한국게임학회 논문집
제1권 제1호 2001년 11월

컴퓨터게임을 위한 2D 충돌 감지 알고리즘 비교 분석에 관한 연구

이영재*
*yjlee@hcc.ac.kr

A Comparative study On 2D Collision Detection Algorithms For Computer Games

Young Jae Lee*

Abstract

Collision is a brief dynamic event consisting of the close approach of two or more objects or particles resulting in an abrupt change of momentum or exchange of energy because of interaction. Collisions play very important role in computer graphics, computer games and animations fields. Collisions can supply active interaction between cyberspace and real world and give much interests for making nice games so reasonable collision detection algorithms are needed. Collision detection algorithms should satisfy being fast and accuracy. In this paper, we survey the 2D collision detection algorithms between geometric models. We present several methods and system available for collision detection.

Key Words *: Collision , collision detection, interaction, object*

1. 서 론

충돌은 상대적으로 운동하는 물체 또는 입자가 근접 또는 접촉해서 상호작용을 미치는 현상이다. 이같은 충돌현상에는 물체의 모양, 크기, 운동량, 물성 등등 여러 가지 충돌에 영향을 미치는 물리학적인 파라메터가 존재하며[1] 점과 점의 충돌로부터 2차원 충돌, 3차원 충돌 등 차원에 따라 구분할 수도 있다.

이같은 충돌현상을 미리 예상하고 그에 대한 반응을 생각하기 위한 충돌감지 기능은 컴퓨터 그래픽, 애니메이션, 컴퓨터 게임 등 다양한 분야에 꼭 필요한 필수적인 기능이다.[1-4]

컴퓨터 게임에서의 충돌은 실세계에서 일어나는 일들을 그대로 시뮬레이션하여 보여주는 다이내믹 시뮬레이션 시스템(Dynamic simulation system)과 같은 정밀성을 요하는 것은 아니지만 정확하고 빠른 감지가 가능한 특성을 만족해야 한다.

컴퓨터 게임 분야에서 오브젝트간의 다양한 충돌 이벤트는 필수적인 것이다. 왜냐하면 게임에서 발생하는 모든 이벤트가 바로 오브젝트의 충돌에서 시작되는 것이기 때문이다. 액션게임의 경우 주인공이 적에게 맞아 데미지를 입거나 슈팅게임의 전투기가 미사일에 맞아 폭발하는 등의 처리는 충돌처리 없이는 불가능한 것이다.[9] 따라서 게임을 구성하는 다양한 오브젝트(주인공, 적, 총알, 장애물 등)들이 주어진 조건에 따라 충돌했을 경우, 이를 감지하고 필요한 처리를 해주어야 하며 이에 따른 적절한 충돌감지 알고리즘의 개발이 필요하다.[1-10] 따라서 오브젝트의 전체형상을 하나의 단위로 감지하는 원, 사각형 사용방법이 주종을 이루었으나, 최근에는 프로세서의 성능향상에 기인하여 각종 다양한 방법의 개발과 더불어 애니메이션에서 사용하는 여러 기법 등도 적용해 사용하고 있다.

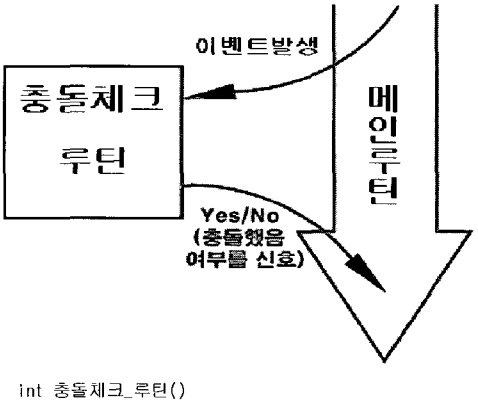
2장에서는 게임에서 발생할 수 있는 충돌의 종류에 대해 알아보고, 3장 및 4장에서는 지금까지 개발된 충돌 알고리즘과 이들의 장, 단점을 비교해 보고, 5장에서 결론을 맺는다.

