http://dx.doi.org/10.7583/JKGS.2012.12.4.23

효율적인 충돌 검출을 위한 거리 기반 SAP 알고리즘※

오민석, 박성준
호서대학교 게임공학과
ominseok@naver.com, sjpark@hoseo.edu

Distance-based SAP Algorithm for Effective Collision Detection

Min-Seok Oh, Sung-Jun Park
Game Engineering, Hoseo University

요 약

충돌 처리는 게임의 물리 법칙을 구현하기 위하여 매우 중요한 요소 중의 하나이며, 게임을 생동감 있게 하기 위한 과정이다. 충돌 처리는 매우 많은 연산이 필요하기 때문에 게임 성능에 중대한 영향을 미친다. 이를 해결하기 위해서 연산량을 줄이는 방식의 연구가 많이 진행되었고, 대표적으로 SAP 알고리즘이 많이 사용되고 있으나 반복되는 연산이 있어 효율성이 떨어진다. 본 논문에서는 충돌 처리의 연산량을 줄이고 SAP 알고리즘의 문제점을 해결하기 위하여, 거리 기반 SAP 알고리즘을 새롭게 제안한다. 본 논문에서 제안한 알고리즘으로 만든 시뮬레이션 프로그램을 이용하여 FPS를 측정하는 실험을 진행한 결과, 제안한 알고리즘을 사용하는 경우가 사용하지 않는 경우에 비해서 FPS가 약 2~33배 높았기 때문에 충돌 처리의 효율이 향상되었다고 판단할 수 있었다.

ABSTRACT

The collision processing is one of the essential factors to realize physical principles in the game, and it gives liveliness to the game. The collision processing requires a large amount of operations, and significantly affects the game performance. To address this problem, many studies have been conducted to reduce the operation volume, and the SAP algorithm is being widely used. However, its efficiency is low because it involves repetitive operations. In this study, a distance-based SAP algorithm was proposed to reduce the operation volume for the collision processing and address the problem of the SAP algorithm. A test was conducted to measure the FPS using the simulation program, which was developed with the proposed algorithm. The FPS was 2-33 times higher with the proposed algorithm, which indicated that the efficiency of the collision processing was improved.

Keywords : Collision Detection, SAP, Sweep and Prune, Game, Broad Phase

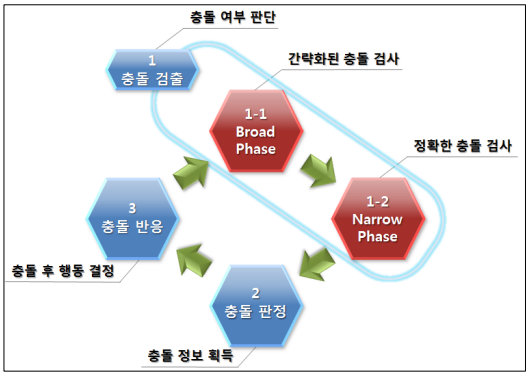
접수일자 : 2012년 07월 06일 심사완료 : 2012년 07월 31일
교신저자(Corresponding Author) : 박성준 (sjpark@hoseo.edu)
※ 이 논문은 2010년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(2010-0023184)

1. 서 론

[그림 1]에서 보는 바와 같이 일반적으로 충돌 처리라 함은 충돌 검출, 충돌 판정, 충돌 반응으로 나눌 수 있다. 충돌 검출은 충돌이 일어났는지 여부를 판단하는 것을 의미하고, 충돌 판정은 충돌 부분을 찾아내는 것을 의미하며, 충돌 반응은 충돌의 결과로 어떤 행동을 취할 것인지 결정하는 것을 의미한다[1,2].

이 중에서 충돌 검출은 BP(넓은 범위의 처리 단계, Broad-Phase)와 NP(좁은 범위의 처리 단계, Narrow-Phase)의 두 가지로 그룹화 할 수 있다[1,3].

BP는 충돌 검출을 정확하게 하는 것이 아니라 간략화 된 방식을 통하여 충돌 가능성이 없는 것들을 제외하는 단계를 의미한다. 이 단계에서 검출된 쌍이 실제로 충돌될 수도 있고, 되지 않을 수도 있다. BP에서 검출된 충돌 쌍들에 대해 NP에서 정확한 검사를 하여 실제로 충돌 하는지를 검출한다. 이렇게 두 단계에 걸쳐 충돌 검출을 진행하는 이유는 NP가 많은 계산을 필요로 하기 때문에 BP에서 미리 걸러내어 불필요한 계산을 줄이기 위해서이다[4,5,6].



