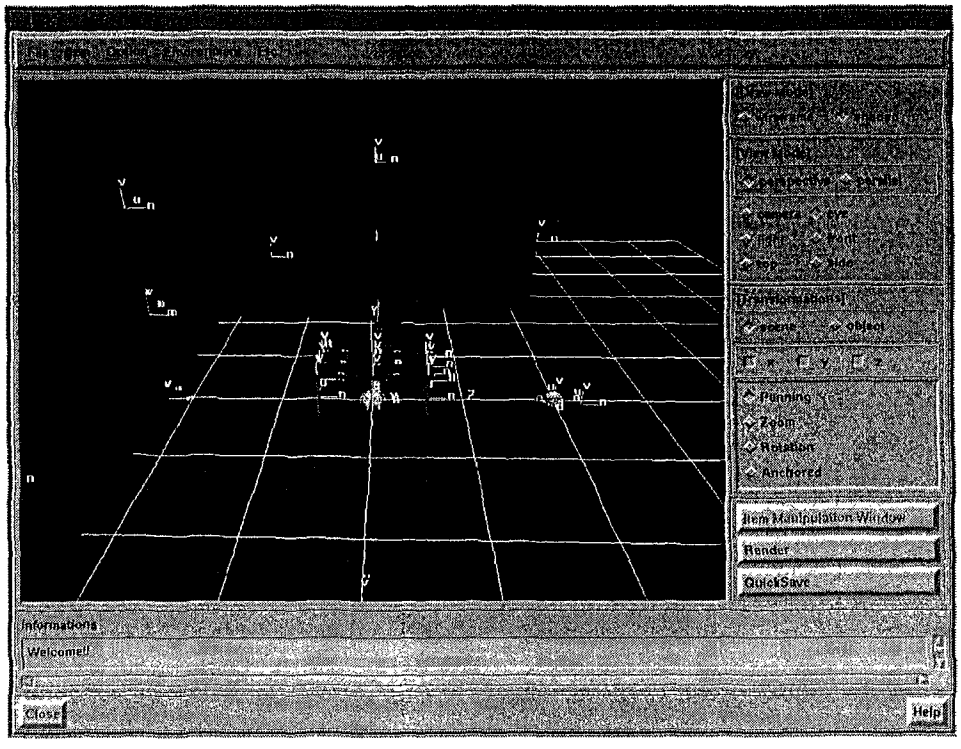


그림 2.33 화면의 확대, 이동, 네비게이션 등으로 변화된 전경 화면

준다. 그림 2.33과 같이 전경을 보기 위해 시도하든지 아니면 스스로 보기 좋은 모양으로 전경을 조작하여 볼 수 있다. 조작을 할 때에는 마우스 왼쪽 버튼을 사용하면 된다. 사용방법은 간단하기 때문에 쉽게 사용할 수 있다.

현재의 전경은 오른쪽에 Eye가 선택되어 있다면 눈에서 보는 전경이다. 이것을 Camera로 선택하였다면 카메라의 시점에서 볼 수 있도록 하여 주는 것이고 그 밖의 것은 메뉴를 참조하면 된다.

라. 물체목록의 선택

특정 물체목록을 선택해 보기 위해서는 오른쪽 메뉴에서 Transformations의 scene과 object 중 object를 활성화시킨 상태에서 화면에 나타나는 물체목록을 직접 마우스로 선택하거나 혹은 상단 메뉴에서 Item의 부 메뉴 중 Select by Name을 사용할 수 있다. 그러면 이름별로 선택할 수 있도록 작은 윈도우가 뜨는데 item 0을 선택하면 다음과 같은 그림이 뜰 것이다.

이렇게 선택된 물체목록은 Item Manipulation Window에서 조작되어 질 수 있는데 이것은 다음에 다루도록 하자.

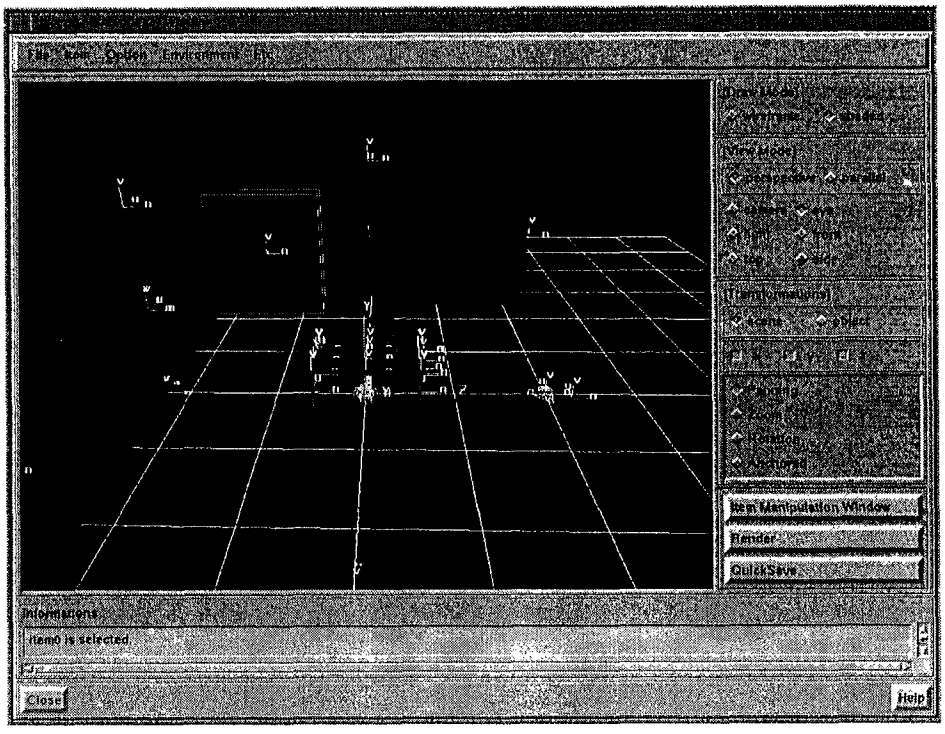


그림 2.34 선택된 물체목록

마. 물체목록 삽입

현재 전경 조작창에 새로운 물체목록을 삽입하기 위해서는 메뉴에서 item -> insert를 선택하면 file -> load를 선택했을 때 생성되는 창과 비슷한 창이 생성된다. 이 창은 물체목록을 삽입하기 위한 것이므로 filter란이 *.itm이라 되어있는 것을 볼 수 있다. 여기서 보이는 디렉토리 중에서 data를 선택하면, 해당 디렉토리에 존재하는 여러 물체목록 파일들이 나오게 된다.

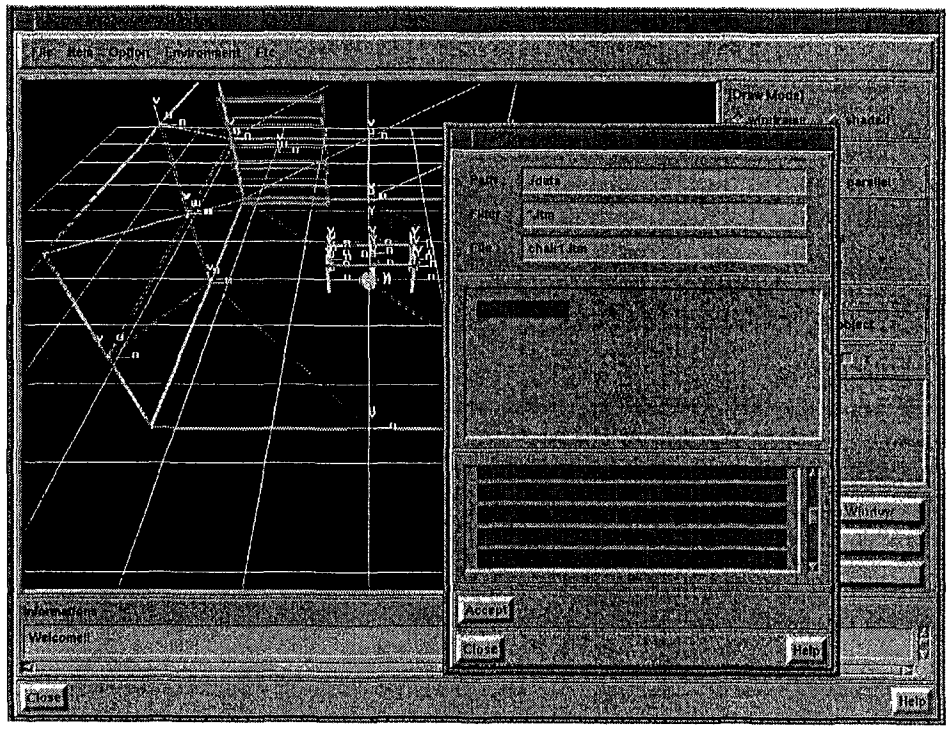


그림 2.35 물체목록 삽입창

현재 전경조작창에 의자를 하나 삽입하려면, 물체목록 파일중 chair1.itm을 accept하고 close 버튼을 선택한다. 그러면 그림 2.36과 같이 전경조작창에 커다란 의자가 나타난다.

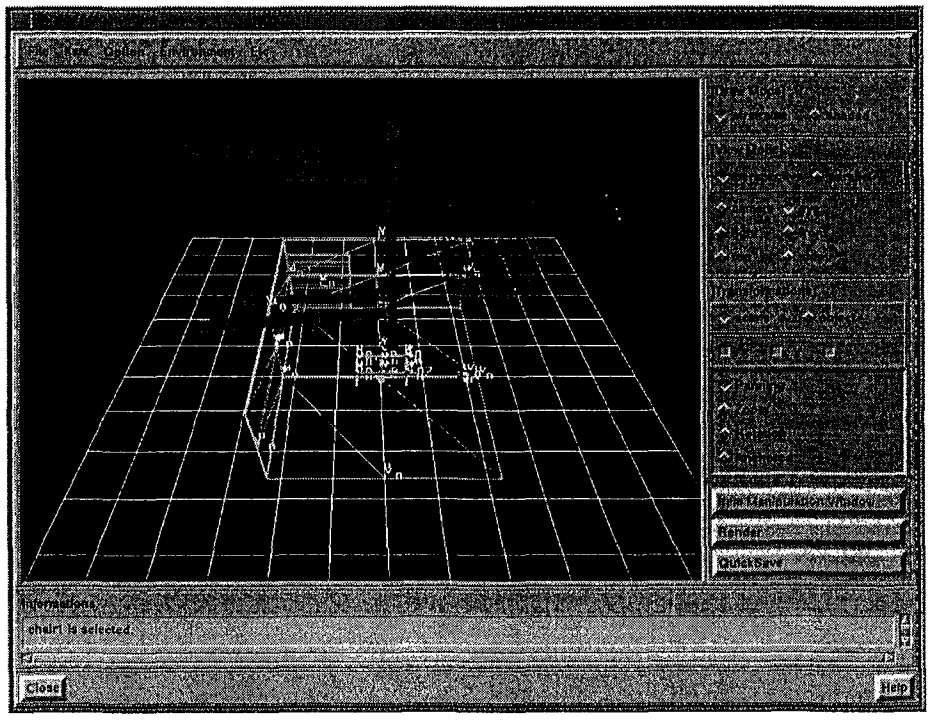


그림 2.36 의자 삽입 장면

바. 물체목록의 조작

마의 물체목록 삽입에서, 삽입된 의자는 방의 크기에 비해 그 크기가 너무 크다. 의자를 조작하여 방의 크기에 맞도록 조정하여야 한다. 우선 오른쪽에 있는 전경조작도구에서 object 라디오 버튼을 누른다. 그러면 그 아래쪽으로 변형도구가 나타난다.

object 라디오 버튼을 눌렀으므로 이제 물체 단위로 조작이 가능하다. 전경조작창에서 지금 삽입한 의자를 선택해 보자. 그러면 의자 주위에 녹색의 경계상자가 나타날 것이다. 이것은 의자가 선택되었음을 나타낸다.

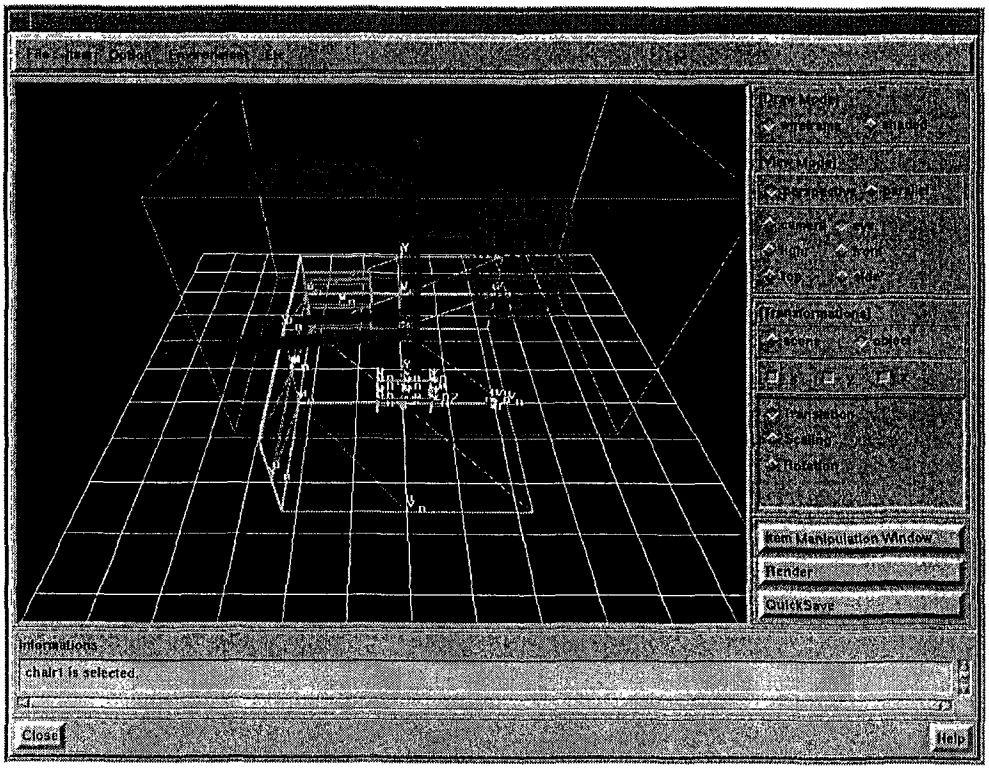


그림 2.37 의자를 선택한 화면

선택된 의자를 회전시키기 위해 전경조작도구 중 View Mode에서 front를 선택한다. 전경 데이터를 앞에서 바라봄으로써 의자를 정확하게 회전시킬 수 있다. 전경조작도구 중 Transformations에서 rotation을 선택할 후 n-axis만 회전시킨 것은 그림 2.38와 같다.

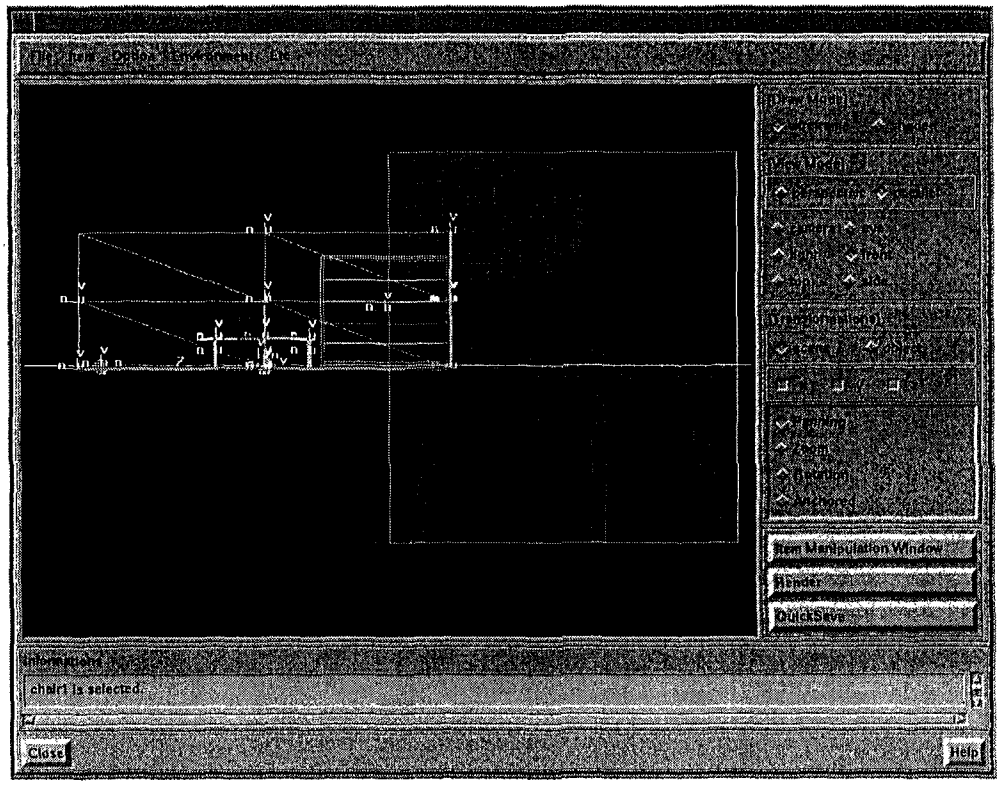


그림 2.38 의자를 회전시킨 화면

이제는 의자의 크기를 작게 하려면, 의자는 계속 선택된 상태로 두고 전경조작도구 중 Transformations에서 scaling선택 후 u, v, n-axis를 모두 선택된 상태에서 크기를 작게 해준다.

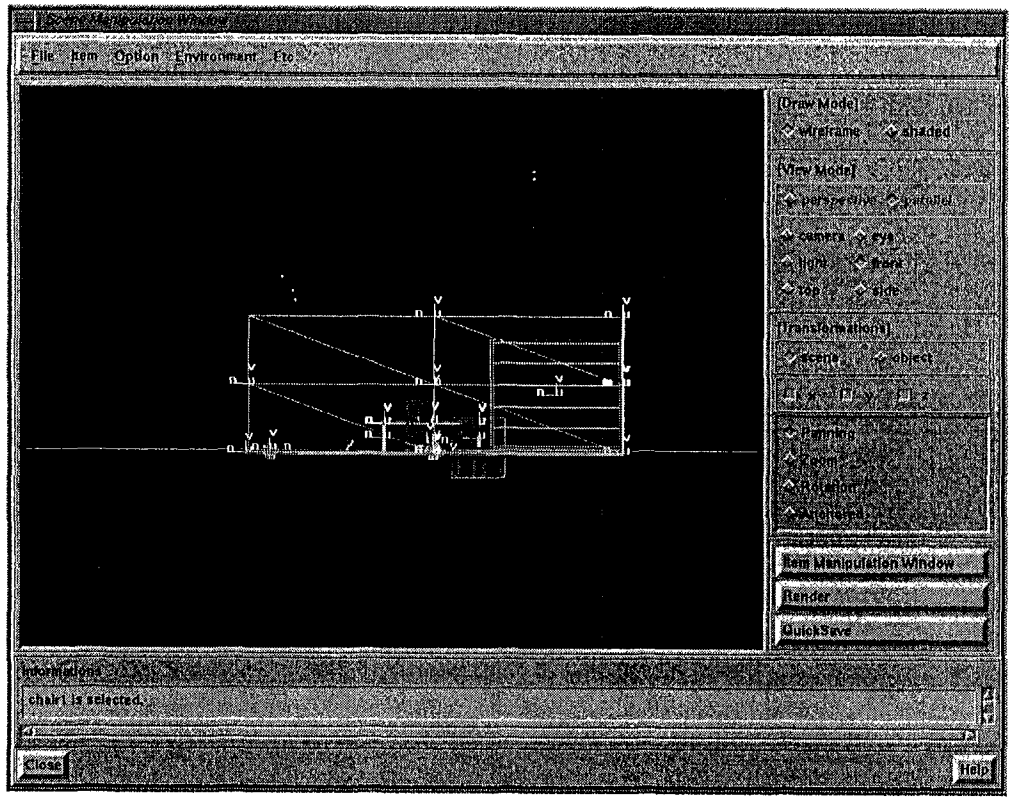


그림 2.39 의자의 크기를 작게 한 화면

마지막으로 이동을 하는데, 전경조작도구 중 Transformations에서 translation을 선택한 후 y만을 이용하여 의자가 바닥에 닿도록 이동시킨 후, View Mode에서 eye로 다시 선택한 후 translation이 선택된 상태에서 x, z축으로만 이동시켜 그림 2.40과 같이 만든다.

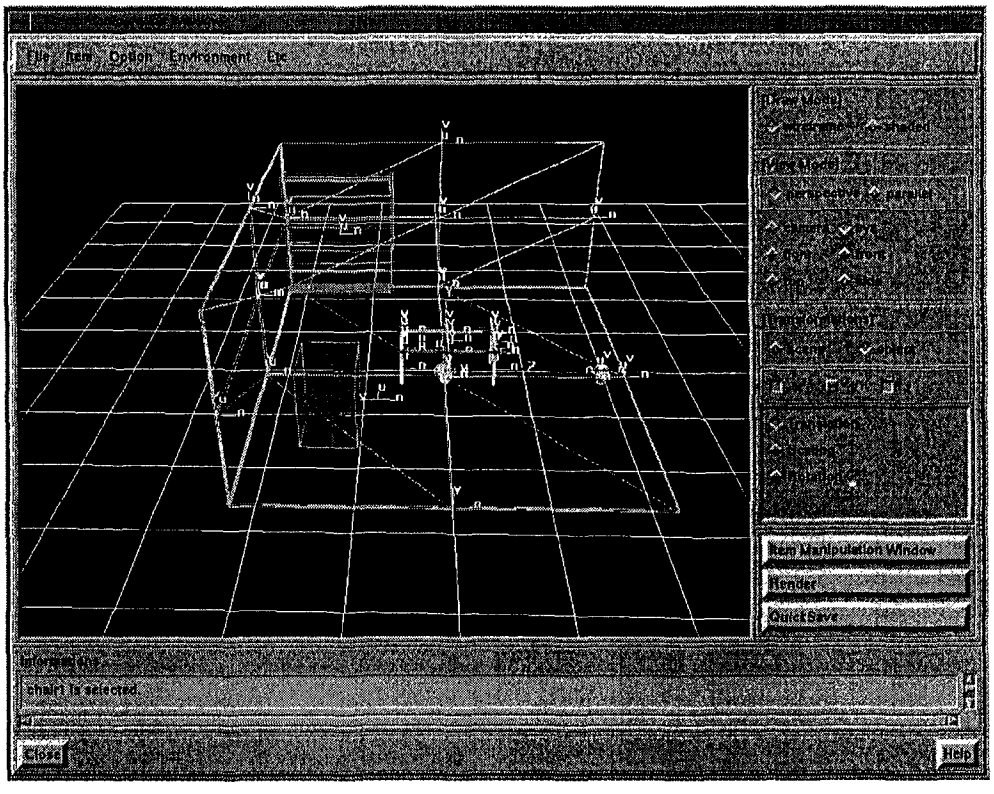


그림 2.40 의자를 이동시킨 장면

사. 물체목록의 복사

위와 같은 방법으로 data나 vrmldata 디렉토리 안에 있는 모든 물체목록 화일들을 전경조작창에 배치 할 수 있다. 하지만 전경조작창에 이미 존재하는 아이템을 하나 더 배치하고 싶을 때에는 아이템 삽입보다 더 편한 방법이 있다. 그것은 물체목록에 대한 복사인데, 우선 전경조작도구 중 Transformations에서 object를 선택한 후 복사하고자 하는 아이템을 선택한다. 그런 다음 메뉴에서 item -> copy를 선택한다. 화면상에는 아무런 변화가 없으나 item이 겹쳐있기 때문에 그러한 것이고 Transformations에서 translation으로 쉽게 다른 곳에 배치할 수 있다.

아. 광원의 추가

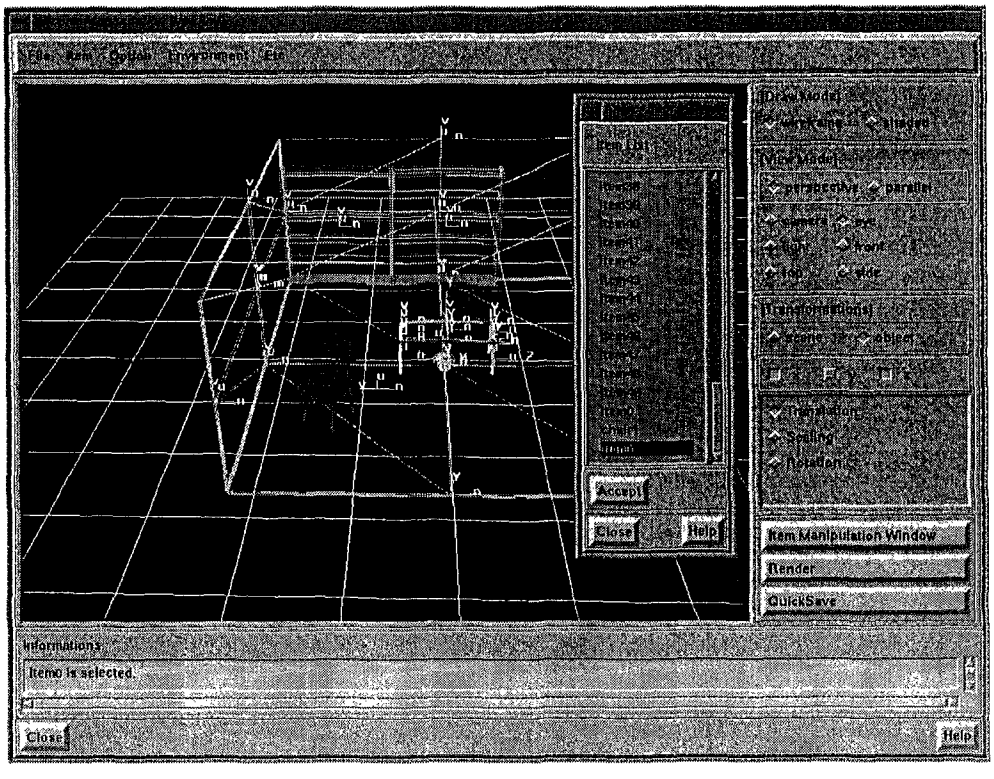


그림 2.41 책장의 복사

현재 작업하는 전경에는 아직 광원이 추가되어있지 않으므로 렌더링에

필요한 광원을 추가해야 한다.

Environment - Light를 선택한다. Default Lights 버튼을 누르고 Create 버튼을 누르면 Light Creation Control Pannel이 나온다. ambient light를 선택하고 Intensity는 0.3 정도로 한 후, Accept 버튼을 누르면 Ambient light가 추가된다. 만약 색깔을 바꾸고 싶다면, Color에서 R, G, B값을 바꾸면 광원의 색깔이 바뀐다.

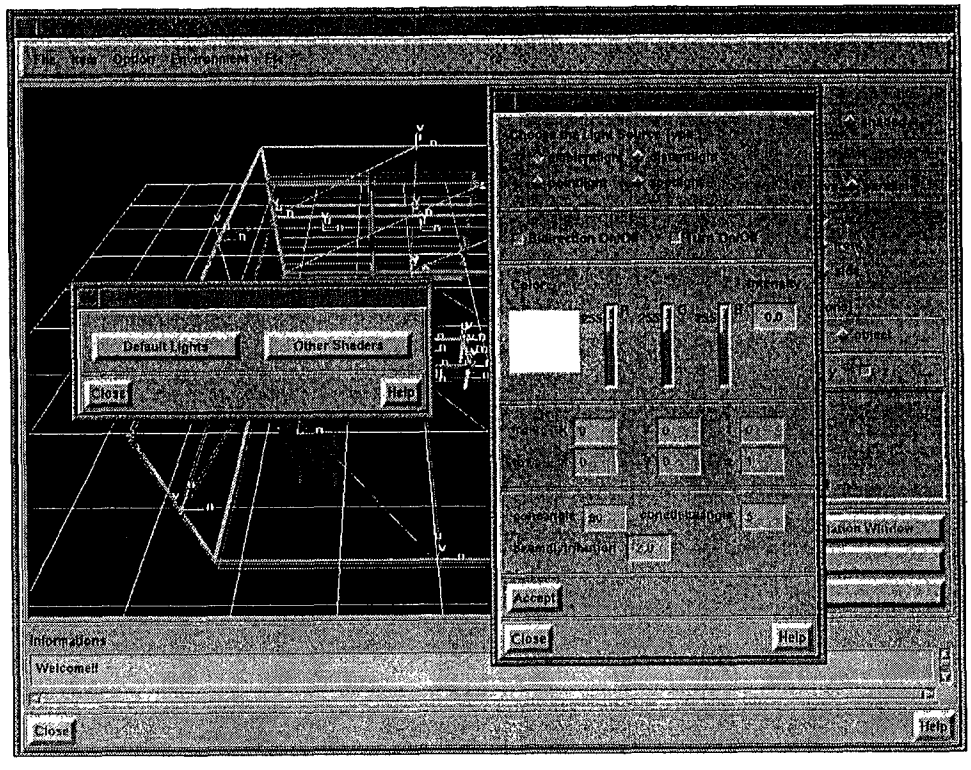


그림 2.42 광원의 추가

자. 렌더링

자신이 원하는 전경으로의 조작이 끝나면 렌더링을 하여 좋은 화질의 이미지를 얻을 수 있다. 렌더링 할 장면은 카메라에 좌우되므로 View Mode를 eye에서 camera로 바꾼다. camera로 바꾸어 화면이 바뀌면 렌더링하고 싶은 장면이 보이게 적절히 camera를 조절한다. camera를 조절하는 방법은 위에서 설명했던 eye를 조절하는 방법과 같다.

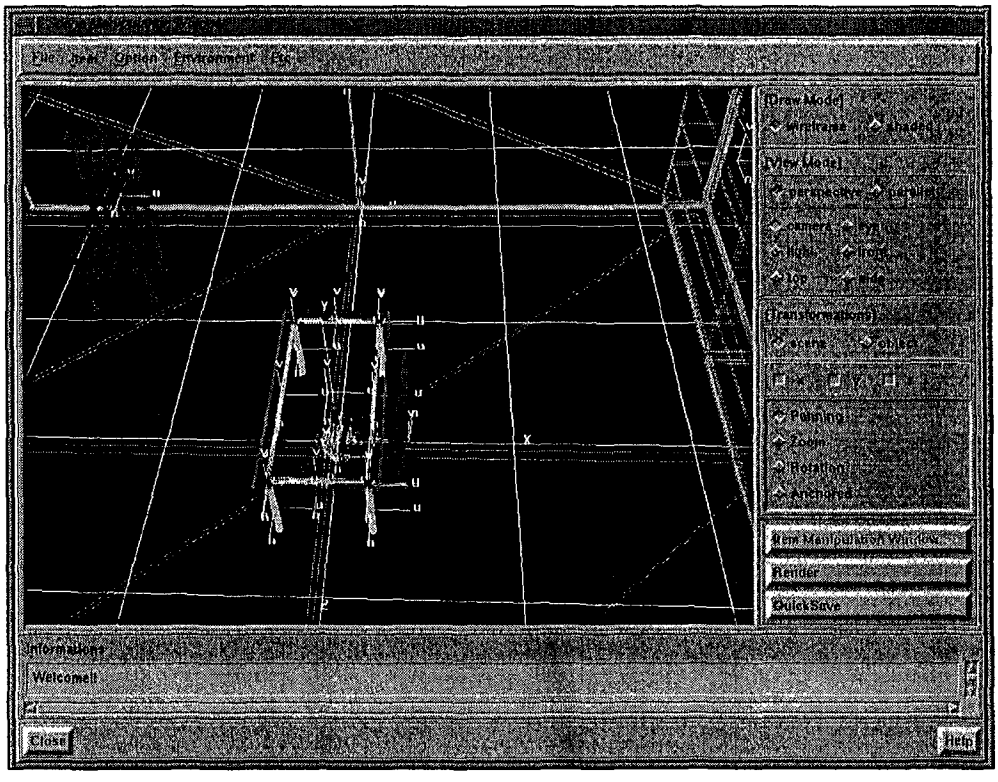


그림 2.43 View Mode를 camera로 바꾼 모습

렌더링할 준비가 끝났으면 이제 Render 버튼을 누른다. Render 버튼을 누르면 Rendered Image 창과 Rendering 창이 나온다.

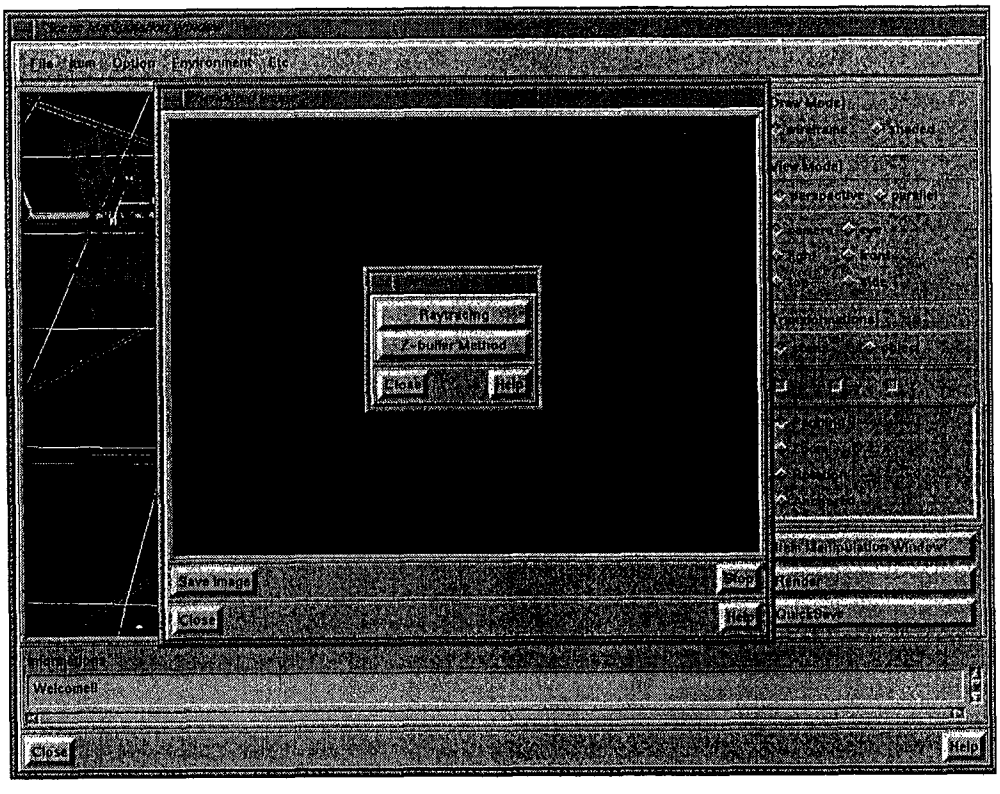


그림 2.44 렌더링 방법의 선택

Rendering 창에서는 렌더링의 방식을 Raytracing과 Z-buffer Method 방법 중 하나를 고를 수 있다. 여기서는 Raytracing 방법을 고른다. Raytracing을 고르면 Raytracing Options 창이 생성된다.

그림자를 생기게 할 것이므로 Compute Shadows 옵션을 On으로 하고 Backface Culling도 On으로하여 렌더링 속도를 빠르게 한다.

Raytracing의 광원 추적을 4번까지 하도록 하고 OK를 누르면 Rendering이 시작된다.

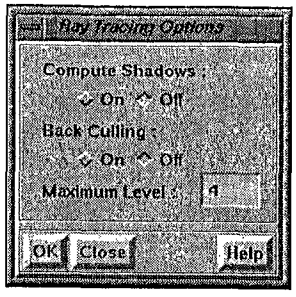


그림 2.45 Ray Tracing 방법의 옵션 설정 창

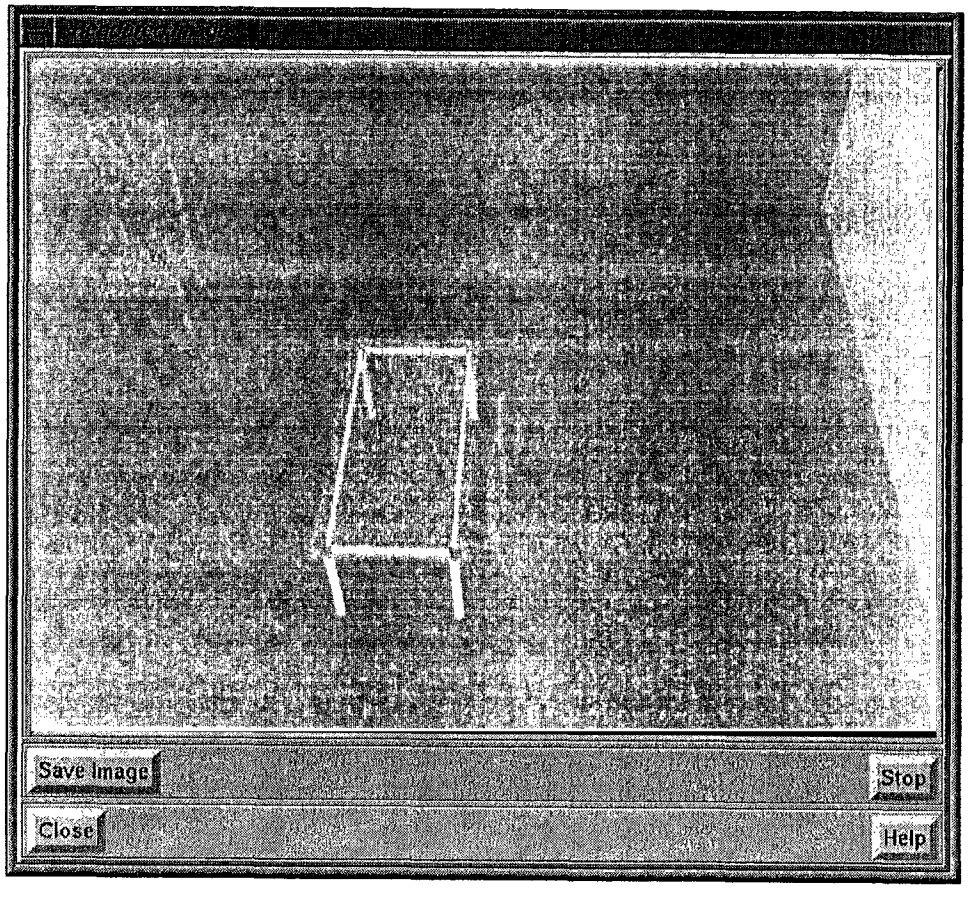


그림 2.46 최종적으로 Rendering한 장면

차. 세이더 지정

물체들은 단색으로 칠하는 것이 아니라, 무늬를 입히고 싶을 경우에는 세이더를 사용한다. 그림에 나와있는 물체 중에서 의자에 나무 무늬 세이더를 입혀보기로 한다.

우선, 세이더를 입힐 의자를 선택한다. 좌측 구석에 있는 의자를 객체 변환 모드에서 선택하자.

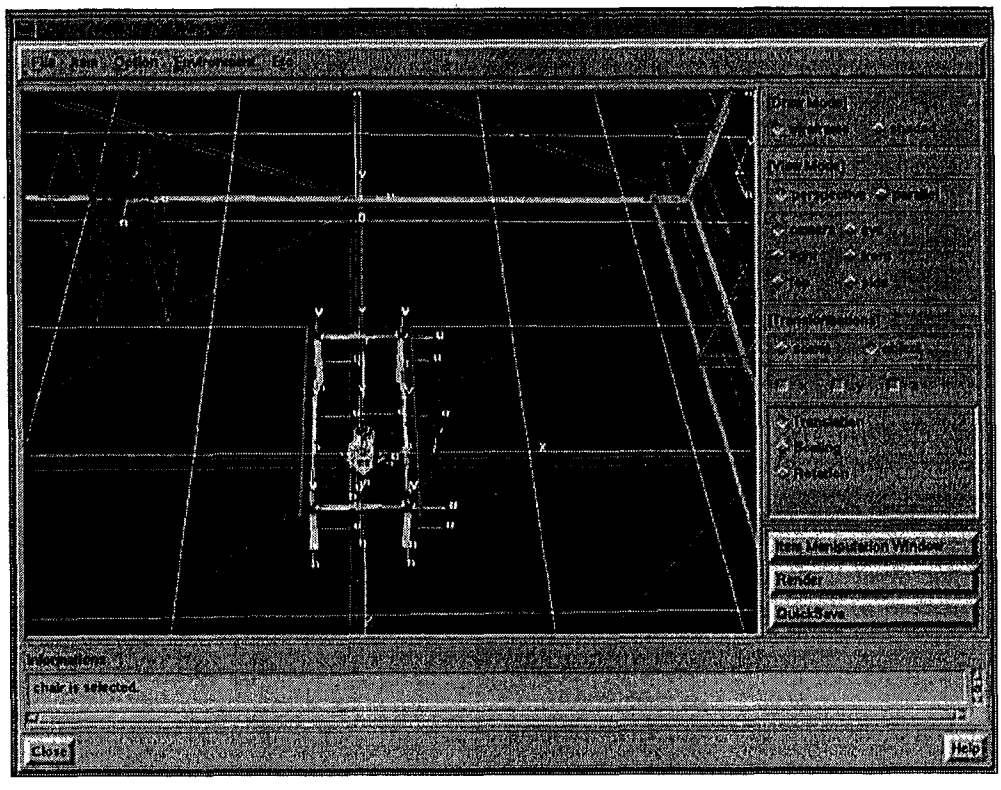


그림 2.47 의자를 선택한 화면

그 다음, Object manipulation window 버튼을 눌러 Object manipulation window를 연다.

