가상 환경 탐험 시스템 상에서 효과적인 충돌 탐지에 관한 연구

박남일[1],김동훈[2],이상락[3],성미영[4],박종승[5]
인천대학교 컴퓨터 공학과[1 2 3 4 5]
{nam1park[1],dinoman[2],srlee[3],mysung[4],jong[5]}@Incheon.ac.kr

Study on Efficient Collision Detection in Virtual Environment Navigation System

Namil Park[1], DongHoon Kim[2], Sangrak Lee[3], Meeyoung Sung[4], Jongseung Park[5]
Department of Computer Science and Engineering, Univ. of Incheon[1 2 3 4 5]

요 약

본 논문은 3 차원 가상 환경을 탐험하는데 있어서 빠르고 효과적으로 충돌 탐지를 검출하는 방법을 제안하고자 한다. 넓은 가상 공간상에서 개체가 증가하는 것에 비례하여 충돌 탐지의 계산 비용은 기하 급수적으로 증가한다. 이를 효과적으로 처리하기 위하여 BSP-tree 분할 방식과 경계 기둥을 사용한다. BSP-tree 분할 방식은 3 차원의 넓은 가상 공간을 여러 하위 공간으로 나누어 충돌 탐지가 이루어지는 공간을 축소한다. 이를 통하여 충돌 탐지 개체의 수가 증가하는 것에 따라 기하 급수적으로 증가하는 경계 기둥의 충돌 탐지 비용을 줄이는 효과를 얻을 수 있다. 경계 기둥은 축소된 하위 공간 상에서 개체간 실제 충돌 탐지가 일어날 가능성 및 충돌 여부를 빠르고 간편하게 판별하도록 한다.

Keyword : 충돌 탐지, 공간 분할, 경계 기둥

1. 서 론

컴퓨터 그래픽스는 의료용 어플리케이션, 군사 시뮬레이션, 로봇 시뮬레이션 등 여러 곳에서 활발히 연구 되고 있다. 이런 분야를 살펴볼 때 특징적인 것은 모두 가상 환경 속으로 인간이 쉽게 몰입이 가능하도록 초점을 맞추고 있다. 이것은 게임 분야에서도 예외 일 수 없다.

MMORPG(Massively Multiple Online Role Playing Game)나 FPS(First Person Shooter) 게임과 같은 3 차원 가상 현실 내에서 즐기는 게임을 살펴보면, 게임에 몰입을 위해서 많은 알고리즘을 사용한다. 그 중에서도 충돌 탐지는 개체를 만지거나 붙잡을 수 있는 효과, 움직이는 개체를 멈추어 서게 하는 효과, 움직이는 물체의 운동에너지를 제 2 의 개체에 전달하는 효과 등 실제 현실과 같은 물리 세계를 경험 할 수 있도록 해준다[1][6][7].

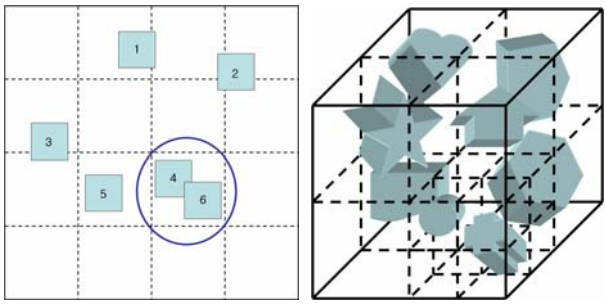
충돌 탐지는 크게 두 단계로 나뉘어 진다. 첫 번째 단계는 충돌 가능성 여부 탐지를 단순화하여 빠르게 찾아내는 단계로 나뉜다. 다음 두 번째 단계는 충돌 가능성이 있는 물체들의 충돌 여부 및 충돌 지점을 계산하는 단계로 나뉜다[6][7].