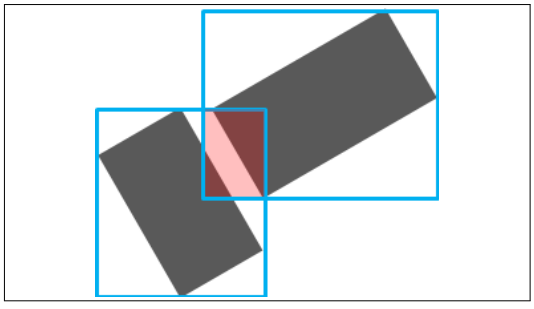
[그림 1] 충돌 처리 개요

본 논문에서는 임의의 위치에서부터 오브젝트의 BV(Bounding Volume)의 위치까지의 거리 정보를 가지고 SAP(Sweep And Prune, Sort And Sweep) 알고리즘의 일부분을 차용해서 사용한, BP에서 사용할 수 있는 알고리즘의 하나를 개발 했으며, 해당 알고리즘의 대한 성능 평가 및 고찰을 위한 실험을 수행 하였다. 실험 결과, 제안한 알고리즘을 이용하지 않았을 때보다 약 2~33배 빠른 속도를 보여주었으며, 이를 통하여 BP에서 사용할 수 있는 새로운 알고리즘을 알 수 있었다[2].

2. 관련 연구

BP에는 여러 가지 알고리즘을 사용할 수 있는데 All-pair test, SAP 알고리즘 등이 있다[1,3].

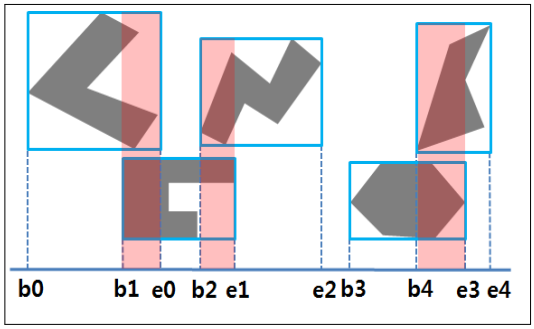
[그림 2]에서 보는 바와 같이 All-pair test 알고리즘은 철저한 검색 방식으로 각 개체의 BV가 교차하는 것을 판단하여 충돌쌍을 검출하는 방식이다. All-pair test 알고리즘은 BV의 모양에 따라 다양한 교차 판단 방식이 있으며, 가장 대표적인 방식은 직육면체끼리의 교차 판단 방법이다. 이 방법은 직육면체 모양의 BV의 각 축간 최소값과 최대값을 구한 뒤 다른 물체의 BV에서 구한 값과 비교를 해서 변이 겹치는지 판단하는 것을 각 축마다 반복하고, 모든 축에 겹치면 교차한다고 판단한다.



[그림 2] All-pair test 알고리즘

[그림 3]에서 보는 바와 같이 SAP 알고리즘

은 임의의 축에서 각 BV의 시작점(b0~b4)과 끝점(e0~e4)을 이용해서 교차 판단을 하고, 이것을 축마다 반복해서 모든 축에서 교차할 때 충돌한다고 판단하여 충돌쌍을 검출하는 방식이다. SAP 알고리즘은 한 프레임에서 다음 프레임으로 갈 때 물체들이 위치나 방향을 거의 바꾸지 않는다는 시간 일관성을 이용하여 이전 프레임의 정보를 보관하여 갱신하기 때문에, 움직이지 않는 오브젝트가 많을 경우 효율적이다[2,4].



[그림 3] SAP 알고리즘

[그림 4]에서 보는 바와 같이 본 논문에서는 SAP 알고리즘의 시간 일관성 부분은 제외하고 교차 판단을 하는 부분만 차용해서 사용하는 알고리즘, 거리 기반 SAP 알고리즘(D-SAP 알고리즘, Distance-based Sweep And Prune Algorithm)을 제안한다. D-SAP 알고리즘은 SAP 알고리즘에서 사용한 3개의 축의 위치/크기를 비교하는 대신에 거리를 이용해서 한번만 비교하기 때문에 효율적이고, BP에서 사용할 수 있다.

