전처리된 Serialized Octree를 이용한 연속적인 쿼드트리 공간 랜더링

°
염창근, 박경환
동아대학교 컴퓨터공학과

Continuous Quadtree Space Rendering using Serialized Octree with Preprocessing

°
Chang-gun Yeom, Kyung-hwan Park
Dept. of Computer Engineering, Dong-A University

요 약

실시간 랜더링은 가상 공간과 사용자가 상호작용이 가능함을 말한다. 이런 실시간 랜더링 기법 중에서 옥트리는 보이지 않는 면을 검출함(Back Face Culling)에 있어서 매우 유용하다. 그러나, 지형처럼 데이터가 끊임없이 이어져서 연속적이라면 사용자의 시점이 가상 공간을 벗어나 새로운 영역을 참조하려고 할 때에 옥트리는 새로운 공간을 표현함에 있어서 실시간을 지원하지 못한다. 이러한 옥트리의 공간 분할 작업을 미리 계산해서 저장하고 실시간 랜더링 시에는 참조하도록 하며 전처리된 옥트리가 하나의 셀이 되어 다수의 옥트리 셀을 묶어 쿼드트리를 구성하고 카메라에 보이는 옥트리들을 결정하기 위해서 삼각형의 포함 알고리즘을 적용하여 연속적인 공간 데이터의 실시간 랜더링을 구현하였다.

1. 서론

3차원 그래픽은 인간에게 있어서 가장 자연스럽고 현실감 있는 표현 방법이다. 현실 세계의 정보들은 모두 3차원 정보로 표현되며 눈이라는 감각 기관을 통해서 평면 이미지로 투영된다. 움직임이 없는 사진은 현실감이 떨어짐을 잘 알고있다. 이것은 감각 기관이 사진에서는 상호작용을 할 수 없음을 의미하기도 한다. 즉, 현실감이란 2차원과는 달리 공간과 인간이 상호작용이 가능해짐으로써 얻게 되는 동적 정보란 걸 알 수 있다. 가상 현실과 같은 시스템은 이처럼 실세계와 유사한 환경을 조성하고 시스템과 인간의 상호 정보 교환이 가능하도록 만든 것이다. 게임 역시 3차원 그래픽 응용 분야로서 활발히 연구되고 있는데 이런 게임에서도 현실감 있는 그래픽을 구현하는 기법이 다양하다[1][2]. 하드웨어의 성능이 비록 비약적인 발전을 이루었지만 앞서 언급한 현실감을 얻기에는 아직 부족하다. 그러나, 3차원 그래픽 가속기의 성능의 발전과 더불어 다양한 알고리즘의 개발로 완벽하지는 않지만 어느 정도 현실감을 느낄 수 있는 그래픽을 얻게 되었다. 이 들 알고리즘의 기본은 먼저, 상호 작용이 가능한 초당 30프레임 이상의 화상 갱신과 복잡한 계산은 미리 계산해 두어서 실시간에서는 단지 참조만 하는 기법, 마

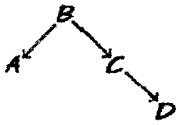
지막으로 공간을 쪼개어서 랜더링 파이프라인에 들어가는 폴리곤의 수를 줄였다는 것이다. 마지막에 기술한 공간을 분할하는 방법은 현재도 연구가 진행중인 분야이며 각 알고리즘 또한 상호 보완해서 혼합된 알고리즘도 이용되고 있다. 본 논문은 공간 표현 기법 중 옥트리[3]에 관하여 문제점을 지적하고 건물 내부의 표현이나 다양한 지형 등에서 동시에 이용할 수 있도록 제안한다.

2. 공간 분할 기법

어떤 공간을 구성하는 3차원 데이터의 양이 방대해지면 화면에 출력하기까지의 과정에 많은 부하를 주게 되는데 이럴 경우, 다양한 그래픽 응용 분야 게임, CAD, 가상 현실, 각종 시뮬레이션 - 에서 실시간 랜더링을 지원함에 있어서 엄청난 부하가 따른다. 이를 개선하기위해 카메라에 보이는 물체만을 랜더링하는 알고리즘이 상당수 개발되었는데 BSP, PVS, 옥트리가 가장 대표적인 자료구조이다.

