가상 벽과 충격 모델에 기반한 단순하지만 효과적인 레이싱 게임용 차량 바퀴 시뮬레이션 기법

강 영 민*

Simple but Effective Vehicle Wheel Simulation based on Imaginary Wall and Impulse Model for Racing Game

Young-Min Kang*

요 약

자동차 경주 게임은 실시간에 시뮬레이션 될 수 있는 사실적인 물리 모델을 요구한다. 자동차 경주 게임에서는 움직임의 사소한 오류도 쉽게 눈에 띄며, 다른 게임과 같이 상호작용적으로 동작해야 한다. 타이어와 지면의 정확한 물리를 모델링하여 이를 실시간 환경에 구현하는 것은 어려운 일이다. 이 논문에서는 효율적이며 효과적인 "가상 벽(imaginary wall)" 모델을 제안한다. 이 기법은 사용된 물리 모델의 단순성 때문에 쉽게 구현이 가능하며, 시뮬레이션의 결과가 자동차 경주 게임에 사용하기에 충분한 사실성을 가진다.

ABSTRACT

Racing game requires plausible physics model that can be simulated in realtime. Minor artifacts in racing games are easily noticed, and any kinds of games should work interactively. It is difficult to model the accurate tire-ground physics and to integrate the model into realtime environments. In this paper, an efficient and effective "imaginary wall" model was proposed. The method can be easily implemented because of the simplicity of the physical model used, and the result of the simulation is realistic enough for the racing games.

키워드

실시간 환경, 게임, 레이싱 게임, 물리기반 모델링, 충격 모델, 가상 벽

Ⅰ. 서 론

자동차 경주 게임을 위한 시뮬레이션 기법은 *정확성*과 *효율성*이라는 두 가지 모순적인 목표를 추구한다. 일반적으로 사람들은 자동차의 움직임에 매우 익숙하기 때문에 자동차 경주 게임에서 나타나는 자동차의 움직임에 사소한 오류가 있을 경우 쉽게 이를 인식하게 된다. 더구나 게임에 있어 상호작용성은 필수적인 것이므로 자동차 경주 게임은 매우 사실적인 물리 모델을 실시간에 시뮬레이션 해야 한다.

게임 환경에서 사실적인 자동차 경주 장면을 생성하기 위해서는 자동차의 역학적 모델이 우선적으로 요구되며 이 역학적 모델을 물리적 법칙에 따라 시뮬레이션 하게 된다. 가상 객체의 움직임을 물리적 사실성을 갖도록 생성하는 기법을 물리 기반 모델링이라고 하며, 이 분야는 컴퓨터 그래픽스에서 가장 중요한 주제들 가운데 하나로

* 동명대학교 정보통신대학 게임공학과

접수일자 : 2006. 7. 26

자리 잡고 있으며, 입자, 강체, 변형 가능한 객체 등을 시뮬레이션 하기 위한 다양한 방법들이 잘 정형화되어 많은 게임 프로그램들에서 널리 활용되고 있다.

자동차 경주 게임에 나타나는 가상 차량들은 강체 모델을 이용하여 적절히 모델링할 수 있다. 강체의 움직임과 상호 접촉, 충돌 반응 등은 매우 잘 정리되어 있는 문제이며, 강체로 구현된 가상 차량의 사실적인 움직임 역시 쉽게 계산하여 가시화할 수 있다.

물리 기반 모델링을 사용할 때, 자동차의 움직임은 자동차의 바퀴와 지면 사이의 상호작용에 의해 크게 좌우된다. 따라서 자동차 시뮬레이션에서 가장 중요한 요소는 자동차 바퀴가 지면에서 어떤 행동을 보이는지를 계산하는 것이다.

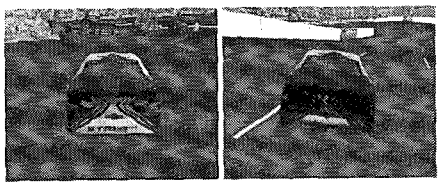

그림 1. 실험을 위해 개발된 레이싱 게임 환경
Fig. 1 Experimental Racing Game

바퀴는 일반적으로 타이어로 쌓여 있으며, 이 타이어들은 바퀴가 주행 경로에서 벗어나 미끄러지지 않도록 하는 역할을 수행한다. 이러한 효과는 타이어와 바퀴 사이의 강한 마찰력을 통해 시뮬레이션할 수 있다.

