3D 온라인 게임을 위한 지형생성기

성명건, 이석희, 황성진, 박경환
동아대학교 컴퓨터공학과

A Terrain Generator for 3D On-line Games

Myung-gun Sung, Serk-hee Lee, Sung-jin Hwang, Kyung-hwan Park
Dept. of Computer Science, Dong-A University
E-mail : personar@donga.ac.kr,
gameazit@hotmail.com,
hsj0801@hanmail.net,
khpark@daunet.donga.ac.kr

요 약

3D 게임 엔진 중 그래픽 엔진이 차지하는 중요도는 다른 엔진들보다 월등히 높다. 그래픽 엔진에는 여러 가지 필요한 요소들이 들어가는데, 지형을 만들어내는 것 역시 캐릭터를 제작하는 것만큼이나 중요한 역할을 한다. 3D 지형이 좀더 현실과 흡사하기 위해서는 실세계와 유사한 가상의 세계를 만들어야 한다. 3D 지형을 만드는 방법은 여러 가지가 있으나 자신이 원하는 지형을 직접 만들어서 이를 3차원으로 바꿔서 게임 엔진에서 사용할 수 있고, 또한 게임 엔진 내부에서의 최적화를 위해서는 가장 기본적으로 쓰이는 그레이 스케일 이미지를 사용한 높이 맵으로 지형을 생성하는 게 유리하기 때문에 본 논문에서는 그레이 스케일 이미지를 이용한 지형생성기를 개발한다.

1. 서론

국내 게임 시장에서 사용자들에게 가장 좋은 반응을 보이는 플랫폼이 온라인 게임 시장이다. PC 게임이 1999년의 66%에서 2001년 22%로 감소하는 데 비해 온라인 게임은 23%에서 43%로의 증가하는 양상을 보이고 있다. 이는 PC 기반의 게임들이 점차적으로 온라인게임으로 전환될 것이라는 것을 예측할 수 있게 한다. 또한 PC 기반 게임 전반에서 2배 이상으로 2D 게임보다 3D 게임이 개발되고 있으며 이런 편차는 더욱 커질 것으로 전망된다. 물론 플랫폼이 다르지만 업소용, 가정용 게임기 시장은 이보다 더 큰 차이를 보이고 있으며 점차 2D 게임들은 게임시장에서 아주 작은 부분을 차지하게 될 것이다. 아직까지 국내 온라인 게임 시장에서 상용화 된 게임 중 많은 사용자층을 확보하고 있는 게임들의 약 70%가 2D 온라인 게임들이다. 이것만을 보면 아직까지는 2D 게임이 강세를 보이고 있으나 상용화되지 않은 베타판들의 게임을 살펴볼 경우에는 약 60%의 게임들이 3D로 제작되고 있다[1]. 이 수치는 일반 PC 게임에서 보이고 있는 현상과 비슷하게 거의 대부분의 게임들이 3D로 제작될 것이라는 예상을 할 수 있으며, 실제적으로도 3D 온라인 게임들이 제작될 계획이 훨씬 많다. 3D 온라인 게임의 발전이 늦은 이유는 온라인이라는 특수상황에 묶여 있기 때문이다.

본 논문의 목표는 3D 온라인 게임을 제작하기 쉽게 해주는 3D 온라인 게임 엔진을 만드는 것이다. 목표로 하고 있는 3D 게임 엔진에는 기본적으로 그래픽 엔진과 네트워크 엔진이 필수적이다. 이외에 추가적인 엔진들이 더 필요하게 될 것이다. 이 중에서도 3D 그래픽 엔진이 3D 게임엔진이라고 불릴 만큼 가장 핵심을 이루는 엔진으로 전체적인 맵 속에서 지형을 생성하고 그 위에 캐릭터나 객체를 올려놓아 게이머가 캐릭터를 통해 게임을 실행할 수 있는 전체적인 환경을 구축하게 되는 것이다. 이런 환경을 구축하게 되는 가장 토대가 되는 것이 지형 생성이다. 어떠한 산이나 언덕을 지나다닐 수도 있고, 건물의 내부를 움직일 수도 있는 것이다.

본 논문은 전체 3D 온라인 게임 엔진의 세부 목표로써 그래픽 엔진에서 사용하는 지형 생성기를 개발하는 것이다.

