3D게임에서 충돌검출 모델의 효율성 분석

강윤미*, 박용범**
단국대학교 전자계산학과
e-mail:ymkang@dku.edu

Efficiency Analysis of Collision Detection Model In 3D-Game

Yun-Mi Kang*, Young B. Park**
Dept of Computer Science, Dan-Kook University

요 약

좋은 3D 엔진이란 객체들의 상호 작용을 실세계와 유사하게 표현하는 물리학 엔진을 말한다. 충돌은 이런 상호작용 중의 하나이며, 충돌 유무 검사와, 충돌 지점, 충돌후의 반응을 다룬다. 대부분의 물리학 엔진과 같이 충돌 검사도 정확하게 검출하려면 많은 시간이 소요된다. 그래서 개발자들은 정밀도와 시간을 절충할 수 있는 기법을 사용한다. 이렇게 사용되는 기법들이 얼마만큼의 효율성을 가지는지 검증하는 방법은 많이 제시되지 않았다. 본 논문에서는 효율성의 기준을 연산 시간과, 정확도로 책정했으며, 실제 개발에 적용되고 있는 알고리즘을 게임에서 발생하는 표본 상황에 적용하여 테스트 한 결과를 분석하고, 게임에 응용될 수 있는 모델을 제시한다.

1. 서론

3D 게임의 환경을 만드는 데 있어서 실시간적인 물리학 엔진은 게임 세계의 객체들 사이의 실제감 있는 상호 작용을 제공한다. 이러한 물리학 엔진은 게이머에게 좀 더 큰 현실감을 제공하며, 게이머는 더욱더 흥미로운 게임을 즐길 수 있다.[1] 하지만, 실제감 있는 엔진은 상당히 많은 자원을 요구하기 때문에, 아무리 성능이 좋은 컴퓨터 일지라도 실제 게임에 적용하기는 어렵다. 이러한 문제점 때문에 개발자들은 한정된 자원에서 최적을 속도를 낼 수 있는 엔진을 개발하기 위해서 노력한다.[4][9]

충돌 검출은 실시간 물리학 엔진의 중요한 부분들 중 하나이다[1]. 3D 게임에서 충돌 검출은 매 프레임 마다 수행되며, 충돌의 유무를 검사하고, 충돌한 지점, 충돌 후의 반응을 다룬다.[5] 본 논문에서는 충돌의 유무를 검사하는 부분만 다루고자 한다.

현재 사용되는 충돌 검출법은 vertex와 vertex, vertex와 edge, vertex와 face, face와 edge, sphere와 edge, sphere와 face, box와 edge, box와 face, box와 box, sphere와 sphere, 등 이 있다.[3][7][10] 결정적으로 어느 것이 좋다고 말할 수 없는 이유는 게임 종류나, 각 상황마다 그 효율성이 다르기 때문이다.[9] 본 논문에서는 기존에 사용되고 있는 충돌 검출법을 소개하고, 세밀한 충돌 검출을 할수록 소요되는 시간의 증가분을 분석한다. 그리고 게임의 종류나 상황에 맞는 모델을 제시하려 한다.

2. 충돌 검출법과 특징

대부분의 충돌 검출 모델의 특징을 살펴보면 속도와 정밀도는 반비례 관계에 있으며, '게임과 객체의 특징에 따라 적용되는 모델이 틀리다는 사실을 알 수 있다. 따라서 개발자는 시간과 속도를 고려하여 각 상황에 적합한 충돌 검출법을 선택 또는 개발해야 한다. [9]

일반적으로 충돌 검출 계산은 게임이 2D이냐 3D이냐에 따라서 두 종류로 나뉘고, 움직이는 객체와 환경을 구성하는 정적인 기하구조 사이의 충돌 검출인지, 아니면 움직이는 객체들 사이의 충돌 검출인지에 따라 2종류로 나뉜다.[4]

특히 3D에서는 얼마만큼의 정확한 충돌 검출이

필요하냐에 따라 적절한 방법이 고안되어야 한다. 예를 들어 두 캐릭터가 칼싸움을 한다면, 칼이 충돌된 지점에 상처가 나게 해야 하므로 정확한 충돌 검사가 필요하다. 반면 미사일을 떨어뜨린 지점의 폭파장면과 같은 정확한 충돌 지점 보다는 충돌의 유무만 파악하고 폭파장면을 연출할 수 있다.

