3차원 게임 가상환경 생성 및 제어

이재문[0] 장현덕 이용덕 이명원*
수원대학교 컴퓨터학과
*수원대학교 인터넷정보공학과
breakdance@lycos.co.kr, mwlee@suwon.ac.kr

Generation and Control of a 3-dimensional Game Virtual Environment

Jae Moon Lee[0], Hyun Duck Jang, Yong Duck Lee, and Myeong Won Lee
Dept. of Computer, The U. of Suwon
Dept. of Internet Information Engineering, The U. of Suwon

요 약

본 논문에서는 인터넷 게임 개발에 있어서 배경 화면을 구성하는 3차원 가상환경 생성 및 가상환경을 돌아다니면서 주변을 관찰할 수 있게 하는 카메라 처리 기능을 포함한 가상환경 제어 방법에 대해 기술한다. 여기에 포함되는 기술로서 3차원 지형 메쉬 데이터 처리, 맵에디터, 장면 처리, 캐릭터의 충돌 처리 및 게임 제어 방법 등을 소개한다. 그리고, 시맨틱 가상환경 구현을 목적으로 가상환경상의 각 위치에 대해 의미를 부여할 수 있도록 현재 위치에 대한 정보 입력을 가능하게 해주는 위치 기반 동적 정보 입력기에 대해서도 설명한다.

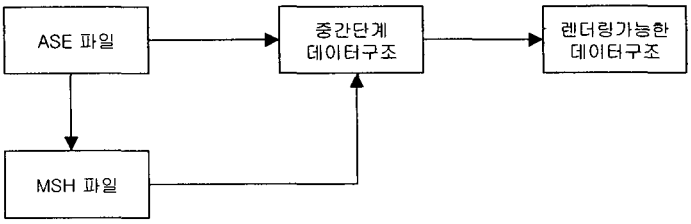
1. 서론

게임 프로그램에서 유저가 실제 게임과 접하게 되는 인터페이스인 클라이언트 어플리케이션 프로그램은 게임 로직 모듈과 게임 엔진 모듈로 구성된다. 게임 로직 모듈은 게임 시나리오에 따라 게임을 진행하는 로직이 들어 있는 모듈로, 게임 내에 존재하는 캐릭터, 주변 사물, 지형, 건물 등의 객체가 어떻게 그려질지에 관해서는 관여하지 않고 어떤 객체가 그려질 것인가에 대해서만 관여한다. 다시 말해서 게임의 전체 흐름을 통제하는 모듈이다. 따라서 이 모듈은 하드웨어, OS 혹은 API에 종속되지 않는 모듈에 해당한다. 이렇게 종속되지 않게 하기 위해서는 OS나 API에 관한 인터페이스를 제공해 주어야 하는데, 이것이 게임 엔진 모듈이다[1].

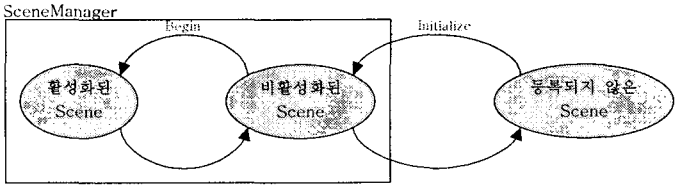
게임 엔진 모듈은 게임 로직에서 나타나는 모든 객체들의 행위 및 표현에 필요한 모든 작업을 담당하게 된다. 게임 엔진 모듈에는 세부적으로 다시 여러 가지 모듈로 구성되어 있는데, 렌더링 라이브러리, 지형관리, 객체 관리, 리소스 관리, 애플리케이션 프레임워크, 네트워크 모듈로 구성되어 서로 유기적으로 작동하면서 게임에 필요한 기능을 제공한다[2].

