온라인 분산게임 서버의 충돌처리 설계

The Collision Processing Design of an Online Distributed Game Server

이승욱
동명정보대학교 게임공학과

Sung-Ug Lee(sunlee@tit.ac.kr)

요약

최근의 MMORPG 게임은 심리스 월드로 분할하는 분산 서버를 구축하고 있다. 본 연구는 이러한 분산 서버 간의 공유영역에 해당되는 지역에 대한 충돌처리를 다룬다. 분산 서버간의 공유 영역에 대한 경계영역을 동적으로 조정하기 위해 DLS을 사용하고, 광선과 단말 노드 간의 충돌 위치 관계를 통하여 이웃 노드를 빠르게 탐색한다. 이렇게 구해진 노드의 값을 통하여 객체 간의 충돌처리를 판별한다. 이것은 각 서버가 공유 영역에 대한 정보를 계속 보유할 필요가 없고, 서버간의 경계 영역을 포인터를 이용하여 빠르게 탐색할 수 있게 한다. 충돌은 계층적 경계상자를 이용하여 인접한 개체의 값들을 그룹으로 이진트리로 구축한다. 이러한 처리 방법은 처리량을 이분화 시켜 효과적으로 처리량을 줄일 수 있다. 또한 실시간적으로 발생되는 개체 정보의 변경 시 공유영역에 대한 복사가 필요하지 않으므로 네트워크 트래픽에 대한 처리량도 효과적으로 줄일 수 있다.

■ 중심어 : | MMORPG | 심리스 월드 | 분산처리 | DLS(Dynamic Load Sharing) | 계층적 경계상자 |

Abstract

Recently, a MMORPG(Massively Multi-play Online Role Playing Game) has built distribute server by Seamless world. This paper proposes an efficient collision detection method. DLS is used to dynamically adjust spatial subdivisions in each the boundary regions of distribute server. We use an index table to effectively utilize the relationships between in the nodes and can perform the collision detection efficiently by reconstructing nodes of the tree. Also, we maintain the information for the boundary region to efficiently detect the collections and adjust the boundary regions between distributed servers by using DLS. As the DLS uses pointers, the information for each server is not needed and the boundary regions between the distributed servers are efficiently searched. Using node index points, the construction table can be made to find between ray and neighborhood node, In addition, processes for Network traffic reduce because a copy of the boundary regions is not needed when a object moves with realtime.

■ Keyword : | MMORPG | Seamless World | Distribute Processing | DLS(Dynamic Load Sharing) | HBB (Hierarchical Bounding Box) |

* 이 논문은 2005학년도 동명정보대학교 학술 연구비 지원에 의하여 수행되었습니다.

접수번호 : #050817-001
접수일자 : 2005년 08월 17일
심사완료일 : 2005년 10월 24일
교신저자 : 이승욱, e-mail : sunlee@tit.ac.kr

I. 서 론

3D MMORPG 게임과 같이 수천 명이 동시 접속 가능한 게임에서의 서버의 처리는 모든 연산이 복수의 프로세스나 복수의 컴퓨터로 나누어 처리하는 분산 환경이 필요하며, 효율적인 처리를 위해서는 인접한 지역의 정보를 효과적으로 처리하는 방법이 필요하다. 특히 인접영역에 대한 공유영역을 설정할 경우 최소한 클라이언트가 게임 월드나 참여자가 인식할 수 있는 범위만큼은 공유영역이 설정되어야 한다. 만약 이보다 작아진다면 서버 경계에 가까이 있는 참여자는 경계의 반대편에 있는 다른 참여자가 볼 수 있는 것을 볼 수 없는 경우가 생길 수 있기 때문이다. 그리고 충돌처리에 대한 요구는 캐릭터가 충분히 높거나 그렇지 못한 출입구를 지나려 할 때와 같이 객체가 입구를 통과할 수 있는지 여부를 판별하는 경우 등 다양한 부분에서는 보다 정밀한 처리가 요구된다. 이로 인하여 공간 차원환경과 시간 차원환경에서 충돌처리의 요구는 증가되고 있고, 서버와 클라이언트의 처리 속도에 많은 영향을 미치고 있다. 본 논문에서는 기존의 연구들이 주로 하나의 서버에서 게임 월드내의 서버의 처리에 주안점을 두고 충돌처리 방법을 다룬데 비해 본 연구는 넓은 게임 월드를 갖는 3D MMORPG 게임 엔진에서 게임 월드의 여러 서버간의 동적공간에서의 충돌처리에 관한 처리 방법을 다룬다. 이를 위하여 분산처리 개념을 바탕으로 본 연구에서는 공간을 입체적으로 분할하여 탐색할 수 있는 알고리즘과 서버간의 동적 경계를 조정하기 위한 효과적인 처리 방법을 통하여 분산 환경에서의 충돌처리를 다룬다[1].