세이더는 물체목록 단위가 아니라 물체 단위로만 지정할 수 있기 때문에, 물체 하나로만 이루어진 물체목록이라 하더라도, 물체를 선택해야만 한다. 의자를 다시 선택하자.

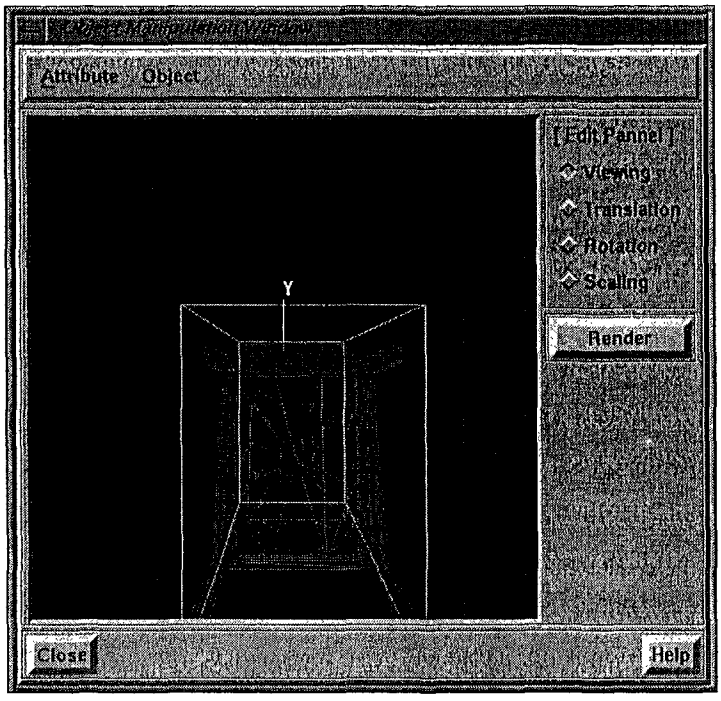


그림 2.48 Object Manipulation 창

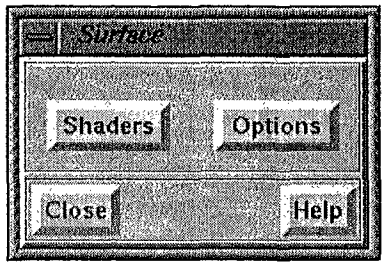


그림 2.49 Surface 세이더 창 (Shaders와 Options 중 선택할 수 있다.)

메뉴에서 Attribute->Surface Shader를 선택한다. 그러면, 그림 2.50과 같은 창이 뜬다.

일단, 옵션은 무시하고, 바로 Shaders를 누른다. 그러면, 등록된 세이더들의 목록이 나오는데, 나무 무늬 세이더를 입히기 위해 기본 세이더중의 하나인 wood를 선택한다.

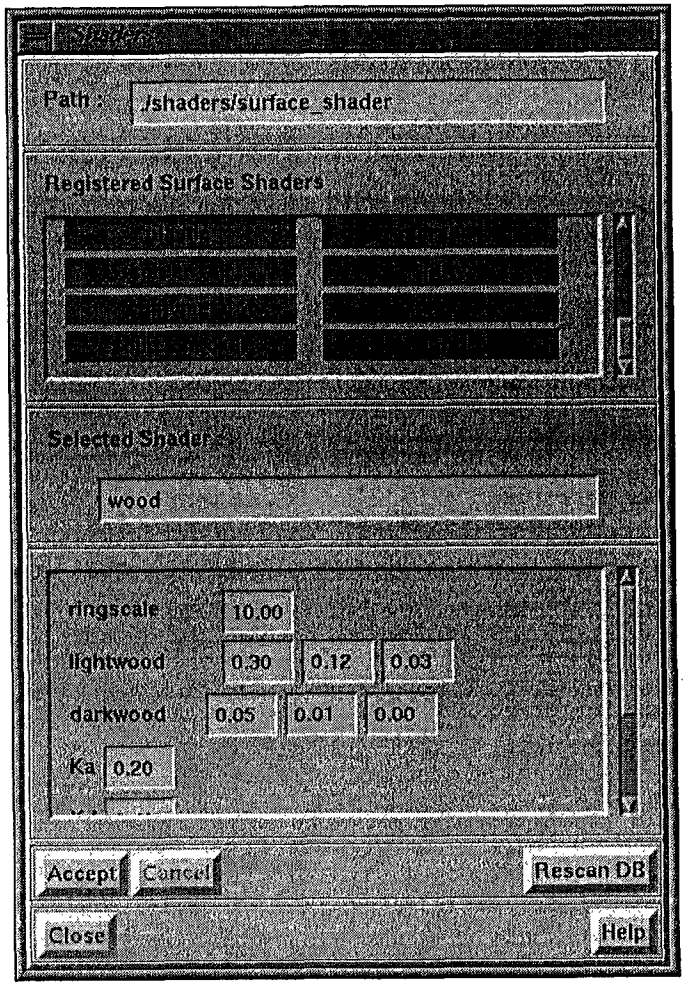


그림 2.50 세이더로 wood를 선택한 화면

Accept 버튼을 누르면, 세이더가 지정된 것이다.

처음의 Scene Manipulation Window로 돌아와, 의자가 잘 보이게 카메라를 조정하고, Render 메뉴를 실행해 확인해보자.

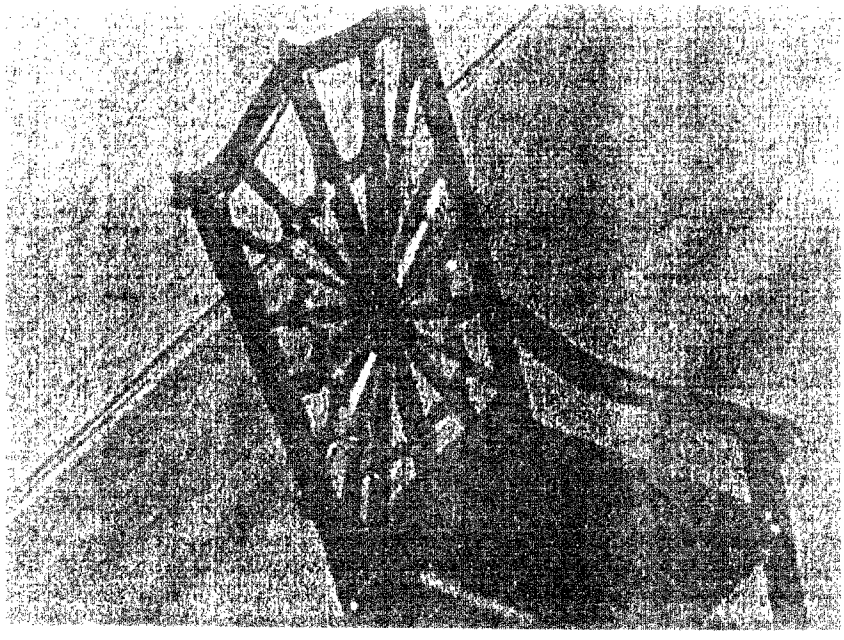


그림 2.51 의자에 세이더를 입히고 렌더링한 모습

카. 텍스쳐 입히기

물체에 셰이더를 입히는 것이 가장 자연스럽지만, 미리 준비된 그림이 물체의 표면에 나타나게 하는 것으로 충분하다면, 텍스쳐를 사용한다.

벽에 텍스쳐를 적용하려면, 벽을 선택해서, Object Manipulation Window를 불러낸 다음 다시 벽을 선택하는 것까지는 셰이더를 입히는 방식과 같다. 그 다음에 Surface Shader 메뉴를 선택하는 대신, Attribute->Texture를 선택한다. 그러면, Texture Mapping Menu가 나온다.

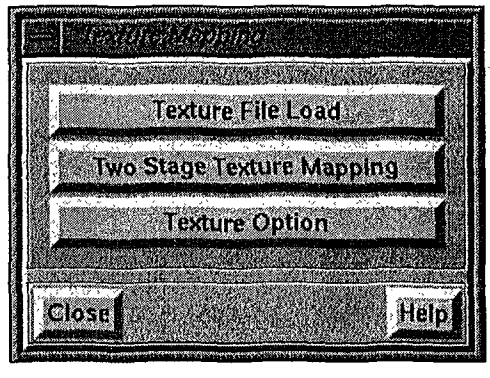


그림 2.52 Texture Mapping 창

여러 가지 옵션을 조정할 수 있지만, 간단한 예를 들기 위해 곧바로 Texture File Load를 선택한다. 확장자가 tif거나 tiff인 파일들의 목록이 나오는데, 이 파일들 중에 아무 것이든 텍스쳐로 지정할 수 있다.

이 중에서 벽에 붙이는 그림으로 box.tif를 선택해보자. 왼쪽 옆의 상자에 tiff 파일의 내용이 나오기 때문에 원하는 그림을 쉽게 고를 수 있다.

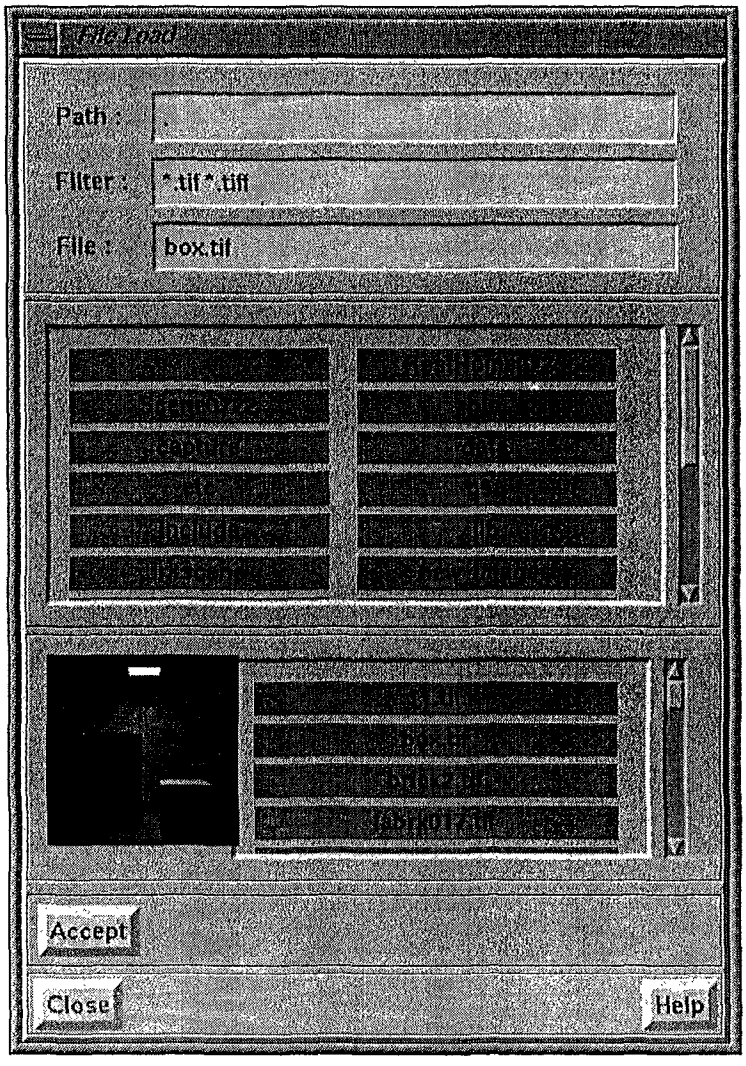


그림 2.53 텍스쳐를 불러들이는 화면

마찬가지로 Accept를 누르면 텍스쳐가 지정된다. 잘 되었는지 렌더링을 걸어 확인해 본다.

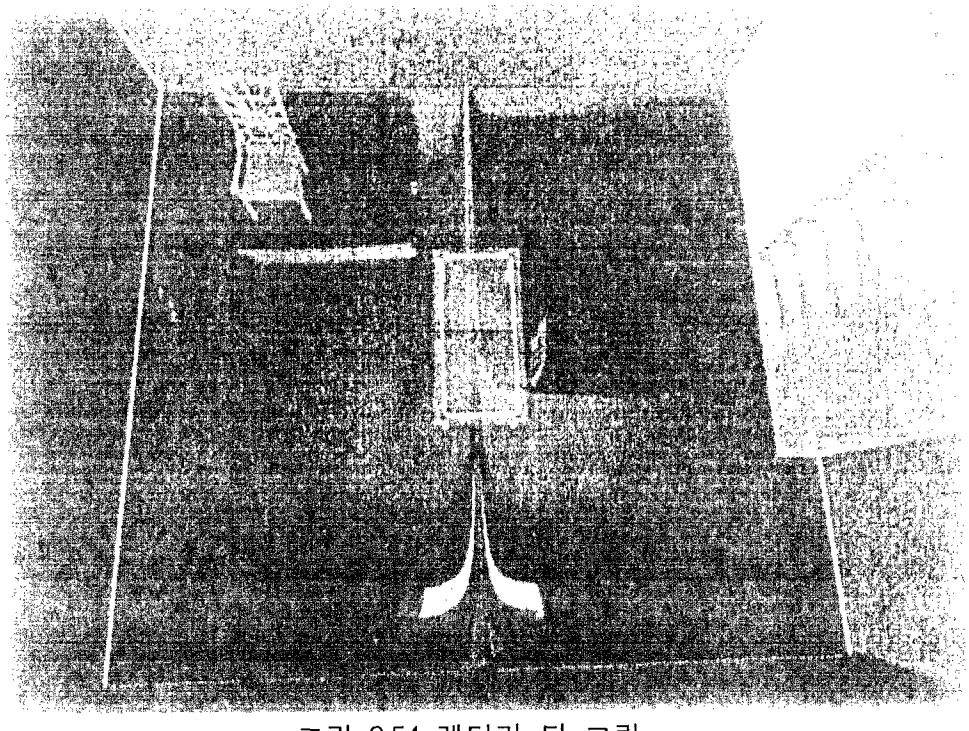


그림 2.54 렌더링 된 그림

타. 전경 저장

메뉴에서 file -> save로 간 다음 sert2.scn으로 저장한다.

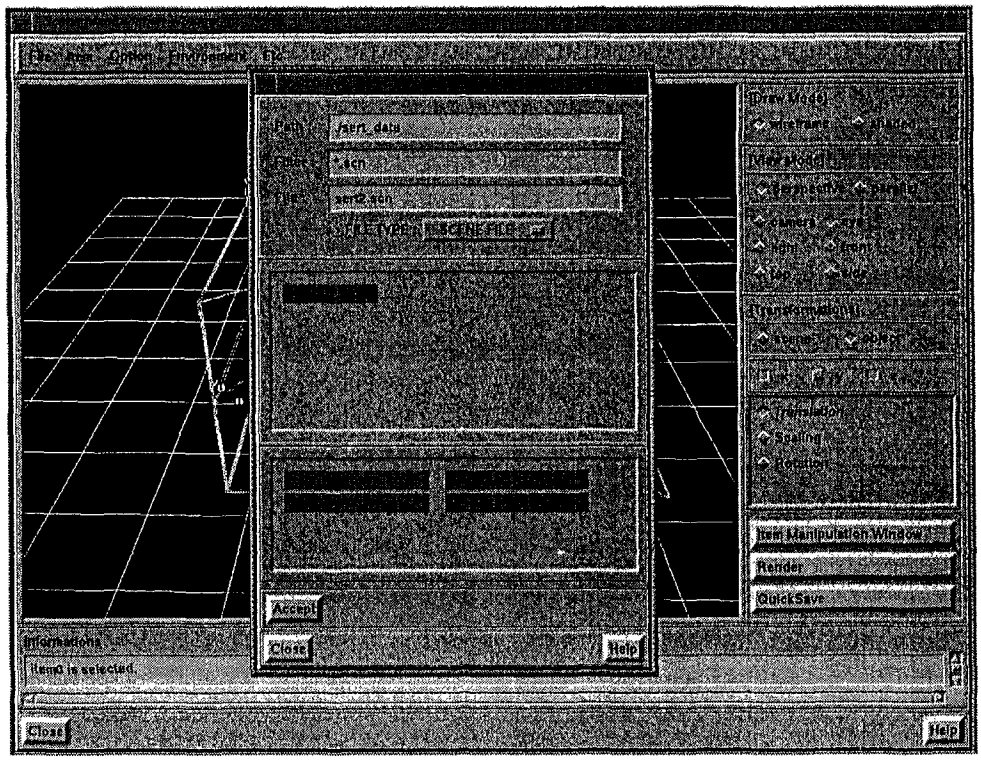


그림 2.55 전경의 저장 화면

제 3 절 가상환경공유를 통한 협력작업의 지원

1. 연구 내용

분산가상환경(distributed virtual environment)이란 다수의 사용자들이 네트워크 상에 만들어진 가상의 공간에 모여서 정보를 공유하고 서로 상호작용 할 수 있도록 해주는 환경을 말한다. 이러한 환경을 구축하게 되면 지역적으로 떨어져 있는 사용자들간의 효과적인 협력 작업(computer supported cooperative work)이 가능하게 되며, 또한 가상 현실(virtual reality)과 같은 응용 소프트웨어를 효율적으로 개발할 수가 있게 된다. 이러한 환경을 구축하는데 있어 중요한 고려 사항중의 하나는 바로 네트워크의 전송 능력이다. 최근의 응용 소프트웨어들은 보다 더 사실적인 가시화(visualization)와 상호 작용(interaction)을 원하고 있으며, 넓은 지역에 위치한 더 많은 사용자가 함께 참여할 수 있는 환경을 요구하고 있다. 따라서 네트워크의 비약적인 발전에도 불구하고 일관성(consistency)과 함께 확장성(scalability)은 분산가상환경에 대한 연구에 있어 가장 중요한 문제들로 간주되고 있다.

본 연구에서는 변화하는 환경과 요구에 맞게 한 명의 사용자만을 지원하던 기존의 시스템에서 탈피하여 여러명의 사용자들이 통신망에 연결된 컴퓨터를 통해 서로 협력적으로 작업할 수 있는 시스템으로 발전시켰다. 여기에는 네트워크와 가상현실의 기법들이 결합되어 마치 여러 명의 사용자가 하나의 가상적인 공간에 같이 들어와 작업하는 것과 같은 효과를 주게 된다. 각각의 사용자들은 화면을 통해 다른 사용자의 작업의 결과와 이동 등을 실시간으로 보게 될 것이며 이 결과는 실제 렌더링에도 영향을 미치므로 사용자들이 전경을 나누어서 효율적으로 작업할 수 있게 된다.

이때 물론 기존 모듈과의 연결을 고려하였으며, 또한 다중 사용자 체제에서 발생할 수 있는 여러 문제점들을 제어하기 위한 구조를 포함하였다.

또한 뒤에 언급될 시스템들의 장단점들을 분석하고 이를 기반으로 하여 분산가상환경 하에서 다중 사용자간의 협력작업(multi-user computer supported cooperative work)을 필요로 하는 응용프로그램 개발에 효과적으로 사용되어질 수 있는 MIRAGE라고 명칭한 네트워크 툴킷을 설계하고 구현하였다. 이 네트워크 툴킷은 다중 사용자들간의 접속과 해제 및 메시지 전송을 관리하는 기능들을 제공하며, 지역적으로 떨어져 있는 사용자들의 원활한 협력작업을 위해 다중 서버를 이용할 수 있도록 하고 있으며, 여러 사용자들이 함께 작업 할 때 문제가 될 수 있는 일관성 유지의 문제를 처리하기 위한 방법을 툴킷 차원에서 지원하고 있다. 위의 기능들을 이용하여 다중 사용자 협력작업을 지원하는 응용 프로그램을 제작할 수 있으며 그 밖의 네트워크 응용 프로그램 제작에도 이용될 수 있다. MIRAGE 툴킷은 독립적인 범용 네트워크 툴킷을 목표로 하고 있어 가능한 일반적인 방법으로 네트워크 기능들을 제공하고 있으며 툴킷 사용자는 각 응용분야에 필요한 기능들에 대한 메시지를 정의하고 주고받음으로써 필요한 작업을 수행하도록 응용 프로그램을 제작하게 된다.

2. 기존의 분산 가상 환경 시스템에 대한 소개

최근에 개발된 대표적인 분산 가상환경 시스템 및 툴킷들을 살펴보면 다음과 같다. 우선 SIMNET[3-8]은 DARPA에서 1985년부터 개발한 분산 상호작용 시뮬레이션에 대한 표준이다. 이는 군사용으로 개발되었으며 초기 훈련 단계에 적은 비용으로 실제 차량을 운전하는 경험을 제공하고 실제 전장에서의 이루어지는 상황을 체험할 수 있도록 하는 것을 목적으로 한

다. SIMNET은 시뮬레이션 표준인 동시에 프로토콜의 역할도 하여 위와 같은 목적에 필요한 PDUs(Protocol Data Units)를 정의하고 있으며 네트워크 대역폭 사용을 가능한 줄이기 위한 데드-레코닝(dead-reckoning) 알고리즘을 제시하고 있다. DIS (Distributed Interaction Simulation)[3-8]는 SIMNET보다 새로운 시뮬레이션 표준이며 프로토콜이다. DIS의 목적은 기본적으로 SIMNET과 같으나 시뮬레이션의 복잡성과 사실성에 차이가 있다. SIMNET이 평평한 지형만을 묘사하고 하늘과 지표만을 가시화하던 것에 비해 DIS는 지형의 곡면을 묘사하며 전장에 영향을 줄 수 있는 가능한 많은 것들을 가시화하는 것을 목적으로 개발되었다. 한편 NPSNET[3-7]은 NPS(Naval Postgraduate School)에서 개발된 네트워크 기반 가상 환경 시스템으로서, SIMNET과 DIS를 계승하고 있으며 대규모 가상환경(large-scale virtual environment)을 구축하는 것을 목적으로 하고 있다. 독립적으로 개발된 시뮬레이터들 간에 어플리케이션 단계의 통신이 가능하도록 하기 위해 DIS 프로토콜을 이용하며 인터넷을 통한 대규모 분산 시뮬레이션을 지원하기 위해 인터넷 그룹 통신의 표준인 IP 멀티캐스팅을 이용하고 있다.

MR Toolkit[3-2]은 헤드 마운트 디스플레이나 3 차원 트랙커, 데이터 글러브와 같은 가상현실 디바이스를 사용하는 가상환경에서의 실시간 상호작용을 제공하기 위한 소프트웨어 개발 도구이다. 소프트웨어의 각 기능들을 팩키지 형태로 제공하고 있어 확장성을 높이고 있는 것이 특징이다. 본래에는 일인 사용자를 위한 사용자 가상환경을 위한 시스템이었으나 피어 팩키지(peer package)가 추가되면서 네트워크를 통한 다중 사용자의 통신이 가능하게 되었다. DIVE[3-5](Distributed Interaction Virtual Environment)는 peer-to-peer구조로 구현된 다중 사용자 분산 가상환경 시스템이다. DIVE에서는 IP 멀티캐스팅을 이용하여 확장성을 높

이는 한편 사용자들의 공유 정보의 일관성을 위하여 하드웨어 멀티캐스팅 위에 소프트웨어적으로 신뢰성을 보장하는 층을 만든 신뢰성 있는 멀티캐스트 프로토콜(reliable multicast protocol)을 이용하고 있다. BrickNet[3-3]은 클라이언트-서버구조로 구현된 네트워크 기반 가상 환경 소프트웨어 툴킷이다. 객체 지향적인 가상 환경 구조를 가지고 있으며 서로 다른 내용을 가지고 있는 클라이언트들이 서버에 연결하여 다른 클라이언트들과의 정보를 공유한다. 마지막으로 Massive[3-6]은 협력 가상 환경(collaborative virtual environment) 연구의 일환으로 개발된 가상 현실 원격 회의 시스템(virtual reality teleconferencing system)이다. 이 시스템은 다중 사용자들이 오디오와 그래픽 또는 텍스트의 임의의 미디어의 조합을 통해 통신하는 것을 가능하게 한다. 또한 이 시스템에서는 작업에 참여한 사용자들의 다른 사용자들에 대한 지각(perception)이 그들 간의 상대적인 위치와 방향에 따라 다르게 이루어질 수 있도록 하는 상호작용의 공간적 모델(spatial model of interaction)에 따라 통신이 제어된다.

3. 상호작용과 통신

가. 다중 사용자 시스템의 구조

본 연구에서는 그림 3.1에 보이는 것처럼 다중 사용자 지원 모듈(multiuser supporting module)을 추가함으로서 확장이 이루어졌다. 다중 사용자 지원 모듈은 크게 두 부분으로 나뉘어 진다. 네트워크를 통한 직접적인 통신에 관여하는 통신 관리자(communication manager)와 사용자 인터페이스, 전경 데이터베이스 등과의 정보 교환을 통해 다중 사용자 시스템의 정보를 일관성을 유지시켜주기 위한 일관성 관리자(consistency

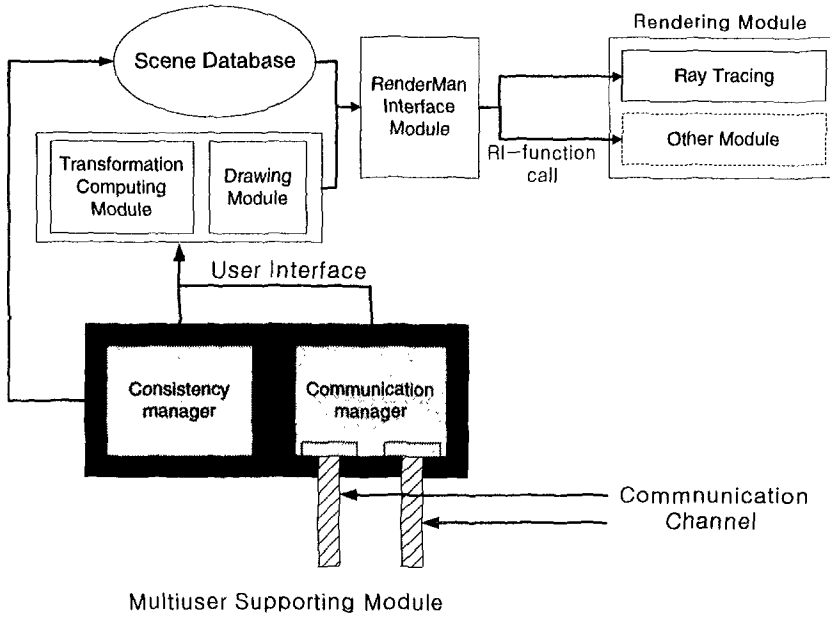


그림 3.1 다중사용자 시스템으로 확장된 구조

manager)가 그것이다.

통신 관리자는 통신에 필요한 모든 업무를 담당한다. 통신에 필요한 각종 초기화 작업, 전송하고자하는 정보의 패킷(packet)화와 전송, 읽어들인 패킷의 일관성 관리자로의 전달 등이 주된 기능이 된다. 통신 관리자는 그림 3.1에서 보듯이 통신을 위해 두 개의 채널(channel)을 사용하는데 하나는 데이터그램 소켓(datagram socket)[3-1]을 사용하는 IP 멀티캐스팅을 위한 채널이고, 또 하나는 스트림 소켓(stream socket)을 사용하는 신뢰성 있는 멀티캐스팅을 위한 채널이다. 이렇게 두 가지 채널을 유지하는 이유와 효용성에 대해서는 뒤에서 자세히 언급될 것이다. 본 시스템에서 전달된 패킷의 인식은 모두 이벤트 유도(event-driven)식으로 이루어진다. 즉, 사용자의 작업도중에 외부로부터 패킷이 전달되어 왔다

면 이 이벤트에 의해 모든 제어는 통신 관리자로 넘어가게 되고 통신 관리자는 두 채널을 모두 살펴보아 들어온 패킷을 읽어들이게 된다. 이 정보는 일관성 관리자로 넘겨져서 패킷 정보의 분석과 그에 따른 작업을 수행한 후에 다시 사용자의 작업을 재개하게 된다. 통신 관리자는 통신 라이브러리의 형태로 제작되었기 때문에 응용 프로그램에 매우 독립적이다. 따라서 이 관리자는 렌더링 시스템이 아닌 다른 응용 분야로의 이식이 가능하다.

일관성 관리자는 들어온 패킷에 알맞은 작업을 수행한다. 입력패킷들이 대부분이 각자의 프로세스가 가지고 있는 데이터베이스 내용의 갱신에 대한 요구이다. 또한 뒤에서 설명하겠지만 특별한 권한을 가진 프로세스에게만 요구되는 작업일수도 있다. 이 경우에는 대개 참가가들의 그룹에 대한 참가와 탈퇴 등, 그룹의 유지를 위한 작업에 해당한다. 이 일관성 관리자는 응용 프로그램에 따라 하는 작업의 내용이 달라지기 때문에 응용 프로그램에 의존적일 수 밖에 없으며, 이 일관성관리자는 이루어지는 작업의 내용에 따라서 통신을 위한 프로토콜이 설계되었다.

나. 그룹의 생성과 통신 모델

한 명의 사용자라도 다중 사용자 기능을 통해 시스템을 수행시키면 그룹이 생성된다. 시스템이 시작하면 먼저 이 시스템이 시작하면 먼저 이 시스템은 자신이 그룹에 참가하고자 한다는 내용을 알리는 조인(join)패킷을 멀티캐스팅 한다. 적정한 시간이 지나도록 아무런 반응이 없을 경우, 이 시스템은 네트워크 상에 어떠한 가상 작업환경도 구축되지 않는 것으로 인식하고 자신이 그룹을 생성하는 리더 프로세스가 되며, 그 프로세스가 있는 호스트에 신뢰성 있는 멀티캐스팅을 위한 서버를 구동시킨다. 리더 프로세스에 의해 생성된 서버 프로세스는 멀티캐스팅을 위한 여

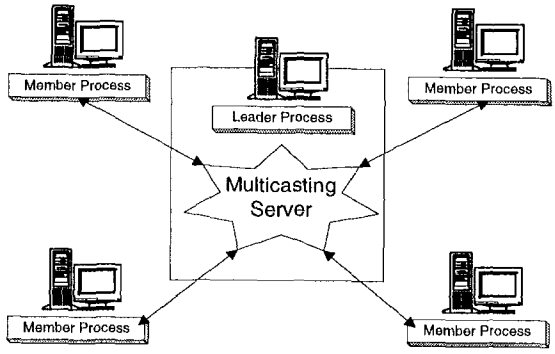


그림 3.2 네트워크상에서의
시스템들간의 통신 모델

러 정보들을 초기화하고 신뢰성 있는 멀티캐스팅을 위해 리더 프로세스와 연결(connection)을 이룬다. 만약 이미 다른 리더 프로세스가 존재할 경우, 이 리더 프로세스는 조인 패킷을 받아들여서 그룹에의 참가를 허가하는 패킷을 전달하고 그룹 참가에 필요한 부수적인 정보들을 전달하게 된다. 이 정보 중에는 현재 리더 프로세스와 멀티캐스팅 서버가 있는 기계에 대한 정보가 포함되기 때문에 새로 참가하는 프로세스는 멀티캐스팅 서버와의 커넥션을 이루게 되어 최종적으로 그림 3.2의 통신 모델과 같은 그룹을 형성하게 된다.

위에서 언급한 바와 같이 본 시스템에서는 IP 멀티캐스팅 외에도 신뢰성 있는 멀티캐스팅을 사용한다. 이것은 전달하고자 하는 정보의 특성에 따른 전달방법의 차별화에 의한 것이다. 예를 들어 가상공간상에서 다른 참가자의 움직임을 전달하고자 할 때는 그 물체의 위치정보나 변환 행렬 등을 전달해야 하는데 이런 정보들은 그리 정확한 전달방법을 요구하지 않는다. 왜냐하면 움직이는 여러 프레임들 중에서 한 프레임 정도에 해당하는 정보의 변형은 그리 큰 영향을 미치지 않기 때문이다. 하지만 그런 정보들은 실시간으로 전송되어 보여져야 하기 때문에 빠른 전송 속도를

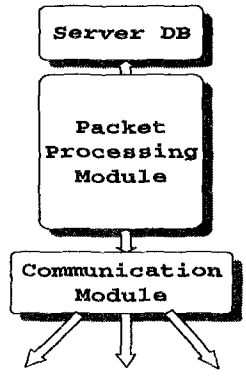


그림 3.3 신뢰성 있는 멀티 캐스팅을 위한 서버의 구조

요구한다. 그러나 물체의 색을 변경했다거나 텍스쳐를 입힌 경우, 이 사실을 전달하기 위한 정보들은 그리 빈번하게 발생하지는 않지만 정보가 도중에 유실되거나 순서가 달라질 경우 시스템의 운영과 데이터베이스의 일관성이 치명적일 수 있다. 이런 각 정보들의 특성에 대한 관찰의 결과로 빠른 전송 속도를 요구하거나 그리 정확한 전달을 필요로 하지 않는 정보들은 IP 멀티캐스팅을 이용하고 빈번하지는 않으나 정확한 전송을 요구하는 정보들은 신뢰성 있는 멀티캐스팅을 이용하는 전송방식의 이분화 전략을 채택했다.

본 연구에서는 신뢰성 있는 멀티캐스팅을 구현하기 위해 이를 위한 서버를 병행하여 개발하였다. 이 방법은 DIVE 시스템에서 취했던 접근방식과 같은데, 각 프로세스들은 그림 3.2에서와 같이 이 서버와 스트림 소켓을 통해 신뢰성 있는 통신을 위한 커넥션을 유지하게 된다. 본 시스템을 위해 설계된 신뢰성 있는 멀티캐스팅을 위한 서버의 구조는 그림 3.3과 같다.

여기서 통신 모듈(communication module)은 서버로 들어오는 패킷이 있는가를 항상 검사하게 된다. 만약 어떤 커넥션을 통해 들어오는 패킷이 있다면 이 정보는 패킷 처리 모듈(packet processing module)로 전달이 되는데, 여기서는 패킷의 내용에 따라 여러 가지 작업이 있을 수 있다. 단순한 멀티캐스팅을 위한 패킷일 수도 있고, 선택적인 전송을 필요로 하는 패킷일 수도 있다. 마지막의 경우에는 서버 데이터베이스의 정보를 갱신하는 작업까지 필요로 하는데, 서버 데이터베이스는 현재 그룹에 가입한 프로세스들의 정보를 가지고 있으며 이들은 멀티캐스팅 외에도 뒤에서 설명하는 사용자들간의 자원(resource)관리를 위한 정보로 사용되기도 한다.

다. 데이터베이스와 통신 프로토콜

여러 사용자들이 참가하는 가상공간에 대한 데이터베이스는 언제나 일관적이어야 한다. 이 말은 곧 한 사람이 보는 물체의 모양이나 속성들이 다른 사람이 같은 위치, 같은 시점에서 보았을 때 똑같아야 한다는 것이다. 또한 이런 가상공간의 정보들은 실시간으로 사용자의 시점에 따라 화면에 보여져야 하므로 각 프로세스들마다 데이터베이스의 복사본을 유지해야 한다. 각 프로세스들마다 중복하여 정보를 유지한다는 것은 매우 비효율적으로 보이겠지만 실시간 작업을 위해서는 매우 유리하다. 화면에 보여질 전경의 매 프레임마다 계산을 각 프로세스가 담당하면 전경 데이터의 전송량이 줄어들기 때문이다. 이렇게 각 프로세스마다 전경 데이터베이스를 중복하여 가지는 경우 각 데이터베이스들간의 정보를 일관성 유지가 매우 중요해지는데, 이를 위해 위에서 언급한 바와 같이 신뢰성 있는 멀티캐스팅을 이용한 것이다.

현재 프로세스에서 갱신된 데이터베이스의 내용은 다른 프로세스들에게

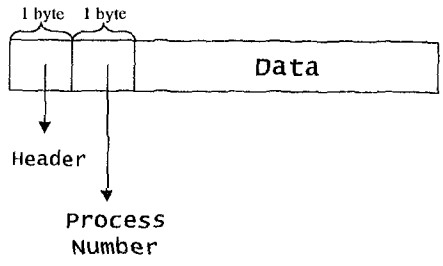


그림 3.4 통신을 위한
프로토콜의 구조

전달되어 갱신되어야 한다. 이를 위해 각 프로세스들간의 통신을 위한 프로토콜을 설계하였는데 프로토콜의 설계는 전송 속도와 작업의 효율성에 영향을 미치는 이유로 최소의 정보량을 목표로 설계되었다. 본 시스템에서 사용하는 프로토콜의 구조는 그림 3.4와 같다. 그림에서 보는 바와 같이 프로토콜의 첫 바이트(byte)는 이 패킷이 전달하고자 하는 정보의 의미를 나타내는 헤더(header)부분이고, 그 뒤의 한 바이트는 이 패킷을 전송한 프로세스의 번호를 저장하고 있다. 리더 프로세스는 참가하는 프로세스마다 각각 고유의 번호를 부여하는데, 이 번호가 전송하는 패킷마다 들어가게 된다. 그 뒤로 실제 필요한 데이터들이 저장되는데 이 데이터들은 헤더의 내용에 따라 가변적인 크기를 가지며 다양한 형태의 정보들이 담겨있다.

통신 관리자가 수신한 패킷은 일관성 관리자에게로 전달되는데, 여기서는 이 패킷의 헤더와 프로세스 번호를 해석하여 이 정보에 따라 데이터를 알맞은 방법으로 처리하게 된다. 전달된 패킷의 내용은 크게 데이터베이스의 갱신을 요구하는 것과 그룹의 유지와 관리를 위한 정보를 전달하는 패킷으로 나뉘어진다. 각각의 식별자들은 리스트 상에서의 번호와 트리 구조 내에서의 탐색순서에 기초한 번호를 전달함으로서 구현되었다. 이는

각 물체 목록들이나 광원들에 할당된 이름들이 중복되거나 존재하지 않아도 되도록 했기 때문에 이름을 식별자로 사용할 수 없는 이유에서이다. 또한 셰이더나 텍스쳐와 같이 정보량이 많은 데이터들은 여러 개의 패킷으로 나누어서 보내게 되는데 수신 프로세스의 입장에서는 이렇게 분리되어 들어오는 정보들을 재구성하여 원래의 정보로 복원하는 일을 수행할 수 있어야 하는데 이러한 작업 또한 일관성 관리자에서 하게 된다.

4. 다중 사용자 제어 구조

가. 사용자들간의 우선권과 자원 관리 방법

가상환경을 지원하는 다중 사용자 시스템에 있어서 사용자들이 서로간의 존재를 인식할 수 있는 기능의 지원은 필수적이다. 각 사용자들이 서로의 모습을 인식하기 위해서는 각 사용자를 나타낼 수 있는 시각적인 객체가 존재해야 하는데 이것을 아바타(avatar)라 한다. 대개의 가상환경 시스템에서의 아바타는 사람의 형상을 띄는 것이 보통이지만 본 시스템에서는 렌더링 시스템의 특성상 사람 모양의 객체들이 빈번하게 처리해야 할 데이터로서 나타낼 수 있기 때문에 이를 피하고 안경 모양의 아바타를 도안하였다. 이 아바타는 사용자마다 각각의 색이 다르므로 색으로서 구분이 가능하며 각 아바타의 움직임으로서 그 사용자의 움직임을 인식할 수 있다. 또한 각 사용자가 전경 내 자원을 선택하여 조작할 수 있으므로 이러한 움직임들 또한 화면으로 확인할 수 있다.

그러나 이렇게 가상공간상에서 여러 사용자가 동시에 작업을 하게 되는 경우 여러 문제점을 야기할 수 있다. 예를 들어 어떤 사용자가 새로운 전경을 읽어들이게 되었다면 데이터베이스의 일관성을 위해 다른 사용들도 이 전경을 읽어들이게 된다. 또한 어떤 참가자가 점유하고 있는 객체를

다른 참가자가 동시에 이를 조작할 수 있다면 서로간에 큰 혼란이 일어날 수 있다. 그렇기 때문에 다중 사용자 시스템에서는 이런 파일이나 전경내의 자원들에 대한 관리가 매우 중요한 문제이며 이런 문제점을 해결하기 위한 노력을 병행하였다.

