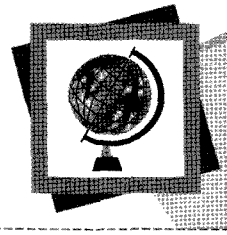
| 특집 04 |



게임 물리의 개요

백 낙 훈
(경북대학교)

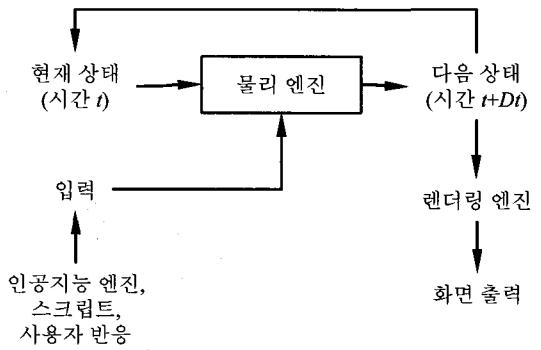
목 차

1. 서 론

3차원 게임에서는 대상이 되는 물체의 움직임들을 자연스럽게 생성하려면, 실세계(real world)의 물리 법칙들을 준수하도록, 또는 그런 것처럼 보이도록 하는 처리가 필요하다. 게임 내에서의 물리 법칙 처리는 궁극적으로 고전 역학 법칙의 구현에 있으므로, 기본적인 구조는 대동소이하다. 컴퓨터 게임 시장의 확대와 함께, 게임 개발 과정에서 게임 엔진의 형태가 광범위하게 쓰이기 시작하면서, 물리 법칙 처리 부분은 물리 엔진(physics engine)으로 제공하는 것이 일반적이다[1]. 따라서, 이 글에서는 게임에서의 물리 법칙에 대한 처리 과정을 물리 엔진의 관점에서 살펴보도록 하겠다.

컴퓨터 그래픽스 분야에서는 이미 물리 기반 모델링(physically-based modeling)에서 이러한 물리 법칙의 처리에 대한 연구 결과가 다양하게 제시되어 있다[2, 3]. 이 분야에서는 처리 시간에 대한 제약이 심하지 않은 반면에, 컴퓨터 게임에서는 어떠한 화면 출력도 정해진 시간 내에 이루어져야 하므로, 실시간 처리를 더욱 강력하게 요구한다. 또, 사용자의 입력에 즉각적으로 반응하는 대화형 처리 방식도 강력히 요구된다. 이러한 측면들을 고려한다면, 게임에서의 물리 법칙 처리는 (그림 1)에서와 같이, 주어진 시간 t 에서의 상태(state)에 게임을 구성하는 다른 부분들에서 들어오는 입력들을 반영하여, 시간 $t+\Delta t$ 에서의 새로운 상태를 물리 법칙에 맞게 계산한 후, 렌더링(rendering) 부분으로 넘기는 역할을 수행한다고 볼 수 있다.

최근에는 3차원 게임의 개발 과정에서 물리 엔진을 사용하는 것이 일반적이다. 물리 엔진(또는 이에 준하는 물리 법칙 처리 과정)을 사용하지 않으면, 물체의 움직임을 표현하는 과정에서 미리 설정한 움직임이 나오게 하는 방식을 택할 수밖에 없기 때문에, 사실성이 떨어지기 쉽다. 반면에, 물리 엔진은 거의 모든 경우에 대



(그림 1) 물리 엔진의 역할

해서 매우 사실적인 움직임을 만들 수 있다는 장점을 가진다. 또, 이미 개발된 물리 엔진을 사용한다면, 게임 개발 과정을 대폭 단축시킬 수도 있다. 물론, 물리 엔진 자체의 개발은 상당한 시간과 비용을 필요로 하지만, 상업적으로 판매되는 물리 엔진을 사용하여 전체 일정과 비용을 줄일 수 있다.

반면에, 물리 엔진이 가져오는 문제점도 몇 가지 있다. 우선, 물리 법칙의 적용은 필연적으로 더 많은 계산 시간을 필요로 하기 때문에, 게임을 구성하는 다른 부분들에서 필요로 하는 시간을 희생해야 할 수도 있다. 또, 역설적으로, 물리 법칙에 위배되는 움직임을 만들어야 하는 경우에는 물리 엔진과는 별도의 예외 처리 부분을 작성해야 하기 때문에 별도의 시간과 비용을 필요로 한다[4].