II. 3D 온라인 게임의 처리 시스템

본 연구에서는 3D MMORPG 유형의 게임 환경의 서버 클라이언트모델을 통하여 분산 서버 처리 개념과 입체적인 공간 분할방법에 관해 다루었다. 이것은 게임월드의 처리 대상을 제한하기 위한 효율적인 방법이 될 것이다. 분산 서버를 통하여 운영되는 온라인게임은 게임 월드에 참여하는 참여자 상호 간의 시각과 게임 월드의 동적 상태 관리를 통하여 게임이 이루어지게 된다. 즉 대규모 온라인 게임 시스템에서 네트워크 대역폭과 프로세스 성능 및 서버와 클라이언트의 자원관리 처리는 게임의 핵심 기술이다. 따라서 대규모 온라인 게임에서는 최상의 성능을 발휘할 수 있게 하기 위해서 온라인 게임의 특성에 따라 자원을 효과적으로 관리할 수 있는 기법이 고려되고 있다. 이를 위한 방법으로 한 대의 컴퓨터가 갖는 동시 접속 참여자의 수를 제한하고 게임 월드의 참여자를 여러 대의 서버로 분산시켜 게임의 성능을 유지시키는 처리 방법의 고려가 필요하다[2].

1. 분산 처리 시스템 유형

일반적인 온라인 게임에서 사용되고 있는 분산 시스템 유형인 존(Zoned)월드는 다른 영역으로 이동은 포탈을 통하여 이루어진다. 공간에 대한 분할 영역은 정적으로 이루어지며, 공간은 방으로 처리된다. 이와는 다른 처리 방법인 심리스(Seamless) 월드는 공간들을 분할하고 분할된 공간을 서버들 간의 행동을 자동적으로 이루어지도록 서버 간의 통신 수단을 추가하여 각 서버가 다른 서버의 경계와 가까운 물체들이 무엇인지 서로 알려주며, 다른 곳에 있는 물체에 대한 처리를 자신이 갖고 있는 복사본을 가지고 수행할 수 있게 함으로써 실현한다.

1.1 3차원 월드의 공간 표현법

3차원 공간을 분할하기 위한 처리 방법으로 첫 번째는 폴리곤 개수를 기준으로 공간을 재귀적으로 분할시키는 처리 방법과 옥트리와 같은 트리 구조를 이용하여 분할 수준의 한계를 미리 결정하는 처리 방법이 있다. 본 연구에서는 트리의 루트 노드의 경계 입방체가 보인다고 판정된다면 트리를 탐색해 가면서 자식 노드를 탐색한다. 이러한 처리 과정을 일정한 한계 수준까진 만 재귀적으로 분할한다. 그리고 화면을 단순히 분할하지 않고 폴리곤의 개수를 기준으로 균등 분할을 함으로써, 메모리의 오용이 적고 클리핑 및 충돌 처리 시 폴리곤을 정확하고 빠르게 검출할 수 있다. 이렇게 하기 위해서는 객체에 대한 처리는 낮은 레벨에서 발생해야 한다[2-4].

2. 분산 서버에 대한 분할영역의 탐색

3차원 게임 공간의 분할 영역에 대한 영역을 탐색하기 위해 단말 노드와 이웃 노드의 포인터 계산 알고리즘을 이용하여 노드 및 서버간의 경계영역을 탐색한다.

2.1 포인터를 이용한 노드의 탐색

3차원 공간의 표현을 위해서는 처리를 단순화 시켜야 한다. 이것은 공간을 분할 시켜 처리 영역을 제한시키는 것이다[4][7].