마찰력은 차량이 지면에 작용하는 힘과 연관되어 있다. 그러나 차량 바퀴가 지면에 가하는 힘은 차량의 무게가 각각의 바퀴에 어떻게 분산되어 있는지를 계산해야 하기 때문에 간단히 계산되지 않는다. 더구나, 마찰력 모델을 사용할 때 발생할 수 있는 문제는 계산된 마찰력이 충분히 크지 않을 경우에는 자동차 바퀴의 미끄러짐이 어느 정도 발생할 수 있다는 것이다. 그런데 타이어와 지면이 서로 미끄러지지 않도록 충분히 큰 마찰력을 계산하는 것이 그리 쉬운 일이 아니다. 연구자들은 몇 가지 복잡한 모델을 제안하여 타이어와 지면 사이의 마찰을 시뮬레이션하려고 하였지만[1, 2], 이러한 모델은 계산 복잡도와 구현상의 어려움 때문에 게임 환경에 적용하기가 쉽지 않다.

실시간 타이어 시뮬레이션은 어려운 문제로 여겨지고 있으며, 준-경험적(semi-empirical) 모델들이 일반적으로 사용된다[3]. 그러나 사실상 업계 표준이라 할만한 "파세이카 모델(Model of Pacejka)"에 사용되는 파라미터들은 지나치게 복잡하며 직관적으로 수정하기가 쉽지 않다[3]. 따라서 게임과 같은 환경에서는 적절한 단순화가 필요하다. 단순화를 통해 얻어진 모델은 계산의 효율이 높고 구현도 용이하지만 극도로 사실적인 자동차 주행 동작을 생성하는 데에는 제약을 가질 수밖에 없다. 이 논문은 효율적이면서 효과적인 차량 시뮬레이션 기법을 제공하여, CPU가 제한적인 시스템에서도 매우 사실적인 레이싱 게임을 구현할 수 있게 한다.

II . 단순한 접근 방법의 문제

바퀴의 움직임을 표현하는 데에 필요한 정확한 물리 모델을 시뮬레이션하기 위해서 요구되는 높은 수준의 계산 비용을 피하기 위하여 게임 환경에서는 단순한 형태의 운동학적 접근법이 종종 사용되기도 한다. 이 절에서는 차량 애니메이션 문제에 적용할 수 있는 두 가지 효율적인 방법을 소개하고 이들 기법의 문제점을 살펴볼 것이다.

첫 번째 방법은 사용자가 원하는 주행 경로를 따라 차량이 정렬되도록 하기 위해 차량이 가져야 하는 각속도를 계산하는 방법이다[4].

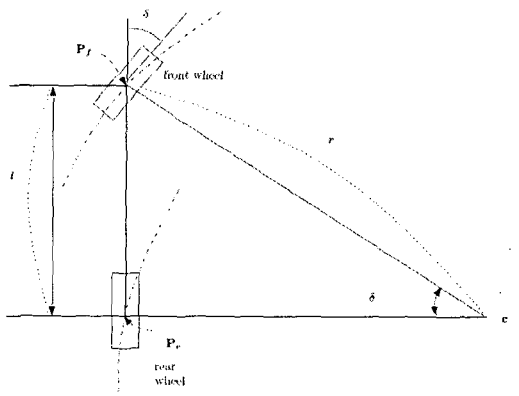

그림 2. 바퀴의 궤적
Fig. 2 Trajectory of Wheels

그림 2에 나타나 있는 것과 같이 앞바퀴의 중심점인 P_f와 뒷바퀴의 중심점 P_r 사이의 거리가 l로 나타나 있으며, 앞바퀴 진행 방향과 차량이 바라보고 있는 방향이