대부분의 3차원 게임에서는 물리 엔진을 사용함으로써 얻게 되는 이득이 더 많기 때문에, 게임 개발 과정에서 물리 엔진을 채택하는 것이 일반적이다. 이 글은 물리 엔진의 역할과 구조에 대해서 개략적으로 설명하여, 게임 내에서의 물리 법칙 처리 과정을 보이고자 한다. 2절에서는 물리 엔진들에서 필수적으로 처리해야 하는 물리 법칙들을 소개하고, 이로부터, 전형적인 물리 엔진의 구조를 설명하겠다. 3절과 4절에서는 좀더 고급 기능인 충돌 처리와 다관절체 처리에 대해서 간단히 설명하겠고, 5절에서 상업적으로 판매되는 물리 엔진들과 최근의 하드웨어 지원 추세를 소개한 후, 6절에서 결론을 맺겠다.

2. 물리 엔진의 구성

물리 엔진은 근본적으로 물리 법칙들을 컴퓨터 프로그램으로 구현한 결과이다. 화면에 나오는 물체들의 움직임에 가장 큰 영향을 미치는 물리 법칙은 Newton의 제2운동 법칙인 $\mathbf{f} = m\mathbf{a}$가 될 것이다. 즉, 질량이 m인 물체에 가해지는 힘 $\mathbf{f}$는 그 물체에 가속도가 $\mathbf{a}$인 운동을 일으킨다. 시간 t의 흐름에 따른 물체의 속도 $\mathbf{v}(t)$는 가속도 $\mathbf{a}$가 계속 더해진 상태, 즉 가속도의 적분 형태로 표현할 수 있고, 물체의 위치 $\mathbf{x}(t)$는 물체의 속도를 적분해서 구할 수 있다. 이러한 관계를 적분식으로 표현하면 다음과 같다[5, 6].

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

이제 한 물체에 가해진 힘 $\mathbf{f}$는 위의 식들을 차례로 계산해 나감으로써, 최종적으로 물체의 위치 $\mathbf{x}(t)$를 이동시키는 역할을 수행한다. 게임 내에서는 사용자가 마우스를 클릭하거나, 총을 쏘거나, 자동차의 핸들을 꺾거나 하는 다양한 행동들이 모두 각 시간에서의 힘 $\mathbf{f}(t)$를 제어하는 역할을 한다. 반대로, 해당 물체의 입장에서는 어떠한 입력이든 일관되게 힘 $\mathbf{f}(t)$의 형태로 가해진다.

물체의 운동만을 고려한다면, 위의 수식들을 그대로 구현해도 충분하다. 예를 들어, 자유낙하 운동과 같이, 원하는 운동에 대한 정보가 미리 알려져 있고, 운동 중에 다른 힘이 가해지지

않는 경우라면, 해석학 방법으로 적분식들을 미리 풀어서, 다항식 형태로 만든 후에 그 수식을 그대로 적용하는 것도 가능하다. 즉, 자유 낙하하는 물체에 대해서는 중력 가속도 **g**만 적용될 것이고, 많은 물리학 교과서들이 다음과 같은 수식을 제시하고 있다.

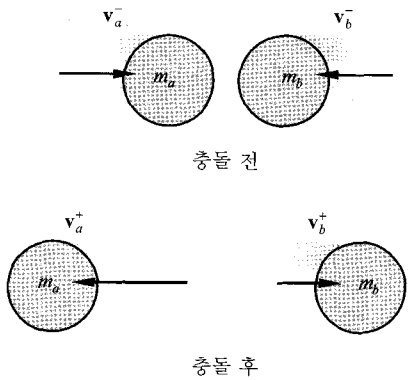
$$\begin{aligned}\mathbf{a}(t) &= \mathbf{g}\\ \mathbf{v}(t) &= \int \mathbf{g}\,dt = \mathbf{g}t\\ \mathbf{x}(t) &= \int \mathbf{g}t\,dt = \frac{1}{2}\mathbf{g}t^2\end{aligned}$$

반면에, 일인칭 슈팅 게임(first-person shooting game)이나 레이싱 게임과 같이, 실시간으로 다양한 형태의 힘들이 가해지는 경우라면, 해석학 방법으로 적분을 구하기가 곤란하기 때문에, 수치해석 방법들로 실시간에 그 해를 구해야 한다. 이 경우에는 식 (1)을 그대로 쓰기 보다는 양변을 시간 t에 대해서 미분해서 다음과 같은 미분 방정식 형태로 바꾸어서 수치해석 방법을 적용하는 것이 유리하다.

