2002년 한국정보처리학회 춘계학술발표논문집 제9권 제1호

게임엔진의 공간 분류기법 비교분석

장인걸, 허원, 권기달, 신동규, 신동일
세종대학교 컴퓨터공학과
e-mail : {ig8961,heowon,kinight1976,shindk,dshin}@gce.sejong.ac.kr

Comparative Analysis of Spatial Sorting technologies in Game Engines

In-gaul Jang, Won Heo, Ki-Dal Kwon ,Dongil Shin, Dongkyoo Shin
Dept. of Computer Engineering, Sejong University

요 약

본 논문에서는 게임엔진에서 쓰이는 공간 분류기법의 종류인 BSP, 포털, PVS 에 대한 기본 개념과 각각의 장단점을 설명하였다. 그리고 상용게임엔진에서의 공간 분류기법을 비교하고 여러게임엔진에서 어떤 공간 분류기법이 사용되는지를 논술한다.

1. 서론

장면을 렌더링 하는 과정은 객체들의 깊이 우선순회에 의해서 결정이 되지만 장면이 복잡하고 시점이 들어갈때는 깊이 우선순회에 의해서는 장면을 제대로 그릴수가 없고 부정확하게 그려질 것이다. 공간 분류에서 기본이 되는 개념은 화면에 하나의 픽셀에 대해서 여러 번 그려지는 것을 피하는 것이다.

" Culling"은 객체의 부분들을 제거하는 것을 뜻하는 용어로서, 전체 객체가 될 수도 있으며 제거되는 부분들은 시점에서 볼 수 없는 부분이다[1].
게임에서 정확한 충돌 감지나 빛과 개체의 교차와 같은 어려운 문제점들은 공간적 관계의 문제이다. 폴리곤의 테두리 표현으로 정의된 개체가 그 주변 개체에 대해 어디에 있는지 알아야 한다.

복잡하고 느린 알고리즘을 통해서 명시적으로 그 위치를 알아 낼 수 있다. 예를 들어, 공간을 가로질러 뻗는 한 줄기 빛이 일련의 폴리곤 중 어느 하나에 비추는 것을 보려 한다고 하자. 이 동작을 수행하는 느린 방법은 빛이 비친 각각의 모든 폴리곤을 명시적으로 테스트 하는 것이다. 폴리곤과 빛의 교차는 단순한 적업이 아니다. 따라서 몇 천개의 폴리곤이 있는 경우 알고리즘의 속도는 급격히 감소할 것이다. 그러나 폴리곤의 공간적 관계가 많은 도움을 줄 수 있다. 광선이 이 폴리곤을 비추지는 않지만 전체 광선이 완전하게 폴리곤이 존재하는 면 앞에 있다고 말할 수 있다면 첫 번째 폴리곤의 뒤에 있는 어떤 것도 테스트 할 필요가 없다.

다시 말해서 공간 분류기법은 게임에서 정확하고 빠른 렌더링 처리를 하기 위해 필요하다. 본 논문에서는 BSP, 포털, PVS 에 대한 개략적인 개념과 각각의 장단점을 알아보고, 상용 게임엔진에서의 공간분류기법을 비교하고 결론을 맺는다.

2. 관련 연구

2.1 BSP((Binary Space Partitioning) 개념

BSP 트리는 이진 트리의 특수한 형태의 개념 중 하나이다. BSP 트리는 공간의 범위를 표현한다. 트리는 단일 노드를 비롯하여 여러 개의 노드로 이루어질 수 있다. 다시 말해서, 분할을 표현하는데 사용되는 데이터 구조이다. 트리는 두개의 자식 노드를 가진 노드와 자식 없는 리프로 구성된다. 노드는 트리의 한 부분이며 트리라는 공간의 분할을 나타낸다. 분할을 통해 두 개의 새로운 공간을 만들어지며, 두 공간 중의 하나는 노드의 앞에 다른 하나는 노드의 뒤에 만들어 진다[1].