이루는 각도를 δ라고 하자. 앞바퀴와 뒷바퀴가 동일한 차량에 붙어있기 때문에 이 두 바퀴는 동일한 회전 중심인 c 를 기준으로 돌게 된다. 주행 방향을 결정하기 위해서는 이 c 를 중심으로 차량이 회전하는 각속도를 구하기만 하면 되는 것이다. 이 각속도는 차량의 속도와 앞바퀴가 그려내는 원의 원주(circumference)를 알면 구할 수 있다. 차량의 속도는 이미 알려져 있거나 매우 간단히 구할 수 있는 값이므로 원주만 계산하면 된다. 앞바퀴와 회전중심 사이의 거리가 r 이라면 이 원주의 길이는 $2\pi r$ 이 되는데, 이 r 은 다음과 같이 구할 수 있다.

$$r = \frac{l}{\sin\delta} \tag{1}$$

속도가 v 인 차량이 회전 중심 c 를 기준으로 전체 원주 C 를 돌아 원래의 위치로 오는 데에 걸리는 시간은 간단히 $C/|v|$ 로 구할 수 있으며, 이 값은 다시 표현하면 $2\pi r/|v|$ 가 된다. 차량의 각속도 ω 는 360도, 즉 2π 를 도는 데에 필요한 시간 $2\pi r/|v|$ 로 나누면 된다. 즉 차량의 각속도는 다음과 같다.

$$\omega = 2\pi/(2\pi r/|v|) = |v|/r \tag{2}$$

식 2에서 구한 각속도를 적용하면 차량의 방향이 자연스럽게 변경되어 앞바퀴가 향한 방향으로 부드럽게 주행 방향이 변경되게 된다.

다른 한 가지 방법은 차량이 앞바퀴가 향하고 있는 방향으로만 가속된다고 가정하는 방법이다[5]. 이 모델은 지나친 단순화로 사실적인 시뮬레이션일 불가능하다.

운동학적 접근법이 실시간 환경에서도 잘 동작할 수 있을 정도로 효율적이기는 하지만, 고급 수준의 자동차 게임은 이러한 단순한 모델을 통해 간단히 구현될 수가 없다. 이러한 운동학적 기법의 첫 번째 약점은 이들 기법에서는 자동차의 모든 바퀴가 지면과 항상 닿아 있다는 가정을 하고 있다는 것이며, 각각의 개별적 바퀴들의 움직임을 독립적으로 시뮬레이션할 수가 없다는 것이다. 실제의 경우 차량의 바퀴는 지면과 종종 떨어지기도 하며, 이렇게 지면에서 떨어진 바퀴는 주행 방향을 결정하는 데에 아무런 역할을 수행하지 않아야만 사실적인 주행 시뮬레이션이 가능하다. 따라서 차량 바퀴가 개별적으로 시뮬레이션 되지 않는 모델에서는 흔들리거나 덜컹거리면서 주행하는 사실적인 차량 시뮬레이션이 불가능하다는 것이다.

각속도를 계산하여 주행 방향을 변경하는 모델에는 또 한 가지 심각한 약점이 있다. 식 2에 사용되는 주행 반경 r 의 값이 매우 큰 값이 될 수 있다는 것이다. 이 반경이 매우 큰 값이 되는 경우는 δ가 매우 작은 값을 갖는 경우로, 이는 차량의 바퀴를 틀지 않고 차량이 앞으로 곧게 전진하는 경우이다. 예를 들어 차량 앞바퀴가 차량의 전진 방향과 정확히 일치할 경우 반경 r 은 무한대 값이 되어 구하는 것이 불가능하다. 실제 차량 주행에서 대부분의 시간은 이 δ가 작은 값을 갖는다. 따라서 이 모델을 실제로 적용하는 것은 그리 간단한 문제가 아니며, 차량이 전면을 향하면서 미세하게 주행 방향을 바꾸는 가장 일반적인 경우를 제대로 시뮬레이션 하지 못한다.

Ⅲ . 충격 생성자로서의 바퀴 모델

단순화된 접근법들이 가지는 약점들 때문에 사실적인 레이싱 시뮬레이션을 저비용의 계산 부담으로 구현하는 것이 쉽지 않다. 이 절에서는 높은 수준의 레이싱 게임을 가능한 최소의 비용으로 생성할 수 있는 모델을 제안한다. 제안되는 기법은 물리 기반 모델링 기법을 활용하여 게임의 물리적 사실성을 유지하지만 계산 비용은 단순하면서도 효과적인 "*가상 벽 모델(imaginary wall model)*"을 통해 크게 낮출 수 있다.