2. 컴퓨터 게임에서의 충돌

기하학적인 모델들의 상호 작용에 발생할 수 있는 충돌을 빠르고 정확하게 감지하는 방법은 컴퓨터 그래픽과 게임 분야 뿐 아니라 모델링, 로봇공학 등의 분야에서도 가장 중요한 이슈 중 하나이며, 이에 대한 연구도 활발하게 진행중이다[1-7][10]

컴퓨터 게임의 경우 충돌은 가상과 현실과의 상호작용을 제공하고 A.I(인공지능)와 더불어 게임을 재미있게 만드는 중요한 요소이다. 따라서 충돌에 대한 종류를 규정짓는 것이 필요하다. 예를 들어 슈팅(Shooting)게임에서의 충돌 이벤트는 다음과 같이 분류할 수 있다.[8]

① 적 무기에서 아군 캐릭터와의 충돌 여부를 감지
② 아군 캐릭터에서 적 무기와의 충돌 여부를 감지
③ 아군 무기에서 적 캐릭터와의 충돌 여부를 감지
④ 적 캐릭터에서 아군 무기와의 충돌 여부를 감지
⑤ 아군 캐릭터에서 적 캐릭터와의 충돌 여부를 감지
⑥ 적 캐릭터에서 아군 캐릭터와의 충돌 여부를 감지 등과 같이 아 군 캐릭터와 적군 캐릭터 사이의 충돌, 적의 무기, 총알, 획득해야 할 아이템과의 충돌, 장애물과의 충돌까지 충돌체크 루틴이 필요하다. 이를 루틴으로 표현해 보면 그림1과 같다.[8]




```
int 충돌체크_루틴()
    {
            int collision = 0;
            충돌 감지 알고리즘 수행;
            if(충돌감지) collision = 1;
            return collision;
    }
```

그림 1. 충돌 감지 루틴
Fig. 1. Collision detection routine

3. 충돌 감지 알고리즘

3.1 원을 사용한 방법

원을 사용하는 방법에서는 이들의 반지름을 사용한다. 즉 이들 두 원의 반지름 합을 상수로 정해 이 상수보다 같거나 작은 경우 충돌로 감지하는 방법이다.

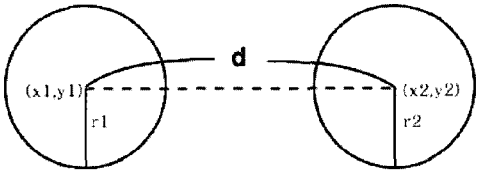


그림 2. 원을 사용한 충돌 감지
Fig. 2. Collision detection using circle

두 원 사이의 거리 d는 식(1)과 같이 나타낼 수 있다.

$$d = \sqrt{(x_2-x_1)^2+(y_2-y_1)^2} \qquad (1)$$

d : 두 원간의 거리

이 경우 $d \le r_1+r_2$ 이라면 두 원의 충돌이 발생한 경우이다. 원 둘레에 대한 좌표값은 $x_n^2 + y_n^2 = r_n^2$ 를 사용해 구할 수 있다.

원 사용방법은 접촉, 충돌 감지를 위한 계산이 간단한 장점을 가지고 있으나 원 형태가 아닌 경우 예를 들면 직사각형의 형태를 가지고 있는 경우엔 적합하지 않으므로 제한적으로 사용되고 있다.

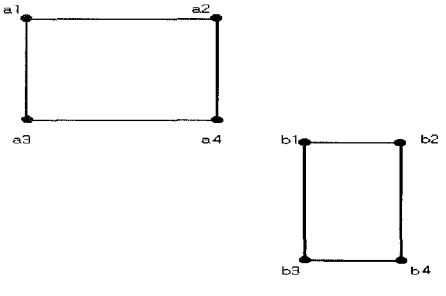


그림 3. 사각형을 사용한 충돌 감지
Fig. 3. Collision detection using rectangular

3.2 사각형을 사용한 방법

사각형은 4개의 특징점 좌표를 가지고 있으며 이들은 동일한 x 좌표, y 좌표를 가지고 있으므로 이들의 좌표값을 이용하여 충돌여부를 확인할 수 있다.