본 시스템에서는 파일에 대한 관리는 사용자마다 차등적인 권한(priority)을 부여함으로서 해결하였다. 모든 파일에 다한 권한은 리더 프로세스만이 갖기 때문에 다른 사용자들은 전경을 읽거나 저장할 수 없다. 그러나 물체 목록의 삽입은 어떤 프로세스라도 가능하다. 그리고 전경내의 자원에 대해서는 잠금(locking)기법을 사용하여 한 사용자가 점유하여 조작하고 있는 자원에 대해서는 다른 참가자들이 접근 할 수 없도록 하였다. 만약 그 자원을 조작하고 싶다면 그 자원에 대한 잠금이 해제될 때까지 기다려야 할 것이다. 이를 실제로 구현하는데 있어서 다음에서 설명하는 2단계 잠금(2-phase locking)기법을 고안하였다.

나. 2 단계 잠금 기법

각 프로세스들은 자신을 제외한 다른 참가자들이 현재 어떤 자원을 점유하고 있는지에 대한 정보를 가지고 있다. 이것은 참가자들이 하나의 자원을 점유하게 되었을 때 이 정보를 다른 프로세스들에게 전달함으로서 유지되는데, 이 정보들을 이용하여 잠금 조작이 이루어진다. 한 사용자가 원하는 자원을 선택했을 때 이 자원이 현재 다른 사용자들이 점유하고 있는 자원들과 일치하는지를 검색한다. 만약 이미 점유되었을 경우 사용자의 이 자원에 대한 접근은 거부된다. 점유되지 않았을 경우 이 자원을 그 사용자가 점유할 수 있도록 하고 이 정보를 다른 프로세스들에게 전달시키는 것이 기본적인 과정이다.

그러나 만약 동시에 두 사용자가 점유되지 않은 하나의 자원에 대한 접

근을 시도했다면, 아직 다른 참가자의 점유 사실이 전달되기 전이므로 두 사용자가 동시에 접근을 허락 받은 상황이 발생할 수 있다. 이런 경우를 막기 위해 본 시스템에서는 신뢰성 있는 멀티캐스팅을 위한 서버에서도 잠금 조작을 수행할 수 있도록 하였다. 즉, 서버에서도 각 참가자마다 점유하고 있는 자원에 대한 정보를 서버 데이터베이스에 저장하고 있어서, 자원의 점유에 대한 최종적인 허가는 여기서 내리게 된다. 프로세스 차원에서의 잠금 검사를 통과하였다면 그 프로세스는 서버에게 이 자원을 점유해도 되는지를 질의하고, 서버는 자신의 정보에 의거하여 허가나 거부를 하게 된다. 만약 허가가 될 경우 서버는 다른 프로세스들에게 자원의 점유 사실을 전달하게 되고, 허가를 원한 프로세스는 그 자원에 대한 접근을 할 수 있게 된다. 이런 2단계 잠금을 사용하게 되면 동시에 사용자가 점유를 원해도, 서버가 일을 처리하는 것은 순서적이므로 뒤에 들어온 점유 요구는 거부되어 안정적인 관리를 할 수 있게 된다. 이런 2단계 잠금 조작에 대한 개요가 그림 3.5에 잘 나타나 있다.

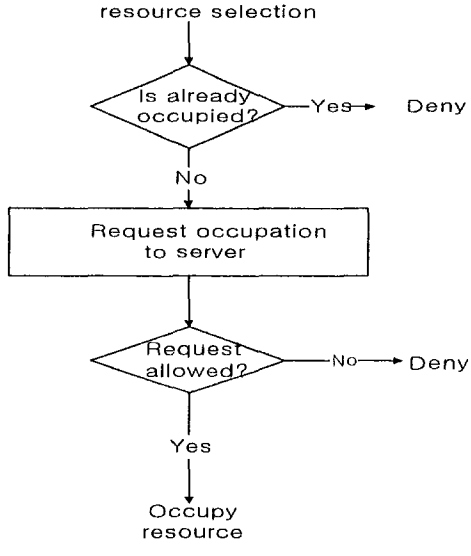


그림 3.5 2단계 잠금 알고리즘

5. MIRAGE 네트워크 툴킷의 소개

가. 네트워크 툴킷의 구조

MIRAGE는 분산 가상환경의 구축 및 다중 사용의 협력작업을 지원하기 위해 제작된 네트워크 툴킷으로 그림 3.6과 같은 구조를 갖는다. 본 툴킷은 소켓 인터페이스(Berkeley socket interface)[3-1]를 이용하고 있으며, 크게 시스템 층(system layer)과 응용 층(application layer)으로 나뉜다. 시스템 층은 기본적인 네트워크 기능을 제공하는 함수들을 제공하며 응용 층은 특정한 방식의 응용프로그램을 작성할 때 그에 알맞은 편의 함수들을 제공한다. 응용 층의 함수들은 내부적으로 시스템 층의 기능들을 호출하여 필요한 작업을 하게 된다. 시스템 층은 다시 네트워크 기반 모듈(network base module)과 일관성 유지 모듈(cobject module)로 나뉜다. 네트워크 기반 모듈은 네트워크의 기본적인 기능들을 다루는데, 크게 접속과 해제를 담당하는 연결 모듈(connection module)과 메시지의 전송과 메시지 핸들러를 유지 관리하는 메시지 핸들링 모듈(message handling module)로 구성된다. 일관성 유지 모듈은 여러 사용자가 동시 다발적으로 공유 정보의 내용을 변경시키려 할 때의 문제들을 처리하는 기능을 제공한다. 한편 응용 층에서는 클라이언트-서버구조로 응용 프로그램을 작성할 때 필요한 클라이언트 모듈(client module)과 서버 모듈(server module) 그리고 이를 확장한 다중 서버 모듈(multi server module)이 제공된다. 응용 프로그램은 직접 시스템 층을 이용하여 제작될 수 도 있으며, 클라이언트-서버 구조를 이용하는 경우에는 클라이언트 모듈과 서버 모듈을 이용하여 제작될 수 있다.

나. 데이터 전송 모델

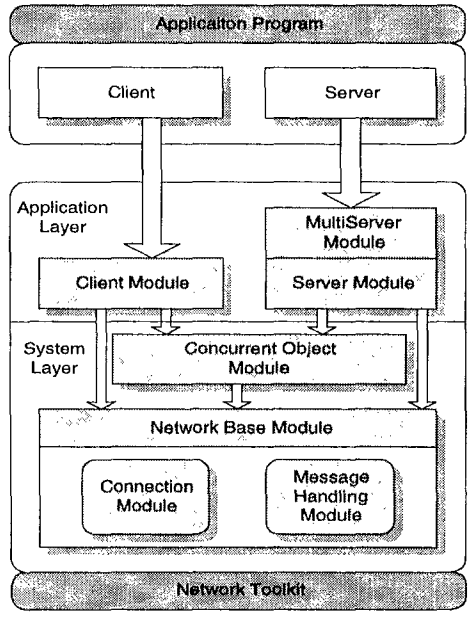


그림 3.6 네트워크 툴킷 구조

다중 사용자 분산 환경에서 사용할 수 있는 데이터 전송 방식은 다음과 같이 크게 3가지로 나눌 수가 있다[3-4]. 유니캐스팅(unicasting)은 두 호스트간에 일대일의 접속 관계를 유지하며 한번에 한 호스트에만 메시지를 보낼 수 있다. 사용자의 수가 증가하면 네트워크 접속 관계도 많아지며, 한 메시지를 여러 호스트에게 전송하는 경우 여러 번의 메시지 전송이 필요하여 많은 호스트가 참가할 경우 확장성을 제한하는 요인이 된다. 브로드캐스팅(broadcasting)의 경우에는 한번의 메시지 전송으로 모든 호스트에 메시지를 전송하게 된다. 이는 한번에 복수개의 메시지 전송을 처리하므로 많은 호스트들이 협력작업에 참여하는 경우 확장성을 높이는 역할을 하게 된다. 그러나, 협력 작업에 참여하고 있지 않은 호스트들에게도 메시지가 전송되므로 OS(operating system) 수준에서 메시지가 도착했을 때 이 메시지가 해당 호스트에 전달 되야 하는 메시지인지 아닌지를 검사해야 하므로 작업 성능을 낮추는 요인이 된다. 멀티캐스팅

(multicasting)의 경우에는 상대적으로 최근에 지원되기 시작한 방식으로 원하는 호스트 그룹에 메시지를 전달하는 것이 가능하다. 한번의 메시지 전송으로 필요한 호스트에만 선택적으로 메시지 전송이 가능하므로, 유니캐스팅에서 발생하는,호스트의 수가 많아질 때 메시지 전송도 많아지는 문제와 브로드캐스팅에서 발생하는 협력 작업에 참여하지 않은 호스트들의 작업이 지연되는 단점을 해결 할 수 있다.

MIRAGE 툴킷에서는 신뢰성을 보장하는 TCP 유니캐스팅과 신뢰성이 보장되지 않는 IP 멀티캐스팅의 두 가지 데이터 전송 프로토콜을 병렬적으로 지원한다. 즉 반드시 올바르게 전송되어져야 하는 정보는 TCP를 통해 전송하고 그렇지 않은 정보는 IP를 통해 전송하여 신뢰성과 전송 속도를 최대한 만족시킬 수 있도록 하고 있으며, 한 응용 프로그램에서 두가지 전송 채널을 함께 사용하여 필요에 따라 채널을 선택하여 데이터를 전송할 수 있도록 하였다.

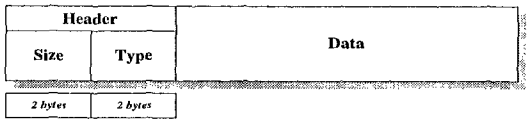


그림 3.7 메시지 패킷 구조

다. 메시지 패킷의 구조 및 핸들링 모델

본 네트워크에서 사용하는 메시지 패킷의 구조는 그림 3.7과 같다. 각각 2바이트의 메시지 크기(Size)와 메시지 종류(Type)가 메시지 헤더(Header)를 구성하며 이후에 실제 메시지를 구성하는 데이터(Data)가 온다. TCP 유니캐스팅을 이용하는 경우나 IP 멀티캐스팅을 이용하는 경우나 메시지의 구조는 항상 같으며 이러한 패킷을 통해 표 3.1과 같은 3가지 종류의 메세지가 전송된다. 시스템 메시지는 네트워크 툴킷의 고유의 작업을 위해 사용된다. 예를 들어 두 호스트 간의 접속이나 연결 해제

표 3.1 메시지 종류

메시지 구분어	메시지 종류
MESG_SYS	시스템 메시지
MESG_USYS	유저 시스템 메시지
MESG_USER	유저 메시지

(CONNECTION REQUEST, CONNECTION DISCLOSE), 일관성 유지를 위한 메시지(COBJECT ALLOC, COBJECT LOCK)등 툴킷이 기본적으로 제공하고 있는 기능들이 이 메시지를 주고받음으로써 구현된다. 시스템 메시지에 대한 핸들러는 툴킷 자체에서 제공하며 일반 사용자는 개입할 수 없다. 또한 특별한 경우를 제외하고는 사용자가 직접 이 시스템 메시지를 전송하는 일은 없으며 네트워크 작업의 각 단계에서 정의된 함수를 호출하면 결과로 해당하는 시스템 메시지가 전송된다. 유저 시스템 메시지는 네트워크 작업의 각 단계 중 정의된 시점에서 발생되어 사용자에게 전달된다. 예를 들어 네트워크 접속이 성공한 후에는 AFTER CONNECTION ACCEPTED라는 메시지가 발생하며 새로운 메시지를 전송 받은 후에는 AFTER MESSAGE RECEIVED란 메시지가 발생한다. 이러한 유저 시스템 메시지는 실제로 네트워크 통해 전달되는 메시지는 아니며 네트워크 툴킷을 이용하고 있는 각 사용자의 프로세스에서 특정한 순간에 발생되어 해당 사용자에게만 전달되는 메시지이다. 사용자에게 메시지가 전달되는 방식은 메시지 핸들러를 통한다. 즉 사용자가 필요한 유저 시스템 메시지에 메시지 핸들러를 등록시켜 놓으면 유저 시스템 메시지가 발생할 때 해당 메시지 핸들러가 호출된다. 이러한 기능을 이용하면 디버깅 과정 중에 쉽고 일반적인 방법으로 툴킷이 작동하는 상황을 모니터할 수 있으며 그 외에도 사용하기에 따라서 매우 유연하고 강력한 프로그래밍이 가능하다. 마지막으로 유저 메시지는 실제로 사용자가 작성하는 네트워크 응용 프로그램의 작업을 위해 사용되는 메시지이다. 툴킷에서 요구하는 조건에 벗어나지 않도록 유

저가 정의하고 그에 해당하는 메시지 핸들러를 등록하면 메시지를 전송받았을 때 그에 필요한 핸들러가 호출되어 원하는 작업을 수행하게 된다.

본 네트워크 툴킷을 이용한 응용 프로그램의 제작은 유저 메시지를 정의하고 필요한 메시지들에 대한 핸들러를 제작하여 툴킷에 등록시키는 작업으로 이루어진다. 메시지 핸들러의 구조는 다음과 같이 메시지가 전송되어져 온 소켓의 디스크립터, 메시지의 종류, 크기 그리고 데이터로 이루어진다.

```
void message_handler( int sd, int type, int size, char *buf );
```

위에서 설명한 내용들은 TCP 유니캐스팅을 이용한 경우와 IP 멀티캐스팅을 이용한 경우에 차이가 없으며, 동시에 여러 개의 네트워크 소켓을 이용할 경우 메시지 핸들러를 등록하는 방식에 따라서 각 소켓마다 다른 메시지 핸들러를 이용할 수도 있으며 또한 모두 같은 메시지 핸들러를 이용할 수도 있다. 이러한 능력은 어떤 한 프로세스가 서버가 되는 동시에 클라이언트가 되려고 할 경우에 각 메시지 핸들러를 따로 관리하여 보다 체계적인 프로그래밍을 가능하게 해주며, 6에서 기술할 다중 서버 구조를 이용할 때에도 서버로부터 전송되는 메시지와 클라이언트로부터 전송되는 메시지를 나누어 관리함으로써 일관성 있는 프로그래밍을 가능하게 하여 준다.

6. 다중 서버 시스템으로의 확장

가. 다중 서버 구조

현재의 분산 가상 환경 시스템들이 추구하고 있는 중요한 목표 중의 하나는 바로 대규모 사용자를 수용할 수 있는 환경을 구축하는 것이다. NPSNET[3-7]에서는 이 대규모 환경을 1000명 이상의 사용자나 동적인 객

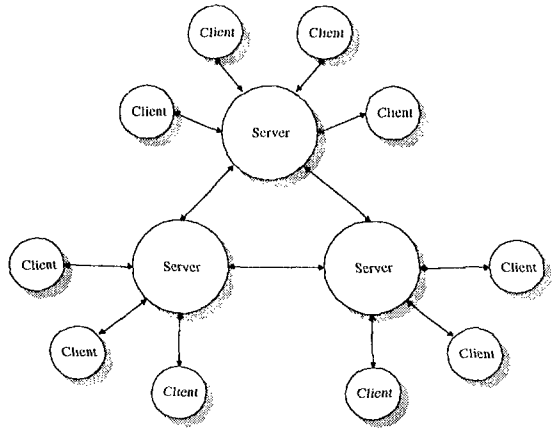


그림 3.8 다중 서버를 통한 클라이언트
서버 구조

체들을 동시에 지원할 수 있는 환경이라고 정의하고 있다. 이러한 대규모 다중 사용자 시스템을 구축하는 것은 정부 차원에서나 상업적인 차원에서나 대단히 중요한 일이다. 그러나 현재의 네트워크의 성능으로 실제로 이러한 규모의 가상 환경을 구축하기에는 어려운 점이 많으며 따라서 이를 극복하기 위한 다양한 방향으로의 연구가 진행 중이다.

본 MIRAGE 툴킷에서는 그림 3.8에 도시된 바와 같이 기존의 클라이언트-서버 구조를 확장한 다중 서버를 통한 클라이언트-서버 구조 기능을 제공하도록 설계하여 참여할 수 있는 사용자의 수를 가능한 많게 하려는 노력을 하였다. 단일 서버의 경우에는 클라이언트의 수가 증가할수록 한 서버에게 주어지는 부담이 급격히 증가하게 된다. 반면 다중 서버를 이용할 경우 각 서버가 담당해야 할 클라이언트의 수가 적어지기 때문에 위에 경우에 비하여 서버에게 주어지는 부담이 감소하게 된다. 그러나 한 클라이언트에서 전송한 메시지가 다른 클라이언트에 도착하기까지 두 개의 서버를 지나야 하기 때문에 메시지 전송의 경로가 하나 더 증가하는 단점도 있다. 다중 서버 구조는 서버의 부담을 줄인다는 장점 외에도 그림 3.9와

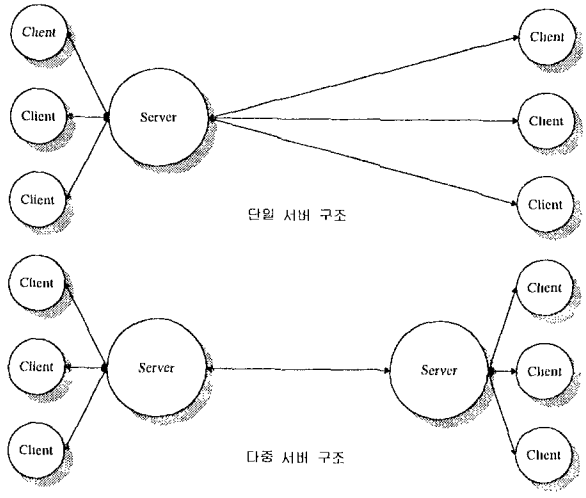


그림 3.9 다중 서버를 통한 원거리
네트워크의 감소

같이 원거리 네트워크 통신의 양을 줄일 수 있다는 장점이 있다. 즉 지역적으로 떨어져 있는 그룹간의 협력작업이 필요한 경우 각 지역마다 지역적인 서버를 두고 그 서버가 해당 지역의 클라이언트들을 관리하게 함으로써 원거리 네트워크는 서버들 간의 연결에만 필요하게 된다. 협력작업이 지역적으로 멀리 떨어진 사용자들의 작업에 적용되어 시간과 비용을 줄이는 데 크게 도움이 될 수 있으므로 이러한 다중 서버 구조가 이용되는 것이 바람직하다.

나. 다중 서버 구조에서의 일관성 유지 모델

다중 서버 구조로 확장할 경우 일관성 유지 방법에 대해서 다시 고려해야 한다. 서버가 담당하는 중요한 역할중의 하나가 일관성 유지의 기능을 담당하여 다중 사용자들이 공유 정보를 동시적으로 이용할 때 발생하는 경쟁 문제를 해결해 주는 것이다. 단일 서버에서는 서버가 하나였기 때문에 독점적으로 일관성 유지의 책임을 지는 것이 가능했다. 그러나 서버

수가 증가하여 다중 서버 구조를 이루게 되면 서버가 유일하지 않기 때문에 기존의 일관성 유지 방법을 사용할 수 없게 된다. 다중 서버 구조에 적합한 일관성 유지 모델로서 다음과 같은 방법들을 생각할 수가 있다.

첫 번째는 다중 서버들 중의 한 서버가 일관성 유지의 기능 독점적으로 담당하는 것이다. 이 방법은 기존의 방법을 그대로 확장한 것으로 개념적으로 간단하고 구현하기 쉽다는 장점이 있다. 그러나 이 방법은 일관성 유지를 위해 선택된 서버의 부담을 증가시키는 면이 있으며 일관성 유지를 담당한 서버의 지역 그룹에 속하지 않는 클라이언트들은 지역 그룹에 속한 클라이언트들보다 항상 잠금 요청의 응답을 받는데 불리한 점이 있다. 일관성 유지 서버가 관리하는 그룹에 속한 클라이언트는 직접 잠금 요청에 대한 결과를 알 수 있으나, 그렇지 못한 클라이언트들은 항상 자신의 지역 서버를 통해야 하기 때문에 전자의 경우보다 메시지 전송의 경로가 2배로 증가하게 된다. 이러한 불평등은 서버와 서버간의 원거리 네트워크의 거리가 길어질수록 더욱 크게 나타나게 될 것이다

두 번째 방법은 그림 3.10과 같이 한 서버가 독점적으로 일관성 유지의 역할을 담당하는 것이 아니고 모든 서버가 일관성 유지에 참여할 수 있는 방식이다. 이 구조에서는 각 서버가 일관성 유지의 책임을 나누어 맡아 한 서버가 독점적으로 일관성 유지를 담당할 때의 부담을 나누고 각 클라이언트가 받던 불평등한 잠금 요청에 대한 응답 지연을 해소 할 수 있다. 이 구조는 다시 일관성 유지의 대상이 되는 공유 객체를 어떤 서버에서 관리할 지가 결정되면 그 공유 객체가 사라질 때까지 계속 그 서버가 일관성 유지를 담당하는 정적인 방법과, 협력 작업 수행 중에 어떤 공유 객체에 대한 일관성 유지 관리 담당하는 서버가 계속해서 변경되는 동적인 방법이 있을 수 있다. 전자의 방법은 공유 객체를 관리하는 서버가 고정되어 있으므로 서버간의 추가적인 일관성 유지가 필요하지 않으나 상황에

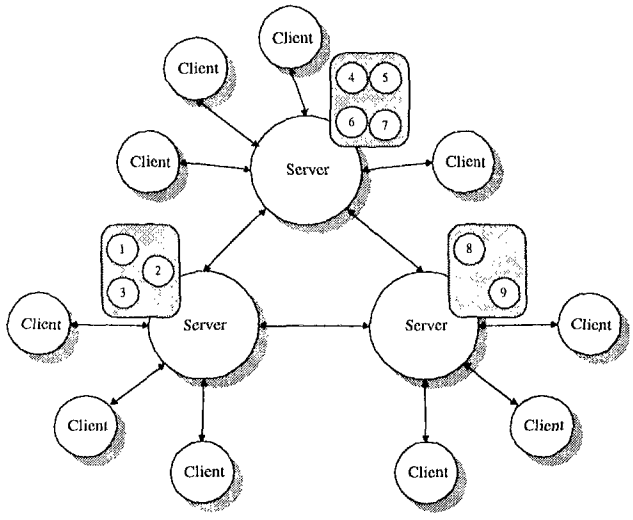


그림 3.10 다중 서버가 일관성 유지에
참여하는 방식

따라서 공유 객체에 대한 일관성 유지가 몇몇 서버에 편중될 수 있다는 단점이 있다. 후자의 방법은 어떤 서버가 어떤 공유 객체에 대한 일관성 유지를 담당하고 있는지 항상 알고 있어야 하기 때문에 추가적인 서버간의 일관성 유지가 필요하다. 후자의 방법은 또한 어떠한 방식으로 공유 객체에 대한 일관성 유지 권리를 서버간에 부류(floating)하게 할 것인지에 따라 다양한 방법이 존재할 수 있다. 예를 들어 공유 객체에 대한 추가와 삭제가 일어날 경우 서버간에 관리하는 공유 객체의 수가 균형을 이루도록 공유 객체에 대한 일관성 유지 권리를 주고받는 방식도 생각할 수 있으며, 가장 최근에 클라이언트로부터 공유 객체에 대한 잠금 요청을 받은 서버가 그 공유 객체에 대한 일관성 유지 권리를 갖도록 하는 방법도 생각할 수 있다.

본 네트워크 툴킷에서 제시하는 다중 서버 일관성 유지 모델은 서버간의

구별 없이 모두 일관성 유지에 참가하는 방법을 기반으로 하여 가장 최근에 공유 객체에 대한 잠금 요청을 받은 서버가 그 공유 객체에 대한 일관성 유지 권리를 갖도록 하는 동적인 방법이다. 실시간 시스템에서는 사용자가 어떤 작업을 한 후 그에 대한 반응이 되돌아오기까지의 응답시간이 매우 중요하다. 이 응답시간이 가능한 빨라야 하는 것도 중요하지만 여러 사용자들간에 공평하게 주어지는 것도 중요하다. 따라서 기본적으로 모든 서버가 일관성 유지에 참여하여 특정 클라이언트들에게 유리한 상황을 만들지 않도록 하는 것이 바람직하다고 할 수가 있다. 가장 최근에 공유 객체에 대한 잠금 요청을 받은 서버가 그 공유 객체에 대한 일관성 유지 권리를 갖도록 하는 방법은 비교적 쉽게 구현할 수 있으며 기존의 메모리 계층구조에서 이용하는 캐쉬의 개념과도 상응하는 면이 있다. 두 그룹의 성격이 다르고 지역적으로 떨어져 있는 상황에서 다중 서버 구조를 통해 협력 작업을 수행하는 중이라고 할 때 각 그룹이 주로 관심을 가지는 공유 객체에 차이가 있을 수 있다. 따라서 이러한 경우 특정 공유 객체는 특정 그룹에서만 잠금 요청이 나올 가능성이 높으며 그 그룹을 담당하고 있는 서버가 그 공유객체에 대한 일관성 유지 권리를 갖고 있는 것이 반응 시간을 줄이는 데 유리하다고 판단된다.

다. 다중 서버의 효용성에 대한 시뮬레이션

앞 절에서 설명하였듯이 다중 서버 구조를 이용하게 되면 서버들의 부담이 감소하고 원거리 네트워크의 수를 줄일 수 있는 장점이 있으나 한 클라이언트가 발송한 메시지가 다른 클라이언트에게 도달하기 위해 지나는 네트워크 경로가 하나 더 증가하는 단점도 존재한다. 따라서, 다중 서버를 이용할 경우 실제로 효율을 높일 수 있는 지에 대해서 살펴볼 필요가 있다. 본 연구에서도 이러한 점을 고려하여 다중 서버 기능을 추가한

표 3.2 실험 환경

	s0	s1	b0	b1 b2 b3	p0	p1 p2
CPU	R4400	R4400	UltraSparc	UltraSparc	R5000	R4400
OS	Irix	Irix	Solaris	Solaris	Irix	Irix
용도	서버	클라이언트	서버	클라이언트	서버	클라이언트

네트워크 툴킷을 렌더링 시스템에 응용하기 전에 다중 서버 구조의 효용성에 대한 시뮬레이션을 해보았다. 시뮬레이션은 서울 소재의 본 연구실의 SGI 워크스테이션 2대 s0과 s1을 하나의 지역 그룹으로 설정하고 미국 버클리(Berkeley)대학의 SUN 워크스테이션 4대 b0, b1, b2, b3, 그리고 포항공대의 SGI 워크스테이션 3대 p0, p1, p2를 각각 지역그룹으로 연결하여, 총 3 지역 그룹이 네트워크로 연결한 상황에서 다중 서버 구조에서 메시지를 주고받는 데 걸리는 시간과 단일 서버 구조에서 메시지를 주고받는 데 걸리는 시간을 비교하였다. 구체적인 실험 환경은 표 3.2와 같다.

실험은 다음과 같이 크게 3 가지 경우로 나누어 진행하였는데, 각각의 경우는 다시 다중 서버를 사용하는 경우와 단일 서버를 사용하는 경우로 나누어 시간을 비교하였다.

실험 1

(1) 다중 서버: s0, b0

서버 s0에 s1에서 클라이언트 2개 연결

서버 b0에 b1, b2에서 각각 클라이언트 1개씩 연결

(2) 단일 서버: s0

서버 s0에 s1에서 클라이언트 2개, b1, b2에서 각각 클라이언트 1개씩 연결

(3) 단일 서버: b0

서버 b0에 s1에서 클라이언트 2개, b1, b2에서 각각 클라이언트

1개씩 연결

실험 2

(1) 다중 서버: s0, b0

서버 s0에 s1에서 클라이언트 3개 연결

서버 b0에 b1, b2, b3에서 각각 클라이언트 1개씩 연결

(2) 단일 서버: s0

서버 s0에 s1에서 클라이언트 3개, b1, b2, b3에서 각각 클라이언트 1개씩 연결

(3) 단일 서버: b0

서버 b0에 s1에서 클라이언트 3개, b1, b2, b3에서 각각 클라이언트 1개씩 연결

실험 3

(1) 다중 서버: s0, p0, b0

서버 s0에 s1에서 클라이언트 2개 연결

서버 p0에 p1, p2에서 각각 클라이언트 1개씩 연결

서버 b0에 b1, b2에서 각각 클라이언트 1개씩 연결

(2) 단일 서버: s0

서버 s0에 s1에서 클라이언트 2개, p1, p2, b1, b2에서 각각 클라이언트 1개씩 연결

(3) 단일 서버: p0

서버 b0에 s1에서 클라이언트 2개, p1, p2, b1, b2에서 각각 클라이언트 1개씩 연결

(4) 단일 서버: b0

서버 b0에 s1에서 클라이언트 2개, p1, p2, b1, b2에서 각각 클라이언트 1개씩 연결

표 3.3 예비실험

	s1	p1	b1
s0	79824	118796	849107
p0	120283	29875	872675
b0	643891	872675	853

위와 같은 구조로 연결이 완성되는 순간 모든 클라이언트들은 1에서128 바이트 사이의 임의의 크기의 메시지를 서버에게 전송하고 서버는 이를 다른 클라이언트들에게 전송한다. 메시지를 받은 클라이언트들은 메시지를 잘 받았다는 ACK메시지를 서버에게 전송하고 서버가 모든 클라이언트들로부터 ACK메시지를 받으면 이를 다시 원래 메시지를 전송한 클라이언트에게 알린다. 이 메시지를 받은 클라이언트는 다시 위의 작업의 반복하여 총 20개의 메시지가 전송되면 작업을 중단한다. 위의 작업은 모든 클라이언트들이 동시적으로 하는 작업이다. 위의 과정 중에서 클라이언트가 하나의 메시지를 전송하고 모두 잘 받았다는 메시지를 받기까지의 시간의 평균을 비교 대상으로 삼았으며 모든 실험은 두 번씩 시행하였다. 표 3.3은 기본적인 네트워크 전송 속도를 알기 위해 행한 예비 실험의 결과이다. 상단의 s1, p1, b1은 클라이언트이며 좌측의 s0, p0, b0이 서버이다. 시간 측정 방법은 앞에서 설명한 것과 같으나 서버와 클라이언트가 1대1의 관계만을 유지한다. 실험 결과의 수치는 마이크로 초 단위이다. 결과를 보면 원거리 네트워크 전송이 지역적인 전송보다 더 오래 걸린다는 것을 확인할 수 있으며 각 지역그룹 내에서도 서로 차이가 있음을 확인할 수 있다.

3가지 경우에 대한 각각의 실험 결과가 그림 3.11, 그림 3.12 및 그림 3.13에 나타나 있는데, 다중 서버를 이용한 경우가 그렇지 않은 경우보다 일관되게 더 빠른 메시지 전송 속도를 갖음을 알 수 있다. 다중 서버의

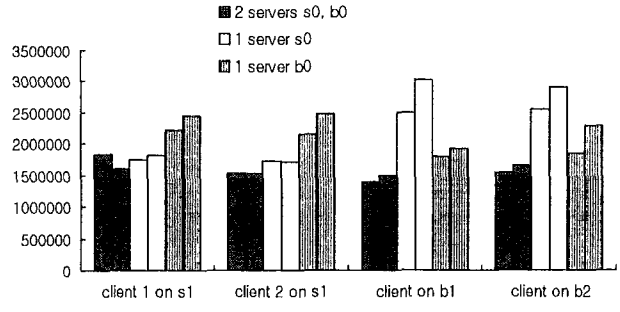


그림 3.11 실험 결과 1

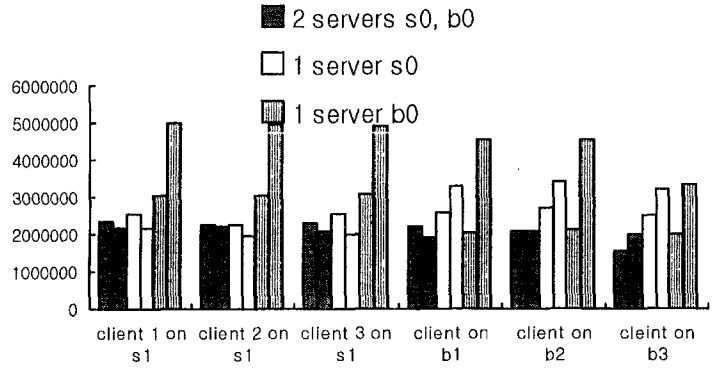


그림 3.12 실험 결과 2

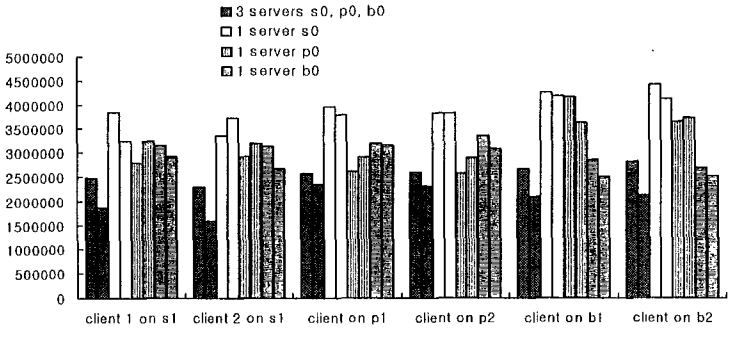


그림 3.13 실험 결과 3

경우 대체적으로 모든 클라이언트들이 비슷한 전송 속도를 갖는다. 이 경우를 자세히 살펴보면 1차 실험과 2차 실험의 결과의 차이가 상대적으로 크지 않다는 것을 알 수 있으며, 버클리 측의 지역그룹에 속한 클라이언트의 전송 속도가 약간 빠르게 나타나고 있는데 이는 예비 실험의 결과에서 보여지듯이 지역 내에서의 네트워크 속도의 차이라고 생각되어진다. 단일 서버를 이용한 경우에는 1차 실험과 2차 실험의 결과 차이가 크게 나타나고 있으며 서버가 있는 측에서 연결한 클라이언트들이 원거리 네트워크를 통해 연결된 클라이언트들 보다 더 빠른 전송 속도를 갖는 것을 알 수 있다. 이와 같은 실험결과를 통해 클라이언트가 많아지고 원거리 네트워크가 이용되는 경우, 다중 서버 구조를 이용한 경우가 그렇지 않는 경우보다 전송 속도 면에서 보다 더 효율적임을 확인할 수 있었다. 뿐만 아니라 모든 클라이언트들에게 상대적으로 비슷한 전송 속도를 제공하고 있는 것도 바람직한 면이라고 생각되어 진다.

7. 다중 사용자 렌더링 시스템의 개발

본 연구에서는 지금가지 설명한 MIRAGE 네트워크 툴킷을 이용하여 본 연구실에서 개발해온 SGRT 그래픽스 렌더링 시스템을 다중 사용자 시스템으로 확장하였다. 본 장에서는 SGRT 렌더링 시스템에 대해 간략하게 살펴보고 다중 사용자 시스템에 사용된 통신 모델과 사용자 그룹이 형성되는 과정 그리고 각 클라이언트들의 데이터 베이스를 관리하는 방식에 대해 간략하게 설명한다.

가. SGRT 시스템 개요

SGRT는 렌더맨 인터페이스[3-11](RenderMan Interface)를 기반으로 하

며, OpenGL[3-12,3-13]의 실시간 렌더링 기능을 이용하여 전경 편집(scene editing) 기능을 제공하고 광선추적법(ray tracing)[3-10]과 Z-버퍼방법에 기반을 둔 Reyes방법[3-14]으로 최종적인 이미지를 생성하는 고급 렌더링 시스템이다. 전경 편집 기능을 이용하면, 다양한 인터페이스 기능을 통하여 고화질의 영상 생성의 대상이 되는 3차원 전경(scene)의 개개의 물체에 대한 성질들(모양, 위치, 색상, 텍스춰, 투명도 등)을 편집할 수 있으며, 카메라를 자유롭게 이동시켜 가면서 원하는 곳의 장면을 렌더링 할 수 있다. 구체적으로 전경 편집기능에는 3차원 물체를 불러들여 적절한 위치에 배치하고 색상과 텍스처, 쉐이더와 같은 속성들을 설정하는 기능이 있으며. 광원을 만들어 원하는 위치에 배치하고 광원의 속성을 지정하며 카메라 옵션과 같은 렌더링 환경을 설정하는 기능이 있다. 위의 작업들은 모두 OpenGL을 이용하여 구현한 실시간 렌더링의 기능을 이용하기 때문에 최종적인 렌더링보다는 못하지만 적절한 화질의 장면을 항상 보여주어 필요로 하는 최종 이미지를 보다 쉽고 빠르게 제작하는데 도움을 준다. 전경 편집 마친 후에 렌더링 기능을 선택하면 지금까지 제작된 전경이 광선추적법이나 Reyes 방법을 이용한 고급 렌더링 모듈로 보내져서 최종적인 장면을 렌더링 하게 된다. 현재 SGRT 시스템은 C/C++, X, Motif, OpenGL 환경에서 수년간 개발되어 왔는데 현재 약 3만 라인의 C/C++ 프로그램으로 구성되어 있으며 계속하여 그 기능을 확장하고 있다. 이 시스템을 분산 환경에서의 다중 사용자의 협력 작업을 지원하는 렌더링 시스템으로 확장할 경우 상당한 양의 전경 편집 작업 (예를 들어 여러 개의 방으로 구성된 대형 건물 안에 있는 물체들에 대하여 렌더링 관련 인자를 설정하려 할 때)을 효과적인 공동 작업으로 수행할 수 있게 되어 작업의 효율성을 제고하게 된다.

나. MIRAGE를 이용한 SGRT의 확장

그림 3.14에는 MIRAGE 툴킷을 이용하여 확장한 다중 사용자를 위한 SGRT의 사용 예를 보여주고 있다. 이 예에서는 3명의 사용자(1명과 나머지 두 명은 안경으로 표시됨)가 텍스트 대화 상자를 통하여 대화하며, 공유 물체에 대한 협력작업을 하고 있다. 확장된 다중 사용자 렌더링 시스템은 클라이언트-서버 모델을 기반으로 하고 있으며, 메시지 전송은 TCP 유니캐스트 채널과 IP 멀티캐스트 채널을 복합적으로 이용한다. 편집과정 중 빈번히 발생하는 중간 정보들 중 신뢰성이 반드시 보장되어야 하는 정보들(예를 들어 한 물체를 선택하여 움직일 때 최종 위치)은 신뢰성이 보장되는 TCP 유니캐스트를 이용하였고, 그렇지 않은 정보들(예를 들어 움직일 때 지나가는 중간 위치)은 신뢰성이 보장되지 않으나 빠른 IP 멀티캐스팅을 이용하여 효율성을 높이도록 하였다.

협력 작업에 참여하는 사용자들의 프로세스들은 각각 클라이언트가 되어 서버와 접속을 이루고 다른 클라이언트들과 메시지를 주고받게 된다. 이때 클라이언트들은 서버와의 관계에 따라 전역 리더 클라이언트(global leader client)와 지역 리더 클라이언트 (local leader client), 그리고 일반적인 클라이언트로 구성되는데 각 클라이언트들의 역할은 다음과 같다.