게임 애플리케이션에서 고려해야 할 사항 중 하나로 게임 인터페이스 부분도 고려해야 한다. 게임 개발에 있어서 유저가 가상공간으로 몰입할 수 있도록 사용하기 편리한 인터페이스를 제공해야 한다. 본 연구에서는 일반적인 게임에서와 같이 키보드와 마우스를 사용하여 진행되도록 하고, 마우스의 이동으로 캐릭터의 방향을 설정하도록 한다. 키보드를 사용해서 캐릭터를 움직일 수 있으며, 게임 도중에 여러 가지 설정을 가능하게 해준다.

본 연구에서는 위와 같은 게임 어플리케이션 개발에 있어서 공통적으로 필요한 기능으로서 3차원 가상 환경의 생성 및 제어 방법, 그리고 게임 진행상의 제어를 위한 콘솔 명령어의 구성에 대해 기술한다. 또한, 시맨틱 가상환경 구현을 위한 한 방법으로서 위치 기반 환경 정보 입력기에 대해서도 소개한다[3].



(그림 1) 메쉬 데이터 생성 과정



(그림 2) Scene 의 생명 주기

2. 메쉬 데이터 관리

게임에서 캐릭터나 가상환경을 생성할 때는 보통 3DS MAX와 같은 모델러를 이용하게 된다. 3DS MAX에서는 ASE(Ascii Scene Export) Exporter를 제공하기 때문에 메쉬 정보를 텍스트 파일로 얻을 수 있다. 이와 같은 텍스트로 표현되는 메쉬 정보는 다루기 쉽고, 수정이 용이하다. 그런데, 이 파일에는 응용에 따라 직접 사용하지 않는 정보가 많이 포함되어 있고, 파일 크기가 매우 크다는 단점이 있다. 예를 들어, 본 연구에서 생성한 가상환경의 일부인 경복궁을 모델링한 메쉬 정보 파일은 약 12메가 바이트정도 차지하였다. 이렇게 큰 크기의 메쉬 정보 파일은 게임을 실행할 때 문제가 된다. 게임을 실행할 때마다 이 파일을 로드하여 파싱하는 과정을 거쳐 그래픽스 라이브러리에서 사용할 수 있는 데이터 타입으로 재구성하는데에는 많은 시간이 걸리게 된다.

그래서, 본 연구에서는 MSH 라는 고유의 바이너리 파일 포맷을 고안하여 텍스트 파일보다 작은 크기를 가지며, 메쉬 변환에서도 상당한 성능 향상을 얻게 되었다. 메쉬 정보를 로드하는 과정 중에서 파싱하는 작업이 가장 오래 걸리기 때문에 이 과정을 생략하게 된다. 본 연구의 메쉬 관리 모듈에서는 ASE파일을 MSH파일로 변환해주는 별도의 애플리케이션을 제공하여 ASE파일은 쉽게 MSH로 변환하여 클라이언트 애플리케이션에서 사용할 수 있도록 해준다.

3. 장면 관리

게임에서는 로고 화면, 인트로동영상, 메인 메뉴, 게임 진행 화면 등 여러 장면이 존재하게 된다. 이러한 각 장면들은 서로 독립적이기 때문에 장면별로 관리되는 로직이 필요하다. 이러한 장면들은 하나의 장면에서 다른 장면으로 전환되기도 하고, 장면이 시작할 때 처리해 주어야 할 작업과 장면이 끝날 때 처리해 주어야 할 작업이 존재하게 된다. 이러한 장면의 생명주기를 프레임워크에서 관리하게 된다.

장면 관리 프레임워크에는 SceneManager 및 Scene의 2개 클래스를 제공한다. 구현할 내용은 Scene 클래스를 상속받아 5개의 가상 함수 Initialize, Dispose, Begin, End, DoFrame을 재정의하여 SceneManager에 등록한다. 이렇게 하여 SceneManager는 장면을 관리하게 되며, (그림 2)의 생명 주기에 따라 함수를 호출하게 된다. SceneManager는 등록된 장면들을 정수형의 아이디로 구분하며 특정 장면을 활성화 하거나 제거할 때 이 아이디를 사용한다.

