물리 엔진에 관한 고찰 : 실시간 물리 기술을 중심으로※

하유종°, 박경주*
중앙대학교 첨단영상대학원 영상학과°*
freecafe@korea.com, kjpark@cau.ac.kr

A study on game physics engine focused on real time physics

Youjong Ha, Kyoungju Park*
Dept. of Image Eng., Graduate School of Advanced Imaging Science Multimedia & Film, Chung-Ang University

요 약

본 연구는 게임 물리 엔진을 실시간 물리 기술의 관점으로 고찰한다. 실시간 물리 기술이란 물리 시뮬레이션 기술을 게임에 적용하기 위해서 간략화 하는 기술을 말한다. 조사 대상으로 상용 물리 엔진인 Havok Physics SDK와 NVIDIA PhysX SDK를 선택하였고, 오픈 소스기반 물리 엔진인 ODE와 Bullet을 선택하였다. 그 결과 물리 엔진은 강체 역학, 변형체 시뮬레이션, 유체 시뮬레이션을 구현하고 있었고, 실시간 시뮬레이션을 위해서 수식의 간략화, 충돌 처리의 효율성 재고 등 소프트웨어 측면의 기술과 멀티 코어 CPU의 이용, PPU, GPU 활용 등 병렬처리 하드웨어 기술을 사용하고 있었다.

ABSTRACT

This paper analyzes the four game physics engines in terms of real time techniques. Real time physics is the technology that simplifies the physics-based simulation to apply for the real time applications such as game. Our study includes two commercial physics engines, Havok's Physics SDK and NVIDIA's PhysX SDK, and two open source projects, Open Dynamics Engine and Bullet physics engine. As a result, most of them covers rigid body dynamics and some include either deformable body simulation or fluids simulation, or both. For real time simulation, they adopt the simplified numerical methods, the effective in collision detection/response, and also use the parallel processing hardwares, i.e., multi core CPU, physics processing unit(PPU), or graphics processing unit(GPU).

Keyword : physics engine, game physics, physics-based animation

접수일자 : 2009년 06월 29일
일차수정 : 2009년 09월 14일
심사완료 : 2009년 09월 25일
※ 이 논문은 2009년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(No. 2009-0054429, 2009-0074861).

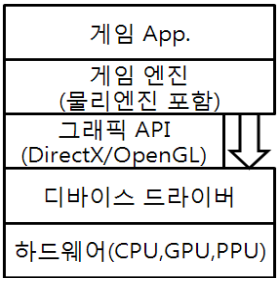
1. 서 론

근래에 게임 제작사들 사이에 물리 엔진에 관한 관심이 고조되고 있고, 실시간감이 필수불가결한 요소인 게임에서도 계산량이 상대적으로 많은 물리엔진을 차용하여 게임을 제작하고 홍보하고 있다. 게임 캐릭터와 주변 환경의 시뮬레이션에서 물리적인 시뮬레이션을 수행하여 게임의 실감성을 높이고, 사용자 만족도를 증대시키고자 하고 있다. 예를 들어, 물체의 충돌을 표현할 때 애니메이션 기법을 이용하여 프레임 별로 물체의 움직임을 묘사하는 대신, 힘과 질량의 관계식으로부터 물체의 이동 방향과 위치를 계산해서 움직임을 표현한다. 즉, 물리 엔진을 이용함으로써 보다 자연스러운 움직임을 표현하는데 그 목적이 있다.

기획한 콘텐츠를 제작할 때 필요한 물리 시뮬레이션 기술들이 무엇인지 파악하고, 그 기술들만을 선택적으로 구현한 가벼운 물리엔진을 채택하여 적용 개발하는 것이 콘텐츠를 가볍게 할 수 있는 필수 불가결한 요소라 하겠다. 또한, 온라인 게임, 네트워크 게임, 단말용 게임 등 플랫폼의 스펙 상 개인용 컴퓨터보다 알고리즘의 컴플렉서티가 낮아야 할 당위성이 있는 대부분의 경우에는 적절한 물리 기술과 적합한 물리 엔진의 선택, 그리고 맞춤형 엔진 적용 등이 게임 콘텐츠의 경쟁력에 영향을 주게 마련이다. 따라서 본 논문에서는 물리엔진을 소개하고, 대중화된 물리엔진의 기능과 실행시간 특성 등을 비교 분석하여 게임 개발자들의 물리엔진 선택을 돕고자 한다.

