넓은 지형처리를 위한 CLOD가 적용된 옥트리

이승욱
동아대학교 컴퓨터공학과
e-mail : sunguglee@yahoo.co.kr

Octree Partition Method using CLOD for Large-scale Environments

Sookng-ug Lee
Dept. of Computer Engineering, Donga University

요 약

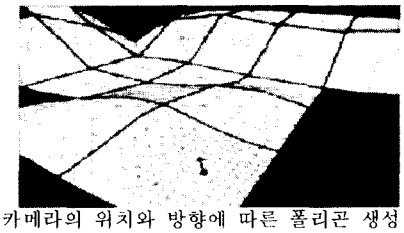
3D MMORPG(Massive Multi-play Online Role Playing Game) 게임은 넓은 3 차원원지형을 실시간으로 표현 되어야 하며, 많은 어려움이 따른다. 본 논문에서는 이러한 지형 처리를 위하여 메쉬나 버텍스, 혹은 폴리곤으로 사실적인 지형처리와 렌드링 속도 향상을 위하여 3 차원 폴리곤을 동적으로 생성 시키는 방법을 이용하려고 한다. 넓은 지형을 처리 하기 위해서는 전체를 한번에 표현하기 보다는 페이지 단위로 처리하기 위하여 격자화된 타일로 이루어진 맵으로 처리할 수 한다. Height field 처리 기법은 일정한 영역을 페이지 단위로 구분하고 처리할 수 있다. 옥트리를 이용하여 공간을 입체적인 컬링 방법으로 분할하고, 이를 세부 수준으로 나누어 처리하기 위해 CLOD(Continuous Level of Detail) 개념을 적용할 수 있다. 거리의 변화에 따라 지형을 표현하는 vertex 들을 병합 또는 삭제함으로써 그 표현의 정도를 동적으로 달리 할 수 있는 CLOD 를 이용하여 카메라의 위치와 방향에 따라 적절한 폴리곤을 생성해 낼 수 있다. 본 논문은 3 차원의 넓은 외부 지형을 실시간으로 처리할 경우 발생되는 그래픽 문제를 해결하기 위해 사용되는 방법 중에서 대표적인 방법을 통하여 효율적인 처리 기법을 제시하려 한다.

1. 서론

3 차원 가상공간에 광활한 지형을 실시간으로 사실적 지형 처리를 위한 부분은 게임의 중요한 요소로 작용한다. 지형은 게임의 배경으로 지형은 다른 객체들과 플레이어가 상호 작용하여 게임 세계를 만들어 내게 된다. 3 차원 공간 상에 사용되는 거대한 지형 데이터와 개체를 실시간으로 처리하기 위해서는 많은 어려움이 따른다. 고려해야 할 사항을 세가지로 요약 한다면 다음과 같은 것들이 있다. 우선은 3 차원 게임에 사용되는 모델의 폴리곤 수를 최대한 줄여야 한다. 실사와 같은 영상을 얻기 위해서는 폴리곤의 수가 많으면 많을수록 좋겠지만 게임과 같은 환경에서는 무엇보다도 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 3 차원 그래픽에 사용하기에는 적당하지 않다. 세 번째로 공간을 쪼개서 렌더링 파이프 라인에 들어가는 물체의 수를 줄이고 카메라의 뷰 볼륨 안에 있는 물체만 렌더링 하도록 한다. 이러한 부분들은 개임에 있어 그래픽처리 성능을 향상시키는 아주 중요한 요소이다. [3][7][10]