장면에는 두 가지 종류의 장면이 존재한다. 하나는 등록되었지만 활성화되지 않는 장면이고, 다른 하나는 등록되어 있는 장면 중 활성화되어 있는 장면이다. 하나의 SceneManager에는 여러 개의 장면이 등록될 수 있지만 활성화될 수 있는 장면은 하나이다. 따라서 현재 활성화되어 있는 장면이 있는 상태에서 다른 장면을 활성화하려고 한다면 현재 활성화되어 있는 장면은 비활성화된 장면으로 변하게 된다.

3. 3차원 지형 메쉬 구성

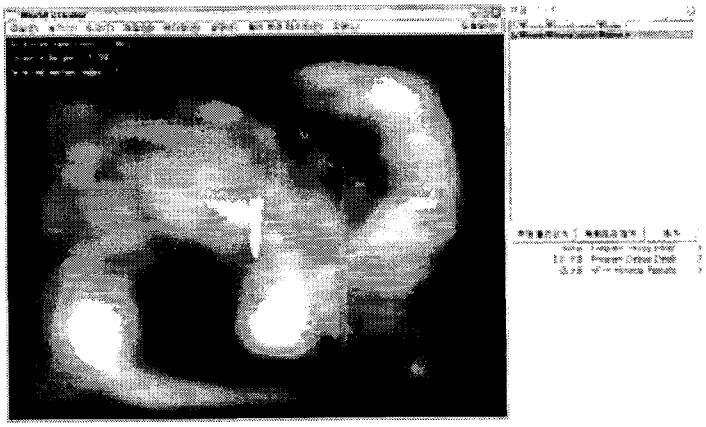
가상 환경을 구성하는 구성요소에는 건물, 지물 그리고 지형이 존재한다. 건물과 지물은 외부 툴에서 저작되어 메시 관리 라이브러리에 의해서 관리된다[4]. 지형에 관해서는 따로 저작되지 않고, 특수한 방법을 사용되어 렌더링된다. 지형에 관한 최적화 방법에는 여러가지가 있지만 이 모듈에는 ROAM(Realtime Optimally Adapting Meshes) 방식을 사용하여 렌더링 된다[5].
ROAM은 지형을 표현하는데 있어서 LOD(Level Of Detail)을 구현하는 알고리즘 중 하나이다. LOD란 메쉬를 표현할 때, 멀리 떨어진 메쉬는 여러 개의 폴리곤이 하나의 픽셀로 렌더링되기 때문에 폴리곤 수를 줄여서 렌더링을 해도 화질에 차이가 없음을 기본적인 아이디어로 출발한다. 따라서 멀리 떨어진 메쉬는 폴리곤 수를 줄여서 렌더링 하여 그래픽스 파이프라인에 전달되는 폴리곤 수를 줄인다. 이렇게 되면 계산해야 할 폴리곤이 줄기 때문에 렌더링 속도가 빨라진다.

ROAM에서는 동적으로 폴리곤 수를 조절하는 기능을 제공한다. 메쉬를 삼각형으로 나누는 과정을 테셀레이션(Tessellation)이라 부르는데, ROAM에서는 매 프레임마다 렌더링해야 할 폴리곤을 테셀레이션하게 된다. ROAM은 일정 영역으로 구분되는 패치단위로 테셀레이션을 하는데, 각각의 패치에서 테셀레이션 레벨이 다르면 메쉬가 깨져버리는 문제가 발생한다. ROAM에서는 모든 삼각형에 대한 이웃한 삼각형을 가지고 있어서 테셀레이션할 때 이웃한 삼각형도 테셀레이션함으로서 이러한 문제를 해결한다.

4. 맵 에디터

캐릭터나 배경 등의 가상환경이 제작된 후에는 연속적으로 가상환경과 캐릭터를 이용하여 게임 컨텐츠를 제공해야 하는데, 마야나 3DS MAX와 같은 범용적인 툴로 제작한다는 것은 매우 힘든 일이다. 캐릭터가 가상환경상에서 게임 시나리오에 따라 움직이도록 해주고 이 때 발생할 충돌 처리를 사전에 조정해주는 역할을 하는 맵 에디터가 필요하게 된다. 이 도구는 맵 디자이너가 가상환경을 게임 내용에 맞추어 쉽게 저작하는데 목표를 두고 있으며, 맵 디자이너는 이 도구를 이용하여 다음의 두 가지 종류의 공간을 저작할 수 있다.