다음으로 3D 충돌 검출에 사용되는 경계 볼륨과 두 볼륨의 충돌 검출법을 소개한다.

2.1 경계 볼륨

3D에서 객체를 표현하는 것은 폴리곤이며, 폴리곤을 구성하는 것은 하나하나의 픽셀이다. 완벽한 충돌 검출을 하려면 각각의 폴리곤, 또는 폴리곤을 구성하는 픽셀들의 위치를 계산해야 하지만, 한정된 컴퓨터 자원 때문에 그 많은 시간을 소비하기는 어렵다. 그래서 각 객체를 감싸는 최대한 작은 볼륨을 만드는데, 객체를 구성하는 볼륨이 구 형태 이면 경계구(BS : Bounding Sphere)라 하고, 육면체 모양이면, 경계상자 또는 OBB(Oriented Bounding Box), AABB(Axis-Aligned Bounding Box) 라고 한다.[2]

BS는 아케이드 스타일의 우주 전투기 게임에 적합하며, 충돌 검출 방법이 간단다고 구현도 쉽기 때문에 2D나 3D에서 많이 사용된다. AABB는 game world의 축과 평행한 정육면체 형태이기 때문에 볼륨을 형성하는 시간을 절약한다. OBB는 game world축에 정렬되지 않은 육면체 형태이며 볼륨을 형성하는데 AABB 보다 많은 연산을 필요로 한다. 하지만, 객체에 가장 잘 맞는 경계상자를 만들 수 있고 충돌 검출 결과가 AABB보다 정확하다.[6]

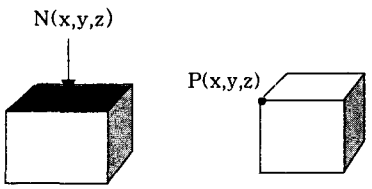
세밀한 충돌 검출을 위하여, 하나의 객체를 여러 부분으로 분할하여 각 부분을 하나의 볼륨으로 구성하는 방법이 있다[3]. 사람 캐릭터를 예로 들면, 서 있는 사람이라면, 전체를 직육면체로 표현 할 수 있다. 하지만 팔 다리가 움직이고, 정확한 충돌 검출이 필요한 경우라면 사람 객체를 머리, 몸통, 팔, 다리로 분할한다.[3][8] 이때 머리는 BS 또는 AABB, 몸통은 AABB, 팔과 다리는 OBB로 표현한다. 좀더 세밀한 충돌 검출을 원한다면 팔을 윗부분과 아랫부분으로 다시 나눌 수 있을 것이다. 이렇게 분할한 것을 하나의 트리로 구성하면 여러 개의 볼륨이 사람이라는 객체를 구성한다.

충돌 검출 시간을 줄이기 위한 전략중 하나로 계통적으로 범위를 좁혀나가면서 충돌을 검출하는 방법이 있다.[1] 움직이는 사람일 경우 사람객체의 BS를 먼저 만들어 충돌 검사를 한다. 충돌이면 좀더 세밀한 검사를 위해서 볼륨을 OBB 또는 AABB 로 하고 계산한다. 이 방법은 모든 객체에 정밀한 충돌 검사를 하는 것이 아니라, 세밀한 검사를 할 필요가 있는 객체를 선별하는 경우에 적합하다.

