Maya에서 다량의 파편 제어 Plug-in 구현

왕성장*, 김창헌*
*고려대학교 컴퓨터정보통신대학원 디지털정보・미디어공학과
e-mail:sjwangfx@korea.ac.kr

A Large Number of Fragments Control Plug-in Framework for Maya

Sung-Jang Wang*, Chang-Hun Kim*
*Dept of Digital Information & Media Engineering, Graduate School of Computer & Information Technology, Korea University

요 약

일반적으로 온전한 형태의 구조물을 파괴하고 물리적인 움직임을 주기 위해서는 리지드 바디 시뮬레이션(Rigid Body Simulation)을 이용한다. 그러나 다량의 복잡한 파편을 시뮬레이션하기에는 리지드 바디 시뮬레이션은 과도한 계산량과 소요시간으로 인해서 그 효용성이 상당히 떨어진다. 이에 리지드 바디 시뮬레이션보다 상대적으로 계산량과 소요시간을 줄여주고, 효과적으로 다량의 복잡한 파편의 움직임을 사실적으로 시뮬레이션하기 위한 방법을 제시하며 Maya의 C++ API 및 MEL을 사용하여 구현하였다.

1. 서론

현재 상용되고 있는 3D 컴퓨터 그래픽 영상 제작 소프트웨어로는 Maya, 3DS Max, XSI, Houdini 등이 있으며, 영화 및 드라마에서 사실적인 효과나 환상적인 장면을 연출하기 위해서 많이 쓰이고 있다.

3D 컴퓨터 그래픽 환경에서 캐릭터나 물체를 움직이기 위해서는 수작업으로 일일이 지정하는 방식이나 키프레임을 이용한 보간 방식을 주로 사용한다. 전자의 경우는 비교적 사실적인 움직임의 생성은 가능하지만 작업량이 매우 많다는 단점이 있고, 후자는 작업량은 적지만 움직임이 그다지 사실적이지 못하다는 단점을 지니고 있다. 그래서 적은 작업량만으로 사실적인 애니메이션을 제작하기 위한 대안으로 제시된 것이 물리기반의 애니메이션 제작방식이다. 이 방식은 물체의 초기 위치와 마지막 위치 등과 같은 적은 양의 정보만을 설정한 후, 중력과 마찰 등의 물리학적인 요소를 움직임에 적용하여 자동으로 애니메이션을 제작한다. 따라서 다량의 파편과 같은 물체는 리지드 바디 시뮬레이션이라는 물리기반의 애니메이션 제작방식으로 자연스러운 움직임을 만들어 낼 수가 있다. 하지만 대부분의 경우 알고리즘이 복잡하여 프레임 당 생성시간이 길고, 결국 제작에 소요되는 시간이 많아진다는 것이 단점으로 지적되고 있다[1].

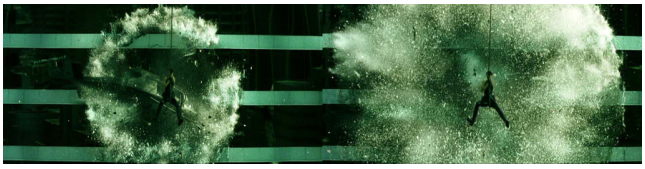
본 논문은 Maya가 제공하는 시뮬레이션 시스템 기반하에서 리지드 바디 시뮬레이션(Rigid Body Simulation)을 사용하는 대신 파티클 시뮬레이션(Particle Simulation)을 사용해서 상대적으로 계산량과 소요시간을 줄이며 다량의 파편을 사실적으로 애니메이션할 수 있는 방법을 제시하고, Maya의 C++ API 및 MEL을 사용하여 구현하여서 3D 컴퓨터 그래픽 영상 제작에 도움을 줄 수 있도록 하였다.

2. 관련연구 및 배경지식

지난 몇 년간 리지디 바디 시뮬레이션을 고려한 다수의 방법들이 제안되어 왔다. Milenkovic은 회전하지 않는 다수의 다면체와 구를 시뮬레이션하는 방법을 제안하였고, Brogan은 분산 환경에서 많은 수의 가축들을 시뮬레이션하였다[2][3]. Miritich는 물리 기반의 리지드 바디 시뮬레이션 생성에 걸리는 시간을 단축하기 위하여 Jefferson이 1985년도에 제안한 Timewarp 알고리즘을 리지드 바디에 적용하는 방식을 제안하였다[4]. 이 알고리즘은 다수의 리지드 바디 애니메이션 제작에 걸리는 시간을 크게 감소시켰다. 더 나아가서 2001년도에 Timewarp 리지드 바디 시뮬레이션을 개선하여 컴퓨터 게임과 같은 실시간 애니메이션 생성을 가능하게 하는 방법을 제시하였다[1].

