Journal of Korea Multimedia Society Vol. 11, No. 1, January 2008(pp. 88-95)

이진공간분할(BSP)과 잠재적가시공간(PVS) 알고리즘을 이용한 실내렌더링 속도개선 방법

김병선[†], 권순각[††], 김성우[†††], 이중화[††††], 정재진[†††††]

요 약

본 논문에서는 이진공간분할(BSP), 잠재적가시공간(PVS) 알고리즘을 이용한 효과적인 실내렌더링 방법을 제시한다. 제안된 방식은 BSP 알고리즘과 PVS 기법을 동시에 이용하여 BSP 영역번호를 나누는 과정에서 BSP 영역에 오브젝트를 추가로 포함시킴으로써 오브젝트를 효과적으로 출력시킨다. 또한, 플레이어 위치를 중심으로 PVS를 검사하여 필요한 부분만 출력시킨다. 실험 결과로 부터 제안된 기법은 BSP와 PVS를 경계영역뿐만 아니라 오브젝트도 같이 처리함으로써 렌더링 속도가 향상됨을 보인다.

An Improvement Method of Indoor Rendering Speed Using BSP and PVS Algorithms

Byoung-sun Kim[†], Soon-kak Kwon[††], Seong-woo Kim[†††], Jung-hwa Lee[††††], Jai-jin Jung[†††††]

ABSTRACT

In this paper, we present an efficient indoor rendering method using BSP(Binary Space Partitioning) and PVS(Potentially Visible Space) algorithms. The proposed method displays efficiently the objects by including the objects in the process of separating the number of BSP area, by jointly using BSP and PVS algorithms. Also, the screen which is checked by PVS from player's location is displayed. From the simulation, we can see that the proposed method improves the rendering speed because of processing the objects with the edges of BSP and PVS.

Key words: BSP(이진공간분할), PVS(잠재적가시공간), Indoor Rendering(실내렌더링)

1. 서 론

게임산업이 2D기반에서 3D로 넘어감에 따라 2D에서 사용되던 화소방식은 벡터방식으로 출력이 바뀌게 되었다. 벡터방식의 출력은 부동소숫점 연산을 이용하게 되는데 부동소숫점 연산은 일반 정수 연산에 비해서 처리하는데 많은 시간이 요구된다. 그래서 연산을 최소화시킬 방법의 필요성이 대두되었으며 BSP(Binary Space Partitioning)와 PVS(Potentially Visible Space) 같은 방식이 사용되게 되었다[1-6].

※ 교신저자(Corresponding Author) : 권순각, 주소 :부산시 부산진구 엄광로 995(614-714), 전화 : 051)890-1727, FAX : 051)890-2629, E-mail : skkwon@deu.ac.kr
접수일 : 2007년 10월 2일, 완료일 : 2007년 11월 12일
[†] 동의대학교 컴퓨터소프트웨어공학과
(E-mail : soul0613@nate.com)
[††] 종신회원, 동의대학교 컴퓨터소프트웨어공학과
[†††] 정회원, 동의대학교 컴퓨터소프트웨어공학과
(E-mail : libero@deu.ac.kr)
[††††] 정회원, 동의대학교 컴퓨터소프트웨어공학과
(E-mail : junghwa@deu.ac.kr)
[†††††] 정회원, 동의대학교 디지털문화콘텐츠공학과
(E-mail : dothan@deu.ac.kr)
※ 본 연구는 2006학년도 동의대학교 교내연구비에 의해 연구되었음

BSP는 공간을 분할하는 정보를 담고있는 2진 트리를 의미한다. 처음 BSP 알고리즘은 월드 안에 폴리곤들을 정렬할 목적으로 개발되었다. 개발 당시에는 그래픽 카드에 하드웨어 가속 Z버퍼가 존재하지 않았고 소프트웨어 버퍼링은 너무 느렸기 때문에 BSP 정렬이 필요하였다. 오늘날에는 하드웨어 가속 Z버퍼가 존재하기 때문에 이러한 기능으로 BSP가 사용될 필요가 없다. 대신에 BSP는 다른 부분들의 최적화에 다양하게 쓰이고 있다. 실제로 BSP를 이용하여 속도를 높인 첫 게임은 존 카멕과 존 로메로가 개발한 DOOM이었다. 그 이후로는 대부분의 FPS (First Person Shooting) 게임이 이 기술을 사용하게 되었다[7].