2.1 BSP (Binary Space Partitioning)

BSP[2]는 공간을 2개로 나누어 저장하는 갈라지는 나무 구조를 사용한다. BSP 트리는 깊이 정렬, 충돌 탐지, 보이는 표면 결정과 같은 많은 목적을 위해 사용되는데 BSP 트리의 타입에는 노드 트리와 리프 트리의 2가지 타입이 있다. 이들 트리는 깊이 정렬과 폴리곤 분할을 이용하여 구현된다. 이 자료구조는 건물 내부와 같은 공간과 공간이 문이나 창과 같은 통로를 통해서 구분되는 공간에 적합하며 뷰어의 시점에 의해서 BSP트리를 중위 순회하여 그려질 폴리곤을 정렬하는 알고리즘이다.



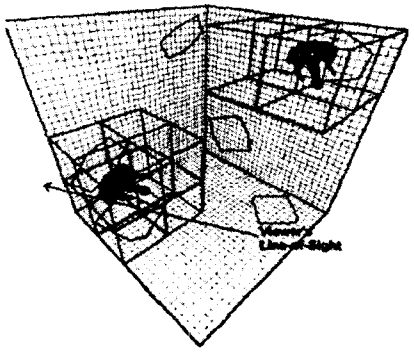
[그림 1] BSP 트리

2.2 PVS (Potentially Visible Set)

공간상에서 폴리곤들은 다른 폴리곤에 가려서 보이지 않는 경우가 많다. 이 때 보이는 폴리곤들만을 랜더링 하도록 결정해서 출력하는 알고리즘으로 지형을 리프(Leaf) 노드로 할당하고 여러 개의 영역으로 나눈 후 각 영역에서 보인다고 결정할 수 있는 영역들을 리스트로 구성하여 Lookup 테이블에 미리 저장해두고 게임이 실행될 때 카메라의 위치를 참조하여 어느 영역에 포함되는지를 테이블에서 찾아서 보이는 영역만을 랜더링하는 알고리즘이 PVS[2]이다.

2.3 옥트리 (Octree)

옥트리[2][3]란 트리의 자식 노드가 8개를 가지는 트리를 옥트리라고 한다. 옥트리란 공간을 구성하는 폴리곤을 임의의 영역으로 나누는 방법으로 모든 폴리곤을 다 포함하는 최상위 노드를 루트(root)로 하여 자식 노드들은 부모 노드 크기의 1/8 크기를 갖는 정육면체가 된다. 이런식으로 최하위 노드가 가지는 폴리곤의 개수가 한계치에 이르면 더 이상 자식 노드로 분할하지 않는 알고리즘이다. 또 다른 분할 방법으로 자식 노드를 폴리곤의 개수와 상관없이 균등 분할할 수 있는데 노드의 분할은 응용 목적에 따라 선택할 수 있다[4][5].



[그림 2] 공간 분할된 옥트리

2.4 쿼드트리

쿼드트리[6][7]란 옥트리의 2차원 버전이라고 볼

수 있다. 전체 2차원 지형 영역을 일정한 크기의 셀(cell)로 계속해서 4등분하여 임계값을 넘지 않을 때까지 분할해준다. 쿼드트리나 옥트리나 모두 공간 분할의 자료구조 이지만 전체 가상 공간이 초기에 모두 제공되어야만 분할작업이 수행되어지기 때문에 동적인 공간의 확장이 어렵다.