예를 들면 a4의 좌표를 (x4,y4)라 하고 b1의 좌표를 (x1',y1')라 하면 (x1')x4 || y1') y4) 경우와, a1의 좌표를 (x1,y1)라 하고 b4의 좌표를 (x4',y4')라 할 때 (x1)x4' || y1) y4') 경우 중 하나만 만족하면 충돌이 발생하지 않은 경우이다. 이 방법은 충돌체크가 간단한 장점은 있으나 오브젝트의 형상(예: 삼각형)에 따라 정밀한 감지가 어렵기 때문에 환경에 따라 제한적으로 사용할 수 있다.

3.3 픽셀 데이터를 이용한 방법

3.3.1 다각형 형태 분석 방법

삼각형, 사각형 등 다양한 형태의 다각형을 하나의 간단한 수식으로 표현할 수 있다면 좌표값에 따른 값의 확인을 통하여 충돌여부를 확인할 수 있다. 예를 들어 삼각형의 세 꼭지점을 A, B, C 라하면 삼각형의 변에 대한 식을 다음과 같이 구할 수 있다.

$$AB(t) = (1-t)A + tB \quad (t\text{는 0부터 1까지 변함}) \qquad (2)$$
$$BC(t) = (1-t)B + tC \quad (\text{“}\,)$$
$$CA(t) = (1-t)C + tA \quad (\text{“}\,)$$

이들의 식을 구간 별로 조정해 보면

$$AB(t) = (1-t)A + tB \quad (t\text{는 0부터 1까지 변함}) \qquad (3)$$
$$BC(t) = (2-t)B + (t-1)C \quad (t\text{는 1부터 2까지 변함})$$
$$CA(t) = (3-t)C + (t-2)A \quad (t\text{는 2부터 3까지 변함})$$

그런 다음 t 가 0부터 3까지 변할 때 식(4)와 같이 Max 함수를 사용해 이들을 하나의 식으로 나타낼 수 있다.

$$ABC(t) := Max(-0.5 + |t-1.5|,0)A + Max(1-|t-1|,0)B + Max(1-|t-2|,0)C \qquad (4)$$

식 (4)를 이용해 사각형, 오각형으로 확장하여 이들을 표현할 수 있으며, 이를 원래의 좌표식에 대한 비와, 위치에 대한 좌표를 사용해 x 좌표에 해당하는 y 좌표를 구할 수 있다. 그러나 이 방법은 오브젝트의 크기와 수에 따라 계산량이 많아지는 단점이 있다.

3.3.2 세그멘테이션 데이터 사용 방법

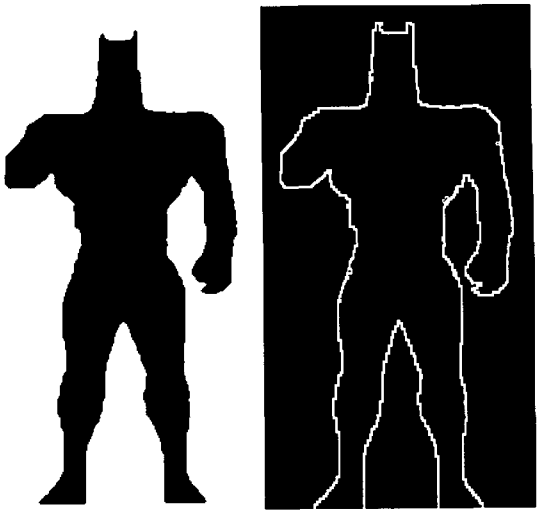


그림 4. 스프라이트 예

Fig. 4. An example of sprite

그림 4의 그림 (a)는 스프라이트의 한 예이다. 이 스프라이트를 외곽추출(Segmentation)을 하면 그림 (b)와 같은 데이터를 얻을 수 있다. 이 데이터를 충돌맵으로이용하면 정확한 충돌 여부를 판별할 수 있다. 그러나 이 경우에는 많은 계산량과 데이터가 필요하므로 빠른 반응을 요하는 게임에서 적용하기엔 어려움이 있다.

이 같은 단점을 극복하기 위하여 오브젝트 영역을 N으로 대분하여 부위 별 충돌의 유무를 결정하고, 정밀 충돌을 요하는 부분에 한해 부분 충돌맵 데이터를 이용방법이다. 이 때 부위를 몸통부위, 방어부위, 필살부위 ...등으로[9] 구성하면 더 효과적일 수 있다. 그러나 이를 위해서는 별도의 충돌맵 파일 작성이 필요하다.