PVS는 실제로 눈에 보이는 부분만 조사해서 출력할 수 있게 해주는 기법이다. 하지만 순수 BSP와 PVS같은 기법은 실제로 실내렌더링을 수행하는데 최적화되어 있지 않다. 그래서 BSP와 PVS를 이용하여 실내렌더링에 적합한 방식 연구가 필요하게 된다.

제안된 방식은 BSP 영역에 오브젝트를 삽입함으로써 오브젝트가 PVS와 BSP의 영향을 받게 만들어서 최종단계에서 비교할 대상의 수를 줄인다. 비교대상의 수를 줄여서 렌더링 속도향상을 시켜줄 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 BSP 알고리즘에 대해서 알아보고, 3장에서는 PVS에 대해서 알아보며, 4장에서는 제안된 방식에 대해서 서술한다. 그리고 5장에서는 제안한 방식을 구현 및 성능평가를 하고, 6장에서 결론을 맺는다.

2. BSP(Binary Space Partitioning) 방식

BSP 트리의 생성[8]에 대한 기본 아이디어는 장면(scene)의 일부인 폴리곤의 집합을 취하고 그것들을 더 작은 집합으로 분할하는 것이다. 여기에서 각 하위집합은 convex(凸 요철의) 폴리곤의 집합니다. 즉 이 하위집합의 각 폴리곤은 같은 집합 내의 모든 다른 폴리곤의 앞쪽에 있다는 것이다. 예를 들면, 폴리곤 1의 모든 점이 폴리곤 2의 평면과의 거리 값이 양수라면 폴리곤 1은 폴리곤 2 앞에 있다고 할 수 있다. 안쪽을 바라보는 폴리곤으로 구성된 육면체(cube)는 convex 집합이지만, 바깥쪽을 바라보는 폴리곤으로 구성된 육면체는 그렇지 않다.

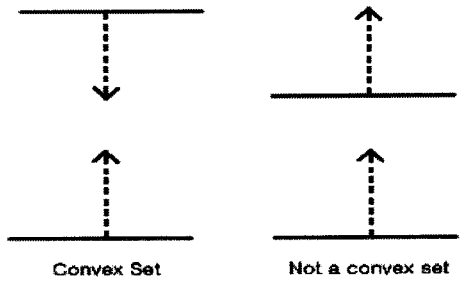

그림 1. convex 집합과 non-convex 집합의 차이

구현을 위해서는 폴리곤의 집합이 그림 1과 같은 convex집합인지 판단하는 과정이 필요하다. 이 과정에서 요구되는 기술은 폴리곤과 점의 위치 판단, 두 폴리곤의 앞뒤 판단, 최적 분할폴리곤 생성 등의 기술이 요구된다.

폴리곤과 점의 위치판단은 3개의 정점으로 평면의 방정식을 만들고 평면의 방정식에 점을 대입해서 그 값이 양수인지 0인지 음수인지를 체크하여 점이 평면의 위치를 파악한다. 두 폴리곤의 위치 판단은 비교할 폴리곤과 기준 폴리곤을 정하고 기준폴리곤으로 평면방정식을 만들고 비교할 폴리곤의 정점들과의 위치판단으로 통해서 비교할 폴리곤이 기준폴리곤의 앞에 있는지 뒤에 있는지 판단한다. 최적 분할폴리곤 생성은 두 폴리곤의 위치 판단할 때 만약 폴리곤이 교차할 경우 교차하는 면을 두 개로 분리시킨다.

BSP 트리를 위해 필요한 구조체는 다음과 같이 정의될 수 있다.

```
class BSPTree
{
  BSPTreeNode RootNode   // 트리의 루트 노드
}

class BSPTreeNode
{
  BSPTree Tree  // 이 노드가 소속되어 있는 트리
  BSPTreePolygon Divider  // 두 개의 하위 트리의
                            중간에
                          // 놓여 있는 폴리곤
  BSPTreeNode *RightChild // 이 노드의 오른쪽 하
                            위 트리
  BSPTreeNode *LeftChild  // 이 노드의 왼쪽 하위
                            트리
  BSPTreePolygon PolygonSet[] // 이 노드에서의 폴
                                리곤 집합
}
```