효율적인 시뮬레이션을 위하여, 차량은 4 개의 바퀴가 부착된 강체 모델로 구현되었다. 차량의 움직임은 강체 시뮬레이션 기법에 의해 대부분 결정되며, 바퀴는 바퀴와 지면이 만나는 위치를 판단하는 것에 사용되며, 원하는 주행 경로를 따르기 위해 차량에 충격을 가하는 역할만 수행한다. 차량의 바퀴는 바퀴가 구르는 방향으로는 낮은 마찰력을 가지지만, 주행 방향에서 어긋나는 쪽으로 미끄러지려고 할 때는 강력한 마찰력을 발생시키도록 모델링 되었다. 따라서 차량 바퀴의 회전이나 바퀴와 차량을 연결하는 조인트 등에 대한 물리 시뮬레이션은 수행하지 않는다. 제안된 기법에서는 차량 바퀴의 회전 운동이 단순히 낮은 마찰을 적용하고 바퀴가 향하는 방향으로 차량을 가속하는 것으로 모델링되었다. 차량 바퀴 진행 방향 이

외의 어떤 방향으로든 바퀴가 미끄러져서는 안되며 이를 위하 강한 마찰력이 발생하여 즉각적으로 차량을 붙잡아 둘 수 있어야 한다. 이런 종류의 강한 힘이 매우 짧은 시간에 순간적으로 객체의 속도를 변화시키는 현상은 충격으로 표현된다.

두 객체가 서로 충돌할 때, 충돌하는 두 객체가 서로 뚫고 들어가지 않도록 막아주는 충격량을 계산할 수 있다. 이 충격량을 계산하기 위하여 몇 가지 중요한 요소들을 정의하자. 객체 A와 B의 질량을 각각 m_a, m_b라고 하자. 두 객체가 충돌하는 위치는 p_c로 정의되며, 두 객체의 무게 중심은 각각 x_a와 x_b이다. x_a에서 시작하여 충돌 위치 p_c로 가는 벡터를 r_a로 정의하며, r_b 역시 비슷한 방식으로 정의된다. $\hat{n}$은 충돌 법선 벡터이며 두 개의 충격량 가운데 하나는 이 충돌 법선 벡터 방향으로 작용하여 객체 B의 속도를 변경하며 다른 하나는 반대 방향으로 작용한다. 두 객체가 충돌 위치에서 가지는 속도를 각각 $v_a^{P_c}$와 $v_b^{P_c}$로 표현한다. 두 강체에 작용하는 충격량을 계산하는 방법은 매우 잘 연구되어 있으며[6, 7, 8], 그 값은 간단히 계산할 수 있다. 객체 B에 작용하는 충격량을 J_b라고 하면, 이 값은 다음과 같다.

$$J_b = \left(\frac{-(1+\epsilon)(v_b^{P_c} - v_a^{P_c}) \cdot \hat{n}}{\hat{n}^T (K_a + K_b)\hat{n}}\right)\hat{n} \tag{3}$$

이때 ϵ은 탄성 계수이며, K 는 각각의 객체가 가지는 특성을 표현하는 행렬로서 $E/m + r^* I^{-1} r^*$ 로 계산된다. 이때 E 는 항등행렬이며, I 는 관성 텐서(inertia tensor), r^*는 벡터 r 의 외적 행렬이다. 즉 K_a와 K_b는 다음과 같이 계산할 수 있다.

$$K_a = E/m_a + r_a^* I_a^{-1} r_a^*$$
$$K_b = E/m_b + r_b^* I_b^{-1} r_b^* \tag{4}$$

바퀴는 구르는 방향으로 움직여야 하며 이 방향을 벗어나려는 움직임을 막아야 한다. 이런 상황은 바퀴가 뚫고 들어갈 수 없는 가상의 벽이 존재하는 것처럼 생각할 수 있다. 잘 정리되어 있는 충격 모델을 이용하여 침투가 일어나지 않는 강체 시뮬레이션을 구할 수 있으므로, 이러한 충격 모델을 가상의 벽에 적용하여 주행 경로를 효과적으로 제어할 수 있다.