3.3.3 부분 분할 데이터를 이용한 방법[6]

부분 분할 데이터(bounding data)를 이용한 방법은 오브젝트의 트리나 특징점을 기준으로 원, 사각형을 이용해 오브젝트를 바운딩해 위치를 설정하고 이들 좌표를 데이터베이스로 사용해 충돌을 감지하는 방법이다.

그림 5의 (a)(b)는 특징점 혹은 트리를 구성해 부분 형상으로 나누고 이를 원이나 박스를 사용하여 오브젝트 의 부분형상으로 나누어 데이터 베이스로 활용하는 방법이다. 그림(a)에서는 c1,c2,c3 등 3개의 원을 사용하는 경우이며, 그림 (b)의 경우 트리를 사용해 박스(r1,r2) 2개를 사용한 경우이다. 이 방법은 원과 박스를 오브젝트의 형상에 따라 세밀하게 나눔에 따라 정밀한 충돌 감지가 가능하나 충돌을

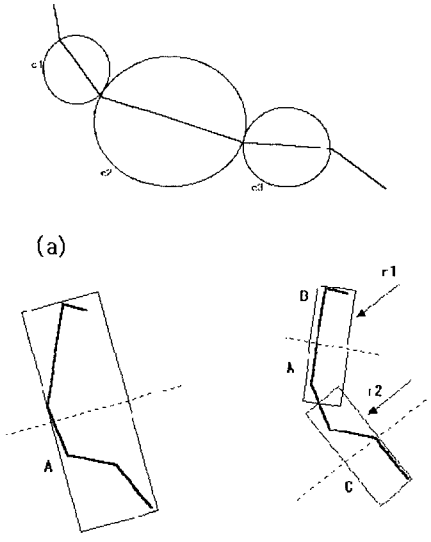


그림 5. 부분 분할 데이터를 이용하는 방법
Fig. 5. Collision detection using partial shape data

위한 비교 데이터 량이 많아지는 단점이 있다.

3.3.4 Cyrus-Beck clipping 알고리즘[3-4]

Cyrus-Beck clipping 알고리즘은 convex rigid body에 대한 충돌감지 알고리즘이다.

이 알고리즘에서는 충돌하는 경우를 다음 그림과 같이 두 가지 경우로 대분한다. 즉 한 물체의 꼭지점(vertex)이 다른 물체의 면(polygon)에 충돌하는 경우와 또 다른 경우로는 물체의 변(edge)이 다른 물체의 면 경계에(polygon boundary) 충돌하는 경우이다. 즉 한 물체의 면경계가 다른 물체의 면경계와 충돌하는 경우이다. 그림 6에서 보는 것과 같이 굵은 선으로 표시된 무한직선에 대해 가는 선으로 표시된 무한 직선을 자른다고 생각해 보자. 즉, 굵은 선으로 표시된 무한직선의 양쪽을 그림 6과 같이 in, out으로 가정했을 때 가는 선의 in 쪽에 있는 부분만을 나타내고자 한다. 이때 이 굵은 선의 외부 수직벡터(outward normal vector)를 그림과 같이 $\vec{n}$(vector n) 으로 표시하고 굵은선 위의 임의의 한점을 a, 자르고자(clipping)하는 선사이에 존재하는 임의의 점을 각각 b(굵은선 밖에 존재), c(굵은선 안에 존재)라 할 때 벡터 $\vec{n}$ 과 $\overrightarrow{v_{ba}}(\overrightarrow{v_{bo}}-\overrightarrow{v_{ao}})$, $\overrightarrow{v_{ca}}(\overrightarrow{v_{co}}-\overrightarrow{v_{ao}})$,(0는기준점) 사이에서 그림 6과 같은 관계가 성립한다. 즉, 굵은선 밖에 있는 임의의 점 b와 굵은선 상의 임의의 점 a 사이의 벡터 $\vec{v}$와 $\vec{n}$ 의 내적은 항상 0보다 크다.

