CLOD 를 활용한 충돌검출과 옥트리 분할 기법

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

A Collision Detection and Octree Partitioning Method using CLOD

Sookng-ug Lee Kyung-hwan Park
Dept. of Computer Engineering, Dong-A University

요 약

본 논문은 기존의 3D 게임 엔진에 실시간으로 상호 작용이 가능하고 3D MMORPG(Massive Multi-play Online Role Playing Game) 게임에 적합한 가상 공간을 표현하기 위한 필요한 기술을 분석하고 이를 활용하려 한다. 기존의 머드 게임에 3 차원 기술을 적용하고, 3 차원 물체를 모델링 하는데 있어서 메쉬나 버텍스, 혹은 폴리곤으로 사실적인 지형 처리와 렌드링 속도 향상을 위하여 3 차원 개체의 폴리곤을 동적으로 생성시키고 가시성 판단이나 충돌 검출을 위한 방법으로 Height field 처리 기법과 거리에 변화에 따라 다르게 모델링 된 데이터를 선택적으로 사용하는 CLOD(Continuous Level of Detail) 처리 기법과 입체 컬링 방법으로 옥트리를 이용하여 가상공간을 분해 하기위한 자료 구조로 사용한다. 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 기존의 3 차원 공간을 표현하기 위하여 사용되고 있는 옥트리 구조를 이용하여 공간을 분할하고, 이를 세부 수준으로 나누어 처리하기 위한 LOD(Level of Detail)와 CLOD 개념을 이용하여 외부지형을 폴리곤으로 표현하는 방법에 대한 처리 기법과 가시성 판단이나 충돌 검출을 위한 방법을 제시하려 한다.

1. 서론

3D 게임엔진을 이용하여 실시간으로 그때 그때 상황에 따라 배경 및 캐릭터의 모습을 처리한다는 것은 PC 의 연산 처리 속도와 메모리의 한계로 많은 어려움이 따른다. 그래픽 가속기의 등장과 최적화된 알고리즘과 어셈블리로 구성된 라이브러리를 사용하므로 실시간 렌더링이 어느 정도 가능하게 되었다. 3D 게임 엔진과 같은 실시간 상호 연동되어 작동하면서 초당 30 프레임 이상을 보여줄 수 있도록 처리해야 한다. 실시간 렌더링을 하기위해서는 그래픽 라이브러리를 사용하여 프로그래밍을 통하여 구현한다. 본 논문은 그래픽 라이브러리로 Microsoft 의 DirectX 와 C++을 이용하여 3D 게임엔지에서 필요한 3 차원 외부 지형 및 개체의 폴리곤을 동적으로 생성시키고 가시성 판단이나 충돌 검출을 위한 방법을 제시하려고 한다.

2. 3 차원 공간 지형 및 개체의 표현 방식

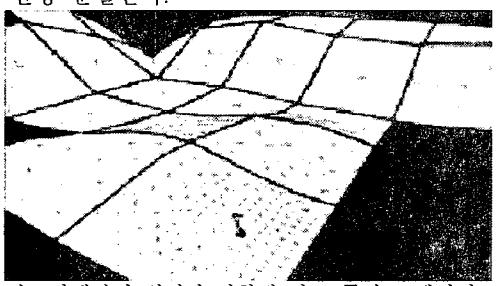
3 차원 공간 상에 사용되는 거대한 지형 데이터와 개체를 실시간으로 처리하기위해서는 많은 어려움이 따른다. 고려해야 할 사항을 세가지로 요약한다면 다음과 같은 것들이 있다. 우선은 3 차원 게임에 사용되는 모델의 폴리곤 수를 최대한 줄여야 한다. 실사와 같은 영상물을 얻기 위해서는 폴리곤 수가 많으면 많을수록 좋겠지만 게임과 같은 환경에서는 무엇보다도 속도가 문제가 되기 때문에 될 수 있으면 폴리곤 수를 줄인다. 두 번째로 미리 계산할 수 있는 것들은 미리 계산하여 처리한다. 실제로 광원의 처리 같은 경우 이

를 계산하는 것은 많은 시간을 필요로 하기 때문에 실시간 3 차원 그래픽에 사용하기에는 적당하지 않다. 세 번째로 공간을 쪼개서 렌더링 파이프라인에 들어가는 물체의 수를 줄이고 카메라의 뷰 볼륨 안에 있는 물체만 렌더링 하도록 한다. 이러한 부분들은 게임에 있어 그래픽처리성능을 향상시키는 아주 중요한 요소이다. [3][7][10]