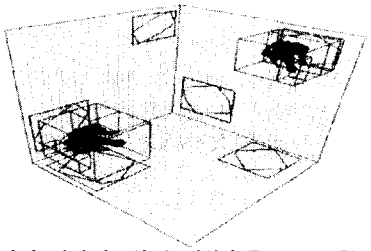
2. 3 차원 가상 공간의 지형데이터를 처리하는 방법

3 차원의 가상 공간의 지형 데이터를 처리하는 방법으로 거대한 지형 데이터를 실시간으로 처리할 경우 지형을 고정적인 정점 좌표들을 주어서 표현할 경우 지형이 광범위해서 당연히 속도 저하가 생긴다. 광범위한 지형의 각 정점의 좌표를 거리에 따라 효율적으로 적절한 폴리곤 수를 조절할 수 있는 데이터 저장 방법이 필요하다. 지형 데이터를 표현할 때 지형의 높이 정보만을 저장하고 폴리곤은 내부적인 알고리즘으로 생성해 내는 방법이다. Height field 란 일정한 간격이 (x,y)좌표에 대한 z 축 성분인 높이 값을 말한다. Height field 의 경우에는 저장 공간을 줄이기 위한 방법으로 z 축의 데이터만을 저장한다. 하지만 매번 z-buffer 를 읽어야 하기 때문에 속도 저하가 야기되고, 매번 역 행열을 계산해야 하므로 속도가 저하된다. 이를 개선하기 위한 방안으로 Height field 에 의해 저장된 점 좌표를 8 개의 화면으로 균등 분할한다.



[그림 1]카메라의 위치와 방향에 따른 폴리곤 생성의 변화

데이터를 처리하기 위한 방법으로 옥트리를 이용하여 화면을 단순히 분할하는게 아니라 폴리곤의 개수를 기준으로 균등 분할을 시도함으로써, 메모리의 오용이 적다. 옥트리의 렌더링의 기본 개념은 일단 가장 최상단의 옥트리부터 시작하여 8 개 점의 각도를 구해서 화면 시야 여부를 결정한다. 8 개의 점이 다 시야 안이라면 그 안의 옥트리를 그냥 더 이상의 클리핑도 필요 없이 렌더링에 필요한 노드를 검출한다.



[그림 2]주위의 입방체로부터 세부수준으로 분할된 옥트리

8 개의 점이 모두 밖에 있다면 그 안의 옥트리는 화면에 보이지 않는 게 확실하므로 노드가 검출되지 않고 2 가지의 경우가 아니라면 다시 하위 옥트리로 내려가 다시 앞의 2 가지를 비교하게 된다. 옥트리에 의해 화면 분할이 끝나면 카메라의 viewing furstum 과 옥트리는 모든 노드와의 포함 연산을 통해 보여지는 노드를 검출한다. 카메라의 거리가 아닌 화면상에 나타난 크기를 기준으로 이력 문턱 값이 적용된 LOD 가 적용하게 되는데, LOD 적용시 확대율의 계산의 경우 발생할 수 있는 카메라의 거리가 아닌 화면상에 나타난 크기를 기준으로 모델의 세부 수준을 결정해야 한다는 문제와 빠르게 반복되는 수주 전환의 문제를 이력 문턱 값을 적용함으로 하나의 값이 아니라 하나의 범위에 대한 문턱 값 적용으로 LOD 의 빈번한 변화를 방지할 수 있다.

반면 CLOD 는 거리의 변화에 따라 지형을 표현하는 vertex 들을 병합 또는 삭제함으로써 그 표현의 정도를 동적으로 달리 할 수가 있다. 물론 실행시간에 폴리곤을 생성하는 오버헤드는 있지만 정점으로 표현될 경우 방대한 양의 LOD 폴리곤 데이터를 여러 개 갖게 되는 메모리의 낭비를 막을 수 있고 카메라의 위치와 방향에 따라 적절한 폴리곤을 생성해 낼 수 있는 장점이 있다.

[적용 단계]

1 단계 3 차원의 가상 공간의 지형 데이터를 Height field 를 이용하여 3 차원 점 좌표로 변환한다.

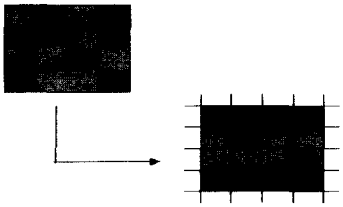
2 단계 옥트리를 이용하여 Height field 점좌표를 8 개의 화면으로 균등 분할한다.

3 단계 카메라의 viewing furstum 과 옥트리를 모든 노드와의 포함연산을 통해 보여지는 노드를 검출한다.

4 단계 CLOD 를 적용하여 실시간으로 폴리곤을 생성한다.