전역 리더 클라이언트

- 새로 작업그룹에 참가하는 클라이언트에게 고유 번호를 할당
- 새로 작업그룹에 참가하는 지역 리더 클라이언트들에게 참여에 필요한 정보 전송
- 지역 리더 클라이언트의 역할
- 일반 클라이언트의 역할

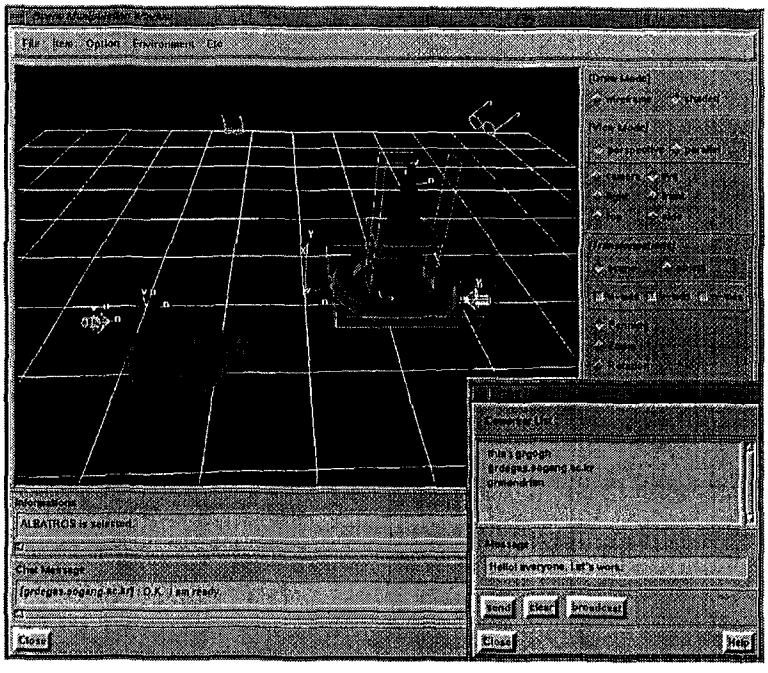


그림 3.14 다중 사용자용 *SGRT* 사용예

지역 리더 클라이언트

- 지역 그룹을 담당할 서버를 기동시킴
- 지역 그룹에 접속하는 일반 클라이언트들에게 참여에 필요한 정보 전송
- 일반 클라이언트의 역할

일반 클라이언트

- 협력 작업의 참여자로서의 역할

각 클라이언트들은 다음의 과정을 통하여 하나의 다중 서버의 관리를 받는 그룹을 형성하게 된다. 가장 먼저 다중 서버 다중 사용자 시스템을 가동시키는 클라이언트는 전역 리더 클라이언트가 된다. 시스템이 시작하면 먼저 단일 서버 시스템에서와 마찬가지로 IP 멀티캐스팅을 위한 소켓

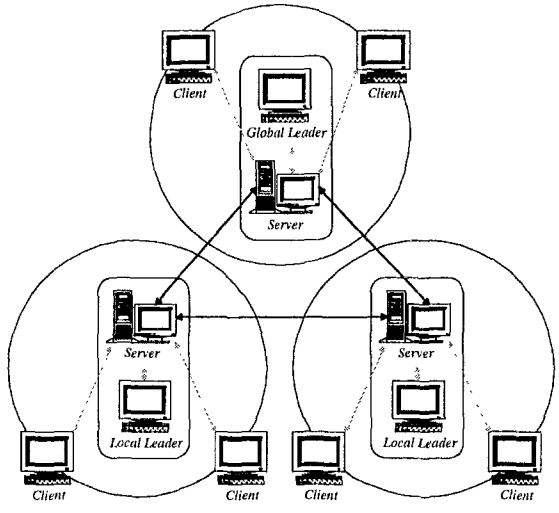


그림 3.15 다중 서버 다중 사용자 통신 모델

을 열고 이 소켓을 통하여 자신이 그룹에 참가하고자 한다는 내용의 메시지를 전송하게 된다. 일정한 시간이 지나도록 아무런 응답 메시지를 받지 못하면 이 시스템은 네트워크 상에 어떠한 가상 환경도 구축되어 있지 않은 것으로 판단하고 자신이 직접 전역 리더 클라이언트가 된다. 전역 리더 클라이언트는 동시에 지역 리더 클라이언트이기도 하여 자신이 속한 지역의 서버를 가동시키고 그 서버와 접속을 이루고 서버에게 자신이 전역 리더임을 알린다. 이후에 참여하는 사용자들은 지역 리더 클라이언트가 될 것인지 일반 클라이언트로써 참여할 것인지를 선택하게 된다. 지역 리더 클라이언트가 된다는 것은 새로운 서버를 가동시켜 지역 그룹을 생성한다는 것을 의미한다. 지역 리더 클라이언트가 되고자 하는 클라이언트는 IP 멀티캐스팅을 이용하여 자신이 지역 리더가 되고 싶다는 요청의 메시지를 전송한다. 전역 리더 클라이언트가 이를 전송 받으면 그룹 참여를 허가한다는 메시지와 그 지역 리더 클라이언트에게 부여되는 고유 핸들을 전송해 준다.

곧이어 그룹 참여에 필요한 정보들을 전송해 주고 기존 서버들의 주소

들을 전송해준다. 위의 메시지들을 전송 받은 지역 리더 클라이언트는 지역 그룹을 담당할 서버를 가동시키고 자신이 지역 리더임을 알리고 서버에게 기존 서버들의 주소를 전송해준다. 기존 서버들의 주소 정보를 전달받은 서버는 이 정보를 이용하여 기존 서버들에게 접속하여 다중 서버 연결 관계를 이루게 된다.

일반 클라이언트의 자격으로 그룹에 참여하고자 하는 클라이언트들은 자신이 속하고 싶은 지역 서버의 주소를 알고 있어야 한다. 이 클라이언트는 IP 멀티캐스팅을 통해 자신이 일반 자격으로 그룹에 참여하고 싶다는 메시지를 전송한다. 전역 리더 클라이언트가 이 메시지를 전송 받으면 그 클라이언트에게 부여되는 고유번호와 그룹 참여를 허락한다는 메시지를 전송해 준다. 이 메시지를 받은 클라이언트는 자신이 알고 있는 서버 주소를 이용하여 서버에 접속하게 되고 해당 서버는 지역 리더 클라이언트에게 요청하여 새로운 클라이언트에게 그룹참여에 필요한 정보를 전송하도록 한다.

위와 같은 과정을 통해 그림 3.15와 같은 통신 모델을 이루게 되는데, 협력 작업에 참여한 사용자들의 클라이언트들이 연결을 한 후에는 본격적인 협력작업이 시작된다. 실제 협력작업은 전경을 구성하는 물체들을 불러오고 물체의 위치와 크기를 변경하며 그 물체가 가지고 있는 속성을 변경하는 과정으로 이루어진다. 사용자의 이러한 작업에 대해서 렌더링 시스템은 각 사용자들에게 보여지는 물체에 대한 정보들이 항상 일치할 것을 보장해 주어야 한다. 본 렌더링 시스템에서는 각 클라이언트 모두 데이터 베이스의 복사본을 가지고 있으면서 물체에 대한 정보가 변경되었을 경우 그에 해당하는 메시지를 주고 받아 각 클라이언트가 데이터 베이스를 갱신하도록 방법을 이용하고 있다. 이렇게 각 클라이언트마다 중복하여 정보를 유지하는 것은 비효율적으로 생각되어질 수 있으나, 실시간 작

업을 위해서는 매우 유리하다. 화면에 보여질 전경을 계산하는 과정을 각각의 클라이언트들이 지역적으로 해결하기 위해서는 전체 전경에 대한 정보를 각 클라이언트들이 가지고 있어야 한다. 또한 변경된 내용만을 전송하면 되므로 메시지의 수를 줄일 수 있었다. 여러 사용자가 동시에 한 물체의 수정 권한을 얻으려 할 때 발생하는 경쟁문제는 네트워크 툴킷의 일관성 유지 기능을 그대로 이용하여 구현하였다. 즉, 새로운 물체를 삽입하려 할 때 COBJECT ALLOC REQUEST를 통해 서버로부터 핸들을 할당 받고, 사용하고자 할 때 COBJECT LOCK REQUEST를 통해 수정 권한을 얻도록 하였다.

8. 결 론

네트워크 성능의 발달과 함께 기존의 단일 사용자를 위한 시스템들을 다중 사용자의 협력 작업을 지원하는 시스템으로 확장하는 연구가 활발히 진행중이다. 그러나 현재의 네트워크 성능만으로 대규모 사용자의 협력 작업을 지원하는 시스템을 개발하는 것은 무리가 있으며 따라서 보다 많은 사용자를 지원하기 위해서 다양한 방식의 연구가 진행되고 있다. 본 연구에서도 이러한 추세에 동참하여 다중 사용자 시스템 제작을 위한 네트워크 툴킷을 개발하였다. 본 네트워크 툴킷에서는 보다 많은 사용자가 함께 참여할 때에도 성능의 큰 감소 없이 작업할 수 있도록 하기 위해 다중 서버 구조의 클라이언트-서버 방식을 택하였으며, 이 구조를 통하여 클라이언트를 관리하는 서버의 부담을 줄일 수 있으며 특히 지역적으로 떨어진 곳에 위치한 사용자들이 협력 작업을 하는 경우에 원거리 네트워크 전송의 수를 줄여 확장성을 높이는 데 큰 도움을 줄 수 있다. 또한 본 네트워크 툴킷에서는 잠금 기법을 통해 여러 사용자들이 동시에 공유 정

보의 내용을 변경하려 할 때 발생할 수 있는 문제를 처리하고 있는데, 이 방식을 기초로 다중 서버 구조에 적용하기에 알맞은 방식을 고안하였으며 이를 툴킷 기능의 일부로 구현하였다. 이렇게 개발된 네트워크 툴킷을 이용하여 기존에 개발된 전경 편집 기능을 갖춘 단일 사용자 렌더링 시스템을 다중 사용자 시스템으로 확장할 수 있었다.

제 4 절 Rendering of Spherical Light Fields for Real Time Rendering (실시간 렌더링을 위한 영상기반 렌더링 기법 연구)

1. Introduction

One of the most important goals of computer graphics is to generate realistic images from complex scenes. Several approaches to image synthesis have been developed including polygonal rendering, ray tracing, and radiosity. Application of these traditional methods involves two complicated tasks: geometric representation of three dimensional (3D) scenes and the physical description of lighting attributes of the scenes.. Considerable work has been undertaken to enhance the realism of generated pictures and to reduce the computational complexities of algorithms. Despite rapid advances in modeling and rendering, creating realistic pictures of virtual environments requires great computational expense and the results are still far from real-time computation.

A different approach, called image-based rendering, has been developed recently. Unlike traditional geometry-based rendering, image-based rendering techniques generate realistic pictures from a set of pre-acquired images. Usually, the source of the pre-computed images is built from digitized photographs or from synthesized

images. Using photographs from the real world makes it possible to skip most of the traditionally laborious modeling and rendering processes. The pre-processed imagery allows the implementation of rendering complex scenes with a constant amount of computation per frame, most likely at real time or interactive rates. This is so even when the source images are synthesized from the virtual world, in which case they may still go through the modeling and rendering processes.

The idea of image-based rendering has long been applied in texture mapping and environment mapping [4-5, 4-4, 4-13]. In Chen [4-6], multiple environment maps are created from cylindrical panoramic images at discrete points, and they are used to compose images seen from locations with continuously changing viewing directions. Image morphing has also been a popular technique for image-based rendering [4-2, 4-20]. In Chen et al. [4-7], the authors made use of the view interpolation approach to synthesize 3D scenes where regularly spaced synthetic images with their computed depth maps are smoothly interpolated as a camera moves. In Debevec et al. [4-9], a hybrid approach combining both geometry-based and image-based methods was presented for modeling and rendering architectural scenes from a set of still photographs.

McMillan et al. [4-17] presented an image-based rendering system, based on sampling, reconstructing, and resampling the flow of light. This, they called a plenoptic function, where a cylindrical projection was used as the plenoptic sample representation. The

plenoptic function was defined as the pencil of rays visible from any point in space, at any time, over any range of wavelengths [4-1]. When time is fixed, the relationship is a function of five variables that describe the flow of light at a point (x, y, z) in a given direction (θ, ϕ). The 5D plenoptic function can be simplified to 4D when views of an object need to be generated from outside its convex hull. This 4D space may be considered a function of the space of oriented lines.

Two similar parameterizations of this 4D function have been presented recently. Levoy and Hanrahan [4-15] parameterized rays by their intersection with two parallel planes. Two pairs (s, t) and (u, v) of parameters on the planes defined an oriented ray, and they restricted the ranges of the parameters so that points on each plane lay within convex quadrilaterals. This parallelepiped of the pencils of rays was called a light slab. The 4D function, called a light field, was created using an appropriate number of such light slabs. In [4-12], Gortler et al. chose a cube to bound an object, and for each face, parameterized the rays in a similar way to create the 4D function, Lumigraph.

Our work extends previous research based on the concept of the plenoptic function. We present a new parameterization and representation of light flow based on spheres, called spherical light field. While methods using spherical coordinates are thought to require substantially more computation than those using planar or cylindrical coordinates, we show that spheres can also be used

efficiently in representing and resampling the flow of light. The rendering algorithms used in previous studies [4-5, 4-12] are "image-space" algorithms in that they have the same basic structure as ray casting. Our rendering algorithm is different in that it is an object-space"' algorithm that can be embedded easily into the traditional polygonal rendering system. It is accelerated easily by 3D graphics boards that support the primitive functionality such as viewing and smooth shading, and allows a simple-to-implement hybrid rendering in which complex regions are rendered from sampled spherical light fields, while less complex regions are rendered from simple polygonal models.

Like the other approaches quoted, our image-based rendering system must handle a huge amount of data. While the successful encoding techniques of previous approaches can be incorporated into our rendering system for compression, we also introduce another possible compression scheme based on wavelets. As well as the proposed technique, that can be easily adapted to compressed the light field and lumigraph data, offers as high compression ratios as the previous methods, it naturally creates a multi-resolutional representation of the light flow, that can be exploited effectively in the future applications. We show how to access the compressed data efficiently using a modified significance map and an incremental decoding technique, and report experimental results on several test data sets.

2. Spherical Light Field Rendering

2.1. Representation of Spherical Light Fields

In this section, we propose a new parametrization of the oriented rays in 3D space. Our approach differs from previous works in that a sphere is used as the convex hull of a bounded object (See Figure 1). Each point on the surface of the sphere is parameterized by two variables (θ_p, ϕ_p). Then, an oriented ray in the space is determined by associating a direction (θ_d, ϕ_d) with each point on the sphere. This results in a different 4D parametrization of the plenoptic function C $\equiv$ (R, G, B) = Ray(θ_p, ϕ_p, θ_d, ϕ_d), which we call a spherical light field.

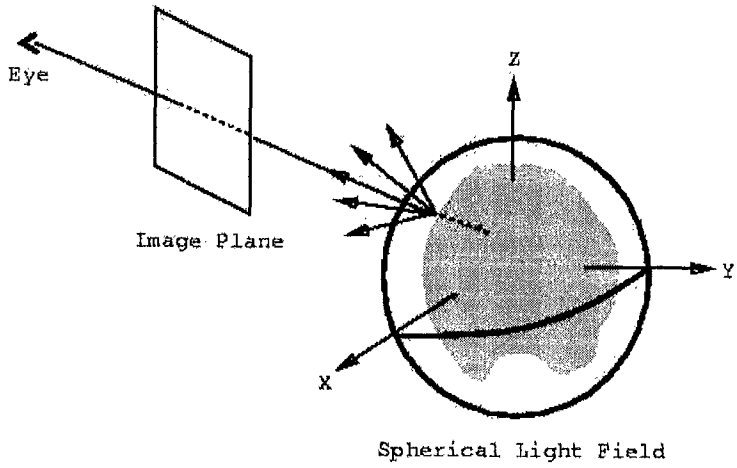
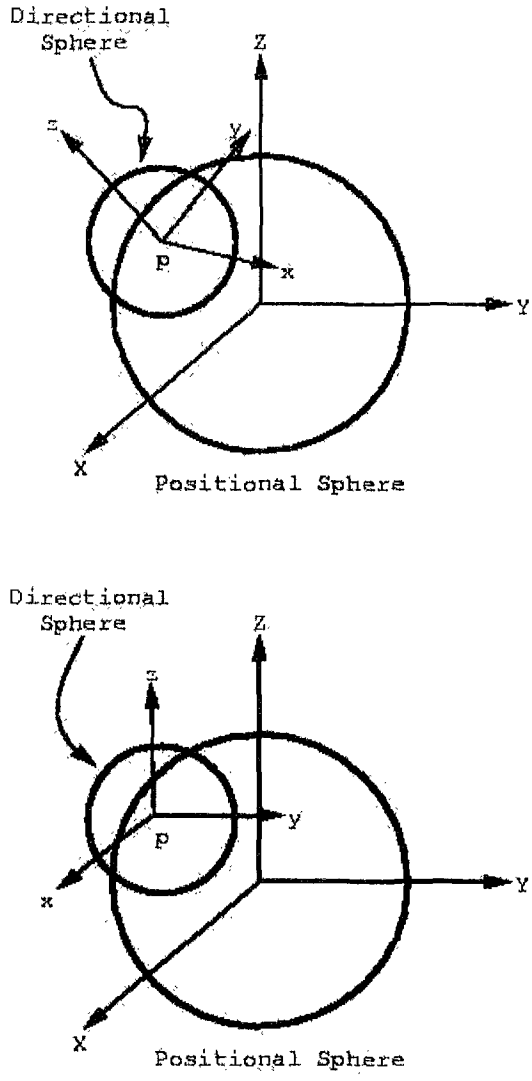


Figure 1. The Representation of a 4D Spherical Light Field

The parametrization can be viewed as a collection of small directional spheres clinging to a large positional sphere (Figure 2). Consider the positional sphere that is a unit sphere, centered at the origin, bounding objects in the scene. Points p = (x, y, z) on the sphere can be parameterized by two variables θ_p and ϕ_p ($0 \leq \theta \leq 2\pi$ and $-\frac{\pi}{2} \leq \phi \leq \frac{\pi}{2}$) that represent the longitude and latitude, respectively. The direction at p is also parameterized by two variables (θ_d, ϕ_d). A natural choice of the coordinate system for directions (θ_d, ϕ_d) is the spherical frame field where the z axis is normal to the surface at p, and the x and y axes are in the directions to the parallel and meridian passing through p, respectively (Figure 2(a)) [4-18]. The coordinate system is natural in the sense that it is appropriate for representing the upper hemisphere that is usually enough to describe the flow of light. In our framework, however, we choose a different frame, whose origin is at p, and the three axes are parallel to those for the positional sphere. This coordinate system has the advantage that a given direction is described by the same parameter values for all the directional spheres. This coordinate system results in efficient computation, especially when a parallel projection is used.

The use of spheres provides a symmetric representation of the complete flow of light, which makes it easy to handle arbitrary viewpoints and directions without exceptions. While spheres are generally less efficient than planes or boxes from a computational

viewpoint, our rendering method is designed to access information stored in a sphere, as efficiently as possible.



(a) (b)

Figure 2. Two Coordinate Systems for 4D Parameterization

The use of spheres provides a symmetric representation of the complete flow of light, which makes it easy to handle arbitrary viewpoints and directions without exceptions. While spheres are generally less efficient than planes or boxes from a computational viewpoint, our rendering method is designed to access information, stored in a sphere, as efficiently as possible.

2.2 Discretization of Spherical Light Fields

The spherical light field has been defined as a function in a continuous, four dimensional space. In practice, to be used in a computational framework, the functional space must be discretized or be sampled. Mathematically speaking, the function Ray can be expressed as a combination of two functions f_p and f_d : Ray $(\theta_p, \phi_p, \theta_d, \phi_d) = f_d(\theta_d, \phi_d) = (f_p(\theta_p, \phi_p))(\theta_d, \phi_d)$, where $f_p: V \rightarrow (V \rightarrow C)$, $f_d: V \rightarrow C$, and $V = \left\{(\theta, \phi) \mid 0 \leq \theta \leq 2\pi, -\frac{\pi}{2} \leq \phi \leq \frac{\pi}{2}\right\}$. That is, f_p is a function defined on a sphere whose value is a function f_d, in turn defined on another sphere. Hence, the task of sampling the 4D spherical light fields may be reduced to the problem of the finite approximations of two spheres.

Functions on a sphere have been frequently used in computational sciences. For example, large collections of data sets in the earth and space sciences and in astrophysics are sampled on the sphere.

Although the sphere is a relatively simple manifold, many techniques that work for the plane do not easily extend to the sphere. When we approximate the sphere by a polyhedron, it is desirable that the vertices are evenly distributed on the sphere, and that the faces have the same shape. In the strictest sense, however, only the five Platonic solids, whose sides are congruent simple equilateral polygons, have their vertices evenly distributed. Hence, when we arbitrarily control the number of vertices and faces that represent the levels of detail of the approximations, in most cases we would get faces of different shapes.

As pointed out in [4-11], the best we can do is a reasonable compromise between simplicity, regularity, and multiple symmetry. The geodesic tessellation of the icosahedron was used in [4-11] for rendering and managing spherical geographic data. This approximation produces a polyhedron with a good deal of regularity. In our framework, however, we start from the octahedron, where each triangular face corresponds to the eight regular patches on the sphere, each corresponding to the octants of the xyz-axes. Figure 3 *illustrates how each triangular face is subdivided recursively into* four finer triangles. Three new vertices are selected in the centers of the circular arcs that connect the vertices of the triangles. In contrast to geodesic construction, in which the great circles of the sphere are used, we use the circles that are orthogonal to the axes of the coordinate system. This choice

produces a triangular approximation which is rather irregular in the sense that differently shaped triangles are generated. Although uneven distribution of the vertices may seem to cause problems, we find that, against our expectations, the irregularity produces better rendering results with fewer aliases. With our tessellation scheme, it is also cheaper than the geodesic tessellaton to locate a triangle that contains a given direction because only simple comparisons with x, y, and z coordinates are necessary.

As mentioned previously, we need to tessellate two different spheres, the positional sphere and the directional sphere. For the positional sphere, we assume that the function f_p is discretized so that it is defined at the vertices of the approximating polyhedron, and the polygonized sphere is stored in a conventional triangular mesh form. The approximating polyhedron for the positional sphere can be any polyhedron made of an arbitrary number of vertices. One the other hand, the function f_a on the directional sphere is assumed to have values at the barycentric centers of the triangular faces, generated from recursive subdivision of the base triangles of an octahedron. Since the value of f_a is an RGB (or RGBA) color, we can imagine a flat-shaded polyhedron where the color of each triangle is the value assigned to the center of the triangle. Figure 5(a) displays an example of flat-shaded directional sphere with level-5 discretization.

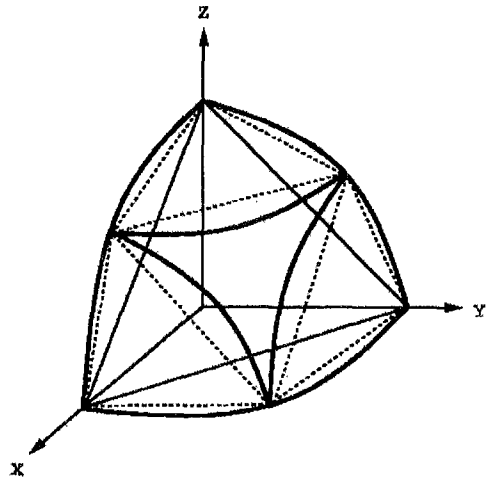


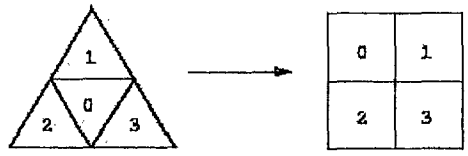
Figure 3. Recursive Subdivision of Base Triangles

The discretized spherical data for the directional spheres are linearized for effective storage and manipulation as illustrated in Figure 4. The process is in fact a reordering of the sampled spherical data into a two dimensional array. When a triangular patch is subdivided into four sub-patches, they are labeled, and are reordered into an array as in Figure 4(a). This labeling continues as the triangular paths are recursively subdivided, and each triangle of the final polyhedron is named by concatenating the labels corresponding to the region. Figure 4(b) shows an example of the level 2 subdivision and its array representation. Notice that a quadtree is implicitly constructed for each base triangle of the octahedron corresponding to one-eighth of the directional sphere. In this representation, we view the sphere as a tree that is described as follows:

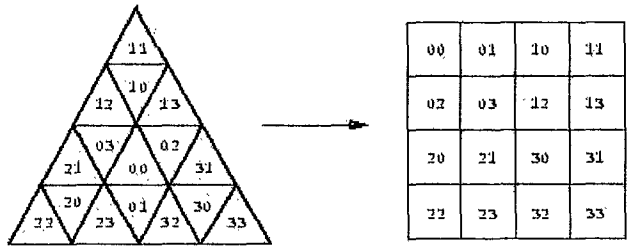
- The root has two children corresponding to the upper and lower hemispheres.

- The two children have four children corresponding to the four base triangles.

- The children are the roots of the quadtrees that represent the base triangles.



(a) Labeling order



(b) Level two subdivision

Figure 4. Linearization of Spherical Data

(a) A Flat-Shaded Directional Sphere (b) Two Linearized Spheres

Figure 5. The Directional Spheres and its Linearization

When the base triangle is subdivided up to level 5, $4^5 = 1024$ triangles are generated, and the data associated with the centers are stored in a $2^5 \times 2^5$ array. Each hemisphere is then put into a $2^6 \times 2^6$array, and a whole discretized directional sphere is represented by a $2^7 \times 2^6 = 128 \times 64$ array. Figure 5(b) shows a 128×128 image that has been reordered from two spheres, where the left half is from the sphere in Figure 5(a). When the base triangle is discretized up to level 4, a directional sphere is represented by a 64×32 array.

2.3. Polygonal Rendering of Spherical Light Fields

So far, we have discussed the representation of discretized spherical light fields that are serialized into two composite spheres. We now describe how an image is displayed from a given spherical light field. Levoy et al. [4-15] and Gortler et al.

[4-12] produced an image using algorithms that have the same main for-loops as a ray tracer. For each ray generated pixel-by-pixel, the corresponding (s, t, u, v) coordinates are computed, then, a resampling process is carried out to compute the color along the ray. These methods may be classified as image-space algorithms. On the other hand, our approach is an "object-space" algorithm that is easily embedded into a polygonal rendering system.

Our viewer generates an image by rendering the triangles of the polyhedron that discretize the positional sphere. The rendering process is a smooth-shading of a polyhedron. Given a camera position and a projection type, the back-faced triangles of the sphere are culled first. For every front-faced triangle, color is associated to each of its vertices. To do this, the projection direction for the vertex, parameterized by (θ_p, ϕ_p), is decided, and it becomes a point (θ_d, ϕ_d) on the directional sphere corresponding to the vertex. The color that is assigned to the vertex is the functional value Ray(θ_p, ϕ_p, θ_d, ϕ_d) of the spherical light field. Then the triangle is projected onto an image plane, and the colors at the vertices are bilinearly interpolated at the pixels it covers.

Most of the simple rendering process can be implemented easily in a conventional polygonal rendering system such as OpenGL. The shading part is programmed in just a few OpenGL commands. First, the smooth shading model is selected by the glShadeModel() function with the mode parameter GL_SMOOTH. Then the triangles are drawn by

the glVertex() functions with the colors assigned to their vertices by the glColor() functions. Back-face culling and viewing are routine in polygon-based rendering. Graphics boards that offers primitive functionality such as viewing, culling, and smooth shading, are available today at moderate prices. Hardware acceleration of spherical light field rendering is hence quite affordable even on low-end workstations or PCs.

The only major part that must be computed in software is that of extracting the color of a vertex (θ_p, ϕ_p) from the directional sphere $f_a = f_p(\theta_p, \phi_p)$. When the projection direction is (θ_d, ϕ_d), the color becomes $f_d(\theta_d, \phi_d)$. Since the directional sphere is now defined on the discretized space, accessing the function involves a resampling process. We find a simple nearest-neighbor filtering technique adequate. Filtering based on linear interpolation from the nearest samples on the directional sphere would result in few aliasing artifacts. However, interpolation-based filtering is expensive, especially when real-time rendering is desirable. As stated above, the directional sphere is subdivided into eight patches, each corresponding to an octant of the xyz-axes. Then each patch is recursively refined into four subpatches. When recursion is repeated up to level 5, we obtain $8 \times 4^5 = 8192$ directions on the discrete sphere which are enough for nearest-neighbor resampling in most cases. When the maximum level is 4, there are $8 \times 4^4 = 2048$ directions. In the case of this fewer number of directions, using nearest-neighbor filtering may cause some local

distortion in a displayed image with perspective projections, because each projector is modified to its nearest one among the 2048 directions.

Such a problem does not occur, however, in parallel projections. We chose the coordinate systems for the directional spheres in such a way that the axes have the same orientation. Once the direction of projection is determined, all the points on the positional sphere take on the same (nearest) direction for the directional sphere. Hence there is no local distortion, though the filtering may produce a little jerky movement as a sequence of images is displayed.

It has often been stated that techniques using spherical coordinates require substantially more computation than those using planar coordinates. We have designed our image-based rendering system so as to minimize the computation necessary for extracting data from spherical light fields. As noted above, two spheres must be accessed. Since the front-faced triangles on the positional sphere are simply projected and shaded, access of the sphere is just trivial. On the other hand, the color data corresponding to the nearest direction must be extracted from the directional sphere, given a projection direction. As the spherical data is created by recursive subdivision, we traverse the quadtree down to the leaf. Figure 3 illustrates how the triangular patches are subdivided. Instead of geodesic circles, we use circles that are orthogonal to the principal axes. This choice makes it cheap to

query which child to take in traversal because only simple comparisons with the x, y, and z coordinates are necessary. As described below, we also attempted to minimize the computation necessary for accessing the compressed data.

3. Wavelet-Based Compression

Like the light field and lumigraph, a huge amount of storage is necessary to represent a nontrivial spherical light field. For example, we used approximately 64K points for the positional sphere when an octahedron is used as a base polyhedron. When each of the eight quadtrees have level 5 for the directional sphere and 24 bits are used for a color, $64 \times 2^{10} \times 8 \times 4^5 \times 3 = 1.5$ GB of storage is required. If we use quadtrees with a maximum level of 4, the storage required is 384MB.

To make implementation of spherical light fields practical, the data must be compressed. In [4-15], the authors describe a compression system consisting of fixed-rate vector quantization followed by entropy coding. To quantize a light field, they first partition the source data into a set of sample vectors, and construct a codebook of reproduction vectors called codewords that best approximate the samples. Then the sample vectors in the source are replaced by indices of the closest approximating codewords from the codebook. The codebook and indices are further compressed by gzip which is an implementation of Lempel-Ziv coding. They report

an overall compression ratio of a factor of more than 100 : 1, while up to 24 : 1 compression is possible using the vector compression that decides the actual amount of run time memory. In [4-12], the authors guess that a 200 : 1 compression ratio could be achievable with almost no degradation.

In this section, we propose another compression technique, designed for the spherical light field, which can be easily adapted to the light field and lumigraph. Notice that these data contain substantial coherence, hence a good compression technique must remove redundancy well. Furthermore, the compression technique must provide a low-cost random access to compressed data. Recall how the rendering process tries to access the pixels of linearized directional spheres. For each directional sphere image, only one (or a few when interpolation is done) sample is extracted, and the access pattern is rather random. Most compression techniques, which place some constraints on random access, are not appropriate for displaying the spherical light field because they often fail to decode an individual sample quickly.

Our compression method, based upon wavelets, produces as high compression ratios as the previous one used for the light field [4-15]. Our method is designed to be able to extract colors of individual samples, accessed in an arbitrary order, from the compressed data efficiently, using a modified significance map and an incremental decoding technique. It also provides a multi-resolution representation of the spherical light field that

can be utilized in the future applications.

Wavelets are a mathematical tool for representing functions hierarchically, and they have had a great impact in several areas of computer graphics [4-8, 4-22]. They also provide powerful tools for multi-resolution representation of data and can be used for data reduction. Wavelets have been applied successfully to image compression, say [4-10, 4-21, 4-23] An image is compressed by applying a two dimensional wavelet transform to compute coefficients representing the image, and then disregarding the coefficients smaller in magnitude than a specified criterion.

Schröder et al. discussed how to construct wavelets for functions defined on the sphere. Since the colors on the directional spheres are functions defined on the sphere too, their work is highly appropriate in this case. However, we choose to work on the linearized image of the directional spheres. We aim at building a wavelet-based compression scheme which is appropriate not only for the spherical light field but also for the light field and lumigraph, based on the cubical representation. Furthermore, notice that each directional sphere is rather coarsely sampled, for example, 32×64 and 64×128 points when the discretization levels are 4 and 5, respectively. It would not be a good idea to compress data on each sphere handling them separately. Since the directional spheres corresponding to the neighboring points on the positional sphere contain much coherence, it would produce a better result to put several spherical data in one unit, and process them together,

in which case, linearization gives a simple computational scheme.

In our implementation, we use the nonstandard Haar wavelet decomposition as in [4-3]. The Haar wavelet transformation is not one of the best wavelet filters, but it is computationally efficient, which is one of the most important factors in this application. Our experience tells that the Haar wavelets are good enough for compressing the spherical light field. Suppose an image is represented conceptually by a quadtree whose root corresponds to the whole image and whose descendants represent recursively subdivided regions. Decomposition is carried out by applying the Haar transform from the leaves all the way up to the root. The color values of the leaves can be thought to be the average colors of the corresponding regions represented by pixels. Then four average colors c_1, c_2, c_3, and c_4 in four children are recursively decomposed as one average color c_{LL} and three details c_{HL}, c_{LH}, and c_{HH} using the wavelet transform:

$$c_{LL} = \frac{c_1 + c_2 + c_3 + c_4}{4}, \quad c_{HL} = \frac{c_1 - c_2 + c_3 - c_4}{4},$$

$$c_{LH} = \frac{c_1 + c_2 - c_3 - c_4}{4}, \quad c_{HH} = \frac{c_1 - c_2 - c_3 + c_4}{4}$$

The four values arise from separable application of vertical and horizontal filters, here c_{LL} represents the average of pixel colors in the subregions, and c_{HL}, c_{LH}, and c_{HH} contain detail information on how the color varies in the four subregions. During decomposition, the four colors are replaced by the average color and three details (See Figure 6(a).), and then the average colors

are repeatedly decomposed. The decomposition process converts the image into a wavelet image that can be stored in an array of the same size using a proper ordering of the coefficients.

As noted above, level 4 and level 5 discretization of directional spheres generates 64×32 and 128×64 images, respectively. In our compression scheme, we use as a unit image the image of size $2^7 \times 2^7 = 128 \times 128$ that amounts to 8 or 2 linearized directional spheres, discretized up to level 4 or 5, respectively. While the decomposition process produces a new wavelet image that has a reduced number of non-zero coefficients, it does not necessarily imply that the new image takes up less storage space. Consider an RGB image whose pixels are represented in three bytes. Each channel in the unsigned character type has a value ranging from 0 to 255. Since the repeated divisions during decomposition generate fractional numbers in each channel, the average colors and details cannot usually be stored in three bytes. Using floats (4 bytes), however, requires 12 bytes for color that may lead to an inefficient compression scheme. In fact, it is possible to exactly encode an image in integer arithmetic using slightly more bits per channel. For example, it is not hard to see that only 10 bits for each channel is sufficient when the so-called reversible S-transformation is employed to a 2D image [4-16].

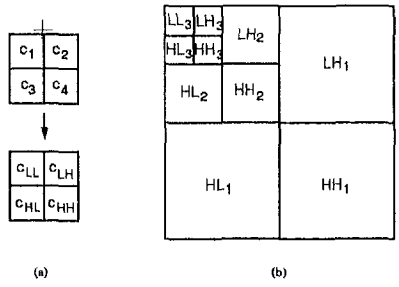


Figure 6. Wavelet Decomposition

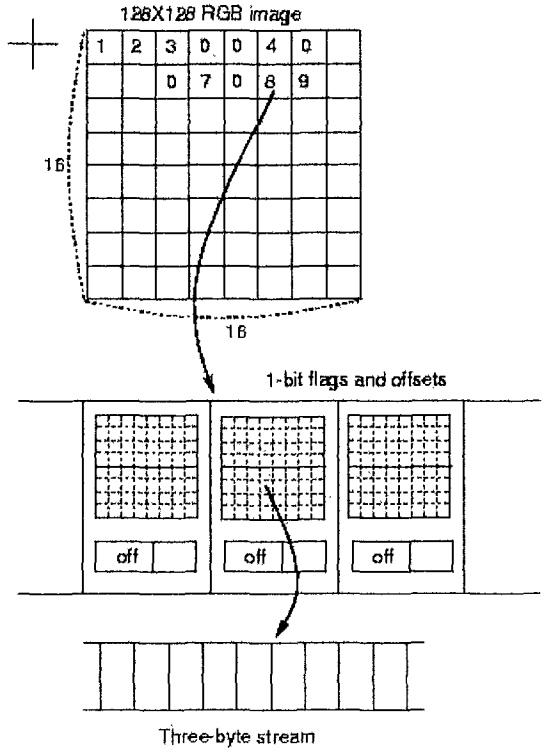


Figure 7. Wavelets Encoding Scheme

In our method, we start from a 128×128 RGB image, in which three bytes are used for each pixel, and apply the wavelet transform three times to the image. Figure 6(b) shows a diagram of the resulting wavelet decomposed image in which the subscripts represent the levels of decomposition. The details in the blocks LH_1, HL_1, and HH_1 represent the finest scale wavelet coefficients and the first-level averages in LL_1 are again decomposed, and so on. Notice that the 16×16 block LL_3 contains the average colors, each corresponding to an 8×8 subregion of an image. During decomposition, we use enough precision, say four bytes per channel, to calculate the average and detail coefficients without round-off errors. Once the decomposition process is done, the coefficients are stored using one byte per channel, that is, three bytes per pixel. Particularly, the 16×16 averages are stored as unsigned integer, and the remaining details are stored in signed integer. Although the mimimum precision required is 10 bits per channel when the S-transformation is employed, in which the floor operations must be carried out repeatedly, we choose the simpler Haar transformation for its computational simplicity. The information loss occurs when the coefficients are rounded off into three bytes. However, we find that the round-off errors in this stage do not cause much trouble in compressing the spherical light field. Now we have a 128×128 (partially) wavelet decomposed image whose coefficients are represented in three bytes. In the next stage, we delete (replace by zero) the vector-valued coefficients whose

magnitude, measured in the L^2 norm, is smaller than a given threshold τ. Then only the non-zero coefficients are enumerated in a three-byte stream in the order described below.

As emphasized before, it is critical to be able to quickly reconstruct a color of an arbitrary pixel from a compressed image. Hence, there must be some mechanism that supports a quick random access. In our compression technique, the 128×128 image is partitioned into 16×16 subblocks, where each subblock represents 8×8 subimages (Figure 7). We first scan the subblocks in the left-to-right, up-to-down fashion, tagging with zero the subblocks whose coefficients are all zero, and with positive integers in the increasing order the subblocks that contains at least one non-zero coefficient. Since each tag can be represented using one byte, 256 bytes of storage is enough (Note that even when all the subblocks contains non-zero coefficients, 256 bytes are sufficient using some trick.). Each subblock with a non-zero tag is then scanned in the left-to-right, up-to-down fashion again, enumerating the non-zero coefficients in the three-byte stream. In addition, an auxiliary chunk of memory is allocated for this subblock that contains an 8×8 1-bit flag block and offset information. The one-bit flag block, requiring 8 bytes, contain the significance map, or the binary information as to whether the coefficients in the subblock are zero or not. The offset information off, represented in two bytes, contains the positions, in the three-byte stream, of the first non-zero coefficients in the ordering. A problem here is how

to retrieve, efficiently, the value of a coefficient with index (i, j) in a decomposed 128×128 image whose non-zero coefficients are listed in the three-byte stream. To do this, we first check the tag of the subblock that contains the (i, j) element. If it is zero the coefficient is simply null. Otherwise we look at the auxiliary memory corresponding to the subblock that contains the 1-bit flags of the coefficients. Let (i', j') be the index of the coefficient (i, j) in the subblock. If the flag for the index (i', j') is 0, then the coefficient is zero. If not, we must carry out some computation to get the correct position or address of the (i, j) coefficient in the three-byte stream.

The position of the coefficient having the index (i', j') in the proper stream can be computed by adding its displacement value to off. When the flag is 1, the displacement is the number of coefficients that precede it in the enumeration whose flags are 1. To count the number efficiently, we use a precomputed indexing table T(*) with $2^{16} = 65536$ entries. Given a word made of two bytes, the table returns the number of bit 1 in the word. Hence, the correct number can be counted by accessing the table only a few times (Note that a proper number of zeros must be padded, in the word, from the position (i', j') position in the last access.).

We shall now analyze briefly, the costs that must be paid to access a coefficient in the compressed image. When the tag for the subblock that contains the coefficient is zero or its 1-bit flag is 0, that is, when the coefficient is zero, the cost is trivial. When

its flag is 1, that is, when the coefficient is non-zero, a few table accesses, 2.5 on the average, and a few additions are necessary. The typical ratio of non-zero coefficients after wavelet compression we use in our implementation is less than 10 percent. This implies that accessing a coefficient in the compressed image involves little cost.

Once the non-zero coefficients have been enumerated in the three-byte stream, the coefficients are quantized. We view the data in the stream as a 24-bit true color image, and simply apply the vector quantizer that converts a 24-bit image into a 8-bit indices and a color table whose entries are three bytes long. In order to minimize the overheads caused by the color table, we build a color table for several, say four or eight, 128×128 images. Since the images corresponding to the neighboring vertices on the positional sphere are very similar to each other, sharing one table does not harm the quality of vector quantization.