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

물리 엔진의 관점에서는 물체의 운동 이외에 고려해야 할 것이 물체들끼리의 충돌에 대한 처리이다. 두 물체가 충돌하는 경우에 대해서는 또 다른 물리학 법칙인 운동량 보존(conservation of momentum) 법칙을 적용해야 한다. (그림 2)에서와 같이, 두 물체 a, b가 충돌하였을 때, 각각의 질량을 m_a, m_b라 하고, 충돌 직전의 각 물체의 속도를 $\mathbf{v}_a^-$, $\mathbf{v}_b^-$, 충돌 직후의 속도를 $\mathbf{v}_a^+$, $\mathbf{v}_b^+$라 하면, 운동량 보존의 법칙은 다음의 수식으로 표현할 수 있다:

$$m_a\,\mathbf{v}_a^- + m_b\,\mathbf{v}_b^- = m_a\,\mathbf{v}_a^+ + m_b\,\mathbf{v}_b^+. \qquad (3)$$



(그림 2) 물체의 충돌 전후

즉, 충돌 직전에 두 물체 a, b의 운동량의 합은 충돌 직후의 두 물체의 운동량의 합과 일치한다. 이 법칙에서는 새로운 물리량인 운동량(momentum) **P**에 대한 처리가 필요하다. 운동량(momentum) **P**는 물체의 질량과 속도의 곱으로 정의되고, $\mathbf{P} = m\mathbf{v}$로 계산할 수 있다. 운동량은 그 자체로서도 의미를 가지지만, 이를 미분하면, 뉴튼의 운동 제2법칙 $\mathbf{f} = m\mathbf{a}$를 다르게 표현하는 것이 가능하다. 즉, $\mathbf{P} = m\mathbf{v}$의 양변을 시간 t로 미분하면 다음의 수식을 얻을 수 있다.

$$\frac{d}{dt}\mathbf{P} = m\frac{d}{dt}\mathbf{v} = m\mathbf{a} = \mathbf{f}$$

따라서, $\frac{d}{dt}\mathbf{P} = \mathbf{f}$이고, 이는 $\mathbf{f} = m\mathbf{a}$에 대한 다른 표현으로 볼 수 있다. 물리 엔진의 관점에서는 가속도 **a**를 직접 사용하는 대신, 운동량 **P**를 사용하면, 충돌 처리까지 일관되게 표현할 수 있다는 장점을 가진다. 이제 식 (2)를 가속도 **a**가 아니라 운동량 **P**의 관점에서 다시 쓴다면, 다음과 같은 수식이 된다.

$$\frac{d}{dt}\mathbf{P} = \mathbf{f}$$
$$\mathbf{v} = \frac{1}{m}\mathbf{P}$$
$$\frac{d}{dt}\mathbf{x} = \mathbf{v} = \frac{1}{m}\mathbf{P}$$

위의 식에서 속도 $\mathbf{v}$에 대한 수식은 계산 과정에서 나온 것일 뿐이므로, 꼭 필요한 미분 방정식만을 남긴다면, 다음의 형태가 된다.

$$\frac{d}{dt}\mathbf{P} = \mathbf{f}$$
$$\frac{d}{dt}\mathbf{x} = \mathbf{v} = \frac{1}{m}\mathbf{P} \qquad (4)$$

수치 해석 방법으로 위의 미분 방정식들에 대한 해를 구한 후에는 필요하다면 가속도와 속도를 다음과 같이 구할 수 있다.

$$\mathbf{a} = \frac{1}{m}\mathbf{f} = \frac{1}{m}\left(\frac{d}{dt}\mathbf{p}\right)$$
$$\mathbf{v} = \frac{1}{m}\mathbf{P}$$

물리 엔진은 식 (1), (2), (4) 중의 어느 형태로든 구현할 수 있지만, 수치해석 방법들에서의 효율성을 고려해서 식 (4)를 쓰는 것이 일반적이다.