2.2 두 경계 상자의 충돌

OBB는 객체의 중심으로 좌표계를 생성하므로, 모든 OBB들은 고유의 좌표계를 갖는다. 따라서 두 OBB의 충돌을 검사 하기위해서는 서로 다른 두개의 좌표계를 동일하게 변환해야 한다.[5] 경계 상자의 충돌 검사는 위와 같은 좌표계 변환 작업을 거친 후에 이루어진다.

첫 번째로 가장 많이 사용되는 방법은 <그림 1>의 OBB B 의 정점 P 가 OBB A의 내부에 있는지를 검사하는 방법이다. 이 방법은 face와 vertex의 교차 판정 이론에 근거한다.[10][11][12]



<그림 1. OBB A 와 OBB B >

이 방법의 OBB B의 정점 P가 OBB A의 한 평면의 어느 곳에 위치하는지를 검사한다. N은 평면의 방향을 나타내는 것으로 평면의 외부와 내부를 구분하는 벡터이다. 평면의 방정식을 잠깐 살펴보면,
평면의 방정식이 다음과 같을 때,

$Ax+Bx+Cx+D=0$ ..... < 수식 1>

<수식 1>을 <그림 1>의 방법에 적용시킨다.

평면의 Nomal을 $N(x,y,z)$ 라 하고,

평면의 한 점을 $P(x,y,z)$ 라고 정의한다.

$N(x,y,z)$ 와 $P(x,y,z)$ 를 평면의 방정식에 적용하면, <수식2>를 얻을 수 있다.

$Nx*Px+Ny*Py+Nz*Pz-D=0$ .....<수식2>

<그림1>모델의 정점의 위치는 <수식2>로 판별이 가능하다.

$Nx*Px+Ny*Py+Nz*Pz>D$: 평면의 바깥쪽

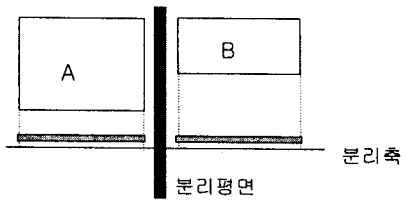
$Nx*Px+Ny*Py+Nz*Pz<D$: 평면의 안쪽

$Nx*Px+Ny*Py+Nz*Pz=D$: 평면에 포함

<그림 1>의 A에서 6개의 평면의 방정식을 구하고, 6개의 face normal을 구한다. B의 정점 P가 A의

6개 평면의 내부에 위치한다며, A와 B는 충돌한 것으로 판단한다. 여기서 말하는 평면은 평면의 방정식으로 표현되는 무한한 평면을 의미하기 때문에 정점이 6개의 모든 평면의 내부에 위치할 경우에만 충돌이다.

이 방법은 face와 vertex의 교차여부를 판정하므로 edge와 edge, face와 edge의 충돌 여부를 판단하는 것은 어렵다. 예를 들어 긴 칼로 나무를 베는 상황인 경우, face와 vertex의 검사로는 정확한 충돌 검출이 어렵다. 이 결점을 보안한 것으로 David Everly 가 제안한 'Intersection of Convex Objects:The Method of Separation Axes'[6][8] 기법이 있다. 분리축이라는 것은 <그림 2>과 같이 두 객체를 하나의 축에 투영했을 때, 두 객체를 구분하는 평면이 존재한다면, 그 축을 분리 축 이라고 한다.



< 그림 2. Separation Axes >

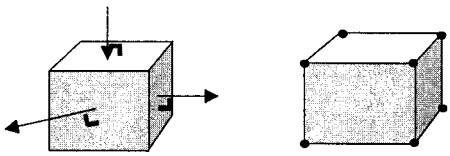
이 방법은 두 OBB간에 계산해야 할 분리축이 15개 이며, 분리축이 15개가 존재 할 때만, 충돌로 판정한다. 상당히 높은 정밀도를 가진 방법이지만, 구현이 복잡한 단점이 있다.[9]

3. 실험 과정

본 논문에서는 가장 많이 사용되는 두 OBB의 충돌 검사를 모델로 했다. <그림3> 은 <그림1>와 같은 경우로 A의 Normal이 외부를 향하게 하고 B의 각 정점의 위치를 검사하는 것이다. 가장 기본적인 방법으로 이것을 Regular Detection 이라 한다.