3. 3 차원 가상 공간의 지형데이터를 처리하는 방법

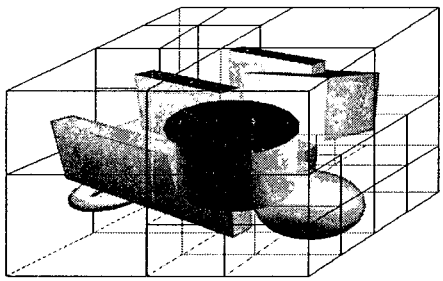
본 논문은 3 차원의 가상 공간의 지형데이터를 실시간으로 처리할 경우 지형을 고정적인 정점 좌표들을 주어서 표현할 경우 지형이 광범위해서 당연히 속도 저하가 생긴다. 광범위한 지형의 각 정점의 좌표를 거리에 따라 효율적으로 적절한 폴리곤 수를 조절할 수 있는 데이터 저장방법이 필요하다. 지형데이터를 표현할 때 지형의 높이 정보만을 저장하고 폴리곤은 내부적인 알고리즘으로 생성해 내는 방법이다.

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 행렬을 계산해야 하므로 속도가 저하된다. 이를 개선하기위한 방안으로 Height field 에 의해 저장된 점 좌표를 8 개의 화면으로 균등 분할한다.



[그림 1]카메라의 위치와 방향에 따른 폴리곤 생성의 변화

옥트리의 렌더링의 기본 개념은 일단 가장 최상 단의 옥트리 부터 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. 8 개의 점이 모두 밖에 있다면 그 안의 옥트리는 화면에 보이지



[그림 2] 주위의 입방체로부터 세부수준으로 분할된 모습

않는게 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 포함연산을 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 문턱값을 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 지형데이터를 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.

① 1 단계 구현

- 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
 for ( x = nStartX; x < nEndX; x++ ) {
    for ( z = nStartZ; z < nEndZ; z++ )
       MATRIX_TransformWithHomogenous (
```

```
pCamera->W2P,
pWorld->pVertex[z * pWorld->nWidth + x] );
    }
}
```

pCamera : 카메라 개체
pCamera->POV : 시점자의 위치(PointOfView)

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;
}
```

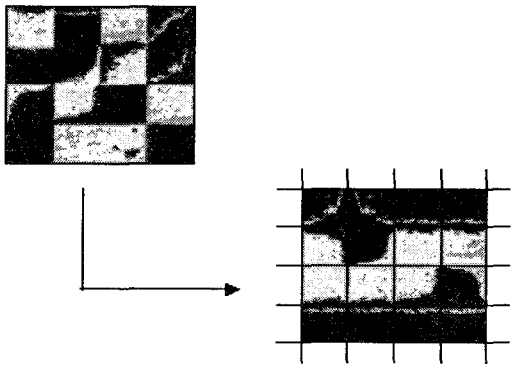
④ 4 단계

- 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 개체 사이의 거리에 어떠한 문턱 값을 적용하는가에 있다. 이 방법에는 다음과 같은 문제점을 내포하고 있다. 첫째로 카메라와 시야사이를 고려하지 않은 부분으로 개체가 카메라로부터 얼마나 떨어져 있다고 해도 카메라와 시야 각이 매우 작으면 화면상에 나타난 개체의 크기가 생각보다 크기 때문에 저 수준 모델을 사용해서는 안 된다.
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.

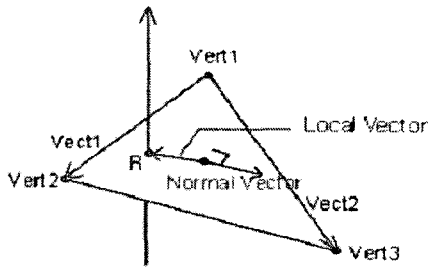
4. 3 차원 지형의 충돌 처리

본 논문에서 제시하는 3 차원 지형의 충돌방식은 3 차원 지형의 Height field 에 의해 지형데이터를 일정한 크기의 폴리곤으로 생성하는 페이지형식의 지형으로 생성하게 된다. 즉 3 차원 지형을 하나의 격자 칸(사각형)의 맵으로 분할되어 링크드 리스트로 저장될 수 있다. 3 차원 공간상의 좌표로 맵의 타일번호를 알아낸다. 객체들간의 충돌 검출 시 그 객체들이 있는 칸을 기준으로 충돌 판정을 할 수 있다. 이 경우 충분히 가까운 위치에 있는 것이 틀림없는 객체에 대해서만 충돌 판정을 시도하면 되므로 총 충돌 판정회수를 줄일 수 있다.



[그림 3] 맵소스를 타일처럼 맵핑

객체들의 충돌 판별 시 사용되는 식으로는 폴리곤의 평면방정식을 이용할 수 있는데 다음과 같다.