물리엔진의 개요

물리 엔진은 게임 시나리오에 독립적이다. 즉, 여러 게임에 적용될 수 있는 공통의 물리 연산 코드이다[1]. 코드 최적화를 통한 계산 효율성이 특히 중요하기 때문에 하드웨어 의존성이 높다. 따라서 게임 엔진을 제작하는 회사조차 물리 엔진은 타사 제품을 도입하는 경우가 많다. [표 1]과 [표 2]는 게임 엔진과 적용된 물리 엔진을 기술한다.



[그림 1] 블록 다이어그램 : 물리엔진은 게임과 그래픽 API 사이의 미들웨어이다. 그러나, 일부 물리 엔진은 디바이스 드라이버를 직접 호출한다.

물리엔진은 게임 엔진의 일부로 통합되어 있거나 별도의 라이브러리로 구성된다. [그림 1]과 같이 게임 애플리케이션과 그래픽 API(예 : OpenGL, DirectX)사이의 미들웨어이다. 물리 엔진은 기본적으로 역학 시뮬레이션 기능을 제공한다. 게임에서는 전통적으로 강체 역학이 활용 되지만 최근 변형체(deformable body)와 유체로 응용이 확장되고 있다. 일반적으로 물리 시뮬레이션은 계산량을 많이 요구하기 때문에 GPU와 같은 병렬처리 하드웨어의 활용이 필수적이다. 플레이스테이션이나 X-Box용 물리 엔진은 셀 프로세서를 활용하며, PC 기반 물리 엔진들은 멀티 코어 CPU를 이용하거나, PPU, 또는 GPU를 이용한다.

게임에서 물리 엔진을 사용하는 이유는 실시간 물리 시뮬레이션을 수행하기 위해서이다. 실시간이 아닌 일반적인 시뮬레이션은 통상 오프라인 시뮬레이션이라 한다. 여기서 실시간이란 인간이 화면으로부터 상호작용감을 느끼는 속도인 15-72fps 범위를 말하며[2], 관례적으로 게임 화면의 갱신 속도인 30fps를 의미한다. 따라서 실시간 물리 시뮬레이션을 수행하기 위해서는 오프라인과는 다른 접근 방법이 요구된다. 이것을 실시간 물리 기술이라 정의한다. 즉, 게임 물리 엔진의 목표는 시뮬레이션의 정확성이 아닌 시각적으로 효과적인 물리 기반 애니메이션을 추구한다.

2장은 연구 대상 선정 및 방법, 3장은 각 물리 엔진의 특성 및 비교, 4장은 결론, 참고문헌 순서로 기술한다.

2. 연구 대상 선정 및 방법

게임 엔진과 물리 엔진의 관계를 파악하기 위해 먼저 인터넷 사이트[3,4]를 참조하여 2007년 2월부터 2009년 2월 사이에 출시된 게임 및 사용 게임 엔진들을 조사하였고, 해당 게임 엔진이 어떤 물리 엔진을 사용하고 있는지 조사하였다. 또 오픈 소스 프로젝트를 대상으로 많이 사용되는 게임 엔진과 그 호환 물리 엔진을 조사하였다. 결과는 [표 1,2]와 같다. 단, 오픈 소스의 경우에는 개발자가 임의로 게임 엔진과 물리 엔진을 조합해서 사용할 수 있기 때문에 호환성에 대한 책임은 개발자에게 있다.

[표 1] 상용 게임 엔진과 물리 엔진

게임 엔진	물리 엔진
Havok	Havok
Unreal	PhysX
Gamebryo*	PhysX**
CryEngine	자체
Source	Havok
JupiterEx	Havok

* Gamebryo는 순수 렌더링 엔진임
** 차기 버전에서 지원 예정

[표 2] 오픈 소스 기반 게임 엔진과 물리 엔진

게임 엔진	물리 엔진
Ogre	ODE, Bullet
Nebular	ODE
Irrlicht	Bullet
TGE(Torque)	자체 (기초적인 수준)

기타 오픈 소스 물리 엔진으로 Chipmunk, Tokamak 등이 있다.

상용 게임 엔진은 크라이텍의 CryEngine을 제외하고, 물리 엔진으로 인텔의 Havok과 엔비디아의 PhysX 2개 제품을 채택하고 있다. 오픈 소스 게임 엔진은 ODE와 Bullet을 채택하고 있다. 참고로, Havok과 PhysX와 Bullet은 게임 콘솔인 PS3와 X-Box 플랫폼을 위한 별도의 버전을 제공한다. 특히 Bullet은 소니 엔터테인먼트 유럽이 개발한 PS3용 게임 엔진 Phyre의 물리 엔진으로 채택되었다. 본 논문은 [표 1]과 [표 2]에 근거하여 ODE, Bullet, Hovok, PhysX 4개 제품을 중심으로 물리 엔진에 관해 고찰한다.