이제까지는 설명의 편의를 위해서 직선 운동에 대해서만 고려했는데, 3차원 물체에 대해서는 물체 자체의 회전에 대한 고려도 필요하다. 고전 역학에서는 이미 직선 운동에 대응되는 회전에 대한 법칙들이 제시되어 있으므로, 이들에 대해서 위와 비슷한 처리 과정을 그대로 적용할 수 있다. 우선, 물체의 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 2차 텐서(tensor), 즉 3×3 행렬 형태로 표현된다. 관성 모멘트를 정확하게 계산하려면 꽤 복잡한 적분식을 풀어야 하지만, 일반적인 게임에서는 정밀한 운동보다는 그럴싸하게 보이는 정도면 충분하기 때문에, 비교적 계산이 간단한, 주어진 물체를 둘러싸는 원기둥이나 구에 대한 관성 모멘트를 계산해서 근사값으로 사용해도 충분하다.

회전 운동 시에, 물체의 방향(orientation)은 쿼터니언(quaternion) $\mathbf{q}(t)$로 표현하는 것이 효과적이다. 쿼터니언은 복소수를 3차원으로 확장한 개념으로, 3차원 상에서의 특정한 방향을 1개의 쿼터니언으로 표현할 수 있다. 물체의 회전에 대한 각속도(angular velocity) $\boldsymbol{\omega}(t)$는 3차원에서의 회전 중심축에 회전 속도를 곱한 값이다. 즉, 각속도 $\boldsymbol{\omega}(t) = (\omega_x, \omega_y, \omega_z)$는 그 방향이 회전축 방향이고, 크기 $|\boldsymbol{\omega}(t)|$가 회전 속도인 3차원 벡터로 정의된다. 시간 t에 대한 물체의 방향 변화 $\frac{d}{dt}\mathbf{q}$는 다음과 같이 각속도 $\boldsymbol{\omega}(t)$와 방향 $\mathbf{q}(t)$에 대한 식으로 계산할 수 있다:

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

여기서, $[0,\boldsymbol{\omega}(t)]$는 쿼터니언 계산의 편의를 위해, 각속도 $\boldsymbol{\omega}(t)$를 쿼터니언 형태로 표현한 것을 의미한다.

각운동량(angular momentum) $\mathbf{L}$은 운동량에 대한 대칭 개념으로, 다음과 같이 관성 모멘트 $\mathbf{I}$와 각속도 $\boldsymbol{\omega}$의 곱으로 표현할 수 있다.

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

이 수식은 직선 운동에서의 운동량 $\mathbf{P} = m\mathbf{v}$와 대칭성을 가지고, 이를 미분하면 힘 $\mathbf{f}$에 대응되는 토크(torque) $\boldsymbol{\tau}$가 된다. 이제 운동량을 미분

방정식 형태로 해석했던 것과 같은 과정을 적용하면, 회전 운동은 각운동량에 대해서 다음의 미분방정식들로 표현할 수 있다.

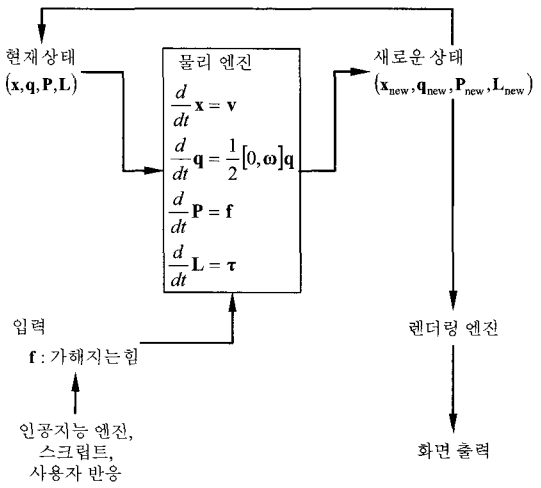
$$\frac{d}{dt}\mathbf{L} = \boldsymbol{\tau}$$
$$\frac{d}{dt}\mathbf{q} = \frac{1}{2}[0, \boldsymbol{\omega}]\mathbf{q}.$$

이제 직선 운동과 회전 운동을 한꺼번에 처리하기 위해서는, 대상 물체의 위치(position) $\mathbf{x}$, 방향(orientation) $\mathbf{q}$, 운동량(momentum) $\mathbf{P}$, 각운동량(angular momentum) $\mathbf{L}$에 대한 다음의 미분방정식들을 수치해석 방법으로 풀면 된다.