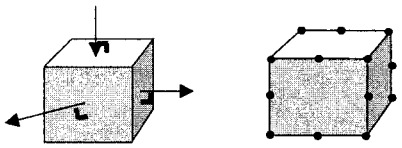
B의 한 점과 A의 모든 평면에 대해서, (평면의 방정식)<0 이라면, B의 한 점은 A의 내부에 존재하는 것이고, 충돌이라고 판정한다. OBB B의 8개의 정점들을 모두 검사 했을 때, 충돌이 아니라고 판정되면, A의 정점과 B의 평면을 검사한다. 한번에 검사해야 하는 정점의 수는 총 8*2로 총16개가 되며, 최악의 경우 6*8*2개의 정점을 검사한다.

이 경우는 face와 vertex 간의 충돌을 검사하는 것이므로 face와 edge, edge와 edge의 충돌 판정에는 미흡한 단점이 있다.



<그림 3. A와 B의 Regular Detection>

이를 보안하기 위해서 각 edge에 정점을 추가하여 정확도 높은 모델을 단계별로 구현했다. Regular Detection 모델에서 각 edge에 정점을 하나 추가한 것을Test1 Detection(T1), 2개 추가한 것을 Test2 Detection(T2)이라고 하면 이런 모델을 Test7 Detection(T7)까지 만들어서 실험했다. 그리고 가장 중요한 것은 detection의 정확성을 알기 위해 Full Detection(F)을 추가하는 것이다. F는 연산시간을 고려하지 않은 100%에 가까운 detection이 가능한 모델이다.



<그림 4 Test1 Detection>

첫 번째 실험에서는 collision space(충돌 검사 영역)를 OBBSIZE *5로 설정하고, OBB 개수를 10개부터 45개까지 늘리면서 R, T1, T2, ... T7, F을 실행했다. 각 레벨의 결과는 연산 시간, 충돌횟수, 오차를 출력한다. 다음은 실험 과정이다.

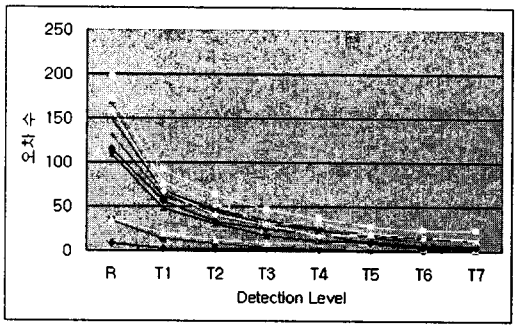
① 100%에 가까운 충돌 검출 함수 구현
② R, T1, T2, T3... T7모델 구현
③ collision space와 OBBSIZE, OBB개수를 고정시키고, R, T1, T2, ... T7, F를 실행
④ F를 제외한 각 모델의 연산 시간, 충돌 횟수, 오차 횟수를 측정한다.
⑤ 각 모델에 따른 연산시간의 증가와 오차의 감소를 비교 분석한다.

두 번째 실험은 OBB 크기가 오차율과 연산 시간에 영향을 주는 정도를 알아본다. collision space과 OBB개수를 고정시키고 OBBSIZE를 10에서 40으로 변화를 주면서 실험했다.

4. 실험 결과

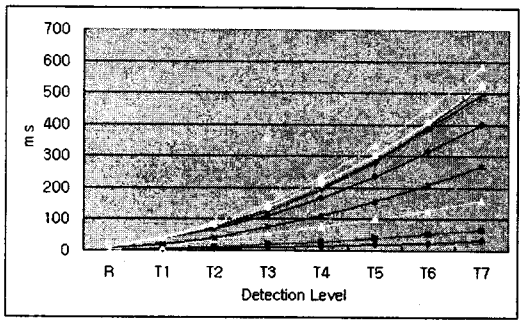
<그림 5>는 첫 번째 실험의 결과로 객체 수에 상관없이 레벨이 높일수록 누승 형태로 오차율이 적어진다.