Doom 과 같은 2D 응용프로그램에서 최상위 수준의 트리는 전체 2D 세계를 표현한다. 루트 노드에는 하나의 선 방정식이 포함되며 세계를 두 개의 부분, 즉 선의 앞과 뒤에 놓여진 부분들로 분할한다. 이들 각 부분은 하위 트리로 표현되며, 그 자체가 BSP 트리이다. 또한 노드에는 분할에 사용된 선방정식의 일부가 되는 선분도 포함된다. 선 세그먼트는 높이나 텍스쳐 ID 와 같은 다른 정보를 사용하여 세계에서 하나의 벽을 표현한다. 하위 트리는 자신들이 표현하는 공간에 다른 벽이 없는 경우에는 리프가 된다. 다른 벽이 없는 경우 그 벽은 다시 한 번 공간 분할에 사용될 것이다.

BSP 트리 작업을 수행하는 데는 두 가지 기본적인 방법이 있다. 첫 번째 방법인 노드 기반 BSP 트리의 경우, 분할에 사용되는 면과 폴리곤 두 가지 요소가 노드에 포함된다. 리프는 비어 있다. 두 번째 방법인 리프 기반 또는 리프형 BSP 트리의 경우, 노드에는 면만 포함된다. 리프에는 볼록 공간의 테두리를 형성하는 모든 폴리곤이 포함된다. 장면의 PVS(Potentially Visible Set)을 계산하는 경우에서는 리프 기반 BSP 방법이 사용된다.

BSP 는 트리를 생성하는데 사용되는 폴리곤 집합이 개체의 테두리 표현을 나타내는 경우 매우 유용하다. 이론적으로 개체의 내부는 솔리드(Solid)로 구성되고 외부는 빈 공간으로 둘러싸여지며 중간에서 폴리곤들과 만난다. 트리가 완벽한 경우 각 리프는 솔리드 또는 빈 공간을 표현한다.

2.3 BSP 트리 알고리즘

BSP 트리에 대해 연산을 수행하기 위한 몇 가지 알고리즘을 살펴보자.

2.3.1 폴리곤 정렬

BSP 트리를 사용하는 첫 번째 방법은 지정된 시점으로부터의 거리별로 정렬된 폴리곤 목록을 읽어 오는 것이다. 이 방법은 정확히 렌더링하기 위해 폴리곤을 뒤가 앞에 오는 순서로 그려야 할 때 하드웨어 z 버퍼 이전에 사용된다. 이 방법도 유용하긴 하지만 초기에 거부하는 경우에는 z 버퍼 렌더링이 더 빨라지므로 정확히 렌더링 하려면 알파 혼합 폴리곤을 뒤가 앞에 오도록 렌더링 해야 한다.

2.3.2 점의 위치 테스트

사실상 BSP 의 가장 훌륭한 점은 점의 위치를 테스트 하는데 사용할 수 있다는 점이다. 점과 트리가 제공되었을 때 그 점의 순수 리프에 놓여 있는지 여부를 알 수 있다. 이 점은 특히 충돌 검사 시에 유용하게 사용된다[2][4].

트리의 각 가지에서 면에 대해 점을 테스트하기만 하면 된다. 점이 앞에 있는 경우에는 앞가지 아래에 배치하고 뒤에 있는 경우에는 뒷가지 아래에 배치하면 된다. 점이 폴리곤과 동일한 평면상에 있는 경우에는 두개 중 하나를 선택하면 된다. 비로소 리프에 이르면 점이 위치한 공간의 영역을 발견하게 된다. 솔리드가 될 리프의 경우에는 점이 솔리드 공간에 놓여질 것이다. 그렇지 않은 경우에는 다른 공간에 놓여질 것이다.

2.3.3 선분 테스트

이 알고리즘은 선의 양쪽 끝점간에 볼 수 있는 명확한 선이 존재하는 경우에는 참을 반환하고 그렇지 않는 경우에는 거짓을 반환한다.

다른식으로 표현하면 선분이 솔리드가 아닌 리프에 위치한 경우에만 참을 반환한다고 말할 수 있다.

루트에서 시작하여 선분을 노드에 있는 면과 비교한다. 선분이 완전히 앞쪽에 있으면 프론트 목록에 추가하고, 완전히 뒤에 있으면 백 목록에 추가하는 것이다. 선분이 면의 양쪽에 걸쳐 있으면 면의 앞쪽에 있는 부분과 뒤쪽에 있는 부분으로 나눈 다음 각각을 재귀적으로 호출한다. 선분의 한 부분이 솔리드 단위에 이르면 볼 수 있는 선이 없다고 보고 거짓을 반환한다.

