韓國 컴퓨터情報學會 論文誌
第10卷 第3號, 2005. 7.

2005-10-3-4-3

Empty space BSP트리를 이용한 3D 게임 렌더링 엔진 설계

김 학 란*, 박 화 진**

3D Game Rendering Engine Degine using Empty space BSP tree

Hak-Ran Kim*, Hwa-Jin Park**

요 약

본 논문에서는 실시간 3차원 온라인 게임을 위한 게임 렌더링 엔진을 설계하고자 한다. 기존에 렌더링 속도를 높이기 위하여 대표적으로 퀘이크 엔진에서는 공간분할을 위해 BSP트리를 사용하였다. 국내에서도 급격히 증가하고 있는 3D 온라인 게임을 개발하기 위한 게임엔진이 필요하다. 현재는 고사양의 하드웨어 가속기 개발로 인해 렌더링 시간을 단축시키고 있지만 아직도 저 사양의 시스템을 사용하고 있는 게임 업체에서도 사용 가능한 소프트웨어적인 렌더링 시간 단축을 위한 엔진의 개발이 여전히 요구되고 있다. 따라서 Empty space BSP트리를 이용하여 PVS look-up테이블을 구축하여 렌더링 시간을 줄인 게임엔진을 설계하고 구현하였다.

Abstract

This paper aims to design Game Rendering Engine for real-time 3D online games. Previous, in order to raise rendering speed, BSP tree was used to partitioned space in Quake Game Engine. A game engine is required to develop for rapidly escalating of 3D online games in Korea, too. Currently rendering time is saved with the hardware accelerator which is working on the high-level computer system. On the other hand, a game engine is needed to save rendering time for users with low-level computer system.Therefore, a game rendering engine is which reduces rendering time by PVS look-up table using Empty space BSP tree designed and implemented in this paper

▸ Keyword : 3D Game Rendering Engine, BSP, PVS, Quake Engine

• 제1저자 : 김학란
• 접수일 : 2005.06.03, 심사완료일 : 2005.07.03
* 숙명여자대학교 컴퓨터과학, ** 숙명여자대학교 멀티미디어과학

Ⅰ. 서 론

3D 게임엔진은 렌더링 엔진, 애니메이션 엔진, 사운드 엔진, 인공지능 엔진, 서버 엔진 등으로 구성되는데, 렌더링 엔진은 애니메이션 기술, 장면관리 기술, GPU 기반기술, SDK기반 기술 등을 포함한다[1]. 본 논문에서는 렌더링 엔진 중 장면관리 기술에 대해서 고찰하고자 한다.

실시간 3차원 게임에서는 처리 속도가 매우 중요한 문제로 렌더링 처리 속도를 높이기 위한 다양한 방법들이 연구되고 있다. 그 중 하나가 넓은 지형과 건물 등 배경을 위한 데이터를 효율적으로 처리하여서 실행시간을 단축시키는 방법으로 3가지 사항을 고려하여야 한다. 첫번째로 3차원 게임에 사용되는 모델의 폴리곤 수를 최대한 줄여야 한다. 실사와 같은 영상을 얻기 위해서는 폴리곤의 수가 많으면 좋겠지만 게임 환경에서는 속도가 문제가 되기 때문에 될 수 있으면 폴리곤 수를 줄여야 한다. 두 번째로 미리 계산할 수 있는 것들은 미리 계산하여 처리한다. 실제로 광원의 처리 같은 경우 이를 계산하는 것은 많은 시간을 필요로 하기 때문에 실시간 3차원 그래픽에 사용하기에는 적당하지 않다. 세 번째로 공간을 나누어 렌더링 파이프 라인에 들어가는 물체 수를 줄이고 카메라의 뷰 볼륨 안에 있는 물체만 렌더링 하도록 한다. 이러한 부분들은 게임에 있어 그래픽처리 성능을 향상시키는 아주 중요한 요소이다[2][3]. 렌더링 파이프라인 안에 들어가는 폴리곤의 수를 줄이는 방법은 보이지 않는 장면인 경우 구별하여 파이프라인에 들어가지 않도록 하고, 물체가 겹쳐져 있는 경우에도 어떤 물체가 카메라 앞에 있는지 결정하는 알고리즘이 필요하다. 이런 처리 과정을 통해서 속도를 향상시킬 수 있다.