①단계 구현

격자화된 타일들로 이루어진 맵을 이용하며, 해당 좌표를 가지고 바로 그에 해당하는 타일을 알아낼 수 있다. 한번에 모든 맵을 처리하는 것이 아니라 일정 영역을 제한하여 처리한다.



[그림 3] 맵소스를 타일로 맵핑

```
typedef struct tagHEIGHTFIELD {
    int     nWidth;
    int     nDepth;
    float   fXTrans;
    float   fZTrans;
    int cVertex;
    VERTEX* pVertex;
    float fTileWidth;
    float fTileDepth;
}
```

nWidth : 맵의 가로 너비 (격자화된 타일)
nDepth : 맵의 깊이 (Z 축 타일)
fXTrans : X 축 좌표 fZTrans : Z 축 좌표
cVertex : 점들의 개수 Pvertex : 정점 데이터
FtileWidth : 타일의 가로 너비.(X 축상 스케일)
FtileDepth : 타일의 깊이(Z 축상 스케일)

```
#define SEIGHT_WIDTH      16
#define SEIGHT_DEPTH      16
```

- 3 차원의 가상 공간의 지형데이터를 Height field 를 이용하여 3 차원 점 좌표로 변환 처리부분

```
void HEIGHTFIELD_Transform(HEIGHTFIELD* pWorld,
CAMERA* pCamera)
{
 int nStartX, nEndX;
 int nStartZ, nEndZ;
 int x, z;
 pWorld->fXTrans=pCamera->POV.x/pWorld-
>fTileWidth;
 pWorld->fZTrans=pCamera->POV.z/pWorld-
>fTileDepth;
// 일정 범위로 제한하기 위해, 처리할 영역을 계산
 nStartX = (int)pWorld->fXTrans;
 nStartZ = (int)pWorld->fZTrans;
 nEndX=min((nStartX+SEIGHT_WIDTH),pWorld-
nWidth);
 nStartX=max(0,(nStartX-SEIGHT_WIDTH));
nEndZ=min((nStartZ+SEIGHT_DEPTH),pWorld->nDepth);
```

```
nStartZ=max(0,(nStartZ-SEIGHT_DEPTH));
// 해당 정점을 변형 처리
for ( x = nStartX; x < nEndX; x++ ) {
for ( z = nStartZ; z < nEndZ; z++ )
MATRIX_TransformWithHomogenous (
pCamera->W2P,
pWorld->pVertex[z * pWorld->nWidth + x] );
}
}
pCamera
pCamera->POV
```

② 2 단계

- 옥트리를 이용하여 Height field 점 좌표를 8 개의 화면으로 균등 분할 방법 구현

화면을 8 등분하고 화면의 중앙에서 8 개 박스를 만들 수 있다. 그 바운드 박스 안에 들어온 폴리곤을 골라 임시 변수에 등록한다.(일정한 한계를 결정하는데, 2 가지 방법이 대표적으로 쓰인다. 하나는 이 옥트리 내의 폴리곤 개수가 어느 정도 이하면 더 이상 쪼개지 않는 것과, 옥트리를 나누는 한계를 결정하는 것. 보통 유동적이고 효율적인 관리를 위해서는 전자의 방법이 일반적임) 임시 저장된 폴리곤이 어느 정도 이상이면 옥트리안에 등록시키지 않고 다시 이 옥트리를 분할한다. 계속 분할하다 폴리곤의 갯수가 어느 정도 이하로 떨어지면 옥트리 구조체 내에 그 데이터들을 등록시키고, 분할을 멈춘다.

```
BuildOctree()
{
if(NumPolys>POLY_THRESHOLS)
            for(int i =0;i < 8; i++)
            {
        BuildNode(n->Child[i],Iin);
                BuildOctree(n->Child[i];
}
}
```

BuildOctree()의 기능을 살펴보면 다음과 같다.

- 노드에 대한 경계 입방체를 만든다.
- 그 입방체 안에 어떤 다각형들이 들어가는지 판단한다.

