3D 게임엔진을 위한
역학 시뮬레이터 및 개발도구의 설계

김성찬[0] 황요한 김동균 신동규 신동일
세종대학교 컴퓨터공학과
{kimschan[0], xfilen, kd999, shindk, dshin}@gce.sejong.ac.kr

Design of Mechanics Simulator and
Software Development Kit for 3D Game Engine

Seong-chan Kim[0] Yo-han Hwang Dong-kyun Kim Dong-kyoo Shin Dong-il Shin
Dept. of Computer Engineering , Sejong University

요 약

본 논문에서는 3차원 게임 엔진에서 역학엔진의 개발을 지원하는 역학 시뮬레이터 및 개발도구에 대해 설계하였다. 3차원 게임 엔진을 개발하는 과정에서 역학 현상을 실시간으로 테스트하여 동적이며 인터렉티브한 게임 효과와 사용자가 역학 시뮬레이터를 통해 실시간으로 테스트한 환경을 개발도구에서 적합한 API를 생성하여 라이브러리형태로 제공함으로써 개발기간의 단축 및 개발 공정에 관련된 프로세스 처리를 효율적으로 할 수 있도록 하였다.

1. 서 론

90년대 초 중반 이후로 많은 3차원 게임 등장을 하였다. 하지만 초창기의 엔진들은 현실세계에서 일어나는 현상을 무시하고 3차원으로 화면을 그리는 것에만 목적이 있었다. 사용자들은 이런 게임의 내용에 실증을 느꼈고 좀 더 사실적인 내용을 요구하였다. 90년 대 중반 이후에 물리 엔진에 대한 중요성이 게임 개발자들에 의해 대두되기 시작했고 이는 게임에 혁신을 가져왔다. 많은 현실 세계의 물리현상이 게임에 적용이 되었으며 이제 게임에서 보이는 현상은 사용자가 이해할 수 있었다.

본 논문에서는 이러한 물리엔진의 제작을 지원하는 역학시뮬레이터와 개발도구에 대한 설계에 대하여 기술하였다. 2장에서는 물리엔진에 관련된 내용에 대해 기술하고, 3장,4장에서는 역학 시뮬레이터와 개발도구에 대하여 기술한다. 끝으로 5장에서는 결론 및 향후 전망에 대하여 기술한다.

2. 관련 연구

3차원 게임에서 물리엔진에 대한 논의는 1996년 Game Developer지 10월호에 연재된 Chris Hecker의 기사 'Physics, The Next Frontier'[1]에 의해서 주장이 되기 시작했다. 그는 게임 개발자들에게 게임의 각 요소 및 개체를 인식하는 기본적인 개념으로의 물리학의 중요성에 대해서 주장을 했으며 멀지 않은 미래에는 게임을 제작하는데 필요한 요소로 작용할 것이라고 말했다. Chris Hecker의 기사 이후에 많은 개발자들이 게임에서의 역학의 중요성에 대해서 언급을 하였으며 몇 몇 게임 엔진이 이를 바탕으로 제작이 되었다. 물리엔진은 크게 두 부류로 나누어 볼 수 있다. 현실에 가까운 사실성을 보장하는 물리엔진과 엄격한 현실세계의 규칙 대신 좀 더 자유를 허용하는 물리엔진이 있다. 전자의 물리엔진의 예로는 Mathengine에서 개발한 *Karma*[2]와 Havok의 *Havok*[3]이 있으며 후에 *ipion* 엔진[4]이 Havok GDK에 추가되어 BSP, 충돌검사, 충돌역학, 강체역학, 차량역학 등을 처리하며 하프라이프[5], 워크래프트3[6]와 3D 벤치마크의 자동차 데모에서 사용되었다. 또한 Electronic Arts의 2015스튜디오에서 개발한 MOHAA[7]에서 사용된 엔진은 퀘이크3[8] 엔진으로 퀘이크1,2에 비해 물리적인 효과의 표현이 많은 부분 발전이 되어 나타났다. 여기에 Volition에서 개발한 Red Faction[9][10]은 물리엔진과 기하구조의 변형을 통한 현실세계의 물리현상에 대해 사실적으로 접근했다. 후자의 물리엔진의 예로는 캐릭터 개발 엔진과 플레이스테이션2에 사용된 Emotion Engine[11]이 있다. 이 엔진들은 현실적인 환경을 무시하고 게임에서 게임 디자이너에 의해 설계된 환경에서의 현상을 표현하며 과장되거나 혹은 왜곡시킴으로써 그 효과를 극대화시켜내려고 한다. 앞서 말한 두 엔진의 종류는 모두 해외에서 개발이 되었거나 계속 개발중인 추세이며 국내에서의 물리엔진에 대한 개발은 미흡한 실정이다.

