# 게임 컨텐츠의 기획과 게임엔진 기술

김성환* · 박태준**

## 1. 서 론

최근 컴퓨터 게임 설계 및 기반기술에 대한 관심이 활발해지고 있다. 컴퓨터 게임을 컴퓨터를 사용한 게임이라고 간단히 정의한다면, 컴퓨터를 사용하지 않고도 즐길 수 있는 많은 게임 컨텐츠들이 있다. 우선 "게임 (game)" 이라는 단어 자체를 조사하면, 유러피언 계통의 "gehem"에서 파생된 단어로, 즐겁게 뛰어 놀다 라는 의미를 가지며, 사전적인 의미를 찾아보면, 놀이, 유희, 즐거움, 오락 등의 포괄적인 정의를 가지고 있다. 또는 배고픔이라는 1차적인 욕망을 해결하고 나서의 또 다른 유희라는 욕망을 만족하기 위한 다양한 수단이라고 생각할 수 있다 [1].

게임컨텐츠를 제작하기 위한 기술은, 주관적인 즐거운 경험을 일반화하고 게임의 즐거움을 배가하는 게임 기획 분야에서부터, 기획된 게임을 실제 컴퓨터를 통해 구현하도록 도와주는 게임제작 기술 (게임엔진기술) 등이 필요한 복합기술이다. 본 논문에서는 게임의 기획, 게임엔진에 대한 사항을 살펴보기로 한다.

## 2. 게임컨텐츠의 기획

### 2.1 개인적인 경험의 일반화 및 변형

게임 컨텐츠가 사람의 욕망, 혹은 즐거움을 만족하기 위한 수단이라면, 실제 게임은 어떻게 설계가 되는 것일까 하고 의문을 가지게 된다. 그림 1은 1972년 ATARI사에서 만든 최초의 상용게임인 <PONG>이라는 게임이다 [1]. 1970년, 테니스 또는 탁구 놀이를 하던 A 군은 넘어온 공을 놓치지 않고 받아쳐 넘겨 상대방을 공격하면서 즐거움을 느끼게 된다. 이제 이런 주관적인 즐거움을 극대화하기 위해, A군은 즐거움을 주는 부분만을 비정상적으로 강조하게 되고, 이 과정에서, '테니스' 또는 '탁구'가 가지고 있는 다른 대부분의 특색은 탈색된다.

이제 한 개의 놀이가 만들어지고, 놀이를 즐기는 많은 사람들에 의해, 게임은 점차로 변화되게 된다. 일례로 상대편이 없어도 게임을 할 수 있도록 상대편을 벽돌 벽으로 만들어두면, <블록 깨기>라는 게임이 만들어지게 된다. <블록 깨기>

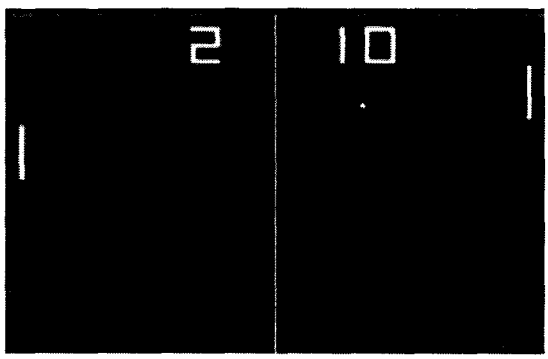



그림 1. 최초의 상용 게임 PONG

* 서울시립대학교 디지털미디어대학 컴퓨터과학부 조교수
** 한국전자통신연구소 가상현실연구부 선임연구원

에서 벽돌 벽이 서서히 움직이면서 플레이어에게 접근해오면 점점 더 스릴감을 느끼게 되면서 즐거움은 더해가게 되는데, 그림 2의 <스페이스 인베이더>는 PONG 게임의 변형이라고 볼 수 있다.

계열적 진화, 변형에 의한 확산, 새로운 아이디어의 등장까지. 게임의 역사에 등장했던 수많은 게임을 위한 하나의 체계를 만드는 것이 가능하며, 게임 기획의 시작이 된다.

주관적 경험의 형상화 또는 변형으로 인해 만들어지는 게임들은, 특정한 사람의 개인적인 경험에서 시작하는 이유로 인해, 특정 사람에게만 재미있는 경험이 될 수도 있다. 하지만 이런 개인적인 경험에는 인간이기 때문에 공유하는 어떤 보편적 측면을 가지고 있으며, 게임 기획의 출발은 이런 개인적인 경험에서 보편적인 측면을 찾아내는 것이다.

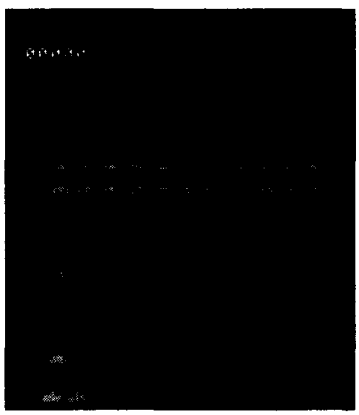


그림 2. PONG의 변형. Space Invader

## 2.2 게임의 형식장르

보편적인 측면이 찾아지면, 이 보편적인 측면을 어떻게 다른 플레이어의 경험 속에서도 재현시켜낼 수 있는가를 연구하는 것이다. 게임 기획자는 게이머를 자신이 경험했던 그 현장으로 데려갈 수는 없지만, 그 현장에 함께 같이 갔을 때 느낄 수 있는 그런 감정을 플레이하면서 느껴야만 한다. 여기서 게임의 형식을 연구하게 되는 것이다. 과연 인터페이스는 어떻게 해야 할 지, 게임의 기본적 논리는 무엇으로 해야 할 지, 게임의 그래픽은 또 어떤 느낌을 가져야 할 지. 이 모든 고민은 현장에 데려가지 못하면서도, 현장에 있는 것처럼 느끼게 해야 한다는 이 상황에서 비롯되는 것이다. 게임의 기본적 논리는 게임의 장르로 귀결할 수 있는데, 게임의 장르를 1982년 처음 정의한 크로포드는 다음과 같이 구분하고 있다 [1].

- Skill and Action 게임: 결투 게임, 미로 게임, 스포츠 게임, 패들 게임, 레이싱 게임, 기타 게임 등
- Strategy 게임: 어드벤처, D&D 게임, 워 게임, 기회 게임, 교육용 게임, 상호관계 게임

Skill and Action 게임은 사람의 반사 신경을 이용하는 게임 대부분이 이 장르에 포함된다. 즉 실시간 플레이, 그래픽과 사운드에 대한 강조, 키보드 보다 조이스틱 혹은 패들의 사용이 이루어지는 게임이며, 이 장르의 게임을 플레이하는 게이머는 손과 눈의 협동 그리고 빠른 반응 행동 시간이 필요하다. 그림 3의 마이크로소프트사의 F15 시뮬레이션 게임 이 범주에 넣을 수 있다.

