論文93-30A-3-13

3차원 컴퓨터 애니메이션을 위한 충돌 검색 및 반응 계산

(Collision Detection and Response Calculation for 3-D Computer Animation)

金 賢 埈*, 慶 宗 晏**

(Hyun Joon Kim and Chong Min Kyung)

要 約

일반적인 애니메이션 시스템에서 많은 물체들이 움직일 때 물체들 간의 관통을 막기 위해서 충돌을 검색해 내는 기능이 필요하다. 또한 애니메이션 시스템의 일종인 다이나믹 시뮬레이션 시스템은 충돌 반응의 기능이 있어야 하는데 이는 충돌 발생후 동력학을 이용해 실세계의 동작을 시뮬레이션하여 보여 주는 방법이다. 본 논문에서는 불필요한 계산양을 줄이고 물체를 정렬시킴으로써 충돌 검색을 가속시키는 방법을 제안하고, 충돌 검색과 반응 계산을 수행하는 다이나믹 애니메이션 시스템을 구현하였다. 이 시스템에서는 사용자가 마찰계수, 탄성계수, 중력 여부, 물체 모양, 외부 힘, 토크등을 입력하면 동력학을 이용하여 실세계의 동작을 시뮬레이션하여 외곽선 형태로 보여 준다.

Abstract

A mechanism for collision detection in general animation system is necessary to prevent the interpenetration among multiple objects. On the other hand, a dynamic simulation system which is a part of animation system simulates realistic motions using dynamics after the collision, which is called collision response. In this paper, a method for reducing the CPU time for collision detection by removing redundant calculations and object sorting is proposed. A dynamic simulation system including collision detection and response function was implemented to demonstrate the proposed methods, where the input data as elasticity, friction, gravity, object shape, external force and external torque are given by the user. The system simulates motions of multiple objects using dynamics, and generates the wireframe display.

Ⅰ. 서 론

컴퓨터의 계산 능력 증대, 컴퓨터 그래픽 기능의 확대등에 힘입어 근래 들어서 컴퓨터 애니메이션에 대한 관심은 날로 커져 가고 있으며 그 쓰임도 TV, 광고, 영화등 날로 증가, 확대되어 가고 있는 상황이다. 일반적으로 컴퓨터 애니메이션에는 3가지 접근 방법이 있다.

첫째는 keyframe animation 방법이다. 이 방법은 기존의 만화영화 제작 과정과 유사한 것으로써 기존의 만화 영화 제작에서는 그림을 그리는 사람을 키프레임 처리인(keyframer)과 중간 분할 프레임 처리인(in-betweener)으로 구분하는데 키프레임 처리인은 이야기 전개중 중요한 부분만(keyframe)을 그리고, 키프레임 처리인보다 덜 숙련된 중간 분할 프레임 처리인은 이 키프레임을 기준으로 1초에 24장 이

*學生會員, **正會員 韓國科學技術院 電氣 및 電子工學科
(Dept. of Electrical Eng., KAIST)

接受日字:1991年 3月 25日

상씩 중간 분할 프레임을 그린다. keyframe animation 방법이란 이 중간 분할 프레임 처리인이 하는 역할을 컴퓨터로 대체하는 방법으로 사용자가 키프레임을 컴퓨터에 입력하면 컴퓨터가 보간법(interpolation)등의 방법으로 중간 분할 프레임을 만들어 내는 것이다.

둘째는 procedural animation 방법이다. 이 방법은 C 언어나 Prolog등의 프로그래밍 언어를 기초로 애니메이션을 위한 고급 언어를 만들고 이 언어를 가지고 프로그래밍 함으로써 애니메이션 하는 방법이다.

셋째는 dynamic simulation 방법이다. 위에서 설명한 두가지 방법에서는 실제에 가까운 이미지를 만들어 내기 위해서 능숙한 사용자를 필요로 하고, 또 능숙한 사용자라 하더라도 실세계에서 일어나는 일을 정확히 예측하고 애니메이션하기는 무척 어렵다는 단점이 있다. 따라서 이와같은 단점을 보완하기 위해서 컴퓨터로 하여금 동력학을 기반으로 실세계에서 일어나는 일을 시뮬레이션하여 보여 주는 것이 dynamic simulation 방법이다. 이 방법은 가장 실세계와 유사한 동작을 얻는데 반해 계산양이 많을 뿐만 아니라 사용자가 원하는 동작들을 정확히 얻을 수 없다. 왜냐하면 dynamic simulation 방법은 사용자가 힘을 물체에 가하고 거기에 따르는 동작을 시뮬레이션하는 방법이기 때문에 사용자가 물체에 얼마만큼의 힘을 주어야 원하는 동작을 얻는지를 정확히 모른다는 것이다. 그러나 실세계와 가장 유사한 동작을 얻는다는 장점 때문에 이 방법에 대한 많은 관심과 단점에 대한 보완책이 연구되고 있다.[7]

