물리 엔진의 소개

백 낙 훈*
(경북대학교 컴퓨터과학과)

1. 서 론

컴퓨터가 출현한 이후, 컴퓨터를 원래의 목적이 아닌, 다른 용도로 사용하려는 시도는 꾸준히 있어 왔다. 컴퓨터 게임(computer game) 분야는 그러한 시도들 중에서 상업적 성공을 거둔 것들 중의 하나이다. 이 분야는 몇몇 비주류 프로그래머들로부터 시작되었지만, 점차 상업적인 가능성을 보여 주었고, 꾸준한 발전을 거쳐, 최근에는 수백 명이 몇 년에 걸쳐 전력 투구하는, 대형 프로젝트들로 변모하고 있다[1].

프로젝트의 크기가 커지면서, 게임 분야에 특화된 개발 툴(development tool)들이 속속 출현하게 되었다. 특히, 게임 엔진(game engine)은 일반적으로 게임 개발 과정에서 사용되는 공통 요소(common element)들을 모아 놓은 미들웨어(middle-ware)라고 볼 수 있다. 게임 엔진은 초기에는 화면 출력을 위한 랜더링(rendering) 기능들이 핵심이었지만, 점차 게임의 다른 구성 요소들도 포함되기 시작하면서, 최근에는 렌더링 엔진은 물론이고, 네트웍 엔진(network engine), 인공 지능 엔진(artificial intelligence engine), 스크립트 엔진(script engine) 등의 세분화된 요소들이 포함되고 있다.

물리 엔진(physics engine)은 게임 엔진의 세분화된 구성 요소들 중의 하나로, 게임 내에 출현하는 물체들의 운동을 자연스럽게 하는 데에 특화된 소프트웨어 라이브러리(software library)로 볼 수 있다. 즉, 대상이 되는 물체의 움직임이 실세계(real world)의 물리학 법칙들을 준수하도록, 또는 준수하는 것처럼 보이도록 함으로써, 게임 플레이어(game player)들에게 사실감을 주고자 하는 데에 목적이 있다. 특히, 최근에 들어와 3차원 상에서 사실적인 화면을 강조하는 게임들이 급속히 증가하면서, 물리 엔진에 대한 요구가 증가하고 있다.

기존의 컴퓨터 그래픽스(computer graphics) 분야에서는 컴퓨터 애니메이션(computer animation) 제작 과정에서 이미 유사한 요구가 있어 왔고, 물리 기반 모델링(physically-based modeling)이라는 세부 분야에서 상당한 연구 결과가 제시되어 있다[2,3]. 그러나, 컴퓨터 게임은 컴퓨터 애니메이션과는 전혀 다른 특성들을 가지기 때문에, 이들 연구 결과들을 즉각적으로 사용하기에는 무리가 따른다. 예를 들어, 전형적인 컴퓨터 애니메이션 제작 과정에서는 제작된 화면들을 최종적으로 애니메이션으로 변환하는 방식이기 때문에, 각 화면의 제작 시간보다는 최종 품질을 강조한다.

반면에, 컴퓨터 게임에서는 어떠한 화면 출력도 정해진 시간 내에 이루어져야 한다는, 실시간 처리(real-time processing)에 대한 요구가 매우 강력하다. 또, 일방적으로 화면만을 보여주는 애니메이션과는 달리, 사용자의 입력에 반응해야 한다는, 대화형 처리(interactive processing)에 대한 제약도 따른다. 이러한 측면을 고려한다면, 물리 엔진의 역할은 그림 1에서와 같이, 주어진 시간 t에서의 상태(state)에 게임 엔진의 다른 요소들에서 들어오는 입력들을 반영하여, 시간 $t+\Delta t$에서의 새로운 상태를 물리학 법칙에 맞게 계산하여, 렌더링 엔진으로 넘기는 것으로 볼 수 있다.

게임 개발 과정에서 물리 엔진의 사용이 증가하는 것에는 분명한 이유가 있다. 물리 엔진을 사용하기 이전에는 물체의 움직임을 표현하기 위해서는 어떠한 법칙 보다는 임의로 설정된 움직임이 나오게 하는 방식을 택할 수 밖에 없었기 때문에, 사실

[이 논문은 2004년도 경북대학교 학술진흥연구비에 의하여 연구되었음]

*E-mail: nbaek@knu.ac.kr

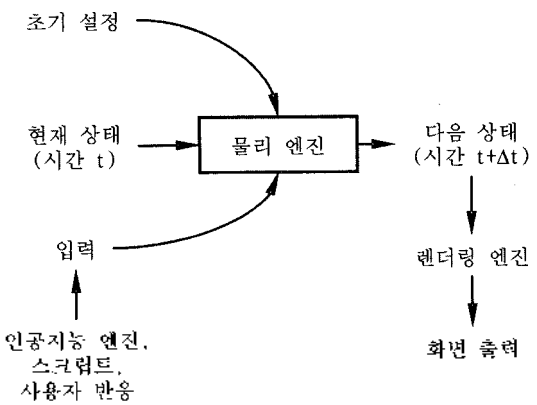


그림 1. 물리 엔진의 역할

성이 떨어지는 경우가 많았다. 반면에, 물리 엔진은 모든 물체들에 대해서 매우 사실적인 움직임을 만들 수 있다는 장점을 가진다. 또, 이미 개발된 물리 엔진을 사용하게 되면, 게임 개발 과정을 획기적으로 단축시킬 수도 있다. 물론, 물리 엔진 자체의 개발은 상당한 시간과 비용을 필요로 하지만, 최근에 와서는 상업적으로 판매되는 물리 엔진을 사용하여 전체 일정과 비용을 줄일 수 있다[1,4].