[그림 4] D-SAP 알고리즘

3. 시스템 구성

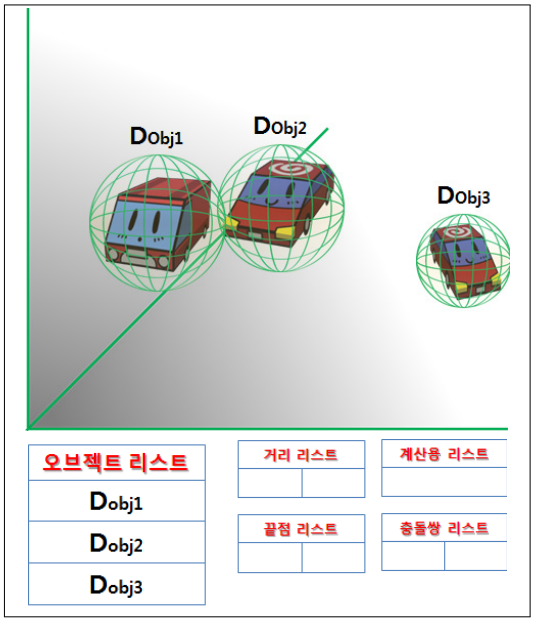
3.1 D-SAP 알고리즘

본 논문에서 제안한 D-SAP 알고리즘은 Clear List, Calc Distance, Prev Process, Process의 4단계이며 충돌 처리를 할 때 BP에서 실행한다.

[그림 5,6]에서 보는 바와 같이 D-SAP 알고리즘 1단계는 사용하는 리스트를 초기화하는 단계이다. 오브젝트 정보를 담고 있는 오브젝트 리스트(list_Object)를 제외한 모든 리스트는 매회 재사용하기 때문에 초기화한다.

```
procedure Clear_List
  clear list_Distance;
  clear list_End_Point;
  clear list_End_Point_Calc;
  clear list_Coll_Pair;
end
```

[그림 5] D-SAP 알고리즘 1단계 의사코드

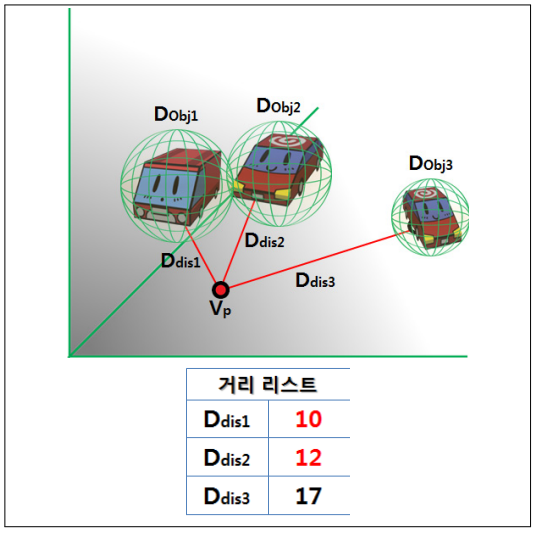


[그림 6] D-SAP 알고리즘 1단계 Clear_List

[그림 7,8]에서 보는 바와 같이 D-SAP 알고리즘 2단계는 임의의 위치(V_p)로부터 오브젝트의 BV의 위치(V_{bp})까지의 거리(D_{dis})를 계산해서 거리 리스트(list_Distance)에 삽입하는 단계이다.

```
procedure Calc_Distance
  Vp = Create_Pivot_Point();
  for each Dobj in list_Object do
    Vbp = Dobj->Get_BV_Pos();
    Ddis = Calc_Distance_Between(Vbp,Vp);
    insert Ddis into list_Distance;
  end
end
```