$$\begin{aligned}
\frac{d}{dt}\mathbf{x} &= \mathbf{v} \\
\frac{d}{dt}\mathbf{q} &= \frac{1}{2}[0, \boldsymbol{\omega}]\mathbf{q} \\
\frac{d}{dt}\mathbf{P} &= \mathbf{f} \\
\frac{d}{dt}\mathbf{L} &= \boldsymbol{\tau}.
\end{aligned} \tag{5}$$

물리 엔진은 (그림 3)에서와 같이, 정해진 시간 간격마다 이 미분 방정식을 해석하여, 시간 $t+\Delta t$에서의 새로운 상태 $(\mathbf{x}_{\text{new}}, \mathbf{q}_{\text{new}}, \mathbf{P}_{\text{new}}, \mathbf{L}_{\text{new}})$를 계산한다. 짧은 시간 간격 Δt마다 이 계산을 수행하여 그 결과를 누적해 나가면, 해당 물체는 결과적으로 물리 법칙에 따라 운동하는 것처럼 보이게 된다.

미분 방정식의 해석에 있어서는 다양한 수치해석 방법들이 사용될 수 있는데, 식 (5)의 미분방정식들은 전형적인 상미분 방정식 형태이므로, Euler method나 Runge-Kutta method를 사용할 수 있다[7, 8]. Euler method보다 Runge-Kutta method가 더 복잡한 계산을 필요로 하지만, 계산의 정확도가 높기 때문에, 최근에 와서는 게임에 사용되는 PC들의 성능이 높아지면서, 점차 Runge-Kutta method를 선호하는 추세이다.



(그림 3) 물리 엔진에서의 처리 과정

3. 충돌 및 다관절체 처리

물리 엔진의 가장 기본적인 기능인, 물체의 사실적인 움직임을 생성하는 것은 앞 절에서 설명한 바와 같이, 물리 법칙들의 미분 방정식을 해석하는 과정으로 구현할 수 있다. 반면에, 3차원 상에서 물체들이 사실적으로 움직이기 위해서는 추가로 충돌 처리(collision handling)가 반드시 구현되어야 한다. 이 과정은 좀더 자세하게는, 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 모델링에서의 기법들이 적용되어야 하고, 이 때문에, 일반적인 물리 엔진은 충돌 탐지 기능과 충돌 처리 기능을 모두 제공한다. 컴퓨터 그래픽스 분야에서는 이미 충돌 처리 전반에 걸친 많은 연구 결과들이 제시되어 있다.

충돌 반응에서 가장 핵심이 되는 것은 이미 앞 절에서 설명한 운동량 보존 법칙이다. 식 (3)

에서는 운동량이 완전히 보존되는, 완전 탄성 충돌을 표현했지만, 실세계에서의 충돌은 소리나 물체의 내부 변형 등으로 전체 운동량의 일부가 소실되어, 다음과 같은 비탄성 충돌이 된다.

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

여기서, E는 비탄성 충돌에서 소실되는 운동량을 의미한다. 실제 게임 엔진에서는 충돌 시에 적당한 양의 운동량을 감소시키는 대신, 대응되는 양 만큼의 효과음이나 불꽃과 같은 특수 효과(special effect)를 적용시킬 수 있다.

충돌 반응을 처리하기 위해서는 충격량(impulse)의 개념을 사용한다. 충격량은 운동량의 변화로 정의되고, 물체 a에 가해지는 충격량은 다음과 같이 계산한다.

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

물체 b의 경우는 운동량 보존 법칙에 따라, 물체 a에 가해진 충격량의 부호만 바뀐 충격량이 가해지기 때문에, 별도로 계산하지는 않고, 최종 계산 결과의 부호만 바꾸어서 적용하면 된다.

위에서 정의한 충격량은 충돌 전후에 물체의 속도를 순간적으로 바꿀 정도로 매우 큰 힘이 아주 짧은 시간 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:

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

이러한 계산 방식은 물리 엔진에서 이미 계산된 힘 $\mathbf{f}$에 impulse force $\mathbf{F}$만 추가함으로써 충돌을 처리할 수 있다는 장점을 가진다.

결과적으로 충돌 반응 단계에서는 충격량 $\mathbf{J}$를 계산한 후, 다음의 식으로 각 물체의 운동량을 새로 계산한다:

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

경우에 따라서는 이러한 간접 계산 대신, 물체의 속도를 아래와 같이 직접 변화시키는 방법을 사용하기도 한다:

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

물리 엔진은 위와 같은 방식으로 3차원 물체들 간의 충돌에 따른 변화를 처리할 수 있다.