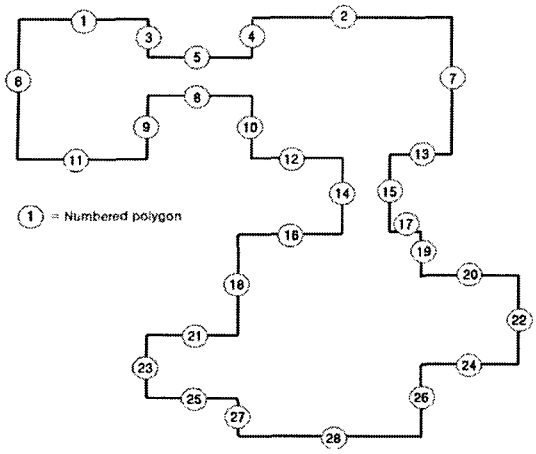

그림 2. 폴리곤의 집합

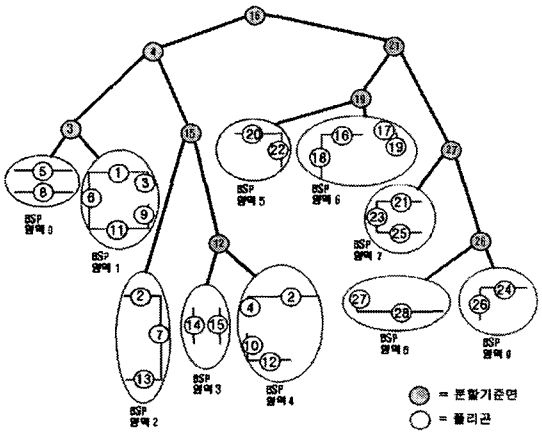

그림 3. BSP 트리 구조

BSP 트리 구조체[7]를 기본으로 하고, 그림 2의 폴리곤 집합을 BSP트리로 생성한다고 가정하면, 최종단계는 그림 3과 같은 BSP 트리가 만들어진다.

3. PVS(Potentially Visible Space) 방식

PVS방식은 FPS게임을 만들 때, 일반적으로 사용되는 비가시면 제거 기법이다. 비가시면 제거 기법에는 포탈 렌더링방식이 있다. 이 기법은 BSP 트리의 이점을 이용하는 데 있어서 매우 적절하다. 또한 포탈 렌더링은 BSP로는 수행할 수 없는 거울효과(mirror)와 감시 카메라(surveillance camera)와 같은 좋은 부수 효과들을 포함하고 있다.

포탈 렌더링 방식은 3D 게임상의 세계를 다른 곳과 연결된 포탈(입구, portal)을 이용하여 여러 가지 섹터로 나누어서 적절히 출력하는 방식이다. 섹터는 convex 이며 닫혀진(closed) 폴리곤 집합이다. 여기에서 닫혀졌다는 의미는 폴리곤에 부딪히지 않고서는 섹터에서 섹터의 바깥쪽으로 라인을 그릴 수 있는 방법이 없다는 것이다. 이것은 각 노드에 있는 각 홀(hole)이 반드시 포탈 폴리곤으로 채워져야만 한다는 것을 의미한다. 포탈 폴리곤의 배치는 수작업으로 하거나 자동으로 할 수 있다.

포탈 렌더링의 구현방법은 프러스텀을 가지고 관찰자의 위치에서부터 장면을 렌더링할 때, 그것이 포탈 폴리곤에 부딪히면 포탈은 프러스텀을 깎아낸다. 그리고 나서 같은 관찰자의 위치로부터 새로운 프러스텀을 가지고서 인접 섹터가 렌더링된다. 이것은 매우 간단한 접근이며, 재귀 함수에 적절하다. 보일 수 있는 오브젝트들은 쉽게 구별할 수 있다. 왜냐하면 장면 프러스텀은 플레이어가 보게 되는 것과 정확히 동일하게 제한되어 있기 때문이다.

그림 4에서 관찰자의 위치는 V이며, 원래의 프러스텀은 F1이다. F1이 포탈 폴리곤 P1을 만나면, 그것은 절단되고 F2로 재명명된다. 나중에 F2가 포탈 P2와 P3를 만나면 그것은 F3 와 F4로 절단된다. 포탈 P4와 부딪히면, F3은 F5로 깎여 나가고, F4는 F6으로 깎여 나간다. 이 과정은 재귀 함수로 이루어진다[9].

포탈 렌더링에 있어 중요한 문제점은 포탈의 배치이다. 포탈을 수작업으로 배치하는 것은 맵 설계자에게 기술과 매우 많은 시간을 요구한다. 따라서 자동 포탈 배치를 위한 좋은 알고리즘이 필요하다.