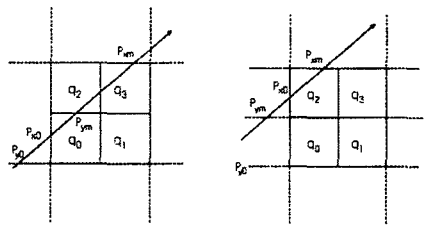

그림 1. 분할영역에 대한 진입면의 결정

이러한 제한된 처리 영역을 탐색하기의해서는 [그림 1]과 같이 직선의 방정식을 이용하여 광선이 노드에 진입하는 정점과 경유하는 역역을 찾는다. [그림 1]이 가리키는 값은 p_{x0}, p_{y0}, p_{z0}의 비교에 의해 광선의 노드에 대한 진입 면을 구하고, 진입 면이 결정되면, 진입하는 노드를 분할하는 면과의 정점인 p_{xm}, p_{ym}, p_{zm}과의 관계에 의해 광선이 경유하는 자식 노드 영역을 찾을 수 있다. 각 면에 대한 진입 값 p_0, p_1의 분할 면에 대한 p_m은 두 값의 평균이다. 자식 노드 중 p_0과 p_1은 인덱스 값에 따라 p_0, p_1, p_m중에의 값이다. 현재 노드의 이웃 노드는 광선의 출구 면에 의해 결정된다. 광선의 출구 면은 광선의 진입 면을 결정하는 요소로 p_{x1}, p_{y1}, p_{z1}의 최소값이 광선의 출구 면을 나타낸다. 축에 맞춰진 경계 상자가 원점을 중심으로 $0 \le i \le 2$일 때 범위를 e_i라고 한다면, 상자에 의해 채워지는 공간 범위는 $[-e_0, e_0]\times[-e_1, e_1]\times[-e_2, e_2]$이 된다. 이것은 세 개의 평행면들에 대해 한 번에 한 쌍씩 $t \in [0, 1]$일 때 광선 $(p_0, p_1, p_2)+(td_0, d_1, d_2)$을 테스트한다. 광선의 초기 간격은 $[t_0, t_1]=[0,1]$이며, $d_0 \neq 0$이면 광선은 면과 평행하지 않으며 교점이 $t_i=-(e_0+p_0)/d_0$일 때 발생한다. 또한, $d_0>0$이면 x_0가 증가함에 따라 광선 매개변수 t가 증가한다. $t_i > t_1$이면 선분은 면 밖에 있으며, t_0에서 조정이 필요치 않다. 그렇지 않은 경우, $t \in [t_0, t_1]$이며 최소 매개변수 값은 $t_0 = t_1$로 갱신된다. $t_i = t_1$인 경우, 광선은 한 점에서 면과 충돌을 통하여 영역을 결정한다. 처리 방법은 먼저 광선과 루트 노드와의 교점을 p, 각 노드의 중점을 m, 광선과 루트 노드와의 교점 Δd는

$$\Delta d = p - m(node) \qquad (1)$$

Δd의 부호에 의해 자식 노드의 인덱스가 결정된다. 즉 $sign(\Delta d_x)$는 Δd의 부호가 음이면 0, 양이면 1을 반환한다.

$$(sign(\Delta d_x)\ll 2)\,|\,(sign(\Delta d_y)\ll 1)\,|\,sign(\Delta d_z) \quad (2)$$

식(2)의 연산을 통하여 +방향으로 이동 할 경우 이진 비트 값으로 10(2)의 값을 가지고, -방향으로 이동할 경우 01(1)을 가진다. 평행이동 시에는 00(0) 값으로 n은 최대 1010(6)의 값을 가질 수 있다.

[그림 2]는 최종 단말 노드가 검출될 때까지 식 (2)을 이용하여 검출한다. 검출된 노드가 있다면 currentNode를 돌려준다.

```
procedure get_node(Node currentNode, Vector P)
begin
  while (currentNode != TEMIAL) do
  begin
    Vector D=p-mid_point(currentNode);
 (sign(Δd_x)<<2 | (sign(Δd_y)≪1 | sign(Δd_z);
  currentNode = n-th child of currentNode;
  end
  return currentNode;
end
```

그림 2. 현재 노드 인덱스를 구하기 위한 알고리즘

2.2 이웃 노드 탐색

현재 노드에 대한 다음 노드는 탐색의 경우는 다음과 같은 세 가지 경우 중 한 가지 유형이다. 첫째, 면을 공유하는 노드를 광선이 경유할 경우. 둘째, 광선이 모서리를 경유하여 다음 노드로 진입할 경우. 셋째, 광선이 꼭지점을 통과하여 다음 노드로 진입할 경우이다[5][6]. 이들 경우 모두 각축에 대해서 + 방향과 - 방향이 있다. 출구면의에 대해 + 방향을 2로 0의 - 방향을 1로 축과 평행인 방향은 0으로 하면 이 값을 이용하여 광선이 현재 공간의 어느 곳을 경유하는지 알 수 있다.

