물리엔진을 활용한 실감형 게임콘텐츠의 설계 및 제작

최종화°, 권기달, 신동규, 신동일
세종대학교 컴퓨터공학과
com97, knight, shindk, dshin@gce.sejong.ac.kr°

Design and Development of Realistic Game Contents Using a Physics Engine

Jonghwa Choi° Kidal Kwon, Dongkyoo Shin, Dongil Shin
Dept. of Computer Engineering, Sejong University

요 약

일반적인 게임 콘텐츠에 있어서 수행되는 현실세계에 대한 표현은 물리적으로 연속적이지 않다. 이는 실제 콘텐츠의 이용적인 측면에서 사용자에게 거부감을 유발시키기도 하여 게임의 재미를 반감시키기도 한다. 이 논문에서는 물리엔진을 활용한 실감형 게임콘텐츠의 설계 및 제작을 통해서 실시간적인 오브젝트에 대한 물리현상을 물리엔진을 이용하여 처리하는 기법을 보여준다. 또한, 오브젝트의 물리적인 동작을 제어하기 위해 물리엔진의 설계에 대해서 보여주며 물리엔진과 연동되는 렌더링 엔진 및 그것을 이용한 콘텐츠의 제작에 대해서도 서술한다.

1. 서 론

물리엔진을 활용한 실감형 게임콘텐츠 제작은 아직 게임 분야에서는 많은 활용이 이루어지지 않은 단계라고 말할 수 있다. 현재 대부분의 PC에는 고성능의 3D 그래픽 칩이 탑재된 그래픽 카드가 일반적이므로 수많은 폴리곤들에 대한 렌더링을 처리함과 동시에 그에 대한 물리적인 계산이 무리 없이 수행가능하다. 오히려 효율적인 물리작용을 산출해내는 사용하기 편리한 물리엔진의 부재가 더 큰 이유로 작용한다. 이 논문에서는 실감형 게임콘텐츠 제작을 위하여 물리엔진을 적용한 게임콘텐츠의 제작 및 물리엔진과 연동된 렌더링엔진의 구현방법에 대해서 제시한다.

2. 게임용 물리 엔진의 구성

이 논문에서 제시되는 3D 게임 물리엔진은 강체(rigid body)를 처리하는 기능을 위주로 구현되었다[1]. 기본적인 폴리곤을 단위로 하여 폴리곤들은 그들의 속성(mass, position, rotation)들을 가지고 있고 폴리곤들의 집합을 통하여 강체모델형성을 만들고 있다. 또한 강체의 특성에 따른 폴리곤들의 연동관계(Joint)를 설정할 수 있도록 설계되어있다. 폴리곤들의 집합들을 통한 내부충돌검사와 외부폴리곤간의 충돌검사가 이루어 질수 있도록 설계되어있다.

다음의 그림 1은 물리엔진의 포괄적인 구성요소들을 표현한 것이다. 여기서는 물리엔진의 구성에 있어서 고려해야할 중요한 두가지를 나열한다. 그중 하나가 각 폴리곤들에 대한 연동관계를 정립하여 각 폴리곤에 대한 이동을 제어하는 것이다. 예를 들어 자동차라는 하나의 독립된 오브젝트를 제작하기 위해서는 몸체를 구성하는 하나의 Box 폴리곤과 바퀴를 구성하는 네 개의 Sphere폴리곤들에 대하여 바퀴의 이동경로(World에서의 하나의 폴리곤으로써의 움직임)에 대한 제어를 Joint 내부에서 설정되고 있다. 자동차 바퀴의 움직임은 Joint 종류(JointBall, JointHinge, JointSlide, JointContact)에 따라