일반적인 애니메이션 시스템에서 여러 물체가 동시에 움직일 때 흔히 발생하는 문제는 한 물체가 다른 물체를 관통한다는 것이다. 이 문제를 해결하기 위해 사용자가 아예 충돌이 일어나지 않도록 하거나 충돌을 미리 예상하고 그에 대한 반응을 생각해서 작업을 하는 방법이 있을 수 있으나, 이런 방법은 충돌 예상이 쉽지 않고 사용자가 생각하지 못하는 충돌이 발생할 수 있으므로 충돌 검색(collision detection)기능은 애니메이션 시스템에서 필요한 기능이라 할 수 있으며 특히 실세계에서 일어나는 일을 그대로 시뮬레이션 하여 보여 주는 dynamic simulation system에서는 충돌 검색 기능이 필수적이다. 또한 dynamic simulation system에서는 충돌후 충돌에 따른 물체들의 움직임을 실세계와 유사하게 재현해야 한다. 그러므로 dynamic simulation system에서는 이를 위해 충돌 반응(collision response)에 대한 기능이 반드시 필요하며, 실세계와 유사한 동작을 시뮬레이션 하기 위해 역학 관계식을 이용한다.

본 논문에서는 애니메이션 시스템에서 유용한 충돌 검색 알고리듬과 그의 검색 시간을 줄이는 방법에 대해 제안하고, dynamic simulation system에서 필수적인 충돌 반응의 구현에 대해 설명한다.

Ⅱ. 충돌 검색 알고리듬

충돌 검색 알고리듬은 시스템이 지원할 수 있는 물체가 어떤 종류인가에 따라 변형되거나 달라질 수 있는데 본 장에서는 Moore[2] 등이 제안한 convex rigid body에 대한 충돌 검색 알고리듬에 대해 설명한다. Rigid body가 충돌하는 경우는 그림 1과 같이 두 가지 경우가 있다.[1] 즉 a)경우와 같이 한 물체의 꼭지점(vertex)이 다른 물체의 면(polygon)과 충돌하는 경우이고 또다른 경우는 b)와 같이 한 물체의 선분(edge)이 다른 물체의 면경계(polygon boundary)에 충돌하는 경우이다.

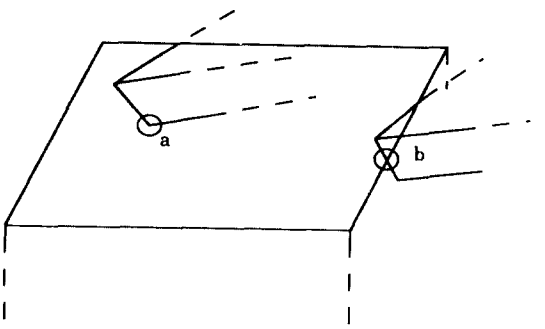

그림 1. 물체 간 충돌의 두가지 경우
Fig. 1. Two different cases for collisions between objects

이 두가지 검색의 기본이 되는 것은 clipping 알고리듬의 하나인 Cyrus-Beck clipping 알고리듬[3]으로 그 기본적인 개념은 다음과 같다. 그림 2에서 보는 바와 같이 굵은 선으로 표시된 무한 직선에 대해 가는 선으로 표시된 무한 직선을 자른다고 생각해 보자. 이때 이 굵은 선의 outward normal vector를 그림과 같이 벡터 n으로 표시하고, 굵은 선위의 임의의 한 점을 a, clipping하고자 하는 선상에 존재하는 임의의 점을 각각 b(굵은 선 밖에 존재), c(굵은 선 안에 존재)라 할 때 벡터 n과 $v_{ba}(v_{bo}-v_{ao})$, $v_{ca}(v_{co}-v_{ao})$ (o 는 기준점) 사이에는 그림 2에서 나타낸 것과 같은 관계가 성립한다. 즉 굵은 선 밖에 있는 임의의 점 b와 굵은 선 상의 임의의 점 a 사이의 벡터

v와 n의 내적은 항상 0보다 크다.

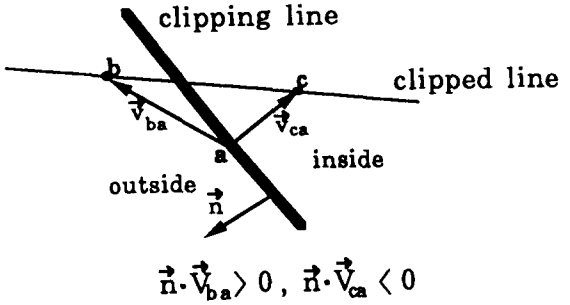

그림 2. Cyrus-Beck clipping 알고리듬의 기본 원리

Fig. 2. Fundamental principle of Cyrus-Beck clipping algorithm.