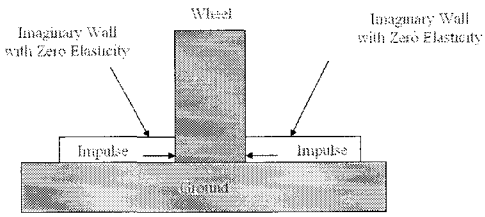

그림 3. 바퀴 시뮬레이션을 위한 가상 벽 모델
Fig. 3 Imaginary Wall Model for Wheel Simulation

그림 3은 가상 벽 모델의 개념을 시각적으로 보여주고 있다. 그림에 나타나 있는 바퀴는 가상의 벽 사이에서 앞뒤로 부드럽게 움직일 수 있다. 가상의 벽은 고정적인 위치에 있는 것이 아니라 바퀴를 따라가며 바퀴의 방향에 따라 자신의 방향을 지속적으로 바꾸는 객체이다. 가상 벽이 수행하는 유일한 물리적 작용은 바퀴가 자기 자신을 뚫고 들어오는 것을 막도록 충격을 생성하는 것이다.

차량의 속도가 충분히 커서 차량이 노면에 미끄러지는 현상이 발생하는 경우가 아니라면, 차량의 바퀴는 이 가상 벽을 뚫고 들어가서는 안 된다. 이러한 모델은 바퀴가 가상 벽과 지속적으로 충돌하는 것으로 모델링할 수 있으며, 이 충돌은 충격량에 의해 쉽게 처리할 수 있다. 이때 바퀴는 두 종류의 충격량을 생성한다. 하나는 지면으로부터 가해지는 충격량이며 (그림 4 (a)), 다른 하나는 가상 벽으로부터 가해지는 충격량이다 (그림 4 (b)).

제안된 기법에서, 각각의 바퀴가 가지는 실제 속도를 이용하기 때문에 더욱 사실적인 바퀴 움직임을 생성할 수 있다. 그림 4 (b)에서 바퀴 중심의 속도가 v 로 표현되어 있으며, 네 개의 바퀴가 가지는 속도는 서로 다른 값이 될 수 있다. 속도 v 는 두 개의 성분으로 분해된다. 그 중 하나는

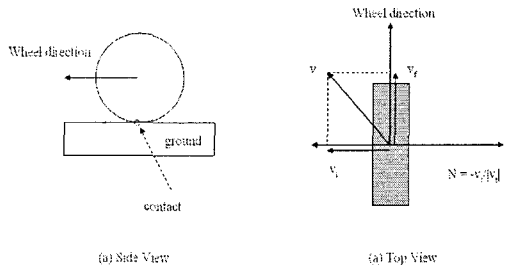

그림 4. (a) 지면과 (b) 가상 벽으로부터의 충격
Fig. 4 Impulses from (a) the Ground, and (b) the Imaginary Wall

바퀴 진행 방향 성분 v_f 이며, 다른 하나는 이에 수직한 성분 v_t 이다. 따라서 가상 벽이 바퀴에 가하는 충격량의 방향은 $-v_t/|v_t|$ 이다. 충격량의 방향은 쉽게 계산할 수 있으며, 이 방향을 충돌 법선 벡터 $\hat{n}$ 으로 사용할 수 있다.

가상 벽과 바퀴의 충돌에서 탄성 계수는 0으로 설정되어야 하는데, 그 이유는 바퀴가 벽을 뚫고 들어갈 수도 없지만, 동시에 바퀴가 벽으로부터 튕겨져 나오는 현상이 발생해서도 안 되기 때문이다. 가상 벽은 고정 객체로 취급할 수 있으므로 K 행렬이 0 행렬이 된다. 따라서 바퀴에 가해지는 충격량 J 는 다음과 같이 쉽게 계산할 수 있다.

$$J = \left(\frac{|v_t|}{\hat{n}^T K_w \hat{n}}\right)\hat{n} = \left(\frac{-v\cdot\hat{n}}{(K_w\hat{n})\hat{n}}\right)\hat{n} \quad (5)$$

이때 K_w 는 바퀴의 중심에서 계산된 K 행렬이다. 각각의 바퀴는 서로 다른 $K_w, \hat{n}, v, v_t$ 의 값을 가지며, 이 값들은 시간에 따라 지속적으로 변화한다.