하나는 실내 공간을 저작할 수 있는데 실내 공간을 저작할 때는 범용 툴로 저작된 내부 공간을 읽어 들여서 게임 엔진에 최적화된 데이터 구조로 실내 공간을 저장하는 기능을 갖는다. 또한 실내 공간 내에 정적으로 위치하는 객체를 배치하는 기능을 갖는다. 또한 실내 공간의 BSP (Binary Space Partition)[4] 와 PVS (Potentially Visible Set)[5]을 생성하여 엔진이 더 효율적으로 실내 공간을 렌더링 할 수 있게 한다. 또 하나의 다른 기능은 실외 공간을 저작하는 기능인데, 지형의 높낮이와 지형 타일의 텍스춰를 설정하고, 랜덤 지형 생성 및 raw 비트맵 파일을 통해 초기지형을 생성할 수 있다. 실내 공간 저작과 마찬가지로 실외 공간에 존재하는 정적인 객체를 배치시킬 수 있다. 또한 방대한 크기의 실외 공간을 표현하고, 관리하기 위해서 ROAM 기법을 사용한다. (그림 3)은 구현된 맵 에디터를 보여준다.



(그림 3) 맵 에디터

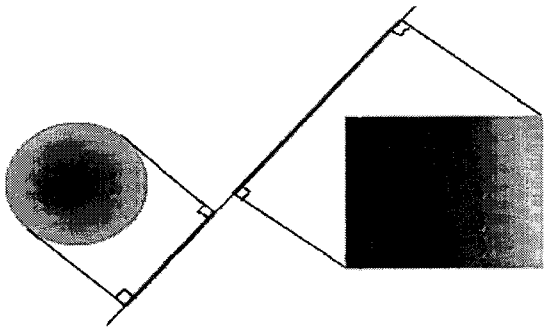
5. 충돌 처리

게임 진행에는 물체간의 충돌이 발생하였을 경우 처리하는 방법이 제공되어야 한다. 이러한 처리를 위해서 충돌 검사/검출(Collision Detection)이 필요하다.

모든 메쉬들은 3각형을 기본 도형으로 하여 렌더링되어 있어서 충돌 검사 역시 3각형 단위의 검사를 수행한다. 일반적인 가상 환경에서 삼각형의 개수는 수만 개가 넘기 때문에 이 모든 삼각형에 대해 검사를 수행한다는

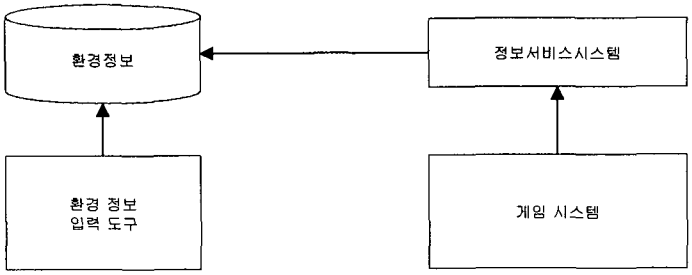
것은 불가능하다. 따라서 가상 환경의 영역을 나누어 충돌할 것 같지 않은 공간의 삼각형들을 충돌 검사에서 제외하는 방법을 사용해야 한다. 이를 위해 본 연구에서는 옥트리(Octree)를 사용하였다.

그리고, 객체의 모든 삼각형에 대해서 충돌을 검사하는 방법은 효율적이지 않아서, 메쉬를 전부 포함하는 AABB(Axis Aligned Bounding Box, 축에 정렬된 경계 상자로 가상 환경 좌표계의 XYZ축과 평행한 상자)를 정의하여 메쉬의 각 삼각형을 검사하는 대신에 이 경계 상자를 검사함으로서 속도를 향상시키도록 하였다[6].