1. 꼭지점의 충돌

위에서 살펴본 Cyrus-Beck clipping 알고리듬을 기본으로 그림 1의 a)와 같은 꼭지점 충돌을 검색하는 방법을 살펴보자. 이해를 돕기 위해 2D의 경우를 먼저 살펴보면 그림 3과 같이 a 물체와 b 물체는 꼭지점 충돌이 일어난 상태이며 b를 window로 보고 a의 각 꼭지점에 대해서 window의 각 edge에 대한 내적 계산을 하게되면 꼭지점 V_4가 물체 b 안에 들어와 있는 것을 알게 되어 충돌을 알아낸다. 이 경우 a를 window로 보고 b의 각 꼭지점을 테스트 하게되면 충돌을 알 수 없으므로 항상 a에 대해서 b를, b에 대해서 a를 테스트해 보아야 한다.

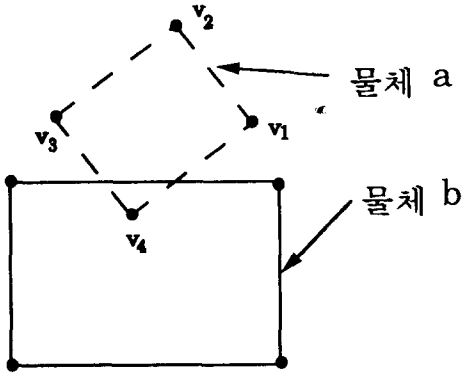

그림 3. 물체 a의 한 꼭지점과 물체 b의 선분의 충돌

Fig. 3. Collision of a vertex of object 'a' against a edge of object 'b'.

위의 예제를 3D로 확장해서 생각하는 것은 선분은 면으로, 선분의 outward normal vector는 면의 outward normal vector로 생각하면 된다. 결국 이 방법은 어떤 한점이 어떤 물체 안에 포함되어 있는가, 없는가를(inclusion test) 검사함으로써 충돌 검색을 해낸다.

2. 선분의 충돌

그림 1의 (b)와 같이 선분의 충돌을 체크하는 방법은 한 물체의 모든 선분에 대해 각각 다른 물체의 모든 면과 교차점 계산을 하고, 각 선분을 교차점에 의해 생기는 몇개의 세그먼트로 나눈후 그 세그먼트의 중점에 대해 위에서 살펴본 포함 계산(inclusion test)를 한다. 예를 들면 그림 4 a)와 같을 때 한 물체의 선분 V_1V_4와 다른 한 물체의 모든 면과의 교차점 계산을 하면 그림 4의 b)와 같이 직선은 몇개의 세그먼트로 나뉘어 지게 된다(그림에서는 4개의 세그먼트). 그후 각 세그먼트의 중점에 대해 위에서 살펴본 포함 테스트를 하게된다. 그림 4에서 보는 바와 같이 직선 1의 모든 세그먼트의 중점은 밖에 있는데 반해, 직선 2의 두번째 세그먼트 중점은 안에 있으므로 충돌이 일어난 경우이다. 위와 같은 계산을 한 물체의 모든 선분에 대해서 수행하면 된다. 이런 계산은 꼭지점 충돌 검색에서와는 달리 테스트 한번만으로도 족하다.

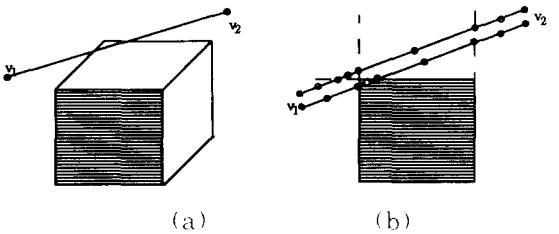

그림 4. 다른 두 물체의 선분 간의 충돌

Fig. 4. Collision of two edges belonging to different objects.

Ⅲ. 충돌 검색 알고리듬의 개선

Ⅱ장에서 살펴본 것을 이용하여 물체 A 와 B의 충돌 검색 알고리듬을 정리하면 다음과 같다.

(Step 1)

물체 A의 모든 꼭지점을 B에 대해 포함 테스트를 하여 충돌이 있는가를 찾는다. 충돌이 없으면 step2로 간다.

(Step2)

물체 B의 모든 꼭지점을 A에 대해 포함 테스트를 하여 충돌이 있는가를 찾는다. 충돌이 없으면 step3로 간다.

(Step3)

물체 A의 모든 선분(B의 모든 선분)을 B의 면(A의 모든 면)과 교차점 계산을 하고 이에 대한 세그먼트의 중점에 대해서 포함 테스트를 하여 충돌이 있는가를 검색한다.