이에 본 논문에서는 게임 엔진 개발자들이 좀더 용이하게 게임엔진에 역학 효과를 적용할 수 있도록 역학 시뮬레이터와 개발도구를 설계하였다.

3. 역학 시뮬레이터 및 개발도구의 설계

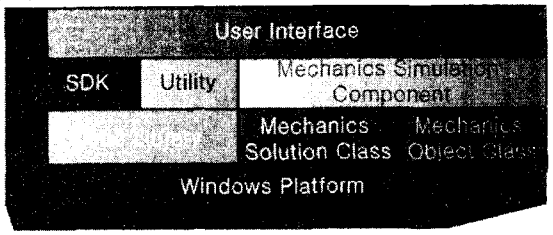
본 논문에서는 실시간 및 역학적인 효과를 기대하는 3D 게임 엔진을 제작할 경우 필요한 역학현상에 대한 가상 실험 시뮬레이터를 제공함으로써 3D 게임 엔진 제작에 편리성을 제공하는 개발도구에 대해 설계하였다.

다음은 전체 시스템의 개요 및 설계 구조이다.

3.1. 역학 시뮬레이터 및 개발도구의 개요

본 논문에서 설계하고자 하는 역학 시뮬레이터의 구조는 [그림1]과 같다. [그림1]에서 시뮬레이터의 구조는 크게 세 부분으로 분류된다. 첫 번째 부분은 역학 시뮬레이터를 이용하는 사용자의 입력을 받아들이고 사용자의 입력에 따라 시뮬레이션된 데이터를 개발도구로 반출시켜주는 역할을 하는 User Interface와 시뮬레이션에 필요한 도구를 제공하는 Utility이다. 두 번째 부분은 역학 현상을 처리하는 컴포넌트 객체의 집합이다. 이 객체의 집합은 하부구조로 각각의 역학 현상에 필요한 Mechanics Solution Class와 Mechanics Object Class를 갖는다. 끝으로 세 번째 부분은 SDK와 Code Library이다. 이 부분에서는 사용자들이 역학 시뮬레이션 후에 선택한 시뮬레이

션을 사용자들의 엔진 개발에 개발 API로 사용할 수 있도록 하기 위해 라이브러리 혹은 DLL의 형태로 제공한다.