[그림 3]은 옥트리로 3차원 공간을 분할한 모습과 이동시 갖게 되는 영역에 대한 포인 값이다. 최종 단말 노드가 검출될 때까지 3차원 공간에 대한 노드를 검출한다. 현재 노드와 자식 노드 간의 교점 식을 구한다. 교점은 식 (1)에 의하여 재귀적으로 currentNode의 값이 8 이하의 서버 노드를 검출하기 위하여 노드를 탐색한다. 이웃 노드의 탐색은 다음의 세 가지 조건을 만족하는 대상 면을 가진 면의 노드와 이웃으로 간주한다.

첫째, 면들의 법선들이 반대 방향이다. 둘째, 원본 면의 모든 정점들이 대상 면의 위 또는 안에 놓여 있다.

셋째, 원본 면의 크기가 대상 면의 크기보다 작거나 같다. 그리고 옥트리를 이용하여 충돌 면을 검출하기 위해서는 입방체의 여섯 개의 면들을 공유하는 여섯 개의 이웃 노드가 필요하다. 입방체의 한 면의 이웃은 그 입방체보다 크거나 같은 크기의 노드이어야 한다. 더 작은 노드는 이웃 노드가 될 수 없다. 이웃 노드를 연결시키는 알고리즘은 옥트리의 각 노드 면을 트리의 동일한 또는 높은 수준의 다른 노드면 들의 비교를 통하여 조건을 만족하는 면들을 이웃으로 연결시키는 것이다.

[표 1]에서와 같이 현재 노드에서 x, y, z의 이동 위치에 따라 단말 노드의 위치를 쉽게 찾아 갈 수 있도록 정의 했다. 그리고 이웃 노드를 탐색하기 위하여 현재 노드가 3이고 광선이 현재 노드의 $+x$축으로 향한다면 이웃 노드는 7이 탐색된다. [그림 3]은 식 (2)에 의하여 부모 노드와 자식 노드 값을 보여주고 있다.

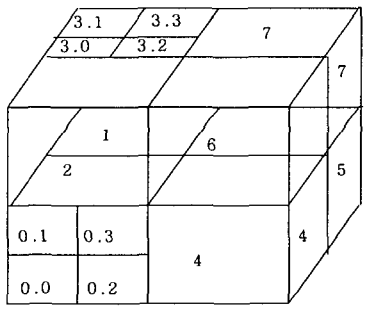

그림 3. 식 (2)에 의한 노드 값

표 1. 이웃 노드의 인덱스

출구면	XYZ	노드 인덱스	
000000			
000001	-z	3->0, 1->4	0->6, 4->2
000010	+z	0->3, 3->5	1->7, 2->4
000100	-y,	1->3, 3->2	0->4, 2->6
000101	+y	0->1, 2->3	1->5, 3->7
000110	-x	1->5, 0->4	3->1, 2->0
010000	+x	1->3, 3->7	0->2, 2->6
000110	-y,+z	2->4, 3->5	3->6, 1->4
001010	+y,+z	0->4, 1->5	2->6, 3->7
010010	-x,+z	1->5, 0->4	3->1, 2->0
100010	+x,+z	0->2, 1->3	4->6, 5->7
010100	-x,-y	3->0, 1->4	0->6, 2->4
011000	-x,+y	2->1, 3->5	1->7, 0->5
100100	+x,-y	1->2, 2->4	3->5, 0->6
101000	+x,+y	0->1, 2->3	4->5, 6->7

표 2. 노드의 이동면

현재 서버노드 상태	출구면 YZ	출구면 XZ	출구면 XY
0	4	2	1
1	5	3	END
2	6	END	3
3	7	END	END
4	END	6	5
5	END	7	END
6	END	END	7
7	END	END	END

[그림 3]과 [표 2]에서, 서버 노드가 1일 때 XY평면을 광선의 출구로 한다. 그때 좌표는 1 -> 5로 이동된다. 서버 노드가 1일 때 XZ평면을 광선의 출구로 하면 좌표

는 1 -> 3으로 이동된다. 그리고 서버 노드가 1 일 때 XY 평면을 광선의 출구로 할 때 서버 노드 1은 이미 XY 평면에 위치하므로 그 자체가 광선의 출구가 된다.