반면에, 물리 엔진이 가져오는 문제점도 몇 가지 있다. 우선, 물리 법칙의 적용은 필연적으로 더 많은 계산 시간을 필요로 하기 때문에, 게임 엔진의 다른 요소들에서 필요로 하는 시간을 희생해야 하는 경우가 생길 수 있다. 또, 역으로, 물리 법칙에 위배되는 움직임을 표현하고자 하는 경우에는 물리 엔진과는 별도의 예외 처리 부분을 작성해야 하기 때문에 가외의 시간과 비용을 필요로 할 수도 있다.

현 시점에서는 물리 엔진이 가져오는 이득이 더 많기 때문에, 게임 개발 과정에서의 물리 엔진의 사용은 더욱더 늘어날 전망이다. 반면에, 게임 개발자의 입장에서는 물리 엔진의 사용이 그렇게 반가운 일만은 아니다. 대부분의 게임 개발자들은 프로그래밍이나 디자인 관점에서는 숙련되어 있지만, 물리 엔진에서 다루어야 하는 물리량(physical quantity)들이나 물리 법칙들에는 익숙하지 못하기 때문에, 물리 엔진을 사용함에 있어 생소함을 느끼기 마련이고, 이 때문에, 물리 엔진은 사용하기 어렵다는 의견이 나오게 된다.

이 글에서는 물리 엔진에 대해서 전반적으로 소개함으로써, 물리 엔진의 사용이 용이하게 하는 데에 목적이 있다. 2절에서는 일반적인 물리 엔진들을 이해함에 있어 필수적인 물리량과 물리 법칙들을 소개하고, 이로부터, 전형적인 물리 엔진의 구조를 설명하겠다. 3절에서는 좀 더 고급 기능인 충돌 처리와 다관절체 처리에 대해서 설명하겠고, 4절에서 상업적으로 판매되는 물리 엔진들을 소개한 후, 5절에서 결론을 맺겠다.

2. 물리 엔진의 구성

2.1 물리량들

물리 엔진은 기존의 물리 법칙들에 기초하기 때문에, 물리량들에 대한 이해가 필수적이다. 이 절에서는 전형적인 3차원의 강체(rigid object)가 가지는 물리량들에 대해서 간단히 설명하겠다.

물체의 질량(mass) m은 물체 고유의 스칼라 양(scalar quantity)으로, 직선 운동에 대한 저항으로 볼 수 있다. 즉, 고전 역학의 운동 법칙에 따라, 힘 $\mathbf{f}$와 가속도 $\mathbf{a}$의 관계를 $\mathbf{f}=m\mathbf{a}$로 본다면, $\mathbf{a}=(1/m)\mathbf{f}$로 볼 수 있고, 같은 힘에 대해서, 질량이 클수록 운동에 대한 저항이 크다고 볼 수 있다.

질량 중심(center of mass) $\mathbf{x}_{cm}=(x_{cm}, y_{cm}, z_{cm})$은 물체 내부의 한 점으로, 어떠한 방향으로 이 점에 힘을 가하더라도 해당 물체에 회전 운동은 일어나지 않는다는 특성을 가진다. 즉, 질량 중심에 가해진 힘은 물체에 직선 운동만을 일으킬 수 있다. 이 특성 때문에, 물리 엔진에서는 질량 중심을 각 물체의 물체 좌표계(object coordinate system)의 원점(origin)으로 사용한다. 질량 중심은 각 물체마다의 고유한 성질이고, 무게 중심(center of gravity)이라고도 표현한다.

질량이 직선 운동에 대한 저항이라면, 관성 모멘트(moment of inertia) $\mathbf{I}$는 회전 운동에 대한 저항을 의미한다. 3차원 물체에서는 회전 운동의 중심축이 어떤 방향으로도 설정될 수 있기 때문에, 흔히 3×3 행렬인 2차 텐서(tensor)로 표현하고, 계산 과정은 다음과 같다:

$$\mathbf{I} = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{yx} & I_{yy} & -I_{yz} \\ -I_{zx} & -I_{zy} & I_{zz} \end{bmatrix},$$

$$I_{xx} = \int(y^2+z^2)\mathrm{d}m$$
$$I_{yy} = \int(x^2+z^2)\mathrm{d}m$$
$$I_{zz} = \int(x^2+y^2)\mathrm{d}m$$
$$I_{xy} = I_{yx} = \int(xy)\mathrm{d}m$$
$$I_{yz} = I_{zy} = \int(yz)\mathrm{d}m$$
$$I_{zx} = I_{xz} = \int(zx)\mathrm{d}m.$$

복잡한 형태의 물체에 대해서는 관성 모멘트를 계산하기가 쉽지 않기 때문에, 게임과 같이, 근사 계산만으로 충분한 효과를 얻을 수 있는 경우에는 바운딩 박스(bounding box)와 같은, 비슷한 크기의 좀더 단순한 형태에 대한 관성 모멘트를 근사값으로 사용하는 것이 일반적이다. 또, 물체 좌표계는 흔히 관성 모멘트의 계산이 쉬운 방향을 좌표축으로 설정하는 것이 일반적이다.