3. 각 물리 엔진 특성 분석 및 비교

물리 엔진은 강체, 변형체, 유체, 파괴 시뮬레이션 기능을 제공한다. 강체 역학은 물체의 회전이나 이동처럼 물체의 형태가 변하지 않는 운동을 다룬다. 역학 부분과 충돌 부분으로 나누어 구현되는 것이 일반적이다. 충돌 처리에 있어서 현재 대부분의 물리 엔진은 강체와 강체, 강체와 변형체, 강체와 유체의 충돌 처리는 만족스런 수준에 있으나, 이외의 상호작용에는 한계를 드러낸다.

변형체는 연체라고도 하며 옷감과 같이 형태의 변형이 가능하다. 오프라인 시뮬레이션에서는 유한요소법이 많이 사용되지만 물리 엔진에서는 질량-스프링 시스템이 사용된다.

물, 불, 연기와 같은 유체 시뮬레이션은 특히 컴퓨터 계산량을 많이 요구하기 때문에 게임에서는 전통적으로 프로시저 방식이 사용되었다. 프로시저 방식이란 물리 법칙이 아닌 화면에 표시되는 결과에 초점을 맞춘 애니메이션 기법을 말한다. 예를 들면, 텍스처와 팔레트를 이용한 불, Perlin 노이즈 함수를 이용한 구름, 빛의 굴절과 레이 트레이싱을 이용한 물, 환경 맵을 이용한 반사 등을 포함한다[5]. 하이트 필드를 이용한 기술도 프로시저 방식에 해당하며, 주로 얕은 물 시뮬레이션에 적용된다.

본 논문에서는 비압축 Navier-Stokes 방정식과 같은 유체 방정식을 사용하지 않는 프로시저 방식은 유체 시뮬레이션에서 제외한다.

파괴 시뮬레이션은 재료의 성질을 표현한다. 즉, 나무나 금속에 충격을 가했을 때 물성에 의존적인 변형과 파괴를 시뮬레이션 한다. Pixelux사는 버클리 대학의 James O'Brien의 지원을 받아 실시간 유한 요소 기술을 구현하고 있다[6].

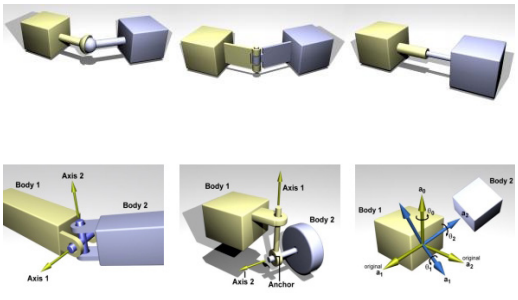
2.1 Open Dynamics Engine (ODE)

ODE는 Russel Smith에 의해 개발되었다[7]. 매뉴얼에 의하면 기술적 배경은 Baraff의 시그라프 강연 자료[8]이며, 운동 방정식은 Trinkle-Stewart의 논문[9]과 Anitescu-Potra의 논문[10]을 기반으로 관절이 있는 강체 시뮬레이션을 구현하고 있다. ODE에 대한 이론적 이해나 활용에 관한 레퍼런스로는 [11,12]가 있다.

전체적으로 ODE는 역학 시뮬레이션 엔진과 충돌 검사 엔진 2개 부분으로 구현 되어 있다. 개별적인 강체를 시뮬레이션 하는 대신, 강체들을 그룹 단위로 처리한다.

예를 들면, 각 그룹은 온/오프 상태를 갖는다. 오프 상태인 그룹은 시뮬레이션에서 제외된다. 온 상태인 그룹과 오프 상태인 그룹이 충돌하면 오프 상태인 그룹은 온 상태가 되면서 시뮬레이션에 포함된다. 또, 온 상태인 그룹이라도 지속적으로 상태 변화가 없다면, 자동으로 오프 상태가 된다.

ODE는 1차 임플리싯 오일러 적분을 사용한다. 임플리싯 오일러 적분은 빠르고 안정적이지만 다소 부정확하다. 강체에 작용한 힘은 적분 스텝 동안 누적 되고, 다음 스텝에서 합력이 물체에 작용한다. 이 값은 스텝이 종료될 때마다 0으로 재설정된다. 강체에 힘이 작용하면 직선운동과 회전운동이 발생한다. 강체의 운동 방향과 크기를 제한하는 것을 구속 조건이라 하며, [그림 2]와 같이 조인트 형태로 모형화 된다.