보통은 트리 안의 각 포탈이 반드시 트리의 분할 폴리곤에 의해서 정의된 면과 일치해야 한다. 이러한 각 면들의 외부에서 포탈 폴리곤이 생성되면, 그것은

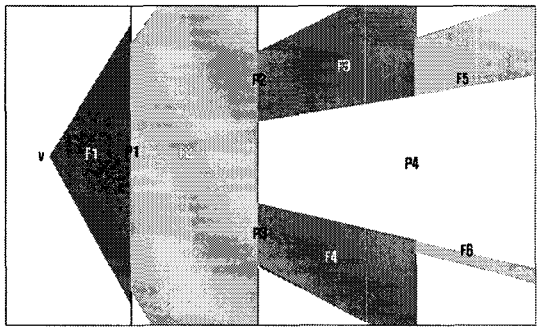

그림 4. 시점 프러스텀 절단

초기에는 그것이 위치한 노드의 경계 상자(bounding box)보다 큰 네 변을 가진 폴리곤이다. 그리고 나서 각 포탈 폴리곤은 그것이 위치한 노드의 하위 트리로 밀어 넣어진다. 포탈 폴리곤이 그것의 하위 트리 중 하나의 노드를 통해 전달되면, 그 노드의 분할 폴리곤에 의해 정의된 면이 그것을 절단하게 되며, 그 노드가 단말(leaf)이라면 그 노드에 있는 폴리곤들에 의해서도 절단된다. 폴리곤이 절단되면, 두 개의 결과 조각이 트리의 최상위로부터 아래로 보내진다. 포탈 폴리곤이 절단되지 않는다면, 그것은 현재 방문 중인 노드의 하위 트리로 전해지게 된다. 이것은 만약 그것이(포탈 폴리곤) 면의 앞쪽 방향에 있다면 그것은 오른쪽 하위 트리로 전달될 것이며, 뒤쪽 방향에 있다면 그것은 왼쪽 하위 트리로 전달될 것임을 의미한다. 만약 현재 노드의 분할 폴리곤에 의해 정의된 면과 일치한다면, 그것은 양쪽 하위 트리로 보내지게 된다.

4. 제안된 방식

본 장에서는 제안된 방식에 대해서 설명한다. 제안된 방식은 만들어진 BSP 영역에 오브젝트도 포함시킴으로써 렌더링시 검사할 오브젝트 개수를 줄이고 불필요한 오브젝트 렌더링을 줄임으로 속도개선 효과를 가져온다. 이를 위하여 BSP 영역을 분리하고, BSP 영역 경계면에 대한 포탈번호를 설정한다. 그림 5는 그림 2의 폴리곤 집합에 적용하여 BSP 영역을 나누는 직접적인 예를 나타낸 것이다. BSP 영역은 포탈에 의

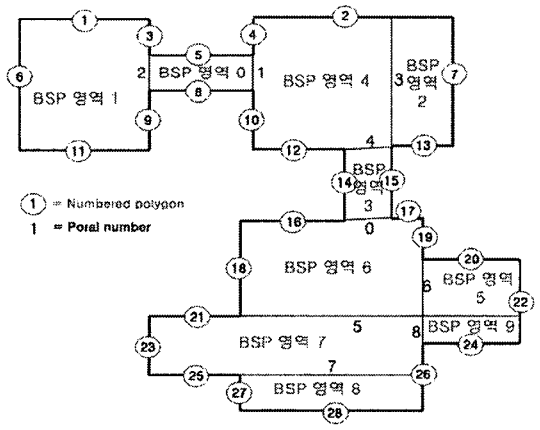

그림 5. 폴리곤 집합에 BSP 알고리즘을 이용하여 영역을 분할한 모습

표 1. 포탈 테이블

포탈 번호	BSP 영역번호	BSP 영역번호
0	3	6
1	0	4
2	0	1
3	2	4
4	3	4
5	7	6
5	9	5
6	6	5
7	7	8
8	7	9

하여 서로 구분되어 나누어질 수 있다. 각 포탈에 인접한 BSP 영역 번호를 나타내면 표 1과 같다.

구해진 포탈테이블과 전체 BSP 트리를 이용하여 렌더링을 수행한다. 그림 6에서와 같이 경계면을 출력할 때 처음 주어진 공간 번호를 인자 값으로 넘겨받게 되고 그 공간의 경계면을 출력하게 된다. 그리고나서 그 공간에 오브젝트를 검사한 후 가시거리 안에 있다면 그 오브젝트를 출력하게 된다. 그 다음으로 그 영역에 포탈이 있는지 검사하고 만약 그 영역에 포탈이 있다면 그 포탈이 가시영역에 들어왔는지 검사한 후 가시영역 안에 있다면 포탈테이블을 참조하여 연결된 영역이 어디인지 알아낸다. 이상의 과정이 반복 수행하게 된다.