각 물체의 형태는 물체 좌표계에서 표현할 수 있지만, 물체의 운동을 표현하려면, 세계 좌표계(world coordinate system)를 사용하여야 한다. 시간 t에 따라, 물체 좌표계의 원점이 세계 좌표계의 $\mathbf{x}(t)$에 위치하고, 물체 좌표계의 방향(orientation)을 사원수(quaternion) $\mathbf{q}(t)$로 표현한다면, 물체 좌표계에서의 한 점 $\mathbf{p}_{\text{local}}(t)$은 대응되는 세계 좌표계에서의 점 $\mathbf{p}_{\text{world}}(t)$와 다음과 같은 관계를 가진다.

$$\mathbf{p}_{\text{world}}(t) = \mathbf{q}(t)\mathbf{p}_{\text{local}}\mathbf{q}^{\text{conj}}(t)+\mathbf{x}(t),$$
$$\mathbf{p}_{\text{local}} = \mathbf{q}^{\text{conj}}(t)(\mathbf{p}_{\text{world}}(t)-\mathbf{x}(t))\mathbf{q}(t). \qquad (1)$$

여기서 $\mathbf{q}^{\text{conj}}(t)$는 $\mathbf{q}(t)$의 켤레(conjugate) 사원수이다.

물체가 이동하는 선속도(linear velocity) $\mathbf{v}(t)$는 물체의 질량 중심의 위치에 대한 변화율로 정의되고, 시간 t에 대한 미분을 이용하여 다음과 같이 계산된다.

$$\mathbf{v}(t) = \dot{\mathbf{x}}(t) = \frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \left(\frac{dx(t)}{\mathrm{d}t}, \frac{dy(t)}{\mathrm{d}t}, \frac{dz(t)}{\mathrm{d}t}\right)$$

물체의 회전에 대한 각속도(angular velocity) $\boldsymbol{\omega}(t)$는 3차원에서의 회전 중심축 단위 벡터에 회전 속도를 곱한 형태로 표현된다. 즉, 각속도 $\boldsymbol{\omega}(t)=(\omega_x, \omega_y, \omega_z)$는 그 방향이 회전축 방향이고, 크기 $|\boldsymbol{\omega}(t)|$가 회전 속도인 3차원 벡터가 된다. 시간 t에 대한 물체의 방향 변화 $\dot{\mathbf{q}}(t)$는 다음과 같이 각속도 $\boldsymbol{\omega}(t)$와 방향 $\mathbf{q}(t)$에 대한 식으로 계산할 수 있다.

$$\dot{\mathbf{q}}(t) = \frac{1}{2}[0, \boldsymbol{\omega}(t)]\mathbf{q}(t)$$

여기서, $[0, \boldsymbol{\omega}(t)]$는 사원수 계산의 편의를 위해, $[0, \boldsymbol{\omega}(t)] = (0, \omega_x, \omega_y, \omega_z)$로 설정된다.

세계 좌표계에서, 물체 내부의 한 점 $\mathbf{p}_{\text{world}}(t)$는 식 (1)에서와 같이 표현되므로, 이 점에서의 선속도는 시간 t에 대한 미분을 취함으로써, 다음과 같이 계산할 수 있다.

$$\dot{\mathbf{p}}_{\text{world}}(t) = \boldsymbol{\omega}(t)\times(\mathbf{p}_{\text{world}}(t)-\mathbf{x}(t))+\mathbf{v}(t)$$

즉, $\mathbf{p}_{\text{world}}$에서의 선속도는 질량 중심의 선속도 $\mathbf{v}(t)$와 물체의 각속도 $\boldsymbol{\omega}(t)$를 모두 반영하게 된다.

고전 역학에서의 운동 법칙들에 따라, 물체에 힘 $\mathbf{f}$를 가하면, 물체는 운동을 하게 된다. 이 때, 힘 $\mathbf{f}$가 질량 중심에 가해진다면, 직선 운동만 일어나지만, 그 외의 경우는 모두 직선 운동에 추가하여 회전 운동도 일어난다. 이러한 회전 운동을 일으키는 물리량이 토크(torque)이고, 점 $\mathbf{p}_{\text{world}}$에 힘 $\mathbf{f}$를 가했을 때의 토크 $\boldsymbol{\tau}$는 다음과 같이 계산된다.

$$\boldsymbol{\tau}(t) = (\mathbf{p}_{\text{world}}(t)-\mathbf{x}(t))\times\mathbf{f}(t)$$

이 때, $\boldsymbol{\tau}$의 방향은 회전축 방향이 되고, $\boldsymbol{\tau}$의 크기는 각가속도의 크기가 된다.

물체에 작용하는 힘들 중에, 중력(gravity force)이나 공기 저항, 유체 저항 등은 물체 전체에 작용하기 때문에, 회전을 일으키지는 않는다. 반면에, 물체의 한 점에 작용하는 힘은 반드시 회전을 일으키게 되고, 토크에 의한 회전을 반영하여야 올바른 움직임을 표현할 수 있다.

2.2 물리 법칙들

물리 엔진에서 주로 다루게 되는 고전 역학에서 가장 기본이 되는 물리 법칙들은 다음에 제시된 뉴튼(Newton)의 운동 법칙들이다[5,6].