2.4 BSP 트리의 이용분야

Z-sorting polygon 들이 순서대로 그려지게 하는데 적합하다.

BSP 면들에 대한 충돌검사(Collision detection)는 매우 빠르고 쉽다.

Node Clipping 은 렌더링 속도를 향상 시킬 수 있는 또 다른 미리 계산된 BSP 트리이다.

PVS(Possible Visible Set)은 렌더링 시간을 더욱 더 감소시킬 수 있도록 BSP tree 와 같이 계산될 수 있다.

2.5 선택과 충돌 검사

월드를 표현하는 BSP 트리가 주어졌을 때, 선택(paciking)작업에서는 선 방사선 또는 선분 같은 선형성분이 월드의 객체들을 교차하는 여부를 판단한다. 이 개념은 BSP 트리를 순회하며 재귀적으로 선형 성분을 분할한다는 것이다. 월드 다각형을 표현하는 리프 노드에 도달된 후에 선형 하위 성분이 존재하는 경우, 원래의 선형 성분은 월드에서 하나의 객체를 교차한다. 순회에서 교차한 리프 노드에 이를 때 정확한 교점을 계산할 수 있다.

객체를 표현하는데 두 개의 BSP 트리가 사용되는 경우와 객체가 이동하지 않고 있을 경우에 BSP 트리를 사용하면 객체의 교차를 계산할 수 있다. 교차 영역이 비어 있지 않은 경우에는 객체들이 현재 충돌상태에 있는 것이다. BSP 트리 간 부울 연산을 구현하여 계산적 입체 기하학 형태의 일반적인 지원을 제공할 수 있다 하더라도, 다른 트리의 모든 다각형 면에 대해 하나의 트리에서 각 다각형면을 분할하게 되므로 처리 비용이 많이 든다. 또한 객체가 이동하고 있지만 모양이 변경되지 않는 경우, BSP 트리는 모델 공간 정보를 표현하므로, 분할 평면들이 객체가 이동할 때마다 월드 공간 좌표로 변환되어야 한다. 움직임 때문에 테교차 테스트가 훨씬 더 복잡해지며, 트리 간 부울 연산도 일반적으로 처리 비용이 많이 든다. 경제 볼륨에 대한 기하학적 정보를 사용하는 거리 계산이나 축 분리 테스트에 기초하므로 이들 방법을 사용하는 것이 훨씬 더 경제적이다.

2.6 가시도 결정

시점이 주어졌을 때, 가시도 결정은 월드의 어느 부분이 해당 위치에서 보여질 수 있는지를 결정하는 작업을 말한다. 다각형 객체들로 채워진 월드에서 볼 수 있는 부분을 알고 있다면 렌더러에게 전달되는 데이터를 최소화할 수 있다. 이는 폐색(Occlusion)의 개념과 관련이 있다.[2] 장면에서 가려지는 객체들은 렌더러에서 처리할 필요가 없다. 가시도 정보는 현재 시점에서 볼 수 없는 정적 장면의 경우, 가시도 정보는 사전 처리 단계로서 계산될 수 있다. 그러나 시점이 이동할 수 있는 경우, 시간이 지남에 따라 볼 수 있는 객체가 달라진다. 동적으로 볼 수 있는 객체를 정확히 판단하는 일은 대개는 처리 비용이 많이 드는 작업이다. 대부분의 시스템에서는 근사값을 구하여, 렌더러에게 전송되지만 보이지 않을 수도 있는 객체들의 수를 최소화한다.

BSP 트리는 가시도 판별에 사용할 수 있다. 뷰 공간(3D)에서 작동하는 방법과 화면 공간(2D)에서 작동하는 방법을 설명한다. 두 경우 모두, 분할된 월드를 표현하며 앞에서 뒤로 방식의 분류에 사용되는 BSP 트리가 이미 존재한다. 이 트리를 월드 트리라고 한다. 두 번째 BSP 트리는 가시도 정보를 저장하는데 사용된다. 이 트리를 가시도 트리라 한다.