3. 전처리된 옥트리

3.1 옥트리 분할의 부하

폴리곤을 옥트리로 분할함에 있어서 실시간 랜더링을 적용하려면 먼저 각 노드에 속하는 폴리곤을 결정해야 하는 선행 작업이 있다. 폴리곤이 특정 노드에 속하는지를 결정하는 방법으로는 첫째, 각 노드를 구성하는 경계상자(bounding box)의 8개 꼭지점에 가상공간을 구성하는 모든 폴리곤의 세 꼭지점이 포함되는지의 여부를 판단해야 하고 둘째, 각 폴리곤마다 현재 노드를 구성하는 경계상자의 12 모서리 중 하나라도 폴리곤을 통과하여 지나가는지 또한 결정해주어야 한다. 이 두가지 조건 중 하나라도 만족하면 폴리곤은 현재 노드에 속한다고 결정되며, 차후 실시간 랜더링 시 카메라의 뷰잉 영역에 노드가 포함되면 이들 노드에 속해 있는 폴리곤들이 랜더링 파이프라인으로 보내어져서 스크린 좌표로 변환되어 출력되어진다.

그러나, 랜더링 작업이 시작되려면 앞에서와 같이 특정 폴리곤을 임의의 노드에 할당해야만 하는데, 여기에 걸리는 시간의 부하는 실시간 랜더링에는 적합하지 못하다. 이것은 표현될 지형이 동적으로 변하는 레이싱 게임이나 온라인 머그 게임등에는 부적합하다[6]. 즉, 아웃도어(outdoor) - 건물 외부에서 보여지는 각종 지형 데이터 - 랜더링에서 옥트리는 좋은 선택이 아니다. Indoor라 해도 건물 내부 역시 랜더링 이전에 폴리곤 분할이 요구된다. 즉, 옥트리는 동적으로 객체의 정보가 바뀌는 응용 분야에서는 적합하지 않음을 의미한다. 그러나, 미리 폴리곤들이 분할만 되어 있다면 다양한 실시간 응용 분야로 적용이 가능해짐을 역설할 수 있을 것이다.

3.2 옥트리 Serialization

메모리에 위치한 객체의 상태를 프로그램이 종료되어도 차후 재실행될 때 이전의 정보를 유지하거나 다른 시스템의 메모리에 동일한 정보를 갖도록 객체의 정보가 스트림(stream)으로 변환되어 복사가 될 때 사용하는 기법으로 serialization을 이용한다. 이는 시스템의 안정성을 유지하기위한 기법으로도 사용하지만 본 논문에서는 옥트리의 객체 내부 정보를 그대로 저장하기 위하여 적용되었다. 옥트리 class가 가지는 내부 정보 중 중요한 필드로 노드가 포함하고 있는 폴리곤의 개수 정보와 할당된 면(face) 정보의 포인터를 들 수 있다. 나머지는 단지 부모와 자식간의 관계를 나타내는 링크 필드 정도이다.

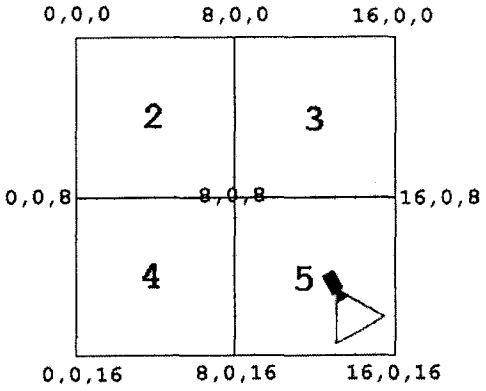
옥트리를 serialize하려면 먼저 루트 노드부터 자식 노드 모두를 한번씩 탐색하면 되는데 이진 트리라면 순회 방법으로 세가지 즉, 중위, 전위, 후위가 있지만 옥트리는 자식 노드가 8개 이므로 중위, 후위 운용은 적합하지 않다. 남아 있는 순회 방법이 전위 순회 이지만 오히려 옥트리를 가장 잘 저장할 수 있는 순회 방법이다. 옥트리는 자료 구조를 구성할 때 재귀 호출을 이용하는데, 이것은 트리를 루트에서 최하위 리프 노드로 방문하면서 다시 상위 노드로 옮겨가는 형태다.

따라서, 옥트리 serialization을 이용할 때도 루트에서부터 전위 순회를 적용하여 각 옥트리 노드를 만날 때 마다 현 노드의 상태 정보를 그대로 파일로 기술하면서 마지막 리프 노드까지 동일하게 수행해 주면 된다.