따라서 본 논문에서는 은면제거의 목적으로 고안된 BSP 트리를 3차원 가상공간을 분할하는 데 사용하여 게임 렌더링 엔진을 디자인 함으로써 성능을 높이고자 한다.

뛰어난 장면관리를 위해 대표적으로 ID소프트에서 1995년 BSP를 기반으로 퀘이크(Quake)라는 게임엔진을 개발하였다. 이후 퀘이크 엔진에서 제시한 온라인 게임 엔진의 기본적인 특징을 크게 개선한 언리얼(Unreal) 게임 엔진, 대규모 그룹 플레이를 가능하게 한 토르크(Torque) 게임 엔진, MMORPG를 위한 터바인(Turbine) 게임 엔진, NetImmerse 엔진, Genesis3D 등이 개발되었다. 국내제품으로는 게임산업 개발원의 G-blender와 한국전자통신연구원의 Dream 3D개발 이후 현재 여러 게임엔진이 개발되고 있는 실정이다[4]. 퀘이크 엔진은 렌더링 엔진에 대한 다양한 요구사항을 대부분 충족시키므로 본 논문에서 설계한 게임 렌더링 엔진의 기술 또한 기본적인 공간분할 개념은 퀘이크 엔진과 유사하다.

퀘이크 엔진이 개발되던 당시와는 달리 현재 하드웨어의 뛰어난 성능으로 말미암아 BSP트리를 사용한 장면분할로 렌더링 시간을 단축시키는 것은 과거만큼 중요하지는 않지만, 본 논문에서 설계하고 구현한 게임엔진은 저 사양의 시스템을 사용하여 게임을 개발하고자 하는 사용자들도 사용할 수 있도록 하는 것이 주된 목적이다. 또한, BSP트리를 구성하는데 있어 본 논문에서는 Empty space로 나누어 PVS의 look-up 테이블을 구성하여 렌더링 시간을 줄였으며, convex하지 않은 오브젝트를 허용하여 나누어지는 폴리곤을 최대한 줄이는 방법을 사용하고, 나누어진 폴리곤을 다시 합쳐서 렌더링 할 때 같은 위상을 가진 폴리곤들을 하나로 처리함으로써 렌더링 시간을 감축하였다.

Ⅱ. 퀘이크 엔진

게임엔진이란 용어를 보편화한 퀘이크 엔진은 1999년 소스를 공개함으로써 널리 알려졌으며 이 엔진을 사용하여 상용화된 게임은 둠(DOOM), 퀘이크3 등이다. 퀘이크 엔진에서 렌더링 엔진 분야에서는 장면관리를 위한 기술요소를 다음 <표 1>과 같이 사용하고 있다[1].

표 1 장면관리 기술요소
Table 1 Technical elements of scene management

Spatial sorting	Indoor	BSP, Potal & Cell
	Outdoor	Quadtree, Octree, View Frustum Culling
	Special Effects	Cloud, Fire, Lens flare, Lightning, Particle Systems, Water
Level of Detail	Terrain	CLOD, RAOM
	Mesh	Progressive Mesh

퀘이크 엔진에서 Indoor를 표현하기 위해 사용한 BSP 트리는 탐색시간이 O(logn)으로 빠르지만 트리의 깊이가 커지는 단점이 있다.