2.3 포인터를 이용한 부모 노드의 탐색

[표 1]의 경우 다음과 같은 문제점을 가지고 있다. 테이블에 해당되는 노드는 동일한 부모를 가진 이웃 노드들이다. 그러나 테이블에 표시되어 있지 않은 경우에는 부모가 다른 이웃 노드를 의미한다. 부모가 다른 이웃 노드를 구하기 위해서는 식 (1)을 사용하여 노드 값을 구한 후 이웃한 부모의 노드에 해당하는 값을 얻을 수 있어야 한다. 그러므로 현재 노드의 인덱스만 알면 다음 노드가 현재 부모에 속하는 노드인지 알 수 있다. 여기에서 구해진 인덱스 값을 이용하여 식 (2)에 대입하여 노드 포인터를 얻는다. 현재 노드가 경계 영역에 있지 않다면 이웃 노드에 대한 포인터는 일반적인 탐색 방법을 이용하여 탐색한다.

그러나 경계 영역에 존재한다면 이웃 노드의 부모 포인터는 중요한 의미를 지닌다. 경계영역에 대한 이동과 다른 분산서버로의 이동시 부모 포인터는 중요하게 다루어진다. 동일한 레벨의 노드의 이동을 가능하게 하기 위해서는 [표 1]에서 제시된 포인터 값과 [그림 4]의 인접서버에 대한 포인터에 대한 처리의 적용으로 구할 수 있다.

```
procedure proc_termial(Node *n, Message *key)
begin
while (true)
  if (n==0 && Right_Move) then
    return proc_termial(Node_point→Node_point+1);
  else if(n==0 &&Down_Move) then
    return proc_termial(Node_point→Node_point-2);
  else if(n==1 && Left_Move)
    return proc_termial(Node_point→Node_point+1);
  else if(n==1 && Down_Move)
    return proc_termial(Node_point→Node_point-2);
  else if(n==2 && Right_Move)
    return  (Node_point→Node_point+1);
  else if(n==2 && Up_Move)
    return proc_termial(Node_point→Node_point+2);
  else if(n==3 && Left_Move)
    return proc_termial(Node_point→Node_point+1);
  else if(n==3 && Up_Move)
    return proc_termial(Node_point→Node_point+2);
  end if
end while
end
```

그림 4. 인접한 서버 간의 이동을 위한 포인트

3. 동적 분할 시스템 설계를 위한 처리 방법

동적으로 공간을 분할 할당하는 처리 알고리즘을 DLS(Dynamic Load Sharing) 알고리즘이라 정의한다. DLS알고리즘은 k-레벨 서버에 의해 서버 S_i로 정의된다. [그림 5]은 N개의 동일한 크기의 셀로 나누어진 게임 월드를 보여준다. 서버 N은 $n \leq N$ 로 구성된다. S_1은 S_2의 이웃한 서버이다. 이를 다음과 같이 표현한다. 이웃 노드는 $NB(S_i)$로 표시하며, K는 서버의 레벨이다. S_i의 K는 서버의 레벨은 $D(S_i, K)$로 표시한다.

$$D(S_i, k-1) = \begin{cases} NB(S_i), \text{ if } k = 1 \\ D(S_i, k-1) \bigcup D(S_j, 1), S_i \in D(S_i, k-1), \text{ if } k > 1 \end{cases}$$

[그림 5]에 기술된 코드는 Seamless 월드의 DLS의 처리 알고리즘을 구현했다. 전체 서버가 할당될 때까지 서버 파티션을 나누고 각 서버는 파티션의 영역을 메모리에 적재하게 된다.

서버 레벨 S_2의 첫 번째 레벨로 $D(S_i, 1)$= $\{S_1, S_3, S_{10}\}$와 서버의 두 번째 레벨인 $D(S_2, 2)$ = $\{S_1, S_{10}, S_3, S_0, S_{18}, S_{11}, S_4\}$를 가리킨다.

3.1 동적 분산 서버 영역 처리

3차원 공간 영역을 분산 서버로 나누어 관리하기 위한 분할 공간을 8개의 영역을 나눈다. 트리의 단말 노드를 경유하여 이동하게 된다. [그림 4]는 네 개의 서버를 나타내며, 두 서버는 인접해있다. 노드 포인트는 식 (3)으로 구할 수 있다.