본 논문은 중력이 존재하는 3 차원 가상 공간 탐험을 목적으로 하는 시스템에서 탐험을 하는 개체와 공간상의 개체와의 효과적인 충돌 탐지를 제안한다. 이를 위하여 가상 공간을 BSP-tree(Binary Space Partitioning-tree)를 이용하여 분할하고, 경계 기둥을 사용하여 충돌 가능성 및 충돌 지점을 계산하도록 한다. 이 방법을 이용하여 넓은 3 차원 가상 공간에서 추가되는 개체의 수에 따라 기하 급수적으로 증가하는 충돌 탐지의 계산을 줄여 가상 공간을 탐험하는 것에 있어 효과적인 충돌 탐지를 할 수 있는 방안을 제시한다.

2. 충돌 탐지의 이론적 배경

2-1 가상 공간의 분할

가상 공간 상에서 충돌 탐지를 계산하여야 하는 개체 n이 늘어남에 따라서 충돌 탐지의 계산은 $O(n^2)$만큼 증가한다. 이렇게 기하급수적으로 늘어나는 충돌 탐지 계산을 효과적으로 줄이기 위해서는 먼저 공간을 분할하여 처리 영역을 줄여 계산되는 개체의 수를 줄이는 것이 효과적이다. 가상 공간의 분할은 근본적으로 2 차원과 3 차원상에서 각각의 공간을 분할하는 것은 크게 차이가 없다. 그림 1 의 (a)는 2 차원 상에서 공간을 분할하여 충돌 처리 영역을 줄여 충돌 탐지를 할 수 있도록 표현한다. 공간을 여러 하위 공간으로 나누었고, 같은 하위 공간을 공유하는 개체 4, 6 에 한해서 충돌 탐지 계산을 할 수 있도록 한다. 그림 1 의 (b)는 2 차원의 공간을 3 차원으로 확장하여 공간을 분할한 것을 표현한다[2][4].



(a) (b)

그림 1. 가상 공간을 분할하여 처리 공간을 축소

BSP-tree 는 가상 공간을 분할할 때 전체 공간을 반복적으로 하위 공간으로 나누어 자료를 구분하고, 구분된 자료를 다시 분류하여 묶는데 강력한 기능을 발휘한다[3].

BSP-tree를 사용하여 공간을 분할하는 방법을 간단히 도식화 하여 나타낸 것은 그림 2 와 같다. 전체의 공간 A는 X축에 의하여 두 하위 공간의 B와 C로 나뉘게 된다. 다시 하위 공간 B는 Y_1축에 의하여 하위 공간 D와 E로 나뉘게 되고, 하위 공간 C는 Y_2축에 하위 공간 F와 G로 나뉘게 된다. 이 방법을 반복적으로 수행을 하여 가상 공간을 여러 하위 공간으로 나누어 표현을 한다.

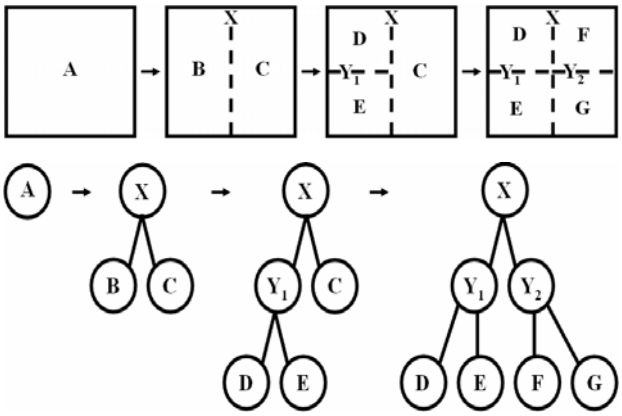


그림 2. 가상 공간을 BSP-tree 를 이용하여 분할

하위 공간을 나누는 방법 중 BSP-tree 와 유사한 관계를 갖고 있는 방법으로는 Quadtree 와 Octree 가 존재한다. Quadtree 는 4 개의 새로운 하위 공간으로 반복적으로 분할하는 방법이고, Octree 는 8 개의 새로운 하위 공간으로 반복적으로 분할하여 나타내는 방법이다.

2-2 경계 영역 처리

경계 영역이란 충돌 처리 개체를 둘러 싸고 있는 하나의 영역으로 충돌 탐지에서는 이 경계 영역이 서로 교차되는 것을 허용하지 않는다. 다시 말해서 이것은 경계 영역 내부에 포함되어 있는 개체들 간에 서로 충돌이 일어나지 않음을 의미한다[5].