네 개의 바퀴를 가진 차량을 시뮬레이션 할 때에는 각각의 바퀴가 지면을 뚫고 들어가지 않도록 방지하는 네 개의 충격이 발생하는데 이것 역시 간단히 계산할 수 있다. 가상 벽에 의한 충격과 지면의 충격을 계산하여 차량에 적용하면 차량의 바퀴가 향하고 있는 방향과 완벽하게 일치하는 차량의 주행 움직임을 생성할 수 있게 된다. 더구나 충격량 모델은 차량의 바퀴가 항상 지면에 닿아 있다는 가정을 하지 않으며, 각각의 바퀴는 상호 연관성 없이 독립적으로 시뮬레이션 될 수 있다. 이것은 가상 벽 모델이 운동학적 접근법에 비해 훨씬 더 다양한 상황을 표현할 수 있으며, 차량이 좌우로 흔들리거나 덜컹거리면서 주행하는 모습도 사실적으로 생성할 수 있다는 것을 의미한다.

Ⅳ. 사실적 움직임을 위한 추가적 고려

식 5에 나타나 있는 충격 모델이 적용되면 바퀴가 향하고 있는 주행 방향을 벗어나 바퀴가 미끄러지는 현상이 일어날 수가 없다. 이러한 모델은 저속에서 움직이는 차량의 주행 방향을 제어하는 데에 매우 효과적이다. 그러나 종종 고속으로 이동하는 차량이 차량의 방향을 바꿀 때에 주행 곡선 바깥으로 차가 기울면서 미끄러져 나가는 경우가 종종 발생한다. 이러한 현상이 발생하는 이유는 차량 바퀴의 마찰이 차량의 미끄러짐을 방지할 정도로 충분히 크지 않아 원심력을 이기지 못하기 때문이다. 이러한 현상은 매우 자연스러운 현상임에도 앞서 살펴본 가상 벽 모델은 지나치게 차량의 움직임을 제한하여 이런 현상이 발생하지 않게 된다. 사실적인 자동차 경주 게임을 구현하기 위해서는 차량이 자연스럽게 지면을 미끄러지면서 제어를 잃게 되는 현상을 표현할 수 있어야 한다.

차량이 지면에서 미끄러져 제어를 잃게 되는 드리프팅 (drifting) 현상은 마찰력인 충분하지 않은 경우에 발생한다. 앞서 언급한 바와 같이 제안된 기법은 마찰력을 사용하여 바퀴를 제어하지 않는다. 충격 모델이 이러한 현상을 표현할 수 있도록 하기 위해서는 가상 벽이 바퀴의 침투를 완벽히 막는 것이 아니라 어느 정도의 침투를 허용할 필요가 있다. 다시 말해, 가상 벽이 딱딱한 벽이 아니라 부드러운 재질로 이루어져 차량 바퀴가 벽을 밀고 들어올 수 있도록 해야 하는 것이다. 이러한 부드러운 벽 모델은 탄성 계수를 음(negative)으로 설정함으로써 구현할 수 있다. 직관적인 모델링을 위하여, 드리프팅 파라미터 μ를 정의하였다. 이 파라미터는 0에서 1 사이의 값을 가지는데, 이 값이 0인 경우에 바퀴는 가상의 벽에서 아무런 충격도 받지 않고 지면 위를 미끄러지게 되며, 반대로 이 값이 1인 경우에는 가상 벽을 전혀 뚫고 들어갈 수 없어 바퀴의 방향에 따라 차량이 완벽히 제어된다. 이는 벽의 탄성 계수 ϵ을 $\mu-1$ 로 설정함으로써 쉽게 구현할 수 있으며, 충격량은 다음과 같은 방식으로 쉽게 구할 수 있다.

$$J = \left(\frac{-(1+\epsilon)v\cdot\hat{n}}{(K_w\hat{n})\hat{n}}\right)\hat{n} = \left(\frac{-\mu v\cdot\hat{n}}{(K_w\hat{n})\hat{n}}\right)\hat{n} \quad (6)$$