2004년도에는 Blast Code(FerReel Animation Labs 사)라는 Maya 용 플러그인이 상용화되어 구조물의 파괴 장면에서 다량의 파편 제작에 쓰이고 있다[5].

다년간에 걸쳐서 알고리즘이 개선되었지만 여전히 리지드 바디 시뮬레이션은 물체와 물체가 서로 겹쳐 있게 되면 연산 오류를 일으키고, 물체 간에 충돌을 계산하는데 상당한 시간을 필요로 한다. 이에 반해 파티클 시뮬레이션은 각 파티클의 위치 값만을 고려해서 계산하므로 리지드 바디 시뮬레이션에 비해 상대적으로 계산량과 소요시간이 적게 들고 파티클을 물체로 대체할 수도 있다.



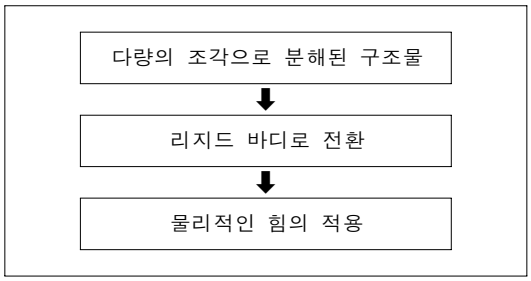
(a) “The Matrix”, 1999

(b) “X-MEN III”, 2006

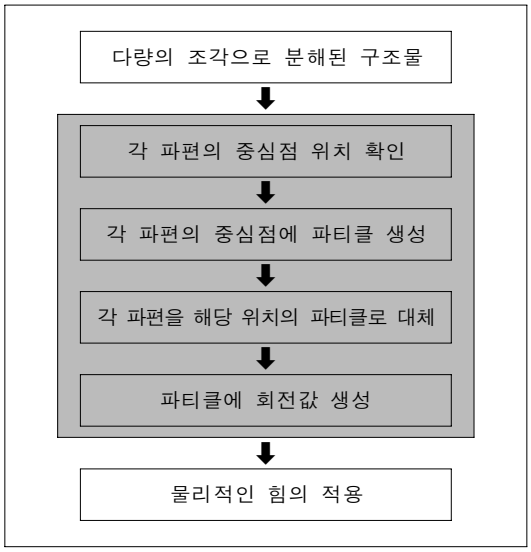
(그림 1) 다량의 파편 효과가 구현된 영화

3. 설계 및 구현

이 절에서는 리지드 바디 시뮬레이션보다 계산량과 소요시간이 적게 들며 제어가 용이한 파티클 시뮬레이션을 기반으로 하여 온전한 형태의 물체로부터 다량의 파편을 제어할 수 있는 방법을 (그림 2) (b)와 같이 제시하였다.



(a) Rigid Body Simulation (일반적인 경우)



(b) Replaced Particle Simulation

(그림 2) 다량의 파편 제어 시뮬레이션 방법

3.1 각 파편의 중심점에 대한 위치 확인용 더미 생성

절대좌표 체계에서의 위치값을 확인하기 위해서 각 파편이 위치한 곳에 더미를 생성한다. MEL 소스 코드는 다음과 같다.

```
select -r "shard*";

string $obj[] = `ls -sl`;
string $loc[];
int $numObj = size($obj)/2;
int $i, $j=0;

for ($i=0; $i<$numObj; $i++) {
        spaceLocator -p 0 0 0;
        rename "pLoc#";

        $loc[$i] = "pLoc" + ($i+1);
}

for ($i=0; $i<$numObj; $i++) {
        pointConstraint -offset 0 0 0 -weight 1 $obj[$i] $loc[$i];
}
```