[그림 7] D-SAP 알고리즘 2단계 의사코드



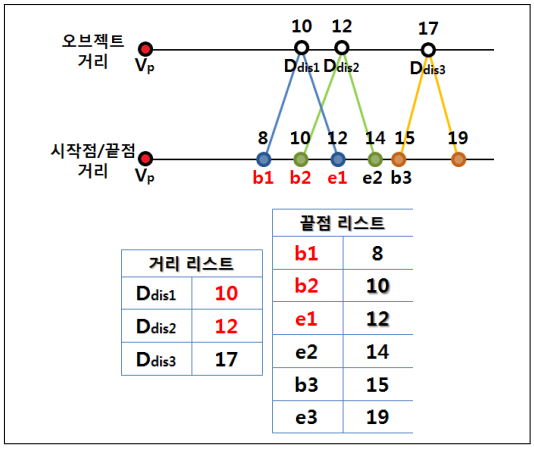
거리 리스트	
Ddis1	10
Ddis2	12
Ddis3	17

[그림 8] D-SAP 알고리즘 2단계 Calc_Distance

[그림 9,10]에서 보는 바와 같이 D-SAP 알고리즘 3단계는 2단계에서 계산된 거리 리스트의 원소(D_{dis})들로 시작점(EP_{begin}, 거리-BV 크기의 반, $S_{dis}-S_{size}/2$)과 끝점(EP_{end}, 거리+BV 크기의 반, $S_{dis}+S_{size}/2$)을 계산해서 끝점 리스트(list_End_Point)에 삽입하는 단계이다. 이 시작점과 끝점에는 자신이 시작점인지 끝점인지를 구분하는 값과 데이터 값(거리±BV크기의 반, 정렬하기 위해서 필요하다.)과 그 점의 주인인 오브젝트의 포인터가 들어있다. 이 리스트는 반드시 정렬해야 한다.

```
procedure Prev_Process
  for each Ddis in list_Distance do
    Ssize = Ddis->Get_Obj()->Get_BV_Size()/2;
    Sdis = Ddis->Get_Distance();
    EPbegin = Create_End_Point(Sdis-Ssize, true);
    insert EPbegin into list_End_Point;
    EPend = Create_End_Point(Sdis+Ssize, false);
    insert EPend into list_End_Point;
  end
end
```

[그림 9] D-SAP 알고리즘 3단계 의사코드



거리 리스트	
Ddis1	10
Ddis2	12
Ddis3	17

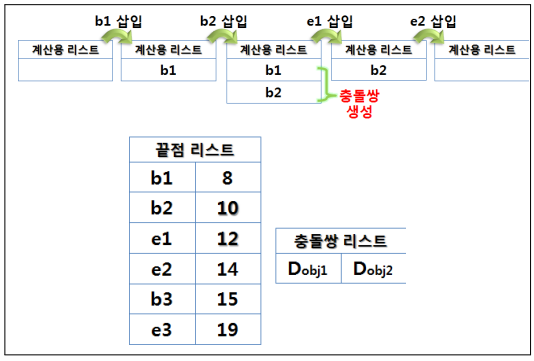
끝점 리스트	
b1	8
b2	10
e1	12
e2	14
b3	15
e3	19

[그림 10] D-SAP 알고리즘 3단계 Prev_Process

[그림 11,12]에서 보는 바와 같이 D-SAP 알고리즘 4단계는 생성된 끝점 리스트의 원소(EP_p)들을 이용해서 충돌쌍(CP_p)을 검출하는 단계이다. 처리중인 끝점 리스트의 원소가 시작점이면 계산용 리스트(list_End_Point_Calc)에 삽입한다. 이때 계산용 리스트에 다른 시작점이 있다면, 그 두 개의 시작점(EP_p, EP_{calc})으로 충돌쌍을 만들어서 충돌쌍 리스트(list_Coll_Pair)에 삽입한다. 처리중인 끝점 리스트의 원소가 끝점이면 계산용 리스트에서 해당하는 시작점을 찾아 삭제한다. 이 일은 끝점 리스트의 모든 원소들을 처리할 때까지 반복한다.

```
procedure Process
  for each EPp in list_End_Point do
    if EPp->Is_Begin() is true then
      for each EPcalc in list_End_Point_Calc do
        CPp = Create_Coll_Pair(EPp->Get_Obj(),
                               EPcalc->Get_Obj());
        insert CPp into list_Coll_Pair;
      end
      insert EPp into list_End_Point_Calc
    else if EPp->Is_Begin() is false then
      EPmp = Find_Matching_End_Point(
                  EPp, list_End_Point_Calc);
      erase EPmp from list_End_Point_Calc
    end
  end
end
```

[그림 11] D-SAP 알고리즘 4단계 의사코드



[그림 12] D-SAP 알고리즘 4단계 Process

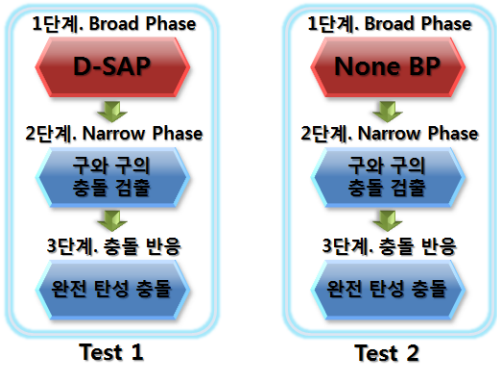
3.2 D-SAP 알고리즘을 이용한 시뮬레이션의 구성

[그림 13]에서 보는 바와 같이 시뮬레이션 프로그램은 BP에서 D-SAP 알고리즘을 이용했을 때와 이용하지 않았을 때의 차이를 알아보기 위해서 3단계의 두 가지 테스트로 진행했다.