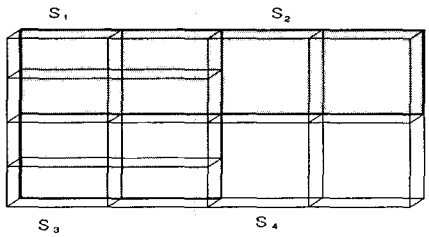

그림 5. 분산 처리를 위한 분할된 4개 서버와 경계 영역

[그림 5]의 S는 각 분산서버의 영역을 의미하며 식(3)은 경계면의 이동시 이웃 서버에서의 최종 단말 노드의 경계 영역에 해당되는 노드를 이동할 수 있도록 하기 위한 처리 식이다.

$$Node\_point = n_0 \times 4(8) + n_1 \times 4(8) + n_2 \times 4(8) + n_3 \ldots + n_i \quad (3)$$

n : 부모 노드

0~i : 노드의 레벨 값

서버의 경계 영역은 메모리에 적재되고 충돌처리와 같은 물리적인 처리를 위하여 참조된다. [그림 5]는 분할된 공간의 서버 간의 영역을 나타낸다. 색으로 표시된 영역이 공유 영역이다. 공유 영역에 위치한다면 DLS 알고리즘에 의해서 경계 영역에 해당되는 영역의 노드는 메모리에 적재된다. 공유 영역에 대한 서버 간의 노드 복사가 필요하다. [그림 6]은 서버 간의 경계 영역에 따라 이동 영역 나타낸다. D는 3차원 공간의 분산 서버를 의미하며, [그림 5]와 [그림 6]을 통하여 최종 단말 노드의 경계영역에 대한 이동 경로를 표현한 처리 알고리즘을 보여준다.

서버 1의 인접 노드 :
Right_Move→D(S_1→Node_point, S_2→Node_point+1)
Down_Move→D(S_1→Node_point, S_2→Node_point-2)
서버 2의 인접 노드 :
Left_Move→D(S_1→Node_point, S_2→Node_point+1)
Down_Move→D(S_1→Node_point, S_2→Node_point-2)
서버 3의 인접 노드 :
Right_Move→D(S_1→Node_point, S_2→Node_point+1)
Up_Move→D(S_1→Node_point, S_2→Node_point+2)
서버 4의 인접 노드 :
Left_Move→D(S_1→Node_point, S_2→Node_point+1)
Up_Move→D(S_1→Node_point, S_2→Node_point+2)

그림 6. 인접한 서버 간의 이동을 위한 포인트

```
procedure DLS_Server(Server sever_number, int node_level)
begin
 k=node_level; //서버의 레벨을 셋팅한다
 if ( k==TEMIAL) then
    DLS_Server(sever_number, node_level -1);
 endif
 while ( OL_i > 0) or ( k <= MAX_LEVELi)) do
 //send load_sharing to neighbors only
  for(each S_j∈D( i,1)) do
   initialize(socket(group,this));
   EL_i = send(load_sharing, OL_i, S_j, k);
   if ( EL_i > 0) then
     transfer( EL_i <= 0) break;
   endif
  end for
  k = k + 1;
end while
end
```

그림 7. 인접한 서버 간의 메모리 적재

[그림 7]은 [그림 5]와 [그림 6]에서 보여준 처리 알고리즘을 의사 코드를 이용하여 표현할 코드이다. 서버간의 경계 영역을 메모리로 적재하기 위한 처리를 보여준다. 만약 이웃한 서버의 경계영역에 해당되는 단말 노드가 존재하지 않는다면, 이웃한 단말 노드의 부모 노드를 검출하게 된다.

III. 계층적 경계상자를 이용한 충돌검출 처리속도의 개선

충돌 시스템 설계에서 고려하여야 하는 중요한 부분 중의 하나는 충돌검출에 사용되는 개체의 수가 많다는 것이다. MMORPG와 같은 유형의 게임은 거대한 지형에서 충돌에 관하여 처리하여야 하고 개체들을 충돌 그룹으로 충돌을 처리한다고 해도 많은 비용이 소요된다. 그러나 계층적 경계상자를 이용하여 처리를 단순화 시킨다면 충돌검출에 소요되는 비용을 감소시킬 것이다. 처리에 필요한 영역을 대상으로 계층적 경계상자끼리의 충돌을 검출하기 위하여, 선택된 노드를 계층적 경계상

자로 구분하여 가까이 있는 개체들을 [그림 8]과 같은 형식으로 한데 묶은 계층적 경계상자(HBB: Hierarchical Bounding Box)로 만든다. 이때 고려되는 부분으로는 첫째, 계층적 구조의 구축방향 둘째, 경계 볼륨의 구축 방향 셋째, 충돌정보의 처리방법 넷째, 게임과 같은 동적인 환경에서 충돌 예측에 관한 데이터의 유지 방법 등이 고려되어야 한다. 이 계층적 경계상자는 인접한 노드와 인접한 서버의 노드 포인터를 앞의 처리 알고리즘으로 얻을 수 있다.