3.2 더미를 이용하여 각 파편의 중심점에 파티클 생성

파편이 위치한 순서대로 대응하는 파티클을 각 파편의 중심점에 차례대로 생성한다. MEL 소스 코드는 다음과 같다.

```
float $px[], $py[], $pz[];
string $getPx[], $getPy[], $getPz[];
string $pos;

for ($i=0; $i<$numObj; $i++){
        $getPx[$i] = "getAttr " + $loc[$i] + ".tx;";
        $getPy[$i] = "getAttr " + $loc[$i] + ".ty;";
        $getPz[$i] = "getAttr " + $loc[$i] + ".tz;";

        $px[$i] = eval($getPx[$i]);
        $py[$i] = eval($getPy[$i]);
        $pz[$i] = eval($getPz[$i]);

        $pos += " -p " + $px[$i] + " " + $py[$i] + " " + $pz[$i];
}

// particle -p $px[0] $py[0] $pz[0] -p $px[1] $py[1] $pz[1] ...;
string $makeParticle = "particle" + $pos + ";";
eval($makeParticle);
rename "debrisParticle1";
setAttr debrisParticleShape1.particleRenderType 2;

delete "pLoc*";
```

3.3 각 파편을 해당 위치의 파티클로 대체

모든 파편을 절대좌표 체계에서 원점에 위치시키고 위치값을 0으로 초기화시킨다. 그리고 각 파편에 대응하는 파티클의 순서(Particle ID)대로 파티클이 위치한 곳으로

파편이 나타날 수 있도록 대체시킨다. 그러면 온전한 형태의 구조물을 다시 형성한다. MEL 소스 코드는 다음과 같다.

```
// move shards to origin axis, and then freeze transfomation
for ($i=0; $i<$numObj; $i++) {
        select -r $obj[$i];
        move -rpr 0 0 0 ;
        makeIdentity -apply true -t 1 -r 1 -s 1 -n 0;
}

// make debrisParticle Instancer (Ojbect Index: Particle ID)
string $inst;

for ($i=0; $i<$numObj; $i++) {
        $inst += " -object " + $obj[$i];
}

string $pTailOption = "-cycle None -cycleStep 1 -cycleStepUnits Frames
    -levelOfDetail Geometry -rotationUnits Degrees -rotationOrder XYZ
    -position worldPosition -objectIndex particleId -age age";
string $pName = "debrisParticleShape1";
string $makeInstancer = "particleInstancer" + " -addObject " + $inst + $pTailOption
    + " " +  $pName;
eval($makeInstancer);

for ($i=0; $i<$numObj; $i++)  {
        select -r $obj[$i];
        HideSelectedObjects;
        select -cl;
}
```

3.4 각 파편이 대체된 파티클에 회전값 생성

회전 변수를 만들어서 x, y, z 각 축에 대해서 임의의 초기값을 설정하고 애니메이션되는 매 frame 마다 파티클의 회전속성에 더해줌으로써 튀어나가는 파편을 회전시킬 수 있는데, 다음과 같이 MEL 소스 코드를 적용하여 생성할 수 있다.

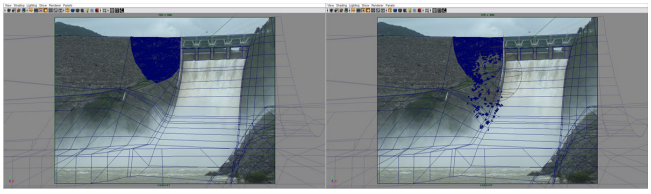
```
addAttr -ln "rotateRate"  -dt vectorArray  debrisParticleShape1;
setAttr -e-keyable true debrisParticleShape1.rotRate;
addAttr -ln "rotatePP"  -dt vectorArray  debrisParticleShape1;
setAttr -e-keyable true debrisParticleShape1.rotPP;

dynExpression -s
    "debrisParticleShape1.mass = rand(1, 1.2);\r\n\r\n
    debrisParticleShape1.rotateRate = <<rand(-5,5),rand(-1,1),rand(-.5,5)>>*2;"
    -c debrisParticleShape1;
dynExpression -s "rotatePP += rotateRate/mass;" -rbd debrisParticleShape1;
particleInstancer -e -name instancer1 -rotation rotatePP debrisParticleShape1;
```