시뮬레이션 프로그램의 1단계는 정확한 충돌 검출 전에 간략화해서 충돌 가능성이 없는 것들을 제외하는 단계, 즉 BP이다. 이 단계에서는 D-SAP 알고리즘을 이용한 테스트와 이용하지 않은 테스트, 두 가지로 나눠서 진행했다. D-SAP 알고리즘을 이용한 테스트는 D-SAP 알고리즘을 통해 충돌쌍 리스트를 생성했으며, D-SAP 알고리즘을 이용하지 않은 테스트는 모든 오브젝트끼리 중복되지 않는 충돌쌍을 만들어서 충돌쌍 리스트를 생성했다.

시뮬레이션 프로그램의 2단계는 정확한 충돌 검출을 하는 단계, 즉 NP이다. 1단계에서 만들어진 충돌쌍들을 이용해 실제로 충돌하는지 검사하여 중복되지 않는 모든 충돌쌍을 구했다. 1단계에 관계없이 두 가지 테스트 모두 동일한 알고리즘을 적용했다.

시뮬레이션 프로그램의 3단계는 검출된 충돌쌍들을 가지고 반응을 하는 단계, 즉 충돌 반응 단계이다. 본 논문과는 관계가 없는 부분이기 때문에 영향을 최소화하기 위해 구체와 완전탄성 충돌을 이용했고, 마찬가지로 1단계에 관계없이 두 가지 테스트 모두 동일한 알고리즘을 적용했다.



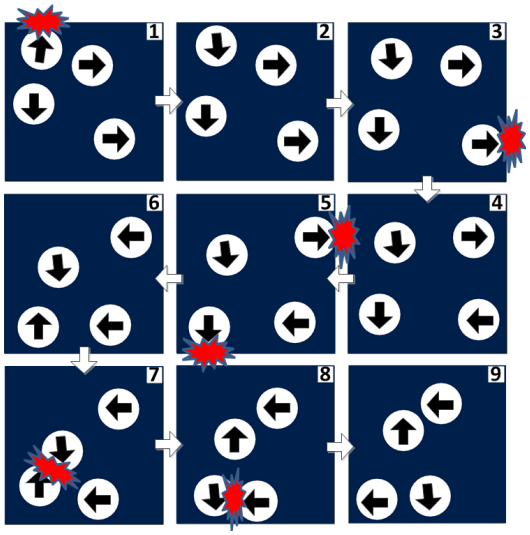
[그림 13] 시뮬레이션 프로그램의 구성

3.3 D-SAP 알고리즘을 이용한 시뮬레이션의 구현

시뮬레이션 프로그램은 실험 결과를 가시적으로 정확하게 평가하기 위하여, 시점을 위에서 아래를 바라보게 하였고, 서로 겹치지 않게 오브젝트를 생성한 뒤에 x축과 z축 위에서만 등속도로 이동하도록 했으며, 마찰력 등의 외부 요인을 적용하지 않았다. 오브젝트가 윈도우 창을 벗어나면 화면 안쪽으로 들어오도록 방향을 조절했다.

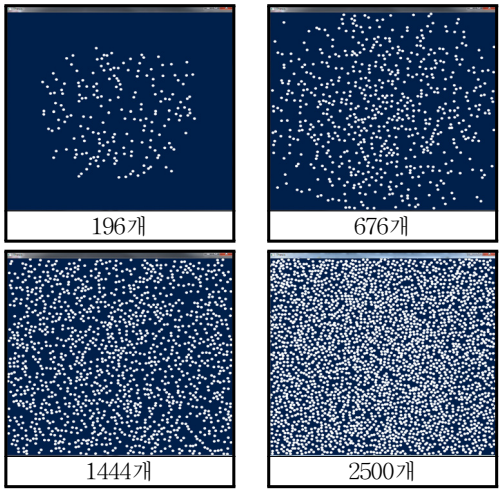
모든 오브젝트는 서로 충돌 처리하며 완전탄성 충돌을 하도록 했다.