드리프팅 파라미터는 다음과 같이 자동적으로 계산할 수 있다.

$$\mu = \mu_{max}\left(1 - \frac{|v|}{v_{max}}\right) \quad (7)$$

이때 μ_{max} 는 드리프팅 파라미터가 가질 수 있는 최대값이며, v_{max} 는 차량이 가질 수 있는 최고 속력이며 $|v|$ 는 현재 차량의 실제 속도이다. 이 식은 차량의 드리프팅 경향을 제어할 수 있는 다음과 같은 식으로 수정할 수 있다.

$$\mu = max\ \{\mu_{max}(1 - \frac{|v|}{v_\theta}), 0\} \quad (8)$$

이때 v_θ는 설정이 가능한 임계 속력으로 이 속도 이상이 되면 드리프팅 파라미터가 0이 되어 차량이 제어를 완전히 잃고 미끄러지게 된다.

V. 실험 결과

제안된 기법을 검증하기 위하여 시범적인 레이싱 게임 환경을 개발하였다. 그림 1은 실험에 사용된 레이싱 게임 환경에서 동작하고 있는 가상 객체의 모습을 보여주고 있다. 실험용 게임의 렌더링 환경은 한국전자통신연구원에서 개발한 Dream3D 게임 엔진을 활용하여 구현되었다.

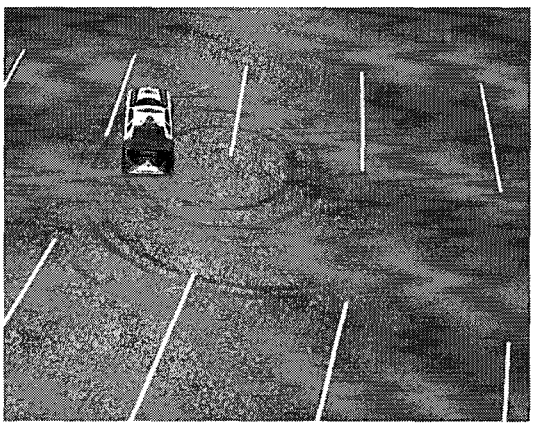

그림 5. 제안된 기법에 의한 주행 제어
Fig. 5 Path Control by the Proposed Method

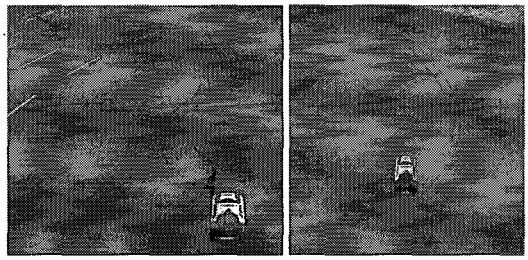

그림 6. 사실적인 시뮬레이션 결과:
(a) 차량의 요동 (b) 드리프팅
Fig. 6 Realistic Simulation Results:
(a) Jolting Vehicle, and (b) Drifting

그림 5는 제안된 기법의 제어능력을 보이고 있다. 차량이 그림에서와 같이 급하게 방향을 바꾸며 회전하는 경우도 잘 생성할 수 있었다. 이때 사용된 주행 트랙은 경사를 가지도록 만들어졌으며 그림에서 볼 수 있는 바와 같이 이 경사면에서 움직이는 차의 실제적인 동작처럼 차량 바퀴가 노면과 떨어졌다 다시 접촉하는 현상이 발생함을 스키드(skid) 마크를 통해 확인할 수 있다. 이와 같이 제안된 기법은 차량 바퀴를 개별적으로 시뮬레이션 함으로써 바퀴와 지면 사이의 사실적인 상호작용을 표현할 수 있다.