하지만 객체 수가 적을수록 오차변화율이 적다. 이것은 collision space에 객체수가 적다면 T1레벨정도로도 충분한 detection이 가능함을 보여준다.



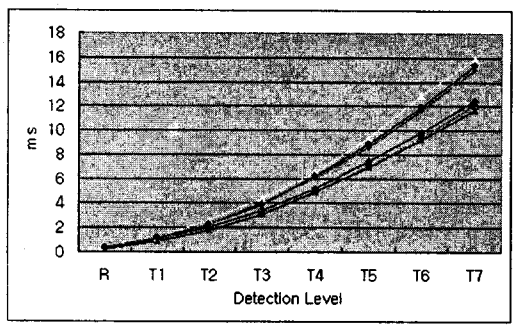
<그림 5 Level에 따른 오차율의 변화>

<그림 6>은 레벨에 따른 연산시간의 변화를 보여준 것이다.



<그림 6.Level에 따른 시간의 변화>

레벨을 높일수록 연산시간이 누승으로 증가한다. 역시 객체 수가 적을 시에는 시간증가율이 적었다.



<그림 7. 객체 크기에 따른 시간의 변화>

<그림7>은 객체 수를 고정시키고 OBBSIZE와 연산시간의 관계를 알아 본 것이다. 연산시간은 OBBSIZE와 관계가 없었으며, 충돌횟수가 많을 때 연산시간이 더 오래 걸리는 것을 알 수 있었다.

모든 실험을 종합해보면, 연산 시간은 객체 크기와는 상관없다. 충돌이 많이 발생하는 상황일수록 연산시간은 지연된다. 또한 객체 수가 적을수록 오차율도 적었다. 모든 증가곡선은 누승 형태임을 고려하면, 게임 상황에 적합한 레벨을 제시할 수 있다.

위 실험 결과를 게임에 응용하기 위해서는 collision space에 몇 개의 객체가 있는지를 알아야 한다. 예를 들어, 전투 아케이드 게임의 경우 아무리 객체가 많더라도, 객체의 크기가 작으므로 T1레벨에서도 정확도 높은 detection이 가능하다. 밀리터리 슈팅 게임의 경우 collision space의 객체 수가 적으며, 상당히 높은 정밀도가 필요하므로 T6, T7레벨이 적합하다.

5. 결 론

본 논문에서는 충돌 검출의 효율성을 속도와 정확도를 기준으로 측정했으며, 많은 충돌 검출 모델 중에서 개발하는 게임에 적합한 모델을 선택하기 위한 실험의 필요성을 보였다.

현재는 실험결과를 분석하여 효율성을 수치화하는 일과 충돌 검출 방법에 영향을 줄 수 있는 다른 요인들에 대한 연구가 진행 중이다.

참 고 문 헌

[1] game programming Gems / Mark Deloura 외 공저, 류광 역
[2] Dynamic Collision Detection Oriented Bounding Boxes / David Elberly
[3] Advanced Collision Detection Techniques / Nick Bobic
[4] 3D 게임 프로그래밍& 컴퓨터 그래픽을 위한 수학 / Eric Lengyel 저, 류광 역
[5] 생생한 게임 개발에 필요한 기본 물리 / Daved burg 저, 황혁기 역
[6] Intersection of Convex Object : The Method of Separating Axes / Daved Eberly
[7] Collision Detection / 2002. 민프레스 세미나(김병기)
[8] Fast Overlap Test for OBB / 이만희
[9] http://www.gpgstudy.com
[10] http://www.magicsoftware.com
[11] http://www.netian.com/~proline1
[12] http://www.wzsoft.com/~leechen