[그림 2] 조인트의 종류 (그림 출처: [7])

시뮬레이션이 진행되면서 오차로 인해 강체와 조인트가 서로 분리 되는 경우가 있다. 이를 조인트 오류라 한다. 조인트 오류를 수정하기 위해서 스텝마다 강체들을 다시 조인트의 축 방향으로 정렬시키기 위한 추가적인 힘이 요구 되는데, 오류 감소 파라미터인 ERP를 사용하여 오차를 줄인다. 구속력 혼합 파라미터인 CFM을 사용하여 조인트의 연성을 조절한다. 정방형 대각행렬이다. 0이면 구속력이 엄격하게 적용되며, 양의 값이 있으면 어느 정도 위반을 허용한다. ERP와 CFM를 이용하면 강체를 이용한 질량-스프링 시스템의 구현도 가능하다. ERP와 CFM은 다음 수식에 의해 계산된다.

$$ERP=hk_p/(hk_p+k_d)$$
$$CFM=1/(hk_p+k_d)$$

여기서 h는 스텝 크기이고, kp는 스프링 상수, kd는 댐핑 상수이다.

충돌 컬링(collision culling)을 이용하면 효율적인 충돌 검사가 가능하다. N개의 개체에 대해서 충돌 검사를 수행하려면 O(N2)회의 반복을 필요로 한다. O(N2) 알고리즘은 비효율적이기 때문에 성능을 높이기 위해서는 대상(N)을 축소해야 한다. 충돌 가능성이 있는 개체를 사전에 선별하는 기법을 충돌 컬링이라 하며, 공간 객체가 사용된다.

예를 들어, 간단한 박스 로봇의 충돌 검사는 다음과 같이 구현 가능하다. 먼저 초기화 부분에서 공간 객체(로봇)를 생성한다. 이어서 공간 객체가 삭제 될 때 삽입된 객체(박스)를 함께 삭제할 것인지 여부를 결정한다. 이제 박스의 수 만큼 삽입 함수를 호출하여 공간 객체에 박스를 추가한다. 마지막으로 시뮬레이션 루프 안에서 공간에 대한 충돌 검사 함수를 호출한다. 이 함수 내부에서 충돌 컬링 작업이 수행된다. 충돌 가능성이 있는 개체 쌍이 검출되면, 사용자가 지정한 콜백 함수가 호출된다. 콜백 함수에서 개체에 대한 충돌 검사 함수를 호출하는 방식으로 충돌 검사를 구현한다.

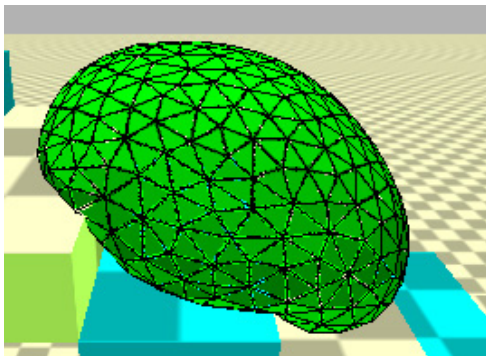
3.2 Bullet

Bullet은 Erwin Coumans에 의해서 개발되었다[13]. 강체 역학과 충돌, 변형체(옷감과 로프)를 구현하고 있으며, 오픈 소스 프로젝트임에도 불구하고 PC, Linux, OSX, PS3, X-Box360, wii, iPhone 등 다양한 플랫폼을 지원한다. 그러나, IBM Cell SDK, 엔비디아 CUDA, OpenCL 등 외부 개발 도구를 사용하고 있다. 이론적 배경은 [14,15,16]이다.

특징적인 실시간 기술로는 먼저 엄격한 고정 스텝의 적용을 들 수 있다. 스텝 크기가 다른 경우에는 자동적으로 보간값이 적용된다. 충돌 처리는 GJK 거리 계산 알고리즘과 EPA 침투 거리 계산 알고리즘을 병용하여 구현되어 있다. 먼저 넓은 범위에 대해서 AABB에 기반한 BVH 트리 알고리즘을 사용하여 빠른 충돌 검사를 수행하고, 그 다음 좁은 범위에 대해서 최적의 알고리즘을 적용한다. 충돌 테이블에 복수의 충돌 검사 알고리즘이 등록되어 있으며, 충돌체의 타입에 따라서 최적의 알고리즘이 선택된다. 사용자 알고리즘을 추가할 수 있으며, 사용자 마스크 비트를 사용하여 충돌 검사를 필터링 할 수 있다.

변형체는 스프링 시스템이 아닌 강체 역학에 기반하여 구현되어 있다. 충돌 객체에서 상속 받은 연체 객체는 [그림 3]과 같이 삼각형 메쉬를 사용해서 생성된다. 각 삼각형의 정점과 평면을 이용해서 충돌 검사가 수행된다. 따라서 삼각형의 테셀레이션 밀도가 높아야 충돌의 누락을 방지할 수 있다. 또, 계산 효율성을 위해 변형체는 자동으로 볼록 변형체 클러스터로 분할된다.