Extracting a color for a projection direction from the directional sphere is equivalent to reconstructing a color from a wavelet-encoded compressed image. The reconstruction process is the reverse of decomposition in which four average colors c_1, c_2, c_3, and c_4 are computed from one average color c_{LL} and three details c_{HL}, c_{LH}, and c_{HH} using the formulae :

$$c_1 = c_{LL} + c_{HL} + c_{LH} + c_{HH} ,$$

$$c_2 = c_{LL} - c_{HL} + c_{LH} - c_{HH} ,$$

$$c_3 = c_{LL} + c_{HL} - c_{LH} - c_{HH} , \text{ and}$$

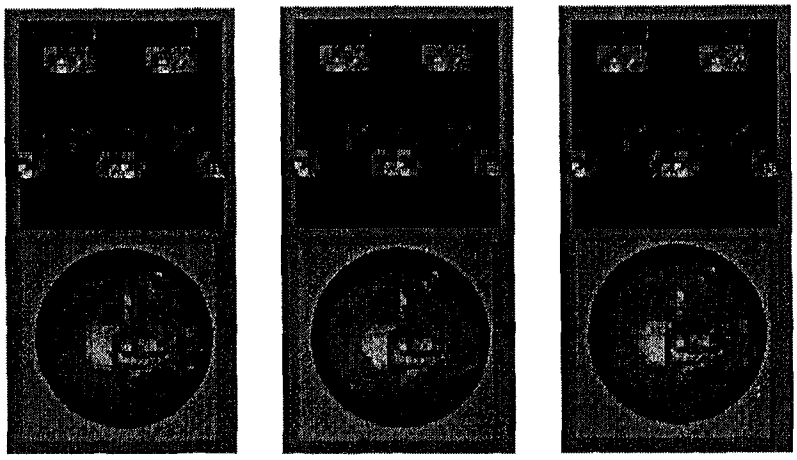
$$c_A = c_{LL} - c_{HL} - c_{LH} + c_{HH}$$

To extract the color of a specific pixel, that is, a specific direction, from the wavelet-encoded image, it is necessary to traverse the (conceptual) quadtree from the root down to the corresponding leaf, applying the above formula repeatedly. Since we have applied the wavelet transform three times, the reconstruction formulae are applied three times. To make the reconstruction computation as efficient as possible, it is carried out incrementally. Notice that as a mouse is moved to rotate objects in a scene, the projection directions of a point in the positional sphere change gradually. Hence, when we access a pixel in the image for the directional sphere, the chances are that it is near the pixel that has just been accessed. This implies that two adjacent reconstruction processes usually share a large portion of the paths from the root to the leaves in the quadtree. This observation suggests that the previous computation results may be reused and so, rather than discard the sets of four average colors for nodes in the path, we store them in a stack. When the next pixel is reconstructed, we compare the previous with the current paths to find the common path. Then the average colors of the common ancestors are reused without computing them again. This incremental computation enhances the performance of the reconstruction process.

Before turning to the next section, we report a quantitative analysis on the compression ratios. Firstly, a given 128×128 image takes up $2^7 \cdot 2^7 \cdot 3$ bytes. To store the tags for a compressed image,

256 bytes of memory is necessary (See Figure 7 again.). For a subblock that contains at least one non-zero coefficient, an auxiliary memory is allocated where 8 bytes (8×8 bits) are used for the 1-bit flags, and 2 bytes are used for the offset information. Let α be the ratio of the non-zero coefficients used after wavelet compression, that is, $\frac{\#\ of\ coefficients\ in\ the\ three-byte\ stream}{\#\ of\ the\ whole\ coefficients\ used}$, and β be the rate of the subblocks with non-zero tags that contain at least one non-zero coefficient, that is, $\frac{\#\ of\ non-null\ subblocks}{\#\ of\ the\ whole\ subblocks}$.

Then, the compressed image consumes $2^7 \cdot 2^7 \cdot \alpha + 10 \cdot 256 \cdot \beta + 256$, bytes and dividing it by $2^7 \cdot 2^7 \cdot 3$ bytes gives the compression rate $\rho = \frac{\alpha}{3} + \frac{5}{96}\beta + \frac{1}{192}$. This figure does not include the overheads for the color table: When the color table is shared by four and eight images, the additional cost are $\frac{256 \cdot 3}{2^7 \cdot 2^7 \cdot 3} = \frac{1}{64}$, and $\frac{1}{128}$, respectively.



(a) $\alpha = 10\%$ (b) $\alpha = 7\%$ (c) $\alpha = 5\%$

α	β	compr. ratio ρ
0.10	$\frac{100}{256}$	$\frac{3010}{49152}$ (6.1%)
0.07	$\frac{100}{256}$	$\frac{2532}{49152}$ (5.1%)
0.05	$\frac{86}{256}$	$\frac{2068}{49152}$ (4.2%)

Figure 8. Examples of Wavelet Compression

Notice that the second and the third terms in ρ are the costs that must be paid to store the necessary significance maps of wavelet coefficients. In our scheme, this information allows a low-cost random access. It could be further compressed using the Zerotree or Horizon embedded coding techniques, but that only placed constraints on random access to the compressed data. Figure

8 illustrates three examples of compression from the directional sphere image shown in Figure 3.

The compression ratios we expect to achieve with this encoding scheme for the directional sphere data range from 10:1 to 30:1, depending on the coherence in data. To design a compression technique using wavelets, we have compromised between good compression ratio and fast random access ability. Our compression technique could be useful for other applications that can benefit from the multi-resolution representations.

4. Experimental Results

The complete image-based rendering system has been implemented on an SGI Indigo 2 workstation with a 200MHz R4400 CPU, 256 Mbytes of main memory and a High IMPACT graphics board. The performance results for three spherical light fields with the different parameters are summarized in Table 1. The test spherical light fields were created using our volume ray casting software from the "UNC head"' data set which is a $256 \times 256 \times 225$ CT scan of a human head. We created four spherical light fields for two classifications and two discretization levels of the directional spheres. The raw spherical light fields we have generated take about 384 Mbytes to 1.5 Gbytes of storage. The data size is proportional to the number of vertices in the positional sphere, and to the number of triangles in the directional spheres.

Positional Sphere			Dis. level for dir. Sphere	Size of raw data (MBytes)	Ratio of Non-zero wavelet coefficients used	Size of compressed data (MBytes)	Compression Ratio
base	# of vertices	# of triangles					
8	65,538	131,072	4	384.0	15%	35.2	10.9 : 1
					10%	28.6	13.5 : 1
20	40,962	81,920	4	233.5	15%	22.0	10.6 : 1
					10%	17.8	13.1 : 1
16	32,770	65,536	5	768.1	10%	49.9	15.4 : 1
					5%	54.3	22.4 : 1

Table 1. Result of Data Compression

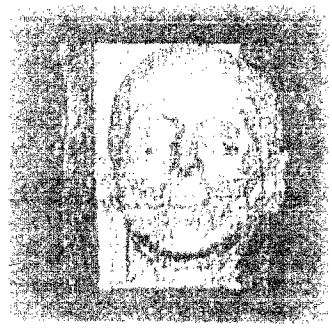
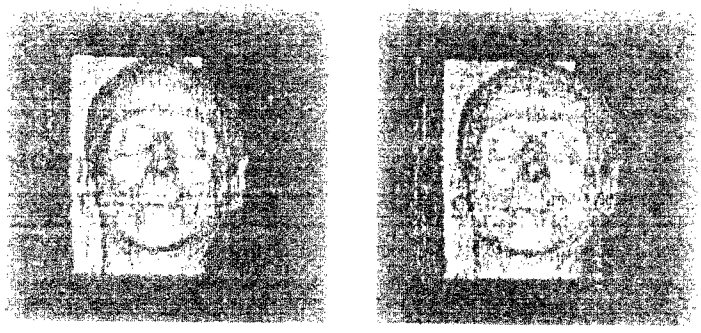
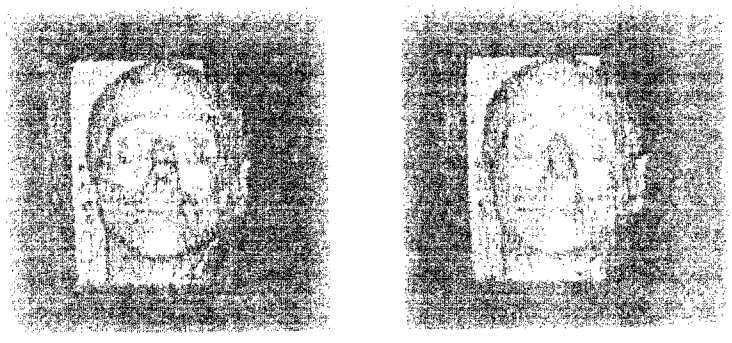


Figure 9. Ray Casting of the Human Head with Skin



(a) 15%, 35Mbytes (b) 10%, 29Mbytes

Figure 10. Level 4 Renderings (an Octahedral Base)



(a) 15%, 22Mbytes (b) 10%, 18Mbytes

Figure 11. Level 4 Renderings (an Icosahedral Base)

Figure 9 shows a 256×256 image generated with a parallel projection of the CT data classified with an opaque skull and semitransparent skin. It was rendered directly from the CT data by our volume rendering software for comparison purposes. We used a ray casting algorithm which is optimized using octrees and early ray termination [4-14]. In the implementation, four bytes (one for density, two for normal directions, and one for normal's magnitude) were allocated per voxel for fast classification and lighting computation, hence it takes about 57 Mbytes of memory for the resolution $256 \times 256 \times 225$. Each rendering usually took longer than a minute.

The four images in the following two figures were extracted from two spherical light fields with the level 4 resolution for the directional sphere. Two spherical light fields of level 4

discretization were constructed. For the first one, the positional sphere was created by recursively subdividing an octahedral base up to level 7 where the total size of the data amounts to 384.0 Mbytes. For Figure 10(a) and (b), the spherical light fields were compressed by deleting 85 and 90 percents of wavelet-encoded coefficients in the decomposed wavelet images, where the resulting compressed data take 35 and 29 Mbytes of storage, respectively. These data could be further compressed using the entropy coder such as gzip, in which case the data sizes become 27 and 21 Mbytes. (These figures are not important when we are interested in the run-time memory requirement.) The positional sphere of the second level 4 data was generated by level 6 subdivision of an icosahedral base, where the total size amounts to 233.5 Mbytes. The images in Figure 10(a) and (b) were generated from the spherical light fields that were compressed by deleting 85 and 90 percents of wavelet-encoded coefficients in the decomposed wavelet images, where the resulting compressed data take 22 and 18 Mbytes of storage, respectively.

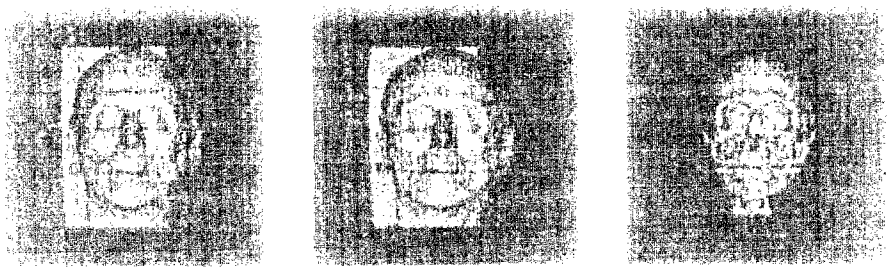
When 15% of coefficients are used, the rendering produced images whose quality is competitive with that of ray casting. We observe that the rendered images still maintain a good quality when 10% of the coefficients are used.

To measure the running time for the level 4 spherical light fields, the head was rotated 360 times at one degree angle increments. For the first data, it took about 0.44 second per frame

on the average. To render the second data which has fewer vertices on the positional sphere, it took about 0.29 second per frame on the average. Notice that the rendering time is proportional to the number of vertices on the positional spheres.

The image-based rendering task is broken down into two major computations: 0.07 to 0.11 second is consumed per frame for polygonal rendering (viewing and shading) of the tessellated positional sphere. This part is accelerated by the graphics hardware, and the running time is almost independent of output image size. The remaining portion of rendering time is spent mostly in accessing 65538 and 40962 (for each data, respectively) compressed directional spheres to associate colors with the vertices of the positional sphere. It is run in software, and hence relies on the speed of CPU. Our preliminary implementation on a PC with a 200MHz Pentium Pro CPU, 128 Mbytes of main memory and an Intergraph Intense 3D graphics board (without texture memory), produces a little faster timing performance. The timings per frame are found somewhat irregular, which is due to the way the wavelet-compressed linearized directional spheres are accessed. To get a color from a directional sphere, its conceptual quadtree is traversed downward corresponding to a given projection direction, applying the reconstruction formulae to each node on the path. The effect of the incremental computation stands out when a large portion of the paths are shared by two adjacent accesses. When the object is rotated gradually, the pixels corresponding to two

adjacent accesses are very close to each other in the linearized images. In probability, two close pixels have most ancestors in common, but that is rather irregular depending on their locations. (Two adjacent pixels can share only the root in quadtrees.) Notice that the first rendering takes more computations because the whole stacks of the directional spheres must be constructed. Then, the decreased timings for the following renderings show how the incremental traversal technique works.



(a) 10% of Coeff (b) 5% of Coeff (c) Perspective

Figure 12. Level 5 Renderings - a Base that has 16 polygons

In Figure 12, three images from the spherical light field with the level 5 discretization for the directional sphere, are shown. Two data sets for (a) and (b) were compressed so that 10 and 5 percent, respectively, of coefficients in the wavelet encodings were used. The resulting compressed spherical light fields took 50 Mbytes and 34 Mbytes, respectively. The image in (c) was produced with a perspective projection. For the level 5 data, the object was also rotated 360 times by one degree, and it took about 0.26 second per frame on the average. It took roughly twice as long for

perspective because the projection directions had to be computed for each vertex. As mentioned earlier, when a parallel projection is used, the level 5 does not necessarily produce output images of higher quality. The image quality is more affected by how many coefficients are deleted in compression. The higher compression ratios are required for the level 5 data due to their huge size, and they degrade image quality more as a result. The projection directions on the directional spheres are more densely sampled when the level is 5. This results in a smoother transition in an animation movie, however, we observe that the level 4 is good enough in most cases.

(a) 256×256 (b) 384×384 (c) 512×512

Figure 13. The Human Head with Skin at Three Resolutions

Figure 13 shows images rendered from the same level 4 spherical light field at three different image resolutions. The running time

is nearly independent of the image resolutions. The computational bottleneck is accessing the compressed directional spheres, which is dependent only on the size of spherical light fields. The other viewing and shading computations are affected by output image size, but remains almost unchanged as accelerated by hardware. As the resolutions get larger, the number of pixels a projected triangle covers in the image plane, increases. Hence the bilinear interpolation between the vertex colors in the final stage of polygonal rendering makes images somewhat smooth.

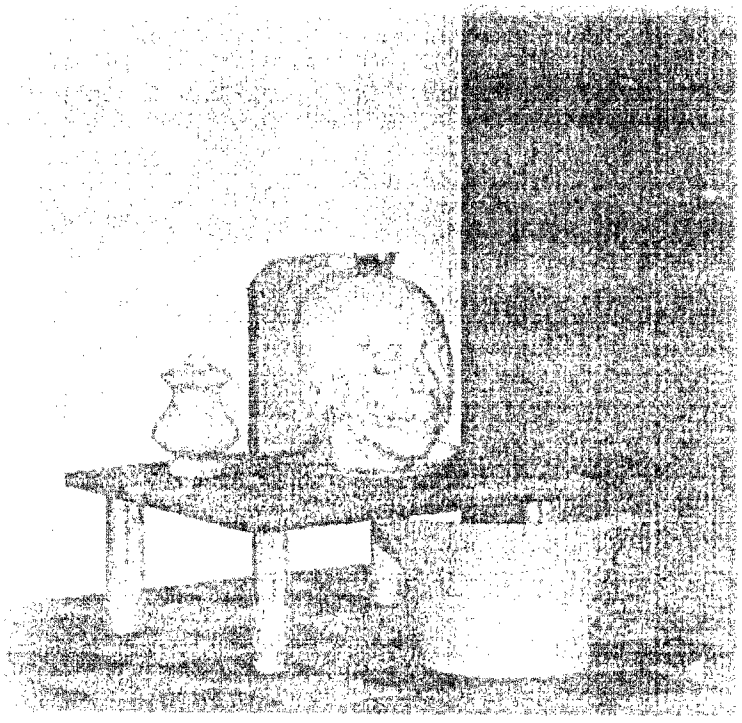


Figure 14. Image-Based Rendering Intermingled with Polygonal Rendering

The last figure (Figure 14) shows how naturally our object-space image-based rendering algorithm intermingles with traditional polygonal rendering. (The jagged stuff behind the head is a material in the CT data, classified with bone and skin.) The head was rendered from a spherical light field, and the rest was rendered from polygonal models. In the context of polygonal rendering, the tessellated positional sphere can be regarded as a special graphics primitive that is made up of triangles and color information at the vertices. Adding this new graphics primitive will extend the conventional polygonal rendering into a simple-to-implement hybrid rendering in which complex regions are rendered from spherical light fields, while less complex regions are rendered from simple polygonal models.

5. Conclusions

In this paper, we have described an image-based rendering framework, based on a new parameterization of the flow of light in space. We showed that the spherical light field is an effective representation that allows efficient computations, and is easily added in the graphics pipeline based on polygonal rendering. In addition, we introduced an encoding scheme based on wavelets for compression of the huge data resulting from sampling of the spherical light field.

The rendering computation is split into two stages: the compressed

directional spheres are accessed to get the colors for the vertices of the tessellated positional sphere, then the triangles on the positional sphere are projected and smoothly shaded in the image plane. The second part is easily accelerated by the graphics hardwares, used in a conventional polygonal rendering system. This object-space approach implies a feasible direction on how the traditional polygonal rendering pipeline can be extended to include the image-based rendering technique.

As shown in the examples, the time and space complexities of our method depend largely on the performance of the first part, which is in close connection with the compression scheme. We have presented an encoding technique based on wavelets. As well as it producing as good a compression ratio as the relavent methods, the wavelets offer a systematic encoding of the flow of light in multi-resolution form. It was shown that the incremental decoding makes it efficient to reconstruct colors from the wavelet-compressed image.

We are currently investigating the relationships between the number of triangles and vertices in the tessellated positional spheres, and the image quality. The data size is proportional to the number of vertices, hence reducing the number without degrading image quality will enhance the timing and space performances. Since, in our implementation of wavelet-compression, one unit contains two or eight directional spheres with the same (θ_p, ϕ_p) coordinates, we could say our encoding is $(2+\alpha)$-dimensional. We are

now compressing the 4D spherical light fields using 3D and 4D wavelets hoping to get higher compression ratios without harming decoding speeds.

제 5 절 방대한 볼륨 데이터의 효과적인 렌더링 기법 연구

1. 볼륨 가시화

볼륨 가시화(volume visualization)는 의학, 유체역학, 기상학 등과 같은 분야에서 발생되는 3차원 이상의 추상적인 수치 데이터를 컴퓨터 그래픽스(computer graphics) 이론을 이용하여 가시화함으로써 데이터에 내재되어 있는 유용한 정보에 대한 효과적인 분석을 수행하는 과학적 가시화(scientific visualization) 기법이다[5-10]. 하지만 가시화를 필요로 하는 대부분의 볼륨 데이터들은 상당히 방대한 크기를 갖고 있기 때문에 이들에 대한 가시화는 상당히 많은 비용과 시간을 필요로 한다. 따라서 수백 *MByte*에서 수십 *GByte*에 이르는 이러한 방대한 데이터를 효과적으로 처리하기 위한 특별한 기법이 고안되어야만 한다[5-13, 5-18, 5-19].

최근에 미국의 NLM(National Library of Madicine)에서는 인체 내부에 대한 심도 있는 연구를 위해 남여 각각의 시신에 대해 CT(Computed Tomography), MRI(Magnetic Resonanace Imaging), RGB 이미지 데이터를 생성하는 Visible Human Project를 수행하였다[5-15, 5-16]. Visible Man으로 이름붙여진 남자 데이터는 머리에서 발까지를 1*mm* 간격으로, 그리고 Visible Woman이라는 여자 데이터는 1/3*mm* 간격으로 하여 생성되었다. 이때, 각 단면 데이터는 512x512의 픽셀(pixel) 해상도(resolution)를 갖고, 실제 값은 12*bits*로 저장되었다. 결국, Visible Man은 512x512x1878의 해상도를 갖게 되고, 전체 데이터의 크기는 약 15*Gbyte*가 되며, Visible Woman의 경우는 약 40*Gbyte*의 크기를 갖는다.

기존의 대부분의 가시화 방법들은 전체 볼륨 데이터를 일단 메인 메모

리(main memory)에 로드(load)한 상태에서 가시화를 수행한다. 따라서, 매우 큰 메인 메모리를 갖는 고성능의 병렬 컴퓨터(parallel computer)를 사용하는 경우가 아니라면 방대한 크기의 볼륨 데이터를 이러한 단순한 기법으로는 가시화할 수 없다. 따라서 본 연구에서는 이와같이 수백 *MByte*에서 수십 *GByte*에 이르는 매우 방대한 크기의 볼륨 데이터를 일반적인 크기의 메인 메모리를 갖는 (예를들어, 64*MByte* 또는 128*MByte*) 일반적인 컴퓨터에서 가시화하기 위한 새로운 가시화 기법을 제안한다. 제안된 기법은 볼륨 데이터를 일반적인 컴퓨터의 메인 메모리에 로드할 수 있는 크기로 압축하는 전처리(preprocessing) 과정을 수행한다. 사용자는 볼륨 데이터를 가시화하기 위해 압축된 데이터를 메인 메모리로 로드하게 되고, 이후의 가시화 과정에서는 전체 데이터를 메인 메모리에 로드한 후 수행되는 기존의 가시화 기법을 사용할 수 있다. 이와같은 가시화 기법이 구현되기 위해서는 압축된 데이터로 부터 임의의 위치 (i, j, k)의 복셀값(voxel value)만을 빠르게 복원(reconstruction)할 수 있는 랜덤 액세스(random access)가 가능해야 한다. 이것은 메인 메모리에 로드된 압축 데이터로부터 원하는 위치의 복셀값을 액세스할 때, 추가적인 메모리가 불필요하다는 것을 의미한다. 랜덤 액세스와 함께 고려해야 하는 또 다른 중요한 요소는, 압축 데이터로 부터 복셀값을 복원할 때 소비되는 시간이다. 볼륨 가시화 자체가 많은 비용과 시간을 요구하지만 가시화 기법의 효율성을 판단하는 중요한 기준 중의 하나가 수행 시간이기 때문에 복원에 소비되는 시간은 가능하면 최소화 되어야 한다. 실제로 볼륨 데이터를 탐사(navigation)하는 시스템의 경우, 복셀값에 대한 액세스 패턴(pattern)은 매우 복잡하고 전체 데이터중에서 일부의 데이터만이 가시화에 이용된다. 그리고 인터렉티브한(interactive) 수행을 위해서는 가시화 기법 자체의 실시간 수행(real time processing)뿐만 아니라, 데이터

에 대한 실시간 액세스도 가능해야 한다.

그러나 대부분의 기존의 압축 기법들은 랜덤 액세스의 가능 여부나 디코딩 속도 보다는 압축율(compression ratio)과 복원 화질(image quality)에 주로 초점이 맞추어져 있다. 예를들어, 허프만(Huffman) 코딩, JPEG, 런-렝스 인코딩(run-length encoding)등과 같은 일반적인 압축 기법들은 임의의 위치 (i, j)에 대한 픽셀값(pixel value)을 얻기 위해 전체 데이터를 복원해야만 하기 때문에 랜덤 액세스가 불가능하다 [5-5, 5-20]. 그리고 대부분 1,2차원의 데이터를 압축의 대상으로 하기 때문에 복원을 위해 소비되는 시간에 대한 고려는 그리 중요한 문제로 인식되지 않았다. 물론 이러한 기존의 방법들이 매우 높은 압축율과 좋은 복원 화질을 갖는다는 사실은 이미 알려져 있으나, 랜덤 액세스와 복원 속도가 중요한 요소가 되는 본 연구와 같은 응용에서는 적당하지 않은 기법들이다.

Ning[5-14]은 벡터 양자화(vector quantization)에 기반을 둔 볼륨 데이터의 압축 기법을 제안하였다. 이 방법에서 벡터는 부분 블럭(subblock)에 대한 밀도값과 미리 계산된 노말 벡터(normal vector)로 구성된다. 이것들은 양자화되어 코드북(codebook)에 저장되고, 각 부분 블럭들은 이에 대한 적절한 인덱스로 이뤄지게 된다. 이렇게 양자화된 자료 구조로 부터 인덱스를 이용해 액세스된 벡터를 이용해 렌더링(rendering)을 수행하고 이를 순 서에 맞게 합성(composition)함으로써 최종 이미지를 생성하였다. 이 방법에서 복셀의 디코딩은 코드북의 단순한 액세스로 가능하기 때문에 빠른 랜덤 액세스를 지원한다. 그러나, 128x128x128의 볼륨 데이터를 압축한 실험 결과에 의하면 압축율은 5:1정도이고, 복원된 이미지에서는 블럭화 현상(blocky effect)와 컨투어링 현상(contouring effect)이 나타는 문제점을 나타낸다. Ghavamnia[5-4]는 2

차원 이미지에 대한 라플라시안 피라미드(Laplacian pyramid) 기법을 볼륨 데이터로 확장하여 적용했다. 그는 가우시안 로우-패스 필터(Gaussian low-pass filter)를 이용하여 라플라시안 피라미드라고 불리는 간단한 계층적 자료 구조를 만들고, 이것을 양자화를 통해 인코딩했다. 복셀값은 이 자료 구조를 밑에서 위로 방문하면서 복원되며 효과적인 복원을 위해 캐쉬(cache)의 이용을 제안했다. 그러나 이 방법으로 생성된 이미지는 그리 좋은 복원 화질을 보여주지 못하고 있다. Thomas[5-24]는 Visible Human 데이터의 2차원 이미지에 대해 여러가지 손실 압축(lossy compression)기법들을 이용하여 압축을 수행한 실험 결과를 보여주고 있는데, 이 결과에 의하면 웨이블릿(wavelets)을 이용한 경우가 가장 좋은 결과를 나타낸다. 이 실험에서는 전형적인 압축 단계인 웨이블릿 변환(transform), 벡터 양자화, 무손실 압축(lossless compression)의 3단계 순서를 따라 압축을 수행했다. 기본적인 연구 목적이 효과적인 데이터의 압축과 전송(transmission)에 있고, 2차원 이미지들 사이에서 존재하는 중복(redundancy)를 고려하지 않은 2차원 압축이라는 점에서 볼륨 데이터의 압축과는 거리가 있으나 웨이블릿 압축 기법 자체의 우수성은 이 결과를 통해 확인할 수 있다. Muraki[5-11,5-12]는 볼륨 데이터에 대해 3차원적인 웨이블릿 압축을 처음으로 적용하였다. 그러나 그의 연구 결과에는 압축된 데이터를 가시화할 때 사용되는 실제 복원 데이터의 크기에 대한 언급이 없고, 랜덤 액세스나 복원 속도에 대한 고려도 없다. 또한 복원된 데이터에 의해 가시화된 이미지의 화질도 그리 좋지 않음을 보여준다.

본 연구에서는 매우 방대한 크기의 볼륨 데이터를 효과적으로 가시화할 수 있는 압축 스킴(compression scheme)을 제안한다[5-8,5-9,5-17,5-26,5-27]. 이 기법은 제한된 메인 메모리를 갖는 일반

적인 범용의 워크스테이션이나 PC를 이용하여 매우 방대한 크기의 볼륨 데이터를 가시화할 수 있도록 만들어 주는데, 이때 사용자는 전체 데이터가 메인 메모리에 로드된 것처럼 느끼게 된다. 대부분의 기존 압축 방법들은 높은 압축율과 우수한 복원 화질을 나타내는 대신 빠른 랜덤 액세스의 지원이 불가능하다는 한계를 갖는다. 그러나, 본 연구에서는 구현된 기법은 어느 정도의 복원 화질을 유지하면서 높은 압축율과 빠른 랜덤 액세스가 가능하다는 특징을 갖는다. 또한, 웨이블릿을 이용한 3차원 압축 기법을 기반으로 하므로, 웨이블릿에 내재된 좋은 성질인 다해상도(multi-resolution) 표현이 가능하다는 장점을 갖고 있다.

2. 웨이블릿과 데이터 압축

웨이블릿이란 함수(function)을 계층적으로(hierarchically) 표현하는 수학적인 도구이다. 이것은 함수를 다해상도 형태로 분해하는데, 이러한 성질은 컴퓨터 그래픽스에서 기하 모델링(geometric modeling), 전역 조명 모델(global illumination model), 애니메이션(animation), 그리고 이미지나 볼륨 데이터의 압축 등과 같은 분야에서 다양하게 응용되고 있다[5-1, 5-2, 5-3, 5-6, 5-21, 5-23].

웨이블릿 변환에는 여러가지 기저(basis)가 사용될 수 있는데, 가장 간단한 것이 Haar 기저이다. 이것은 다른 기저들에 비해 간단하고 계산량이 적다는 장점을 갖고 있다. 즉, 변환과 복원 과정이 덧셈, 뺄셈 그리고 쉬프트 연산(shift operation)만으로 단순하게 수행될 수 있기 때문에 속도가 빠르다. 따라서 본 연구와 같이 빠른 랜덤 액세스를 필요로 하는 응용이나 많은 데이터를 실시간으로 복원해야 하는 응용에서 상당히 적합한 기저라고 할 수 있다. 그러나, Daubechies와 같이 일반적으로 많이 사용

되는 기저들에 비해서 복원 화질이 상대적으로 좋지 않은 단점을 갖는다.

웨이블릿 변환을 통해 주어진 데이터를 분해(decomposition)하고 평균값을 갖는 부분에 대해 재귀적으로(recursively) 변환을 수행함으로써 전체 데이터를 해당 기저와 계수(coefficient)들의 선형 결합(linear combination)으로 표현할 수 있다. 그리고 변환 과정에서 생성된 전체 계수들 중 절대값의 크기(magnitude)가 어떤 한계값(threshold)보다 작은 계수들을 잘라냄으로써(0으로 치환함으로써), 데이터를 압축할 수 있다 [5-23].

3. 압축 스킴

가. 3D 웨이블릿 변환

앞절에서 설명한 웨이블릿을 이용한 압축 기법을 3차원으로 확장하여 볼륨 데이터의 압축에 적용하는 실험이 Muraki에 의해서 수행되었다 [5-11,5-12]. 그는 1차원에서 정의된 기저 함수들의 모든 가능한 텐서 프로덕트(tensor product)를 이용하여 3차원상에서 정의되는 정규직교 기저를 생성하고, 변환을 통해 볼륨 데이터에 대한 웨이블릿 계수를 계산하였다. 그 이후 과정은 전형적인 압축 방법을 그대로 적용하였는데, 실제로 압축되는 계수들을 어떻게 인코딩했는지, 압축된 데이터를 이용해 가시화를 수행하는 순간에 얼마나 많은 실제 메모리가 필요한지에 대한 언급이 없다. 또한, 복원된 데이터를 이용해 가시화를 수행한 결과 이미지도 그리 좋은 화질을 보여주지 못하는 수준이고, 압축 데이터로 부터의 랜덤 액세스나 복원 속도 등은 전혀 고려의 대상이 아니었다.

본 연구에서는 전체 볼륨 데이터를 16x16x16의 크기를 갖는 단위블럭(unit block)이라 불리는 부분 블럭으로 분할하고, 각 단위블럭에 대해 3

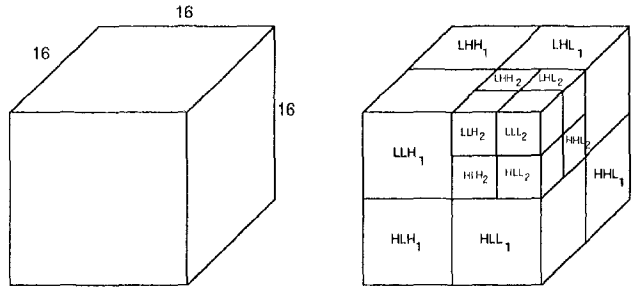


그림 5.1 3차원 웨이블릿 분해

차원 웨이블릿 변환을 2회 반복적으로 수행하는 방법을 사용하였다. Haar 기저를 사용하는 경우의 변환 수식은 다음과 같이 표현되는데, 이미 언급했듯이 덧셈, 뺄셈, 그리고 쉬프트 연산만으로 간단히 계수들을 얻을 수 있다.

$$
\begin{aligned}
c_{LLL} &= (c_1 + c_2 + c_3 + c_4 + c_5 + c_6 + c_7 + c_8) / 8 \\
c_{LLH} &= (c_1 + c_2 + c_3 + c_4 - c_5 - c_6 - c_7 - c_8) / 8 \\
c_{LHL} &= (c_1 + c_2 - c_3 - c_4 + c_5 + c_6 - c_7 - c_8) / 8 \\
c_{LHH} &= (c_1 + c_2 - c_3 - c_4 - c_5 - c_6 + c_7 + c_8) / 8 \\
\\
c_{HLL} &= (c_1 - c_2 + c_3 - c_4 + c_5 - c_6 + c_7 - c_8) / 8 \\
c_{HLH} &= (c_1 - c_2 + c_3 - c_4 - c_5 + c_6 - c_7 + c_8) / 8 \\
c_{HHL} &= (c_1 - c_2 - c_3 + c_4 + c_5 - c_6 - c_7 + c_8) / 8 \\
c_{HHH} &= (c_1 - c_2 - c_3 + c_4 - c_5 + c_6 + c_7 - c_8) / 8
\end{aligned}
$$

여기서, 각 식의 우변에 있는 c_i, $1 \le i \le 8$ 들은 단위블럭 내의 2x2x2 부분 영역에 해당하는 8개의 계수들이고, 좌변의 c_{LLL}은 그 부분 영역의 평균값을 의미하고, 나머지 7개의 계수들은 필터링 시퀀스(filtering sequence)에 해당하는 디테일(detail)값을 나타낸다(예를들어, c_{LHL}는 x, y, 그리고 z축에 대해 로우-패스 필터(low-pass filter), 하이-패스(high-pass filter), 그리고 로우-패스 필터를 반복 적용한 것을 의미한다. 또한 3개의 축 모두에 대해 로우-패스 필터를 적용한 경우가 평균값

c_{LLL}이 된다.). 분해가 많아지면 복원을 위해 소비되는 시간도 길어지게 되므로, 복원 속도를 고려해서 단위블럭에 대해 반복적으로 2회의 분해를 수행하였다. 이러한 분해를 수행한 후에 생성된 계수의 갯수는 변환을 위해 사용된 계수와 동일한 갯수를 갖게 된다 (위의 식에서 볼 수 있듯이 8개의 c_i로 부터 8개의 계수가 발생한다.). 따라서 그림 5.1과 같이 주어진 단위블럭에 대해 분해가 수행된 계수들는 단위블럭과 동일한 크기로 저장될 수 있다.

나. 인코딩

본 절에서는 앞에서 설명한 웨이브릿 변환 과정을 거쳐서 생성된 계수들을 어떻게 효과적으로 저장할 것인지에 대해 설명한다. 본 연구에서는 높은 압축율과 빠른 랜덤 액세스를 동시에 만족시키는 인코딩 스킴을 제안한다. 전형적인 웨이블릿 압축 방법은 변환 과정, 양자화 과정, 그리고 무손실 압축을 수행하는 인코딩 과정의 세가지 단계로 이루어진다. 첫번째로, 변환 과정은 입력 데이터를 웨이블릿 필터를 이용하여 다양한 주파수 밴드(frequency band)로 분해하는 과정인데, 이를 통해 웨이블릿 변환 계수들을 얻게 된다. 두번째로, 변환 과정을 거친 계수들은 양자화 과정을 거치게 된다. 이것은 다양한 값을 갖는 계수들을 적당히 한정된 갯수로 제한하는 과정으로 대부분의 손실 압축이 여기서 발생하게 된다. 양자기(quantizer)에는 벡터 양자기와 스칼라(scalar) 양자기가 있는데, 벡터 양자기가 더 우수한 성능을 나타내는 것으로 알려져 있다. 마지막 인코딩 과정은 양자기로부터 나온 심볼들(symbols)을 비트 스트림(bit stream)으로 손실없이 대치하는 것이다. 여기서는 허프만 코딩이나 산술코딩(arithmetic coding)과 같은 가변 길이 코더(variable length coder)들이 주로 사용되는데, 이와같은 인코딩 방법들은 높은 압축율을

얻기 위해 데이터를 비트(bit) 단위로 저장한다[5-5,5-20]. 그러나 이러한 저장 형태는 디코딩 시간을 느리게 할 뿐만 아니라 랜덤 액세스를 어렵게 하는 원인이 된다.

Shapiro[5-22]는 웨이블릿 계수들이 갖는 특별한 성질을 이용하여 상당히 좋은 성능을 갖는 제로트리 인코딩(zerotree encoding)이라 불리는 기법을 구현하였다. 이것은 서로 다른 웨이블릿 밴드들 사이의 자기 유사성(self similarity)를 이용한 것이다. 즉, 어떤 레벨에서의 웨이블릿 계수가 0이었다면, 더 세밀한 레벨(finer level)에서 동일한 위치에 해당하는 계수도 0이 될 확률이 높다는 특성을 이용한 것이다. 이것은 웨이블릿 인코더의 성능을 상당히 향상시켰지만 디코딩 속도가 상당히 느리다는 단점을 갖고 있다. 따라서 본 연구와 같이 빠른 랜덤 액세스를 필요로 하는 응용에서는 적당하지 않은 인코딩 기법이다.

인코딩 스킴을 구현하기 위해서는 압축율, 복원 속도, 복원 화질 사이의 상호 관계(tradeoff)를 고려하여야 한다. 본 연구에서 구현된 인코딩 스킴은 높은 압축율과 빠른 랜덤 액세스라는 두 가지의 목표를 위해서 구현되었다. 다시 말해, 압축된 전체 데이터를 일반적인 크기의 메인 메모리를 가진 컴퓨터로 로드하고, 추가적인 메모리의 사용없이 원하는 인덱스 (i, j, k)의 복셀값만을 빠르게 복원하여 그 부분에 대한 가시화를 효과적으로 수행할 수 있는 인코딩 스킴을 구현하고자 한다. 이러한 목표를 만족하는 압축 스킴을 구현하기 위해서는 웨이블릿 변환 후에 살아남은 중요한 계수들(significant coefficients)의 값 자체 뿐만 아니라 그 값들의 위치도 함께 인코딩 해야만 한다.

그림 5.2는 살아남은 웨이블릿 계수들과 위치 정보들이 어떻게 인코딩 될 수 있는지를 보여준다[1]. 웨이블릿으로 분해된 3차원 볼륨 데이터의

1. 이후의 설명은 Visible Human 데이터를 기본 모델로 설정하고 있다. 그러나 다른 해상도를 갖는 볼륨 데이터에 대해서도 설명된 기법은 큰 수정없이 적용될 수 있다.