경계 영역의 설정을 위해서는 먼저 두 점 사이의 거리를 계산하여야 한다. 각각의 점 $A(A_x,A_y,A_z)$와 $B(B_x,B_y,B_z)$가 주어졌을 때 두 점의 거리는 식 (1)에 의하여 계산된다.

$$Distance_{pp}(A,B) = |A_x - B_x| + |A_y - B_y| + |A_z - B_z|$$

$$or \qquad (1)$$

$$Distance_{pp}(A,B)^2 = (A_x - B_x)^2 + (A_y - B_y)^2 + (A_z + B_z)^2$$

3 차원의 공간에서 충돌 처리를 위한 경계 영역으로 간단히 구할 수 있고, 계산이 편리한 것으로 구를 널리 사용하고 있다. 이런 경계 구의 설정은 개체의 중심으로부터 가장 멀리 위치한 개체의 부분까지의 점의 거리를 식 (1)에 의하여 계산한다. 이 거리를 반지름 값으로 설정하여 개체를 둘러싸는 구를 구하게 되면 경계 구가 설정된다.

그림 3 은 개체를 포함 하는 경계 구를 나타낸 것이다.

그림 3. 개체를 포함하고 있는 경계 구

2-3 개체 간의 충돌 처리 절차

충돌 처리 절차에 있어서 충돌 가능성 여부를 탐지하는 것은 중요한 일이다. 이를 위해서 3 차원 각각의 개체에 경계 구를 설정하고, 이 경계 구와의 거리를 계산하는 것이 필요하다. 이 때 실제로 계산되는 것은 경계 구 S 와 한 점 P 에 대하여 거리를 구하는 것이다. 이것은 식 (2)에 의하여 계산한다.

$$Distance_{sp}(S,P) = \left| P - S_{center} \right| \quad (2)$$

이를 확장하여 구와 구 서로간의 계산은 식 (3)과 같다.

$$Distance_{ss}(S,T) = \left| T_{center} - S_{center} \right| \quad (3)$$

구와 구 서로간의 거리가 서로의 반지름의 합보다 큰 값을 유지하면 충돌이 일어나지 않게 되고, 반지름의 합과 일치할 경우 충돌이 발생한다.

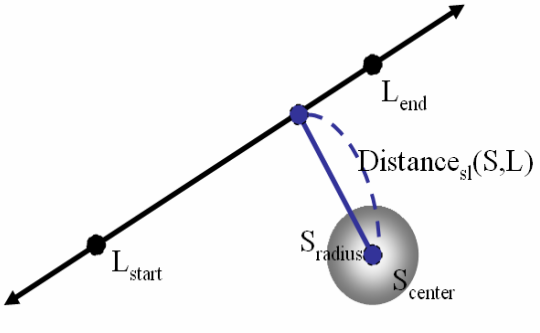


그림 4. 직선과 구와의 거리

구와 선과의 거리의 계산은 그림 4 와 같이 선과 구가 떨어져 있다고 가정할 때 구의 중심으로부터 선에 이르는 거리의 값을 구한다. 구의 중심으로부터 선에 이르는 법선과 만나는 선의 좌표값은 식 (4)에 의하여 구한다.

$$L_{start} + t(L_{end} - L_{start})$$

$$t = \left(\frac{(P - L_{start}) \bullet (L_{end} - L_{start})}{\left| L_{end} - L_{start} \right|^2} \right) \quad (4)$$

이에 의하여 구와 선과의 거리를 식 (5)에 의하여 구한다.

$$Distance_{sl}(S,L) = \left| L_{start} + t(L_{end} - L_{start}) - S_{center} \right| \quad (5)$$

충돌 여부의 검출은 각각의 t 값의 범위에 의하여 식 (6)과 같이 구분하고, 각각의 값이 구의 반지름보다 크면 충돌이 일어나지 않고, 구의 반지름과 같을 때 충돌이 발생한다.

$$\begin{aligned} &0 \le t \le 1, && Distance_{sl}(S,L) \\ &t \le 0, && \left| L_{start} - S_{center} \right| \\ &t \ge 1, && \left| L_{end} - S_{center} \right| \end{aligned} \quad (6)$$