그림 6 (a)는 생성된 애니메이션이 매우 높은 수준의 사실성을 가지고 있음을 보이고 있다. 제안된 충격 모델에 의해 독립적으로 시뮬레이션이 된 바퀴들은 그림에서 보이는 것과 같이 지면과 사실적으로 상호작용하며 차량의 덜컹거림과 같은 효과를 잘 표현하고 있다. 식 6에 표현된 드리프팅(drifting) 모델 역시 사실적인 애니메이션 결과를 생성하는 데에 적용되었으며 그림 6 (b)의 차량은 고속으로 주행하도록 가속된 뒤에 오른쪽으로 회전하도록 조작되었다. 그림에서 볼 수 있는 바와 같이 이 차량은 자연스럽게 드리프팅 현상을 보여주면서 회전하였다. 이상의 실험을 통해 "부드러운 가상 벽" 모델을 이용하여 자연스러운 드리프팅 현상을 표현할 수 있었다.

VI. 결 론

본 논문에서는 차량 바퀴 시뮬레이션을 위한 실시간 기법을 제안하였다. 제안된 기법의 물리적 모델은 구현이 용이하도록 단순한 형태를 갖지만, 레이싱 게임에 충분히 적용할 수 있는 수준의 물리적 사실성을 가지고 있다. 간단하면서도 효과적인 이 기법은 바퀴의 움직임을 제어하는 "가상 벽"이라는 개념을 도입하여 구현되었다.

제안된 기법은 충격을 이용하여 물리적으로 시뮬레이션 되었기 때문에 게임 환경에서 충분한 사실성을 보여줄 수 있을 뿐만 아니라, 물리 모델을 단순성 때문에 게임 개발자들이 빠른 시간에 간단히 구현할 수 있다는 장점을 가진다. 제안된 기법을 이용하여 구현한 시범 게임은 실시간 환경에서 사실적 레이싱 애니메이션을 생성하였다. 더구나 이전의 단순한 방법과 달리 제안된 기법은 각각의 바퀴를 독립적으로 시뮬레이션 하기 때문에 평면 지형 등과 같이 제한된 환경 뿐 아니라 다양한 입체 지형 등에서 자연스럽게 이용될 수 있다.

제안된 기법은 CPU가 제한적인 환경에서도 매우 높은 수준의 자동차 경주 게임을 구현하는 데에 성공적으로 적용할 수 있는 기법이다.

참고문헌

[1] Carlos Canudas de Wit and Roberto Horowitz. Observers for tire/road contact friction using only wheel angular velocity information. In *Proceedings of the 38th Conference on Decision and Control*, pages 3932-3937. 1999.

[2] Xavier Claeys, Jingang Yi, Luis Alvarez, Roberto Horowitz, and Carlos Canudas de Wit. A dynamic tire/road friction model for 3d vehicle control and simulation. In *Proceedings of IEEE Transportation Systems Conference*, pages 483-488. 2001.

[3] Szabolcs Deák. Dynamic simulation in a driving simulator game. *In Proceedings of The 7th Central European Seminar on Computer Graphics*, pages 3932-3937. 1999.

[4] Marco Monster. Car physics for games. *http://home. planet.nl/ monstrous*, 1993.

[5] Marcin Pancewicz and Paul Bragiel. Vehicle physics simulation for cpu-limited systems. In Andrew Kirmse, editor, *Game Programming Gems*, pages 221-230. Charles River Media, Hingham, MA, 2004.

[6] David Baraff and Andrew Witkin. Dynamic Simulation of non-penetrating flexible bodies. In *Proceedings of ACM SIGGRAPH 1992*, pages 303-308. 1992.

[7] David Baraff. Fast contact force computation for nonpenetrating rigid bodies. In *Proceedings of ACM SIGGRAPH 1994*, pages 23-34, 1994.

[8] Brian Mirtich and John F. Canny. Impulse-based simulation of rigid bodies. In *Symposium on Interactive 3D Graphics*, pages 181-188. 1995.

[9] Eran Guendelman, Robert Bridson, and Ronald Fedkiw. Nonconvex rigid bodies with stacking. In *Proceedings of ACM SIGGRAPH 2003*, pages 871-878. 2003.

저자소개

강 영 민(Young-Min Kang)

1996 부산대학교 전산학과 이학사
1999 부산대학교 전산학과 이학석사
2003 부산대학교 전산학과 이학박사
2003년~2005년 한국전자통신연구원
2005년~ 동명정보대학교 게임공학과
※관심분야 : 컴퓨터 그래픽스, 물리기반 애니메이션