그림 3. 시뮬레이션 게임: F15

Skill and Action 게임 중에서, 결투 게임 (combat game)은 직접적으로 폭력적 대결을 하는 게임을 말한다. 게이머가 캐릭터를 움직여, 컴퓨터가 움직이는 적들을 파괴하는 게임이다. 이 게임에서는 당연히 적의 공격을 회피하는 능력과 필요한 순간을 놓치지 않고 적을 공격할 수 있는 능력이 필요하다. 미로 게임 (maze gam)은 미로 모양을 빠르게 움직이면서 목적을 달성하는 게임, 즉 <팩 맨> 같은 게임을 말한다. 스포츠 게임 (sports game)은 말 그대로 스포츠 게임이다. 패들 게임 ( paddle game)은 <PONG> 계열의 모든 게임을 말한다. 게이머는 화면 한 쪽에 나와 있는 패들을 움직여서 떨어지는 공을 쳐낸다. 레이싱 게임 (race game) 역시 '스포츠 게임'처럼 직관적으로 이해할 수 있다.

Skill and Action 게임 장르와 함께 게임을 나누는 장르는 Strategy 게임이다. 전략 게이머들은 조작보다 사고력을 필요로 한다. 전략 게임 중에서 가장 오래된 장르인 어드벤처 게임 (adventure game)에서, 게이머는 목표를 실현할 때까지, 다양한 장애물을 극복해야 한다. 그를 위해서 필요한 도구와 아이템을 모으고, 복잡한 세계를 해쳐나간다. 그런 점에서 어드벤처는 일종의 퍼즐이다. 다만 퍼즐은 게이머에게 제공하는 장애물의 속성이 고정적이라는 차이가 있다. 반면 어드벤처가 제공하는 장애물이 고정적일 수 없다. 그림 4는 대표적인 어드벤처 게임인 원숭이 섬의 비밀이다.

D&D 게임은 현재의 롤 플레잉 게임이다. 롤 플레잉 게임 중에서도 가장 발전적인 시스템으로 인정받았던 것이 '던전 앤 드래곤 dungeon and dragon'이며, 줄여서 D&D라고 불리는 이 시스템은 성, 용, 마법사, 난쟁이들의 동화 같은 세계에서의 모험과 협동, 그리고 경쟁을 구현한 복잡한 게임의 세계를 게이머에게 제공해주었다. D&D에서는, '던전 마스터 Dungeon master'의 안내를 받

그림 4. 어드벤쳐 게임: 원숭이 섬의 비밀

는 몇 명의 플레이어들이 보물을 모으기 위해 출발하는데서 시작한다. 게임에 필요한 건 모여 앉을 테이블과 한 묶음의 종이 뿐이다. 던전 마스터는 게임 구조의 규칙을 제시하고 그 게임의 중재 역할을 한다. 던전 마스터는 모든 사건을 심판할 수 있는 권위를 가진다. 하지만 게임의 환경은 매우 느슨하고 비 형식적이다. 이러한 이유 때문에, D&D는 대중적인 게임이 되었다.

그림 5. RPG 게임: 디아블로

하지만 D&D는 혼자서 즐길 수 없다는 단점과, 간혹 주사위를 던지고 하는 행위가 지루함을 유발했다. 이런 점 때문에 컴퓨터가 등장하면서 사람들은 던전 마스터의 역할을 컴퓨터에게 맡기는

것이 좋지 않을까 생각했다. 그래서 '컴퓨터 롤 플레잉 게임 computer role playing game, CRPG'이 등장하게 된다. 그림 5는 대표적인 RPG 게임인 디아블로 화면이다.

기회 게임 (games of chance)은 '블랙 잭'과 같은 카지노 게임을 말한다. 이런 게임 역시 순발력보다는 패를 읽고, 상대편의 행동에 맞춘 전략을 짜야 한다는 점에서 전략 게임의 영역에 들어간다.

마지막 장르는 '인터퍼스널 게임 inter-personal game'이다. 크로포드는 개인 혹은 집단간의 관계성에 초점을 맞춘 종류의 게임을 탐구하려 했다. 다른 예술이 인간들 간의 의사 소통 관계를 고민했듯이 게임 역시도 이 문제를 고민하게 될 것이라는 것이다. 그는 이 역할을 무엇이 하게 될 지는 예측할 수 없지만, 결국 인간을 즐겁게 하는 것은 인간관계, 인간들 간의 소통 관계에서 오는 경험일 것이라고 판단한 것이다.

## 3. 게임컨텐츠의 제작도구

### 3.1 고 수준 게임 제작 환경의 필요성

초기의 게임은 한 사람이 프로그램, 그래픽, 사운드 등 모든 것을 혼자 개발하기도 하였다. 그러나 사용자의 요구사항이 증가하고 어려워졌으며, 하드웨어의 발전과 더불어 새로운 게임 제작 기술이 발달하게 되었다. 이러한 변화에 의하여 게임 제작은 점차 세분화되었고, 게임엔진의 필요성이 중요시되게 되었다. 그림 6은 마이크로소프트사에서 제작한 XBOX용의 해일로라는 컴퓨터게임이다. 이러한 게임을 제작하기 위해서는 고 수준 프로그래밍 또는 게임저작 환경을 요구하게 되고, 이를 게임엔진이라고 한다 [2,3].

게임엔진은 크게 (1) 3D 렌더링 엔진 (rendering engine), (2) 애니메이션 엔진 (animation engine), (3) 사운드 엔진 (sound engine), (4) 물리 엔진 (physics engine), (5) 인공지능 엔진 (artificial intelligence engine), (6) 네트워크 엔진 (network engine) 등으로 구성된다.

그림 6. XBOX용 해일로

### 3.2 3D 렌더링 엔진의 기본기능

PC 기반의 3차원 게임을 제작하기 위해서는 보통 DirectX나 OpenGL과 같은 3D 그래픽 라이브러리(graphic library)를 이용하여 그래픽 가속 하드웨어의 기능을 활용하는 것이 일반적이다. 이때, 이러한 그래픽 라이브러리들은 입력된 3차원 다각형(polygon)을 일괄적으로 처리하여 주어진 카메라 위치나 방향에 따라 2차원 화면에서 이들이 나타나는 위치와 서로 가리고 가려지는 관계와 광원과의 위치 관계 등을 계산하여 화면을 구성하고, 이들 다각형들의 표면 속성과 이들에 입혀진 텍스쳐 (texture)에 따라 다각형 내부의 색상을 채워주게 된다. 다시 말하여 범용의 그래픽 라이브러리만을 사용하는 경우에도 3차원 게임 제작에 요구되는 기본적인 3차원 객체 출력 기능을 수행할 수 있으며 이를 통한 게임 구현이 가능하다.

하지만 일반적으로 게임 제작 과정에서 요구되는 각종 렌더링 기능은 각종 그래픽 라이브러리가

제공하는 단순한 기능 이외에 보다 많은 다양한 기능을 요구하며, 그래픽 디자이너에 의해 작성된 각종 게임 그래픽 데이터를 화면 출력하는 기능 이외에, 게임 로직, 게임객체의 제어를 통해, 게임 객체가 게임 진행 상황에 따라 게임 환경 내에서 적절히 변형, 이동한 후 빠르게 화면에 출력해주는 기능을 하게 된다.