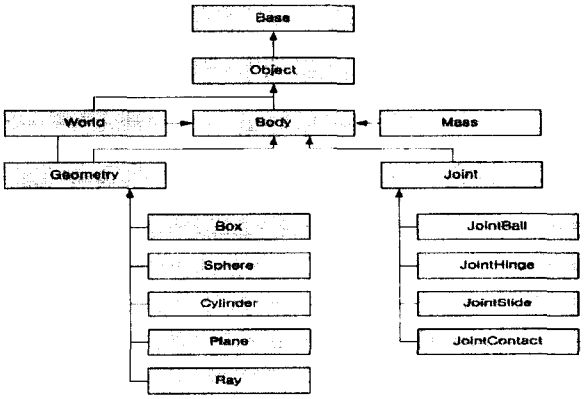

그림 1. 물리엔진구성도

서 각각 다른 움직임을 보인다. 완전히 독립된 오브젝트(자동차)에 대한 이동경로값의 결정을 폴리곤간 설정된 해당 Joint가 담당한다. 또 다른 하나는 각 폴리곤간에 대한 충돌검사이다. 충돌검사는 여러폴리곤들로 구성된 오브젝트의 경우에는 오브젝트간 충돌검사 외에도 오브젝트이 움직임이 있을 시 내부 폴리곤간의 충돌검사도 고려되어야 한다.

위의 그림에서 보면 폴리곤간의 충돌검사는 Geometry Class담당하게 된다. 전체의 노드 구성은 OBB트리로 구성되어 있으며[2] Geometry 에는 각 폴리곤간의 충돌관계(Box-Box, Box-Sphere, Box-Cylinder 등)들을 검사하는 모듈들로 구성되어 있다. 하나의 오브젝트를 생성하기위해 생성된 폴리곤들은 Geometry Group속에 속해져서 오브젝트가 움직임이 있는 경우 먼저 내부 폴리곤들의 상호간의 충돌검사가 행하여진다. 폴리곤들은 자체 속성으로 자신이 Box인지 Sphere인지를 구분하는 인자를 포함하고 있기 때문에 그 인자를 통하여 Box-Box충돌검사를 시행할 것인지 또는 Box-Sphere충돌검사를 시행할 것인지를 결정하게 된다. 이러한 연쇄충돌효과의 비용을 감소하기 위해 병렬구조[3]를 사용하기도 하지만

기본구성 자체가 복셀로 되어있고 그것을 조절할수 있는 매니저의 구성이 있어야 한다는 점에서 컨텐츠에 기반한 실제 시스템에는 많은 실행비용이 따른다.

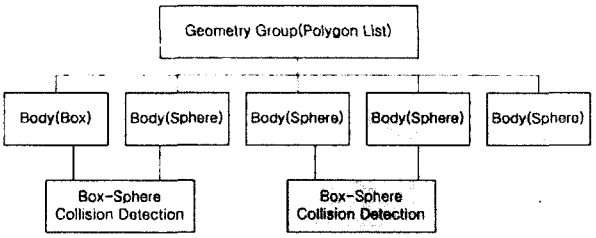

그림 2. 오브젝트(예: 자동차)의 내부충돌리스트 구성

오브젝트에 대한 움직임을 나타내는 두가지 중요한 과정은 구성된 폴리곤들에 대한 각각의 연결움직임과 내부 폴리곤에 대한 충돌검사 및 외부오브젝트에 대한 충돌검사이다. 오브젝트들간의 충돌검사에 대한 부분은 콘텐츠 설계 부분에서 다룬다.

3. 물리엔진과 렌더링 엔진과의 연동

물리엔진을 이용한 콘텐츠 제작 시 먼저 설계되어야 하는 것이 물리엔진과 효율적인 연동구조를 가지는 렌더링 엔진이다. 이 논문에서 제시되는 렌더링 엔진은 DirectX[4] 및 OpenGL[5]을 기반으로 구현되었다. 플랫폼 독립적인 게임엔진[6]의 구성에는 기반설계도의 통일과 세부적으로 사용되는 해당메서드의 통일을 이루어주는 방법으로 되어있다.