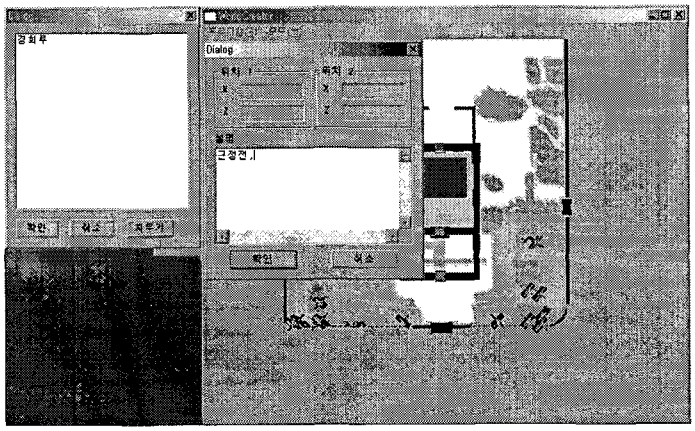


(그림 4) 분리축(Separated axis)

상자와 삼각형의 충돌 검사를 하는 알고리즘 역시 많지만 분리축(Separated Axis)를 사용하여 박스와 삼각형이 교차하는지 검사한다. 분리축이란 두 프리미티브를 분리하는 축을 의미한다(그림 4). 예를 들어 두 프리미티브가 떨어져 있다면 어떠한 축에 투영 시켰을 때 축에 투영된 두 프리미티브가 겹치지 않는다. 이러한 축을 분리축이라 부르며, 분리축 이론을 사용하여 교차 검사함은 이러한 분리축을 찾는 작업이다. 만일 이런 분리축이 존재한다면 두 프리미티브는 교차하지 않는다고 판정이 되며, 어떠한 분리축도 발견되지 않으면 두 프리미티브는 교차한다고 판정이 된다. 충돌 반응 처리에 있어서는, 충돌이 발생했다고 판정이 되면 아바타가 더 이상 그 방향으로 이동을 하지 않도록 하였다.



(그림 5) 환경 정보와 게임 시스템



(그림 6) 환경 정보 입력기

6. 위치 기반 환경 정보 입력

기존의 많은 게임에서의 가상환경은 등장하는 캐릭터와의 관계에 있어서 가상환경을 캐릭터와는 독립적으로 모델링과 렌더링 표현에만 중점을 두어 개발되었다. 이에 비해 본 연구에서의 가상환경은 지리적 의미를 갖고 객체 상호간의 관계나 위치 기반의 환경 정보를 보유할 수 있도록 하였다.

가상환경에서의 지리적 위치와 관련 정보를 동기화시키기 위해서는 가상환경 상에서 캐릭터가 지나가는 위치에 해당 정보를 입력할 수 있는 기능이 필요하다. 이와 같은 정보서비스가 가능하기 위해서는 환경 데이터 구성에 있어서 실세계 지형의 사실적 표현도 중요하지만 지형의 물리적 특성 및 역사적 정보의 시각적 표현과 관리도 중요한 요소가 된다. 본 연구에서는 이와 관련하여 게임 내 가상환경에서 캐릭터가 지나가는 임의의 위치에

정보를 입력할 수 있는 인터페이스를 추가하였다. 그림 5는 게임 프로그램과 정보서비스시스템간의 관계를 보여주며 그림 6은 본 연구에서 개발한 환경 정보 입력기이다.