이제 렌더링엔진의 기능을 구체적으로 알아보면 다음과 같은 기능을 제공해야 한다.

- 기본기능 (3차원 객체화면 출력 기능): 렌더링 엔진은 기본적으로 그래픽 디자이너에 의해 작성된 각종 게임 객체를 화면에 출력할 수 있어야 한다. 객체의 형상, 표면 속성, 그리고 객체에 입혀진 각종 텍스쳐가 렌더링 엔진에서 올바르게 처리되어 최종 화면에 적절하게 출력되어야 하며, DirectX나 OpenGL, 최신 그래픽 하드웨어 가속장치의 기능을 최대한 살려낼 수 있도록 구성되어야 한다.
- 고속화 지원 기능 (실시간 객체 출력을 위한 각종 게임 객체 제어 기능): DirectX나 OpenGL 등의 그래픽 라이브러리를 활용할 경우 다각형 메쉬 (mesh)의 형태로 구성된 각종 게임 객체들은 그래픽 하드웨어 가속장치 내에서 2차원 영상의 형태로 렌더링 되어 화면에 출력되게 된다. 이 때, 아무리 최신의 그래픽 하드웨어 장치를 활용하는 경우에도 처리해야 하는 다각형의 수가 많아지면 영상 생성에 소요되는 시간이 많아지게 되어 렌더링 속도가 줄어들게 된다. 이러한 현상을 방지하기 위하여 렌더링 엔진은 전체 게임 환경 중 카메라 시야에 나타나지 않는 게임 객체나, 화면에 나타나더라도 아주 적은 수의 픽셀 (pixel)만 차지하여 그 화면상 중요도가 낮은 게임 객체의 경우 미리 렌더링 과정에서 제거하거나 보다 적은 수의 다각형으로 표현될 수 있도록 하여 전체적으로 렌더링 속도를 향상시키는 기능을 가지고 있어야 한다.

실제 처리할 다각형의 개수를 줄이는 가장 대표적인 방법으로는, (1) 화면에서 차지하는 픽셀의 수가 적어 그 중요도가 낮은 물체를 보다 적은 수의 다각형으로 표현하는 기법인 객체 LOD (level of detail) 기법과, (2) 카메라 시야에서 벗어나 화면에 나타나지 않는 객체를 미리 렌더링 과정에서 제외하여 전체 속도를 향상시키는 culling 기법으로 나누어볼 수 있다 [4].

LOD 기법이란 Level of Detail 기법의 약자로서 화면상에서 덜 중요하게 표현되는 객체를 구성하는 다각형의 개수를 줄여줌으로서 렌더링 과정에 처리되는 다각형의 수를 줄이는 방법이다. 예를 들어 어느 물체가 크기가 작거나 카메라로부터의 거리가 멀어서 화면상에서 차지하는 픽셀의 개수가 많지 않을 경우 이 객체를 매우 많은 수의 다각형으로 표현하는 것은 비효율적이다. 이 때, 객체의 전체적인 형상은 그대로 유지하면서 훨씬 적은 개수의 다각형으로 구성된 객체를 사용하여 렌더링을 하게 되면 렌더링 과정에 처리되는 다각형의 수를 크게 줄일 수 있게 되어 렌더링 속도를 향상시킬 수 있게 된다 [5].

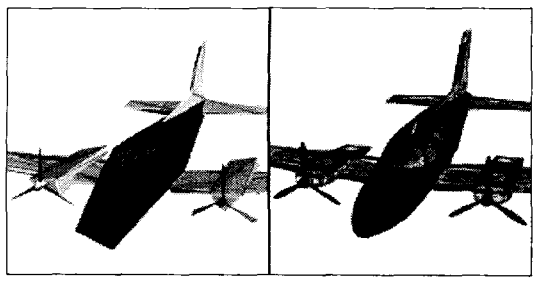


그림 7. LOD 방법 (좌: 150, 우: 13,546 meshes) [6]

그림 7의 예에서는 13,546개의 메시를 가진 비행기를 실시간에 표현하기 위해, 150개의 메시만

으로 표현한 예이다. 먼 거리에 있는 경우, 혹은 자세한 영상이 필요하지 않은 경우 150개 메시를 가진 모델을 이용한다면 속도 개선이 상당할 것으로 예상할 수 있다.

Culling 기법이란 주어진 환경을 렌더링할 때 카메라 시야에 들어오지 않는 객체들을 미리 제외하고 렌더링을 수행하여 전체 렌더링 속도를 향상시키는 방법이다. 이 기법은 특히 많은 수의 다각형으로 이루어진 복잡한 환경에서 카메라가 그 환경의 극히 작은 일부분만을 바라보는 경우가 많을 때 효과를 볼 수 있는 기법이다.

Culling 기법은 그 특성상 (1) Back face culling: 카메라를 향하고 있지 않은 면을 미리 렌더링 과정에서 제외하는 기법, (2) View frustrum culling: 카메라의 뒤에 있거나 카메라 시야 각도에 포함되어 있지 않는 물체들을 렌더링 과정에서 제외하는 기법, (3) Hidden surface culling: 른 물체에 완전히 가려져서 화면에 나타나지 않는 물체를 렌더링 과정에서 제거하는 기법으로 나누어 볼 수 있다.

- 고속화 지원 기능 (각종 쉐이더 (shader) 활용 기능): 과거에는 주어진 그래픽 하드웨어를 이용할 경우 하드웨어에서 제공하는 렌더링 기능만을 이용하여 3차원 객체 출력 루틴을 구현해야만 했다. 하지만 최근 하드웨어 기술의 발달과 GPU (graphic processing unit)의 등장으로 그래픽 하드웨어 가속장치 내의 내부 렌더링 연산을 개발자가 직접 작성한 쉐이더를 통해 수행할 수 있게 되어 과거에는 상상할 수 없었던 새로운 렌더링 알고리즘을 활용한 게임이 점차로 등장하고 있다. 따라서 렌더링 엔진은 게임 개발자가 새로운 쉐이더를 작성한 경우 이를 렌더링 엔진에 포함시켜 렌더링을 수행하도록 해 주는 기능을 제공하여야 한다.
- 지형 (게임환경) 처리 기능: 3차원 게임을 진행하다보면 각종 3차원 객체의 전체 지형 내에서의 위치, 지형과 객체 간 충돌 여부 등을 빠르게 파악할 수 있어야 한다. 또한 그런 과정에도 주어진 카메라 위치에 따라 지형을 빠르게 화면에 출력할 수 있어야 한다. 이러한 지형에 대한 연산의 종류, 실시간 출력에 대한 요구 때문에 지형의 표현을 위한 많은 자료구조와 알고리즘이 제안되어 있다.

렌더링 엔진 구현의 관점에서는 출력해야 하는 환경의 속성에 따라 outdoor 게임과 indoor 게임으로 크게 구분할 수 있다.

Outdoor 게임은 게임 환경의 크기가 방대하며 넓은 지역을 표현하는 지형에 대한 처리가 중요하다. 특히 지형을 다각형으로 표현하는데 있어 그 다각형의 개수를 조정한다거나 지형과 다른 객체 간의 충돌을 어떻게 판단하느냐 등이 큰 문제이다.