○ 관성의 법칙: 물체에 외력(external force)이 작용하지 않는 한, 물체는 현재의 운동 상태를 유지하려 한다.

○ 가속도의 법칙: 물체의 가속도는 물체에 작용하는 힘에 비례하고, 가속도의 방향은 힘의 방향이다.

○ 작용-반작용의 법칙: 물체에 힘이 작용하면, 크기가 같은 힘이 반대 방향으로 작용한다.

이들 중에서도 뉴튼의 운동 제 2법칙에서 제시되는 $\mathbf{f}=m\mathbf{a}$는 물리 엔진의 구현에 있어 핵심이 되고, 일반적으로 운동량을 도입하여 표현한다.

운동량(momentum) $\mathbf{P}$는 물체의 질량과 선속도의 곱으로 정의되고, $\mathbf{P}=m\mathbf{v}$로 계산할 수 있다. 운동량은 그 자체로서도 의미를 가지지만, 이를 미분하면, 뉴튼의 운동 제 2법칙 $\mathbf{f}=m\mathbf{a}$를 다르게 표현하는 것이 가능하다. 즉, $\mathbf{P}=m\mathbf{v}$의 시간 t에 대한 미분은 다음과 같이 계산된다.

$$\begin{aligned}\dot{\mathbf{P}}(t) &= m\dot{\mathbf{v}}(t)\\ &= m\mathbf{a}(t)\\ &= \mathbf{f}(t)\end{aligned}$$

따라서, $\dot{\mathbf{P}}(t)=\mathbf{f}(t)$이고, 이는 $\mathbf{f}=m\mathbf{a}$에 대한 다른 표현으로 볼 수 있다.

각 운동량(angular momentum) $\mathbf{L}$은 운동량에 대한 대칭 개념으로, 다음과 같이 정의된다.

$$\mathbf{L}=\mathbf{I}\boldsymbol{\omega}$$

이 때, $\mathbf{I}$는 관성 모멘트의 텐서 표현이고, 세계 좌표계에서의 관성 텐서는 물체 좌표계에서의 관성 텐서 $\mathbf{I}_{local}$에서 다음과 같이 계산할 수 있다.

$$\mathbf{I}=\mathbf{q}(t)\mathbf{I}_{local}\mathbf{q}^{conj}(t)$$

뉴튼의 운동 제 2법칙을 적용하여, 물체의 가속도, 속도, 위치를 차례로 구하려면, 적분을 사용하여 다음과 같이 계산할 수 있다.

$$\begin{aligned}\mathbf{a} &= \frac{1}{m}\mathbf{f}\\ \mathbf{v} &= \int \mathbf{a}dt\\ \mathbf{x} &= \int \mathbf{v}dt\end{aligned} \qquad (2)$$

이 계산을 미분 방정식 형태로 표현한다면, 다음과 같은 수식도 가능하다.

$$\begin{aligned}\mathbf{a} &= \frac{1}{m}\mathbf{f}\\ \dot{\mathbf{v}} &= \mathbf{a}\\ \dot{\mathbf{x}} &= \mathbf{v}\end{aligned} \qquad (3)$$

가속도 대신, 운동량 P를 사용한다면, 다음과 같은 계산도 가능하다.

$$\begin{aligned}\dot{\mathbf{P}} &= \mathbf{f}\\ \mathbf{v} &= \frac{1}{m}\mathbf{P}\\ \dot{\mathbf{x}} &= \mathbf{v}\end{aligned} \qquad (4)$$

즉, 위의 식 (2), (3), (4)는 같은 계산 과정을 다르게 표현한 것일 뿐이다. 물리 엔진을 구현하는 입장에서는 어떠한 형태로든 뉴튼의 운동 제 2법칙을 표현하여야 하는데, 이들 중에서 가장 계산량이 적은 것을 택하게 되고, 대부분의 물리 엔진에서는 식 (4)를 사용한다.

같은 방법으로 회전 운동을 각 운동량을 사용하여 표현하면, 다음과 같다.

$$\begin{aligned}\dot{\mathbf{E}} &= \boldsymbol{\tau}\\ \boldsymbol{\omega} &= \mathbf{I}^{-1}\mathbf{L}\\ \dot{\mathbf{q}} &= \frac{1}{2}[0,\boldsymbol{\omega}]\mathbf{q}\end{aligned} \qquad (5)$$

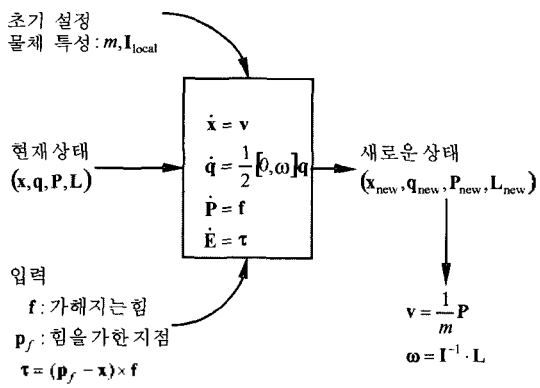


그림 2. 물리 엔진의 처리 과정

정리하면, 물리 엔진은 뉴튼의 운동 제 2법칙을 적용하기 위해, 식 (4), (5)의 미분 방정식들을 사용한다.