2. 관련연구

2.1. 3D 게임 엔진

게임 엔진이라 하면 하나의 게임을 만들기 위한 중추적인 역할을 한다고 말할 수 있다. 시나리오와 사운드도 중요하지만 엔진이 먼저 선행되어야 시나리오와 그래픽을 구현할 수 있는 것이다.

게임 엔진의 구성요소를 그림으로 나타내면 아래와 같다.

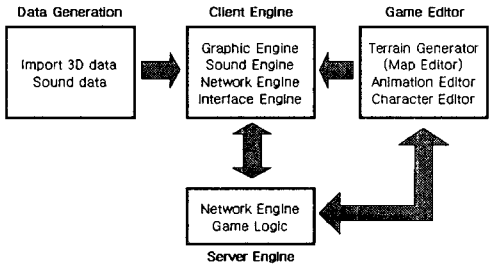

그림 1 게임엔진의 구성도

2.1.1 데이터 생성

데이터 생성은 게임 엔진에서 사용될 데이터를 생성하는 부분으로 게임 엔진 외부에서 불러오는 것이다. 우선 Import 3D data는 3DS MAX나 Lightwave와 같은 3D 그래픽 프로그램에서 모델링한 파일들을 불러와서 사용할 수 있도록 한다. 사운드 역시 외부 프로그램에서 생성한 사운드 파일을 게임 엔진에서 사용할 수 있도록 한다.

2.1.2 게임 에디터

게임 엔진으로 동작시킬 개체들을 만들고 편집하는 도구이다. 게임에서 사용되는 캐릭터들을 만드는 캐릭터 에디터가 있다. 캐릭터나 이외의 동적인 객체들을 움직이게 하는 애니메이션 에디터에는 실제와 근접한 움직임을 위해 모션 캡쳐를 사용하기도 한다. 그리고 3차원 세계를 만들어 주는 맵 에디터가 있다. 맵 에디터에는 실내 환경과 실외 지형을 생성하는 부분이 있다.

2.1.3 클라이언트/서버 엔진

게임 에디터와 외부에서 만들어진 데이터를 가지고 실제 게임을 제작하는 부분이다. 게임 시나리오에 따라 가상의 3차원 세계를 만들고 그 위에 캐릭터를 위치시켜 동작하도록 하는 것이다. 우선 이를 위해 그래픽 엔진에서 그래픽에 관하여 전반적으로 담당한다. 여기에는 에디터에서 만들어진 애니메이션을 게임에 적용시키고, 렌더링을 하며, 렌더링 속도의 향상과 렌더링 시에 시스템의 부하를 줄일 수 있는 최적화 부분으로 LOD(Level Of Detail), 격투 게임에서나 어떤 물체와 부딪혔을 때 겹침을 검사하는 충돌 감지(Collision Detection) 등으로 구성된다. 뿐만 아니라 게임에서 쓰여질 알고리즘과 인공지능을 위한 알고리즘으로 구성된 게임 로직, 게임에 쓰일 사운드들을 위한 사운드 엔진, 게임 인터페이스를 구축할 인터페이스 엔진, 그리고 네트워크로의 데이터 전송으로 온라인 게임이라는 특징을 표현할 네트워크 엔진 등으로 게임 엔진은 구성된다.

3D 게임 엔진은 이 모두를 전부 포함한 것을 가리키거나 그 일부를 가리키기도 한다. 그러나 3D 게임 엔진이라 칭하는 것은 게임의 그래픽 효과의 구현에 중점을 둔 그래픽 엔진을 칭하며, 사실상의 3D 게임 엔진의 성능을 결정짓는 가장 핵심적인 부분이라고 할 수 있다.

2.2. 지형 생성

지형은 단순히 게임에서뿐만 아니라 지리정보 시스템(GIS : Geographic Information System)과 같이 지리학 분야에서도 사용되고 있다. 지형을 표시하기 위해서는 기본적으로 높이 필드(Height field 또는 Height Map)를 사용하며 이를 통해서 지형을 생성할 수 있다. 높이 필드란 저장은 지형의 높이 정보만 하고 폴리곤은 내부적인 알고리즘으로 생성하는 방법으로 일정간격의 (x, y) 좌표에 대한 높이값 z를 뜻한다. x, y축 성분이 일정한 간격으로 존재하기 때문에 z축 데이터만을 저장하면 되므로 방대한 지형에 대한 저장공간을 크게 줄일 수 있다.