BSP트리를 구축하기 위해 다음 (그림 1)과 같이 폴리곤들이 배치되어 있다고 할 때, 각 폴리곤들을 노드로 보고 A를 기준 폴리곤으로 선택하여 앞에 있는 폴리곤과 뒤에 있는 폴리곤을 분류하여 트리를 구축하고 남는 폴리곤이 없을 때까지 재귀적으로 반복하면 (그림 2)와 같다. 그림에서 보듯이 폴리곤 A, K, L은 동일평면에 위치하고 폴리곤 B는 폴리곤 E를 분할하며, 폴리곤의 선택 순서에 따라 결과가 달라지게 된다. 그러나 폴리곤이 E처럼 걸쳐져 있는 경우 아래의 예와 같이 쪼개서 하나씩 각각의 그룹에 보내주는 방법은 각각의 그룹에 폴리곤을 완벽하게 포함시키게 되지만 폴리곤의 갯수가 늘어나게 되는 단점이 있다. 다른 방법은 하나의 폴리곤을 양쪽 모두에 포함시키는 방법이다. 이 경우에는 완전하지 못하므로 추가적인 테스트가 필요하며 한 폴리곤이 두번 표시되지 않도록 폴리곤을 표시했는지 여부를 각 폴리곤에 표시해 주어야 한다[5].

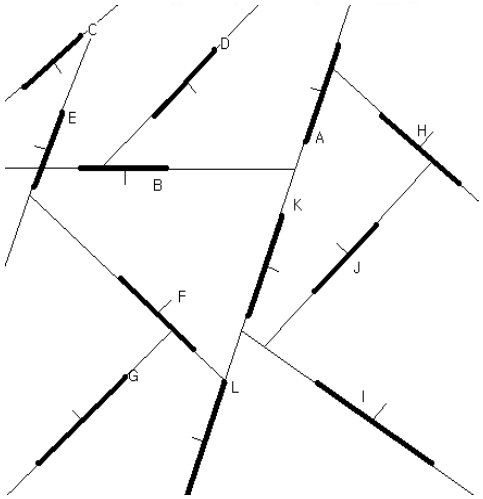
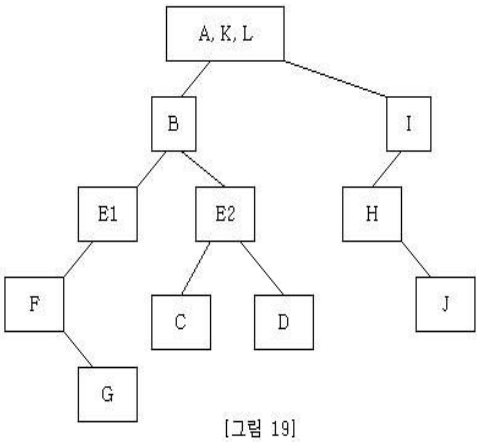


그림 1 Indoor를 형성하는 폴리곤 배치도
Fig. 1 polygon deposition to make indoor



[그림 19]

그림 2 BSP트리
Fig. 2 BSP tree

이와 같은 정보를 저장하기 위한 BSP 노드의 클래스를 정의하면 다음과 같다. BSP 노드는 2개의 연결 리스트(Linked List)를 가지고 있는데 그 중 하나는 동일 평면상에 위치하며 동일한 방향성을 가지는 폴리곤을 저장하고, 나머지 하나는 동일 평면상에 위치하지만 반대 방향성을 가지는 폴리곤을 저장한다.

```
class CBspNode
{public:float PlaneEquation[4]; // 기준 폴리곤에서 얻은 평면의 방정식
LinkedList SamePolyList; // 동일 평면, 동일 방향의 폴리곤 저장
LinkedList OppositePolyList; // 동일 평면, 반대 방향의 폴리곤 저장
CBspNode *pFrontNode; // 하위 Front 노드
CBspNode *pBackNode; // 하위 Back 노드}
```

BSP에서 원하는 영역을 트리를 타고 다니면서 렌더링하는 것은 트리가 깊어질수록 찾아내는 속도가 다소 걸리는 편이다. 물론 트리와 뷰 프러스텀만으로 렌더링은 가능하지만 영역을 찾아내는 데 걸리는 시간을 줄이기 위한 해결책의 하나가 PVS(Potentially Visablity Set)이다. PVS는 폴리곤마다 보여지고 있는 폴리곤들을 미리 계산하여 저장

하는 look-up 테이블을 만들게 된다[6]. 다음 (그림 3)은 PVS를 설명하기 위한 실내 공간이다.