[그림 14]는 시뮬레이션 프로그램의 흐름을 보여주기 위하여, 확대 생성된 오브젝트 4개의 움직임과 충돌을 도식화하여 나타낸 것이다. [그림 14]의 1, 3, 5는 오브젝트(구)가 벽에 부딪혀 방향이 바뀌는 모습이고, [그림 14]의 7, 8은 오브젝트끼리 충돌하는 모습이다. 충돌한 두 오브젝트는 완전 탄성 충돌을 하여 서로의 속도가 교환된다.



[그림 14] 시뮬레이션 프로그램의 흐름

이러한 시뮬레이션 프로그램을 이용하여 오브젝트의 개수를 0개부터 10,000개까지 변화를 주면서 실험을 진행했다. [그림 15]는 오브젝트의 개수의 변화에 따라 바뀌는 시뮬레이션 프로그램의 실행 화면이다. 흰색 점 모양의 오브젝트가 보이며, 오브젝트의 개수가 점차 늘어날 때의 모습을 볼 수 있다. 서로 겹치지 않게 오브젝트를 생성했기 때문에 일정 개수 이상의 오브젝트는 화면밖에 생성되어서 점차 안쪽으로 들어오게 된다. 외부 요인을 계산하지 않기 때문에 모든 오브젝트가 감속 없이 계속 움직이는 모습을 확인할 수 있다.



[그림 15] 시뮬레이션 프로그램 실행 화면

4. 실험 및 결과

4.1 실험 환경의 구성

실험은 2010년 이후 출시된 3D MMORPG를 구동했을 때, FPS가 30이하로 내려가는 저사양과 FPS가 50이상으로 유지되는 고사양의 두 가지 컴퓨터에서 진행했다. 우선 저사양 컴퓨터로 AMD Phenom II X4 810 2.6GHz, 4GB Ram, Radeon HD 4850 512MB를 준비했고, 고사양 컴퓨터로 Intel I7-3770 4.3GHz, 8GB Ram, Radeon HD 6850 1GB를 준비했다. 두 컴퓨터에서 Windows7 64비트 운영체제를 이용하였고, 단일 CPU, 단일 쓰레드만 이용해서 진행하였다.

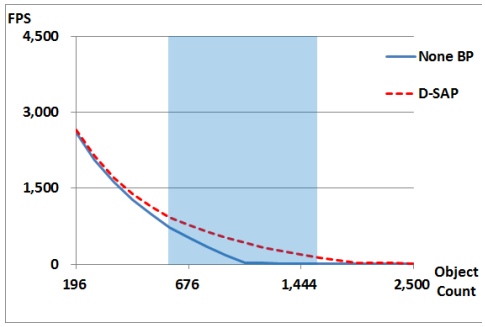
4.2 실험 결과

시뮬레이션 프로그램을 이용해서 얻어낸 결과를 두 가지 항목으로 구분해서 정리했다. D-SAP 알고리즘을 이용했을 경우(D-SAP)와 이용하지 않았을 경우(NoneBP)의 FPS와 그 FPS로 계산한 FPS 효율이다. 이 FPS 효율은 D-SAP의 FPS를 NoneBP의 FPS로 나눠서 얻었다. 만약 이 FPS 효율이 100%라면 D-SAP와

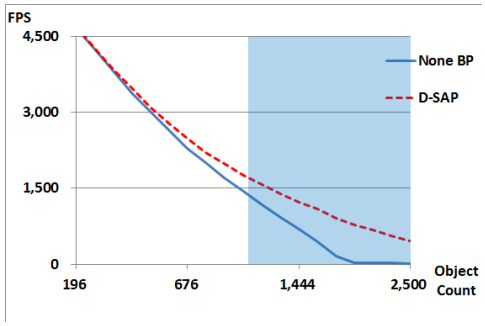
NoneBP의 효율이 동일하다고 할 수 있다.

실험 결과 저사양 컴퓨터와 고사양 컴퓨터 모두 D-SAP 알고리즘을 이용했을 때가 그렇지 않았을 때보다 FPS 및 효율이 약 2~33배 높다고 판단할 수 있었다.

[그림 16,17]은 FPS 측정 결과인데, 두 그래프 모두 전반적으로 D-SAP의 값이 높게 나왔으며, 특히 오브젝트 개수가 저사양 컴퓨터에서 약 500~1,500개일 경우, 고사양 컴퓨터에서 약 1,000~2,500개일 경우 FPS 차이가 크게 나타났다.