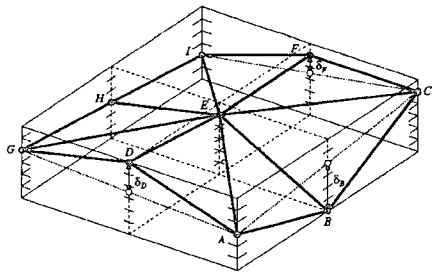

그림 2 높이 필드

그림 2에서와 같이 A에서 F까지 9개의 높이 필드 값들이 저장되고 실제로 렌더링 될 때 필요한 폴리곤은 내부 알고리즘으로 생성해내게 된다.

지형을 생성하는데는 여러 가지 방법이 있으며, 몇 가지 방법을 소개한다.

2.2.1 난수적 지형생성

이는 난수적인 방식으로 지형을 만드는 기법으로,

주어진 격자(그리드, grid)가 있다면 난수를 사용하여 채우는 것으로 실시간 지형 생성의 한 방법으로 쓰이나 이렇게만 해서는 결과물로 나온 지형은 실세계의 지형에 비해 부족함을 보인다.

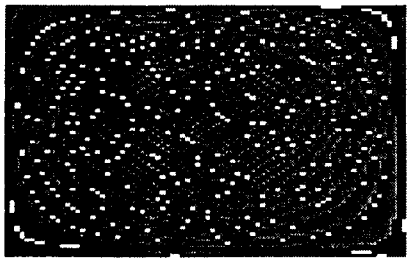

그림 3 난수로 생성한 맵

이는 컴파일러의 난수 생성기에서 생성되는 수가 충분히 랜덤하지 않기 때문이다. 이를 좀더 사실적인 지형으로 만들려면 난수로 의해서 만든 후 추가적인 처리를 해줘야 한다. 또한 그 위에 생성된 격자 각각의 칸의 값을 그 인접한 칸들의 평균값으로 바꾸는 것이다[2]. 더 추가적인 처리방법이 필요하지만 이렇게 생성된 지형은 말 그대로 랜덤하게 생성되었기 때문에 게임 상에서 필요하게 결정된 지형으로 사용하기에는 문제점이 있다. 이 이후에 표현되는 다른 방법과는 달리 랜덤하게 지형의 형태를 생성한 이후 높이 필드를 사용하여 3차원 지형으로 만드는 추가적인 처리가 필요하다.

2.2.2 Fault 알고리즘

다양한 지형적 특징을 나타낼 수 있는 방법으로 이는 2차원 높이 필드에서 모든 포인트의 값을 0으로 초기화 한 뒤 무작위적으로 하나의 선을 긋고 한 영역에 오프셋 값을 더한다. 다시 다음으로 다른 새로운 선을 긋고 나서 한 영역의 모든 값에 값을 더하는 작업을 반복하게 되면 밝은 부분과 어두운 부분이 단계적인 변화를 주면서 나타나게 된다.

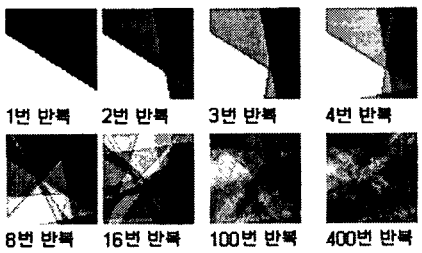

그림 4 무작위 오프셋값 반복

이를 통해 밝은 색일수록 높은 필드, 어두운 색은 낮은 필드를 보이는 지형을 생성할 수 있다[2][3]. 물론 실시간으로 지형을 생성할 수 있으며 현실적으로 그럴듯한 지형을 만들 수 있으나, 무작위로 만들어지는 지형이라는 문제는 잔존한다.