$$n \cdot V_{ba} > 0 \qquad n \cdot V_{ca} < 0$$

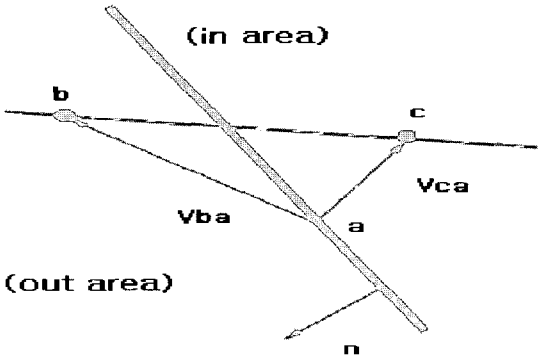


그림 6. Cyrus-Beck clipping 알고리즘의 기본 원리
Fig. 6. The basic principle of Cyrus-Beck clipping algorithm

두 번째의 경우인 무한 평면과 직선의 교차점 계산인 경우 무한 평면의 수학적 표현은 여러 방법으로 표시될 수 있지만 벡터식으로 표시해보면 식(5)와 같다.

$$\vec{n} \cdot \vec{q} = 0 \qquad (5)$$

여기서 $\vec{n}$ 은 무한 평면의 수직 벡터이고 $\vec{q}$ 는 무한 평면 위에 놓인 벡터이다. 여기에서 다시 $\vec{q}$는 $\vec{p} - \vec{p_1}$ 으로 표시할 수 있는데 $\vec{p_1}$은 무한 평면위 특징점의 기준점에 대한 위치 벡터(position vector)이고 p 는 무한 평면에 존재하는 임의의 한 점에 대한 위치 벡터 [x, y, z]이다. 그러므로 식 (5)를 다시 쓰면 평면식은 (6)으로 표시된다.

$$\vec{n} \cdot \vec{p} = \vec{n} \cdot \vec{p_1} \qquad (6)$$

직선의 parametric 표현은 직선의 시작점을 v_1[1] 끝점을 v_2[2] 라 할 때 식 (7)과 같이 표시된다.

$$\vec{\tau}(t) = \vec{v_1} + t \times (\vec{v_2} - \vec{v_1}) \qquad (7)$$

무한 평면과 직선의 교차점 계산은 (7)식을 (6)식에 대입하므로써 t값을 계산할 수 있다. 이때 t가 0과 1사이의 값이고 직선과 무한 평면은 $\vec{\tau}(t_{cross})$ 점에서 교차한다고 하면 이를 식으로 나타내면 식 (8)과 같다.

$$t_{cross} = \frac{\vec{n} \cdot (\vec{p_1} - \vec{v_1})}{\vec{n} \cdot (\vec{v_2} - \vec{v_1})} \qquad (8)$$

이 알고리즘은 정밀한 충돌에 대한 정보를 얻을 수 있으므로 게임 뿐 아니라 애니매이션과 같은 분야에서도 사용가능하다. 그러나 오목형 타입(concave)의 오브젝트인 경우 적용할 수 없는 단점이 있다.

4. 기타 방법[5]

4.1 방법 I

기준선 4개의 좌표를 기준으로 하여 4개의 영역(quadrant, 사분면)으로 구분하고 시작점의 좌표를 기준으로 하여 폴리곤의 처음 꼭지점으로부터 시작하고 카운터를 0으로부터 시작한다. 꼭지점에서 꼭지점을 잇는 변이 시계방향이면 카운터에 1을 증가시키고 반대의 경우엔 카운터에 1을 감소시킨다. 이 경우 만약 변 두개의 영역을 뛰어 넘어 반대 영역으로 넘어간다면 2를 증가시키거나 감소시킨다. 아래 예에서는 3-4에서 카운터가 1이 증가하고, 4-5에서 1이 감소한다. 5-6에서 1이 증가하고 8-9에서 1이 증가한다. 10-11에서 1이 증가하고 11-1에서 1이 증가한다. 이런 식으로 폴리곤의 모든 꼭지점을 계산한 카운터의 값이 4 또응 -4일 때 시작점은 폴리곤의 내부에 있는 것으로 충돌이 발생한 경우이다.