이상과 같이 충돌 검색을 위해서는 모든 꼭지점에 대해서 다른 물체에 대한 포함 테스트와 모든 선분과 다른 물체의 모든 면의 교차점 계산들을 필요로 하므로 많은 계산양이 요구된다. 그런데 위에서 본 충돌 검색 알고리듬에서 첫번째 단계에서 한 계산들로 부터 두번째, 세번째 단계에서 하는 계산들을 테스트에서 제외시키므로써 계산양을 많이 줄일 수 있다. 위의 Step 1,2,3의 과정을 자세히 살펴보기로 하자. 우선, 위에서 설명한 Step 1을 수행한다. 다만 후에 이에 대한 정보를 사용하기 위해 그림 5의 물체 X, Y간 검색과 같은 경우 표 1과 같은 표를 만든다. 각 면에 다른 물체의 꼭지점수 만큼 bit를 할당하여 꼭지점이 면의 내부에 있으면 1을, 외부에 있으면 0을 그 값으로 정해 준다. 어느 한 꼭지점에 대하여 모든 면에 대한 bit값이 1이면, 즉 표 1에서 columnwise AND operation을 하여 나온 결과가 1이면 해당 꼭지점이 다른 물체 내부에 있는 것이고 따라서 충돌이 일어났음을 알게 된다.

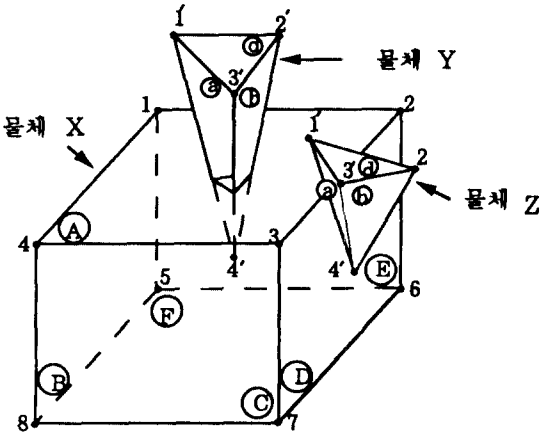

그림 5. 두 물체 간의 충돌(X와 Y, 혹은 X와 Z)
Fig. 5. Collision between two objects(X and Y, or X and Z).

만약 불행히 거꾸로 검색이 이루어졌다면 (Y에 대해 X의 꼭지점의 검사) 표 2와 같은 결과를 얻을 것이다. 표 2에서 보는 바와 같이 AND operation 수행 결과는 모두 0이다.

표　1. 물체 X에 대한 물체 Y의 꼭지점들의 검사
Table 1. Test of the vertices of object Y with respect to object X.

Y의 꼭지점 / X의 면	1′	2′	3′	4′
A	0	0	0	1
B	1	1	1	1
C	1	1	1	1
D	1	1	1	1
E	1	1	1	1
F	1	1	1	1
and	0	0	0	1

그런데 충돌이 일어나는 꼭지점은 상대 물체를 양분하는 면들의 공통 꼭지점에서 일어나므로 그 꼭지점에 대해서만 step 2의 검색을 수행한다. 즉 표 2에서 모든 bit가 1 또는 0이 아닌 면들(a, b, c)은 다른 물체를 양분하는 면이므로 이 면들의 공통 꼭지점을 찾아 이 꼭지점(그림 5에서 4)들에 대해서만 step2의 검색을 한다.

한편 그림 5에서 물체 X와 Z의 충돌 검색의 결과는 표 3과 같고 이상과 같은 방법으로는 이러한 두 물체의 선분 간의 충돌을 검색할 수 없다. 그림 5에서 물체 X, Z의 경우와 같이 선분이 다른 물체의 선분과 충돌이 발생하는 경우는 한 선분이 다른 선분을 공유하는 두 면을 관통함을 알 수 있다. 즉 물체 Z의 선분 4′는 물체 X의 면 A, D를 관통한다. 또 충돌이 발생하는 선분의 두 끝점은 그 면에 대한 in/out test에서 각각 부호가 반대인 결과를 얻는다. 즉, 물체 Z의 꼭지점 1′은 물체 X의 면 A에 대해 바깥쪽이며, 면 D에 대해서는 안쪽이다. 4′는 A에 대해 안

쪽이고, D에 대해서는 바깥쪽이다. 그러므로 선분 충돌이 일어나는 경우는 위와 같은 조건을 만족하는 선분에서만 충돌이 발생하므로 이와같은 선분에 대해서만 step 3의 검색을 행한다.

표 2. 물체 Y에 대한 물체 X의 꼭지점들의 검사
Table 2. Test of the vertices of object X with respect to object Y.