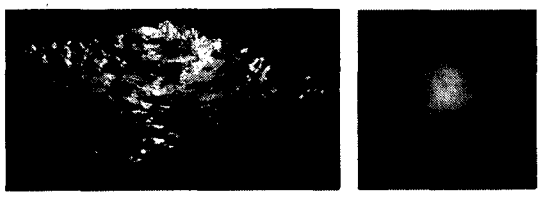


그림 8. 렌더링된 지형 (좌)와 하이트맵 표현 (우)

Indoor 게임의 경우 그 특성상 주어진 환경이 복도나 방 등 실내인 경우가 많으며 따라서 벽 등의 물체가 그 뒤의 물체를 가리고 있는 경우, 또는 옆 방이 열려있는 문을 통해서만 들여다보이는 경우가 많다. 그러므로, indoor 게임의 경우 화면에 표시되지 않는 객체를 빠르게 판단하여 렌더링에서 제외하는 기능이 반드시 요구된다. 지형처리 기능은 실외 환경과 실내 환경으로 각각 최

적화 되게 되며, 실외 환경을 위한 자료구조로는 그림 8과 같은 하이트맵 (height map) 모델을 이용하게 된다 [7,8].

이제, 하이트 맵을 사용하는 경우 세밀한 지형 표현을 위해서는 하이트 맵의 해상도가 매우 높아야 한다. 이 경우 하이트 맵에서 생성되는 다각형의 수가 늘어나므로 렌더링 과정에서 부담이 되게 된다. 따라서 이렇게 하이트 맵으로 표현된 지형에 대한 실시간 출력 기법이 요구되게 되며, 주어진 하이트 맵으로부터 미리 정해진 오차 범위 내에서 가능한 한 적은 수의 다각형으로 구성된 지형을 생성하는 기법들이 많이 발표되어 있다. 그 중, 대표적인 방법인 quad-tree (QT) 방식을 이용한 실시간 지형 출력 예는 그림 9와 같다.

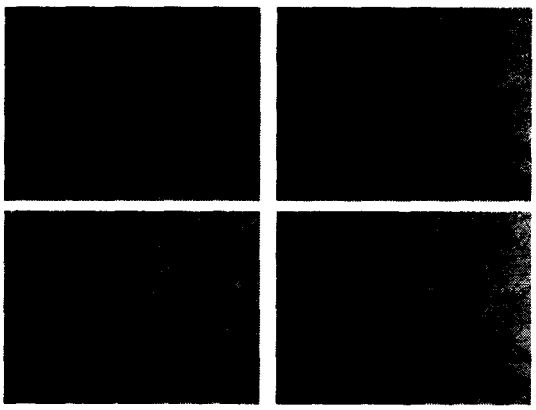


그림 9. 세밀한 하이트맵의 QT를 통한 LOD 제공

실내공간의 경우에는, 벽 등이 막히는 경우가 많고, 실외환경과는 다른 처리가 필요하게 된다. 따라서, BSP (binary space partitioning)와 PVS (potential visibility set)이라는 특별한 형태의 자료구조를 사용하게 된다. BSP 트리는 실시간에 은면 제거 문제를 해결하기 위해 제안되었다. BSP 트리는 물체의 표현 방식이기 보다는 효율적인 렌더링을 위한 공간 분할 방식이라 볼 수 있다. BSP 트리는 각 레벨에서 분할 평면 (splitting plane)을 이용해 공간을 두 개의 부분으로 나눈다. 트리의 모든 비 말단 (leaf) node는 해당 공간을 둘로 나누는 평면을 나타낸다. 말단 node는 더 이상 나눠지지 않는 공간을 나타내고, 그 공간에 포함되거나 걸쳐지는 물체의 자료구조 등을 포함한다.

그림 10은 4개의 물체로 구성된 환경을 BSP 트리를 이용해 분할, 표현한 모습을 보인다. BSP 트리가 구성되면 트리의 비 말단 node에 해당하는 평면들과 카메라의 좌표를 비교함으로써 현재 카메라가 속한 공간을 쉽게 찾을 수 있다. 카메라가 위치한 공간을 알고 있으면 BSP 트리를 이용하여 렌더링할 물체들을 거리 순으로 정렬할 수 있다.

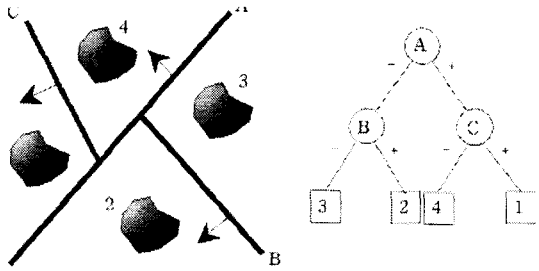



그림 10. 세밀한 하이트맵의 QT를 통한 LOD 제공

제작하고자 하는 실내 게임 환경을 BSP 트리를 이용해 효율적으로 관리한다고 하여도 삼차원 환경 자체가 복잡해지면 화면에 그려야 하는 다각형의 수가 늘어나 렌더링 속도를 저하시킬 수밖에 없다. 따라서 상황에 따라 화면에 보이는 부분이나 필요한 부분만 골라서 화면에 출력하는 기술이 필요하게 되는데, 이러한 잠재적 보일 수 있는 부분을 PVS라고 부른다. 이러한 PVS를 실시간에 계산하는 것은 성능의 저하를 가져올 수 있기 때문에, PVS를 미리 계산해 두면, 카메라가 위치한 특정 말단 node에서 볼 수 있는 말단 node와 볼 수 없는 말단 node를 미리 알게 되므로 필요한

것만 렌더링 할 수 있어 상당한 성능 향상을 얻을 수 있다.

잠재적 가시 집합, 즉 PVS는 환경에 대해 구성된 BSP 트리의 각 말단 node에서 볼 수 있는 타 node들의 리스트로서 렌더링 이전에 전처리 과정을 통해 구성된다. 정적 환경의 경우 PVS를 한 번만 구성하면 되며, 카메라의 이동에 따라 카메라가 속한 말단 node의 PVS에 들어있는 node들만을 렌더링 하여 렌더링 속도를 개선할 수 있다. PVS는 한 말단 node에 속하는 모든 지점에서 어떤 말단 node들이 보이는지를 결정한다. 그림 11은 1 번 공간에 해당하는 말단 node에서 가능한 시야를 보여준다. 이 경우는 1번 말단 node의 PVS에 2, 3, 5, 6번 node에 대한 정보가 저장된다.