이 밖에 차량 및 캐릭터 컨트롤러를 지원하며, 원뿔 비틀림 구속조건을 이용한 봉재인형(ragdoll) 시뮬레이션을 제공한다. 또한 데이터 호환을 위해서 COLLADA 스펙을 지원하기 때문에 마야, 블렌더 등에서 플러그인을 사용하여 데이터를 임포트/익스포트 할 수 있다.

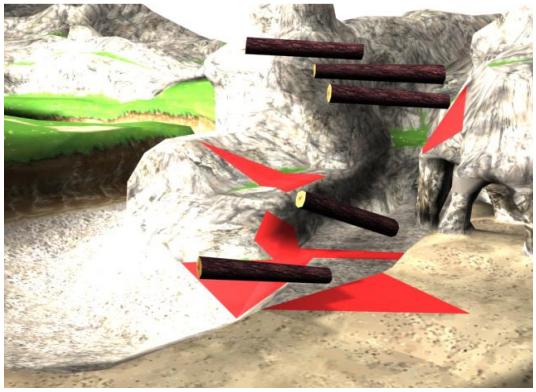


[그림 3] Bullet의 'Softbody Pressure' 예제: Bullet은 질량-스프링 시스템이 아닌 강체 역학을 이용하여 변형체를 구현한다.

3.3 Havok Physics SDK

가장 유명한 게임 엔진의 하나인 Havok 엔진의 물리 엔진으로서 SDK 형태로 배포된다. 멀티 코어 CPU를 사용한 병렬 처리 방식이지만, 최근 GPU를 이용한 옷감 시뮬레이션을 시연하였다. Havok Physics SDK는 강체 역학, 충돌, 질량-스프링 시스템을 이용한 변형체 시뮬레이션을 지원하며, 봉재인형, 차량, 캐릭터 컨트롤러를 지원한다. Havok 엔진은 물리 엔진뿐만 아니라, AI Pathfinding SDK, Animation SDK, Behavior SDK, Cloth SDK, Destruction SDK 등 기능별로 모듈화 된 SDK 형태로 구성되어 있다. 이 중 Physics SDK와 Animation SDK는 Havok 홈페이지에서 무료로 다운로드 받을 수 있다[17].

충돌 검사에서 특징적인 것은 대형 충돌 메쉬를 위한 MOPP(memory optimized partial polytope) 기술이다. MOPP는 적은 메모리를 사용하여 정적 지형 메쉬를 저장할 수 있도록 설계 되었으며, 움직이는 물체와 지형에 대한 빠른 충돌 검사를 수행한다. [그림 4]와 같이 물체와 지형 사이의 전체 충돌 집합을 특정 삼각형들과 물체의 충돌 가능 집합으로 축소시킨다.



[그림 4] Midphase MOPP : MOPP는 통나무와 충돌 가능한 지형 삼각형들을 식별한다. (그림 출처: [17])

충돌 검사시 phantom 객체를 사용할 수 있다. Phantom은 시뮬레이션 공간에는 포함되어 있으나 외력에 반응하지 않는 객체이다. 그러나, 내부적으로 자신과 충돌한 물체들의 목록을 유지한다. 주용도는 시뮬레이션 공간에 어떤 물체가 새로 들어오거나 나가는 것의 감지이다. 예를 들면, 게임에서 캐릭터가 phantom 객체와 충돌하면 NPC의 움직임을 바꾸는 시나리오를 생각할 수 있다.

참고로, PhysX의 kinematic actor와는 다른 개념이다. 다른 위치로 이동은 가능하지만 외력에 의해 움직일 수 없다는 점은 같지만 kinematic actor는 다른 오브젝트를 움직일 수 있다. 즉, kinematic actor는 무한 질량을 가진 가상 물체로 생각할 수 있으며 캐릭터와 물체의 움직임을 직접 제어하고자 할 때 사용 가능하다.

변형체는 Animation SDK의 캐릭터 스키닝과 모핑 기능을 이용해서 구현 가능하며, 유체 시뮬레이션은 지원하지 않는다. 단지 조인트와 스프링을 이용한 프로시저 방식으로 유체의 상호 작용을 구현하고 있다. 파괴 시뮬레이션은 Destruction SDK에 구현 되어 있다.