그림 6의 경계면 출력 부분에 대한 세부동작 순서도는 그림 7과 같다.

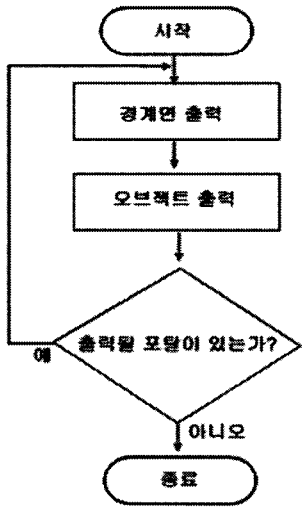

그림 6. 렌더링 순서도

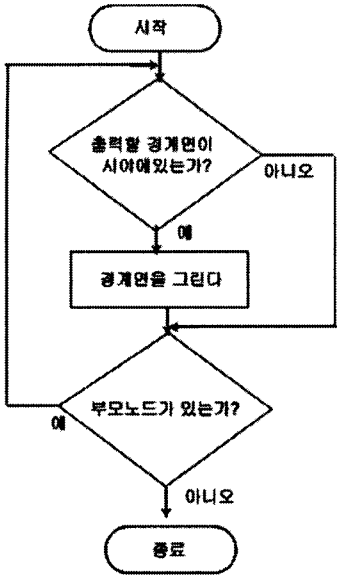

그림 7. 경계면 출력 순서도

그림 7에서는 처음 출력할 경계면이 가시영역에 들어오는지 검사한 후 가시영역에 들어오면 그 노드의 경계면 정보(텍스처, 버텍스)를 읽어와서 DIRECT 3D 출력 설정을 한 후 출력시키고, 그리고 나서 BSP 노드의 부모노드를 재귀적으로 호출하여 부모노드가 없을 때까지 계속해서 반복한다.

그림 8은 깊이 우선 탐색 알고리즘을 이용하여 BSP 영역 생성과정을 보여주는 순서도이다. 재귀 함수로 구성되어 있으며, 우선 BSP 노드 값이 null값인지를 검사한다. null값이라면 작업을 더 이상 진행할 필요가 없기 때문에 return을 호출하게 되며, null값이 아니라면 우선 왼쪽 자식 노드부터 점검을 하게 된다. 그리고 다음으로 오른쪽 자식노드를 점검하게 되고, 그 다음으로 양쪽 자식노드를 점검하는데 두 자식노드 모두가 null값이라면 이 노드를 최말단(리프) 노드라고 정하고 이 노드를 BSP 영역에 설정한다. 최말단 노드 하나당 영역 노드는 하나씩 생성된다. 최말단 노드 개수만큼 BSP 영역의 개수가 나온다. 생성된 BSP 영역을 바탕으로 오브젝트들은 BSP 트리를 이용하여 각 BSP 영역에 데이터가 저장된다. 그림 5에 오브젝트를 추가한 모습은 그림 9와 같다.

그림 9는 제안된 방식을 적용한 경우에 대한 예를 나타낸 것이다. 총 오브젝트가 10개이며 검사해야할 오브젝트는 플레이어의 시야에 들어오는 BSP 영역 0과 BSP 영역 1의 오브젝트 3개이며, 3개의 오브젝