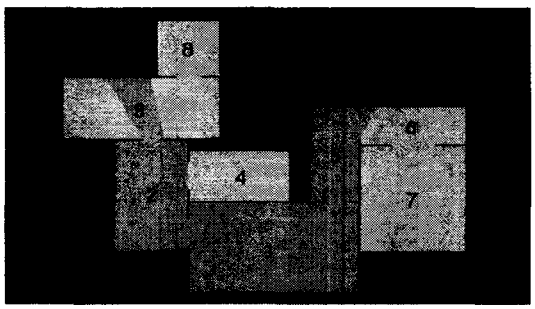



그림 11. 1번 말단노드에서의 PVS (2,3,5,6)

이제 이러한 PVS를 구하기 위해 해결한 문제는, 실제로 2,3번의 노드를 파악하기 위해서는 문틈이 존재하는지를 알아야 한다는 점이다. 어떤 말단 node들이 현재 말단 node로부터 보이는 가를 판단하기 위해서는 말단 node들 사이의 틈, 즉 트인 공간을 알아야 한다. 예를 들어, 두 개의 방이 있는 경우, 1번 방과 2번 방 사이에 놓인 문을 통해 서로를 보게 된다. 그러나 BSP 트리에는 어떠한 곳이 막혔는가에 대한 정보는 있지만, 어디가 문에 해당하는 가에 대한 정보는 없다. 따라서 이러한 문제를 해결하기 위해 임시로 문에 해당하는 다각형을 생성하여 사용하고 PVS 생성이 완료되면 제거한다. 이러한 임시적으로 생성된 문에 꼭 맞는 다각형들을 포탈 (Portal)이라 한다. PVS를 계산하기 위해 필요한 다음 과정은, 각 포탈에 대해서 그 포탈에서 볼 수 있는 말단 node들을 찾는 것이다. 한 말단 node에서 볼 수 있는 다른 말단 node들은 결국 그 말단 node에 속한 포탈들이 볼 수 있는 말단 node의 합이기 때문이다. 각 포탈에서 볼 수 있는 말단 node를 찾기 위해 비 반암부 (Anti - Penumbra) 개념을 사용한다. 반암부는 한 물체에 의해 뿌려지는 그림자를 일컫는다. 비 반암부는 반대의 의미로 빛이 없는 검은 방의 문을 열었을 때, 문을 통해 들어오는 빛의 영역을 의미한다.

- 카메라 이동, 회전, 확대 (zoom-in), 축소 (zoom-out) 제어 기능: 3차원 게임은 카메라의 시점이 2차원 게임에 비해 매우 자유롭다는 특징이 있다. 때문에 렌더링 엔진은 사용자의 카메라 조작에 따라 카메라를 이동 및 회전시키고, 초점거리를 변화시켜 자유롭게 시야를 확대 및 축소하는 기능을 제공하여야 한다. 또한 각 게임 캐릭터의 움직임과 카메라를 연동시켜서 캐릭터 1인칭 시점, 3인칭 시점, quarter view 시점 제어 등의 기능을 제공하여야 한다.
- 광원 및 그림자 제어 기능: DirectX나 OpenGL과 같은 그래픽 라이브러리의 경우 점광원 (point light)이나 방향광원 (directional light), 주변광원 (ambient light) 및 스포트라이트 (spot light)와 같은 기본적인 광원 제어 기능을 제공한다. 이들 광원을 사용하는 경우 그래픽 하드웨어 가속장치에 의해 처리되어 화면에 표현이 되지만 광원의 수가 늘어날수록 속도가 느려진다는 단점이 있다.

따라서 렌더링 엔진에서는 이러한 그래픽 라이브러리에서 제공하는 광원 제어 기능 이외에도 메쉬의 정점 컬러 (vertex color)를 이용한 광원, 그리고 텍스쳐를 이용한 광원 맵 (light map) 기능을 제공하여야 한다. 많은 그래픽 라이브러리에서 점광원, 방향성 광원 등 기본적으로 제공되는 광원들이 있기는 하지만, 이들은 그래픽 하드웨어의 성능에 따라 사용에 제한이 있고, 광원이 추가됨에 따라 많은 계산을 필요로 하게 된다. 또한 Gouraud나 Phong 쉐이딩 역시 고품질의 결과를 내기에는 부족한 점이 있다. 따라서 고품질의 광원 효과를 이미지 형태로 저장해 두었다가 필요시 빠르게 사용할 필요가 대두되었고, 그에 대한 해결책으로 나온 것이 라이트맵 (light map)이다. 라이트맵은 일련의 텍스쳐로써 3차원 환경에서의 조명 정보를 가진다. 조명 정보는 라이트맵에 알파 값이나 색의 형태로 저장이 되고, 후에 다른 텍스쳐와 혼합하는 정보로 이용된다. DirectX나 OpenGL에서 제공하는 멀티 텍스쳐링 기능을 이용하면 라이트맵을 이용해 광원 효과를 연출할 수 있다. 그림 12는 빨강, 파랑, 초록 세 개의 라이트맵 텍스쳐를 물체의 기본 텍스쳐에 입힌 결과를 보인다.

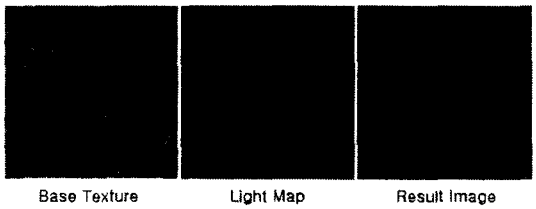



그림 12. 3개의 라이트맵과 적용된 영상

또한 자연스러운 객체의 출력을 위해서는 자연스러운 객체 그림자 처리 기능이 필수적이다. 렌더링 엔진은 이를 위해 그림자 맵 (shadow map) 기능을 비롯한 다양한 그림자 생성 및 출력 기능을 제공하여야 한다.

- 텍스쳐 기반 특수 기능: 최근 그래픽 하드웨어 가속장치의 발전에 따라 과거에는 활용이 불가능했던 텍스쳐 기반의 특수 기능을 게임 내에서 활용할 수 있게 되었다. 대표적인 것이 거울 면 표현을 위해 사용되는 환경 맵 기법과 3차원 객체 표면의 울록볼록한 정도를 텍스쳐로 제어하는 범프 맵 기법이다. 렌더링 엔진은 이러한 텍스쳐에 기반한 여러 특수 기능을 제공해야 한다.

### 3.3 3D 렌더링 엔진의 기타기능

게임을 진행하다보면 눈이나 비, 안개와 같은 각종 대기효과나 캐릭터의 마법, 순간 이동, 공격 및 이에 따른 각종 데미지 효과의 표현, 폭발, 구름이나 물, 연기와 같은 형체가 없는 물체의 표현을 위해 여러 가지 특수효과 출력이 필요하게 된다. 게임에서 특수효과는 게이머를 시각적으로 게임에 끌어들이는 역할을 할 뿐만 아니라, 게임 진행상 아주 중요한 역할을 한다.

특수효과를 만들기 위해 가장 효율적인 방법은 빌보드 (billboard)와 투명도를 가진 텍스쳐/애니메이션 텍스쳐를 이용하는 방법이다. 빌보드는 시점 (view point) 혹은 카메라의 위치에 따라 방향이 변화하는 다각형 (polygon)을 의미하며, 투명도를 가진 텍스쳐와 애니메이션이 있는 텍스쳐와 결합되어 연기, 화염, 폭발, 물방울 등 다각형 메쉬 모델로는 표현하기 힘든 현상을 표현하는데 주로 사용된다 [8]. 그림 13은 빌보드를 사용하여 나무를 렌더링한 예이다.