2.3 물리 엔진의 구현

앞 절에서 설명한 바와 같이, 물체의 운동을 표현하기 위해서는 식 (4), (5)의 미분 방정식들의 해를 구하는 방법이 일반적으로 사용될 수 있다. 따라서, 물리 엔진의 기본적인 기능은 이들 미분 방정식의 해를 구하는 과정으로 볼 수 있다. 그림 2에서 보는 바와 같이, 대상이 되는 모든 물체들의 역학적인 상태는 대상 물체의 위치(position) $\mathbf{x}$, 방향(orientation) $\mathbf{q}$, 운동량(momentum) $\mathbf{P}$, 각 운동량(angular momentum) $\mathbf{L}$로 표현할 수 있다. 사용자의 입력이나 인공 지능 엔진, 스크립트 엔진에서의 입력들은 모두 힘 $\mathbf{f}$와 그 힘이 가해지는 지점 $\mathbf{p}_f$, 그에 따른 토크 $\boldsymbol{\tau}$로 주어지게 된다. 또, 각 물체마다의 고유한 물리량들인 질량 m, 관성 텐서 $\mathbf{I}$ 등은 초기에 미리 설정해 두게 된다. 이제 물리 엔진은 매 순간, 시간 t에서의 현재 상태 $(\mathbf{x},\mathbf{q},\mathbf{P},\mathbf{L})$과 시간 t에 가해지는 $\mathbf{f}$, $\boldsymbol{\tau}$를 사용하여, 다음의 미분 방정식들을 구성하게 된다.

$$\dot{\mathbf{x}} = \mathbf{v}$$

$$\dot{\mathbf{q}} = \frac{1}{2}[0, \boldsymbol{\omega}]\mathbf{q}$$

$$\dot{\mathbf{P}} = \mathbf{f}$$

$$\dot{\mathbf{L}} = \boldsymbol{\tau}$$

이 미분 방정식을 해석함으로써, 시간 $t+\Delta t$에서의 새로운 상태 ($\mathbf{x}_{new}$, $\mathbf{q}_{new}$, $\mathbf{P}_{new}$, $\mathbf{L}_{new}$)가 계산되고, 이 값들로부터 추가로 필요한 물리량들인 속도 $\mathbf{v}$, 각속도 $\boldsymbol{\omega}$가 다음과 같이 계산된다:

$$\mathbf{v} = \frac{1}{m}\mathbf{P}$$

$$\boldsymbol{\omega} = \mathbf{I}^{-1}\mathbf{L}$$

이제 각 물체의 시간 $t+\Delta t$에서의 위치 $\mathbf{x}_{new}$, 방향 $\mathbf{q}_{new}$, 속도 $\mathbf{v}$, 각속도 $\boldsymbol{\omega}$를 모두 알고 있는 상황이므로, 렌더링 엔진을 통하여 화면 상에 대응되는 물체의 현재 상태를 표현하는 것이 가능하다. 물리 엔진은 이 과정을 반복함으로써, 매 순간마다 물리 법칙에 따른 물체들의 움직임을 화면에 정확히 표현할 수 있게 된다.

미분 방정식의 해석에 있어서는 다양한 수치 해석 방법들이 사용될 수 있는데, 주어진 미분 방정식들은 전형적인 상미분 방정식(ordinary differential equation)의 형태를 띄므로, 통상 Euler method나 Runge-Kutta method를 사용한다[7,8]. 예를 들어, Euler method를 적용하여 물체의 새로운 위치 $\mathbf{x}(t+h)$를 계산하고자 한다면, 물체의 이전 위치 $\mathbf{x}(t)$와 속도 $\mathbf{v}(t)$를 이용하여 다음과 같은 계산이 가능하다.

$$\begin{aligned}\mathbf{x}(t+h) &= \mathbf{x}(t) + h\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} \\ &= \mathbf{x}(t) + h\mathbf{v}(t)\end{aligned}$$

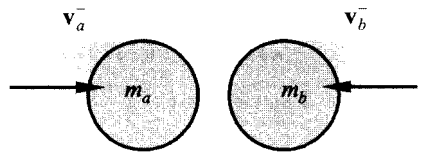
Runge-Kutta method에서는 좀 더 복잡한 계산이 필요하지만, 계산의 정확도가 높기 때문에, 최근에 와서는 게임에 사용되는 PC들의 성능이 높아지면서, 점차 Runge-Kutta method를 선호하는 추세이다.

3. 충돌 및 다관절체 처리

3.1 충돌 처리

물리 엔진의 가장 기본적인 가능인, 물체의 사실적인 움직임을 생성하는 것은 앞 절에서 설명한 바와 같이, 물리 법칙들의 미분 방정식들을 해석하는 과정으로 구현이 가능하다. 반면에, 3차원 상에서 물체들이 움직이는 과정을 표현하게 되면, 반드시 추가로 구현되어야 할 기능이 충돌 처리(collision handling)이다. 이 과정은 세부적으로는 움직이는 물체들 간의 충돌을 감지하는 충돌 탐지(collision detection) 단계와 감지된 충돌에 대해 역학적으로 계산된, 사실적인 운동을 보이도록 하는 충돌 반응(collision response) 단계로 나누어 볼 수 있다.