<그림 9> 벽이 있는 실내 공간

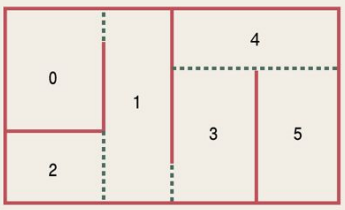


그림 3 벽이 있는 실내 공간
Fig. 3 indoor space with walls

<그림 10> 왼쪽 노드와 리프는 렌더링에서 제외된다.

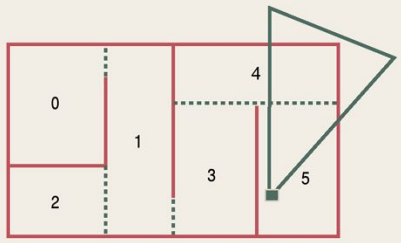


그림 4 5번 리프에 플레이어가 위치하고 있는 경우
Fig. 4 a case: player is on 5th leaf

<그림 11> 뷰 프러스텀 방식을 이용해 모든 공간을 볼 수 있다.

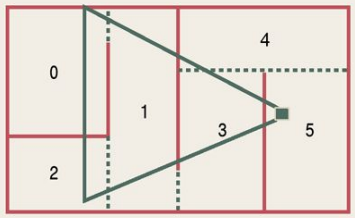


그림 5 뷰 프러스텀 방식을 이용한 공간 투시
Fig. 5 space perspective using view frustum

(그림 3)처럼 0, 1, 2, 3, 4, 5 인덱스의 총 6개 리프가 있는 하나의 BSP 맵이 있다고 가정하면 플레이어는 0~5번의 공간에 위치할 수 있기 때문에 그 공간에서 렌더링이 가능한 리프들을 테이블로 만드는 것이다. 즉 5번 리프에 플레이어가 위치했을 때는 0, 1, 2, 3은 벽에 의해 가려지기 때문에 렌더링할 필요가 없고 단지 4, 5번 리프만 렌더링하면된다. 따라서 각각의 리프 테이블들은 0번 리프 테이블의 경우 0, 1로 1번은 0, 1, 2, 3으로 2번은 1, 2, 3으로 3번은 1, 2, 3, 4로 4번은 3, 4, 5로 5번은 4, 5의 테이블을 가지고 있는 것이 바로 PVS이다.

(그림 4)를 보면 5번 리프에 플레이어가 위치해 있고 아래에서 오른쪽 위로 쳐다보고 있는 상황이라 뷰 프러스텀으로 노드와 리프의 바운딩 박스 체크를 해 나가면 왼쪽 노드와 리프는 렌더링에서 제외되므로뷰 프러스텀 방식과 PVS 방식의 속도 차이는 별로 없다. 그렇지만 (그림 5)의 경우는 결국 뷰 프러스텀 방식만으로는 안 보이는 영역까지 렌더링을 하게 된다. 이 PVS 테이블로 되어 있는 리프 중 보이는 방향에 따라 렌더링할 필요가 없는 리프도 있기 때문에 뷰 프러스텀으로 한번 더 걸러주고 렌더링하는 방법이 가장 좋은 렌더링 방법이다. 보통 BSP의 리프 수는 맵 크기와 폴리곤 수에 따라 다르지만 비교적 큰 맵들은 수천 개 정도 되므로 PVS 테이블 크기가 엄청나게 된다. 4000개의 리프가 있고 각 리프당 평균적으로 500개의 리프를 가진다면, 그리고 2바이트의 리프 인덱스라고 해도 테이블의 크기는 40005002 = 4MB이다. 그렇다면 렌더링을 해야 되는 리프들을 비트(bit)로 켜놓는 비트 플래그 방식으로 접근하면(4000/8)4000 = 2MB이다. 하지만 퀘이크 엔진에서는 RLE압축방식을 이용하여 큰 월드의 렌더링 테이블을 가진다[6].

이상과 같이 퀘이크 렌더링 엔진에서의 indoor를 표현하기 위한 중요한 기술인 BSP와 PVS에 대해 알아보았다.