특수효과의 대표적인 렌즈 플레어는, 그림 14에서와 같이, 카메라의 각도에 의해 렌즈상에 태

양과 같은 밝은 광원의 빛이 잘못 굴절되어 맺히는 허상으로 대개 타원형의 형태로 청색, 적색, 녹

그림 13. 빌보드를 이용한 나무 렌더링

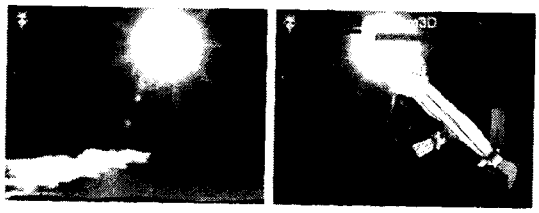


그림 14. 빌보드를 이용한 나무 렌더링

색 계통으로 어우러져 나타난다. 렌즈 플레어는 광환 (corona)과 후광(halo)으로 구성되며 이를 표현하는 반투명 텍스쳐와 빌보드 객체의 조합으로 표현된다.

스프라이트는 마우스 커서와 같은 작은 이미지 조각을 말하는 것으로, 일반적으로 사각형 이미지에 특정 캐릭터와 캐릭터를 둘러싼 투명 부분으로 이루어진다. 이미지의 각 픽셀은 화면상의 픽셀과 일대일 대응하여 화면에 그려진다. 보통 2차원 게임에서 많이 사용되는 기법이며, 스프라이트가 3차원으로 확장된 것이 빌보드라 할 수 있다. 그림 15는 스프라이트를 이용하여 폭발 효과를 나타내기 위해 사용될 수 있는 텍스쳐들이다. 특정 위치에 이 텍스쳐들을 차례로 렌더링하면 폭발 효과가 나타나게 된다.

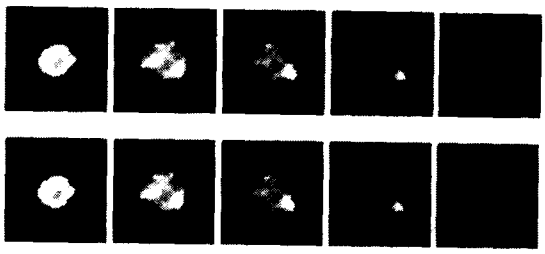


그림 15. 스프라이트를 이용한 폭발현상

파티클 시스템은 특정 알고리즘에 따라 움직이는 작은 입자 (particle)들의 집합을 말한다. 각각의 입자는 생명 주기를 가지며, 생명 주기 동안의 생성, 속성 변화, 움직임, 소멸 등을 알고리즘에 따라 시뮬레이션하여 화면에 출력된다. 이러한 파티클 시스템은 물의 흐름, 폭발, 연기, 불 등의 효과를 표현할 때 사용된다.

모션 블러는 물체가 이동을 하거나, 카메라가 움직일 때, 잔상이 남아 흐리게 보이는 현상이다. 물체가 이동할 때는 전경 물체에 모션 블러가 생기지만, 카메라가 움직일 때는 배경이 흐리게 보인다. 모션 블러 현상을 시뮬레이션하기 위한 방법으로, 축적 버퍼(accumulation buffer)에 연속되는 몇 프레임을 저장하여 최종적으로 각 픽셀의 평균값을 렌더링하는 방법이 있다. 이 방법은 렌더링 속도가 떨어지기 때문에 게임과 같이 실시간 렌더링이 요구되는 응용에는 부적합하다. 다른 방법으로, 버텍스 쉐이더 (vertex shader)를 이용하여 두 프레임을 섞는 방법이 있다.

### 3.4 애니메이션 엔진

3차원 게임의 제작을 위해서는 주어진 3차원 객체의 화면 출력뿐만 아니라, 각 객체의 움직임을 올바르게 표현하고 출력하는 기능이 중요하다. 동적인 캐릭터가 등장하는 게임의 구성을 위해서는 렌더링 엔진 뿐만 아니라 이들 캐릭터의 동작을 생성하는 애니메이션 엔진이 함께 활용되어야 한다.

애니메이션 엔진은 캐릭터나 아이템 등 애니메이션 오브젝트의 동작을 재생 및 실시간 생성하기 위한 엔진이며, 키프레임 (key-frame), 모션 DB 기반의 스키닝 (skinning), 관절체 애니메이션, 얼굴표정 제어, 모션 섞음 (blending), 모션 보간 (interpolation), 역운동학 (inverse kinematics), 버텍스 및 텍스쳐 애니메이션, 계층구조 애니메이션 등의 기능이 제공된다.

그림 16은 애니메이션 엔진의 구조를 나타낸다. 애니메이션 객체 (instance or Actor Instance)는 애니메이션 템플릿 (Template or Actor Definition)으로부터 생성되며, 템플릿을 구성하는 요소로는 관절 구조 (bone hierarchy)와 몸체 (body) 및 행동 (Skill)이 있다. 행동은 관절 (joint)의 궤적 (trajectory or path)의 집합으로 표현된다. 행동은 역운동학과 모션 보간, 모션 캡쳐 데이터를 이용하여 캡쳐된 모션 데이터의 재생뿐만 아니라 실시간 모션 변형을 할 수 있다. 몸체는 캐릭터의 정보 즉, 캐릭터의 텍스춰, 버텍스, 스키닝을 위한 관절 정보를 담고 있으며 모션 데이터를 적용하여 캐릭터가 애니메이션을 하도록 하였다.

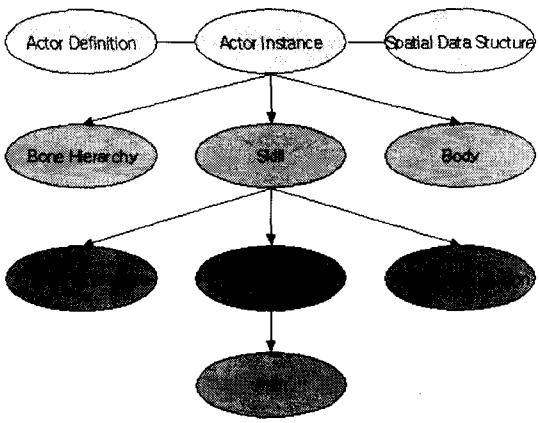



그림 16. 애니메이션 엔진의 구조

### 3.5 사운드 엔진

사운드 엔진은 게임엔진에서 게임 사운드 데이터를 게임 진행 상황에 맞게 스피커에 출력하여 주는 엔진이다 [9]. 사운드 엔진을 통하여 적절한 사운드를 재생할 수 있으며 필요시 일시정지나 반복, 재생 등을 수행 할 수 있다.

사운드 엔진에서는 기본적인 재생기능 이외에 특수효과 기능을 이용하여 사운드를 입체음향으로 만들기도 하고, 필요에 따라 특수효과 필터들을 적용하여 새로운 음향을 만들 수도 있다 [9,10]. 사운드 엔진에서 제공되는 기능들은 (1) 기본적인 사운드 렌더링 기능, (2) 3D 사운드 생성 기능, (3) 사운드 모델 기능, (4) 특수 효과 기능이 있다.