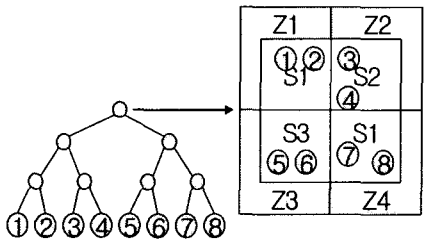

그림 8. 계층적 경계상자를 이용한 분산서버간의 충돌처리

충돌검출은 [그림 8]은 3차원 인접공간에 대한 분할된 공간이다. [그림 8]에서 분할된 공간을 서버가 다루는 영역을 나타내기 위한 영역으로 Z와 공유 영역에 해당되는 서버간의 영역을 S로 나타내었다. 이렇게 구성된 자료를 이용하여 충돌을 검출할 수 있다. 신 그래프의 객체 정보는 객체가 이동하게 되면 수시로 삽입과 삭제에 의하여 자료 값이 변화하게 되고, 객체가 움직임이 발생할 때 마다 현시점의 충돌에 관한 판단을 하게 된다. 빈번하게 발생되는 전체 트리의 노드 정보를 재구성하는 것은 효율적이지 못하다. 변경된 노드의 최하위 노드의 조상 노드인 LCA(Least Common Ancestor)의 정보만 계층적 경계 상자를 이용하여 구축함으로 처리를 단순화 할 수 있다. 이러한 처리 형태는 이진트리의 자료를 구축하지만 단지 왼쪽과 오른쪽의 이웃 노드를 구분하기 위한 형태의 분할방법이다. 분할된 공간의 단위는 경계 쌍을 이루는 단위가 되고, 경계 쌍은 일정한 경계 기준을 이루게 되므로 인접한 개체들끼리 묶어서 여러 개의 개체를 한데 묶은 경계상자들로 만들어 검사하는 것이 된다. 즉 계층적인 경계상자를 이용하면 모든 경계상자들을 충돌검출 대상으로 삼지 않아도 되고, 결과적으로 시간 복잡도는 $O(nlogn)$으로 떨어진다.

Ⅳ. 결 론

3D MMORPG 게임에서 분산 서버들 간의 동적 충돌처리는 서버간의 동적 경계 영역의 설정으로, 인접한 서버 간의 경계 영역을 DLS알고리즘에 이용하여 메모리에 적재한다. 이러한 처리의 결과는 기존의 분산 처리를 시스템에서의 충돌처리를 위한 공유 영역에 해당되는 객체의 정보를 모두 유지할 수 있으며,경계 영역의 크기에 따른 처리속도의 지연의 문제점을 보완할 수 있다. DLS 알고리즘은 공유 영역에 대한 객체의 정보를 메모리에 할 필요가 없으므로 이에 대한 서버의 처리량을 줄일 수 있다. 그리고 서버 간의 경계 영역에 대한 복잡한 구현 없이도 경계 영역을 탐색할 수 있으며, 처리식 만으로 빠르게 동적 경계 영역에 필요한 노드의 값을 구할 수 있다.

[표 3]의 Incremental Tree-Insertion은 빈 트리를 생성시켜 트리를 추가시켜 가며 충돌처리를 한다. Top-Down 계층적 구조 생성에 가장 많이 사용되는 처리 방식으로 처리 속도는 $O(nlogn)$을 보여준다. OBB 트리는 3차원 객체에 대하여 직사각형으로 제한된 박스이다. OBB는 객체를 감싸는 임의의 방향에서 가장 작은 경계 상자가 된다. AABB와 비교할 때, OBB는 일반적으로 더 작은 박스로 경계공간을 만들어 객체들을 묶을 수 있다. 최소 블록을 생성하는데 $O(nlogn)$이 소요되고, 이진트리의 깊이는 $O(nlogn)$이기 때문에 OBB 트리를 생성하는데 소요되는 총 시간은 $O(nlogn^2)$이다.