III. MAD3D

본 논문에서 설계하고 구현한 렌더링 엔진은 MAD3D로 명명한다. 일반적인 3차원 모델링 툴과 달리 실시간 렌더링을 위한 다양한 기술적 요소들을 포함하고 있으며 마이크로소프트의 DirectX 8.0을 기반으로 제작되었다. 렌더링 엔진에서 제공하는 기능은 다양한 텍스쳐 효과와 공간 분할을 통한 효율적인 렌더링, 사용자와의 상호작용 기능을 지원하고 있다. 하지만 indoor에 한정된 렌더링 엔진을 설계하고 구현하였다.

퀘이크 엔진과의 주요한 차이점은 다음과 같다.

3.1 Empty space BSP트리

퀘이크 엔진을 비롯한 일반적인 게임 엔진에서는 2장에서 설명한 BSP트리에서 솔리드한 부분을 이용하여 공간분할을 하지만, MAD3D에서는 empty한 부분을 이용한 공간분할을 하게 된다. Empty space BSP트리를 구성하게 되면 PVS를 위한 테이블 구성이 용이하게 되어, 렌더링 시간을 더 단축시킬 수 있다. 다음 (그림 6)은 간단한 솔리드 오브젝트를 포함하고 있는 월드를 나타내고 있으며, 오브젝트의 각 면(face)을 따라 공간을 분할하였다. 만약 건물 내부에 움직임이 없는 정적 객체가 있다면 그 정적 객체 또한 공간 분할에 모두 포함시키게 된다. 따라서 본 엔진은 건물 내부에 정적 객체수가 적으면 렌더링 속도가 더 빨라지게 된다. 우선 모든 오브젝트의 면을 따라 먼저 분할 한 후 indoor내에 복잡하고 큰 오브젝트가 존재할 경우 최소한 폴리곤이 걸쳐지지 않도록 분할을 하게 된다. 이런 처리 과정은 주어진 공간을 균등 분할하게 되고,비 정상적인 폴리곤의 개수를 줄여서 렌더링 속도를 더 높일 수 있고, 한 개의 오브젝트가 잘라져 보이는 경우를 줄일 수 있다.

(그림7)은 (그림 6)을 바탕으로 한 BSP트리이다. (그림 7)에서 숫자 노드는 솔리드한 방향이며, 알파벳 노드는 Empty space한 방향을 나타내게 된다. 계산된 트리를 바탕으로 PVS를 구성하게 되면, 현재 위치 노드를 중심으로 인접한 노드들을 look-up테이블에 미리 저장하게 되고, 인접한 노드들은 현재 위치하고 있는 노드들에서 볼 수 있는 공간이다. 즉 현재 위치가 B라고 하면, 인접한 노드들은 A와 C가 된다.

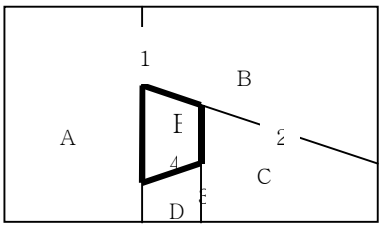


그림 6 간단한 오브젝트를 포함한 월드
Fig. 6 a world with a simple object

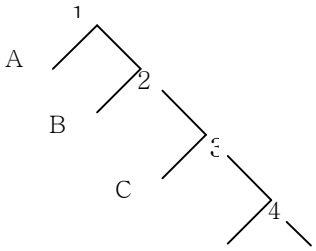


그림 7 Empty space BSP트리
Fig. 7 Empty space BSP tree

(그림 8)은 가상월드를 모델링하여 Empty space BSP트리를 구성하고 컴파일한 후 공간 분할한 면들을 흰색 선으로 나타내어 MAD3D 전용 뷰어를 이용하여 본 모습이다. 이런 와이어 프레임으로 공간분할정보들을 표현하여 줌으로써 월드를 구축하려고할 때, 모델링의 에러 검출이 용이하고, 복잡한 오브젝트의 구성을 알아 볼 때 매우 유용하다.