### 3.6 네트워크 엔진

온라인 3D 게임은 네트워크를 기반으로 각 사용자가 서버에 접속하여 하나의 공간을 공유하면서 서로 여러 가지 상호작용을 하는 것이며, 이러한 상호작용에 게임의 성격을 부여하고 게임 규칙을 적용하여, 사용자들에게 만족감을 주는 것을 목적으로 한다. 또한 3D 환경을 공유하며, 주어진 공간을 공유하는 사용자들이 어느 정도의 일관성을 가지며, 서로의 관계가 모순이 없도록 동작하여야 한다. 이러한 목적을 위한 중요한 2가지 기능은 다음과 같다.

- 관심영역 관리 기능 : 서버는 공유상태의 변화를 사용자에게 알릴 필요가 있을 때 그 변화가 일어난 곳 주위의 특정 영역 내에 있는 사용자에게만 공유상태를 보냄으로써 공유상태의 전체 전송률을 크게 줄일 수 있다. 이때 공유상태를 보내는 영역을 관심영역 (Area of Interest)이라 한다. 서버 엔진에서는 그림 17과 같은 셀 기반의 관심영역 관리 방법을 사용 한다 [11]. 셀 기반 관심영역 관리에서는 게임 공간을 셀이라고 하는 단위 영역들로 나누어서 어느 특정 셀에서 발생된 공유상태의 변경을 그 것 주위에 있는 셀들 내에 있는 사용자들에게만 보내주는 것이다. 그러면, 모든 사용자들의 거리를 일일이 검사할 필요 없이 공유상태를 보내야 하는 사용자에게 바로 보낼 수 있다. 이를 위해서 게임 서버는 사용자, 몬스터 등과 같은 공유객체들을 셀 별로 따로 관리하며 각각의 셀들

은 자신의 주위의 셀들의 리스트를 가지고 있다.

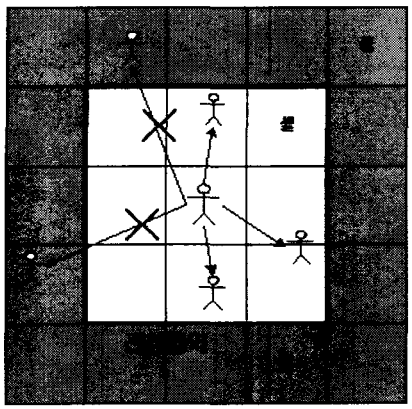


그림 17. 셀기반의 관심영역 관리

- 데드레커닝 알고리즘 (dead reckoning algorithm) : 상에서 수많은 사용자들은 다양한 동적인 상태들을 서로 공유하고 있기 때문에 이러한 공유상태들을 일관성 있게 유지시켜 줄 필요가 있다. 하지만, 네트워크 상에서 다양하게 분포되어 있는 여러 클라이언트 시스템과 서버 시스템 간의 네트워크 지연과 대역폭의 제한으로 인해 모든 사용자들의 공유상태들이 모두 동일하도록 완벽하게 일관성을 유지시켜주는 것은 거의 불가능하다. 일반적으로 눈으로 보이는 화면상의 거의 완벽한 일관성을 위해서는 초당 30번 이상 공유상태들을 주고받아야 한다. 이 경우에 서버는 네트워크 대역폭의 제한 때문에 게임 서버는 많은 사용자들을 수용해줄 수 없을 것이다. 반면에, 공유상태들의 전송률을 지나치게 줄이면 게임 서버는 많은 사용자들을 수용할 수는 있겠지만 일관성이 실시간으로 유지되지 않으므로 사용자들이 서로 다른 상태를 가지게 되어 게임을 제대로 진행해 나갈 수 없게 된다. 이와 같이 서버 성능 (서버의 최대 수용가능 동시사용자)과 게임상태의 일관성은 둘 다 높을수록 좋지만 서로 역비례 관계에 있기 때문에 양쪽을 모두 고려하여 공유상태들의 전송률을 적절히 유지시켜줄 필요가 있다.

이를 위하여 게임 엔진에서는 PC 객체 및 기타 공유객체들의 위치에 대해 데드레커닝 알고리즘을 지원한다. 데드레커닝 알고리즘에서 공유상태를 변화시킨 사용자는 다른 사용자들에게 특정조건이 만족될 때에만 변화된 상태정보를 보내준다. 그리고 변화된 상태정보를 받는 사용자들은 불연속적으로 받은 상태정보를 이용하여 연속적인 값을 만들어 사용한다. 공유상태를 변화시키는 사용자 (공유상태를 보내는 사용자)는 과거에 보낸 상태정보 (상태값, 변화율 등)를 근거로 하여 추정 알고리즘 (Tracking Algorithm)을 이용하여 다른 사용자들이 추정하고 있을 공유상태를 계속해서 추정한다. 그리고 계속해서 추정된 상태값과 실제 상태값과의 오차를 검사하여 이 오차가 미리 정해진 문턱값 이상이 되면 다른 사용자들에게 새로운 상태정보 (상태값, 변화율 등)를 보낸다. 따라서 불필요한 상태정보의 전송을 막을 수 있으며 문턱값을 조정해 줌으로써 상태정보의 전송률을 제어해줄 수도 있다. 문턱값이 크면 클수록 사용자들 간의 공유상태 오차는 커지지만 상태정보의 평균 전송률은 떨어진다. 반면에 문턱값이 작으면 작을수록 공유상태의 오차는 줄지만 평균 전송률이 늘어나게 된다. 한편, 변화된 공유상태를 받는 사용자는 과거에 받은 공유상태를 근거로 하여 현재의 공유상태를 계속해서 추정하다가 새로운 상태정보를 받으면 공유상태를 보정해준다. 이때 불연속적인 상태값이 변하는 것을 막기 위하여 수렴 알고리즘 (Convergence Algorithm)을 사용하여 공유상태를 갱신해준다.

### 3.7 게임엔진의 장점

각 게임 개발사가 보유하고 있는 게임 제작을 위한 각종 노하우 (knowhow)와 기술, 그리고 하드웨어를 제어하기 위한 각종 API의 사용법 등은 게임 하나만을 위해서만 사용되고 폐기되는 것이 아니라, 같은 게임 개발사의 다른 게임의 제작에도 계속적으로 보완 활용된다. 이러한 기술 재사용의 편의성을 위하여 각 게임 개발사는 자사의 노하우 및 기술을 언제라도 수정 및 보완하고 재사용할 수 있는 소프트웨어 엔진의 형태로 관리하고 있다. 게임엔진을 이용하여 게임을 개발할 경우에 개발기간의 단축 및 개발비용을 절감할 수 있다는 장점이 있으며 양질의 컨텐츠를 제작할 수 있다 [12]. 자체적으로 게임엔진을 만들고 이를 이용하여 게임을 개발하거나, 상용화된 게임엔진을 도입하여 개발할 수도 있다. 대표적인 게임엔진은 다음과 같다.