충돌 반응 단계는 필연적으로 물리 기반 모델링 에서의 기법들이 적용되어야 하고, 이 때문에, 일반적인 물리 엔진은 충돌 탐지 단계와 충돌 처리 단계를 모두 구현하여 제공하게 된다. 컴퓨터 그래픽스 분야에서는 이미 충돌 처리 전반에 걸친 많은 연구 결과가 제시되어 있다. 특히 충돌 탐지 단계에서의 연구 결과들은 별도의 문헌이 이미 본 학회



(a) 충돌 전

(b) 충돌 후

그림 3. 충돌 전후의 물리량들

지에 소개된 적이 있으므로, 이를 참고하기 바란다[9].

충돌 탐지 단계에서 충돌이 감지 되면, 충돌이 일어난 시간 t, 충돌의 대상이 되는 물체 a, b와 충돌이 일어난 지점 $\mathbf{p}_f$, 그 지점에서의 각 물체 표면의 법선 벡터(normal vector) $\mathbf{n}$ 등의 정보가 충돌 반응 단계로 넘어온다. 충돌 반응 단계에서는 이들 정보와 대상 물체 a, b의 물리량들로부터 다음 순간의 물체의 물리량들을 새로 계산하게 된다.

충돌 반응에서 가장 핵심이 되는 물리 법칙은 운동량 보존(conservation of momentum)의 법칙이다. 그림 3에서와 같이, 두 물체 a, b가 충돌했다고 가정하자. 두 물체 a, b의 질량을 각각 m_a, m_b라 하고, 충돌 직전의 각 물체의 속도를 $\mathbf{v}_a^-$, $\mathbf{v}_b^-$, 충돌 직후의 각 물체의 속도를 $\mathbf{v}_a^+$, $\mathbf{v}_b^+$라 하면, 운동량 보전의 법칙은 다음과 같은 수식으로 표현할 수 있다.

$$m_a\mathbf{v}_a^- + m_b\mathbf{v}_b^- = m_a\mathbf{v}_a^+ + m_b\mathbf{v}_b^+$$

즉, 충돌 직전에 두 물체 a, b의 운동량의 합계는 충돌 직후의 두 물체의 운동량의 합계와 일치한다는 것이다.

위와 같이 운동량이 완전히 보존되는 경우를 완전 탄성 충돌이라 하는데, 실세계에서의 충돌에서는 소리나 물체의 내부 변형 등으로 전체 운동량의 일부가 소실되어, 다음과 같은 형태의 비탄성 충돌이 일어난다.

$$m_a\mathbf{v}_a^- + m_b\mathbf{v}_b^- = m_a\mathbf{v}_a^+ + m_b\mathbf{v}_b^+ + E$$

여기서, E는 비탄성 충돌에서 소실되는 운동량을 의미한다. 이 소실되는 운동량과 충돌 전의 전체 운동량과의 비율을 탄성 계수(coefficient of restitution) ε이라 하고, 각 물체의 고유 성질이다. $\varepsilon = 1$인 경우에 완전 탄성 충돌이 일어나고, $\varepsilon = 0$인 경우에는 완전 비탄성 충돌이 일어나서 두 물체가 같은 속도로 움직이게 된다.

충돌 반응을 처리하기 위해서는 새로운 물리량으로 충격량(impulse)의 개념이 도입된다. 충격량은 운동량의 변화로 정의되고, 물체 a의 질량에 변화가 없다면, 물체 a에 가해지는 충격량은 다음과 같이 계산할 수 있다.

$$\mathbf{J} = \Delta\mathbf{P} = m\Delta\mathbf{v} = m(\mathbf{v}_a^+ - \mathbf{v}_a^+)$$

물체 b의 경우는 운동량 보존의 법칙에 따라, 물체 a에 가해진 충격량의 부호만 바뀐 충격량이 가해지기 때문에, 별도로 계산하지는 않고, 최종 계산 결과의 부호만 바꾸어서 적용하는 방법을 사용한다.

물체 a의 표면 상에서 충돌이 일어난 지점에서의 단위 법선 벡터(unit normal vector)를 $\mathbf{n}$이라 하면, 충격량은 이 단위 법선 벡터와 방향이 같고, 크기는 충돌에 따른 정보들로 계산된다. 충격량을 계산하는 수식은 이미 역학 분야에서 아래와 같이 제시되어 있다.

$$\mathbf{J} = j\mathbf{n},$$

$$j = \frac{(1+\varepsilon)(\mathbf{n}\cdot(\mathbf{v}_a - \mathbf{v}_b))}{\frac{1}{m_a} + \frac{1}{m_b} + \mathbf{n}\cdot(\mathbf{I}_a^{-1}(\mathbf{r}_a\times\mathbf{n}))\times\mathbf{r}_a + \mathbf{n}\cdot(\mathbf{I}_b^{-1}(\mathbf{r}_b\times\mathbf{n}))\times\mathbf{r}_b}$$

여기서, $\mathbf{r}_a$, $\mathbf{r}_b$는 각각 물체 a, b의 질량 중심에서 충돌 지점까지의 거리 벡터이다.

