햅틱 인터페이스 기반의 가상 마리오넷 시뮬레이션

김수정, 장신유, 김영준

이화여자대학교 컴퓨터학과

kimsujeong@ewhain.net, {*zhangxy*|*kimy*}*@ewha.ac.kr*

Virtual Marionette Simulation Using Haptic Interfaces

Sujeong Kim, Xinyu Zhang and Young J. Kim

[1]Dept. of Computer Science and Engineering, Ewha Womans University, Seoul, Korea

요 약

인터랙티브 컴퓨터 게임과 컴퓨터 애니메이션에서, 유관절체의 움직임을 직관적으로 제어하도록 하는 것은 어려운 문제로 인식되고 있다. 이런 분야에서는 대부분 움직임의 대상이 되는 캐릭터가 많은 관절로 연결되어 있는데, 이 때 각 관절을 사용자의 의도대로 쉽게 조종할 수 있도록 해주는 인터페이스를 디자인하기가 어렵기 때문이다. 본 논문에서는 자유도(DOF)가 높은 캐릭터의 움직임을 제어하기 위해 오랫동안 인형극에서 사용되고 있는 마리오넷 조종 기법[5]을 응용한 마리오넷 시스템을 제안하고자 한다. 우리는 가상 마리오넷 시스템을 물리기반 모델링과 햅틱 인터페이스를 기반으로 구현하였고, 이 시스템을 통해 높은 자유도를 가지는 유관절체 캐릭터의 복잡한 움직임을 쉽게 생성해낼 수 있었다. 그리고 사용자에게 햅틱 포스 피드백을 줌으로써 더욱 정교한 마리오넷을 조작이 가능하도록 하였다. 이 시스템을 일반적인 유관절체에 적용한다면 다양한 움직임을 쉽고 빠르게 생성할 수 있을 것이다.

1. 서론

컴퓨터 애니메이션과 인터랙티브 컴퓨터 게임에서, 유관절체의 모션을 직관적으로 제어하는 것은 어려운 일로 인식이 되어왔다 [3, 6, 9]. 일반적인 게임에 등장하는 사람 모델의 경우 30 자유도(DOF) 이상의 높은 자유도를 가지는데, 이런 모델을 움직임일 때 몸의 각 부분을 쉽게 조종할 수 있도록 하는 인터페이스를 제공하기가 힘들기 때문이다[4].

모션 생성과 제어를 위한 일반적인 기법으로 모션 캡쳐나 수작업을 통해 모션을 합성하여 만들어내는 방법들이 많이 사용되어 왔다[8]. 그러나 이런 방법들은 원하는 움직임을 생성하기 위해서 대용량의 모션 데이터가 필요하거나 반복적인 수작업이 필요하다는 단점이 있다. 게다가 대부분의 작업이 오프라인으로 이루어지므로, 캐릭터와의 실시간 상호작용을 구현하기가 매우 힘들다.

우리는 본 논문에서 이런 문제를 해결하면서 유관절체 캐릭터를 자연스럽게 조종할 수 있는 새로운 방법으로써 마리오넷 조종 기법을 제안한다. 우리는 이 기법을 사용하여 마리오넷 시스템을 구현하고, 가상의 마리오넷을 물리기반으로 시뮬레이션 하였다. 그리고 햅틱 인터페이스를 통해 마리오넷 조종막대(마리오넷을 매달고, 움직일 수 있도록 하는 십자가 모양으로 생긴 막대)의 움직임을 정확하게 재현하였다. 또한 사용자에게 마리오넷의 움직임으로 인해 발생된 힘을 돌려줌으로써 실제 마리오넷을 조종할 때와 같은 느낌이 들 수 있도록 하였다. 이것은 시각적인 정보와 더불어 손의 느낌에 의하여 마리오넷을 조종할 수 있도록 해주기 때문에, 더욱 섬세한 조종을 가능하게 한다.

가상 마리오넷 시스템은 기존의 유관절체 애니메이션 시스템에 비해 다음과 같은 이점을 제공한다.

- 유관절체 캐릭터의 복잡한 움직임을 빠르게 생성할 수 있다.
- 햅틱 인터페이스를 사용하여 가상환경 상의 유관절체 캐릭터를 직관적인 방법으로 조종할 수 있고 그들과 실시간으로 상호작용을 할 수 있다.
- 물리기반 모델링 기법을 사용하였기 때문에 생성된 움직임과 반응이 실제 생활에서 일어나는 것과 매우 유사하다.
- 마리오넷 시스템은 [2]와 같은 게임에 응용될 수 있을 것이다.