3.4 PhysX

엔비디아의 물리 엔진이다[18]. PhysX는 PPU 기반으로 개발되었으나 현재는 GPU 방식(자사의 CUDA 기술)으로 변환 하는 단계이다. GPU PhysX라고 한다. 연체, 유체, 변형체는 완료 되었고, 강체는 진행 중이다. PhysX는 강체, 유체, 옷감, 변형체, 역장 시뮬레이션을 구현하고 있다. 충실한 매뉴얼 및 풍부한 예제를 제공하고 있으나 최대 파티클 수, 최대 폴리곤 수 등 제약 사항이 많다. SDK 바이너리는 엔비디아 홈 페이지에서 무료로 다운로드 받을 수 있으며[18], 소스 코드에 대한 라이선스는 유료이지만, PS3 버전의 경우에는 소니 PSN (PlayStation Network)에 개발자로 가입 되어 있을 경우 무료이다. 참고로, PS3, PSN 버전은 소니에 의해서 관리 되고 있으며, wii 버전은 기능 제한판이다.

충돌 검사에 있어 특징적인 것은 기본 충돌체 이외에 볼록 메쉬, 삼각형 메쉬, 하이트 필드 메쉬 등 다양한 메쉬 충돌체를 지원한다. 메쉬 충돌체를 생성하기 위해서는 쿠킹 작업이 필요하다. 쿠킹은 충돌 검사를 위해서 메쉬 데이터로부터 시뮬레이션을 위한 별도의 자료구조를 생성하는 작업을 말한다. 지형과 같은 대규모 정적 메쉬를 하드웨어로 처리하려면 메쉬 페이징 작업이 필수이다. 그밖에는 강체 역학 시뮬레이션 프로세스와 유사하다.

유체 시뮬레이션은 입자법의 하나인 SPH[19]로 구현되어 있다. 점성이 있는 유체와 없는 유체 두 가지를 지원한다. 입자들의 상호작용 여부를 설정할 수 있으며, 이벤트 기능을 제공한다. 유체에 어떤 이벤트가 발생했을 때, 사용자가 지정한 콜백 함수를 호출하는 기능이다. 현재는 콜백 인터페이스로서 onEmitterEvent와 onEvent 두 개의 함수를 제공한다. 그러나, 유체와 유체의 상호작용은 지원하지 않는다.

변형체 시뮬레이션은 옷감과 연체를 지원한다. 모두 질량-스프링 방식으로 구현되어 있다. 차이점은 연체의 경우 추가로 사면체 메쉬를 생성해야 한다. 즉, 시뮬레이션은 사면체 메쉬를 이용해 수행하고, 결과를 원래의 삼각형 메쉬에 적용하는 방식으로 시뮬레이션이 수행된다. 옷감은 직접 삼각형 메쉬를 이용하여 시뮬레이션이 수행되며, 설명

플래그의 설정 값에 따라 찢어지는 옷감, 금속 옷감(예:드럼통), 압력 옷감(예:풍선) 등 특별한 종류의 옷감을 만들 수 있다. 옷감과 옷감, 옷감과 연체 사이의 상호작용은 지원하지 않는다.

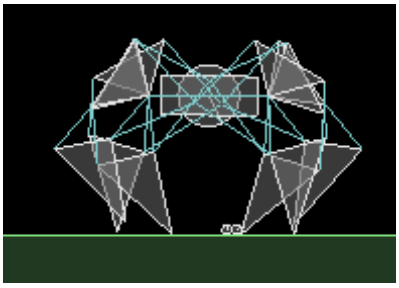
역장 시뮬레이션은 시뮬레이션 영역에 포함된 모든 객체에 작용하는 외력을 모형화 한다. 예를 들면 토네이도 등을 표현할 때 사용할 수 있다.

파괴 시뮬레이션은 아펙스(Apex) 기술을 이용해서 지원한다[20].

이밖에 레이 캐스트, 스윕 테스트(sweep test), 리포트, 캐릭터 컨트롤러, 차량 시뮬레이션을 지원한다.

3.5 아이폰용 물리 엔진

아이폰용 게임 개발은 기본적으로 매킨토시 환경에서 iPhone SDK를 사용하여 수행된다. 아이폰 개발자 등록(등록비: 개인 $99, 기업 $299)을 하면 누구나 자신의 프로그램을 앱 스토어(App Store)에서 판매할 수 있다. 아이폰용 게임 엔진으로는 iTorque, unity3D, Shiva 등이 있으며 물리 엔진으로는 Octave engine casual, Chipmunk, Box2d, Bullet 등이 있다. 모바일 환경에서는 성능상의 이유로 2d 엔진도 다수 활용된다[그림5].