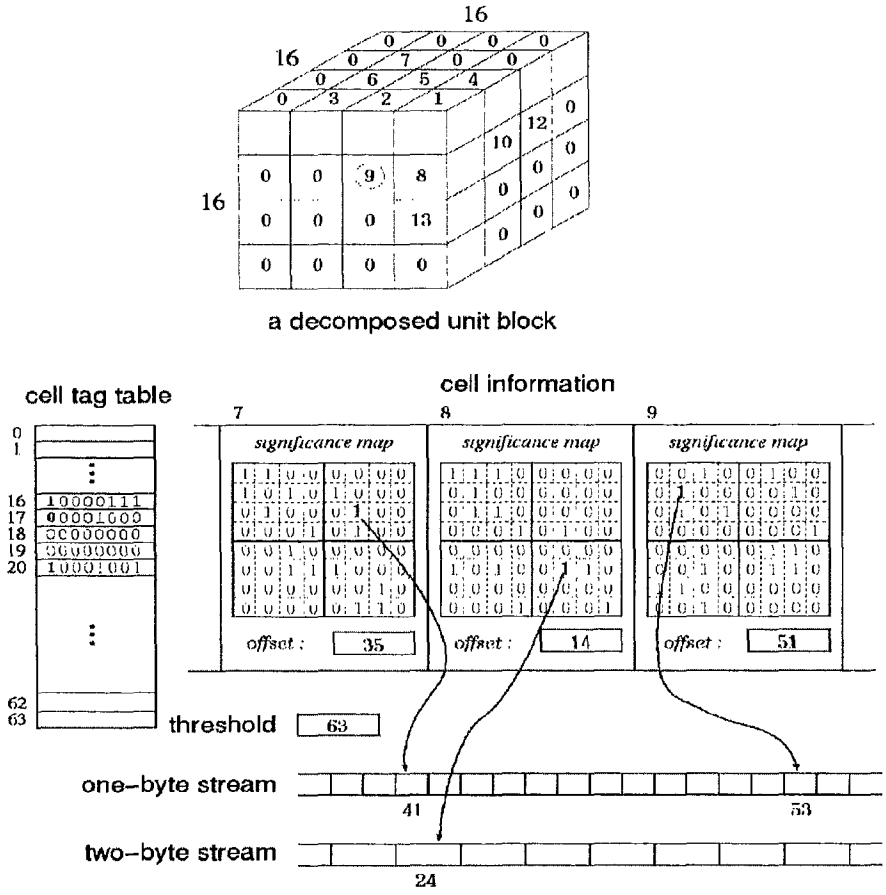


그림 5.2 3차원 웨이블릿 인코딩 스킴

16×16×16 단위블럭을 살펴보면 이미 0으로 치환된 한계값 τ 이하인 계수들이 전체에서 많은 부분(일반적으로, 90% 이상)을 차지하고, 또한 데이터의 인접성(coherence)으로 인해 0인 계수들이 모여있을 가능성이 높다는 사실을 발견할 수가 있다. 이러한 관찰에 근거해, 16×16×16 단위블럭을 웨이블릿을 이용하여 분해한 후, 다시 4×4×4 크기의 셀(cell)이라 불리는 단위의 부분 블럭(subblock)으로 분할한다. 하나의 단위블럭에 대해 $4^3(=64)$개의 셀들이 만들어지는데, 이들에 대해 셀 내부의 계수들

이 모두 0인지 아닌지를 확인하는 과정을 거친다. 만일 셀에 포함된 웨이블릿 계수가 모두 0인 경우라면 해당 셀의 태그(tag)를 0으로 만들고, 0이 아닌 웨이블릿 계수가 하나라도 존재하는 경우라면 1부터 시작해서 하나씩 증가된 태그를 각 셀에 할당한다. 이 태그는 셀 태그 테이블(cell tag table)에 저장되고, 64개의 셀에 대한 각 태그는 1*byte*로 표현될 수 있으므로 셀 태그 테이블의 유지를 위해 64*bytes*가 필요하게 된다.

2번에 걸친 웨이블릿 분해 과정에서는 오차(round-off error)를 줄이기 위해 복셀당 4*bytes*를 사용하여 평균과 디테일(detail) 계수를 계산한다. 원래 주어진 Visible Human CT 데이터가 각 복셀당 12*bits* ($0 \sim 2^{12}-1=4095$ 사이의 값)를 사용하기 때문에 웨이블릿 변환이 수행되는 경우, 평균에 해당되는 계수들은 $[0, 2^{12}-1]$ 범위의 값을 갖게 되고 디테일 계수는 $[-\frac{2^{12}-1}{2}, \frac{2^{12}-1}{2}]$의 범위로 표현된다. 따라서 모든 웨이블릿 계수는 2*bytes*에 저장될 수 있다. 그러나 효과적인 메모리 사용을 위해, 포함하고 있는 모든 웨이블릿 계수가 0이 아닌 셀을(태그가 0이 아닌 셀을) 두 가지 타입(type)으로 구분함으로써 압축율을 높일 수 있다. 첫번째 타입은 셀 내의 모든 계수가 $[-(\tau+128), -\tau]$ 또는 $[\tau, \tau+127]$의 범위에 있는 경우이고, 두번째 타입은 위의 범위에 속하지 않는 계수를 하나 이상 포함하는 경우이다. 첫번째 타입의 경우, 셀 내의 모든 웨이블릿 계수들은 소수 부분을 반올림 함으로써 1*byte*(signed char)로 저장될 수 있고, 이때 발생하는 최대 오차는 0.5가 된다. 두번째 타입의 셀은 2*bytes*(signed short)에 저장된다. 그런데, 모든 2*bytes*의 계수들은 12*bits*로 저장될 수 있기 때문에 상위 4*bits*는 사용되지 않는다. 따라서, 이 4*bits*에 소수 부분에 해당하는 값들을 저장함으로써 정확도를 향상시킬 수가 있고, 이 때 최대 오차는 $\frac{1}{2^5}$로 줄어든다. 이와같이

두가지로 구분된 셀의 타입에 대한 정보는 태그를 저장하는 셀 태그 테이블내에 인코딩 된다. 즉, 태그가 가질 수 있는 최대값이 64이므로 최상위 비트(most significant bit)가 사용되지 않는다. 이와같은 두 가지 타입의 셀에 포함된 계수들은 1-*byte* 스트림(stream)과 2-*byte* 스트림이라 불리는 두 개의 1차원 배열에 저장된다. 이때, 0이 아닌 태그를 갖는 셀 내에도 0인 계수들이 존재할 수 있다. 따라서, 셀 내의 계수가 0인지 아닌지를 확인한 후에 0이 아닌 계수만을 해당 스트림으로 저장하는 과정이 수행되어야 한다. 복원 과정에서 랜덤 액세스를 지원하기 위해서는 위치에 대한 정보도 함께 인코딩해야 하는데, 이를 위해 셀 정보(cell information)이라 불리는 추가적인 정보의 유지가 필요하다. 여기에는 셀 내의 각 계수값이 0인지 아닌지를 표현하는 의미지도(significance map)라 불리는 4×4×4의 비트-플래그(bit-flag)와 현재 셀의 첫번째로 0이 아닌 계수에 대한 해당 스트림 내에서의 주소를 포함하는 옵셋(offset)정보가 포함된다. 각 셀당 의미지도의 저장을 위한 8*bytes*와 옵셋을 위해 2*bytes*가 필요하다.

예를 들어, 그림 5.2에서 9번 셀을 고려하자. 이것은 셀 순서로는 17번째 셀에 해당한다(0부터 인덱스가 시작하므로). 우선, 셀 태그 테이블로부터 0001000라는 이 셀에 대한 태그를 얻을 수 있는데, 이것은 이 셀에 대한 정보가 8번째 셀 정보 블럭(cell information block)에 저장되어 있음을 뜻한다. 또한 최상위 비트가 0인 것은 셀 내의 모든 계수들이 2-*byte* 스트림에 저장된다는 것을 의미한다(실제 구현에서 0은 2-*byte* 스트림을 1은 1-*byte* 스트림을 의미한다.). 만일 의미지도 내의 (5,5)위치에 있는 계수를 디코딩해야 한다면, 이 위치 앞에 나타난 1의 갯수를 계산해야만 한다. 이 경우 앞에 있는 1의 갯수가 10개이고 이것은 구하려는 계수가 셀 내에서 11번째로 0이 아닌 계수라는 것을 의미한다. 따라서,

2-*byte* 스트림 내에서의 주소는 옵셋에 10을 더함으로써 계산된다(10 + 14 = 24).

다. 디코딩

Haar기저를 사용하여 분해되고 압축된 3차원 볼륨 데이터를 복원하기 위한 공식은 다음과 같다.

$$
\begin{aligned}
\widehat{c_1} &= c_{LLL} + c_{LLH} + c_{LHL} + c_{LHH} + c_{HLL} + c_{HLH} + c_{HHL} + c_{HHH} \\
\widehat{c_2} &= c_{LLL} + c_{LLH} + c_{LHL} + c_{LHH} - c_{HLL} - c_{HLH} - c_{HHL} - c_{HHH} \\
\widehat{c_3} &= c_{LLL} + c_{LLH} - c_{LHL} - c_{LHH} + c_{HLL} + c_{HLH} - c_{HHL} - c_{HHH} \\
\widehat{c_4} &= c_{LLL} + c_{LLH} - c_{LHL} - c_{LHH} - c_{HLL} - c_{HLH} + c_{HHL} + c_{HHH} \\
\\
\widehat{c_5} &= c_{LLL} - c_{LLH} + c_{LHL} - c_{LHH} + c_{HLL} - c_{HLH} + c_{HHL} - c_{HHH} \\
\widehat{c_6} &= c_{LLL} - c_{LLH} + c_{LHL} - c_{LHH} - c_{HLL} + c_{HLH} - c_{HHL} + c_{HHH} \\
\widehat{c_7} &= c_{LLL} - c_{LLH} - c_{LHL} + c_{LHH} + c_{HLL} - c_{HLH} - c_{HHL} + c_{HHH} \\
\widehat{c_8} &= c_{LLL} - c_{LLH} - c_{LHL} + c_{LHH} - c_{HLL} + c_{HLH} + c_{HHL} - c_{HHH}
\end{aligned}
$$

이 과정은 분해 과정의 역으로 수행된다. 다시말해, 부모 노드에서 계산된 평균 컬러 값 c_{LLL}과 현재의 노드가 저장하고 있는 7개의 주파수 성분의 계수값 $c_{LLH}, c_{LHL}, c_{LHH}, c_{HLL}, c_{HLH}, c_{HHL}, c_{HHH}$ 을 이용하여 현재 노드가 차지하는 해당 위치의 복셀의 평균값을 계산할 수가 있다. 이때, 최하위 레벨의 노드까지 내려오지 않고 중간의 레벨에서의 결과값을 사용하면 중간 해상도의 이미지를 얻을 수 있는데, 이렇게 중간 해상도로 렌더링이 가능한 것은 웨이블릿 변환이 다해상도 분석(multi-resolution analysis)을 지원하기 때문이다.

그림 5.3의 알고리즘(algorithm)은 분해되고 압축된 하나의 16×16×16의 단위블럭으로 부터 인덱스 (i, j, k)에 위치한 계수가 어떻게 디코딩 되는지를 설명한다. 우선 (i, j, k)가 어느 셀에 포함되는지를 계산한

Input : a 16×16×16 encoded decomposed unit block and an index (*i, j, k*)
Output : the decoded value of the coefficient with index (*i, j, k*)

Find the cell C that contains the index (*i, j, k*).
If the tag for C is 0, return 0. [case 1]
Compute the relative index (*i', j', k'*) in C.
If the bit-flag for (*i', j', k'*) is 0, return 0. [case 2]
Count the number of preceding non-zero coefficients by table access.
Add the displacement to the offset to compute the correct address.
Access the appropriate data stream, and return the value. [case 3]

그림 5.3 웨이블릿 디코딩 알고리즘

다. 그리고 셀 태그 테이블로 부터 그 셀의 태그를 확인하는데, 만일 그 값이 0이라면 셀 전체가 0인 경우이므로 계수의 값은 0이 된다([case 1]). 그렇지 않은 경우라면, 셀 내에서의 해당 인덱스 (i', j', k')을 계산하고 의미지도를 조사한다. 이때, 의미지도의 (i', j', k')에 대한 비트-플래그가 0이라면, 이 계수의 값은 0이다([case 2]). 만일 비트-플래그가 1이라면 인덱스 (i, j, k)의 계수는 0이 아닌 경우이므로 셀 태그의 최상위 비트에 의해 결정되는 셀의 타입을 확인하고, 1-*byte* 스트림 또는 2-*byte* 스트림에 저장된 실제 계수를 액세스 하는 추가적인 계산이 수행되어야 한다. 즉, 의미지도 내에서 (i', j', k') 이전에 나타난 1의 갯수를 세고, 그 값을 셀 정보 내에 저장된 오프셋과 합하여 해당 스트림 내

에서의 적절한 주소를 계산할 수 있다. 이렇게 계산된 스트림 내에서의 위치를 이용하여 인덱스 (i, j, k)의 계수를 정확하게 액세스 하게 된다 ([case 3]).

(i', j', k') 이전에 나타난 1의 갯수를 세는 계산을 효과적으로 수행하기 위해서 16*bits*로 만들어질 수 있는 모든 순열(permutation) 2^{16}(=65536)가지에 대한 테이블을 미리 만들어 사용하였는데, 이 테이블은 16*bits* 입력에 대한 1의 갯수를 유지하고 있다. 따라서 1의 갯수를 세는 계산을 단지 몇 번의 테이블 액세스만으로 가능하도록 함으로써 속도를 향상시킬 수 있다. 실제로 0이 아닌 하나의 계수를 디코딩할 때, 1의 갯수를 세는 수행은 평균 2.5번의 테이블 액세스만으로 가능하다.

4. 실험 결과와 성능 분석

본 실험에서 사용한 테스트 데이터는 Visible Man CT 데이터의 상위 512개의 슬라이스(slice) 이미지로 구성된다. 하나의 CT 슬라이스 이미지의 해상도는 512×512이고, 실제 복셀값은 12*bits*로 저장된다. 따라서 생성된 테스트 볼륨 데이터의 해상도는 512×512×512이고 각 복셀을 2*bytes*로 표현하면 전체적으로 256*MByte*의 크기를 갖는다. 물론 이러한 실험 데이터가 매우 방대한 크기를 갖고 있다고 생각되지는 않지만, 본 연구에서 구현한 압축 스킴에서는 압축 단위로 16×16×16의 단위블럭을 기본으로 하고 있기 때문에, 더욱 방대한 크기의 볼륨 데이터에 대한 실험 결과를 쉽게 추측할 수 있을 것이다.

실험 결과 수치는 175*MHz*, R10000 CPU를 갖고 있는 SGI Octane 워크스테이션을 이용하여 얻은 것이다.

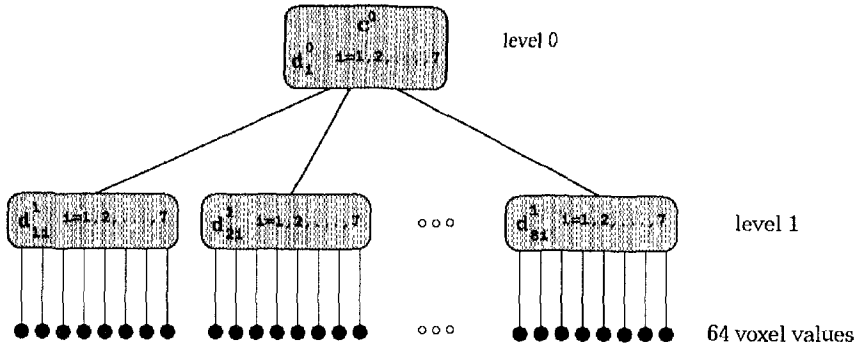


그림 5.4 4×4×4 부분 영역에 대한 팔진트리 표현

가. 복원을 위한 계산 비용

우선 하나의 단위블럭으로 부터 하나의 복셀값을 복원하는 경우를 생각하자. 웨이블릿 변환 과정에서 2번의 변환을 수행하였으므로, 복원 과정도 2번에 걸친 수행이 필요하다. 이것은 2개의 레벨을 갖는 팔진트리(octree)를 방문(traverse)하는 것과 동일한 과정이다. 그림 5.4는 단위블럭의 4×4×4 부분 영역(subregion)을 팔진트리 형태로 표현한 것을 보여 준다. 루트 노드(root node)에서, 평균 계수 c^0는 부분 영역 내에 있는 모든 복셀의 평균값을 갖고 있으며, 7개의 나머지 계수들 $d^0{}_i$ $(i=1,2,\cdots,7)$은 레벨 1에 있는 8개의 자식 노드(child node)들의 평균값(이 값들은 8개의 2×2×2의 부분 영역에 대한 평균값이 된다.)을 계산하기 위해서 필요한 디테일(detail) 값들이다. 결국, 레벨 1에 있는 각 노드에 대한 평균 계수 $c^1{}_j$ $(j=1,2,\cdots,8)$이 루트 노드에 의해 계산되면, 이 값과 8개의 각 노드에 있는 디테일 계수 $d^1{}_{ji}$ $(j=1,2,\cdots,8,\ i=1,2,\cdots,7)$을 이용하여 각 노드의 해당 영역에 대한 8개의 복셀값을 복원하게 된다.

Haar 기저를 사용한 경우, 8개의 복원 공식 각각은 7번의 덧셈/뺄셈만

으로 이루어져 있다. 따라서, 하나의 복셀을 복원하기 위해서는 2개의 레벨을 거쳐야 하므로 14번의 덧셈/뺄셈이 필요하다. 또한, 복원을 위해서, 레벨 0에서 8개 그리고 레벨 1에서 7개, 합해서 15개의 웨이블릿 계수들이 디코딩되어야만 한다. 이미 앞절의 디코딩 알고리즘을 통해 설명한 바와 같이, 하나의 웨이블릿 계수를 디코딩하는 과정은 3가지 경우로 구분된다(그림 5.3). 즉, 포함된 셀 자체가 널(null)인 경우([case 1]), 셀은 널이 아니지만 계수 자체가 0이기 때문에 비트 플래그가 0인 경우([case 2]), 그리고 0이 아닌 계수라서 해당 스트림에 그 값이 저장되어 있는 경우([case 3])로 나눠진다. [case 1]과 [case 2]인 경우의 계산 비용이 상당히 낮다는 것을 쉽게 이해할 수 있다. 그리고 [case 3]의 경우는 평균적으로 2.5번의 테이블 액세스와 해당 스트림의 주소를 결정하기 위한 덧셈의 수행이라는 추가적인 계산을 요구한다. 그러나 웨이브릿 압축 과정에서 0이 아닌 계수의 비율이 3%에서 10%정도 이므로(즉, 97%에서 90%의 계수가 0이므로) 대부분의 경우가 [case 1]이나 [case 2]에 해당된다. 따라서, 하나의 단위블럭으로 부터 하나의 복셀값을 복원하는데 소비되는 전체적인 계산 비용은 상당히 낮다고 말할 수 있다.

볼륨 가시화를 위해 가장 대표적으로 사용되는 기법으로 광선추적법(ray-casting)과 스플래팅(splatting)을 생각할 수 있다. 이 두가지 기법은 복셀값을 액세스하는 패턴에서 상당히 높은 지역성(locality)을 나타낸다. 따라서, 성능의 향상을 위해 단위블럭 내의 4×4×4 부분 영역을 새로운 복원 단위로 생각할 수 있다. 어느 작은 영역의 복셀값만을 액세스 하고자 할 때, 16×16×16 크기의 단위블럭을 복원하는 것 보다는 4×4×4 크기의 부분 블럭을 복원하는 것이 더욱 효과적일 것이다. 이것은 64 (=4·4·4)개의 복셀값에 대한 팔진트리를 방문하는 경우에 해당하고, 8개의 복원 공식 전체를 9번 적용하는 것을 의미한다. 8개의 복원 공

식 전체가 56 (=7·8)번의 덧셈/뺄셈을 필요로 하지만, 중복된 계산을 줄여 최적화하면 56번의 덧셈/뺄셈을 24번의 덧셈/뺄셈으로 간단히 줄일 수가 있다. 따라서, 64개의 전체 복셀들의 복원을 위해서는 9·24번의 덧셈/뺄셈이 수행되어야 한다. 이것은 한 복셀당 $\frac{9\cdot 24}{64}=3\frac{3}{8}$번의 덧셈/뺄셈 연산이 필요함을 의미한다. 4×4×4 크기의 부분 블럭 전체를 복원하기 위해서 필요한 디코딩 되어야 할 웨이블릿 계수는 64개이다. 따라서, 부분 블럭 내의 64개 전체 계수가 일단 디코딩 된다고 할 때, 각 복셀당 평균 디코딩 횟수는 1번이 된다. 결국, 하나의 복셀값을 복원하기 위해 1번의 디코딩과 $3\frac{3}{8}$번의 덧셈/뺄셈 연산 비용이 필요하다는 결론을 얻게 된다. 물론 비트 연산이나 주소 계산을 위한 추가적인 비용이 요구되기는 하지만, 데이터 압축의 효율성과 성능을 고려할 때 이것은 상당히 적은 비용이라고 말할 수 있다.

나. 압축율과 복원 화질

이 절에서는 압축율을 수치적으로 분석하고자 한다. 단위블럭은 압축전에 16×16×16×2 *bytes*를 갖고 있다. 그림 5.2의 인코딩 스킴을 보면, 우선 한계값(threshold)과 태그 테이블을 저장하기 위해 각각 2*bytes*와 64*bytes*가 소비된다. 적어도 하나의 0이 아닌 계수를 포함하는 셀의 경우는 추가적인 정보의 저장이 필요한데, 의미지도의 저장을 위한 8*bytes* (=4·4·4*bits*)와 옵셋을 위해 2*bytes*가 필요하다. 전체 셀의 개수 (4·4·4=64)중에서 적어도 하나의 0이 아닌 계수를 포함하여 0이 아닌 태그를 갖는 셀의 갯수의 비율을 α라고 하자. 이것은 $\frac{\text{널이아닌셀의갯수}}{64}$와 같이 표현되고, 이것은 셀 정보를 저장하기 위해 $(8+2)\cdot 64\cdot\alpha$

*bytes*가 필요함을 의미한다. 웨이블릿 압축 수행 후에 0이 아닌 계수들은 1-*byte* 스트림 또는 2-*byte* 스트림으로 나뉘어 저장된다. 0이 아닌 전체 계수 중에서 1-*byte* 스트림에 저장되는 계수의 비율을 β라고 하면, 이것은 $\frac{1-byte\text{스트림에저장되는계수의개수}}{\text{사용된모든0이아닌계수의갯수}}$와 같다. 또한, 압축에서 사용된 0이 아닌 웨이블릿 계수의 비율을 γ라고 하자. 그러면 압축율 ρ는 다음과 같이 정리될 수 있다. :

$$\begin{aligned}\frac{1}{\rho} &= \frac{2+64+(8+2)\cdot 64\cdot\alpha+16\cdot 16\cdot 16\cdot\gamma\cdot(\beta+2\cdot(1-\beta))}{16\cdot 16\cdot 16\cdot 2}\\ &= \frac{33}{4096}+\frac{5\cdot\alpha}{64}+\gamma\cdot\left(1-\frac{\beta}{2}\right)\\ &\approx 0.008057+0.078125\cdot\alpha+\gamma\cdot(1-0.5\cdot\beta)\end{aligned}$$

위 식 $\frac{1}{\rho}$의 첫번째와 두번째 항은 셀 태그 테이블, 한계값, 그리고 셀 정보를 저장하기 위해 사용되는 비용을 의미한다. 위의 압축율은 제로트리(zerotree) 기법을 사용하는 경우 더 좋아질 수도 있겠지만, 빠른 랜덤 액세스를 지원하지 못하는 한계를 갖는다. 해상도 512×512×512를 갖고 복셀당 2*bytes*로 저장된 256*MBytes* 테스트 볼륨 데이터에 대해 압축을 수행한 결과는 표 5.1과 같이 정리된다. 웨이블릿 압축은 한계값 τ가 주어질 때, τ보다 작은 절대값을 갖는 계수들을 잘라냄으로써(0으로 치환함으로써) 수행된다. 즉, 적절한 한계값 τ를 결정하는 것이 압축에서 중요한 문제가 된다. 본 기법에서는 사용하고자 하는 계수의 비율 $\bar{\gamma}$가 결정되었을 때, 이에 해당하는 τ값을 자동으로 계산할 수 있도록 설계하였다. 가장 이상적인 경우는, 압축 대상이 되는 전체 볼륨 데이터에 대한 모든 웨이블릿 계수들에 대해 정렬(sorting)을 하고 ($\bar{\gamma}\cdot$ 전체웨이블릿계수의갯수)번째로 큰 절대값을 갖는 계수를 찾고, 이것

을 한계값 τ로 설정하는 것이다. 그러나 이러한 방법은 볼륨 데이터의 크기가 방대한 경우에는 거의 실현 불가능한 방법이다.

이미 언급했듯이, 본 연구의 압축 방법은 16×16×16의 단위블럭을 기본으로 압축을 수행한다. 해상도 512×512×512인 테스트 볼륨 데이터의 경우, 32,768 ($=32^3$)개의 단위블럭을 갖는다. 각 단위블럭 i에 대해 웨이블릿 변환을 수행한 후, 전체 웨이블릿 계수 4,096 ($=16^3$)개 중에서 0이 아닌 계수의 비율을 r_i라고 하자. 이 비율은 단위블럭 내의 복셀값의 복잡도가 얼마나 큰가를 단적으로 보여주는 척도(measure)로 근사화(approximation)될 수 있다. 그러므로, 좋은 압축율과 복원 화질을 유지하기 위해, r_i값에 비례하여 각 단위블럭 i에 대한 인코딩 될 계수의 갯수를 할당하고, 전체적으로 사용되는 계수의 비율이 $\overline{\gamma}$가 되도록 하는 것은 상당히 합리적인 방법일 것이다. 테스트 볼륨 데이터에 대해 $512^3 \cdot \overline{\gamma}$개의 0이 아닌 계수들이 32,768개의 단위블럭들에 할당된다. 즉, 각 단위블럭 i에 대해 $n_i = \frac{\gamma_i}{\sum_j \gamma_j} \cdot 512^3 \cdot \overline{\gamma}$개의 계수를 할당하는데, 여기서 $\frac{\gamma_i}{\sum_j \gamma_j}$는 상대적인 복잡도의 척도를 의미하게 된다. 그리고, 각 단위블럭 i는 그 블럭 내에서 n_i번째로 큰 웨이블릿 계수를 한계값 τ_i로 하여 그 보다 작은 절대값을 갖는 계수들을 0으로 대치하는 과정을 수행한다. 실제로 이러한 비례적인 계수 할당 방법을 사용함으로써, 전체 단위블럭의 계수들에 대해 하나의 한계값을 적용하여 압축을 수행했을 때 발생할 수 있는 블럭화 현상(blocky effect)를 줄일 수 있었다.

표 5.1의 결과를 보면, 실제 사용된 계수의 비율 γ가 기대하는 계수의 비율 $\overline{\gamma}$와 동일한 값을 갖지 않는 것을 볼 수 있다. 이것은 단위블럭 내

표 5.1 압축율과 복원 화질에 대한 실험 결과

		$\overline{\gamma}$: Desired Ratio of the Wavelet Coef's Used				
		3%	5%	7%	10%	15%
Compression Performance	Compressed Size (MB)	9.56	14.23	18.26	23.78	32.11
	ρ	26.8:1	18.0:1	14.0:1	10.8:1	8.0:1
	α	0.1303	0.2172	0.2789	0.3527	0.4461
	β	0.7984	0.8352	0.8570	0.8788	0.9032
	γ	0.0295	0.0490	0.0681	0.0965	0.1434
Errors in Voxel Values	SNR (dB)	23.7	26,7	29.3	32.3	36.3
	PSNR (dB)	41.8	44.8	47.4	50.4	54.4
Errors in Normals	Skin (deg)	21.8	16.7	13.2	9.6	6.6
	Bone (deg)	27.3	21.4	17.2	12.9	8.6

의 웨이블릿 계수들 중 한계값 τ_i와 같은 절대값을 갖는 계수가 한개 이상 존재하는 경우가 있기 때문에 발생한다. 그리고 표 5.1에서, α와 β는 32,768개의 전체 단위블럭들에 대한 평균 수치이다. 실험에서는 $\overline{\gamma}$ = 0.03, 0.05, 0.07, 0.10, 그리고 0.15에 대한 압축율을 분석했는데, 8.0:1에서 26.8:1까지의 압축율을 얻을 수가 있었다.

그림 5.5, 그림 5.6, 그림 5.7, 그리고 그림 5.8은 압축하지 않은 원본 Visible Man 데이터의 가슴부분 CT 슬라이스 이미지와 $\overline{\gamma}$ = 0.03, 0.05, 0.07의 값을 사용하여 3차원적으로 압축하고 복원한 3개의 이미지를 비교하여 보여주고 있는데, 이를 통해 상당히 좋은 압축율과 복원 화질을 얻을 수 있음을 확인할 수 있다. 그림 5.9, 그림 5.10, 그림 5.11, 그리고 그림 5.12는 뼈(bone)에 해당하는 물질(material)을 볼륨 광선 추적법

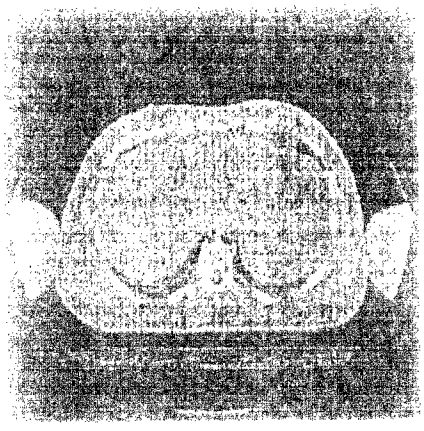


그림 5.5 Original

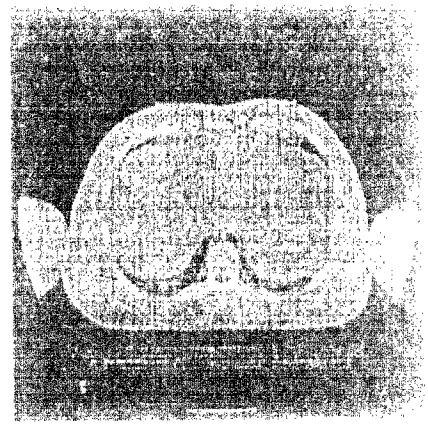


그림 5.6 $\overline{\gamma}$ = 0.07

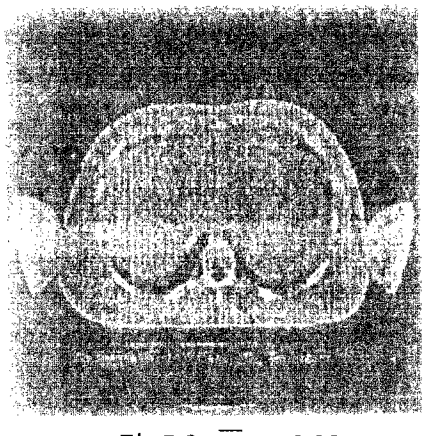


그림 5.8 $\overline{\gamma}$ = 0.03

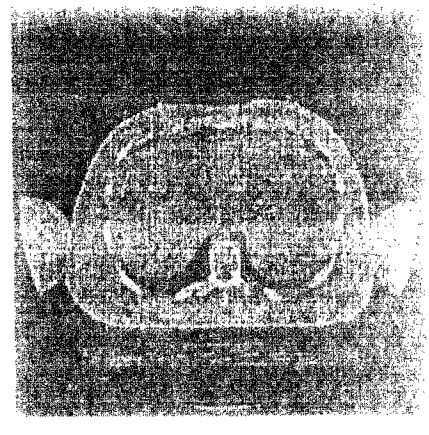


그림 5.7 $\overline{\gamma}$ = 0.05

(volumeray-casting)을 이용하여 이미지를 보여주며, 그림 5.13, 그림 5.14, 그림 5.15, 그리고 그림 5.16은 피부(skin)를 가시화한 결과이다. 본보고서 내용에는 없지만, $\overline{\gamma}$ = 0.10, 0.15로 압축된 데이터를 이용해 가시화한 이미지는 원본 데이터로 부터 얻은 이미지와 거의 구별할 수 없을 정도의 좋은 화질을 갖는다. $\overline{\gamma}$ = 0.03인 경우의 이미지는 원본 데이터로 부터 얻은 이미지와의 차이를 쉽게 구별할 수 있지만, 전체적인 특

징은 대부분 그대로 유지하고 있음을 볼 수 있다[2].

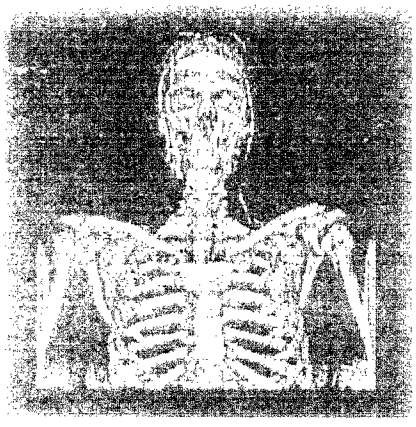


그림 5.9 Uncompress

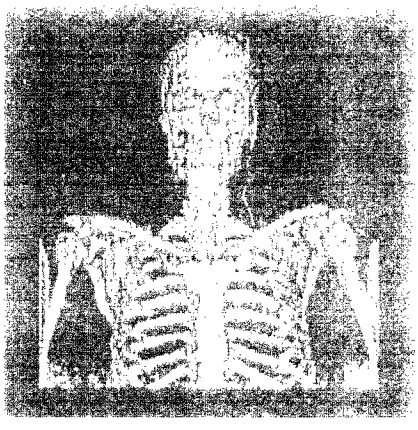


그림 5.10 $\overline{\gamma}$ = 0.07

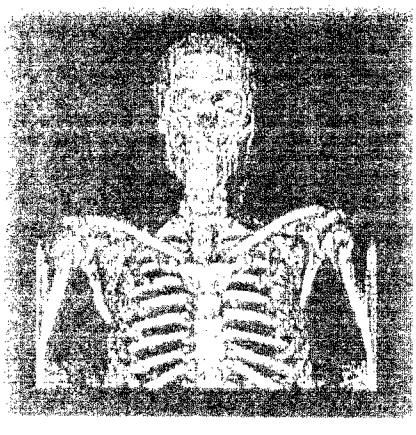


그림 5.11 $\overline{\gamma}$ = 0.05

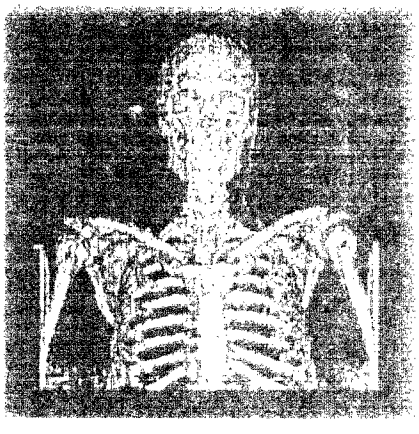


그림 5.12 $\overline{\gamma}$ = 0.03

원본 볼륨 데이터와 압축으로부터 복원된 데이터의 복셀값 사이의 차이(difference)나 왜곡(distortion)을 수치적으로 측정하기 위해, 두 개의

2. 가시화된 뼈의 이미지에서 양쪽 팔 부분에 나타난 앨리어싱(aliasing)과 피부 이미지의 뒤쪽에 나타난 판 부분은 가시화 자체의 문제로 인해 발생된 것이 아니라, 뼈와 피부에 해당하는 밀도값(복셀값)과 유사한 밀도값을 갖는 물질이 같이 가시화됨으로써 나타난 것이다. 이것은 전처리 과정이나 정확한 분류(classification or segmentation) 과정을 통해 제거될 수 있는 문제이다.

2차원 이미지 사이의 차이를 나타내는 척도로서 주로 이용되는 SNR(mean-square Signal-to-Noise Ratio)과 PSNR(mean-square Peak-Signal-to-Noise Ratio)을 3차원 볼륨 데이터에 대한 경우로 확장하여 적용하였다(표 5.1).

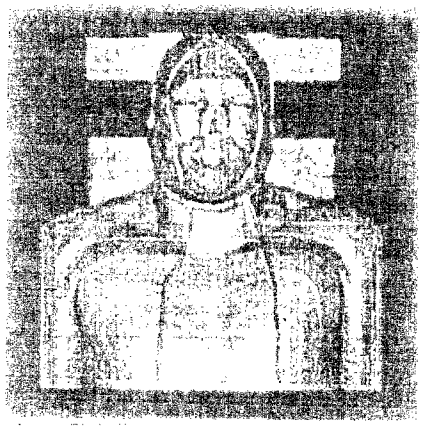


그림 5.13 Uncompress

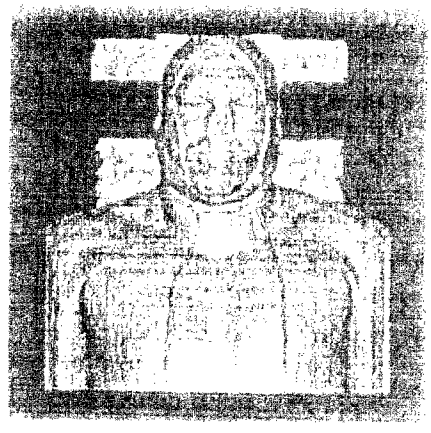


그림 5.16 $\overline{\gamma}$ = 0.03

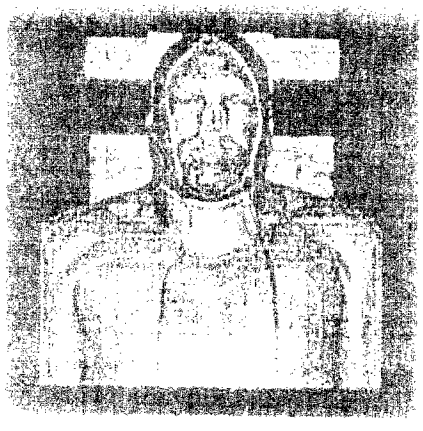


그림 5.15 $\overline{\gamma}$ = 0.05

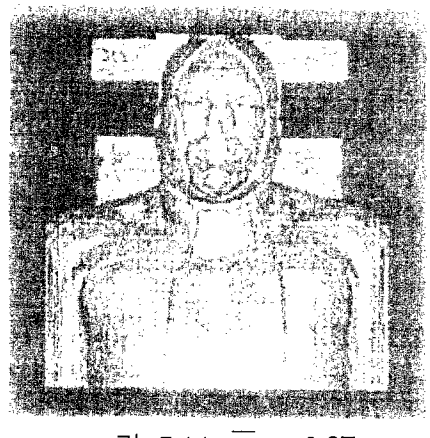


그림 5.14 $\overline{\gamma}$ = 0.07

그리고 원본 볼륨 데이터와 복원된 데이터의 동일한 샘플링 위치에 대한 각각의 노말벡터(normal vector)가 이루는 오차 각도의 크기를 측정하였다(표 5.1). 노말벡터는 볼륨 가시화에서 렌더링된 이미지의 화질을 결

정하는 가장 중요한 요소로 작용할 뿐만 아니라, 볼륨 데이터 내에 존재하는 두 가지 이상의 물질을 분류(classification)하는 과정에서도 중요한 역할을 한다. 일반적으로 이것은 중차(central difference)의 공식을 이용하여 계산되므로 복원된 복셀값의 오차가 큰 경우라면 두 노말벡터 사이의 각도도 커지게 된다. 실제 실험에서는 뼈와 피부 각각에 해당하는 복셀값의 위치에 대해서만 중차공식을 이용하여 노말벡터를 계산하였다. Ning[5-14]은 5:1의 압축율을 갖고 압축된 두 가지 데이터에 대해 SNR과 노말벡터 사이의 오차를 측정한 실험 결과를 분석했다. 이 결과에 의하면, 각 데이터에 대한 SNR의 평균 수치는 17.6*dB*과 17.7*dB*이고 노말벡터 사이의 오차의 평균은 20.0*deg*와 22.6*deg*이다. 이러한 수치와 본 기법의 실험 결과를 비교해 볼 때, 제안된 압축 기법의 성능이 얼마나 우수한지를 확인할 수 있다.