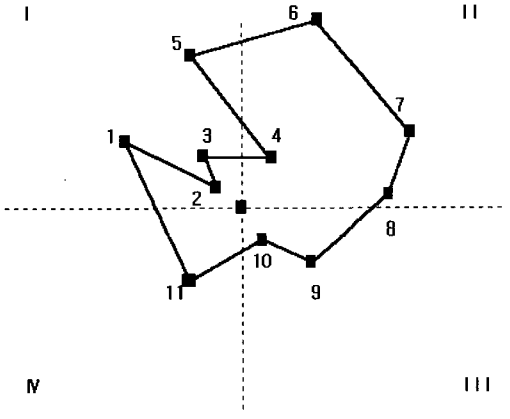


그림 7. 꼭지점 방향을 이용한 충돌 감지 방법
Fig. 7. Collision detection using vertex orientation

방법 I의 경우엔 비교적 효과적인 충돌 감지가 가능하지만 기본적으로 세그멘테이션 과정을 통해 특징점 및 윤곽선 데이터를 가지고 4분할하여 특징점들의 방향을 계산해야 하므로 오브젝트가 많은 경우 계산량이 많아질 수 있으며, 특징점이 불투명한 원과 같은 오브젝트인 경우엔 적용하기 어려운 단점이 있다.

4.2 방법 II

시작점으로부터 폴리곤의 외부에 있는 것으로 정의된 점까지 기준 직선을 그린다. 이 기준 직선이 폴리곤의 변과 몇 번 교차하는지 횟수를 센다. 이때 교차하는 횟수가 홀수이면 시작점이 폴리곤의 내부에 있어 충돌이 발생한 경우이며, 짝수일 경우엔 외부에 있는 것으로 충돌이 발생하지 않은 경우이다. 방법 I에 비해 간단하며 원인 경우에도 적용이 가능한 알고리즘이다. 그러나 기본적으로 특징점 및 윤곽선을 구해 이를 충돌 데이터로 사용해야 하고 접점의 유무를 계산해야 하므로 오브젝트가 많은 경우엔 계산량이 많은 단점이 있다.

4.3 방법 III

방법 III은 시작점과 꼭지점을 연결해 이루는 각의 합이 360인 경우 그 시작점을 다각형의 내부에 있는 점으로 판별하는 방법이다. 이 방법의 경우 꼭지점과 시작점과의 내적을 일일이 계산해야 하므로 효율적이지 못한 단점을 가지고 있다.

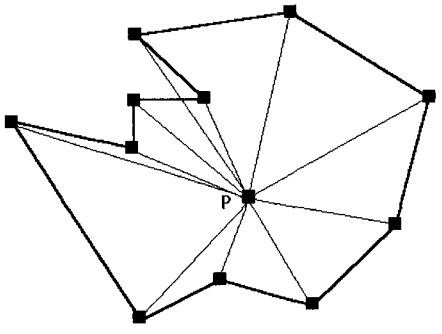


그림 8. 꼭지점과 시작점의 각을 이용한 충돌감지 방법
Fig. 8. Collision detection using vertex and start point

충돌 알고리즘	특 징	장 점	단 점	비 고
원 사용 방법	원반지름과 원사이 거리 이용	알고리즘 계산이 단순	대상의 형태를고려 해야함	
사각형 사용방법	4개의 특징점 사용	알고리즘 계산이 단순	대상의 형태를고려 해야함	2D게임에서많이사용
픽셀 사용방법	다각형 수식 사용	세밀한 감지	대상에따라 계산량 이많아짐	
세그멘테이션 방법	윤곽선 데이타 사용	세밀하고 정확한 감지	대상에 따라 계산량,데이터량 이많아짐	2D게임에서많이사용
부분분할 데이터 방법	특징점 기준으로 분할	세밀한 감지	대상에 따라 계산량 이많아짐	
Cyrus-Beck Algor-ithm	벡터의 내적값을 이용	세밀한 감지	대상에 따라 계산량이 많아짐	3D로확장적용가능
방법I	꼭지점 방향을 이용	세밀한 감지	대상에 따라 계산량이 많아짐	
방법II, III	폴리곤 접점,각도를 이용	세밀한 감지	대상에 따라 계산량 이많아짐	

표 1 충돌 감지 알고리즘 비교 분석

Table 1. A comparative analysis of collision detection algorithms

5. 결 론

지금까지 제안된 2D에서 사용 가능한 다양한 충돌 감지 기법에 대해 알아보았다. Cyrus-Beck clipping 알고리즘의 경우 볼록한 형태만 감지 가능한 제한은 있으나 선분(Edge)은 면(polygon)으로, 선분의 외부 수직 벡터(edge outward normal vector)는 면의 외부 수직벡터(polygon outward normal vector)로 하면 3D로 확장해서 사용하는 것이 가능한 장점을 가지고 있다.