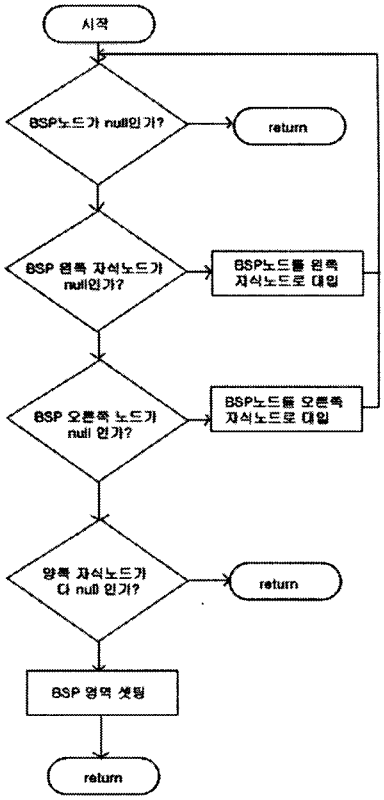

그림 8. BSP 영역 생성 순서도

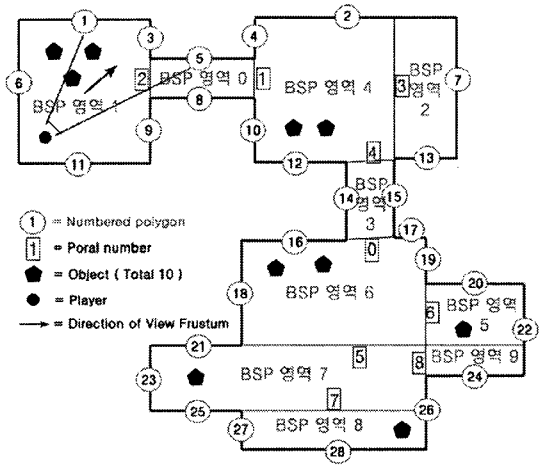

그림 9. 오브젝트를 포함한 제안된 방식

트만 검사하면 된다. 그림 9를 보면 그 중 출력되는 것은 2개이다. 만약 제안된 방식을 적용하지 않을 경우에는 플레이어의 시야에 들어오는 오브젝트 2개를 출력하기 위해서 10개의 오브젝트를 모두 검사해야 한다. 제안된 방식과 그렇지 않는 방식의 출력된 오

브젝트 개수는 서로 같지만, 제안된 방식은 오브젝트 검색 횟수를 크게 줄일 수 있음을 알 수 있다. 그림 9와 같은 경우는 제안된 방식은 3번의 검색횟수를 갖지만 그렇지 않을 경우에는 10번의 검색이 필요하게 된다.

5. 구현 및 성능평가

본 장에서는 실제로 제안된 방식을 구현하고 성능을 평가한다. 자체 개발한 Engine_Type _X 게임엔진[10]에 제안된 방식이 구현되어 있다.

5.1 구현

그림 10에서처럼 Engine_Type_X는 렌더링부분(파티클,UI)과 사운드엔진과 프레임워크를 이용한 애니메이션 엔진으로 구성되어 있다. 전반적인 처리는 Engine_Type_X에서 각각의 엔진들을 초기화시킨다. 그 후 각각의 엔진들은 Engine_Type_X 안에서 유기적으로 서로 연결되어 동작하게 된다. 예를 들면 어떤 처리를 수행할 때 하나의 엔진이 필요한 것이 아니라 2개 이상의 엔진이 같이 동작해야 하는 경우가 많기 때문에 그 부분들은 Engine_Type_X에서 총괄 관리하여 각각의 엔진들에게 필요한 부분에 대해서 명령을 내린다.

각각의 엔진들은 Engine_Type_X에서 받은 명령을 효과적으로 실행시킨다. 그림 11은 BSP가 적용이 된 xMap(지형) 엔진에 대해서 설명한다.

지형 엔진은 맵에디터에서 생성된 지형정보와 BSP 정보를 읽어들여서 출력하는 역할을 담당한다.

그림 11과 같이 맵에디터에서 지형 및 오브젝트

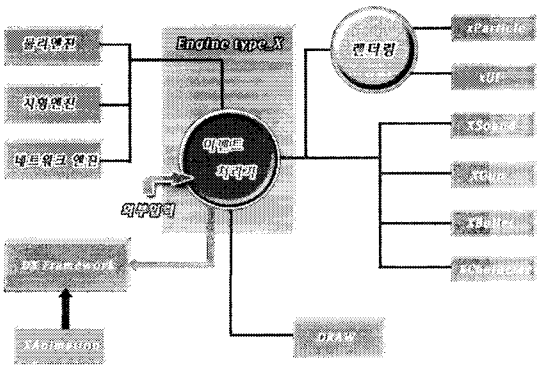

그림 10. BSP 영역 생성 순서도

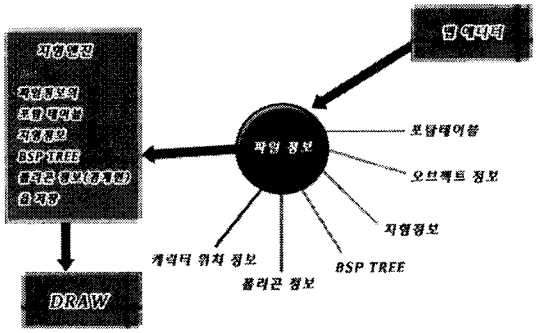

그림 11. 지형 엔진 흐름도

캐릭터 위치, 경계벽 등을 만들면 그 정보는 파일로 저장된다. 저장된 파일은 지형엔진에서 그 파일을 바탕으로 지형과 객체, 캐릭터 위치 등의 정보를 읽어들이고 그 정보를 바탕으로 BSP 영역을 생성한다. 그리고 생성된 영역을 바탕으로 지형의 출력을 향상시킨다.