구와 면 N 의 충돌 처리는 식 (7)에 의하여 계산된 거리로 이것이 구의 반지름 보다 크면 충돌이 일어나지 않고, 그의 반지름과 같게 되면 충돌이 발생한다.

$$Distance_{spl}(S,N) = \frac{\left| N_{normal} \bullet S_{center} + N_d \right|}{\left| N_{normal} \right|} \quad (7)$$

충돌 처리 계산을 한 후 서로의 충돌이 일어날 경우 충돌 반응을 보인 후 화면을 갱신하면 충돌 처리의 일련 과정이 완료된다.

3. 3 차원 가상 공간을 탐험하기 위한 효과적인 충돌 탐지

3 차원 가상 공간을 탐험하는데 빠르고 효과적으로 탐험이 이루어지기 위해서는 복잡한 3 차원의 공간 계산을 2 차원적으로 단순화하여 처리하는 것이 보다 효과적이다. 이를 위하여 가상 공간을 BSP-tree 를 이용하여 분할하고, 충돌 탐지 여부를 경계 기둥을 사용하여 계산한다.

3-1 3 차원 공간을 분할하기 위하여 BSP-tree 의 적용

가상 공간의 탐험을 위하여 공간을 분할할 때

에는 복잡한 3 차원 공간을 직접 여러 개로 분할하는 것 보다는 계산이 편리한 2 차원적으로 생각하여 분할하는 것이 유리하다. 이를 위하여 3 차원의 공간 좌표를 각각 X 축, Y 축, Z 축으로 설정하고, Y 축의 최대값에서 모든 물체를 XZ 평면상으로 투영을 시킨다. 다시 말해서 모든 y 의 값을 0 으로 설정을 하여 공간상에 위치한 모든 물체에 대해여 지표면으로 투영을 시킨다. 그림 5 은 3 차원 공간에 위치한 물체를 XZ 평면 상에 투영하여 간단히 나타낸 것이다.

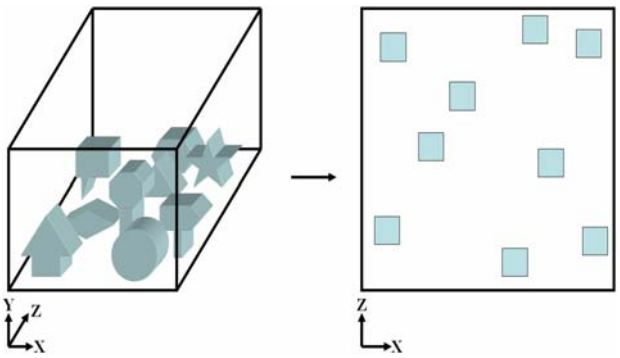
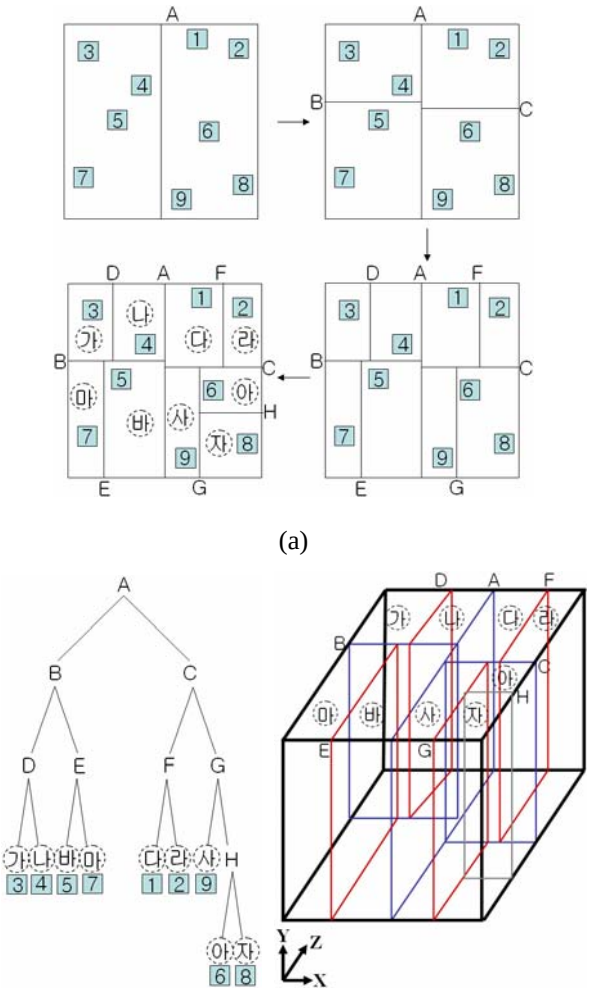