X의 꼭지점 / Y의 면	1	2	3	4	5	6	7	8
a	1	1	0	0	1	1	0	0
b	1	0	0	1	1	0	1	1
c	0	0	1	1	0	0	1	1
d	1	1	1	1	1	1	1	1
and	0	0	0	0	0	0	0	0

표 3. 물체 X에 대한 물체 Z의 꼭지점들의 검사
Table 3. Test of the vertices object Z with respect to object X.

Z의 꼭지점 / X의 면	1′	2′	3′	4′
A	0	0	0	1
B	0	0	0	0
C	0	0	0	0
D	1	0	0	0
E	0	0	0	0
F	0	0	0	0
and	0	0	0	0

구체적인 예를 들어보면 표 3에서 보듯이 면 A를 관통하는 선분은 4′ 1′, 4′ 2′, 4′ 3′ 임을 알 수 있고, 면 D를 관통하는 선분은 1′ 2′, 1′ 3′, 1′ 4′ 임을 알 수 있다. 그러므로 면 A, D를 관통하고 위의 조건을 만족하는 것은 4′ 1′ 선분이다. 따라서 4′ 1′ 선분과 A, D 면과의 교차점 계산을 하게되면 선분과 면을 확장한 무한 평면은 반드시 만나게 된다. 이때 면 A와 선분의 교점을 t_A면 D와의 교점을 t_D라 했을 때, 면 A에 대해서 선분 중 면 안쪽에 있는 부분은 4′ 부터 t_A이고 면 D에 대해서는 t_D부터 1′ 이다. 그림 6에서 보는 바와 같이 $t_A > t_D$이면 t_D부터 부t_A분이 물체의 안 쪽에 들어가 있는 상태이므로 충돌이 일어난 상태이다.

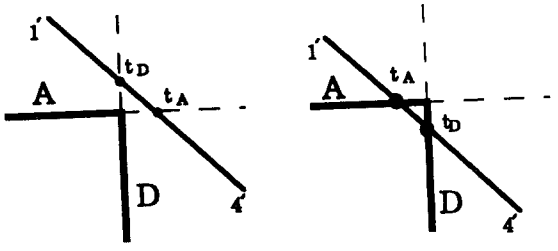

그림 6. 두 선분 간의 충돌을 보여 주는 2차원 투영도
Fig. 6. A 2-D projection of collision between two edges.

Ⅳ. 전체 충돌 검색 시간의 절약

애니메이션 시스템에서 많은 물체가 움직일 때 대부분의 물체는 상당히 떨어져 있는 상태이므로 대부분의 물체간 충돌 검색은 실패하게 된다. 그러므로 모든 물체 사이를 충돌 검색하는 것은 시간 낭비이며 계산 시간을 합리적으로 하기 위해서는 MBB (Minimum Bounding Box) test와 같은 몇가지 테크닉의 사용이 필수적이다. MBB test와 함께 충돌 가능성이 높은 두 물체를 빨리 선택하기 위하여 광선 추적(Ray Tracing)기법에서 흔히 사용하는 space subdivision 방식등을 사용할 수 있겠으나 공간 분할후 매 프레임마다 등록 물체들을 update하는데 시간이 많이 소요되므로 구현된 시스템에서는 물체를 sorting하는 보다 간단한 방법을 채택하여 충돌 검색 시간을 줄였다.

1. 물체의 sorting

MBB test를 가능한 모든 물체 쌍에 대해 검사하

는 것은 $o(n^2)$의 복잡도를 갖는다[6]. 그러므로 이 물체들을 임의의 한 축에 대해 sorting(예를 들어 가장 물체가 넓게 펼쳐져 있는 축)한 후 겹치는 물체들 간에만 검사하는 것은 $o(n\log n)$의 복잡도를 갖는 것이 되고 이 sorting과정은 매 프레임마다 행해져야 한다. 그러나 한번 sorting한 후에는 매 프레임마다 물체간 순서의 변화가 극히 미세하여 극히 적은 수의 물체 이동만으로도 재 sorting이 가능하므로 거의 $o(n)$(amortized complexity)의 복잡도를 가지고 수행할 수 있다.

Ⅴ. 충돌 반응

본 장에서는 충돌이 발생한 후 그 충돌로 인하여 변화하게 되는 물체의 움직임을 실세계와 유사하게 애니메이션하기 위한 물리 관계식을 설명한다[1].

그림 7과 같은 두 물체가 충돌했을 경우 충돌 방정식을 풀기 전에 몇가지 term들을 정의한다.