다. 액세스 속도

본 절에서는 구현된 인코딩 스킴으로 부터 얼마나 빠른 랜덤 액세스가 가능한가에 대해 두 가지 경우에 대해서 각각의 성능을 분석 한다(표 5.2). 첫번째 고려 대상은 순수한 랜덤 액세스(pure random accsee)의 경우이다. 이것은 랜덤하게 발생된 인덱스 (i, j, k) 위치의 복셀값을 수회에 걸쳐 액세스할 때 소비되는 시간을 측정함으로써 이뤄진다. 복원을 위한 오버헤드(overhead)를 측정하기 위해 압축하지 않은 512×512×512의 볼륨 데이터를 배열(array)로 부터 백만회에 걸쳐 랜덤 액세스하고 동일한 수행을 압축된 데이터에 대해 적용하였다. 이러한 "순수한 랜덤 액세스" 시간은 압축된 데이터로부터 액세스하는 경우가 약 4배 정도 시간이 더 소비된다는 결과를 얻었다. 두번째 상황은 블럭 단위로 복원을 수행하는 경우를 고려하는 것이다. 일반적인 볼륨 가시화 기법들을 살펴보

면, 어떤 규칙적인 패턴으로 볼륨 데이터를 액세스하게 되는 경우가 흔히 발생된다. 예를들어, 스플래팅 기법은 시점 방향(view direction)을 따라 앞에서 뒤의 순서(front-to-back order)로 슬라이스(slice) 단위로 복셀을 방문하면서 가시화를 수행한다. 이와같이 규칙적인 액세스 패턴을 갖고 복원이 수행되는 경우, 또는 어느 조그만 특정 영역 내의 복셀값만을 반복적으로 복원해야 하는 경우에는 4×4×4 부분 블럭을 하나의 단위로 하여 복원을 수행하는 것이 더욱 효과적일 수 있다. 이런 경우는 64(=4^3)개의 복셀값이 거의 동시에 복원을 필요로 하는 경우를 의미하는데, 이미 언급한 것처럼 구현된 인코딩 스킴은 4×4×4의 단위에 대해서 전체 복셀이 복원될 때 더욱 효과적인 성능을 발휘할 수 있다. 우선, 512×512×512의 볼륨 데이터 내에 있는 모든 셀이 앞에서 뒤의 순서로 복원되는데 소비되는 시간을 측정하였다. 그리고, 뼈나 피부로 분류된 복셀을 적어도 하나 이상 포함하고 있는 단위블럭 내의 셀만을 복원하는데 걸린 시간을 측정하였다. 이러한 선택적인 복원을 수행하기 위해서 16×16×16의 각 단위블럭에 대해 복셀의 최대-최소값을 유지하는 자료 구조를 이용하였다. 다시말해, 모든 단위블럭들은 앞에서 뒤의 순서로 방문되고, 최대-최소값과 해당 물질(뼈 또는 피부)로 분류된 복셀값의 범위 사이에 교차(intersection)되는 부분이 있는지를 조사하여 이를 만족하는 단위블럭 내의 셀만을 복원하였다. 피부의 경우에 실제로 11,000개의 단위블럭 (16^3 · 11,000개의 복셀)이 복원되었다. 시간 측정 결과에 의하면 모든 단위블럭을 복원하는 경우, 압축된 데이터로 부터의 복원 시간이 원본 데이터의 액세스 시간의 약 1.6배 정도가 됨을 확인할 수 있었다. 그리고 피부에 대해 수행한 복원 시간의 경우, 압축하지 않은 원본 데이터의 경우와 압축 데이터로 부터의 복원 시간이 약 2.4초에서 6.4초 밖에 차이가 나지 않음을 확인할 수 있었는데, 이것은 볼륨 렌더링과 같이 CPU의 계산

		$\bar{\gamma}$: Desired Ratio of the Wavelet Coef's Used					
		Uncompressed	3%	5%	7%	10%	15%
Pure Random		2.18	7.75	8.20	8.62	9.15	9.84
Cell-Wise	All	19.95	30.31	31.58	32.52	33.62	35.05
	Skin	6.70	10.13	10.75	11.32	12.10	13.02

표 5.2 복원 시간에 대한 실험 결과

시간이 수행 시간의 대부분을 차지하고 수백초 이상을 필요로 하는 응용에서는 거의 무시할 수 있는 시간차이다.

5. 압축 기법의 응용

가. 볼륨 네비게이션 시스템

볼륨 데이터를 이용해 탐사를 수행하는 몇가지 시스템들이 개발되었다. [5-7]에서는 Visible Human 데이터의 대장(colon)을 가상적으로 탐사하는 대장 탐사 시스템(virtual colonoscopy system)을 개발했다. 그들은 전처리를 통해 대장만을 추출하고 이를 다각형 데이터(polygon data)로 변환시켜 시스템에서 사용한다. 따라서, 그래픽 하드웨어를 이용하여 빠른 속도로 탐사가 가능하지만 직접 볼륨 렌더링(direct volume rendering)에 의해 생성된 이미지가 아니라는 한계를 갖고 있다. Visible Human 데이터를 매우 섬세하게 가시화하는 시스템으로 Voxel Man이 있다[5-25]. 이 시스템은 CT, MRI 데이터와 RGB 데이터를 이용하여 매우 다양하고 좋은 화질의 이미지 생성하며, 의미 네트워크 지식 기반 시스템(semantic network knowledge base system)을 추가하여 교육용으로 유용하게 이용될 수 있도록 개발되었다. 그러나, Visible Human 데

그림 5.17 볼륨 네비게이션 시스템 SGVN

이터의 머리 부분만을 가시화하기 위해서 96*MBytes* 이상의 메인 메모리를 필요로 하기 때문에, 전체 데이터의 탐사를 위해서는 수십 *GByte*의 메모리가 요구된다는 문제점을 내포하고 있다.

SGVN(SoGang Volume Navigation) 시스템은 현재 개발 중인 시스템으로 본 연구를 통해 개발된 기법을 이용해 압축된 Visible Human 데이터를 로드하여 광선추적법 또는 스플래팅 기법과 같은 직접 볼륨 렌더링으로 인체 내부의 원하는 부위를 탐사하도록 구현되었다(그림 5.17). 이 시스템은 웨이블릿의 다해상도 성질을 이용하여 LOD(Level Of Details)를 구현함으로써 가시화 속도를 향상시켰으며 그 밖에 속도 향상을 위한 여러가지 효과적인 기법들을 포함하고 있다. 카메라의 제어와 가시화 파라메터의 변경 등과 같은 기능들은 그래픽스 사용자 인터페이스(GUI)를 통해 자유롭게 이뤄지므로 일반 사용자도 쉽게 가시화를 할 수 있다. 직접 렌더

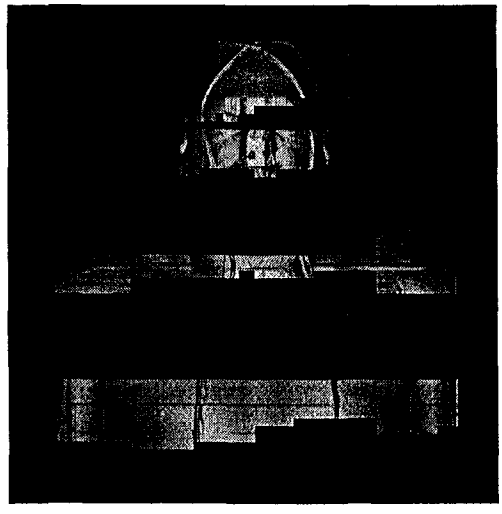


그림 5.18 병렬 렌더링 수행중인
이미지

링을 이용하기 때문에 가시화를 원하는 밀도값의 범위를 바꿔가면서 다양한 물질(material)을 가시화할 수 있고, 압축된 데이터를 사용하기 때문에 메모리의 문제도 자연스럽게 해결될 수 있다. 그러나 높은 압축율을 갖는 데이터를 사용하는 경우, 많은 디테일 정보가 손실된 상태이기 때문에 선명하지 못한 이미지(blurred image)를 생성한다는 단점을 나타낸다. 그러나 이 시스템은 PC를 이용하는 일반적인 환경에서 교육용으로 매우 효과적으로 이용될 수 있을 것이다.

나. 병렬 볼륨 렌더링

병렬 프로그램(parallel programming)의 성능에 가장 중요한 요소가 되는 통신 시간(communication time)과 부하 균형(load balancing)을 효과적으로 해결하기 위한 많은 연구가 병렬 볼륨 렌더링 분야에서도 계속되어 왔다. 만일 본 압축 기법을 병렬 볼륨 렌더링에서 사용하는 경우, 전체 데이터를 각 PE(Process Element)들이 가질 수 있기 때문에 볼륨 데이

터를 주고받기 위해 필요한 통신 시간을 없앨 수 있다는 장점을 갖게 된다. 따라서, 효과적인 부하 균형이 이뤄질 수 있는 알고리즘만 개발되면 거의 선형적인 속도향상(linear speed-up)을 기대할 수 있을 것이다.

본 연구의 압축 기법을 이용한 병렬 볼륨 렌더링에 대한 연구가 Cray T3E 시스템상에서 현재 진행중이다(그림 5.18).

6. 결론

본 연구에서는 웨이블릿을 이용하여 매우 방대한 크기의 볼륨 데이터를 3차원적으로 압축하는 효과적인 압축 스킴을 구현하였다. Visible Human 데이터에 대해 수행한 실험 결과를 통해, 본 기법이 상당히 높은 압축율을 유지하면서 임의의 위치의 복셀값을 빠르게 랜덤 액세스 할 수 있음을 보였다. 이 기법은 방대한 크기의 볼륨 데이터 전체를 메인 메모리에 로드하고, 이를 인터렉티브(interactive)하게 가시화하는 응용에서 매우 효과적으로 이용될 수 있다. 특히 이 기법은 제한된 메모리를 갖는 범용의 워크스테이션이나 PC 상에서도 방대한 볼륨 데이터를 효과적으로 가시화할 수 있게 함으로써, 더 많은 사람들이 방대한 데이터로부터 얻을 수 있는 유용한 정보를 쉽게 획득할 수 있도록 만들어 줄 것이다.

본 연구에서는 Haar 기저를 이용한 웨이블릿 변환과 복원을 이용하였다. 이 기저는 계산적으로는 매우 효과적이지만 이미지 압축에서 일반적으로 사용되는 기저들에 비해 낮은 압축 성능을 나타낸다. 따라서, 현재는 정규직교 Daubechies 기저를 이용한 실험을 수행 중에 있다. 이 기저는 Haar 기저에 비해 더욱 정확한 압축이 가능하지만 계산적으로는 더 많은 비용을 필요로 한다. 그리고, 그 밖의 다른 필터들을 이용한 경우에

대해서도 압축의 정확성과 계산 비용 사이의 상관 관계를 분석할 계획이다. 본 기법을 이용하여 다양한 볼륨 가시화를 수행하는 경우에 동일한 위치의 복셀값을 반복적으로 복원해야 하는 현상이 발생하게 된다. 따라서 이러한 경우, 계산량을 줄이기 위해 복원된 복셀값을 임시로 보관하는 캐쉬(cache) 자료 구조를 고안할 필요가 있다. 복셀 액세스의 지역성(locality)와 인접성(coherence)을 고려할 때, 이러한 자료 구조의 이용은 상당히 많은 중복 계산을 줄이는 효과를 낳을 것으로 예상된다.

마지막으로, 구현된 압축 기법을 사람 인체 내부를 인터렉티브하게 가시화할 수 있는 가상 탐사 시스템(virtual navigation system)으로 확장이 진행중에 있으며 Cray T3E를 이용한 병렬 렌더링에 본 압축 기법을 적용하는 연구가 수행되고 있다.

제 6 절 다중사용자를 위한 장면 데이터베이스 편집 기술에 대한 연구

1. 서론

가. 연구 배경 및 목표

네트워크 환경이 폭넓게 보급되면서 여러 사용자가 동시에 필요한 데이터를 공유하여 사용할 수 있는 기술이 요구되고 있다. 특히, 그래픽스를 이용해 가시화 된 정보는 사용자가 보다 쉬운 방법으로 접근하고 이해할 수 있도록 도와준다. 따라서, 3차원 데이터를 구성하고 가시화하며, 이것을 네트워크 환경에서 공유할 수 있도록 하는 기술은 현재 개발되고 있는 네트워크를 기반으로 한 응용 시스템 개발에 있어서 그 기반이 될 수 것이다.

본 세부과제에서는 다중 사용자가 동시에 접근할 수 있는 장면 데이터베이스를 지원하는데 필요한 기반 기술 연구와 시스템 개발을 목표로 하고 있다. 이런 연구는 현재 많은 분야에서 그 유용성으로 연구·응용되고 있지만 실제 개발이 용이하지 않은 네트워크 환경을 기반으로 하는 그래픽스 응용 시스템을 보다 쉽고 단기간에 개발할 수 있게 하는 기반 기술로 응용될 수 있을 것이다.

이런 시스템의 기반인 그래픽스 분야는 최근까지 주로 독립된 시스템(stand-alone) 환경에서의 작업을 중심으로 이루어져 왔다. 그 결과로 대부분의 그래픽스 시스템은 통신망 장비의 확충과 빠른 접근 속도에도 불구하고, 하나 이상의 사용자들이 공유하기에는 적당하지 않다는 문제를 지니고 있다. 따라서 본 세부과제 연구에서는 이미 존재하는 독립된 그래

픽스 시스템에서 제공하는 3차원 장면을 네트워크 환경에서 다중사용자가 공유·제어하는 방법을 제시하고 그 방법을 기반으로 응용 시스템을 제작하여 실험 및 평가과정을 통해 시스템을 보완하는 것을 목표로 삼는다.

나. 연구추진 전략 및 방법

시스템 개발에 기본적으로 필요한 장면 데이터베이스와의 인터페이스는 Step2000 2차년도에 본 연구팀에서 수행한 "객체 지향 렌더링 인터페이스에 대한 연구"의 결과를 이용하여 확장하였다. 장면 데이터베이스는 하나의 사용자를 위한 렌더링 기능 중심의 설계로 기하학적인 모형과 그에 따른 특성(property) 등의 성격을 갖는 객체들로 장면 데이터베이스를 정의하고 있다. 따라서, 다중 사용자 환경에서 장면 데이터를 공유하는 기능을 지원하기 위해 동적인 특성을 가지는 ActiveObject를 추가하였으며, 네트워크 기능을 보다 편리하게 사용할 수 있는 NetObject 객체를 기반으로 하여 각각의 네트워크 기능을 담당하는 Server, Group, Client를 제공한다. 이런 네트워크 객체는 응용 시스템 개발자로 하여금 네트워크에 대한 전문적인 지식없이 보다 수월하게 네트워크 환경을 구축할 수 있도록 한다.

본 연구에서 개발하는 라이브러리는 객체 지향 방법을 이용한다. 객체 지향 기법이 제공하는 가장 큰 특징 두 가지는 캡슐화(encapsulation)와 계승성(inheritance)이라 할 수 있는데, 캡슐화는 프로그램의 모듈화를 유도할 수 있고 계승성은 렌더링 툴킷이 사용자에게 제공하려는 유연성(flexibility)을 해결해 줄 수 있을 것이다.

다. 관련 연구 고찰

다중 사용자를 위한 시스템의 구성은 네트워크 환경을 기반으로 하기

때문에, 여러 사용자가 보다 효율적으로 공유 공간을 유지하기 위해서 공유되어 있는 데이터의 일관성(consistency) 유지가 요구된다. 네트워크를 통해서 공유된 데이터의 일관성을 유지하는 기능을 제공하는 시스템들로는 MR Toolkit[6-1], SIMNET[6-2], NPSNET[6-3], DIVE[6-4], RING[6-5], WAVES[6-6], BrickNet[6-7]등이 있으며, 이들은 각각 서로 다른 데이터 전송 방법을 채택하고 있다.

MR Toolkit은 가상현실에 필요한 여러 가지 요소에 대한 제어 기능들을 package로 나누어서 가상현실을 제작하는 개발자에게 필요한 부분을 제공한다. 이런 여러 가지 package들 중 네트워크 환경을 제공하는 역할은 peer package가 담당한다. MR Toolkit의 peer package인 경우는 각 노드 사이에 연결되어 있는 point-to-point message 방식으로 데이터 일관성을 유지한다. 이런 경우에는 n개의 노드가 존재한다면, 하나의 노드에서는 일관성을 유지하기 위해 n-1개의 메시지가 사용되어야 한다.

이런 방법은 각 가상 공유 공간의 상태 변경 단계마다 모든 노드가 n-1개의 메시지를 전송하므로 SIMNET, NPSNET에서는 브로드캐스팅(broadcasting) 방법으로 이런 과도한 메시지 전송·처리의 부담을 줄였다. 즉, 가상 공간에 참여하는 노드는 수정된 데이터 메시지를 브로드캐스팅 방식으로 다른 노드에게 전송하므로써 각 가상 공유 공간의 상태 변경 단계마다 모든 노드는 1개의 메시지를 전송하므로 전체적으로 n개의 update message가 전송되는 O(n)의 메시지 전송을 필요로 한다[6-5].

그러나, 이러한 브로드캐스팅 방식도 여전히 모든 노드가 메시지를 처리하는 부담을 가지고 있다. 실제 NPSNET이나 SIMNET의 실험결과를 보면 방대한 가상공간을 시뮬레이션 하는 동안, 상당수의 노드들은 대부분 시간을 전송된 메시지를 처리하는데 보내고 있는 것으로 나타났다. 따라서, 브로드캐스팅은 저가의 기계를 가지고 있는 많은 사용자들을 관리하기에

는 비효율적인 측면을 가지고 있다[6-8]. 또한, 메시지 전송·처리 문제점 외에도 모든 노드가 같은 데이터를 가지고 있고 다른 노드의 상태에 대한 정보를 가지고 있기 때문에, 가상 공간의 데이터가 방대해지면 노드의 능력에 따라서 참여가 제한되며, 노드 수가 증가될수록 시스템 효율이 낮아지는 결과가 나타난다.

DIVE는 point-to-point 방식을 사용하는 네트워크 환경을 제공한다. 그러나 위에서 제시한 MR Toolkit이나 SIMNET, NPSNET과는 달리 데이터를 그룹단위로 나누어 제공하여 노드마다 다른 데이터를 공유할 수 있게 하고, 해당 그룹단위의 멀티캐스팅을 지원하여 각 그룹내의 노드만이 해당 메시지를 받아서 처리하도록 한다. 이런 방식은 MR Toolkit이나 SIMNET, NPSNET에서 나타난 문제점을 감소하였지만 멀티캐스팅을 구현하기 위해서 하드웨어적인 요소를 필요로 하고, 멀티캐스팅 주소가 한정되어 있어 확장성이 낮은 단점을 가지고 있다[6-6].

Wave나 RING은 message manager를 두어 DIVE에서의 멀티캐스팅과 비슷한 기능을 제공한다. Message manager는 노드 사이의 통신을 중재하여 필요한 노드에게 해당 메시지를 전송하는 역할을 담당한다. 즉, 위의 point-to-point나 브로드캐스팅과 같이 모든 노드가 동시에 동일한 메시지를 받아서 처리하지 않고 message manager에 의해서 메시지가 필요하다고 판단되는 노드에게 메시지가 전송되고 해당 노드들만 메시지를 처리한다.

BrickNet도 Wave와 RING과 마찬가지로 message broker를 이용하여 메시지 전송을 제어한다. 이들 Wave, RING, BrickNet은 Client/Server 구조를 가지고 있으며 이런 구조가 갖는 문제점인 낮은 fault tolerance를 개선하기 위해서 여러 개의 서버를 두는 방식을 채택하고 클라이언트에게 데이터 분산·저장함으로써 서버의 부담을 줄이고 확장성을 높이는 방향

으로 구성되었다. 그러나 이런 방식은 데이터를 공유하기 위해서는 해당 노드가 접속되어 있는 서버와의 연결이 요구되므로 서버 사이의 부가적인 메시지 지연이 발생하는 부담을 가지고 있다.

2. 시스템 설계 및 구현

가. 시스템 개요

다중 사용자를 위한 장면 데이터베이스 시스템은 다음과 같이 그래픽스 처리 기능을 위해 OpenGL과 같은 그래픽스 시스템과 디스플레이를 위한 X-window와 같은 윈도우 시스템 및 네트워크에 대한 환경을 제공하는 운영체제를 기반으로 하여 그림 6.1과 같은 구조로 이루어져 있다.

다중 사용자를 위한 장면 데이터베이스 시스템은 크게 3가지로 구성되어 있다. 전체적인 장면 구성을 담당하는 SceneData Manager와 그것을 화면에 표현하고 사용자의 입력을 받아들여 처리하는 Display Manager 그리고, 네트워크 기능을 담당하는 Network Manager가 본 시스템을 구성하는 주요 요소이다. 이 요소들은 네트워크 환경의 다중 사용자 응용 시스템을 구성하는데 보다 편리한 네트워크 인터페이스와 향상된 그래픽스 기능을 제공한다.

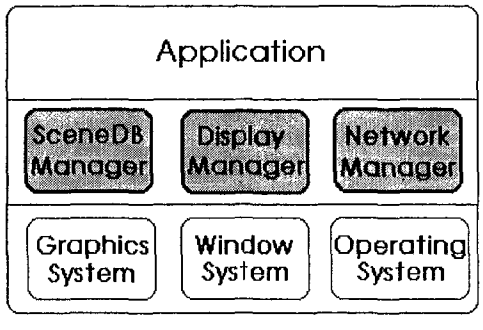


그림 6.1 전체 시스템 구조

(1) 시스템 구현 환경

(가) H/W 환경

Indy, SunSparc 2, SunSparc 10 (with Audio device)

(나) S/W 환경

Graphics System : OpenGL

Window System : X-Window

Operating System : UNIX(Solaris 5.2.1), IRIX 5.3 이상

Programming Language : C++

나. SceneDB Manager

SceneDB Manager는 그래픽스 시스템이 제공하는 기본적인 기능을 기반으로 3차원 공간상에 객체를 계층적으로 표현하고, 객체의 생성·삭제·편집 및 파일이나 네트워크 환경을 위한 스트림으로의 입·출력 기능을 제공한다. 본 시스템에서 제공하는 장면 데이터베이스는 계층적인 구조로 이루어져 있으며 기존의 장면 데이터베이스[6-9]에 동적인 객체 생성을 위한 ActiveObject와 네트워크 환경에서의 데이터 전송을 위해서 스트림으로의 입·출력 기능을 추가하여 구현하였다.

(1) 장면 데이터베이스 구성

장면 데이터베이스는 장면을 이루는 노드와 객체의 특성을 표현하는 Property로 이루어져 있다. 장면을 이루는 노드로는 그림 6.2에서와 같이 기하학적인 형태를 가지는 Geometry와 여러 개의 하위 노드들을 가지는 Container로 이루어져 있다. Geometry와 Container는 객체가 갖는 고유한 속성을 부가할 수 있는데, 이러한 객체의 특성은 Property를 이용하여 표현된다. Property는 객체가 그려지는 형태(point, line, shade)나 빛에 의해 색상을 나타내는 요소들(ambient, diffuse, specular), 표면의 특성으로 나타나는 형태들(transparency, shininess, emission)이나 질

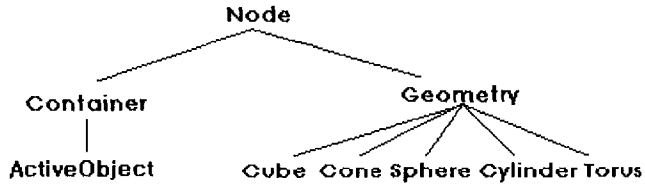


그림 6.2 장면데이터 베이스 계층도

감 표현(texture)으로 구성된다.

(2) ActiveObject

기존의 장면 데이터베이스에서 추가된 부분인 ActiveObject는 동적인 특성을 포함하고 있는 노드로서 사용자에 의한 픽킹이나 타이머에 의한 일정한 주기, 또는 부딪힘 감지(collision detection)와 같은 특정한 조건이 만족될 때 일정한 행동이 실행된다. 이러한 행동은 ActiveObject내의 메시지를 이용하여 특정 함수를 호출하여 실행된다.

ActiveObject는 Container class를 상속받아 표 6.1에서와 같이 일정한 행동을 실행할 객체들의 집합과 실행 함수로 분기하기 위한 메시지와 함수인자를 가지고 있다.

```
ActiveObject {
      visible 1
      name   door
      children {
         Cube {
            width   1
            height   1.5
            length   0.2 } }
      message   MOVE
      argument [ bell.au otherRoom 45 0.9 0 -0.5 ]
    }
}
```

표 6.1 ActiveObject

(3) 장면 데이터베이스의 예

위와 같은 장면 데이터베이스 구성요소를 이용하여 3차원 공간인 거실을 생성하여 보면 그림 6.3에서와 같이 전체 노드의 가장 상위 노드인 livingroom과 거실의 벽을 구성하는 wall node, 전등을 구성하는 lamp node, 액자를 구성하는 frame node들로 이루어져 있다. lamp의 shade는 action을 필요로 하는 ActiveObject이며, 각 노드는 하나의 객체를 구성하기 위해 Cube나 Cylinder, Sphere, Cone과 같은 여러 개의 기하학적인 객체들로 이루어져 있다.

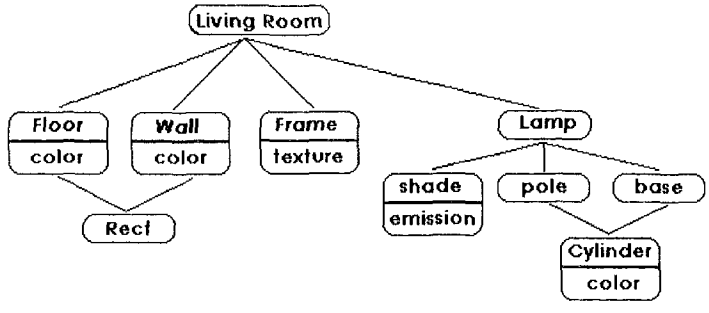


그림 6.3 장면데이터의 예(거실)

다. Display Manager

3차원 장면을 형성한 후 그것을 화면에 표현하는 역할을 담당하는 것이 Display Manager이다. Display Manager는 윈도우를 생성하고 3차원 장면을 표현하기 위한 요소인 Light와 Camera를 설정하며, 사용자의 입력을 받아들이는 Event Handler와 선택된(picking) 객체의 경로(path)를 추적하는 기능을 제공한다. 계층적인 장면 데이터베이스에서 경로는 다중 사용자 환경의 네트워크 상에서의 수정된 객체의 전송을 용이하게 하며, 데이터의 양을 감소시키는 역할을 한다. 즉, 그림 6.3에서 전등의 갓을 선택하였을 때 넘겨지는 경로는 그림 6.4에서와 같이 나타나며 이것을 표현하면 [3 2 0] 이다. 이런 경로는 실제 네트워크 상에서 유용하게 사용

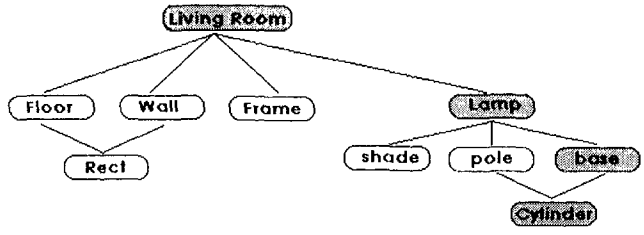


그림 6.4 객체 Picking과 경로

될 수 있다. 즉, 네트워크 상에서 메시지 전송을 줄이면서 선택되거나 수정된 객체를 정확히 표현하기 위해서 경로는 간단하면서도 정확한 정보를 제공한다.

라. Network Manager

(1) 전체 네트워크 구조

본 시스템에서 제안하는 네트워크 구조는 다중 그룹을 두어 네트워크 기능과 데이터관리 기능을 여러 곳으로 분산하여 서버에게 모든 역할이 집중되는 병목현상을 방지한다. 그림 6.5에서와 같이 전체 네트워크 구조는 그룹들의 네트워크 상태를 관리하는 하나의 서버와 여러 개의 그룹들, 그룹에 연결되어 서비스를 제공받는 여러 클라이언트들로 이루어져 있다. 각 클라이언트는 한 그룹에서 다른 그룹으로 이동함으로써 여러 가지 다른 서비스들을 제공받을 수 있으며 각 그룹은 자신에게 연결된 클라이언트만을 관리하여 전송되는 메시지의 양을 최소화할 수 있다. 그림 6.5와 같은 분산된 네트워크 구조는 그 양이 방대한 가상공간과 같은 장면을 여러 개로 나누어 관리함으로써 보다 효율적인 성능을 기대할 수 있다.

(2) 네트워크 시스템 설계

네트워크 환경을 제공하는 시스템을 설계하기 위해서 고려해야 할 사항으로는 다음과 같은 것이 있다.

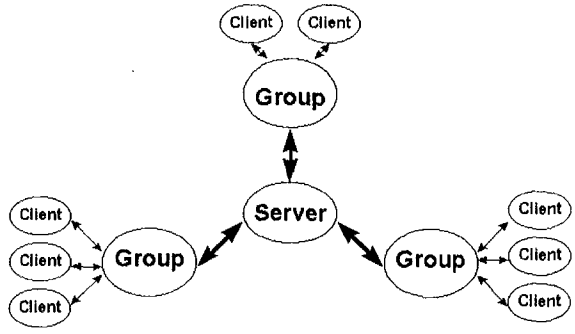


그림 6.5 전체 네트워크 구조

▶ 네트워크 모델 : 분산 네트워크 모델/중앙집중형 네트워크 모델

▶ 메시지 전송방법 :
point-to-point/broadcasting/multicasting/message broker

▶ 데이터 분산 방법 : pull replication/partial replication/shared object

▶ 네트워크 프로토콜 : TCP/IP, UDP/IP

본 네트워크 시스템에서는 데이터의 일관성 유지가 쉽고 노드에 대한 네트워크 정보 관리의 부담이 적은 중앙집중형 네트워크 모델을 사용한다. 이런 네트워크 모델인 경우 서비스를 제공하는 서버의 부담이 가중되는 단점을 가지고 있지만, 이런 단점은 그룹이라는 서비스 노드들을 분산하여 해결하도록 하고 있다. 또한, 노드마다의 메시지 전송은 그룹에서 해당 그룹에 연결된 클라이언트들에게 메시지 전송을 담당하는 message broker형태로 구현되었으며, 데이터는 하나의 노드에 부담을 줄여주는 부분복제(partial replication)방법으로 클라이언트는 서비스 노드인 그룹이 제공하는 장면 데이터를 복제하여 사용한다. 클라이언트, 그룹, 서버 사이의 데이터 전송은 UDP 프로토콜과 socket 인터페이스를 기반으로 노드마다 직접 메시지를 전송하는 유니캐스팅(unicasting)형태로 구현되었다. UDP 프로토콜은 메시지의 도착여부를 보장할 수 없지만 빠른 속도로

메시지를 전송하며, 네트워크 연결과 해지가 쉬운 장점을 가지고 있다.

(3) 네트워크 객체

위와 같은 선택사항을 고려하여 각 노드 사이의 네트워크를 통한 메시지 전송 기능을 제공하는 클래스로 NetObject가 있다. 이 NetObejct는 표 6.2에서와 같이 UDP 프로토콜을 이용한 socket 인터페이스로 구성되었다.

NetObject를 이용하여 그림 6.2에서와 같이 전체 네트워크 시스템을 구성하는 각 노드들은 서버나 그룹 또는 클라이언트의 네트워크 객체 중 하나로 구성된다. 이들 객체는 각각의 역할에 따라서 아래와 같은 기능을

```
class NetObject {
    public :
            NetObject();
           ~NetObject();
            void Adjoin();
            void ReadMessage(char* buff, int size);
            void WriteMessage(char* buff);

            int socketfd ;
            struct sockaddr_in receiveAddr;
            struct sockaddr_in sendAddr;
            int portNumber;
            char hostAddr[20];
           }
```

표 6.2 NetObject Class

제공하여 응용시스템 개발자로 하여금 보다 편리하게 네트워크 환경을 구축하도록 한다. 다음은 그림 6.2에서 제시하고 있는 네트워크 객체의 세부설명이다.

(가) Server

분산되어 있는 서비스 노드들을 관리한다. 즉, 분산된 서비스 노드의 주소와 이름을 관리하여 노드의 이동에 필요한 네트워크 정보를 제공한다. 이런 네트워크 정보는 서버에만 존재하여 각 서비스 노드마다 또는 각 클라이언트마다 모든 노드에 대한 네트워크 정보를 관리하지 않고도

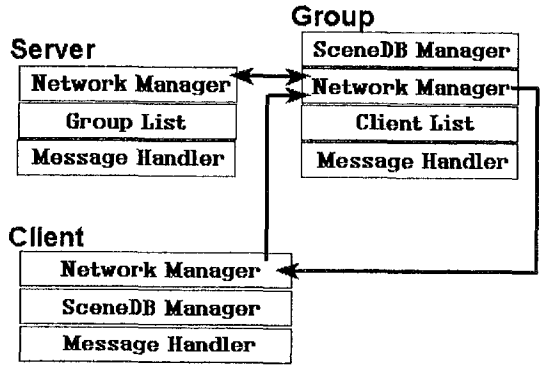


그림 6.6 네트워크 객체 구성요소

네트워크 기능을 제공받을 수 있으므로 네트워크 정보 관리에 대한 부담이 감소된다.

- Network Manager

  NetObject를 이용하여 네트워크로부터 Client나 Group으로부터 들어오는 요청을 처리하고 그에 해당하는 데이터를 전송한다.

- GroupList

  현재 Server에 연결되어 있는 그룹의 정보를 관리한다. 그룹의 이름, 호스트 주소, 포트 번호를 저장하였다가 필요한 요청이 들어오며 해당정보를 제공한다. 이런 네트워크 정보는 Server에서만 관리하므로 다른 노드에서는 이에 대한 정보를 저장·관리할 필요없이 참조하여 사용하면 된다. 따라서, 네트워크 정보의 저장·관리에 필요한 장소 및 시간에 대한 부담을 줄일 수 있다.

- Message Handler

  사용자가 정의한 메시지에 대한 처리를 담당한다. 클라이언트, 서버, 그룹 사이에는 네트워크 연결을 위한 기본적인 메시지만이 제공된다. 그러나, 응용 시스템은 그 종류와 특성에 따라서 노드 사이에 전송되는 다양한 메시지를 필요로 하기 때문에 기본적으로 주어지는 메시지 이외에 사용자가 정의하여 사용하는 메시지와 그에 대한 처리함수가

필요하다. Message Handler는 사용자 정의의 메시지와 그에 따른 핸들러를 등록 받아 관리하고 네트워크로부터 전송되는 메시지를 적절한 함수로 분기(dispatch)하는 기능을 제공한다.

(나) Group

그룹은 클라이언트에게 서비스를 제공하여 주는 기능을 담당한다. 그룹은 서버에게 자신의 위치를 등록하여 여러 가지 종류의 서비스를 분산하여 효율적으로 제공할 수 있는 방법을 제시한다. 응용 시스템 개발자는 자신이 개발할 시스템의 환경을 관리하기 편한 형태나 크기로 분산하여 각 그룹에 맵핑함으로써 시스템을 보다 효율적으로 관리할 수 있으며 방대한 양의 데이터를 필요로 하는 시스템인 경우에도 분산하여 구현할 수 있다. 또한, 그룹은 자신에게 연결된 클라이언트에서 수정된 객체를 그룹 내의 클라이언트에게만 전송하므로 메시지 전송의 부담을 줄이고 필요한 노드에게만 필요한 메시지를 전송하므로 보다 효율적으로 네트워크를 사용한다.

각 그룹은 자신에게 연결되어 서비스를 제공받는 클라이언트의 리스트, 자신이 제공하는 SceneDB, 비동기적인 메시지 전송을 위한 listening process, 사용자 정의 형태의 메시지 처리를 위한 Message Handler를 관리한다.

- Network Manager

  서버나 클라이언트로 전송되는 메시지를 받고 그에 대한 데이터를 전송하는 역할을 담당한다. 그룹에서 제공되는 네트워크를 통한 메시지 전송은 그 그룹에 연결된 클라이언트에게만 효과가 있다. 기본적으로 제공되는 그룹의 네트워크 관리 기능은 그룹이 형성될 때 서버에게 그룹의 위치와 존재를 알리는 'Register' 기능과 클라이언트의 접속을 받고 처리하는 'Adjoin' 기능, 클라이언트에게 장면 데이터를 전

송하는 기능, 서버에게 다른 그룹의 위치를 요청하는 기능들이다.

- Message Handler

  서버와 마찬가지로 그룹에게 전송되는 사용자 정의 메시지를 처리하는 기능을 담당한다. 그룹이 받는 메시지는 그룹을 생성한 후 'AddMessage' 함수를 이용하여 메시지와 처리 함수를 등록하여야 사용 가능하다.

- SceneDB Manager

  그룹은 서버와는 다르게 자신의 장면 데이터를 관리한다. 이 장면 데이터는 클라이언트에게 제공될 것이며 서버와는 독립적으로 관리·유지된다. 이런 장면 데이터는 그 그룹내의 클라이언트에게만 유용하며, 장면 데이터가 변경·수정시 그룹내의 모든 클라이언트에게 변경된 객체의 정보를 전송하는 토대를 이룬다.

- Client List

  그룹은 현재 자신에게 연결된 클라이언트의 위치—호스트 주소, 포트 번호—를 관리하여 필요한 메시지를 전송할 때 사용한다. 이런 클라이언트 리스트는 그룹의 상태를 해당 그룹내의 클라이언트에게만 전송하기 위해 필요한 자료구조이다.

(다) Client

클라이언트는 서비스 노드 사이를 이동하면서 필요한 서비스를 제공받을 수 있는 기능을 제공한다. 즉, 각 클라이언트는 자신에게 전송되는 비동기적인 메시지 처리를 위한 listening process, 사용자 정의 형태의 메시지 처리를 위한 Message Handler, 서비스 노드로부터 전송 받은 장면 데이터를 관리하기 위한 SceneDB로 이루어져 있다.

- Network Manager

  서버나 그룹으로부터의 메시지를 받아서 처리하는 기능을 담당한다.

클라이언트에서 기본적으로 제공되는 네트워크 기능은 클라이언트 생성시 서버에게 등록하거나 그룹에 참여하는 'Adjoin' 기능과 그룹을 옮기는 'Move' 기능 등이 있다.

· Message Handler

그룹이나 서버로부터 전송되는 사용자 정의 메시지를 처리하는 기능을 담당한다. 그룹과 서버에서 전송되는 사용자 정의 메시지는 클라이언트 생성 후 등록하여 사용할 수 있다.

· SceneDB Manager

클라이언트는 그룹으로부터 네트워크를 통해 장면 데이터를 전송받아 사용한다. 네트워크를 통해 전송된 장면 데이터는 클라이언트의 SceneDB를 통해 메모리로 올려오고 관리·수정된다. 클라이언트에 의해 수정된 장면 데이터의 내용은 그룹으로 전송되어 그 그룹내의 다른 클라이언트에게 전송된다.

(라) 기본적인 네트워크 기능

· 네트워크 시작

우선 공유환경에서 서비스를 제공하기 위한 그룹들은 서버에 등록을 한 상태가 되면 사용자가 서비스를 받을 수 있는 상태가 준비된다. 우선 서버가 호스트에 존재하고 그룹이 서버에 등록이 되면 클라이언트는 그림 6.7과 같은 순서로 네트워크를 시작하여 서비스를 제공받을 수 있다.

① Adjoin

클라이언트가 서버에 접속을 요청한다.

② Ack

서버는 클라이언트에 고유한 ID를 할당하고 시작 그룹으로의 접속을 요청한다.

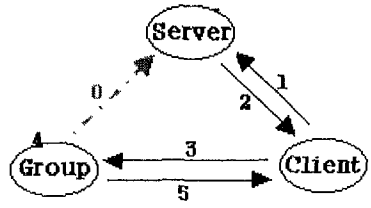


그림 6.7 네트워크의 시작

③ Adjoin

클라이언트는 서버가 제시한 시작 그룹으로 접속을 한다.

④ Insert Client to List

그룹은 접속을 요청한 클라이언트를 리스트에 넣고 관리를 시작한다.

⑤ SendData

그룹은 접속을 요청한 클라이언트에게 장면 데이터를 전송한다.

· 서비스 노드 변경

클라이언트는 자신의 필요에 의해 그룹의 이름을 이용하여 다른 그룹으로 변경한 후 새로운 서비스를 제공받을 수 있다. 단 옮기고자 하는 그룹은 서버에게 미리 등록되어야 한다. 클라이언트는 다음과 같은 과정으로 그룹 사이를 이동할 수 있다.

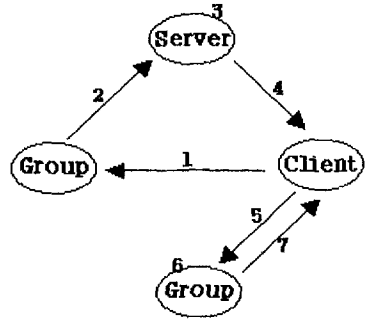


그림 6.8 서비스 노드 변경

① Move

클라이언트는 현재 접속 중인 그룹에게 다른 그룹으로의 이동을 요청한다.

② Move

그룹은 다른 그룹의 위치를 서버에게 요청한다.(클라이언트의 위치와 함께)

③ FindGroup

서버는 요청된 그룹의 존재를 찾아서 그 위치를 얻는다.

④ Goto

서버는 그룹으로부터 받은 클라이언트의 위치를 이용하여 이동하고자 하는 클라이언트에게 새로운 그룹의 위치를 전송한다.

⑤ Adjoin

클라이언트는 서버로부터 전송받은 새 그룹의 위치를 이용하여 새로운 그룹에 접속을 요청한다.

⑥ Insert Client to List

새로운 그룹은 접속을 요청한 클라이언트를 리스트에 넣고 관리를 시작한다.

⑦ SendData

그룹은 접속을 요청한 클라이언트에게 자신의 장면 데이터를 전송한다.

· 그룹단위의 데이터 전송

그룹 내에 접속된 클라이언트에서 장면 데이터를 변화시키거나 자신의 상태를 변화시키면 그 그룹내의 다른 클라이언트에게 이런 변화된 상태의 정보를 전송하여 데이터의 일관성을 유지하도록 한다.

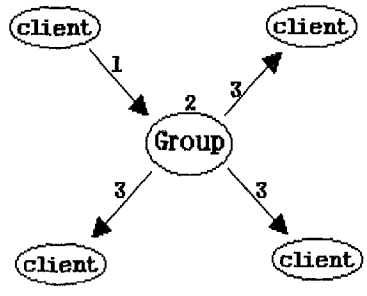


그림 6.9 그룹단위

데이터전송

① Update

클라이언트가 변경된 자신의 장면 데이터를 전송한다.