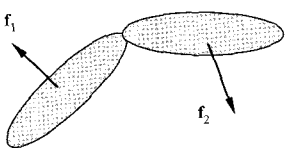
4. 다관절체 처리

물리 엔진에서 다루는 물체는 단일 강체를 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.

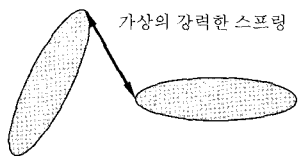
따라서, 물리 엔진에서는 2개 이상의 강체가 관절로 연결된 다관절체에 대한 처리가 필수적이다. 물론 각 강체의 처리는 기존의 물리 법칙을 준수하도록 각각 계산할 수 있지만, 강체들 간의 연결 상태가 반드시 유지되도록 하는 추가 처리가 필요하다. 컴퓨터 그래픽스 분야에서는

constrained dynamics 분야에서 이러한 다관절체를 다루고 있고, 물리 엔진에서도 이들의 연구 결과를 사용한다.

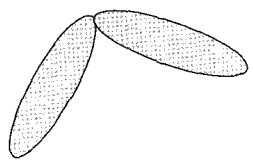
Constrained dynamics 분야에서는 다양한 방법으로 다관절체를 처리하고 있는데, 그 모두를 소개하기는 무리이고, 개념적으로 비교적 간단한 처리 방법인 penalty method를 살펴 보겠다. 이 방법은 관절과 같이, 물체에 주어진 제약 조건(constraint)을 직관적으로 해결한다. 즉, 각 관절 부위에 가상의 매우 강력한 스프링을 삽입하여, 관절 부위에서 두 물체가 떨어지려고 하면, 이 가상의 스프링이 매우 강력한 힘으로 잡아당겨서 강제로 연결을 유지하게 하는 방법이다. 이 방식은 별다른 추가 처리 없이 가상의 스프링을 처리할 수 있다는 점 때문에, 거의 모든 물리 엔진들이 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.



(a) 각각의 물체에 힘이 가해진다.



(b) 관절 부분에 강력한 스프링을 적용



(c) 관절이 연결된 최종 상태

(그림 4) penalty method의 적용

5. 상업적인 물리 엔진들과 PPU

3차원 게임들이 늘어나면서, 게임 제작사들이 물리 엔진을 외부에서 구입하는 것이 일반화되고 있다. 현재 상업적으로 판매되는 물리 엔진들은 대형 게임 엔진을 구성하는 하나의 구성 요소로 판매되기도 하고, 독립적인 물리 엔진의 형태로 판매되어 다양한 렌더링 엔진들과 결합하여 사용되기도 한다. 어느 경우든, 상업적으로 판매되는 물리 엔진들은 PC, Xbox, PlayStation 등의 다양한 플랫폼들을 모두 지원하는 것이 일반적이다.

대표적인 상업적 물리 엔진들로는 Havok[9], Karma[10], Ageia PhysX[11] 등을 들 수 있다. Havok의 경우는 1998년 이후 물리 엔진 분야에 집중해 온 결과물로, 전세계 물리 엔진 시장에서 상당한 비중을 차지하고 있다. 이 물리 엔진은 이미 많은 유명 게임 제작사들이 사용 중이고, "Halo 2", "Max Payne 2", "Half Life 2" 등의 유명 게임 타이틀에 사용되어 일반인들에게도 높은 지명도를 보이고 있다. Karma는 원래 MathEngine에서 사용한 물리 엔진이었고, "Black and White"에 사용되어 유명해 졌다. 현재는 Unreal Engine 2에 통합되어, 내부 물리 엔진으로 사용되고 있다. Unreal Engine의 사용이 늘어나면서, Karma의 사용 역시 상당히 늘어난 상태이다.

Ageia사의 PhysX는 지명도가 높은 편은 아니지만, 최초로 멀티스레드(multithread) 방식으로 물리 엔진을 구현하였고, 하드웨어로 구현된 PPU(physics processing unit)을 지원하는 방식을 택해서, 새로운 방향을 제시하고 있다. 이 물리 엔진은 Unreal Engine 3에서 지원될 예정이고, 다른 제작사들에서도 많은 관심을 보이는 상황이다.