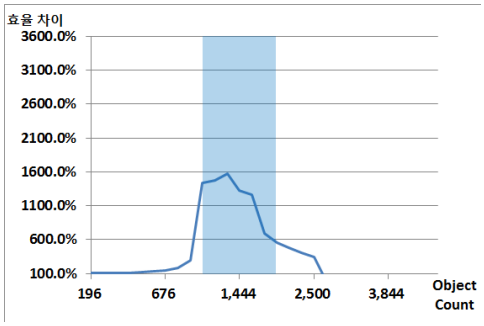


[그림 16] FPS 측정 결과(저사양 컴퓨터)

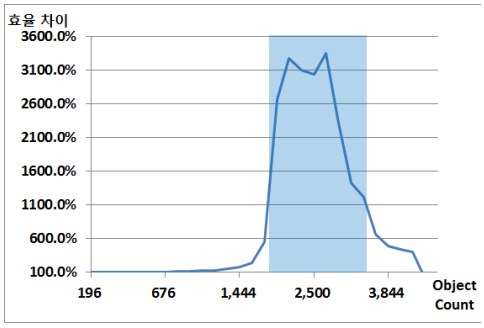


[그림 17] FPS 측정 결과(고사양 컴퓨터)

[그림 18,19]은 FPS 효율을 비교해 본 결과인데, 두 그래프 모두 D-SAP가 효율적으로 나타났으며, 특히 오브젝트 개수가 저사양 컴퓨터에서 약 1,000~2,000개일 경우, 고사양 컴퓨터에서 약 2,000~3,000개 정도일 때 효율적이었다.



[그림 18] FPS 효율(저사양 컴퓨터)



[그림 19] FPS 효율(고사양 컴퓨터)

[표 1]은 저사양 컴퓨터에서 실험한 결과인데, 약 1,000개 미만의 오브젝트가 있을 경우에 100~300% 정도의 효율을 나타냈고, 약 1,000~2,000개의 오브젝트가 있을 경우 500~1500% 정도의 효율을 나타냈으며, 그 이상의 오브젝트가 있을 경우 400% 정도의 효율을 나타냈다.

저사양 컴퓨터에서 약 3,000개의 오브젝트가 있을 경우에 NoneBP는 FPS가 0에 가까운 반면, D-SAP는 10이상을 유지했다.

한편 FPS가 20 미만으로 떨어지는 경우의 오브젝트 개수는 NoneBP는 약 1,000개 이후, D-SAP는 약 2,000개 이후로, 오브젝트 개수의 차이가 약 2배였다.

[표 1] FPS 측정 결과(저사양 컴퓨터)

	FPS 값		FPS 효율
Object 개수	NoneBP	D-SAP	D-SAP/NoneBP
196	2,603	2,660	102.2%
576	718	917	127.8%
1,024	**30**	**429**	**1441.8%**
1,444	**15**	**196**	**1320.5%**
1,764	**10**	**68**	**686.9%**
1,936	**5**	**28**	**551.1%**
2,116	5	24	475.0%
2,304	5	20	402.3%
3,600	0	10	-
4,356	0	5	-
10,000	0	0	-

[표 2]는 고사양 컴퓨터에서 실험한 결과인데, 약 1,700개 미만의 오브젝트가 있을 경우 100~500% 정도의 효율을 나타냈고, 약 2,000~3,000개의 오브젝트가 있을 경우 2,300~3,300% 정도의 효율을 나타냈으며, 그 이상의 오브젝트가 있을 경우 400~600% 정도의 효율을 나타냈다.

고사양 컴퓨터에서 약 4,500개의 오브젝트가 있을 경우에 NoneBP는 FPS가 0에 가까운 반면, D-SAP는 18이상을 유지했다.

한편 고사양 컴퓨터에서 FPS가 20 미만으로 떨어지는 경우의 오브젝트 개수는 NoneBP는 약 2,000개 이후, D-SAP는 약 4,000개 이후로, 오브젝트 개수의 차이가 약 2배였다.