2.6.1 뷰 공간 방법

가시도 트리는 3 차원에 존재하며 처음에는 뷰 절두체를 표현한다. 이 트리에서는 6 개의 분할 평면들이 절두체를 형성한다. 현재 시점이 주어졌을 때, 월드 트리를 순회한다. 순회하면서 만나는 각 다각형은 가시도 트리에 의해 처리되며 부분 다각형으로 분해되는데, 이들 각 부분 다각형은 전볼륨으로 볼 수 있거나 전볼륨으로 볼 수 없다. 각각의 볼 수 있는 부분 다각형은 시점과 부분 다각형의 모서리에 의해 형성되는 새로운 분할 평면 집합을 정의하는데 사용된다. 시점과 해당 평면은 피라미드 형태를 형성한다. 이 피라미드에서 부분 다각형의 이면에 있는 월드의 부분들은 육안으로 볼 수 없다. 이제 가시도 트리는 이 피라미드를 저장하여 월드 트리 순회에서 방문한 다각형의 세부 클리핑 작업에 사용한다.

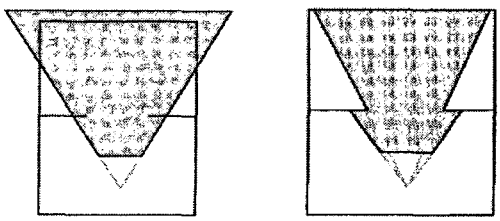
2.6.2 화면 공간 방법

가시도 트리가 2 차원에 존재하며 처음에는 화면에서 그릴 수 있는 픽셀에 해당하는 사각형을 표현한다. 현재 시점이 주어졌을 때, 월드 트리를 순회한다. 순회하면서 만나는 각 다각형은 화면 공간으로 투시된 후, 가시도 트리에 의해 처리되며 부분 다각형으로 분해되는데, 이들 각 부분다각형은 전볼륨으로 볼수 있거나 전볼륨으로 볼 수 없다. 월드 트리는 앞에서 뒤로 방식으로 다각형을 분류하므로, 클리핑 작업에서 얻는 볼 수 있는 부분 다각형은 가시도 트리 계산 시 볼 수 있는 다각형으로 유지된다. 이들 부분 다각형은 응용 프로그램에서 사용할 수 있도록 목록 안에 저장될 수 있다.

3. 포털(Portal)

3.1 포털의 개념

포털은 실내 환경에서 주로 적용되는 것으로, 벽으로 나눠진 방과 같은 구조에서 연결되는 문이나 창과 같은 부분을 말한다. 다시말해서 실내 환경에 대한 장면 관리를 처리하는데 매우 효과적인 방법이다.[3] 외부환경에서는 카메라의 가시대가 보이는 영역을 상징할 수 있지만, 실내에서는 벽과 같은 장애물이 있어 가시대에 포털까지 적용해야 정확한 가시영역을 나타낼 수 있다. 포털 테크닉은 가시영역 내의 물체만을 렌더링하는 것을 말한다.



[그림 1] 포탈에 의해 보여지는 가시도

그림 1 은 포탈의 환경을 보여준다. 왼쪽 그림에서 검정부분은 렌더러가 표준 뷰 절두체에서 처리를 시

도함을 보여주고 오른쪽 그림의 검정부분은 포털에 의해 제한된 영역을 보여준다.

3.2 포털의 장단점

개체가 존재하는 셀의 공간을 알게 되면, 장면은 아주 다루기 쉬운 것이 되어버린다. 다른 개체에 대해 테스트해야 하는 개체에 충돌검사를 수행하려 하는 경우 장면 안에 있는 모든 개체를 다 검사할 필요가 없다. 단지 대상 개체가 속해 있는 각 셀의 개체만 검사하면 된다. 또한 포털 렌더링을 사용하면서 셀 기반 세계에 있다는 장점도 있다. 따라서 관찰 지점에서 볼 수 있는 정확한 셀의 집합을 찾을 수 있다. 뿐만 아니라 관찰지점에서 볼 수 있는 정확한 폴리곤의 집합도 찾을 수 있다.