$\vec{v}_a$, $\vec{\omega}_a$, $\vec{m}_a$는 각각 물체 a의 선속도, 각속도, 질량을 나타내며, $\vec{c}_a$는 물체 a의 무게중심, J_a는 inertial tensor matrix를 나타낸다. $\vec{p}_a$는 물체 a의 무게중심으로부터 충돌점까지의 벡터를 표시한다. 또 충돌 평면(collision plane)이라는 임의의 평면을 정의하는데 충돌 평면이란 충돌점을 포함하는 임의의 평면이다. 편의상 꼭지점이 다른 물체의 면과 충돌했을 경우는 그 면을 충돌 평면으로 정하고 선분끼리 충돌했을 경우는 두 선분끼리 이루는 면으로 잡는다. 또 충돌 평면으로 부터 충돌 프레임(collision frame)을 정의하는데 $\vec{i}$ $\vec{j}$ $\vec{k}$는 각각 충돌 프레임의 단위 벡터로 정한다. 이 때 $\vec{k}$는 충돌 평면의 수직 방향의 성분이고, $\vec{i}$ $\vec{j}$는 평행한 성분이다. 충돌 벡터(impulse momentum) $\vec{R}$은 충돌이 일어나는 짧은 시간에 상호간에 주고받는 힘으로 생각할 수 있으며 단위는 운동량과 같다.

충돌 직후의 새로운 $\vec{v}_a$, $\vec{\omega}_a$를 $\vec{v}_a'$, $\vec{\omega}_a'$라 했을 때 두 물체 1, 2의 충돌 직후에 모르는 값들은

$\vec{v}_1'$, $\vec{v}_2'$, $\vec{\omega}_1'$, $\vec{\omega}_2'$, $\vec{R}$로 15개이며, 이를 풀기 위해 충돌시 가정할 수 있는 15개의 방정식을 세운다[3]. 우선 12개의 선형 방정식은 운동량 보존의 법칙에 따라 아래와 같이 세울 수 있다. 편의상 충돌 벡터 R은 물체 2에서 1로 가해진다고 가정한다.

$$m_1\vec{v}_1' = m_1\vec{v}_1 + \vec{R}$$
$$m_2\vec{v}_2' = m_2\vec{v}_2 - \vec{R}$$
$$J_1\vec{\omega}_1' = J_1\vec{\omega}_1 + \vec{p}_1 \times \vec{R}$$
$$J_2\vec{\omega}_2' = J_2\vec{\omega}_2 - \vec{p}_2 \times \vec{R}$$

나머지 3개의 식은 몇가지 충돌 조건으로 부터 구할 수 있다.

1) 마찰이 무한대일 때

이는 충돌 직후에 충돌점에서의 속도는 같다고 볼 수 있으므로 다음과 같이 표시할 수 있다.

$$\vec{v}_1' + \vec{\omega}_1' \times \vec{p}_1 = \vec{v}_2' + \vec{\omega}_2' \times \vec{p}_2$$

2) 마찰이 없을 때

마찰이 없으므로 충돌 벡터는 충돌 평면의 수직인 면에만 작용된다고 볼 수 있고 세가지의 식은 다음과 같이 표시할 수 있다.

$$\vec{R} \bullet \vec{i} = 0$$
$$\vec{R} \bullet \vec{i} = 0$$
$$(\vec{v}_1' + \vec{\omega}_1' \times \vec{p}_1) \bullet \vec{k} = (\vec{v}_2' + \vec{\omega}_2' \times \vec{p}_2) \bullet \vec{k}$$

3) 마찰계수 γ 가 주어졌을 때

일반적으로 두 물체의 마찰 계수가 같지않으면 마찰계수가 큰 물체의 값을 마찰 계수로 택한다. 마찰 계수 γ 를 다음과 같이 정의한다.

$$\gamma = \frac{|\vec{R}|_{parallel}}{|\vec{R}|_{perpendicular}}$$

먼저 γ 가 무한대일 때로 계산을 한 후 구해진 $\vec{R}$이 위의 식을 만족하면 문제가 없지만 만족하지 않는다면 다음의 식을 세워서 다시 푼다.

$$\vec{R} \bullet \vec{i} = \alpha\vec{R} \bullet \vec{k}$$
$$\vec{R} \bullet \vec{j} = \beta\vec{R} \bullet \vec{k}$$
$$(\vec{v}_1' + \vec{\omega}_1' \times \vec{p}_1) \bullet \vec{k} = (\vec{v}_2' + \vec{\omega}_2' \times \vec{p}_2) \bullet \vec{k}$$

α β 는 다음과 같이 구할 수 있으며 여기서 $\vec{Q}$는 $\vec{R}$의 수직 성분이고, $\vec{P}$는 수평 성분이다.

$$\vec{Q} = \vec{k}(\vec{R} \bullet \vec{k})$$
$$\vec{P} = \frac{\vec{R} - \vec{Q}}{|\vec{R} - \vec{Q}|}$$
$$\alpha = \gamma(\vec{P} \bullet \vec{i})$$
$$\beta = \gamma(\vec{P} \bullet \vec{j})$$

4) 탄성 계수 ε 이 주어졌을 때

두 물체중 작은 탄성 계수를 가지는 물체의 값을 실제적 탄성 계수로 선택하며 위에서 설명한 수식은 실제적 탄성 계수가 0일 때이다. 만약 이 실제적 탄성 계수가 0이 아니라면 위에서 세운 15개의 선형 방정식을 풀고 새로운 충돌 벡터 $\vec{R}_{actual}$을 $(1+\varepsilon_{actual})\vec{R}$로 다시 구한 후 이 $\vec{R}_{actual}$값을 처음에 세운 12개의 식에 대입하여 새로운 $\vec{v}$, $\vec{\omega}$를 다시 구한다.