표 3. 충돌처리 방법의 비교[5-8]

충돌처리 방법	처리 방식	데이터 구조	시간 복잡도
Incremental Tree-Insertion	Tree-Insertion	Tree	$O(nlogn+a)$
OBB Tree	Top-Down	Binary Tree	$O(nlogn^2)$
Sweep-and-Prune	Bubble or Insertion Sorting	Sweep-and-Prune	$O(nlogn+k)$
본 연구 방법	Tree-Insertion, TBV	Octree,HBB	$O(nlogn)$

고정된 정육면체의 충돌처리를 위한 방법으로 Sweep-and-Prune 처리 방법은 3차원에서의 경계 박스 충돌 검사는 $O(nlogn+k)$가 소요된다. 즉 어떤 축상에 n개의 구간을 s_i, e_i로 나타낸다면, $s_i < e_i$이고, $0 \le i \le n$이다. 이를 오름차순으로 정렬한다. 모든 구간들을 정렬하는 데 $O(nlogn)$의 시간이 걸리고, 구간을 순회하는데 $O(n)$, k개의 충돌하는 구간을 저장하는 데 $O(k)$의 시간이 걸려서 총 $O(nlogn+k)$의 시간이 소요된다. 제시된 이론은 충돌처리에 필요한 소요 시간은 $O(nlogn)$이다[5-8].

결론적으로 OBB 트리의 경우 트리를 생성하는데 소요되는 총 시간은 $O(nlogn^2)$로 여기에 제시된 이론보다 많은 시간이 소요된다. 그러나 Incremental Tree-Insertion와 Sweep-and-Prune의 경우 비슷한 성능을 보여준다. 여기에 제시된 시간 복잡도는 표준 데이터를 이용한 것이다. 게임과 같이 객체의 정보가 균등하게 배치되어 있다면 제시된 본 연구 처리 방법은 높은 성능을 보여줄 수 있다. 그리고 서버 간의 경계 영역을 동적 분할하고 포인터를 이용하여 서버 간의 경계 영역을 빠르게 탐색할 수 있다. 광선과 단말 노드 간의 충돌 위치 관계를 통하여 인덱스 값을 알아내고 테이블을 이용하여 이웃 노드를 빠르게 탐색했다. 이렇게 구해진 노드의 값을 통하여 객체 간의 충돌처리를 판별했다. 결과적으로 시간 복잡도는 $O(nlogn)$이 소요된다. 제시된 이론은 충돌처리 뿐만 아니라 향후 분선 서버간의 처리 및 컬링과 같은 처리에 활용할 수 있을 것이다.

참 고 문 헌

[1] 양광호, 심광현, 고동일, 박일규, 김종성, "온라인 게임 서버의 기술동향", 전자통신 동향분석, 제16권, 제4호. pp.14-22, 2001.

[2] T. N. B. Duong and S. Zhou, "A dynamic load sharing algorithm for massively multiplayer online games," In Proceedings of the IEEE International Conference on Networks, pp.131-136, 2003.

[3] J. Sandor, "Octree Data Structures and Perspective Imagery," IEEE Computer Graphics & Applications Vol.9, No.4, pp.393-405, 1985.

[4] 이승욱, 박경환, "A Collision Detection Octree Partitioning Method using CLOD," Korea Information Processing Society, Proc. of the 16th KIPS FALL Conference, pp.615-618. 2001,

[5] A. Fujimoto, T. Tanaka, and K. Iwata, "ARTS: Accelerated Ray-Tracing System," IEEE Computer Graphics & Applications Vol.6, No.4, pp.16-26, 1986.

[6] H. Samet, "Implementing Ray Tracing width Octrees and Neighbor Finding," Computer Graphics, Vol.13, No.4, pp.445-460, 1989.

[7] F. W. Jansen, *Data Structures for Raster Graphics*, Springer Verlag, pp.58-72, 1985.

[8] J. Revelles, C. Urena, and M. Lastra, "An Efficient Parametric Algorithm for Octree Traversal," WSCG'2000 conference, pp.212-219, 2000.

[9] K. Sung, "A DDA Octree Traversal Algorithm for Ray Tracing," Proceedings of Eurographics 91, pp.73-85, 1991.

저 자 소 개

이 승 욱(Sung-Ug Lee) 정회원

- 1991년 8월 : 동아대학교 컴퓨터공학(공학석사)
- 2005년 2월 : 동아대학교 컴퓨터공학(공학박사)
- 2001년 3월~2005년 4월 : 동아대학교 BK21 교수, 초빙교수
- 2005년 4월~현재 : 동명정보대학교 게임공학과 전임교수

<관심분야> : 게임 알고리즘, 분산 네트워크