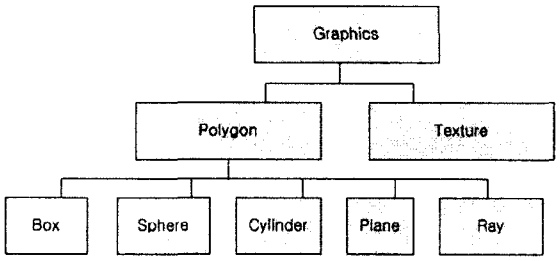

그림 3. 렌더링 엔진의 구성도

위의 그림 3은 렌더링엔진의 내부 구성도이다. Graphics클래스에서는 기본적인 장치디바이스 및 렌더링시 표현되어야할 텍스처 정보 및 화면구성에 따른 카메라설정부분을 포함하고 있다. Polygon 클래스에서는 자식클래스로 다섯 개의 폴리곤들을 가지고 있으면 Polygon 클래스들이 모두 가지는 Draw 메서드를 정의하고 있다. 실제 호출되는 Draw메서드는 각 폴리곤(Box, Sphere, Cylinder, Plane, Ray)에서 각 그리는 방식에 따른 메서드 정의를 포함하고 있다.

여기서 중요한점은 각 폴리곤에 따른 렌더링 객체가 생성되는 것이 아니라 단지 정적으로 각 폴리곤의 종류에 따라 하나씩의 렌더링 객체만이 생성되도록 구성되어야 한다는 것이다. 이는 각 개체의 속성들은 물리엔진에서 생성된 폴리곤의 개체들이 포함하고 있고 렌더링엔진에서의 개체는 단지 물리엔진에서의 개체속성 값만을 전달받아서 화면에 표현하는 역할만 담당한다는 것이다.

다음에 소개되는 소스 코드는 Box 클래스 내부 메서드(Draw)의 예를 DirectX와 함께 사용된 예를 보여주고 있다. 메서드 내부에는 디바이스설정 및 정점버퍼의 생성 및 그에 대한 데이터 조작에 대한 부분이 보여지고 있다.

```
if( FAILED( CD3dDevice->CreateVertexBuffer(10*sizeof(CUSTOMVERTEX),0,
D3DFVF_CUSTOMVERTEX,D3DPOOL_DEFAULT, &BoxVertexBuffer)))
{
        resultValue = -1;
}
...
...

if (resultValue == 0)
{
        float lx = sides[0]*0.5f;
        float ly = sides[1]*0.5f;
        float lz = sides[2]*0.5f;
        ...
        ...
        BoxVertexValues[0] = D3DXVECTOR3(-lx, -ly, -lz);
        BoxVertexValues[1] = D3DXVECTOR3(-lx, -ly,lz);
        BoxVertexValues[2] = D3DXVECTOR3(-lx, ly, -lz);
        BoxVertexValues[3] = D3DXVECTOR3(-lx, ly, lz);
        ...
        ...
        CUSTOMVERTEX* pVertices;
        if( FAILED(BoxVertexBuffer->Lock( 0, sizeof(CUSTOMVERTEX)*10, (BYTE**)&pVertices, 0 ) ) )
                return;
        for (int i = 0; i < 10; i++)
        {
                pVertices[i].position = BoxVertexValues[i];
                pVertices[i].color = 0xffffffff;
        }
        BoxVertexBuffer->Unlock();
        ...
        CUSTOMVERTEX* pVertices1;

        matWorld = D3DXMATRIX(R[0], R[4], R[8], 0, R[1], R[5], R[9], 0, R[2], R[6], R[10],0,pos[0],
pos[1], pos[2], 1);
        CD3dDevice->SetTransform( D3DTS_WORLD, &matWorld );
}
```

그림 4. Box의 Draw메서드(렌더링엔진)

디바이스 설정부분은 Graphics 클래스에서 생성된 장치디바이스를 전달받아서 설정하는 부분으로 처음 Box생성시 한번만 호출되며 정점버퍼의 설정 또한 Box의 경우는 6개의 면으로 구성되기 때문에 정해진 정점의 개수를 기반으로 처음 Box 클래스 객체 생성시 오직 한번만 생성된다. 각 폴리곤에 이동 및 회전에 대한 값들을 물리엔진에서 생성된 폴리곤들의 이동/회전을 통한 결과값을 전달받아서 해당 움직임을 묘사하도록 되어있다.