[표 2] FPS 측정 결과(고사양 컴퓨터)

	FPS 값		FPS 효율
Object 개수	NoneBP	D-SAP	D-SAP/NoneBP
196	4,671	4,706	100.8%
576	2,646	2,783	105.2%
1,024	1,437	1,761	122.6%
1,444	697	1,224	175.6%
1,764	163	905	556.1%
1,936	**29**	**780**	**2664.8%**
2,116	**21**	**671**	**3269.2%**
2,304	**18**	**564**	**3094.0%**
3,600	5	33	663.8%
4,356	5	20	408.9%
10,000	0	5	-

5. 결론 및 향후 연구

본 논문은 BP에서 사용할 수 있는 새로운 알고리즘인 D-SAP 알고리즘을 제안하였다. SAP 알고리즘의 일부를 차용하였으며, 거리의 개념을 추가하여 기존 SAP 알고리즘에서 위치를 거리로 대체하였고, 그에 따라 3개의 축(x,y,z)에 각각 시작점/끝점을 생성하고 정렬 후 검사해야 했던 작업을, 단지 한 번의 거리 계산 후 시작점/끝점을 생성하고 정렬 후 검사하는 작업으로 수정했다.

제안한 알고리즘을 이용해 시뮬레이션 프로그램을 만들어서 실험을 했으며, 그 결과 D-SAP 알고리즘을 이용하지 않는 것보다 이용하는 것이 약 2~33배 효율적이라는 것을 알 수 있었고, 불필요한 연산이 줄었다고 판단할 수 있었으며, 더 빠르다는 것을 알 수 있었다.

향후에는 D-SAP 알고리즘을 수정하여 보다 뛰어난 성능을 가지게 하기 위해 병렬 프로그래밍과 GPGPU를 이용하는 알고리즘을 개발할 예정이며, D-SAP 알고리즘과 BP에서 사용하는 다른 알고리즘과의 성능 비교가 필요하다.

참고 문헌

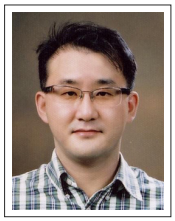
[1] Christer Ericson, “Real-Time Collision Detection”, Morgan Kaufmann, 2005.
[2] Tomas Akenine-Moller, Eric Haines, "Real-Time Rendering", 정보문화사, 2003.
[3] S. Kockara, T. Halic, K. Iqbal, “Collision Detection: A Survey”, Systems, Man and Cybernetics, 2007. ISIC. IEEE International Conference, pp. 4046-4051, 2007.
[4] Daniel J. Tracy, Samuel R. Buss, Bryan M. Woods, “Efficient Large-Scale Sweep And Prune Methods with AABB Insertion and Removal”, Virtual Reality Conference, 2009. VR 2009. IEEE, pp. 191-198, 2009.
[5] Stefan Gottschalk, “Collision Detection Techniques for 3D Models”, 1997.

[6] Dong-Jin Kim, Leonidas J. Guibas, Sung-Yong Shin, "Fast Collision Detection Among Multiple Moving Spheres", Computer Animation '97, pp. 1-7, 1997.
[7] Xavier Provot, "Collision and self-collision handling in Cloth model dedicated to design garments", Institut National de Recherche en Informatique et Automatique(INRIA), pp. 177-189, 1997.
[8] Zhiwen Yu, Hau-san Wong, "GPCD : Grid-based Predictive Collision Detection for Large-scale Environments in Computer Games", Multimedia and Expo, 2006 IEEE International Conference on, pp. 1025-1028, 2006.
[9] Rafael de Sousa Rocha, Maria Andreia Formico Rodrigues, Leandro da Silva Taddeo, "Performance Evaluation of a Hybrid Algorithm for Collision Detection in Crowded Interactive Environments", Computer Graphics and Image Processing, 2007. SIBGRAPI '06. 19th Brazilian Symposium on, pp. 86-93, 2006.
[10] Daniel S. Coming, Oliver G. Staadt, "Velocity-Aligned Discrete Oriented Polytopes for Dynamic Collision Detection", Visualization and Computer Graphics, IEEE Transactions on, pp. 1-12, 2008.
[11] Daniel S. Coming, Oliver G. Staadt, "Kinetic Sweep and Prune for Collision Detection", Proceedings of the Second Workshop in Virtual Reality Interactions and Physical Simulations(VRIPHYS'05), 2005.

오 민 석 (Min-Seok, Oh)

2012년 호서대학교 게임공학과 (학사)
2012년-(현) 호서대학교 게임학과 (석사)

관심분야 : 게임공학, 프로그래밍



박 성 준 (Sung-Jun, Park)

1997년 호서대학교 컴퓨터공학과 (학사)
1999년 건국대학교 컴퓨터공학과 (석사)
2005년 건국대학교 컴퓨터공학과 (박사)
2006년-(현) 호서대학교 게임학과 부교수

관심분야 : 게임공학, 가상현실, HCI, Bioinformatics