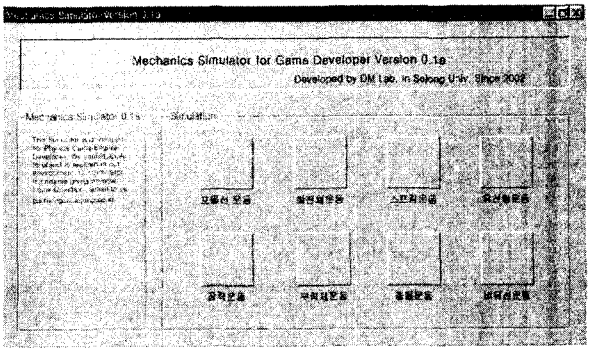


[그림1] 역학 시뮬레이터 전체 구조

3.2 역학 시뮬레이터의 설계 구조

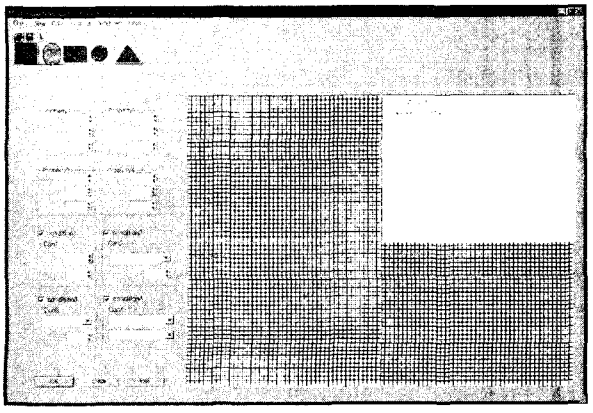
3.2.1 User Interface와 Utility

User Interface와 Utility에서는 사용자의 입력과 동적인 시뮬레이션 환경을 위한 설정 및 사용자들의 역학 시뮬레이션 테스트 환경을 위한 Graphic User Interface를 제공한다. [그림2]는 본 논문에서 구현하기 위해 설계한 역학 시뮬레이터의 초기화면 인터페이스 예시이다.



[그림2] 역학 시뮬레이터 인터페이스 초기화면

사용자가 [그림2]의 인터페이스 초기화면에서 시뮬레이션을 원하는 역학 운동을 선택한 후에 해당 시뮬레이션의 인터페이스 화면이 [그림3]과 같이 나타난다. [그림3]은 사용자가 포물선 운동 시뮬레이션을 선택했다는 가정 하에 역학 시뮬레이터의 하부 시뮬레이션 중 하나인 포물선 운동 인터페이스의 예시로 보여진 것이다. User Interface 부분에서는 각각의 역학 시뮬레이션 환경에 따라 서로 다른 인터페이스 구성을 제공한다.



[그림3] 역학 시뮬레이터 포물선 운동 인터페이스

또한 사용자들에게 편한 인터페이스를 제공하기 위해 drag&drop 기능 등과 시뮬레이션 환경을 3방향의 좌표측면과 실제 3차원 시뮬레이션 장면을 동시에 제공함으로써 보다 사실성이 있으며 오차율이 적은 시뮬레이션 환경을 제공한다. Utility에서는 시뮬레이션에 필요한 렌더링에 관련된 작업과 함께 인터페이스부분에서 필요로 하는 예외처리 및 사용자 이벤트 핸들링을 지원하는 역할을 한다. Utility 모듈에서는 렌더링에 필요한 자료구조는 Mechanics Object Class의 자료구조의 일부와 함께 자체적인 내부 자료구조를 갖는다. 또한 사용자가 시뮬레이션을 완료한 후에 해당 시뮬레이션에 관련된 API를 라이브러리나 DLL로의 반출을 원하는 경우에 SDK로의 연결을 지원하는 역할을 한다.

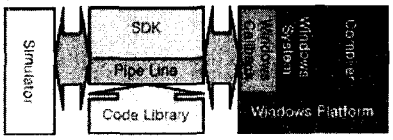
3.2.2 Mechanics Simulation Component

Mechanics Simulation Component는 사용자가 선택한 시뮬레이션 환경을 위한 수학 및 물리 연산을 위한 함수 및 자료구조를 제공하는 컴포넌트의 집합이다. 이 부분은 [그림1]과 같이 2개의 하부구조를 갖는다. 하부구조 중의 하나인 Mechanics Solution Class는 역학 운동의 계산에 필요한 함수들의 집합이다. 이 함수들은 크게 2계층의 구조를 갖는다. 우선 1계층은 수학 연산을 위한 기초적인 함수들로 구성이 된다. 그 위의 2계층은 다양한 역학 시뮬레이션에서 필요한 역학 공식의 해법을 위한 함수들이다. 특히 2계층의 함수들의 구조는 C++에서의 템플릿 구조를 이용하여 역학 시뮬레이션이 진행되는 동안 사용자들에 의한 시뮬레이션 환경의 조건변화를 능동적으로 처리할 수 있는 overriding 구조를 갖는다. 하부구조 중의 다른 하나인 Mechanics Object Class는 수학 연산에 필요한 자료구조를 표현한 클래스들의 집합이다.