4. 구현 결과

(그림 3)은 KBS 춘천총국 개국 63주년 기념 특별 다큐멘터리로 “소양강댐, 29억톤과의 휴전선”이라는 제목의 방송 프로그램을 제작한 일부 화면을 보여주고 있으며, 2007년 12월 14일 KBS 1TV를 통해 방영되었다. 소양강댐이 붕괴할 수 있다는 가정하에 붕괴되는 장면을 제작하기 위해서 본 논문에서 제시한 방법을 실제 방송 프로그램의 영상 특수효과 제작에 사용하였다. 파티클 시뮬레이션에서는 파편 끼리 충돌은 고려하지 않고 시뮬레이션되므로 각각의 파티클에 대체된 파편이 근접한 거리에 있을 경우 서로 뚫고 지나간다. 그러나 리지드 바디 시뮬레이션에 비해서 빠른 시간 안에 효율적으로 다량의 파편 움직임을 제어할 수 있었다.

<표 1>은 Intel Xeon 2.66GHz CPU, 4GB RAM, NVIDIA Quadro FX 3700 그래픽카드를 갖춘 Windows XP Pro OS의 PC에서 리지드 바디 시뮬레이션과 대체된 파티클 시뮬레이션의 소요시간을 비교한 것이다. 150 frame 범위 동안 피라미드 형태의 물체를 100조각으로 분해하여 사방으로 날아가는 테스트를 하였을 경우 대체된 파티클 시뮬레이션은 3.3초로 리지드 바디 시뮬레이션의 8.6초보다 2.6배 빨랐다. 소양강댐의 붕괴 부분은 280조각으로 분해하였는데 150 frame 범위 동안에 대체된 파티클 시뮬레이션은 12.6초 정도의 시간이 소요되었지만 리지드 바디 시뮬레이션은 근접한 물체 끼리 서로 관통하는 현상인 Interpenetration 연산 오류를 발생시킴으로써 연산 불능상태가 되었다.



(a) 다량의 파편제어 시뮬레이션이 적용된 Maya 화면

(b) 완성된 영상의 정지 화면

(그림 3) KBS 1TV 특별다큐 “소양강댐, 29억톤과의 휴전선”, 2007

구분	Rigid Body	Replaced Particle
100조각	8.6	3.3
소양강댐 붕괴 (280조각)	Interpenetration 연산 오류 발생	12.6

<표 1> 시뮬레이션 소요시간 비교 (150 frame 범위, 단위: 초)

5. 결론 및 향후과제

다량의 복잡한 파편의 움직임을 제어하기에는 리지드 바디 시뮬레이션은 과도한 계산량과 소요시간으로 인해서 그 효용성이 상당히 떨어진다. 즉, 물체간 충돌 효과 계산 시 상당한 시간이 소요되며 겹친 부분에서는 오류가 발생하여 시뮬레이션 자체가 중단되는 등의 문제가 발생한다. 이에 반해 파티클 시뮬레이션은 입자간의 충돌 계산은 생략한 채 입자의 위치 값만을 계산하므로 리지드 바디 시뮬레이션보다 상대적으로 계산량과 소요시간을 줄일 수 있다. 따라서 각 파편의 중심점에 파티클을 생성하고 파티클에 대응하는 파편을 순서대로 대체시켜 온전한 물체의 형태대로 재배치 후 파편의 움직임을 효과적으로 제어할 수 있다.

향후과제로 파티클로 대체된 물체의 크기만큼 파티클에 부피를 설정하고 파티클 간에 충돌효과를 적용하여 다량의 파편들이 상호 작용할 수 있는 방법과 2차 파편의 생성 및 제어할 수 있는 방법을 모색하고자 한다.

참고문헌

[1] 민성환, 김창헌, “컴퓨터 게임을 위한 실시간 Timewarp Rigid Body 시뮬레이션”, 2001년도 한국정보과학회 가을 학술발표논문집 Vol. 28. No. 2

[2] Victor J. Milenkovic, “Position-Based Physics : Simulating the Motion of Maqny Highly Interacting Spheres and Polyhedra”, SIGGRAPH 96 Conference Proceedings, pp. 129-136, August 1996

[3] David C. Brogan, Ronald A. Metoyer, and Jessica K. Hodgins, “Dynamically Simulated Characters in Virtual Environments”, IEEE Computer Graphics and Applications, 18(5):58-69, September 1998

[4] Brian Miritich, “Timewarp Rigid Body Simulation”, SIGGRAPH 2000 Conference Proceedings

[5] http://www.blastcode.com

[6] David A. D. Gould, “Complete Maya Programming”, pp. 273-455, 2003