텍스쳐 설정 및 원하는 텍스쳐 설정부분은 Texture 클래스에서 제어한다. Texture 클래스에서는 내장된 Texture의 생성작업을 객체 생성시 담당한다.

Graphics 클래스에서는 3D에 필요한 기본적인 설정 및 메인 윈도우에서의 레퍼런스를 가지고 있고 양연결에 대한 설정을 담당한다. 중요한 점은 Graphics에서 각 폴리곤에 대한 객체포인터를 has-a관계든 is-a관계든 상관없이 상속관계를 가지고 있지 않다는 것이다.[7] 단지 각 폴리곤들은 메인 렌더링을 담당하는 Ggraphics에서 디바이스 부분에 대한 레퍼런스 포인터를 가지고 있다. 내부 메서드에서는 렌더링의 시작을 알리는 StartRender()와 렌더링의 끝을 알리는 EndRender()가 존재한다. 각 폴리곤들은 이 두 개의 메서드 사이에서 각각의 폴리곤들의 렌더링 함수인 Render()를 호출하는 방식으로 전개되기 때문에 메인렌더링 클래스에서 폴리곤에 대한 주소 레퍼런스를 가지지 않아도 된다. 또한 각 폴리곤들은 월드에 몇 개의 폴리곤들이

존재하던지 상관없이 하나의 객체만이 생산되어 Rendering을 담당하며 각 객체에 대한 관리는 물리엔진(MIPhysicalEngine)에서 관리 담당되고 있다.

4. 물리엔진을 이용한 게임콘텐츠 설계 및 제작

이 논문에서 제시되는 콘텐츠는 자동차 시뮬레이션이다. 자동차를 구성하는 폴리곤간의 Joint방식은 JointHinge방식을 사용하고 내부 충돌감지 및 외부 충돌감지를 행하도록 적용되어있다.

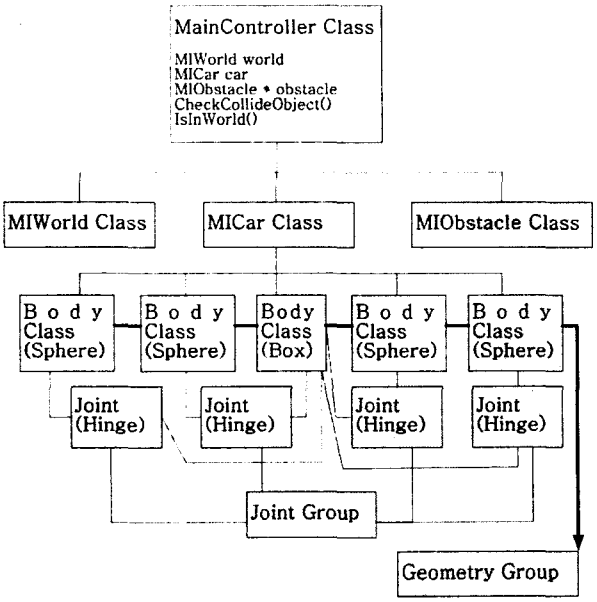

그림 5. Car 콘텐츠 구성도

위의 그림은 Car콘텐츠 제작시 제시되는 설계도이다. 프로세스의 흐름은 World속에 존재하는 전체 오브젝트 및 World Controll은 MainController 클래스에서 담당한다. 전체 구성World/Car/Obstacle에 대한 객체 포인터를 저장하면서 모든 오브젝트에 대한 이동충돌검색 및 월드범위 검색등은 CheckCollideObject()와 IsInWorld()메서드가 담당한다.