4. PVS(Potentially Visible Set)

4.1 PVS의 개념

PVS는 보여질 가능성이 있는 리스트를 기억해 두는 방식이다. PVS는 그 특성상 BSP 같은 공간 분할과 포털(Portal)테크닉을 같이 사용해 구현한다.[3]

BSP 와 포털방법을 합치면 분할된 공간에 해당하는 가시 영역을 구할 수 있다. 다만 이 분할된 공간영역에 대한 포털을 재귀적으로 상호 클리핑해 나가면서 가시도 검사를 하기 때문에 굉장히 복잡하고 시간이 많이 걸린다. 따라서 PVS는 동적인 환경에 대해 실시간으로 구현하는 것은 무리가 있고, 정적인 환경인 경우에 한해 미리 테이블로 만들어두고 실시간 렌더링시 카메라가 속해 있는 공간에 대해 테이블 LookUp을 통한 가시도 리스트를 참조하는 방법으로 사용된다.

4.2 PVS의 장단점

PVS는 정확한 가시도를 표현할 수 없다. 셀의 내부에 있는 모든 점에서 볼 수 있는 셀 집합은 그 셀 내부의 한 특정한 지점에서 볼 수 있는 셀 집합보다 거의 엄청나게 많을 것이다. 따라서 카메라로부터 완전히 가려지는 일부 셀들을 그리게 될 수도 있다. 그러나 채움속도는 저하되겠지만 처리 시간은 향상된다. 어렵게 절두체를 생성하거나 셀 순회를 수행할 필요가 없다. 단지 특정 셀의 비트 벡터를 단계적으로 진행하며, 비트가 설정된 모든 셀들은 그리기만 하면 된다. 단점으로는 알고리즘이 매우 복잡하므로 PVS를 계산하는데 많은 시간이 소요된다. 이로 인해 셀들은 이동시킬 수 없게 되고 글 셀들이 정적인 상태로 남아 있어야 한다.

5. 상용엔진에서의 공간 분류기법 비교

퀘이크 시리즈에서 주로 BSP 를 사용했다. 원하는 폴리곤을 최대한 빨리 찾기 위해 3D 공간을 모델링시부터 분할해 저장해뒀다가 분할된 데이터를 이진 검색으로 찾는 방법이다. 장점은 빠른 레더링과 충돌검사가 쉽다는 점이다. 퀘이크의 단점은 배경중 일부 오브젝트를 제외하고 움직을수 있는 것이 없는 것이다. 고정된 구조이기 때문에 복잡하고 거대한 움직이는 배경을 표현하기 부적합하다. 반면 언리얼 계열 엔진은 포털 렌더링 기법을 사용했다. 포털이란 일정한 문을 중심으로 공간을 분할하고, 현재의 문에서 보이는, 또는 갈수 있는 다른 공간을 미리 계산해 놓는다. 현재 공간내에서의 모든 정점이 고정된 BSP 와는 달리 움직일수 있도록 유연하게 설계함으로써 동적인 처리를 강점으로 내세운다.

공간 분류기법	게임 엔진
BSP	Genesis3D, Titan Engine, ENG32, Basic3D, Coockie's 3D Engine, QLeVe, Golk, Twister, NBFB, Bullet3D, V3D, SGE 3D, Venerium, Jupiter
Portal	Jet 3D, Unreal, Miracle 3D
BSP, PVS	Legus3D, Fly3D, Quake 1, Crystal Space
BSP, Portal	Sparklight, zViewer, MGRage
BSP, Portal, PVS	Carsten's 3D Engine, Gran3D

6. 결론 및 향후 방향

본 논문에서는 게임엔진에서의 공간 분류기법으로 쓰이는 BSP 와 포털, PVS 에 대한 개념과 장단점에 대해서 기술하였다.

빠르고 정확한 렌더링을 필요로 하는 게임엔진에서의 공간 분류기법은 각각의 특징 및 장단점에 따라서 여러 가지를 혼합 또는 단일형태로 쓰이고 있다.

7. 참고문헌

[1]Moller, Tomas, Eric Haines, "Real-Time Rendering", A K Peters Ltd., 1999

[2]BSP TREE FREQUENTLY ASKED QUESTIONS (FAQ) http://www.gamers.org/dhs/helpdocs/bsp_faq.html

[3]Alan Watt, Fabio Policarpo, "3D Games Pealtime Rendering and Software Technology", ADDISON-WESLEY, 2001

[4]Eric Lengyel, "Mathematics for 3D Game Programming and Computer Graphics" CHARLES RIVER MEDIA, 2002