(표 1) 게임 제어 콘솔 명령어

명령어 종류	명령어 인자	기능
set	key_delay	콘솔 모드에서 키 입력의 지연 시간
	console_show	콘솔 모드를 보일지 말지 지정
	rs_fill	렌더링 모드 지정
	g_gravity	게임 내 중력 크기 지정
	info	정보 파일을 읽어옴
	play_cam_record	저장된 카메라 정보를 따라서 뷰를 이동
	record_cam	게임내 카메라 정보 입력을 활성화
	play_cam_speed	뷰가 이동하는 속도 지정
	info	맵 정보 파일을 읽어옴
write	cam	카메라 정보를 파일 시스템에 저장
read	cam	파일시스템에 저장된 카메라 정보를 읽어옴
exec	파일경로	명령어들이 저장된 배치 파일을 읽어와 실행시킴
exit	없음	프로그램 종료
clear	없음	콘솔 화면 지움
	cam_records	저장된 카메라 정보를 지움

7. 게임 제어 인터페이스

게임 시스템에 어떠한 작업을 수행하거나 속성을 변경하는 데에는 하나의 클래스가 수행하는 Facade 패턴을 사용한다. 게임의 실행시간에도 이러한 기능을 제공하기 위해 Facade 클래스에 접근할 수 있는 인터페이스를 제공해야 한다. 이러한 인터페이스가 콘솔 라이브러리이다. 이 콘솔을 사용자에게 시스템에 접근할 수 있는 CUI (Character User Interface)를 제공한다.

다시 말해 콘솔모드는 게임의 여러 기능을 실행 시간에 실행할 수 있도록 구성하는 사용자 인터페이스이다. 기능적인 면에서 본다면 게임을 실행하는 도중에 기능을 사용자가 명령어를 통해서 실행할 수 있다는 장점이 있고, 시스템 관점에서 본다면 게임에 사용되는 기능을 일괄적으로 실행할 수 있는 창구 역할을 한다. 게임에 관한 여러 설정을 하드 코딩하지 않고, 외부 배치 파일로 따로 빼놓음으로서 설정을 좀 더 유연하게 바꿀 수 있다. 현재 사용 가능한 콘솔 명령어는 표 1과 같다.

표 2는 본 연구에서의 게임 가상환경 제어 프로그램에서 정의하는 클래스 종류들이다. 그림 7은 게임 실행 장면들을 보여준다.

(표 2) 게임 프로그래밍 클래스

클래스 종류	설명
Ruin_Application	프로그램의 시작점
Ruin_System	게임의 여러 기능을 제공하는 시스템들을 전체적으로 관리
Ruin_PhysicsSystem	캐릭터와 주변 환경 사이의 상호작용을 계산. 충돌 검사와 충돌 반응 처리 포함.
Ruin_Frame	Win32 API를 이용하여 윈도우를 생성하고 관리하는 클래스
Ruin_Renderer	렌더링에 사용되는 Direct3D 장치 관리
Ruin_Console	게임 설정을 조절할 수 있는 게임내 콘솔 모드 관리
Ruin_InputMap	DirectInput을 사용하여 사용자의 입력을 받아들임
Ruin_SceneManager	게임내 각각의 장면을 관리
Ruin_Scene	게임내 각각의 장면을 정의하는 추상클래스. 실제 게임을 구현할 때 이 클래스를 상속받아 게임 내용을 구현

8 결 론

본 논문에서는 인터넷 게임 개발에 있어서의 3차원 가상환경 생성 및 제어 방법을 제공하는 게임 프레임워크 구현에 대해 기술하였다. 본 시스템 개발에는 윈도우 2000/XP, 비주얼C++, DirectX 9, 마야, 3DS MAX 등이 사용되었다.

본 연구에서는 게임 개발에 필수적인 3차원 가상환경

의 생성, 캐릭터 이동에 따른 카메라 제어, 가상환경을 배경으로 캐릭터가 이동하면서 환경 정보를 입력할 수 있는 정보 입력기, 게임의 진행을 제어할 수 있는 콘솔 명령어 등이 구현되었다.

본 시스템의 특징 중 하나의 환경 정보 입력기는 시맨틱(Semantic) 가상환경의 대표적인 기능으로서 동적 정보 입력이 가능한 점에서 향후 유비쿼터스 환경에의 적용을 고려할 수가 있다.

참고문헌

[1] "Lars Bishop, Dave Eberly, Turner Whitted, Mark Finch, Michael Shantz, "Designing a PC Game Engine", IEEE CG&A, Vol. 18, No. 1, 1998.