[그림 5] Jansen's walker : box2d 예제 (PC에서 컴파일)

3.6 물리 엔진 간 비교

각 물리 엔진이 제공하는 기능은 부록의 [표 3]과 같다. ODE는 기본적인 강체 역학 시뮬레이션과 충돌 검사를 지원한다. Bullet은 오픈 소스임에도 불구하고 다양한 플랫폼을 지원한다. 상용 프로그램인 Havok과 PhysX는 기능면에서 큰 차이는 없으나 PhysX에는 SPH를 이용한 유체 시뮬레이션이 구현되어 있다.

충돌 검사 알고리즘의 공통적 특징은 먼저 비교적 간단한 알고리즘을 사용하여 광역 테스트를 수행하고, 충돌이 검출 되면 더욱 정교한 알고리즘을 적용하여 지역 테스트를 수행하는 단계적 접근 전략이다. ODE는 고전적인 알고리즘만을 제공하는 대신 다른 알고리즘을 쉽게 적용할 수 있도록 개방형 구조를 취하고 있으며, 상용 물리 엔진들은 연속 충돌 검사(CCD)와 같은 진보된 알고리즘을 제공한다.

변형체 시뮬레이션에서 Havok과 PhysX는 질량-스프링 시스템을 이용하며, Bullet은 강체 역학을 이용한다.

캐릭터 컨트롤러는 캡슐과 구체 등 간단한 기하 구조를 이용하여 캐릭터 컨트롤 및 상호작용을 구현한 것이다. PhysX의 캐릭터 컨트롤러는 AABB와 캡슐을 사용하고 있으며, Havok은 볼록 정점 구조(convex vertices shape)을 사용하여 캐릭터의 자세 및 움직임을 제어한다.

차량 시뮬레이션에 있어서 자동차는 강체들의 집합이 아닌 한 개의 객체로 간주되어 역학 시뮬레이션이 수행된다. Havok의 차량 시스템은 항공기와 같은 6축 좌표계를 사용한다. 차량의 운동뿐만 아니라 타이어 자국, 사운드, 카메라 기능을 제공한다. PhysX와 Bullet에는 레이 캐스트를 이용한 바퀴가 구현되어 있다.

봉재인형은 역운동학을 이용하여 구현된다. Havok은 봉재인형 강체 컨트롤러를 제공하고 있으며, PhysX와 Bullet은 6d 조인트와 원뿔 조인트를 지원한다. 참고로 ODE에도 6d 조인트가 있다.

물리 엔진을 사용할 경우 추가적으로 소요되는

공간은 500KB(ODE, DLL version)에서 110MB(Havok, multi-threaded DLL version)까지 편차가 크다. 시뮬레이션 속도 측면에서 대부분의 물리 엔진들이 적응 스텝이 아닌 고정 스텝으로 시뮬레이션을 수행한다. 따라서 과도한 CPU 부하로 인해 필요한 대역을 확보하지 못하면 시뮬레이션이 실패할 수 있다. 이 때 물리 엔진은 시뮬레이션을 안정적으로 재시작(또는 종료)해야 한다.

벤치마킹 테스트와 같은 성능 평가는 본 논문이 다루는 범위를 벗어나기 때문에 포함하지 않았으나, 레퍼런스로서 관련 논문과 구현 프로그램이 있다[21,22].

4. 결 론

본 논문에서는 물리 엔진 4개를 대상으로 실시간 물리 시뮬레이션 기술에 대해서 고찰하였다. 물리 엔진에서는 시뮬레이션의 정확성 보다는 빠르고 안정적인 적분 함수의 선택, 시뮬레이션 대상의 그룹화, 충돌 처리의 효율성 재고 등 소프트웨어 측면의 기술과 멀티 코어 CPU의 이용(Havok), PPU(PhysX), GPU 등 병렬 처리 하드웨어를 활용하는 기술이 사용된다. 렌더링 엔진은 GPU 사용이 필수이지만, 현재의 물리엔진은 대부분 CPU에 기반하고 있다. GPU를 이용한 방식이 성능상 유리한 점이 많지만 CUDA와 OpenCL 사이의 표준화 문제가 아직 해결되지 않은 이슈로 남아 있는 상태이다. 그러나, Bullet은 실험적으로 CUDA와 OpenCL을 적용하였으며, Havok과 PhysX도 GPU 버전으로의 마이그레이션 단계에 있다.

물리 엔진의 선택에 있어서 고려할 점은 용도, 게임 엔진과의 호환성, 비용이다. 또, 게임 기획과 게임 엔진의 기능이 상이할 경우에는 게임 엔진 자체를 수정해야 하는 필요성도 제기되기 때문에 소스 코드에 대한 접근성도 고려해야 한다.