② UpdateSceneData

그룹은 클라이언트로부터 전송된 데이터를 이용하여 자신의 장면 데이터를 변경한다.

③ Update

그룹은 그룹내의 다른 클라이언트에게 변경된 장면 데이터를 전송한다.

(4) Message Handler

현재 구현된 서버, 그룹, 클라이언트는 연결과 해지 및 데이터전송에 관련된 기본적인 메시지만을 처리하도록 구현되었다. 응용 시스템 개발자가 시스템의 필요에 의해서 노드 사이의 메시지를 추가·확장할 수 있는

방법으로 "Message Handler"를 제공한다. 이 메시지 핸들러는 네트워크를 통해서 송·수신을 원하는 메시지를 메시지 핸들러와 함께 등록하여 사용자 정의의 메시지 및 핸들러를 제공하는 것을 목적으로 한다.

(가) 메시지 핸들러의 구조

메시지 핸들러를 이루는 요소는 표 6.3과 같이 네트워크로 전달되는 문자열 형태의 메시지와 그에 해당하는 함수로 이루어져 있다.

```
class MessageElement {
        char Type[20];
        void (*function) (void *);
};
```

표 6.3 Message Handler Class

메시지 핸들러는 이런 MessageElement들을 리스트 형태 관리하여 사용자가 정의하는 메시지와 그 함수를 저장하고, 해당 메시지에 대한 함수를 호출하는 기능을 제공한다.

(나) 시스템에서 제공되는 기본적인 메시지 핸들러

· 서버

메시지	의미
Register	등록(그룹)
Adjoin	접속(클라이언트)
Move	그룹의 위치 요청(그룹, 클라이언트)
Leave	연결해지(그룹, 클라이언트)
End	네트워크 기능 종료 요청(사용자)

· 그룹

· 클라이언트

메시지	의미
AAck	접속허가(서버)
IGoto	다른 노드로의 이동 위치(서버)
SUpdate	장면데이터내의 개체변환(그룹)
UData	개체 장면데이터(그룹) 노트)
Leave	연결해지(클라이언트)
End	네트워크 기능 종료 요청(사용자)

3. 응용 시스템 개발

가. 개요

(1) 응용 시스템 개발 목적

개발된 시스템은 툴킷 차원으로 실제 응용 시스템을 개발하는데 라이브러리 형식으로 사용하거나 클래스를 상속받아서 확장·사용할 수 있다. 본 시스템을 이용한 간단한 형태의 실제 응용 시스템 개발은 현재 개발된 시스템의 성능을 테스트하고 그것을 바탕으로 보다 향상된 기능을 제공하는데 그 목적이 있다. 실제 프로그램 작성시 사용상 불편한 점과 미비점, 그리고 성능 면에서의 검증은 효과적으로 시스템의 단점을 보완할 수 있다.

(2) 응용 시스템 정의

본 응용 시스템은 여러 사용자가 같은 장면 데이터를 공유하여 그 장면 데이터 중 일부를 수정하거나 변화를 가할 수 있는 기능을 제공하고, 서버와 그룹, 클라이언트를 각각 구현하여 같은 호스트나 서로 다른 호스트 사이에서 네트워크로 연결하여 사용할 수 있다. 또한, 여러 가지 장면 데이터를 실험할 수 있는 환경을 마련하기 위해서 여러 그룹을 두어 각 그룹마다 서로 다른 장면 데이터를 관리하고 서비스할 수 있도록 구성하였다.

장면 데이터베이스 중 본 시스템에서는 기존의 장면 데이터베이스 시스

템[6-9]에서 확장한 ActiveObject를 추가하여 구성하였으며 이런 ActiveObject는 사용자의 마우스 픽킹에 의해 선택될 경우 적절한 자신의 행동을 취하게 된다. 예를 들면, 전등의 갓을 픽킹할 경우 불이 켜지거나, 문을 픽킹할 경우 다른 장면으로 옮겨가거나 오디오나 TV, 시계 등을 픽킹한 경우 소리가 나는 현상 등을 볼 수 있다.

응용 시스템에서 제공되는 장면은 거실, 침실, 옥외 구조를 가지고 있는 집 구조를 관찰할 수 있는 것으로 실제 가옥에서 사용되는 도구들의 간단한 형태로 구성되었다.

(3) 응용 시스템 구조

응용 시스템은 그림 6.10에서와 같이 시스템 소프트웨어인 그래픽스 시스템이나 윈도우 시스템, 운영체제를 기반으로 하고 있는 장면 데이터베이스 시스템이 제공하는 인터페이스를 이용하여 구성되었다.

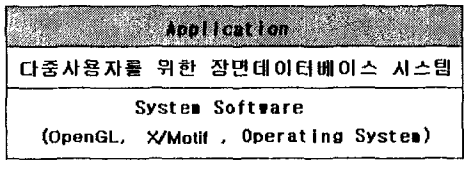


그림 6.10 응용 시스템 구조

나. 응용 시스템 구현

(1) 응용 시스템 전체 구성도

응용 시스템은 그림 6.11에서와 같이 3개의 그룹들과 그것을 관리하는 하나의 서버, 그리고 수를 제한하지 않은 클라이언트로 구성되어 있다. 클라이언트는 서버에 접속한 후 시작 그룹인 "house"로 이동한 후 장면 데이터의 "door" ActiveObject에 의해서 다른 그룹으로 이동하며 장면 데이터를 관찰·변경할 수 있다.

(2) 응용 시스템 구성 요소

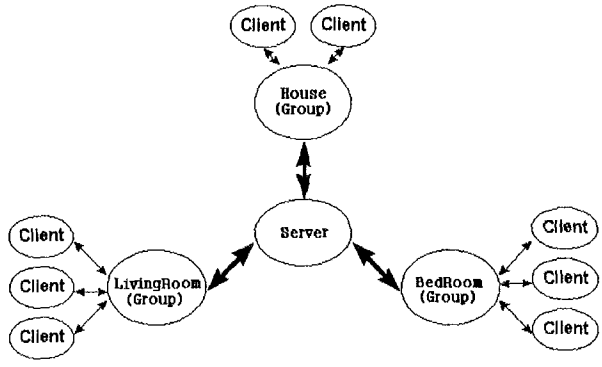


그림 6.11 응용 시스템 구성도

그림 6.11을 구성하는 구성요소인 서버, 그룹, 클라이언트는 Network Manager 이외에도 장면 데이터를 다룰 수 있는 SceneDB Manager 및 사용자의 입·출력을 위한 Window Manager를 사용하여 구성되었다. 다음은 각 세부 구성 요소의 설명이다.

(가) Server

서버는 그룹을 관리하는 리스트와 클라이언트나 그룹으로부터의 메시지를 대기하고 처리하는 NetObject를 가지고 있다. 서버는 이 NetObject를 이용하여 자신이 형성한 포트로 들어오는 메시지를 "polling"하여 전송된 메시지를 기본 핸들러—Register, Adjoin, Move—나 사용자에 의해 정의된 메시지 핸들러를 이용하여 처리한다. 서버는 그래픽스 기능이 필요없기 때문에 간단한 텍스트 형태로 현재 상태가 출력되며, 따라서 SceneDB Manager나 Window Manager를 이용하지 않는다.

(나) Group

그룹은 서버나 클라이언트로부터의 메시지를 대기하였다가 처리하는 Network Manager를 가지고 서버와 마찬가지로 자신이 형성한 포트로부터 "polling" 후 전송된 메시지를 기본 핸들러—Adjoin, Update—나 사용자가 정의한 메시지 핸들러를 이용하여 처리한다. 그룹은 서버와는 다르게 자신의 장면 데이터를 가지고 있다. 단 그룹도 서버와 마찬가지로 간단한

텍스트 형태로 현재 상태를 화면에 출력하며, 이런 그룹의 기능은 가장 간단한 형태만으로도 다중 사용자 시스템에 동참할 수 있게 한다. 즉, 그룹은 윈도우나 그래픽스 환경이 없이도 사용이 가능하다.

(다) Client

클라이언트는 서버나 그룹과 마찬가지로 네트워크를 통한 메시지 처리를 위한 Network Manager를 가지고 있다. 이 Network Manager는 클라이언트가 형성한 포트로 "polling"을 수행하기 때문에 클라이언트는 thread를 이용하여 이 "polling"을 수행한다. 즉 클라이언트는 크게 사용자의 입·출력 부분(윈도우나 마우스)을 담당하는 프로세스와 네트워크로부터 메시지를 대기하는 프로세스로 나누어져 있다. 클라이언트는 구조상으로 보면 크게 세 가지 요소로 이루어져 있다. 즉, 그룹으로부터 전송된 장면 데이터를 자신의 노드에서 관리하기 때문에 그룹과 마찬가지로 SceneDatabase Manager도 필요로 하고, 네트워크를 통한 메시지 처리를 위한 Network Manager부분, 그리고 사용자의 입·출력을 위한 Display Manager로 이루어져 있다. 클라이언트는 서버와 그룹과는 다르게 윈도우 환경 하에서 실행된다. 즉, 사용자의 입력과 출력을 보다 편하게 받아들이고 사용자가 장면 데이터를 화면으로 볼 수 있는 기능을 제공하기 위해서 OpenGL과 X/Motif를 이용한 그래픽스 시스템과 윈도우 시스템 기능을 갖춘 Display Manager가 함께 구현되었다.

(3) 응용 시스템 기능

(가) 새로운 그룹의 추가

현재 응용 시스템에 새로운 장면 데이터를 제공하는 그룹을 추가할 수 있다. 단, 추가할 그룹이 제공하는 장면 데이터는 파일 형태로 이미 형성되어 있어야 한다. 새로운 그룹을 추가하려고 다음과 같이 입력하면 그룹이 제공하는 장면 데이터가 메모리로 올려지고, 네트워크를 통한 메시지

처리를 위해 프로세스를 실행하게 된다. 또한, 서버에 등록되어 클라이언트가 사용할 수 있는 상태가 된다.

group [group_name] [scenedata_filename]

(나) 다양한 특성의 장면 데이터 제공

본 응용 시스템에서 제공되는 장면 데이터는 Property로 Texture, Material, Color를 제공하므로 다양한 특성을 나타내는 장면을 제작할 수

```
property {
  Texture check.txt
}

property {
  Color 0.2 0.1 0.0
}

property {
  Material {
    ambient 0.04 0.04 0.04
    diffuse  0.1 0.1 0
    specular 0.1 0.1 0
    shininess 0.2
    transparency 0.24
   }
}
```

표 6.7 Property 넣기

있다. 이런 특성은 장면 데이터 안에 표 6.7같이 넣음으로서 효과를 낼 수 있다.

(다) ActiveObject를 이용한 분산 노드의 이동

현재 제공되는 응용 시스템에서는 ActiveObject를 이용하여 서비스 노드를 변경하는 기능을 제공한다. 즉, 아래의 그림 6.12에서의 문을 마우스로 선택하면 그림 6.13과 같이 다른 그룹으로 이동한다. 이때 이동하는 새로운 그룹의 이름은 장면 데이터 파일 안에 정의되어 있다.

그림 6.12 거실장면

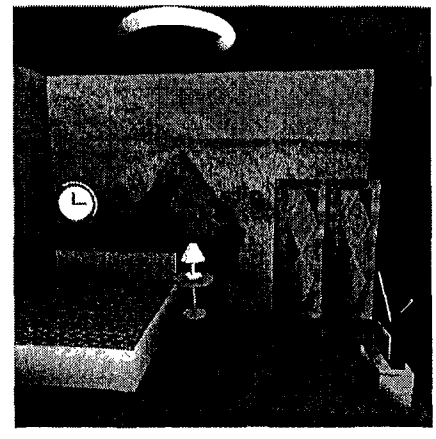


그림 6.13 침실장면

(라) ActiveObject를 이용한 장면 데이터 변경

응용 시스템에서는 사용자가 마우스로 선택된 객체가 ActiveObject인 경우 해당하는 함수를 호출하여 장면 데이터를 변경할 수 있는 기능을 제공한다. 그림 6.12에서 전등의 갓을 선택하면 그림 6.13과 같이 발광체로 변하면서 그 위치에 "Light"를 추가하여 전체 장면의 밝기를 조절할 수 있다. 또한, 그림 6.12의 우측하단의 오디오나 좌측 상단의 시계를 선택하면은 소리를 발생한다. 그 이외에 그림 6.13의 TV인 경우에는 텍스쳐와 소리가 같이 발생하는 ActiveObject이다.

(마) 사용자의 임의의 분산노드 이동

사용자는 그룹의 이름을 알고 있을 경우에 사용자가 임의로 그룹을 이동할 수 있다. 메뉴의 "Move"는 사용자가 그룹이름을 입력하면 해당 그룹으로 이동한다. 단 "Move" 메뉴는 "Start" 메뉴가 실행된 후 실행 가능하다.

(바) 장면 데이터 저장

사용자의 화면에 표현되는 장면 데이터는 현재 접속된 그룹으로부터 전송 받은 것으로 메모리에만 올려져 있다. 사용자가 이 장면 데이터를 자신의 호스트에 직접 저장하려면 "save as" 메뉴를 이용하여 장면 데이터

를 저장할 수 있다.

(사) 장면 데이터내의 객체 이동

사용자는 마우스로 객체를 선택한 후 끌면 마우스의 방향에 따라서 객체를 이동시킬 수 있다.

(아) 시점 변환

사용자는 마우스의 가운데 버튼을 누른 채 이동시키면 현재 시점을 변환시킬 수 있다.

다. 시스템 적용 결과

(1) 네트워크 기능

응용 시스템 개발에서 네트워크 기능은 각 노드의 특성에 따라서 Client나 Group, Server중 하나의 클래스의 인스턴스를 이용하여 사용하거나 기능의 확장을 원할 때에는 클래스를 계승(inherit)받아서 사용한다. 이런 네트워크 객체는 사용할 때 그 내부적인 요소인 포트번호나 socket기능, 네트워크 연결과 해지 등의 세부적이고 전문적인 부분의 이해 없이도 시스템 내에 네트워크 기능을 추가하는데 보다 편리하고 쉬운 방법을 제공하였다.

(2) 객체 경로

장면 데이터의 객체를 수정하거나 변화를 줄 때 네트워크를 통해 다른 노드에게 이 사실을 알릴 필요가 있을 때 수정된 객체의 위치를 객체경로를 통해 정확하고 간결하게 전송하는 기능을 제공하였다. 이런 객체 경로를 이용하여 네트워크로 전송되는 메시지의 크기를 50~60byte 정도를 유지하면서 적은 메시지 크기로 정확한 객체 위치를 파악하는데 많은 도움이 되었다.

(3) 메시지 핸들러

시스템에서 기본적으로 주어지는 네트워크 연결·해지에 관한 메시지 이외에 필요에 의해 메시지를 추가할 경우 사용되었다. 이런 메시지 핸들러는 응용 시스템의 종류에 상관없이 사용자가 임의로 메시지를 수에 제한 없이 추가할 수 있어서 응용 시스템 개발에 있어서 시스템에 적합한 기능을 보다 유연한(flexibility)방법으로 제공하여 사용자 입장에서의 유연성과 확장성을 제공받을 수 있었다.

(4) 타이머에 의한 화면 Refresh

본 응용시스템에서는 현재 화면에 표시된 장면을 일정주기로 refresh하는 기능을 필요로 한다. 즉, 네트워크를 통한 장면 데이터의 변경은 현재 메인 프로세스가 감지할 수 없고, 그것을 위해서는 "polling"이 필요한데 이것은 상당한 부담감이 있으며 사용자의 입력에 의한 데이터 변경마다 화면의 데이터를 다시 그리는 함수 호출없이도 현재 데이터 상태를 자동적으로 나타내주는 기능을 필요로 하기 때문에 X에서 제공하는 타이머를 이용하여 일정시간주기를 가지고 장면을 다시 그리는 기능을 제공한다. 이런 기능을 응용 시스템을 개발하는 과정에서 추가적으로 부여된 기능이다.

(5) Thread를 이용한 Listening process

클라이언트 응용 시스템인 경우에는 사용자 입·출력을 처리하는 메인 프로세스와 함께 동시에 실행되는 네트워크를 통한 메시지 전송을 기다리는 프로세스가 필요하다. 이 두 프로세스는 서로 동시에 실행되어야 하기 때문에 클라이언트 응용 시스템에서는 그룹과 서버와는 다르게 네트워크 기능을 처리하는 프로세스를 "thread"를 이용하여 병렬적으로 수행될 수 있도록 하였다.

제 4 장 연구개발목표 달성도 및 대외기여도

지난 2년간의 연구수행을 통하여 아래의 표와 같이 연구목표를 달성하였다. 원래의 2단계 제안서에서 기술한 연구 목표를 거의 모두 달성하였음을 알 수 있다. 다만 1. Z 버퍼 모듈개발 내용중 레디오시티 부분이 약간 미진한 점이 있으나 현재 이를 계속하여 개발중이므로 단시일 안에 완성될 것이다. 또한 2. 분산 및 병렬 렌더링 내용 중 본 연구에서 개발한 Z 버퍼 모듈을 슈퍼 컴퓨터인 Cray T3E에 포팅하는 과정에서 예기치 못한 하드웨어적인 문제에 부닥쳐 텍스쳐 매핑 부분을 이식하지를 못하였는데 이는 향후 해결되어야 할 문제라 할 수 있다.

본 연구의 주제인 렌더링 관련 기술은 컴퓨터 그래픽스 분야에서 가장 핵심적인 기술중의 하나로서 과거에는 주로 외국의 시스템을 도입하여 사용하는 형편이었다. 지난 2년간의 연구를 통하여 이러한 고급 렌더링 기법을 실제로 구현하고 또한 GUI를 개발하여 봄으로써, 이와 관련하여 상당한 요소 기술을 축적할 수 있었다. 또한 기존의 기법의 개발뿐만 아니라 렌더링 관련 연구를 수행하므로써 상당한 역량을 축적할 수 있었으며, 고급 인력을 양성할 수 있었다. 추후 한 단계 발전된 수준의 개발 및 연구를 통하여 세계적인 개발 역량을 보유할 수 있게 될 것이며, 이는 첨단 영상 관련 소프트웨어 제작 분야의 발전에 기여를 할 수 있을 것이다. 아래의 표에 기술되어 있는 바와 같이 다양한 주제의 연구·개발 목표를 성공적으로 달성하기 위하여 연구 책임자 및 석박사 과정 대학원생들이 뚜렷한 목표의식을 갖고 성실하게 과제를 수행하여 왔다. 그 결과로 전체적으로 원했던 연구 목표를 무난히 달성할 수가 있었다. 지난 2년간의 연구·개발 노력을 통하여 해외 논문지에 2편, 국내 논문지에 3편, 국제 학술대회에 3편, 국내학술대회에 4편 등의 논문을 발표하였거나 또는 발표

예정이며, 현재 2편의 논문이 논문지에 제출중이다. 또한 본 연구와 직접적으로 관련하여 5명의 석사학위 논문을 산출하였다. 본 연구의 많은 부분이 상당한 양의 프로그래밍을 필요로 하는 개발 연구임을 생각하면 이는 적지 않은 수의 논문 발표라 할 수가 있다.

번호	세부연구목표 (연구계획서상에 기술된 목표)	달성내용	달성도 (%)
1	미세다각형을 사용하는 Z-버퍼 방법에 기반을 둔 고급 렌더링 모듈 개발	·Reyes 방법에 기반을 둔 렌더링 모듈 개발 ▷ 기능: geometric transformations, hidden-surface elimination, lighting and shading, rasterization, anti-aliasing based on stochastic samping, texture mapping, shadow, radiosity (일부) ▷ Graphics primitives: polygonal models, quadrics, parametric patches (일부) ▷ 광선 추적법 모듈과의 호환성 유지 ▷ RenderMan Shading Language 기능 지원 ·렌더링 시스템 GUI와의 연결	98
2	분산 및 병렬 렌더링 기법 도입	·통신 툴킷 PVM 및 영상공간분할에 기반을 둔 통신망상의 워크스테이션을 위한 분산 Z-버퍼 렌더링 기법 개발 ·병렬 컴퓨터 Cray T3E 상에서의 병렬 렌더링 기법 구현	95
3	렌더링 시스템 GUI 기능 개선 및 타 시스템과의 인터페이스 개발	·GUI 기능 개선 및 확장 ▷ navigation 기능, texture 설정 기능, 광원 설정 기능, 다중사용자 처리 기능, 기타 ·VRML2.0 파일 입력 기능 추가 ·RIB 파일 입출력 기능 추가	100

번호	세부연구목표 (연구계획서상에 기술된 연구목표)	달성내용	달성도 (%)
4	시스템 평가를 통한 완성도 제고	·타 시스템과의 비교 분석 ·실제 작업을 통한 문제점 파악 및 해결 ·사용에 생성을 통한 간단한 tutorial 제작	100
5	렌더링 시스템으로의 가상환경 개념의 도입	·다중 사용자간의 협력작업을 지원하는 다중서버 구조 네트워크 툴킷 개발 ·다중 사용자간의 3D scene 공유를 위한 기법 개발 ·네트워크 툴킷의 렌더링 시스템으로의 응용	100
6	실시간 렌더링을 위한 영상 기반 렌더링 기법 연구	·구를 기반으로 하는 영상기반렌더링 기법 개발 ·2D Wavelets을 이용한 light fields의 압축 기법 개발 ·3D Wavelets을 이용한 light fields의압축 기법 개발	100
7	방대한 볼륨 데이터의 효과적인 렌더링 기법 연구	·빠른 복원을 지원하는 3D Wavelets을 이용한 3D 의료 데이터의 압축 기법 개발 ·압축된 데이터로부터의 빠른 렌더링 기법 개발 ·Virtual human body navigation system 개발으로의 응용	100

제 5 장 연구개발결과의 활용계획

본 연구에서는 일반적인 상황을 가정한 렌더링 시스템의 프로토타입을 개발하였는데, 이를 통하여 축적한 요소기술들은 추후 가상현실 소프트웨어, 3D CAD 시스템, 애니메이션 소프트웨어 등 관련 소프트웨어 제작시 렌더링 모듈 개발에 유용하게 쓰일 수가 있을 것이다. 예를 들어 VRML 기반 가상환경 시스템 제작에 있어 본 연구에서 개발한 프로토타입 시스템의 코드를 적절히 사용하므로써, 적은 비용으로 렌더링 관련 부분을 개발할 수 있을 것이다. 또한 본 연구에서 개발한 wavelets을 이용한 방대한 의료 데이터의 압축 기법은 빠른 런-타임 복원을 지원하는 새로운 개념의 압축 기법으로서, 가상의 3차원 인체 네비게이션 시스템등 의료분야의 소프트웨어 개발에 유용하게 쓰일 수 있을 것이다. 3단계 연구가 수행될 경우 연구 목표로 설정될 구체적인 상황에 맞는 렌더링 소프트웨어로 발전될 수 있을 것이다

본 과제에서 개발한 렌더링 시스템은 기존의 상용화된 소프트웨어의 기능을 포함하고, 관련 응용 소프트웨어 상품의 개발을 용이하게 하는 효율적인 도구를 제공하고 있다. 이들의 상품화는 특정한 기업에 의존하기 보다는 컴퓨터 그래픽스 컨소시엄을 통하여 여러 기업의 공동개발을 유도함으로써 기반 기술의 저변확대를 추구할 수 있을 것이다. 이는 다른 컴퓨터 그래픽스 관련 소프트웨어 개발을 위한 기술을 축적하게 함으로써, 향후에 응용 소프트웨어 상품의 개발을 유도하게 될 것이다.

여 백

제 6 장 참고문헌

[1-1] J.D.Foley, A. Dam, S.K. Feiner, and J.F.Hughes., *Computer Graphics Principles and Practice*, ADDISON WESLEY, 1992.

[1-2] Robert L. Cook, Loren Carpenter, and Edwin Catmull, "The Reyes Image Rendering Architecture", *ACM Computer Graphics*, Volume 21(Number 4):95-102, 1987.

[1-3] P.S. Hechbert., *Graphics Gems IV*, AP PROFESSIONAL, 1994.

[1-4] J.D. Foley, A. Dam, S.K. Feiner, and J.F. Hughes., *Introduction to Computer Graphics*, ADDISON WESLEY, 1994.

[1-5] A. Watt and M. Watt., *Advanced Animation and Rendering Techniques*, ADDISON WESLEY, New York, 1993.

[1-6] Pixar, *The RenderMan Interface Version 3.1*, 1989.

[1-7] P.S. Hechbert., *Graphics Gems III*, AP PROFESSIONAL, 1994.

[1-8] William T. Reeves, David H. Salein, and Robert L. Cook, "Rendering Antialiased Shadows with Depth Maps", *ACM Computer Graphics*, Volume 21(Number 4):283-291, 1987.

[1-9] T. Duff., "Compositing 3-D Rendered Images," In *Computer Graphics (SIGGRAPH '85)*, Vol. 19, No. 3, 1985, pp. 41-44.

[1-10] T. Porter and T. Duff., "Compositing Digital Images," In *Computer Graphics(SIGGRAPH '84)*, Vol. 18, No. 3, 1984, pp. 49-55.

[1-11] Microsoft., *SOFTIMAGE 3D A Comprehensive User's Guide Reference*, Microsoft Corporation, 1995.

[1-12] Silicon Graphics., *Alias Wavefront Learning Alias V8*, Silicon Graphics Limited, 1996.

[1-13] Lee. M. E., "Development of a Ray Tracing Model for Realistic Image Synthesis," *Master's Thesis, University of Tulsa*, Tulsa, Oklahoma, 1986.

[1-14] Hourcade J., and Nicolas A., "Algorithms for Anti-Aliased Cast Shadows," *Computer and Graphics*, Vol. 9, No. 3, 1985, pp. 259-265.

[1-15] Williams L., "Pyramidal Parametrics," *Computer Graphics(SIGGRAPH '83)*, Vol. 17, No. 3, 1983.

[1-16] Franklin C. Crow., "Summed-area Tables for Texture Mapping," *Computer Graphics(SIGGRAPH '84)*, Vol. 18, No. 3, 1984, pp. 207-212.

[1-17] Bui-T. Phong., "Illumination for Computer Generated Pictures," *Communications of the ACM*, Vol. 18, No. 6, June. 1975, pp. 311-317.

[1-18] Won-yong Jung. *Development of Advanced renderer Using Z buffer through Distributed Computing*. MS thesis, Sogang University, 1996

[1-19] S. Wihitman. A Task Adaptive Parallel Graphics Renderer. *ACM SIGGRAPH Symposium on Parallel Rendering*, pages 27-34, November 1993.

[1-20] T. W. Crockett and T. Orloff, A MIMD Rendering Algorithm for Distributed Memory Architectures. *ACM SIGGRAPH Symposium on Parallel Rendering*, pages 35-42, November 1993.

[1-21] *Toy Story a milestone in 3d animation*, 1995.

[1-22] 정원영, "분산 환경하의 Z 버퍼 방법에 기반을 둔 고급 렌더러의 개발", 서강대학교 석사학위 연구, 1996.

[1-23] A. Belguelin, J. Dongarra, G. Geist, W. Jiang, R. Manchek, B. Moore, and V. Sunde, *PVM 3.2: Parallel Virtual Machine System 3.2 Technical Report*, University of Tennessee, Knoxville TN, Oak Ridge National Laboratory, Oak Ridge TN, Emory University, Atlanta GA, 1992.

[1-24] S. Green. *Parallel Processing for Computer Graphics*, pages 52-54. The MIT Press, 1991.

[1-25] 정효성, *"Reyes 구조 기반 렌더러상에서의 렌더링 옵션들의 상호 연관성 분석 및 가중치 테이블 설계"*, 서강대학교 석사학위 연구, 1997.

[2-1] Pixar, *The RenderMan Interface Version 3.1*, 1989

[2-2] Andrea L. Ames, David R. Nadeau and John L., *VRML 2.0 Source Book*, John Wiley & Sons, Inc., 1997

[2-3] Jamdes Arvo, *Graphics GEMS II*, AP PROFESSIONAL, 1994

[2-4] 임인성, *그래픽 및 렌더링 툴킷 개발*, 연구보고서, 과학기술처, 1996

[3-1] Steven Richard. *UNIX Network Programming*, Prentice-Hall, 1990.

[3-2] C. Shaw and M. Green. The MR toolkit peers package and experiment. In *IEEE Virtual Reality Annual International Symposium (VRAIS 93)*, pages 463-469, IEEE Computer Society Press, September, 1993.

[3-3] Gurminder Singh, Luis Serra, Willie Png and Hern Ng. BRICKNET : sharing object behaviours on the net. In *Proceedings of the IEEE Virtual Reality Annual International Symposium (VRAIS'95)*, IEEE

Computer Society Press, pages 19-25, March, 1995.

[3-4] M. Macedonia and M. Zyda. A taxonomy for networked virtual environments. *IEEE Multimedia*, Vol. 4, Num. 1, pages 48-56, 1997.

[3-5] O. Hagsand. Interactive Multiuser VEs in the DIVE environment. *IEEE Multimedia*, Vol. 3, Num. 1, pages 30-39, 1996.

[3-6] C. Greenhalgh and S. Benford. MASSIVE: a collaborative virtual environment for teleconferencing, *ACM Transactions on Computer-Human Interaction*, Vol. 2, No. 3, pages 239-261, September, 1995.

[3-7] Michael Macedonia, Michael Zyda, David Pratt, Paul Barham and Steven Zeswitz. NPSNET: a network software architecture for large-scale virtual environments. *Presence*, Vol. 3, Num. 4, pages 265-287, Fall, 1994.

[3-8] John Locke. *An Introduction to the Internet Networking Environment and SIMNET/DIS.*

[3-9] G. Singh, L. Serra, K. Fairchild and T. Poston. Visual Creation of Virtual Design Environments and Virtual Worlds Research at ISS. *Presence*, Vol.3, Num. 1, pages 94-107, 1994.

[3-10] T. Whitted. An improved illumination model for shaded display, *Comm. ACM*, Vol. 26, Num. 6, pages 342-349, 1980.

[3-11] Steve Upstil. *The RenderMan Companion*, Addison Wesley, 1998.

[3-12] Mason Woo, Jackie Neider and Tom Davis. *OpenGL Programming Guide : The Official Guide to Learning OpenGL, Version 1.1*, Addison Wesley, release 2, 1997.

[3-13] OpenGL Architecture Review Board. *OpenGL Reference Manual: The*

Official Reference Document to OpenGL, Version 1.1, Addison Wesley, release 2, 1997.

[3-14] Robert Cook, Loren Carpenter, and Edwin Catmull. The Reyes image rendering architecture, *ACM Computer Graphics*, Vol. 21, Num. 4, pages 95-102, 1987.

[4-1] E. H. Adelson and J. R. Bergen. The plenoptic function and the elements of early vision. In M. Landy and J. A. Movshon, editors, *Computational Models of Visual Processing*, chapter 1, The MIT Press, Cambridge, Mass., 1991.

[4-2] T. Beler and S. Neely, Feature-based image metamorphosis, In *Computer Graphics (SIGGRAPH '92 Proceedings)*, pages 35-42, 1992.

[4-3] D. Berman, J. Bartell, and D. Salesin, Multiresolution painting and compositing, In *Computer Graphics (SIGGRAPH '94 Proceedings)*, pages 85-90, 1994.

[4-4] J. F. Blinn and M. E. Newell, Texture and reflection in computer generated images, *CACM*, 19(10):542-547, October 1976.

[4-5] E. E. Catmull, *A Subdivision Algorithm for Computer Display of Curved Surfaces*, PhD thesis, University of Utah, Salt Lake City, Utah, December 1974.

[4-6] S. E. Chen, Quicktime VR - an image-based approach to virtual environment navigation, In *Computer Graphics (SIGGRAPH '95 Proceedings)*, pages 29-38, 1995.

[4-7] S. E. Chen and L. Williams, View interpolation for image synthesis,

In *Computer Graphics (SIGGRAPH '93 Proceedings)*, pages 279-288, 1993.

[4-8] C. K. Chui, *An Introduction to Wavelets*, Academic Press Inc., 1992.

[4-9] P. E. Debevec, C. J. Taylor, and J. Malik, Modeling and rendering architecture form photographs: a hybrid geometry and image-based approach, In *Computer Graphics (SIGGRAPH '96 Proceedings)*, pages 11-20, 1996.

[4-10] R. DeVore, B. Jawerth, and B. Laucier, Image compression through wavelet transform coding, *IEEE Transaction on Information Theory*, 38(2):719-746, March 1992.

[4-11] G. Fekete, Rendering and managing spherical data with sphere quadtrees, In *Proceedings of Visualization '90*, pages 176-186, San Francisco, 1990.

[4-12] S. Gortler, R. Grzeszczuk, R. Szeliski, and M. Cohen, The Lumigraph, In *Computer Graphics (SIGGRAPH '96 Proceedings)*, pages 43-54, 1996.

[4-13] N. Greence, Environment mapping and other applications of world projections, *IEEE CG & A*, 6(11):21-29, November 1986.

[4-14] M. Lovoy, Efficient ray tracing of volume data, *ACM Transactions on Graphics*, 9(3):245-261, July 1990.

[4-15] M. Levoy and P. Hanrahan, Light field rendering, In *Computer Graphics (SIGGRAPH '96 Proceedings)*, pages 31-42, 1996.

[4-16] P. Lux, A novel set of closed orthogonal functions picture coding, Arch. Elek. Übertragung, 31:267-274, 1997.

[4-17] L. McMillan and G. Bishop, Plenoptic modeling: An image-based

rendering system, In *Computer Graphics (SIGGRAPH '95 Proceedings)*, pages 39-46, 1995.

[4-18] B. O'Neill, *Elementary Differential Geometry,* Academic Press, 1996.

[4-19] P. Schröder and W. Sweldens, Spherical wavelets: Efficiently representing functions on the sphere, In *Computer Graphics (SIGGRAPH '95 Proceedings)*, pages 161-172, 1995.

[4-20] S. M. Seitz and C. R. Dyer, View morphing, In *Computer Graphics (SIGGRAPH '96 Proceedings)*, pages 21-30, 1996.

[4-21] J. M. Shapiro, Embedded image coding using zerotrees of wavelet coeffcients, *IEEE Transactions on Signal Processing,* 41(12):3445-3462, December 1993.

[4-22] E. Stollnitz, T. DeRose, and D. Salesin, *Wavelets for Computer Graphics: Theory and Applications,* Morgan Kaufmann Publishers, Inc., 1996.

[4-23] A. Zandi, J. D. Allen, E. L. Schwartz, and M. Boliek, CREW: Compression with reversible embedded wavelets, In *Proc. of Data Compression Conference*, pages 212-221, Snowbird, Utah, March 1995. IEEE

[5-1] C. K. Chui. *An Introduction to Wavelets.* Academic Press Inc., 1992.

[5-2] I. Daubechies. *Ten Lectures on Wavelets.* SIAM, 1992.

[5-3] A. Fournier, editor. *Wavelets and Their Applications in Computer Graphics.* ACM SIGGRAPH '95 Course Notes, 1995.

[5-4] M. H. Ghavamnia and X. D. Yang. Direct rendering of Laplacian

pyramid compressed volume data. In *Proceedings of IEEE Visualization '95*, pages 192-199, Atlanta, October 1995.

[5-5] R. C. Gonzalez and R. E. Woods. *Digital Image Processing*. Addison Wesley Pub. Comp., 1993.

[5-6] M. Grob. *Visual Computing*, Springer-Verlag, 1994.

[5-7] L. Hong, S. Muraki, A. Kaufman, D.bartz, and T. He. Virtual voyage: interactive navigation in the human colon. In *Computer Graphics (ACM SIGGRAPH '97 Proceedings)*, pages 27-34, LosAngeles, August 1997.

[5-8] I. Ihm and S. Park. Wavelet-based 3D compression scheme for very large volume data. In *Proceedings of Graphics Interface '98*, pages 107-116, Vancouver, June 1998.

[5-9] I. Ihm and S. Park. Wavelet-based 3D compression scheme for interactive visualization of very large volume data, *Computer Graphics Forum*, 1998. (Selected for possible publication)

[5-10] A. Kaufman, editor. *Introduction to Volume Visualization*. IEEE Computer Society Press, 1991.

[5-11] S. Muraki. Approximation and rendering of volume data using wavelet transforms. In *Proceedings of IEEE Visualization '92*, pages 21-28, Boston, October 1992.

[5-12] S. Muraki. Volume data and wavelet transforms. *IEEE Computer Graphics and Applications*, 13(4):50-56, 1993.

[5-13] G. M. Nielson and B. Shriver, editors. *Visualization in Scientific Computing*. IEEE Computer Society Press, 1990.

[5-14] P. Ning and L. Hesselink. Fast volume rendering of compressed

data. In *Proceedings of IEEE Visualization '93*, pages 11-18, San Jose, October 1993.

[5-15] NLM. *http://www.nlm.nih.gov/research/visible/visible_human.html*, 1997.

[5-16] NLM. *Electronic Imaging: Report of the Board of Regents*. National Institute of Health, 1995.

[5-17] S. Park, G. Koo, and I. Ihm. Wavelet-based 3D compression schemes for the Visible Human Dataset and Their Applications, *Visible Human Project Conference '98*, Bethesda, Maryland, October 1998. (To appear)

[5-18] T. M. Rhyne, editor. *Visualizing and examining large scientific data sets: a focus on the physical and natural sciences*. ACM SIGGRAPH '94 Course Notes, 1994.

[5-19] L. Rosenblum, editor. *Scientific Visualization: Advances and Challenges*. IEEE Computer Society Press, 1994.

[5-20] K. Sayoood. *Introduction to Data Compression*. Morgan Kaufmann Publishers Inc., 1996.

[5-21] P. Schroder and W. Sweldens, editors. *Wavelets in Computer Graphics*. ACM SIGGRAPH '96 Course Notes, 1996.

[5-22] J. M. Shapiro. Embedded image coding using zerotrees of wavelets coefficients. *IEEE Transactions on Signal Processing*, 41(12):3445-3462, December 1993.

[5-23] E. Stollnitz, T. DeRose, and D. Salesin. *Wavelets for Computer Graphics: Theory and Applications*. Morgan Kaufmann Publishers Inc., 1996.

[5-24] G. R. Thomas and L. R. Long. Compressing and transmitting Visible Human images. *IEEE Multimedia*, 4(2):36-45, April-June 1997.

[5-25] U. Tiede, T. Schiemann, and K. H. Hohne. Visualizing the visible human. IEEE Computer Graphics and Applications, 16(1):7-9, 1996.

[5-26] 박상훈, 임인성, 방대한 볼륨 데이터의 효과적인 압축 기법, *한국정보과학회 춘계학술대회 학회지*, 359-361쪽, 대전, 1998년 4월.

[5-27] 박상훈, 임인성, Visible Human 데이터를 위한 볼륨 탐사 시스템 SGVN의 개발, *한국정보과학회 춘계학술대회 학회지*, 362-364쪽, 대전, 1998년 4월.

[6-1] C. Shaw and M. Green, "The MR Toolkit Peers Package and Experiment," *Proceedings of IEEE VR An. int'l Symp.(VRAIS 93)*, September 1993, pp. 463-469.

[6-2] Michael R. Macedoina, Michael J. Zyda, David R. Pratt, Paul T. Barham and Steven Zeswitz, "NPSNET: A Network Software Architecture for Large Scale Virtual Environments," *Presence*, Vol. 3, No. 4, Fall 1994.

[6-3] James Calvin, Alan Dickens, Bob Gaines, Paul Metzger, Dale Miller and Dan Owen, "The SIMNET Virtual World Architecture," *Proceedings of IEEE VRAIS*, September 1993, pp. 450-455

[6-4] Olof Hagsand, "Interactive Multiuser VEs in the DIVE," *IEEE MultiMedia*, Spring 1996, pp. 30-39.

[6-5] Thomas A. Funkhouser, "RING: A Client-Server System for Multi-User Virtual Environments," *ACM Siggraph Special Issue on 1995 Symp. on*

Interactive 3D Graphics, 1995, pp. 85-92.

[6-7] Rick Kazman, "Making WAVES: On Design of Architectures for Low-end Distributed Virtual Environments," *Proceedings of IEEE VRAIS,* September 1993, pp. 443-449.

[6-8] G. Singh et al., "BrickNet: A Software Toolkit for Network based Virtual Worlds," *Presence,* Vol. 3, No. 1, Winter 1994, pp.94-107.

[6-9] 김성희, 이희웅, "3차원 모델링과 렌더링을 위한 객체 지향 그래픽스 툴킷," *컴퓨터 그래픽스 학회지,* Vol. 2, No. 1, June 1996. pp. 61-67.