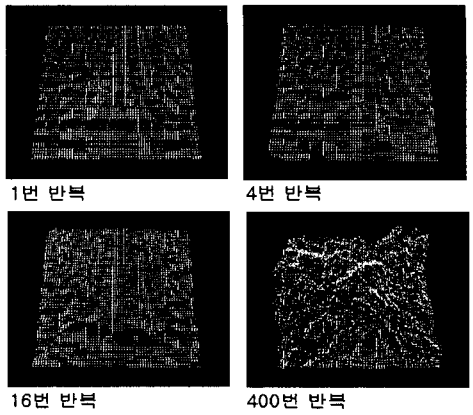

그림 5 생성된 지형

2.2.3 엘리베이션 그리드

자신이 정하는 높이 필드를 사용하여 지형의 면을 생성하는 기법으로 평면 높이 필드에서 지형 생성을 시작한다. 격자의 포인트 중에 하나를 선택하여 원하는 높이 값을 주게되면 산의 꼭대기나 분화구 등을 표현할 수 있다. 반복적으로 이런 처리를 계속하게 되면 지형을 생성할 수가 있다[4]. 물론 이 방법은 위의 두 가지 방법에 비해 자신이 원하는 지형을 3차원으로 만들 수 있다는 점은 있으나 VRML과 같은 특정 애플리케이션에 한정되어 있다.

그림 6 그레이스케일 이미지와 렌더링 이미지

2.2.4 이미지를 사용한 높이 맵

이는 현재 본 논문에서 사용하는 방식으로 그레이스케일의 이미지를 사용하여 3차원 지형을 표현하는 것으로, RGB 형태의 이미지를 사용하는 것보다 구현이 간단하다. 픽셀의 조도가 높이 값에 상응하게 된다. 이 높이 값에 따라 2차원 높이 필드에 메쉬의 높

낮이를 표현함으로써 자연스러운 지형을 표현하는 것이다[5].

2.2.5 Voxel 알고리즘

볼륨픽셀(Volume Pixel)의 합성어로써 3차원 공간에서 한 점을 정의하는 그래픽 정보의 단위로 픽셀에서 x, y 좌표에 위치한 한 점의 색상 및 밀도의 차이를 높이 z 값으로 표시하여 3차원 지형을 생성하는 방식이다. 3D 비행 시뮬레이션 게임이나 방대한 지형을 실시간으로 표현할 때 자주 사용된다[6][7]. 하지만 정확히 말해 복셀은 폴리곤으로 지형을 생성하는 방식이 아니다.

2.2.6 BSP 알고리즘

이진 공간 분할(Binary Space Partitioning) 은 3D 액션 게임에서 가장 유명한 퀘이크 엔진에서 지형을 표현하기 위해 사용되었다[8]. 공간을 분할하는 정보를 담고 있는 2진 트리를 의미하는 것으로, 입체를 구성하는 폴리곤 리스트에서 하나의 값을 루트로 삼고 이를 이진 트리로 구성 두 개의 하위 노드에서 앞쪽 리스트는 기준보다 앞쪽에 있는(front) 폴리곤들로 결정하고 뒤쪽 리스트에는 기준보다 뒤쪽에 있는 폴리곤으로 하여 모든 BSP 트리를 구축한 후 이 트리를 이용하여 PVS(Potential Visibility Set)을 구성한 뒤 추가 적인 처리를 하여 지형을 생성하지만 이는 전체적으로 익스테리어(outdoor)를 표현하는 지형이 아닌 인테리어(indoor)를 만들어 내는데 필요한 알고리즘이다[9][10][11].

2.2.7 LOD(Level of Detail)

컴퓨터에 표현되는 개체들을 표현하는 방법 중 대상 물체와 카메라 사이의 거리에 따라 다른 모델링 데이터를 선택적으로 렌더링하는 방법으로 카메라가 가까이 있다면 많은 폴리곤을 사용하여 세밀하게 표현하고 멀리 떨어져 있다면 폴리곤 수를 줄여 렌더링 속도를 향상시키는 기법으로 현재 지형 생성시 많이 사용되고 있는 기법이나 이에 사용되는 Quad Tree나 Octree 역시 높이 필드의 방식중 하나이고 최적화를 위한 방법으로 지형 생성뿐만 아니라 개체들을 표현할 때도 사용되는 것으로 게임엔진 전반에 걸쳐 사용되는 부분이다[12][13].

3. 시스템의 개발

3.1. 지형생성기의 설계

전체 지형생성을 위한 방법으로 모듈을 사용하여 단순히 지형생성에서 뿐만아니라 다른 3D 처리에서도 사용할 수 있도록 하였다. 지형생성기의 구성요소는 다음과 같다.

3.1.1 2D 비트맵 처리

자신이 원하는 형태의 지형을 그레이 스케일 형식으로 만들면 이를 BMP, JPG 형식으로 만들어 지형 생성기에서 사용할 수 있다. 기본적으로 RGB 모드를 모두 사용하기 때문에 16만 칼라를 모두 사용할 수도 있다. 픽셀의 크기는 180×180이하로 제한을 두었다.