[그림 4] 지형과의 충돌검출

AX+BY+CZ+D=0
A : 평면 방향을 가리키는 벡터(normal 벡터)의 x 성분(즉, x)
B : 평면 방향을 가리키는 벡터(normal 벡터)의 y 성분(즉, y)
C : 평면 방향을 가리키는 벡터(normal 벡터)의 z 성분(즉, z) D : 평면 상수
X : 평면 상의 임의의 한점의 x 좌표
Y : 평면 상의 임의의 한점의 y 좌표
Z : 평면 상의 임의의 한점의 z 좌표

평면의 방향을 나타내는 noraml 벡터 즉, A,B,C 를 구하려면, Cross Product 를 이용해서 구한다. 평면의 한 점인 X,Y,Z 는 주어진 삼각형의 세 정점을 넣어 주면 된다. 상수인 D 는 아래와 같이 구할 수 있다.

D = - (AX + BY + CZ) 로 표현할 수 있다.

거리 = |((ax+by+cz+d))| / sqrt(a^2+b^2+c^2)

여기서 a,b,c 는 법선 벡터 값이고, D 는 그의 상수 값으로 X,Y,Z 는 그 평면과 떨어져있는 점으로 이 점과 평면사이의 거리를 구하면 된다.[11][12][13][14]

```
float HEIGHTFIELD_FitY(HEIGHTFIELD*
                                pWorld,float fX,float fZ)
{
  ...
   nTileX = (int)( fX / pWorld->fTileWidth );
   nTileZ = (int)( fZ / pWorld->fTileDepth );
// 유효한 범위안에 있는지 여부 체크...
   if ( nTileX < 0 || nTileX >= pWorld->nWidth - 2 )
     return 0;
   if ( nTileZ < 0 || nTileZ >= pWorld->nDepth - 2 )
     return 0;
   fDeltaX = fX - ((float)nTileX * pWorld-
                                  >fTileWidth );
   fDeltaZ = fZ - ((float)nTileZ * pWorld->fTileDepth );
//해당 방향을 나타내는 normal 벡터를 구한다.
   VECTOR_Make ( pVertex2, pVertex1, &U );
   VECTOR_Make ( pVertex3, pVertex1, &V );
   VECTOR_CrossProduct ( &U, &V, &N );
   VECTOR_Normalize ( &N );
// 평면 방정식을 이용해, Y 좌표를 얻어 낸다.
D = -( N.x*pVertex1->x+N.y*pVertex1->y+
                                N.z*pVertex1->z );
fY = -( N.x * fX + N.z * fZ + D ) / N.y;
}
```

5. 결론

본 논문은 3 차원 게임엔진설계에서 발생되는 속도 문제를 어느 정도 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 있는 데이터 저장방법과 메모리의 오용이 적고 클리핑 및 충돌 처리 시 폴리곤을 정확하고 빠르게 검출해 낼 수 있다. Height field 는 가시성 판단이나 충돌 검출, 개체관리에 효율적이다. 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 문턱 값이 적용된 CLOD 가 적용되게 된다. 본 논문에서 제시된 방법은 현재 3 차원 가상공간에서의 외부지형과 개체표현 및 가시성 판단이나 충돌 검출을 위한 방법으로 적용한다면 효율적인 처리 능력을 발휘할 수 있을 것이다.

향후 과제로 카메라의 시점과 거리의 변화에 따른 동적인 충돌처리를 위한 방법들을 처리할 수 있는 처리 기법을 제시하려 한다.

[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,최혁종, Real Time 3D Graphics Technique.1999.5 URL: http://www.3dartisan.com/

[11] A Survey; M. Lin and S. Gottschalk. Collision Detection between Geometric Models: Appeared in the Proceedings of IMA Conference on Mathematics of Surfaces 1998.

[12] M. Lin, D. Manocha, J. Cohen and S. Gottschalk. Collision Detection: Algorithms and Applications; Appeared in Proc. of Algorithms for Robotics Motion and Manipulation, pp. 129-142 eds. Jean-Paul Laumond and M. Overmars, A.K. Peters (invited submission)

[13] J. Cohen, M. Lin, D. Manocha and K. Ponamgi, Interactive and Exact Collision Detection for Large-Scale Environments; Technical report TR94-005, Department of Computer Science, University of N. Carolina, Chapel Hill.

[14] J. Cohen, M. Lin, D. Manocha and K. Ponamgi, I-COLLIDE: An Interactive and Exact Collision Detection System for Large-Scaled Environments; Proceedings of ACM Int. 3D Graphics Conference , pp. 189-196, 1995.

Efficient Collision Detection for Animation and Robotics; M. C. Lin