3.3 개발도구의 설계 구조

3.3.1 SDK 와 Code Library

SDK와 Code Library 모듈은 역학 시뮬레이터가 시뮬레이터의 기능만을 가진 어플리케이션 아닌 개발자 도구로서의 기능을 제공하는 부분을 담당한다. 사용자는 역학 시뮬레이터에서 선택한 시뮬레이션을 완료한 후에 해당 시뮬레이션의 효과를 API로 제공받기 원한다면 SDK의 모듈에서 사용자에게 DLL 혹은 라이브러리의 형태로 제공을 받는다. 이에 대한 상세 구조는 [그림4]와 같다.

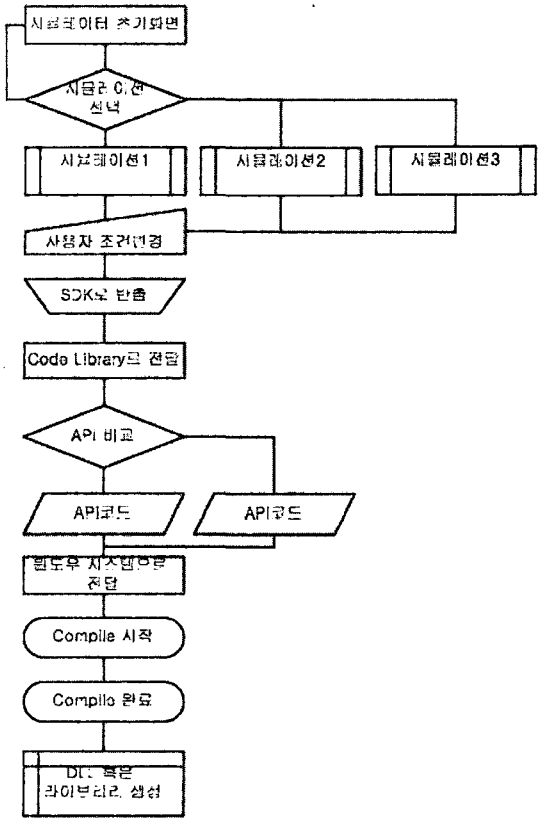


[그림4] 역학 시뮬레이터 SDK 구조

[그림4][12]에서 SDK의 하부 구조인 Code Library는 역학 시뮬레이터의 모든 시뮬레이션에 관련된 API 코드를 압축 된 파일 데이터로 가지고 있는 Class 패키지 형태이다. SDK로부터 해당 시뮬레이션의 ID_table을 전달받으면 Code Library는 ID_table을 통해 이에 해당하는 API코드를 작성하여 SDK로 전달한다. Code Library가 API코드를 직접 작성하는 이유는 본 논문에서 설계하는 역학 시뮬레이터는 사용자들이 시뮬레이션 환경의 조건을 변화시키면서 테스트를 할 수 있는 실시간 시뮬레이션 환경이므로 해당 API들의 파라미터 및 연산의 변화가 유동적이기 때문이다. Pipe Line은 API코드들을 윈도우 플랫폼으로 전달하는 창구의 역할을 한다. Pipe Line이 윈도우 플랫폼 시스템에게 윈도우 Callback 방식으로 컴파일러를 호출하면 컴파일에 필요한 작업이 진행된다. 이 작업이 완료가 되면 컴파일 작업이 실행되고 이를 통해 API 코드들의 DLL이나 라이브러리가 생성이 되는 것이다. 이 작업들은 사용자들은 인식 할 수 없도록 시뮬레이터의 백그라운드에서 수행된다.