위에서 세운 연립 방정식을 풀기 위해서는 물론 그들의 로칼 프레임(local frame)의 값들은 공통된 프레임의 값으로 바꾸어 주어야 하고[1], 선형 방정식은 일반적인 Gauss-Jordan elimination이나 LU-decomposition등의 방법으로 구할 수 있다.

Ⅵ. 결 과

위에서 설명한 충돌 검색 알고리듬과 충돌 반응을 구현하기 위해 rigid body를 지원하는 dynamic simulation 시스템을 구현하였으며, 이의 개요도는 그림 8과 같다.

임의의 크기를 갖는 육면체 1000개를 임의로 만들어 움직이게 하고 충돌 검색을 하였을 때 요구되는 계산양과 검색 알고리듬이 수행되는 시간은 표 4와 같다. Step1에서 두 알고리듬의 계산양이 다른 이유는, 일반적 충돌 검색에서는 한 물체의 꼭지점을 다른 물체의 면과 in/out 검사를 할 때 한 면이라도 out으로 판명이 나면 다른 면과 in/out 검사를 하지 않아도 되는 반면, 개선 알고리듬에서는 table을 만들어야 하므로 그 꼭지점이 면의 안쪽에 있든 밖에 있든 모든 면과 in/out검사를 해야만 하기 때문이다. 그리고 물체를 sorting했을 때와 sorting하지 않았을 때의 비교 data는 표 5와 같다. 그림 9는 시스템으로 애니메이션한 예이다.

표 5. Solbourne 5/600에서 100프레임을 만드는데 필요한 MBB 충돌 검사 시간의 비교

Table 5. Comparison of MBB collision test time for generating 100 frames using Solbourne 5/600.

육면체의 수		10	100	300	500	1000
Sorting 안했을때	MBB test 시간	0.05	12	110	295	1162
Sorting 했을때	Sorting 시간	0.0	0.02	0.05	0.55	1.68
	MBB test 시간	0.0	0.5	3.82	11.4	44.9

표 4. a) 충돌 조건
b) 충돌 검색 알고리즘의 계산량

Table 4. a) Conditions of collision
b) Computational complexities of collision detection algorithm

충돌검색 횟수	총 충돌 횟수	Step1에서 충돌 횟수	Step2에서 충돌 횟수	Step3에서 충돌 횟수
1387	79	26	18	35

a)

	Step1에서 곱셈수	Step2에서 곱셈수	Step3에서 곱셈수	Step3에서 나눗셈수	총 곱셈수	총 나눗셈수	시간
충돌 검색 알고리즘	26,702 × 3	23,571 × 3	376,785 × 3	121,833	437,058 × 3	121,833	3.1 sec
개선된 충돌 검색 알고리즘	66,336 × 3	109 × 3	2,895 × 3	383	69,340 × 3	383	0.7 sec

b)

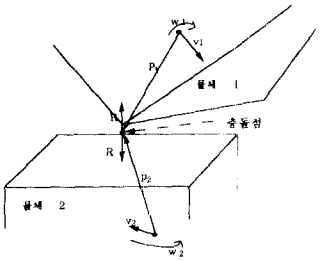

그림 7. 두 single rigid body 간의 충돌
Fig. 7. Collision between two single rigid bodsies