(그림 9)는 BSP 컴파일 후 결과 화면이다.

그림 8 BSP트리를 이용하여 분할된 가상공간
Fig. 8 partitioned virtual space using BSP tree

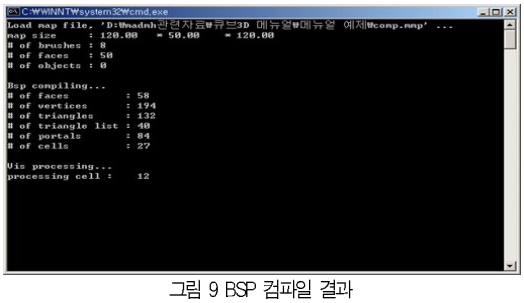


그림 9 BSP 컴파일 결과
Fig. 9 BSP compile result

3.2 월드 구축 및 매핑

MAD3D에서는 모델링 시 월드와 동적 오브젝트로 물체를 구분하게 되는데 이는 공간분할 컴파일 시 BSP트리를 구성하지 않을 물체를 동적 오브젝트로, 트리를 구성할 물체를 월드로 구분하기 위해서다. 즉, 건축물의 내 외벽이나 바닥 천정 등의 곡면이 없고 건축물의 토대가 되는 부분만을 데이터로 처리하게 된다. 본 엔진에서의 동적 오브젝트란 용어는 움직이는 문이나 동적 광원, 캐릭터 등 동적인 객체를 의미하게 되며, 월드는 게임 실행 시 한번 로딩된 후 변하지 않는 모든 객체를 의미한다. 따라서 이미 모델링 할 때부터 렌더링 시킬 부분을 구분함으로써 렌더링 시간을 단축하고자 한다.

퀘이크 엔진에서는 월드의 모델링을 위하여 자체적인 맵 에디터를 사용하고 있으나 본 논문에서의 MAD3D엔진은 오브젝트와 월드 구축을 위하여 3DS Max를 사용한다. 솔리드 모델로 그리드에 맞추어서 생성하며, 매핑의 경우는 기본 매핑 방법인 디퓨즈 맵으로 JPG 이미지 파일을 가져와 텍스쳐 매핑을 한다. 그리드에 맞추어 오브젝트를 생성하게 되면 동일 선상에 존재하는 오브젝트의 경우에 평면방정식에 따른 오차를 줄일 수 있으며, 3절에서 논의하게 될 CSG방식을 이용할 때에도 두 개의 오브젝트를 연결하여 하나의 오브젝트처럼 표현할 수 있다. 즉, (그림 11)의 왼쪽 오브젝트에서 A와 B는 점선으로 표현한 면을 서로 인접하기 때문에 하나의 오브젝트처럼 보이려면 그리드를 맞추지 않으면 떨어져 있는 오브젝트처럼 보여 지게 된다.

다음 (그림 10)은 단순한 네 개의 벽면으로 이루어진 월드를 구축한 예이다.

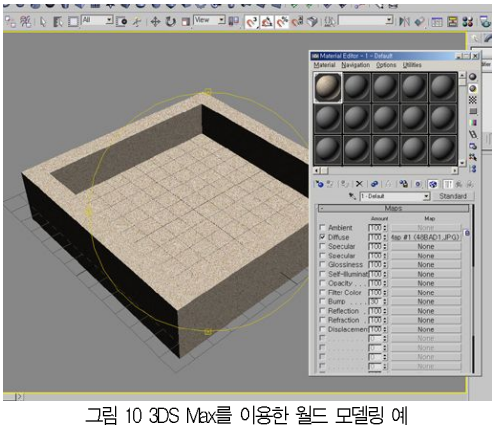
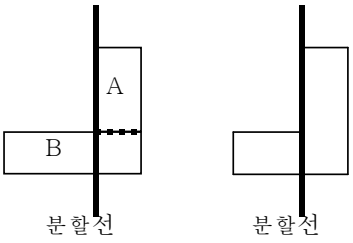