그림 5. 3 차원의 공간을 XZ 평면으로 투영시켜 2 차원으로 표현

2 차원으로 간단히 나타내어진 개체들의 분포에서 개체 하나가 남을 때까지 BSP 형태로 식 (8)에 의한 ZX 평면상의 직선을 이용하여 반복적으로 공간을 분할한다.

$$z = \alpha$$
$$x = \beta \qquad (8)$$

분할된 세부 공간은 다시 트리로 구성한다. 그림 6 의 (a)는 실제 개체의 위치에 따라서 공간을 분할하는 과정을 나타낸다. 그림 6 의 (b)는 이렇게 나누어진 하위 영역을 각각 BSP-tree 의 노드로 나타내고, 각각의 하위 영역에 포함된 충돌 가능성이 있는 물체를 나타낸다. 그림 6 의 (c)는 BSP 를 이용하여 실제로 분할한 가상 공간을 실제 3 차원 형태로 표현하여 나눈 형태로 간략하게 표현한 것이다. BSP 를 이용하여 공간을 분할하고 이것을 트리로 구성하는 것은 가상 공간 상에 분포하고 있는 충돌 처리 대상의 수를 줌으로써 빠른 계산의 이득을 가져올 수 있다.



(a)

(b) (c)

그림 6. BSP-tree 를 이용하여 공간을 분할

3-2 경계 기둥을 이용하여 충돌 개체 검출

경계 기둥은 탐험을 하며 충돌 탐지의 주체가 되는 개체에서 실제 충돌 탐지를 하여 충돌 검출을 구하는 부분이다. 경계 기둥의 설정은 XZ 평면상에서 충돌 탐지 대상의 중심으로부터 대상의 가장 먼 부분까지의 거리를 반지름으로 하는 원을 구한다. 또한 Y 축으로 개체에서 충돌이 일어나는 높이를 각각 Y 의 최소값 H1 과 최대값 H2 를 갖도록 한 후 이것을 중심으로 원기둥을 설정한다. 그림 7 은 충돌 탐지 대상의 경계 기둥 설정을 나타낸다.

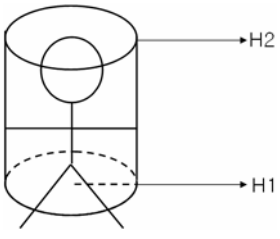


그림 7. 충돌 탐지 대상의 경계 기둥 설정

경계 기둥의 설정 후 3 차원의 가상 공간 탐험을 위한 충돌 탐지의 과정은 전체 공간의 높이 Y 축의 값을 실제 충돌 탐지가 일어나는 높이 H1 에서 H2 에 이르는 경계 기둥의 높이를 중심으로 공간을 분할한다. 분할된 공간을 XZ 평면상으로 투영시키기 위하여 모든 Y 의 값을 0 으로 설정한다. 3 차원 공간상의 모든 물체는 2 차원에 투영하여 표현하게 되면, 탐험의 개체에서 출발한 경계 기둥을 기반으로 한 원을 중심으로 개체간의 충돌 탐지 계산을 한다. 그림 8 은 3 차원 가상 공간에서 경계 기둥을 이용하여 충돌 탐지 과정을 나타낸 것이다.