5.2 성능평가

실험 환경으로는 Windows XP SP2, 펜티엄4 2.8G CPU, 1GB RAM, GeForce 5700LE에서 실험하였다. 자체 개발한 Engine_Type_X 실험엔진을 이용하여 제안된 방식 성능을 평가하였다.

그림 12는 게임엔진의 구동화면으로서 실제 지형에 오브젝트를 30개 띄운 상태에서 제안된 방식을 적용시킨 모습을 캡처한 것이다. 현재 공간의 번호는 0번이며 오브젝트는 0개이지만 실제로 보이는 오브젝트는 12개 이며 또한 BSP 알고리즘을 이용하여 현재 14개의 경계면만을 출력하고 있다.

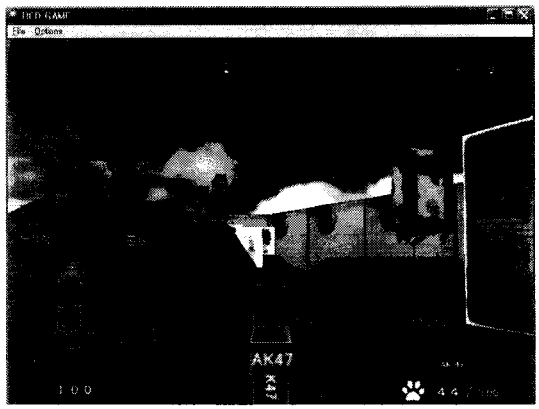

그림 12. Engine_Type_X 게임엔진 구동 화면

표 2는 제안방식과 기존 방식에 대한 성능을 비교한다. 오브젝트 개수는 20, 30, 50, 100로 나누고 BSP 영역은 0~5로 나누었다. BSP 영역별로 렌더링할 경우 재생되는 프레임 수를 나타낸 것으로서, 오브젝트 개수가 20인 경우에 제안된 방식은 전체평균 초당 100 프레임이 재생되지만 기존방식(미적용 방식)은 초당 96.1프레임이 재생됨을 알 수 있다. 즉 평균 3.9 프레임에 대한 재생 이득을 얻는다. 그리고 오브젝트 수가 30개일 경우는 4 프레임, 50개일 경우는 4.4프레임, 100개일 경우는 10.6 프레임으로 제안된 방식으로 인한 재생 프레임수가 증가됨을 알 수 있다.

그림 13은 오브젝트 개수에 따른 재생 프레임 수의 변화 정도를 나타낸다. 오브젝트 개수가 100개 넘어가면서 재생 프레임 수의 차이가 많아지는 것을 알 수 있다. 알고리즘의 복잡도가 있기 때문에 적은 개수의 오브젝트에서는 크게 차이가 나타나지 않지만 오브젝트 개수가 많아질수록 성능의 증가 폭이 커짐을 알 수 있다.

표 2. 오브젝트 개수에 따른 재생 프레임 수

오브젝트 개수	20		30		50		100	
BSP 영역번호	제안방식	미적용	제안방식	미적용	제안방식	미적용	제안방식	미적용
0	107	103	106	103	106	104	105	95
1	95	89	94	88	94	85	90	70
2	96	94	96	94	95	93	98	86
3	98	95	97	94	96	94	95	88
4	99	97	97	94	97	94	95	87
5	105	99	103	96	103	95	97	90
전체평균	100	96.1	98.8	94.8	98.5	94.1	96.6	86