3.3 옥트리 기반 쿼드트리

옥트리는 쿼드트리에서 하나의 셀로 정의하고 이런 셀들을 루트에서부터 분할하여 마지막 리프노드로 옥트리 하나를 대입하는 방법으로 공간을 관리할 수 있다. 옥트리를 이용한 쿼드트리는 각 트리 노드마다 색인이 있으며, 색인은 옥트리의 루트 노드를 가리키는 포인터가 된다. 카메라는 언제든지 특정

영역을 비출 수가 있는데, 카메라에 보이는 옥트리를 구성하기위해 카메라의 방향 벡터와 시야각(FoV)을 이용한다. 카메라는 가상 공간에서 하나의 객체로 간주되므로 옥트리로 구성된 쿼드트리 리스트에서 카메라의 위치를 참조할 수 있다. 동시에, 방향 벡터의 세 원소값 중 X, Z 두 필드만을 취하면 평면의 벡터값을 얻게 된다. 여기서, 카메라의 뷰잉 프러스텀(viewing frustum) 값 중 원거리 컬링 평면에 해당하는 거리값을 곱해주면 카메라에서 볼 수 있는 최장 거리까지 도달하는 위치 벡터를 계산할 수 있다. 위치 벡터가 계산되면 시야각과의 연산을 통해서 시점과 두 꼭지점을 잇는 삼각형이 도출된다.



[그림 3] 옥트리 셀 기반의 쿼드트리 지형

모든 쿼드트리 맵에는 각각 3차원 위치 정보를 포함하는데 각 옥트리의 크기와 중심값을 알면 타일을 구성하는 4개의 꼭지점을 알 수 있고, 카메라의 뷰잉 영역을 의미하는 삼각형과 타일의 꼭지점의 포함 관계를 분석하면 모든 옥트리 노드가 보이는지 여부를 판가름 할 수 있다.

4. 결론

옥트리는 초기에 주어지는 폴리곤 데이터를 공간 분할하여 랜더링 효율을 높이는 좋은 자료구조이지만 분할 시간에 많은 부하가 걸리기 때문에 매우 광범위한 공간 데이터를 다루기에 어렵다. 이런 문제를 해결하기 위해서 옥트리의 상태 정보를 파일로 저장하면 매번 폴리곤 분할에 시간을 소비하지 않아도 되었다. 이런 이점을 이용하여 광범위 데이터에 적용이 가능하며 지형 또는 다양한 공간 데이터를 다룰 때 유용한 쿼드트리 자료구조를 적용해서 각 셀은 독립적인 옥트리로 구성해 줌으로써 보이지 않은 데이터를 사전에 랜더링 엔진에게 제거할 수 있기 때문에 평균적인 랜더링 속도 보장이 가능해진다 옥트리는 기존의 다른 알고리즘이 indoor에 의존적인 것에 반해 outdoor에도 적용할 수 있는 자료구조이므로 향후 범용적인 그래픽 엔진 구현에 쓰이도록 연구가 필요할 것이다.

[참고문헌]

[1] David H. Eberly, 3D Game Engine Design, Morgan Kaufmann

[2] Tomas M ller, Eric Haines, Real-Time Rendering, 1999 by A K Peters, Ltd

[3] Jaap Suter, Introduction To Octrees 1999.4
http://www.flipcode.com/tutorials/tut_octree

[4] Dan Mints, Octree Spatial Partitioning for Real-Time 3D Hidden Surface Removal, May 7, 2000

[5] Dan Mints, Collision Detection and Response with Bounding Spheres and Arbitrarily-Oriented Triangles, May 21, 2000

[6] Thatcher Ulrich, Continuous LOD Terrain Meshing Using Adaptive Quadtrees, February 28, 2000
http://www.gamasutra.com/features/20000228/ulrich_01.htm

[7] Jonathan Ferraris, Background and Theory: What are quadtrees?
http://www.gamedev.net/reference/articles/article1303.asp