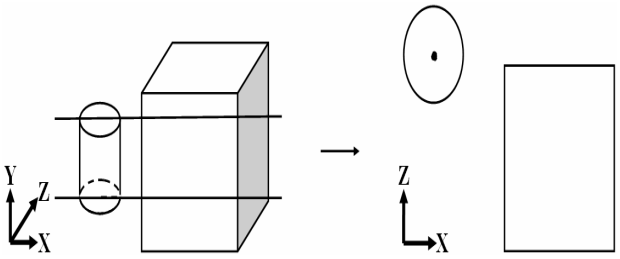


그림 8. 경계 기둥을 이용한 충돌 탐지

충돌 검출 후 처리를 하는데 있어서 문제가 되는 점은 처리의 대상이 되는 개체의 수가 너무 많다는 것이다. 경계 기둥을 이용하여 처리를 단순화 시킨다면 충돌 처리에 소용되는 처리 비용은 절감이 된다. 또한 BSP-tree 로 나누어진 세부 공간상의 물체를 복잡한 3 차원의 계산보다 단순한 2 차원적 계산을 통한 이득을 볼 수 있다.

3-3 구현

넓은 가상 공간에 펼쳐진 개체를 탐험하기 위해서는 효과적이고 빠른 충돌 처리가 필요하다. 이를 처리하기 위해서 첫 번째 단계로는 넓은 가상 공간 영역을 분할하여야 한다. 넓은 가상 공간은 BSP 를 이용하여 분할한다. BSP 를 이용하여 3 차원 공간을 분할하는 것은 3 차원의 복잡한 계산을 2 차원적으로 생각할 수 있는 장점을 준다.

두 번째 단계는 나뉘어진 하위 공간을 중심으로 개체 충돌이 일어날 수 있는 목록을 트리로 구축한다. 그림 9 는 BSP 로 분할된 공간을 트리로 구축한 코드를 표현한 것이다.

```
struct BSP_tree
{
        plane           partition;
        list            polygons;
        BSP_tree        *front, *back;
};

void Build_BSP_Tree (BSP_tree *tree, list polygons)
{
        polygon *root = polygons.Get_From_List ();
        tree->partition = root->Get_Plane ();
        tree->polygons.Add_To_List (root);
        list front_list, back_list;
        polygon *poly;
        while ((poly = polygons.Get_From_List ()) != 0)
        {
                int result = tree->partition.Classify_Polygon (poly);
                switch (result)
                {
                        case COINCIDENT: …; break;
                        case IN_BACK_OF: …; break;
                        case IN_FRONT_OF: …; break;
                        case SPANNING: …; break;
                }
        }
        if (! front_list.Is_Empty_List ())
        {
                …;
        }
        if (! back_list.Is_Empty_List ())
        {
                …;
        }
}
```

그림 9. BSP-tree 의 구축에 관한 코드

세 번째 단계는 경계 기둥을 구축하여 실제 분할된 공간상에서 충돌 탐지를 할 수 있도록 한다. 경계 기둥과 두 번째 단계에서 구축한 BSP-tree 내에 충돌 가능성이 존재하는 개체와의 실제 충돌 탐지를 계산한다. 그림 10 은 개체와 경계 기둥을 이용한 충돌 처리 의사 결정 코드이다.

```
bool TriInRange(float h1, float h2, float r, D3DXVECTOR3 pos,
                D3DXVECTOR3 opos, D3DXVECTOR3 vPoly[], D3DXVECTOR3 *res)
{
    D3DXVECTOR3 vTri2[3];
    float radius=r;
    float oldPos=Player.pos;
    float newPos=Player.pos+Time_Elapsed*Player.velocity;

        for (ict=0;ict<Level->numobj;ict++)
        {

                for(j=0; j<Level->obj[ict].numtri; j++ )
                {
                   vTri2[0] = Level->obj[ict].verts[Level->obj[ict].index[3*j+0]];
                   vTri2[1] = Level->obj[ict].verts[Level->obj[ict].index[3*j+1]];
                   vTri2[2] = Level->obj[ict].verts[Level->obj[ict].index[3*j+2]];

                   if (TriInRange(player.pos.y+3,Player.pos.y+9,radius,newPos,oldPos,vTri2,&res))
                   {
                      bnorm=newPos-res;

                      bnorm.y=0;

                      bignum=sqrt((bnorm.x*bnorm.x)+(bnorm.z*bnorm.z));
                      bnorm.x/=bignum;
                      bnorm.z/=bignum;
                      newPos=res+radiusf*bnorm;

                   }
                }
        }
}
```