[2] Frank D. Luna, DirectX 9를 이용한 3D 게임 프로그래밍 입문, 정보문화사, 2004.

[3] Marc Erich Latoschik, Peter Biermann, Ipke Wachsmuth, "High-Level Semantics Representation for Intelligent Simulative Environments" Proceedings of IEEE Virtual Reality 2005, pp.283-284, 2005.

[4] Greg Snook저, C++와 DirectX9를 이용한 실시간 3D 지형 엔진, 정보문화사, 2004

[5] Tomas Akenine-MollerEric Hanies 공저, 신병석, 오경수 공역, Real-Time Rendering 2판, 정보문화사, 2003.

[6] Akenine-Möller, Tomas, "Fast 3D Triangle-Box Overlap Testing," Journal of Graphics Tools, Vol. 6, No. 1, pp. 29-33, 2001.

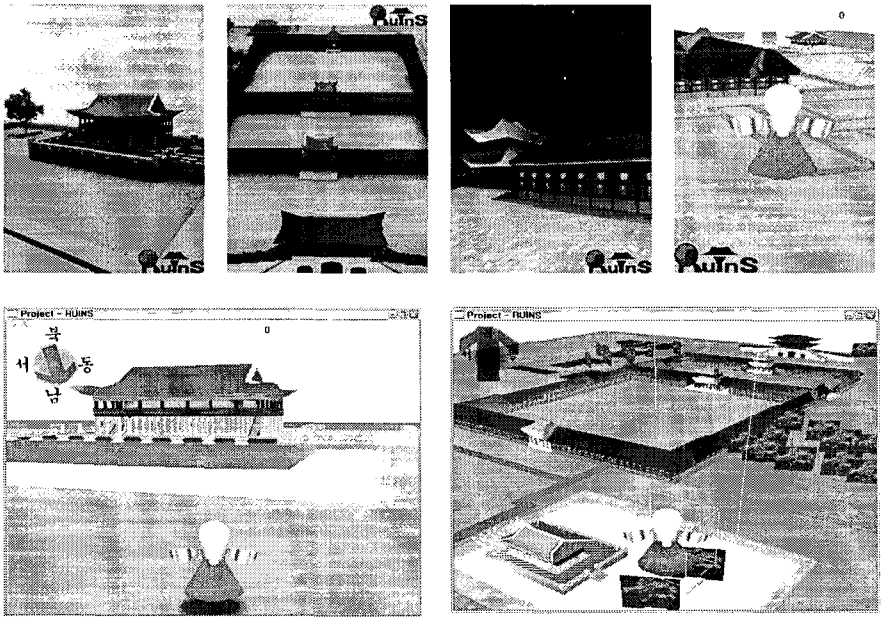
[7] K. Kanev, S. Kimura, "Integrating Dynamic Full-Body Motion Devices in Interactive 3D Entertainment", IEEE CG&A, pp. 76-86, July 2002

[8] Sung-Jin Kim, Falko Kuester, K. H. (Kane) Kim, "A Global Timestamp-Based Scalable Framework for Multi-Player Online Games", IEEE Fourth International Symposium on Multimedia Software Engineering (MSE'02), pp. 2, December 2002

[9] C. Faisstnauer, W. Purgathofer, M. Gervautz, J.-D. Gascuel, "Construction of an Open Geometry Server for Client-Server Virtual Environments", Proceedings of Virtual Reality 2001 Conference (VR'01), pp.105-114, 2001.

[10] J. Purbrick and C. Greenhalgh, "An Extensible Event-based Infrastructure for Networked Virtual Worlds", Proceedings of Virtual Reality, IEEE, pp. 15-21, 2002.

[11] Tolga K. Capin, Hansrudi Noser, Daniel Thalmann, Igor Sunday Pandzic, Nadia Magnenat Thalmann, "Virtual Human Representation and Communication in VLNET", IEEE CG&A, Vol. No. 17, No. 2, 1997.



(그림 7) 게임 실행 장면