참고문헌

[1] Ian Millington, "Game Physics Engine Development", Morgan Kaufmann, pp.3, 2007

[2] Akenine-Moller,T., Haines,E., "REAL-TIME RENDERING 2nd edition", AK Press, 2002
신병석, 오경수 역, "REAL-TIME RENDERING 2판", 정보문화사, pp.20, 2003

[3] http://www.gpgstudy.com

[4] http://www.gamespot.co.kr

[5] Mason McCuskey, "Special Effects Game Programming With DirectX", Premier Publishing, 2001, SPHERE 역, --, 민커뮤니케이션, 2002

[6] http://www.pixelux.com

[7] http://www.ode.org

[8] A.Witkin and D.Baraff, "Physically Based Modelling", ACM SIGGRAPH 2001 Course Notes, 2001.

[9] D.Stwart and J.Tringkle, "An implicit time-stepping scheme for rigid body dynamics with coulomb friction", Proceedings of the 2000 IEEE International Conference on Robotics & Automation San Francisco, CA April 2000

[10] Mihai Anitescu and Florian A. Potra, "A time- stepping method for stiff multibody dynamics with contact and friction", Int. J. Numer. Meth. Engng 2002; 55:753 - 784

[11] Roman Durikovic and Katsuhiro Numata, "Human Hand Model based on Rigid Body Dynamics", Proceedings of the Information Visualisation, Eighth International Conference, pp. 853-857, 2004.

[12] Kosei Demura, "Robot Simulation-Robot programming with Open Dynamics Engine" , Morikita Publishing Co. Ltd., Tokyo, 2007

[13] http://www.bulletphysics.org

[14] Gino van den Bergen, "Collision Detection in Interactive 3D Environments", Morgan Kaufmann, 2003

[15] Kenny Erleben , Jon Sporring, Knud Henriksen, Henrik Dohlmann, "Physics Based Animation", Charles River Media, 2005

[16] Christer Ericson, "Real-Time Collision Detection", Morgan Kaufmann, 2005

[17] http://www.havok.com

[18] http://developer.nvidia.com/object/physx.html

[19] Matthias Müller et al. "Particle-Based Fluid Simulation for Interactive Applications", Eurographics/SIGGRAPH Symposium on

Computer Animation, 2003
[20] http://developer.nvidia.com/object/apex.html
[21] Adrian Boeing et al.,"Evaluation of real-time physics simulation systems", December 2007 GRAPHITE '07: Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia, pp.208-288, 2007
[22] http://www.adrianboeing.com/pal/index.html

[표 3] 물리 엔진 기능 비교

기능	ODE	Bullet	Havok	PhysX
버전	0.11	2.74	6.50	2.8.1
강체 역학	o	o	o	o
충돌 검사	o	o	o	o
변형체	x	옷감, 밧줄, 변형가능 물체	스키닝과 모핑 (SIMD 버전과 float 버전을 제공)	옷감, 연체
유체	x	x	x	SPH
캐릭터 컨트롤러	x	o	o	o
차량	x	o	o	o
봉재인형	6d 조인트	6d 조인트	o	6d 조인트
파괴	x	x	Destruction SDK	APEX
데이터 호환성	x	Maya, Dynamica 플러그인, COLLADA 스펙 지원	Max, Maya, XSI 플러그인	Max, Maya 플러그인
지원 플랫폼	PC (Linux, OSX...)	PC, X-Box 360, PS3, wii, Linux, OSX, iPhone	PC, PS2, PS3, PSP, X-Box, X-Box 360, wii, Gamecube	PC, PS3, PSN, X-Box 360, XBLA, wii, wii-ware
GPU	x	CUDA, OpenCL (테스트 목적)	OpenCL (진행중)	CUDA(진행중)
설치 용량	80M*	1.37G*	930M**	220M
동작 속도	고정/가변	60Hz 고정	60Hz 변경가능	60Hz***

* ODE, Bullet은 컴파일시 생성되는 obj까지 포함.
** Havok Physics SDK와 Animation SDK가 함께 설치 됨.
*** timestep과 substep을 이용한 방식 (예: timestep=1/60, substeps=4로 설정하면 1/60초 동안 4번 substep이 실행됨)

하 유 종(Ha, You jong)

1994 성균관대 금속공학과 (공학사)
2007 서울디지털대학교 멀티미디어학부 (게임학사)
2008- 중앙대학교 첨단영상대학원 영상학과(석사과정)

관심분야: 게임, 물리 기반 애니메이션

박 경 주(Park, Kyoung ju)

2004 Ph.D, Dept. of Computer and Information Science. University of Pennsylvania, PA
2004-2005 Research Professor, Rutger University, NJ
2007년-현재 : 중앙대학교 첨단영상대학원 교수

관심분야 : Physics Based Simulation, Computer Graphics