그림 10. 경계 기둥을 이용한 충돌 처리 의사 결정 코드

이에 대한 처리 효율성은 다음과 같다. 가상 공간상에 존재하는 3 차원의 개체가 n개이고, 각각의 개체를 이루는 면의 개수를 평균 m이라 할 경우 시간 복잡도는 $O(n^2m^2)$을 이룬다. 반면 BSP-tree를 이용하여 공간을 분할하고, 경계 기둥을 사용하여 충돌 탐지를 하였을 경우 O(n*log*n)의 시간

복잡도가 나오게 된다. 또한 3 차원의 복잡한 계산을 단순한 2 차원적으로 이루어지기 때문에 기존의 방식보다 효율성이 뛰어나다. 가상 환경을 탐험하는 시스템 내에서 이러한 방식을 사용한다면 어느 정도의 속도 향상을 얻을 수 있다고 할 수 있다.

4. 결 론

본 논문에서는 3 차원 가상 공간을 탐험하는 것에 있어서 효율적인 충돌 탐지의 방법을 제안하였다. 효과적인 충돌 탐지를 위하여 넓은 가상 공간을 BSP-tree 를 사용하여 분할하였다. 이러한 처리의 장점은 모든 폴리곤에 대해서 충돌 탐지 검출을 하는 것이 아닌 특정 지역에 있는 필요한 개체에 대해서만 충돌처리를 하는 이점이 있다. 이렇게 나누어진 공간은 경계 기둥을 이용하여 충돌 개체와의 충돌 검출을 하도록 하였다.

본 논문에서 사용된 처리 형태는 첫 번째 단계로 넓은 가상 공간을 빠르게 처리하기 위하여 개체를 중심으로 하위 공간으로 나누었다. 두 번째 단계로 나누어진 공간을 BSP-tree 로 구축하였다. 셋째로 하위 공간 내에서 충돌 탐지를 위하여 충돌 탐지 대상을 경계 기둥으로 설정하였다. 경계기둥은 실제 개체간 충돌 탐지 검출을 위한 계산이 이루어지도록 하였다.

효율성은 복잡한 3 차원의 공간 계산을 비교적 계산이 간단한 2 차원으로 계산이 가능하도록 하였다. 또한 공간을 분할하고 경계 기둥을 사용함으로써 공간 속에 3 차원의 개체가 n 개이고, 각각의 개체를 이루는 면의 개수를 평균 m 일 때 시간 복잡도를 O(n*log*n)으로 줄일 수 있었다. 이 방법은 3 차원 가상 공간에서 이루어지는 MMORPG 또는 FPS 게임에서 사용한다면 속도 향상의 이득을 얻을 수 있다.

Reference

[1] Donald Hearn, M. Pauline Baker, “Computer Graphics,” 2nd edition, Prentice-Hall International, pp. 2-34, September 1996.

[2] Dong-Jin Kim, Guibas L. J, Sung-Yong Shin, “Fast Collision Detection Among Multiple Spheres,” Computer Animation '97, pp. 1-7, June 1997.

[3] Huerta J, Chover M, Quiros R, Vivo R, Ribelles J, “Binary Space Partitioning Trees: A Multiresolution Approach,” Information Visualization, 1997. Proceedings., 1997 IEEE Conference on 27-29, pp. 148-154, August 1997.

[4] Jung D, Gupta K. K, “Octree-Based Hierarchical distance maps for collision detection,” Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on Volume 1, pp. 454 – 459, April 1996.

[5] Klosowski J. T, Held M, Mitchell J. S. B, Sowizral H, Zikan K, “Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPs,” Visualization and Computer Graphics, IEEE Transactions on Volume 4, Issue 1, pp. 21-36, Jan.-March 1998.

[6] Sandeep Singhal, Michael Zyda, “Networked Virtual Environment,” Addison-Wesley Professional, pp. 147-180, September 1999.

[7] Tomas Akenine-moller, Eric Haines, “Real-Time Rendering,” 2nd edition, AK Peters, pp. 631-667, July 2002.