그림 10 3DS Max를 이용한 월드 모델링 예
Fig. 10 an example of world modeling using 3DS Max

3.3 Convex하지 않은 CSG(Constructive Solid Geometry) 방식의 사용

퀘이크 엔진에서는 오브젝트를 모델링하기 위하여 CSG (Constructive Solid Geometry) 방식을 사용하게 된다. 일반적인 CSG모델링은 convex한 오브젝트를 서로 조합하는 방식인데 아래 (그림 11)의 왼쪽과 같이 구부러진 오브젝트를 생성하려고 하면 퀘이크 엔진은 솔리드 BSP트리를 사용하기 때문에 두 개의 오브젝트를 연결하여야 한다. 이렇게 할 경우 복잡하고 큰 오브젝트를 위해서는 오브젝트 자체를 분할하여야 하는데 다음 (그림 11)의 왼쪽 오브젝트와 같이 분할 할 경우 결국 나뉘어 지는 폴리곤을 만들게 되는 단점이 있다. 따라서 MAD3D엔진에서는 이런 경우를 방지하기 위하여 Convex하지 않은, 즉 (그림 11)의 오른쪽 오브젝트와 같은 솔리드 모델을 허용하여 나뉘어 지는 폴리곤의 수를 최대한 줄이도록 하였다.

또한, 나뉘어 진 폴리곤의 결합 시에 만약 두 개의 폴리곤이 서로 같은 위상에 속해 있다면, 같은 위상에 속해 있는 모든 폴리곤들을 하나로 처리하게 된다.

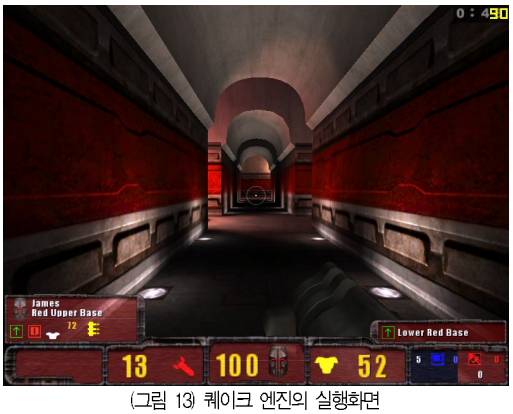


(그림 11) CSG모델링(좌)과 SG모델링(우)
(Fig. 11) CSG modeling(left) and SG modeling(right)

따라서 기존의 퀘이크 엔진에서 사용하는 솔리드 BSP트리를 변형하여 좀 더 렌더링 시간을 단축하도록 하였다.

다음 (그림 12)와 (그림 13)은 MAD3D와 퀘이크 엔진의 성능을 비교하기 위하여 동영상이나 게임의 실행 프레임 수를 계산하는 ATI Tray Tools v 1.0.2.685 프로그램을 이용한 화면 캡쳐 이미지이다. 본 논문에서 비교한 두 개의 엔진은 서로 다른 컴파일과 맵 방식을 사용하므로 사실상 같은 시간당 처리할 수 있는 렌더링 폴리곤 수를 비교하는 것이 어렵다. 따라서 같은 폴리곤 수를 가지는 월드를 구축하여 각각 실행하였을 경우의 프레임 수를 비교하였다. 구축한 월드에 사용된 폴리곤 수는 272개이며, 이 월드 모델에 대한 렌더링 성능을 비교하기 위한 프로그램 실행 후의 결과는 MAD3D엔진의 경우 초당 100프레임을 퀘이크 엔진의 경우 초당 90프레임을 처리하는 것으로 나타났다. 다만, 사용된 텍스쳐나 모델링 한 월드의 경우에 따라 조금씩 다른 결과를 보였다. 단순한 정적인 오브젝트를 가진 월드의 경우 MAD3D엔진의 성능이 더 좋은 것으로 나타났다.