4. 동작 구조

역학 시뮬레이터의 동작구조는 [그림5]와 같다. 최초에 사용자는 역학 시뮬레이터의 초기화면 인터페이스에서 시뮬레이션을 선택한다. 선택한 시뮬레이션의 테스트 환경 인터페이스가 생성이 되고 사용자는 테스트를 시작한다. 테스트를 진행하는 중간에 사용자는 여러 가지 조건을 환경에 적용시킬 수 있으며 이에 대한 결과를 실시간으로 얻을 수 있다. 테스트를 완료한 후에 사용자는 시뮬레이션에 대한 API를 제공받기 위해 DLL 및 라이브러리를 요구 할 수 있다. 시뮬레이터는 사용자의 요구를 Utility를 통해 SDK로 전달한다. 이 때 시뮬레이션 ID_table을 이용한다. 시뮬레이션 ID_table의 내용은 사용자가 시뮬레이션을 테스트했던 환경에서 디폴트로 설정이 된 API중 사용자의 조건변화로 인해 파라미터의 값이 변화되거나 혹은 연산의 변화가 발생한 API들의 목록 및 해당 API의 파라미터의 목록이다. 만약 설정의 변화가 없으면 API 목록만을 전달한다. SDK는 시뮬레이션 ID_table을 Code Library에 전달하고 Code Library에서 해당 API코드를 생성하여 다시 SDK로 전달한다. SDK는 Pipe Line을 통해 윈도우 시스템의 미리 설치된 컴파일러를 호출하고 Code Library로부터 전달된 API 코드들을 컴파일러에게 전달한다. 컴파일러는 API들을 컴파일한 후 DLL 혹은 라이브러리의 형태로 작성하여 사용자가 지정한 디렉토리로 이동시킨다. 사용자는 생성된 DLL 혹은 라이브러리와 함께 텍스트 형태의 문서로 된 API 사용설명서를 제공받는다.



[그림5] 역학 시뮬레이터 동작 순서도

5. 결론 및 향후 전망

본 논문에서는 3D 게임 엔진에서 물리적인 효과를 위한 역학 시뮬레이터 및 개발도구에 대한 설계를 하였다. 3D 게임 엔진의 중요한 요소로 자리잡고 있는 물리엔진의 좀 더 유연한 개발을 위한 도구로서의 역학 시뮬레이터 는 실시간 시뮬레이션을 통한 좀 더 정확한 데이터 및 연산의 기능을 제공할 뿐만 아니라 기존의 다른 SDK와는 달리 어플리케이션 개발에 도움이 되는 DLL 혹은 라이브러리를 제공함으로써 제작 기간의 단축 및 작업 능률의 향상을 기대할 수 있다. 또한 게임 분야에 대한 공학적인 접근을 통하여 게임분야에 대한 공학적인 해석을 시도하였다. 현재 본 논문에서 설계한 역학 시뮬레이터의 구현이 진행 중이며 이후에는 보완 및 수정을 거쳐 더 다양한 역학 현상을 표현하는 도구로 완성할 예정이다.

6. 참고 문헌

[1] Chris Hecker, "Physics, The Next Frontier",Game Developer Magazine, 1996, http://www.gdmag.com
[2] Karma, Mathengine, 2001, http://www.mathengine.com
[3] Havok, Havok, 2001, http://www.havok.com
[4] Ipion, Havok, 2001, http://www.havok.com/ipion
[5] Half-life, Sierra,2001, http://half-lifeps2.sierra.com/
[6] Warcraft3, Blizzard, 2002, http://www.blizzard.com
[7] Medal Of Honor: Allied Assert, Electronic Arts, 2002, http://www.ea.com
[8] Quake3, 2001 id Software, 2001, http://www.quake.com
[9] Tim Schroeder, "Collision Detection Using Ray Casting",Game Developer Magazine, 2001, http://www.gdmag.com
[10] Red Faction, Volition, 2001, http://www.volition-inc.com
[11] Emotion Engine, Sony Computer Entertainment Inc., 2000, http://www.playstation2.com
[12] Dino Esposito, "Professtional Visual c++ Windows Shell Programming" , Wrox, 2000, http://www.wrox.com