최근에 물리 엔진 분야에서는 하드웨어의 지원을 받는 방법에 대한 시도들이 이루어지고 있

다. PlayStation 3나 Xbox 360과 같은 차세대 게임기들은 멀티프로세서 구조를 가지므로[12, 13], 최소한 1개의 프로세서를 물리 엔진 전용으로 할당할 수 있다. 이렇게 되면, 물리 엔진에서 처리할 수 있는 계산량이 상당히 늘어나므로, 더욱 사실적인 장면을 생성하는 것이 가능할 것이다.

PC 기반의 플랫폼에서도 멀티코어를 지원하는 CPU들이 늘어나면서, 멀티스레드로 구현된 물리 엔진에서는 비슷한 효과를 기대할 수 있다. 또, 그래픽스 카드들이 별도의 GPU를 채택하여 성능 향상을 가져왔던 것과 유사하게, PPU를 채택해서 상당한 성능 향상을 가져올 수 있을 것이다. 이 경우, 게임 엔진 전체로 보아서는 게임 진행이나 인공 지능 처리는 CPU에서 담당하고, 렌더링 엔진은 그래픽 카드의 GPU를 사용하면서, 물리 엔진은 PPU를 사용하게 하여, 게임 엔진 전체의 성능을 높이게 될 것이다. 최초의 PPU는 Ageia 사에서 2002년부터 개발에 착수하여, 현재는 개발이 완료된 상태이고, 2005년 GDC와 E3 show에서 발표한 상황이다[14]. 시장에는 2005년 말이나 2006년 초에 Add-on card 형태로 출시될 예정이다. 또한, 일부 마더보드 제작사들이 이 PPU를 지원하는 마더 보드를 출시할 예정이기 때문에, PPU가 GPU처럼 대중화될 가능성이 높아지고 있다.

Ageia 사의 개발 과정과는 무관하게, 한국전자통신연구원에서는 2004년부터 이것과는 다른 관점에서의 PPU 개발에 착수하여 2005년 연말에 프로토타입이 완성될 예정이다[15]. 이 PPU는 현재는 충돌 처리의 가속화에 초점을 맞추었고, 지속적인 기능 추가를 통해서, 2006년에는 강체에 대한 물리 계산을 가속하는, 본격적인 PPU 처리가 가능해 지는 것을 목표로 하고 있다. 최종적으로는 유체(fluid)와 변형 가능한 물체(deformable body)에 대한 처리까지 가능해 질 예정이다.

6. 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 매 순간마다 적용하여, 렌더링 엔진에서 제대로 된 움직임을 보이도록 하는 데 있다. 컴퓨터 그래픽스 분야의 관련 기술들에 실시간 처리와 대화형 처리에 대한 요구 조건들을 추가함으로써 이러한 기능의 구현이 가능해졌다.

컴퓨터 게임들은 이미 3차원의 사실적인 게임으로 바뀌어 가는 추세이고, 이러한 흐름 속에서 사실적인 모션을 보여주기 위해서는 물리 엔진에의 수요가 증가할 수밖에 없다. 이미 상업적인 물리 엔진들이 광범위하게 채택되고 있으며, 하드웨어로 이를 지원하는 추세도 일반화되고 있다. 물리 엔진의 하드웨어 가속을 위한 PPU도 출시될 예정이기 때문에, 물리 엔진의 성능은 빠른 시간 내에 더욱 향상될 전망이다. 물리 엔진 분야는 앞으로도 지속적으로 성장해 나갈 것이며, 더욱 사실적인 모션을 위한 개선 작업과 하드웨어의 지원 등과 같은 가속 기법들의 채택이 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] http://www.havok.com/

[10] http://udn.epicgaems.com/

[11] http://www.ageia.com/

[12] http://www.playstation.com/

[13] http://www.microsoft.com/xbox/

[14] http://www.gamershell.com/articles/901.html

[15] personal communication, 김도형(한국전자통신연구원 디지털콘텐츠연구단 디지털액터연구팀)

저자약력

백 낙 훈

1990년 한국과학기술원 전산학과(학사)
1992년 한국과학기술원 전산학과(석사)
1997년 한국과학기술원 전산학과(박사)
1997년 George Washington University, Visiting Scholar
1998년 경북대학교 전자전기공학부 초빙교수
2001년~2004년 ㈜케이오지 CTO
2004년~현재 경북대학교 전자전기컴퓨터학부 조교수
e-mail : oceancru@yahoo.co.kr