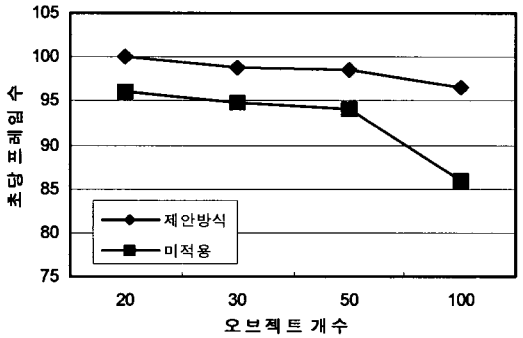

그림 13. 오브젝트 개수에 따른 재생 프레임 수 변화

6. 결 론

실내렌더링은 게임 개발에 있어서 아주 중요한 분야로 자리잡고 있다. 여러 가지 복잡한 기술들이 이 알고리즘과 접목하여 효율적인 처리를 수행하고 있다. 본 논문에서는 BSP 알고리즘과 PVS 기법을 이용하여 렌더링 속도를 향상시키는 방법을 제안하였다. 성능평가로 부터 제안된 방식은 오브젝트수가 50개일 경우는 약4 프레임, 100개일 경우는 약11 프레임으로 재생 프레임수가 증가되어 렌더링 속도를 크게 향상시킴을 알 수 있었다. 이러한 제안된 방식은 게임엔진의 렌더링 부분에서 충분히 활용 가능할 것으로 기대된다.

참 고 문 헌

[1] 김용준, *IT EXPERT 3D 게임 프로그래밍*, 한빛미디어, 서울, 2003.

[2] Kelly Dempski-Andre LaMothe, *DirectX 실시간 렌더링 실전 테크닉*, 정보문화사, 서울, 1996.

[3] 장세찬, *Gof 디자인 패턴! 이렇게 활용한다*, 한빛미디어, 서울, 2004.

[4] 문기영, *게임 개발 테크닉*, 정보문화사, 서울, 2002.

[5] Mark Deloural, *Game Programming Gems*, 정보문화사, 서울, 2001.

[6] Mark Deloural, *Game Programming Gems 2*, 정보문화사, 서울, 2002.

[7] Samuel Ranta-Eskola, http://www.devmaster.net/articles/bsp-trees/.

[8] Daniel Sanchez-Crespo Dalmau, *게임 프로그래밍의 핵심 테크닉과 알고리즘*, 사이텍미디어, 서울, 2005.

[9] Samuel Ranta-Eskola, Hidden Surface Removal, http://www.devmaster.net/articles/hidden-surface-removal/.

[10] 김병선, 김동형, 박경수, 권순각, 권오준, "3D 온라인 게임엔진의 구현," 한국멀티미디어학회 춘계 학술대회, 2007.

김 병 선

2008년 2월 동의대학교 컴퓨터소프트웨어공학과 졸업
관심분야 : 3D 게임엔진, Direct 3D, Shader, 자료구조, 알고리즘

권 순 각

1990년 2월 경북대학교 전자공학과 졸업
1992년 2월 KAIST 전기및전자공학과 석사
1998년 2월 KAIST 전기및전자공학과 박사
1997년 3월~1998년 8월 한국전자통신연구원 연구원
1998년 9월~2001년 2월 기술신용보증기금 기술평가센터 팀장
2003년 9월~2004년 8월 Univ. of Texas at Arlington 교환 교수
2001년 3월~현재 동의대학교 컴퓨터소프트웨어공학과 교수
관심분야 : 멀티미디어신호처리, 영상통신

김 성 우

1991년 2월 KAIST 전기및전자공학과 공학사
1993년 2월 KAIST 전기및전자공학과 공학석사
1999년 2월 KAIST 전기및전자공학과 공학박사
1999년 3월~2002년 2월 한국전자통신연구원 선임연구원
2002년 3월~현재 동의대학교 컴퓨터소프트웨어공학과 조교수
관심분야 : 임베디드그래픽스, 센서 네트워크, 지능 제어

이 중 화

1992년 2월 부산대학교 전자계산학과 졸업
1995년 2월 부산대학교 전자계산학과 석사
2001년 8월 부산대학교 전자계산학과 박사
2002년 3월~현재 동의대학교 컴퓨터소프트웨어공학과 교수
관심분야 : 데이터베이스, 시맨틱 웹, 한글정보처리 등

정 재 진

1990년 성균관대학교 문과대학 독어독문학과 학사
1996년 연세대학교 행정대학원 행정학 석사
2004년 성균관대학교 경영대학원 경영학 박사
1994년~1998년 신세기통신 기획실 전략기획팀 차장
1998년~2001년 정보통신연구진흥원 인력양성사업팀장
2001년~2005년 한국소프트웨어진흥원 디지털콘텐츠사업팀장
2005년~2005년 8월 동신대학교 디지털콘텐츠학과 조교수
2005년 9월~현재 동의대학교 영상정보대학 디지털문화콘텐츠공학과 조교수
연구관심분야 : 콘텐츠 기획.마케팅 전략, 온라인게임 개발전략, IT R&D Management