기타방법I,II,III의 경우엔 계산방법이 간단하지만 특징점을 구하기 위한 전처리 과정이 필요하게 된다.

특히 복잡한 형태의 오브젝트인 경우 계산량이 많아 적용이 어려울 수 있다.

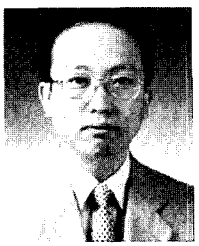
충돌맵을 사용하는 경우 정확한 충돌감지가 가능하지만 오브젝트에 대한 별도의 맵화일 작성이 필요하고, 비교 데이터량이 많은 단점이 있다. 이 같은 단점을 극복하기 위하여 오브젝트 영역을 N으로 대분하여 부위 별 충돌의 유무를 결정하고, 정밀 충돌을 요하는 부분에 한해 부분 충돌맵 데이터를 이용방법 즉, 대략적인 충돌 유무감지와 더불어 정밀한 충돌 감지 방법을 겸용해 사용하는 방법도 제안되어 사용되고 있다. 액션게임인 "데몰리션" 이 그 경우이다.[9] 그러나 아케이드 게임인 "개구리"나 롤프레잉 게임인 "다라시안"[9], "레이디안" 등의 경우는 사각형 방법을 사용해 충돌감지를 한다.

그러므로 어떤 충돌 기법을 사용할지는 게임의 종류와 오브젝트의 성격 등 게임 특성에 따라 장점을 고려한 알맞는 방법을 채택해야 하며 빠르고 정확한 감지가 가능한 알고리즘 이어야 한다.

참 고 문 헌

1. Ming C. Lin, Stefan Gottschalk, "Collision Detection: Between geometric models",Proceedings of IMA Conference On Mathematics of Surface , pp1-20, 1998
2. Ming C. Lin, Dinesh Manocha, Jonathan Cohen and Stefan Gottschalk (1996), "Collision Detection: Algorithms and Applications", in the book "Algorithms for Robotic Motion and Manipula tion",edited by J. P. Laumond and M. Overmars,A.K. Peters Publishing Company, pp. 129-141.
3. M.Moore and J. Williams, "Collision Detection and Response for Computer Animation", Computer Graphics, vol.22 .4, pp 289-298, 1988(Association for Computing Machinery. Proceeding of SIGGRAPH '88)
4. 김현준, 경종민," 3차원 컴퓨터 애니메이션을 위한 충돌 검색 및 반응 계산", 전자공학회논문지 제 30권 a편 제 3호, pp130-138, 1993
5. http://www.darwin3d.com/gdm1999.htm (January 1999:2D Collision Detection)
6. http://www.gamasutra.com/features/20000330/bobic_01.htm
7. Marc Saltzman, 박상호 역, Game Design, pp187-235, (도)민컴뮤니케이션,2001
8. 홍석기 외2, 게임 하듯이 게임 만들기,pp304-309,

pp416-420, 가남사, 1998
9. 김종찬, Windows 게임프로그래머를 위한 X 파일, (주)사이버출판사, 2000
10. Mark Deloura외,류광역, Game Programm ing Gems,pp502-526,정보문화사,2001



이영재

1984년 2월 충남대학교 공업교육대학 전자교육공학과 졸업(공 학사)
1986년 1월 ~1995년 12월 19일 LG전자 부품(주)연구소 (선임 연구원)
1994년 8월 연세 대학교 대학원 전자공학과 졸업(공학 석사)
2000년 8월 경희 대학교 대학원 전자공학과 졸업(공학 박사)
1996년 3월 ~ 2000년 8월 신성 대학 전자과 (조교수)
2000년 9월 ~ 현재 혜전대학 컴퓨터 통신계열 컴퓨터 게임전공(조교수)