- 퀘이크 엔진 : 퀘이크 (Quake) 엔진은 게임 엔진이라는 용어를 처음 사용한 엔진으로 볼 수 있다. Id Software에서 개발한 엔진으로 John Carmack을 중심으로 하는 프로그래머에 의해 개발되었으며 퀘이크 엔진, 퀘이크 II 엔진, 퀘이크 III arena 엔진의 3 가지 버전이 있다. 보통 퀘이크 엔진이라고 불리는 GPL (GNU General Public License) 엔진 소스는 1999년 공개되었으며, 이 소스를 가지고 게임을 개발한 경우에는 소스를 공개하도록 되어 있다. 퀘이크 엔진은 이전의 둠 (Doom)이라는 엔진을 개선한 것으로서 최초의 3D 엔진으로 평가받고 있다.
- 언리얼엔진: 언리얼 (Unrea) 엔진은 Epic Games 사의 Tim Sweeny가 주축이 되어 개발한 엔진으로 퀘이크 엔진 이후에 등장한 엔진으로 퀘이크 엔진의 기능을 대폭 개선한 엔진이며 액션 게임 엔진으로는 가장 훌륭하다는 평을 받고 있다. 퀘이크 엔진을 참고하여 만들었기 때문에 퀘이크의 기본적인 장점을 갖고 있으며 여기에 여러 가지 문제점을 보완하였다.
- 넷이머스 엔진 : 넷이머스 (NetImmerse) 엔진은 NDL이라는 회사에서 개발한 3D 엔진이다. 이 회사는 렌더링 소프트웨어를 개발하는 회사로 넷이머스 엔진은 그 동안의 3D 소프트웨어 기술을 게임에 사용하도록 만든 엔진이다. 이 엔진은 기본적으로 3D 기능을 강조하여 개발된 것으로, 네트웍 기능이나 인공지능 기능은 특정 게임을 위해 별도로 개발하여야 한다. 또한 특정한 게임 컨텐츠 개발의 결과로 만들어진 엔진이 아니라 처음부터 게임 개발도구로 판매하기 위하여 만들어졌다는 것이 특징이다.

넷이머스 엔진은 실내와 실외 환경을 모두 지원하며, 애니메이션에 있어서는 deformable 애니메이션과, skeleton 애니메이션 등의 기능을 제공한다. 파티클 엔진의 경우에도 눈, particle emitter, particle bomb, particle array, particle cloud와 같은 기능을 지원하고 있다. 그리고 3D 사운드를 지원하며 실내의 음향 반사를 고려하여 사운드를 처리할 수 있다. 충돌 감지에 있어서는 움직이는 오브젝트와 움직이지 않는 오브젝트로 구분하는 계층 구조적인 접근방법을 사용함으로서 충돌 감지에 소요되는 CPU 처리시간을 줄이도록 하였다. 한편, 테리언 (terrain)의 LOD (Levle of Detail)에는 progressive mesh 알고리즘을 사용함으로서 카메라의 이동이 있을 경우에도 자연스럽게 처리될 수 있도록 하는 continuous LOD 기능을 제공한다. "Dark Age of Camelot"은 넷이머스 엔진을 이용하여 개발된 게임 가운데 하나이다.

## 4. 결 론

게임컨텐츠 개발을 위해서는, 주관적인 즐거운 경험을 일반화하고 즐거움을 배가하는 게임 기획 분야에서부터, 기획된 게임을 실제 컴퓨터를 통해 구현하도록 도와주는 게임제작기술 (게임엔진기술) 등의 복합기술이 필요하다. 본 논문에서는 게임의 기획, 게임엔진의 구성요소에 대해 소개하였다. 게임컨텐츠의 특성 상, 다양한 플랫폼, 또한 모바일화라는 추세에서 플랫폼에 대한 논의는 차후에 하기로 한다.

## 참 고 문 헌

[1] 김동현 외, "게임 기획론," 홍릉과학출판사, 2003.

[2] A. Watt, F. Policarpo, "3D Games:Real-time Rendering and Software Technology," Addison-Wesley, 2001.

[3] D. H. Eberly, "3D Game Engine Design: A Practical Approach to Real-Time Computer Graphics," Morgan Kaufmann Publishers, 2001.

[4] A. Watt, "3D Computer Graphics, 3rd edition," Addison-Wesley, 2000.

[5] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Mesh Optimization," SIGGRAPH 1993, pp 19-26, 1993.

[6] H. Hoppe, "Progressive Meshes," SIGGRAPH 1996, pp 99-108. 1996.

[7] P. Lindstorm, D. Koller, W. Ribarsky, L. F. Hidges, N. Faust, and G. A. Turner, "Real-Time, Continuous Level of Detail Rendering of Height Fields," SIGGRAPH 1996, pp 109-118, 1996.

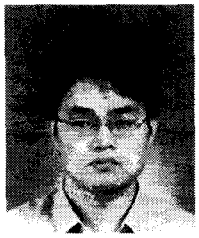
[8] Tae-Joon Park, Soon Hyoung Pyo, Chang Woo Chu, and Byoung Tae Choi, "Design and Implementation of a Rendering Engine for Developing Computer Games," Japan Korea Computer Graphics Conference 2001 proceedings, 2001.

[9] D. R. Begault, "3-D Sound for Virtual Reality and Multimedia," New York, Academic Press Inc., 1994.

[10] M. R. Schroeder, "Natural Sounding Artificial Reverberation," J. Audio Eng. Society, vol. 10, no 3, 1962.

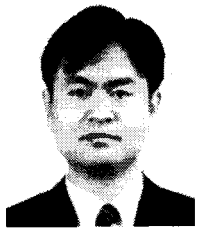
[11] Andrew Rollings, Dave Morris, "Game Architecture and Design," Coriolis, 2000.

[12] 이헌주, 김준애, 임충규, 김현빈, "온라인 3D 게임엔진 표준화," 한국정보처리학회지, 2002년 5월호.



김 성 환

- 1991년 동국대학교 전자계산학과 (공학사)
- 1993년 한국과학기술원 전산학과 (공학석사)
- 1999년 한국과학기술원 전산학과 (공학박사)
- 1996년~2000년 LG전자 IMT-2000 SW 팀장
- 2000년~2001년 Cisco Systems EWBU 책임연구원
- 2000년~현재 한국컴퓨터게임학회 이사
- 2002년 3월~현재 서울시립대학교 컴퓨터과학부 조교수
- 관심분야 : 무선 멀티미디어 통신, 모바일 게임엔진



박 태 준

- 1992년 한국과학기술원 전산학과 (공학사)
- 1994년 한국과학기술원 전산학과 (공학석사)
- 1999년 한국과학기술원 전산학과 (공학박사)
- 1999년 3월~2000년 2월 이스라엘 텔아비브 대학 전산학과 Post Doc.
- 2000년 3월~현재 한국전자통신연구원 디지털컨텐츠연구단 게임SW 기술연구팀장
- 2003년 2월~현재 한국컴퓨터게임학회 이사
- 2004년 4월~현재 모바일 3D 표준화 포럼 정보 및 기획분과 분과장
- 관심분야 : 콘솔, PC용 3D 게임엔진, 모바일게임엔진