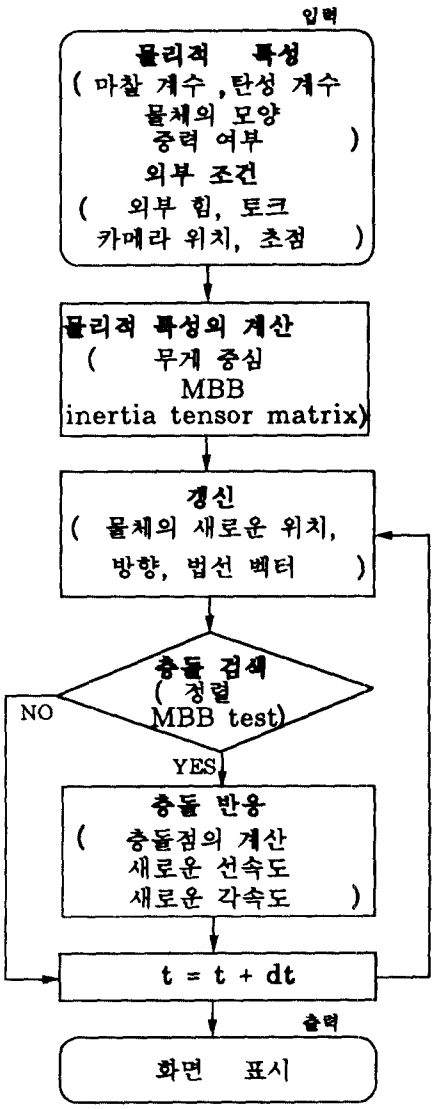

그림 8. 시스템 개요도
Fig 8. System Overview.

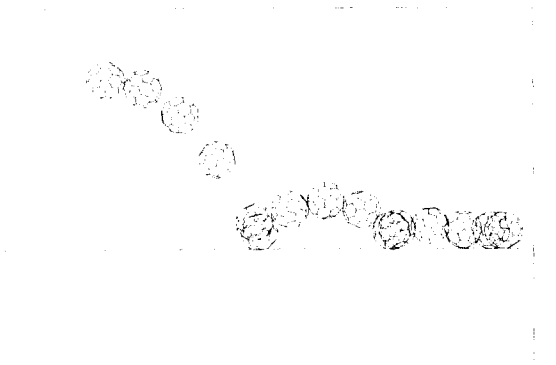

그림 9. 단단한 평면 상에서 다면체의 운동을 보여주는 애니메이션의 예
Fig 9. An animation example showing a bouncing of a polyhedron on a solid floor

Ⅷ. 결 론 및 고 찰

본 논문에서는 dynamic simulation system을 포함한 일반 애니메이션 시스템에서 사용할 수 있는 충돌 검색 알고리듬과 이의 검색 시간을 줄이는 방법을 제안하였고 dynamic simulation system에서 이용될 수 있는 충돌 반응에 대한 이론적 배경과 구현 방법에 대해 설명하였다. 또한 이의 검증을 위해 실제 dynamic simulation system을 구현하였다. 이 시스템에서는 사용자가 물체의 마찰 계수와 탄성 계수를 입력하고 힘을 원하는 물체에 가하므로써 각 물체들은 운동을 하며, 제안된 충돌 검색 알고리듬을 이용하여 주변 물체들과 충돌이 있는가를 검색한 후 충돌이 있다면 역학 관계를 이용하여 시뮬레이션하므로써 실세계와 유사한 물체의 운동을 얻을 수 있었다. 앞으로 system의 확장을 위해 필요한 것들은 우선 시스템이 여러 가지 종류의 articulated rigid body를 지원할 수 있어야 할 것이고, dynamic simulation system의 단점인 motion control의 어려움을 극복하기 위해, inverse dynamic기능의 제공이 요구되어 진다.

參 考 文 獻

[1] J. W. Boyse, "Interference Detection Among Solids and Surfaces," Communications of the ACM, vol. 22:1, pp. 3-

9, Jan. 1979.

[2] M. Moore and J. Williams, "Collision Detection and Response for Computer Animation," Computer Graphics, vol. 22:4, pp. 289-298, Association for Computing Machinery. Proceeding of SIGGRAPH '88.

[3] W. D. MacMillan, *Dynamics of Rigid Bodies*, Dover Publications, Inc. New York, 1936.

[4] J. Wilhelms, "Dynamics for Everyone." Association for Computing Machinery, July 1987. SIGGRAPH' 87 Course 10 Notes: Computer Animation: 3D Motion Specification and Control.

[5] D. F. Rogers, *Procedural Elements for Computer Graphics*, McGraw-Hill Book Company, New York, 1986.

[6] E. Horowitz and S. Sahni, *Fundamentals of Computer Algorithms*, Computer Science, Inc., 1978.

[7] P. M. Isaacs and M. F. Cohen, "Controlling Dynamic Simulation with Kinematic Constraints, Behavior Functions and Inverse Dynamics," Computer Graphics, vol. 21:4, pp. 215-224, Association for Computing Machinery. Proceeding of SIGGRAPH '87.

著者紹介

金 賢 埈 (學生會員)

1967年 2月 26日生. 1989年 2月 아주대학교 전자공학과 졸업. 1991年 2月 한국과학기술원 전기 및 전자공학과 석사학위 취득. 현재 한국과학기술원 박사과정 재학중. 주관심 분야는 Computer Graphics Hardware, Algorithm 등임.

慶 宗 旻(正會員) 第 29 卷 A編 第 2 號 參照
현재 한국과학기술원 전기 및 전자공학과 교수.