3.1.2 3D 구동

수학적인 3차원 부분으로 사각형, 각도, 버텍스, 벡터, 매트릭스, 색상을 표현하는 부분과 렌더링, 라이팅, 프리미티브, 재질, 카메라와 같은 실제적인 구현에 필요한 부분으로 모듈을 나누었다. 이로써 단계를 준 추상화를 하였다.

3.1.3 지형 생성 알고리즘 처리

수학적인 부분과 구현의 부분을 모두 사용하여 지형을 생성하는 알고리즘 하나의 모듈에서 다시 구현하였다.

3.1.4 사용자 인터페이스

메쉬의 높이 값을 사용하여 전체 지형의 높낮이를 정할 수 있고, 색상을 통하여 렌더링된 지형의 기본적인 색상을 정할 수 있도록 하였다. 2차원 비트맵을 불러오는 부분과 텍스쳐 매핑시킬 비트맵을 불러오는 부분으로 나누었고 전체 렌더링된 이미지가 나타날 부분이 있다. 전체 인터페이스는 아래와 같다.

3.2. 지형생성 알고리즘

전체에서 가장 중요한 지형 생성 알고리즘은 아래와 같다. 전체 사이즈에서 각 픽셀의 값을 가지고 이의 그레이스케일 값을 높이의 값으로 정한다. 픽셀의 R, G, B를 사용하여 그레이스케일 형식으로 나온 값을 기본 높이에 곱하여 높이를 계산한다.

```
Grid_size = Picture_width / Mesh_size
// 그리드크기는 전체 픽셀의 넓이에서 메쉬 크기를 나눈 것으로 정한다
Pixel_x, Pixel_y Init

for x, y = 1 to Mesh_size
  GetPixel(Pixel_x, Pixel_y)

  // 버텍스의 내 값을 돌면서
  Polygon.Ver(1)
    Polygon.x = x
    Polygon.z = y + 1
    // y값만 증가
    column = GetPixel(Pixel_x, Pixel_y + Grid_size)
    if column = -1 then
    // 가장 최저이면
      Polygon.y = 0
      // 높이는 최저이고
```

```
    else
      Polygon.y = height * GreyScale(column)
      // 기본 높이 값에 컬럼의 그레이스케일 값을 곱하
여 높이로 사용한다.

  Polygon.Ver(2)
    Polygon.x = x + 1
    Polygon.z = y + 1
    // x, y 모두 증가
    column = GetPixel(Pixel_x + Grid_size, Pixel_y +
Grid_size)
    if column = -1 then
      Polygon.y = 0
    else
      Polygon.y = height * GreyScale(column)

  Polygon.Ver(3)
    Polygon.x = x
    Polygon.z = y
    column = GetPixel(Pixel_x, Pixel_y)
    if column = -1 then
      Polygon.y = 0
    else
      Polygon.y = height * GreyScale(column)

  Polygon.Ver(4)
    Polygon.x = x + 1
    Polygon.z = y
    // x값 증가
    column = GetPixel(Pixel_x + Grid_size, Pixel_y)
    if column = -1 then
      Polygon.y = 0
    else
      Polygon.y = height * GreyScale(column)
```

3.3 지형생성기의 구현

구현한 시스템의 OS는 Windows 2000 Server, 전체 프로그래밍은 Visual Basic 6.0으로 했으며, 3D API로는 DirectX7.0 SDK를 사용했다. 아래는 이를 구현하여 프로그램을 실행한 인터페이스와 결과이다.

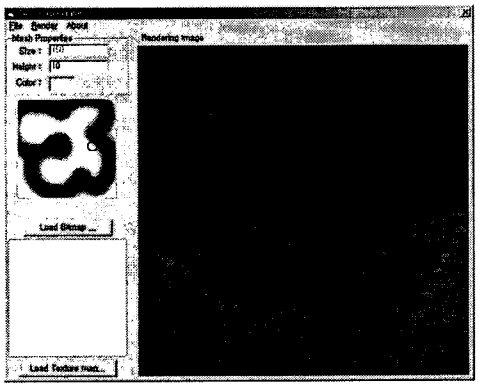

그림 7 인터페이스와 렌더링 결과

메뉴를 통하여 비트맵과 텍스쳐매핑 비트맵을 불러올 수 있고, 렌더링 메뉴를 통해 하드웨어적인 렌더링과 소프트웨어 에뮬레이팅을 선택할 수 있고, 프레임을 솔리드로나 와이어 프레임 또는 포인트 값으로 표