위에서 정의한 충격량은 충돌 전후에 물체의 속도를 순간적으로 바꿀 정도로 매우 큰 힘이 아주 짧은 순간 동안 물체에 가해졌다고 해석할 수 있다. 즉, 충돌을 별도로 처리하지 않고, 기존의 힘과 토크로 해석하기 위해서, 충격량 자체를 물체에 가해진 아주 큰 힘인 impulse force $\mathbf{F}$와 충돌이 일어난 시간 간격 Δt의 곱으로 아래와 같이 표현한다.

$$\mathbf{J} = \mathbf{F}\Delta t$$

$$\mathbf{F} = \frac{\mathbf{J}}{\Delta t}$$

운동량에 대하여 impulse force의 개념이 도입된 것과 같은 방식으로, 각운동량에 대해서는 impulse torque $\tau_{impulse}$의 개념이 도입되고, 이는 다음과 같이 계산된다.

$$\tau_{impulse} = \Delta \mathbf{L} = (\mathbf{p}_f - \mathbf{x}(t)) \times \mathbf{J}$$

이러한 계산 방식은 이미 구현된 물리 엔진의 입력인 힘 **f**와 토크 **τ**만을 새로운 값으로 대체하여 충돌을 처리할 수 있다는 장점을 가진다.

결과적으로 충돌 반응은 충격량 **J**를 계산한 후, 다음과 같이, 각 물체의 운동량을 새로 계산한다.

$$\mathbf{P}_a^+ = \mathbf{P}_a^- - \mathbf{J}$$
$$\mathbf{P}_b^+ = \mathbf{P}_b^- + \mathbf{J}$$

경우에 따라서는 힘과 토크를 이용한 간접 계산 대신, 물체의 속도를 아래와 같이 직접 변화시키는 방법을 사용하기도 한다.

$$\mathbf{v}_a^+ = \frac{1}{m}\mathbf{p}_a^+ = \frac{1}{m}(\mathbf{P}_a^- - \mathbf{J})$$
$$\mathbf{v}_b^+ = \frac{1}{m}\mathbf{p}_b^+ = \frac{1}{m}(\mathbf{P}_b^- - \mathbf{J})$$

위와 같은 방식으로 물리 엔진은 3차원 물체들 간의 충돌에 따른 변화를 물리 법칙에 따라 계산하여 보여 줄 수 있다. 필요에 따라서는 충돌에 따른 효과음의 처리를 위해 사운드를 생성하기도 한다.

3.2 다관절체 처리

물리 엔진에서 다루는 물체는 단일 강체(rigid body)를 출발점으로 하지만, 실세계에서는 이러한 강체들이 서로 연결된 다관절체(articulated body)가 상당한 비중을 차지한다. 예를 들어, 인체는 여러 개의 조직들이 관절 부위에서 연결된 다관절체로 모델링할 수 있고, 자동차의 경우는 차체에 4개의 바퀴가 서스펜션(suspension) 부위에서 스프링(spring)과 댐퍼(damper)로 연결된 다관절체로 볼 수 있다.

따라서, 물리 엔진에서는 2개 이상의 강체가 관절로 연결된 다관절체에 대한 처리가 필수적이다. 물론 각 강체의 처리는 기존의 물리 법칙을 준수하도록 각각 계산할 수 있지만, 강체들 간의 연결 상태가 반드시 유지되도록 하는 추가 처리가 필수적이다. 컴퓨터 그래픽스 분야에서는 이러한 다관절체의 처리를 constrained dynamics라는 세부 분야에서 다루고 있고, 물리 엔진에서도 constrained dynamics의 연구 결과들을 사용하여 처리하고 있다.

Constrained dynamics 분야에서는 다양한 방법으로 다관절체를 처리하고 있는데, 그 모두를 소개하기는 무리이고, 개념적으로 비교적 간단한 처리 방법인 penalty method를 살펴 보겠다[3]. 이 방법은 관절과 같이, 물체에 주어진 제약 조건(constraint)을 직관적으로 해결한다. 즉, 각 관절 부위에 가상의 매우 강력한 스프링을 삽입하여, 관절 부위에서 두 물체가 떨어지려고 하면, 이 가상의 스프링이 매우 강력한 힘으로 잡아 당겨서 강제로 연결을 유지하게 하는 방법이다. 이 방식은 거의 모든 물리 엔진들이 별다른 추가 처리 없이 쉽게 가상의 스프링을 처리할 수 있다는 점 때문에, 간단히 구현할 수 있다. 반면에, 극단적인 상황에서는 제약 조건이 깨어지는 경우도 발생할 수 있다는 단점을 가진다. 컴퓨터 게임에서는 이러한 극단적인 경우를 피할 수 있는 경우가 대부분이기 때문에, 치명적인 단점이 되지는 않는다.

4. 상업적인 물리 엔진들

3차원에 기초한 컴퓨터 게임들이 늘어나면서, 물리 엔진에 대한 수요가 증가하고, 특히 소규모 게임 제작사들에서는 물리 엔진을 외부에서 구입하는 사례가 증가하고 있다. 최근에는 상당히 많은 수의 물리 엔진들이 상업적으로 판매되고 있는데, 이들은 독립적인 물리 엔진의 형태로 판매되어 다양한 렌더링 엔진들과 결합하여 사용할 수 있는 경우도 있고, 내부 구성 요소의 형태로 대형 게임 엔진에 통합되어 판매되는 경우도 있다.