세가지 구성요소에 대한 레퍼런스에 대한 참조사항을 MainController클래스에서 참조하고 있으므로 메서드위치는 위와 같다. 각 폴리곤에 대한 상위오브젝트에 대한 충돌검사는 Main에서 recursive로 이루어진다. 오브젝트간의 충돌검사의 경우는 MainController 클래스에서 전체 오브젝트(MICar 클래스 및 MIObstacle 클래스)에 대한 레퍼런스를 가지고 CheckCollideObject()메서드에서 각각의 오브젝트에 대한 충돌검사가 진행이 되는데 이는 개별 오브젝트의 한계범위를 오브젝트 자체가 가지는 것이 아니라 각 오브젝트를 구성하는 폴리곤 리스트(Body Class List)들 간의 충돌 검사에 따르는 것이다.

Car 구성요소(Body 연결설정)부분에서는 각 Body간의 폴리곤 설정은 각 함수설정(ex:setBox())등으로 이루어져 있다. 이러한 설정으로 인해 초기에 생성된 Body와 속성값이 다른 Body로 변환한다. 카메라에 대한 이동은 물리엔진으로 생성된 Car 오브젝트(Body List)에 대한 해당 포지션의 변경된 값으로 시점 이동을 하게 된다. 여기에서의 해당 포지션값은 Car 오브젝트를 구성하는 각 Body중 중점 Body의 포지션 값을 지시한다.

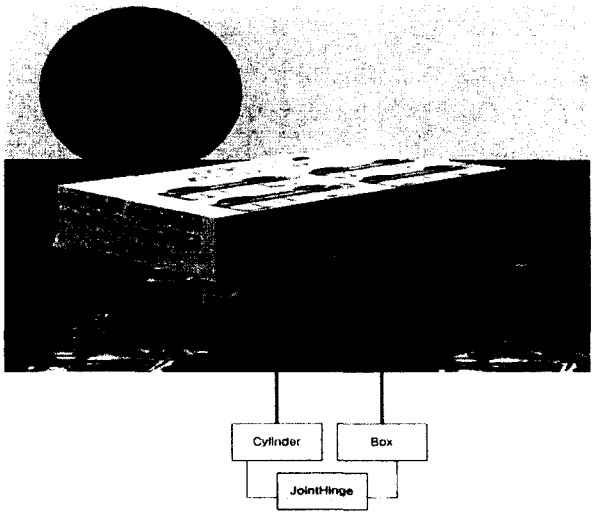

그림 6. 자동차 시뮬레이션 실행화면

3. 결 론

3D 게임 콘텐츠 제작에 있어서 물리엔진의 역할은 콘텐츠에 생명을 불어넣는 기능을 담당한다고 말할 수 있다. 정해진 시나리오에 의해 동작하는 오브젝트는 그 생명력을 잃을 뿐 아니라 동작방식에 대한 문제점을 드러내기도 한다. 콘텐츠 제작에 있어서 그 물리적인 동작방식을 물리엔진을 이용함으로써 수행적 측면의 극대화를 꾀할수 있을 뿐 아니라 구조의 명확성을 보장하여 준다. 다만 물리엔진 설계시에 고려되지 않은 콘텐츠의 물리기능부분에서는 물리엔진의 확장성이 고려되어 있지 않다면 자체 코드의 생성과 함께 기존 물리엔진과의 연동부분에 많은 어려움이 뒤따른다는 점이 파악되었다.

[참고문헌]

[1]김성찬외 3인, 3차원 강체 시뮬레이션 게임을 위한 MMX 아키텍쳐 기반의 물리엔진의 설계, 추계 통신학회 논문집, 2003.11
[2]David H.Eberly, 3D Game Engine Design (Collision Detection), page259-266, 2001.11
[3]Orion Sky Lawlor, A Voxel-Based Parallel Collision Detection Algorithm, *ICS'02*, June 22-26, 2002, New York, New York, USA.
[4]DirectX, Microsoft, http://www.microsoft.com
[5]OpenGL,sgi, http://www.sgi.com/software/opengl
[6]Platform Independent Game Engine
http://www.cs.montana.edu/~charon/thesis/index.php
[7]Gamma,E.,Design Patterns, Addison-Wesley, 1995