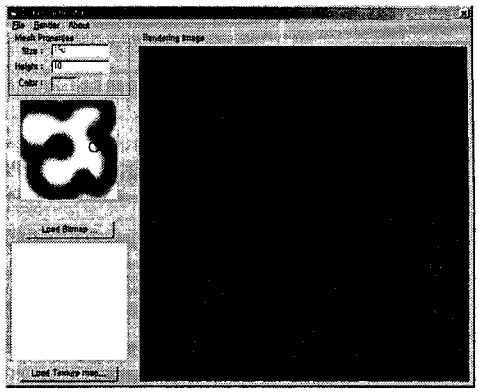

그림 8 와이어 프레임으로 보인 결과

현할 수 있다. 아래의 그림은 와이어 프레임으로 렌더링한 결과를 보이는 것이다.

4. 결론

본 논문에서는 그래픽 엔진의 구성요소인 3D 지형 생성기를 개발하였다. 기술적으로는 가장 기본적인 방법인 높이 맵을 사용하였으며, 이 기술을 적용하는 방식은 비트맵 그레이 스케일을 이용하여 적용하였다. BSP 알고리즘 방식은 내부 공간을 만들어 내는데 최적이지만 외부 지형을 만드는 데는 어려움이 따른다는 것을 알 수 있다. 외부 지형을 생성할 때는 여러 가지 방법이 있었지만 난수적 지형생성 방식은 현실감이 떨어지고, Fault 알고리즘은 자신이 원하는 지형을 만들어 낼 수 없으며 엘리베이션 그리드는 특정 애플리케이션에 한정되어 있으며 Voxel 알고리즘은 3D 비행 시뮬레이션 게임과 같은 형태에서는 적합하나 근거리로 지형을 표현할 때는 적합하지 않았다. 본 논문에서 제시한 방법은 3D 지형을 생성하는 가장 기본적인 방식이라 할 수 있지만 자신이 원하는 지형을 만들 수 있고 게임 엔진 상에서 LOD를 적용시켜 최적화를 시킬 수 있다.

우리는 우선 현재 만들어진 지형 생성기를 게임 엔진과 연동시키기 위하여 Visual C++ 6.0으로 변환할 계획이다. 또한 Visual Studio.NET과의 호환문제 또한 고려하고 있다. 계속된 게임엔진의 개발에서 DirectX보다는 OpenGL로의 전환도 고려하고 있다.

[참고문헌]

[1] 한국게임 산업개발원, "2001년도 상반기 게임개발 동향 통계 분석", 2002, http://www.gameinfinity.or.kr

[2] Mark A. Deloura, Game Programming Gems,

Charles River Media, Inc., 2000.
[4] Mark A. Deloura, Game Programming Gems, Charles River Media, Inc., 2000.
[3] Richard S. Wright, Jr. Micheal Sweet, OpenGL SuperBible second edition, Waite Group Press, 2000
[4] Andrea L. Ames, David R, Nadeau, John L, Moreland, VRML 2.0 Soourcebook second edition, John Wiley & Sons, Inc., 1997
[5] OpenGL @ Lighthouse 3D, "Terrain Tutorial" http://www.lighthouse3d.com/
[6] Mark A. Deloura, Game Programming Gems 2, Charles River Media, Inc., 2000.
[7] 박현우, 최혁종, "Real Time 3D Graphic Technique", 3D Artisan, 1999
[8] 한국게임산업개발원, 2002 대한민국 게임백서, pp575~598, 2002
[9] 신용우, "BSP를 이용한 3D Game Programming", FPS 게임 개발자 통합환경, 2002
[10] 박현우, 최혁종, "Real Time 3D Graphic Technique", 3D Artisan, 1999
[11] 염창근, "전처리된 Serialized Octree를 이용한 연속적인 쿼드트리 공간 렌더링", 제 4권 1호, pp549~552, 멀티미디어학회 춘계학술발표논문집, 2001.
[12] David H. Eberly, 3D Game Engine Design: A Practical Approach to Real-Time Computer Graphics, Morgan Kaufmann Publishers, 2000
[13] 조영식, "3D 객체의 다중해상도 모델 설계 및 구현", 강원대학교 석사학위논문, 2002.2.