③ 3 단계

- 카메라의 viewing furstum 과 옥트리를 모든 노드와의 포함연산을 통해 보여지는 노드를 검출한다.

```
TriInCube(Tri T,Cube C)
{
Vector Trans = C.Center;
Vector Scale = 1.0 / C.Size;
For (int i=0;i<3;i++)
     T.Vert[i] = (T.Vert[i] Trans) / Scale;
If (TriInVoxe[T])
    Return true;
Return false;
}
```

④ 4 단계

- CLOD 를 적용하여 실시간으로 폴리곤을 생성

옥트리에 의해 공간 분할된 3 차원지형을 세부 수준으로 적용하기 위한 방안으로 고려할 수 있는 부분을 살펴보자. 렌더링 할 세부 수준을 선택하는 간단한 방법은 카메라와 개체사이의 거리에 어떠한 문턱 값을 적용하는가에 있다. 즉 카메라의 거리가 아니라 화면상에 나타난 크기를 기준으로 모델의 세부수준을 결정한다는 것이다. 두 번째 문제는 개체가 문턱 값 거리부근에서 계속 움직이는 경우 세부 수준의 전환이 대단히 여러 번 일어나게 된다. 세부수준을 결정을 위해서는 이러한 문제를 해결하기위해서 확대율과 이력 문턱 값의 적용을 통하여 해결할 수 있다. CLOD 의 적용을 통하여 거리의 변화에 따라 지형을 표현하는 vertex 들을 병합 또는 삭제함으로써 그 표현의 정도를 동적으로 달리 할 수가 있다.

3. 결론

본 논문은 3 차원 게임 엔진설계에서 발생되는 속도 문제를 해결하기위한 방안을 모색하려고 Height field 와 옥트리가 가지고 있는 화면분할 방법에 세부 수준의 서로 다른 모델을 적용시킴으로써 렌더링의 성능과 시각적 품질을 높이기 위한 방법을 모색하였다. Height field 는 광범위한 지형을 각 정점의 좌표를 거리에 따라 효율적으로 적절한 폴리곤 수를 조절할 수 있으며 메모리의 오용이 적고 클리핑 및 충돌 처리 시 폴리곤을 정확하고 빠르게 검출해 낼 수 있다. 옥트리를 이용하여 Height field 점 좌표를 8 개의 화면으로 균등 분할함으로써,옥트리는 정적인 지형에 적합하며, 동적으로 움직이는 개체들에 대한 부착 목록을 저장하는 데에도 사용된다. 옥트리에 의해 화면 분할이 끝나면 카메라의 거리가 아닌 화면상에 나타난 크기를 기준으로 이력 문턱값이 적용된 CLOD 가 적용되게 된다. 본 논문에서 제시된 방법은 현재 3 차원 가상공간에서의 외부지형과 개체 표현에 적용한다면 효율적인 처리능력을 발휘할 수 있다.

[참고문헌]

[1] Jaap Suter, Introduction To Octrees 1999.4
URL:http://www.flipcode.com/tutorials/tut_octrees.shtml
[2]Mark Deloura, Game Programming Gems ,CHARLES River MEDIA,INC.2000.8
[3] Astheimer, P. and Pöche, M.-L. Level-of-detail generation and its applications in virtual reality.1994.In Virtual Reality Software and Technology (Proceedings of VRST'1994,8,23-26
[4]Mike Krus,Patrick Bourdot,Françoise Guisnel, and Guillaume Thibault Levels of Detail & Polygonal Simplification Singapore), pages 299-312
[5]Foley, van Dam, Feiner, and Hughes, Computer Graphics: Principles and Practice 2nd Edition, 1987, p 550555.
[6]Hoff, Kenny, Fast ABBB/View-Frustum Overlap Test
[7]Moller and Haines, Real-Time Rendering, 1999, p. 206211, 310312.
[8]Suter, Jaap, Introduction to Octree , 1999.4
URL:http://www.flipcode.com/tutorials/tut_octrees.shtml
[9]Bryan Turner ,Real-Time Dynamic Level of Detail Terrain Rendering with ROAM,Gamasutra 2000,4
URL:http://www.gamasutra.com/features/20000403/turner_01.htm
[10]박현우,최역종, Real Time 3D Graphics Technique.1999.5
URL: http://www.3dartisan.com/