(그림 12) MAD3D엔진의 실행화면
(Fig 12) execution screen from MAD3D engine



(그림 13) 퀘이크 엔진의 실행화면
(Fig 13) execution screen from Quake engine

III. 결론 및 향후 과제

온라인 3D게임이 게임 산업을 주도함에 따라 과다한 비용을 지불하며 외국엔진을 사용하여 게임개발을 하고 있는 국내 실정에서, 국내에서의 게임 개발업체들을 위한 게임 렌더링 엔진이 필요하다. 따라서 본 논문에서는 MAD3D로 명명한 indoor용 게임 렌더링 엔진을 설계하고 구현하였다. Empty space BSP를 이용한 공간분할을 통해 렌더링에 걸리는 시간을 단축시킬 수 있으며, 따라서 CPU를 능가하는 GPU를 탑재하고 있다고 하더라도 여전히 실제 게임에 사용하는 폴리곤의 개수가 늘어가는 추세이므로 시스템사양에 상관없이 렌더링 되는 폴리곤의 개수를 줄이는데 사용될 수 있다.

또한 기존의 방법과 비교하여, 다소 제약이 따르지만 상용화되어 있는 모델링 툴인 3DS Max를 사용할 수 있으며, convex하지 않은 솔리드 객체를 허용하고, 나누어진 폴리곤을 재결합 할 때 위상이 같은 폴리곤들은 하나로 처리함으로써 공간분할에 걸리는 시간을 더 줄일 수 있도록 하였다.

하지만 indoor를 위한 렌더링 엔진이므로 건물 내에서 진행되는 게임에서만 적합하다.

향후 과제로는 MAD3D엔진을 사용한 간단한 데모 게임을 제작하여 가장 적합한 게임형태에 대한 비교 검증이 필요하며, 또한 동적 객체에 대한 처리를 더 추가하여 정적 객체와 동적 객체를 모두 관리할 수 있는 공간분할에 대한 연구와 구현이 필요하다.

참고문헌

[1] 장희동, 김경식, 호서 3D 온라인 게임 엔진 개발, 공업기술 연구소 논문집, pp. 51-61, VOL 22, 2003

[2] Bryan turner, Real-Time Dynamic Level of Detail Terrain Rendering with ROAM, Gamasutra URL:http://www.9amasutra.com/features/20000403/turner_01.html

[3] 이승욱,박경환, 3차원의 넓은 지형처리를 위한 CLOD가 적용된 옥트리 분할 기법, 정보기술연구소 논문집,9권2호, 2002

[4] 이만제, 온라인 게임 기술 동향, 정보과학회지, pp.12-18, 제20권 1호, 2002

[5] David H. Eberly, 3D Game Engine Design , A practical Approach to Real-Time Computer Graphics, MORGAN KAUFMANN PUBLISHERS, San Diego, CA, 2001

[6] 장언일, 3D 엔진제작의 실제, pp. 168-175, 마이크로소프트 12월호, 2003

[7] 박태준, 표순형, 추창우, 최병태, 3D게임용 렌더링 엔진의 제작, HCI 2002 , 2002

[8] 신용우, BSP를 이용한 3D Game Programming, URL: http://www.gamesync.com.ne.kr

[9] Tomas Akenine -Moller, Eric Haines, Real-Time Rendering, AK Peters Ltd , 200

저 자 소 개

김 학 란

1983.2. 숙명여자대학교 전산학과 졸업(학사)

2003.2 숙명여자대학교 정보통신대학원 졸업(석사)

2004.3~현재 숙명여자대학교 대학원 컴퓨터과학 박사과정

한성대학교 멀티미디어공학과 겸임교수

<관심분야> 2D,3D 컴퓨터 그래픽스, 게임, 가상현실, 애니메이션

박 화 진

1983.2. 숙명여자대학교 전산학과 졸업(학사)

1989.2 숙명여자대학교 대학원 전산학과 졸업(석사)

1997년 Arizona State University Computer Science 컴퓨터 그래픽 전공 (공학박사)

1997년-1998년 삼성 SDS 연구소 선임 연구원

1998년-2000년 평택대학교 전임강사

2000년-현재 숙명여자대학교 멀티미디어과학과 교수

<관심분야> 컴퓨터 그래픽, 게임, 3D모델링, 가상현실, 멀티미디어