대표적인 상업적 물리 엔진들로는 Havok 2[10], MathEngine Karma[11], RenderWare Physics[12] 등을 들 수 있다. Havok 2의 경우는 1998년 이후 물리 엔진 분야에 집중해 온 결과물로, 현재 전세계 물리 엔진 시장에서 가장 큰 비중을 차지하고 있다고 알려져 있다. 이 물리 엔진은 이미 50개 이상의 유명 게임 제작사들이 사용한 적이 있거나 사용 중이고, "Max Payne 2", "Half Life 2" 등의 유명 게임 타이틀에 사용되어 일반인들에게도

높은 지명도를 보이고 있다. MathEngine Karma는 대형 게임 타이틀인 "Black and White"에 사용되어 유명해 졌고, RenderWare Physics는 유명한 렌더링 엔진인 RenderWare의 물리 엔진 부분으로 점차 사용자가 증가하고 있다.

특정 분야를 위한 물리 엔진들도 출현하고 있는데, FastCar Library[13]의 경우는 비교적 저가에 적당한 성능을 보이는, 자동차 모션 전용의 물리 엔진을 제공한다. 이렇게 특정 분야에 특화된 물리 엔진의 개발은 점차 그 비중이 커질 것으로 예상된다.

상업적인 물리 엔진 이외에도 프리웨어(freeware)의 형태로, 별도의 비용 없이 사용할 수 있는 물리 엔진들도 출현하고 있다. 가장 유명한 프리웨어 물리 엔진인 Open Dynamics Engine[14]의 경우는 소스 코드까지 공개되어 있으면서도 상업적인 물리 엔진에 필적할 정도의 성능을 보인다. 다만, 상업적인 물리 엔진들이 컴퓨터 게임용의 거의 모든 플랫폼에서 작동하는 데에 반하여, PC의 윈도우(Windows) 운영 체제와 리눅스(Linux) 운영 체제에서만 작동한다는, 게임 개발자 입장에서는 치명적일 수도 있는 단점을 가진다. 이 외에도 DynaMechs[15]를 비롯한 다양한 프리웨어 물리 엔진들이 알려져 있다.

5. 결 론

컴퓨터 게임 산업의 발전에 따라, 개발 툴들에 대한 요구가 커지고 있고, 이러한 흐름 속에서 게임 엔진에의 수요는 계속 증가하고 있으며, 궁극적으로는 거의 모든 상업적 게임들이 게임 엔진을 이용하여 개발되는 추세로 나아갈 것이다. 특히 3차원 게임의 개발에서는 게임 엔진의 기능을 세분화하여, 다양한 요소들이 독립적인 엔진의 형태로 만들어지고 있다.

물리 엔진은 이러한 흐름 속에서 게임 속의 다양한 물체들이 실생활의 물리 법칙이 적용된 사실적인 움직임을 보이도록 함으로써, 컴퓨터 게임의 사실성을 높이는 역할을 담당한다. 물리 엔진의 핵심적인 기능은 고전 역학의 법칙들을 미분 방정식 형태로 해석할 수 있도록 하여, 매 순간마다 물체의 위치와 방향을 새롭게 갱신하여 렌더링 엔진에서 제대로 된 움직임을 보이게 하는 데에 있다. 컴퓨터 그래픽스 분야의 관련 기술들에 실시간 처리와 대화형 처리에 대한 요구 조건들을 추가함으로써 이러한 기능의 구현이 가능해 지고 있다.

컴퓨터 게임들이 점차 3차원의 사실적인 게임으로 바뀌어 가는 추세이고, 이러한 흐름 속에서는 사실적인 모션을 보여주기 위해서는 게임 엔진에의 수요가 증가할 수 밖에 없다. 이미 상업적인 물리 엔진들이 판매되고 있으며, 물리 엔진만을 제작하는 회사들도 출현하고 있다. 물리 엔진 분야는 앞으로도 지속적으로 성장해 나갈 것이며, 더욱 사실적인 모션을 위한 개선 작업과, 특정한 게임 분야만을 위한 특화된 물리 엔진을 제작하는 움직임이 예상된다.

참고문헌

1. Andrew Rollings and Dave Morris, *Game Architecture and Design*, Coriolis, 1999.
2. Alan Watt and Fabio Policarpo, *3D Games: Real-time Rendering and Software Technology*, Addison-Wesley, 2001.
3. David Baraff and Andrew Witkin, "Physically Based Modeling", *SIGGRAPH'03 Course Note #12*, 2003.
4. David M. Bourg, *Physics for Game Developers*, O'Reilly, 2001.
5. David Halliday and Robert Resnick, *Fundamentals of Physics, 2nd Ed.*, John Wiley & Sons, 1981.
6. Richard P. Feynman, Robert B. Leighton and Matthew Sands, *The Feynman Lectures on Physics*, Volume I, Addison-Wesley, 1984.
7. Richard L. Burden and J. Douglas Faires, *Numerical Analysis, 6th Ed.*, Brooks/Cole, 1997.
8. Laurene V. Fausett, *Applied Numerical Analysis Using MATLAB*, Prentice-Hall, 1999.
9. 백낙훈, "충돌 탐지 방법들의 소개", *한국 CAD/CAM학회지*, Vol. 7, No. 3, pp. 47-52, 2001.
10. http://www.havok.com/
11. http://www.mathengine.com/
12. http://www.renderware.com/
13. http://www.oxforddynamics.co.uk/
14. http://www.ode.org/
15. http://dynamechs.sourceforge.net/