2. 가상 마리오넷

2.1 마리오넷 모델링

마리오넷은 [그림1]에 나와있는대로, 12개의 강체(rigid body) - 1개의 구 형태 강체와 11개의 박스 형태 강체가 사용된다 - 로 구성이 되고 각 강체는 11개의 관절로 연결된다. 각 부분은 순서대로 머리, 몸통, 팔 윗부분 2개, 팔 아랫부분 2개, 다리 윗부분 2개, 다리 아랫부분 2개이다. 11개의 관절은, 목, 어깨, 팔꿈치, 엉덩이, 무릎 그리고 발목이며, 각 관절은 전체 몸이 알맞은 움직임을 만들어내도록 하는데 중요한

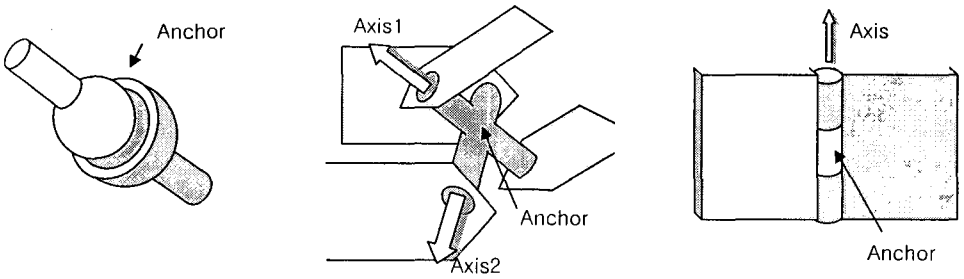

그림 2: 마리오넷 시스템에 사용된 관절의 종류: 볼 소켓 관절, 유니버설 관절, 경첩 관절 (왼쪽부터 순서대로)

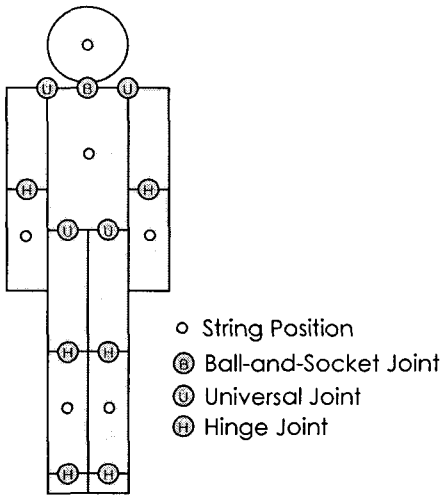

그림 1: 마리오넷 모델링

역할을 하고 있다. [그림1]에서 분홍색 동그라미 표시가 각 관절을 나타내고 있다.

마리오넷 몸의 각 부분을 연결하기 위해 [그림2]의 총 3가지 관절이 이용되었다. 사용된 관절은 볼 소켓 관절(ball-and-socket joint), 경첩 관절(hinge joint), 그리고 유니버설 관절(universal joint)이다 ([11]의 용어 인용). 볼 소켓 관절은 연결된 두 몸체가 항상 떨어지지 않도록 한다. 경첩 관절은 연결된 두 몸체가 정해진 하나의 축을 중심으로 회전할 수 있도록 하고, 유니버설 관철은 회전할 수 있는 축 두개를 제공한다. 각 관절을 구현하기 위해 사용된 제약식(constraint dynamics equation)은 다음과 같이 표현된다[11]:

$$\begin{aligned} J_1 v_1 + \Omega_1 \omega_1 + J_2 v_2 + \Omega_2 \omega_2 &= c + C\lambda \\ \lambda &\geq l \\ \lambda &\leq h \end{aligned} \quad (1)$$

위 식에서 J_i과 Ω_i는 자코비언(Jacobian) 행렬이다. 관절마다 다른 제약 조건(자코비언 행렬)을 가진다. v_1와 ω_1는 첫번째 몸체의 선속도와 각속도이다. 마찬가지로 v_2와 ω_2 는 두번째 몸체의 선속도와 각속도가 된다. c는 관절에 따른 제약 벡터, λ는 제약 힘이고 위의 식을 만족시키기 위해 적용된다. λ는 l부터 h의 범위를 가진다. C는 대각행렬로, CFM 행렬(constraint force mixing (CFM) matrix)이라고 불린다. 이 행렬은 λ를 제약식에 포함시킨다. C의 내용에 따라 특정한 제약이나 효과를 낼 수 있다 [11]. 마리오넷 시스템에서는 볼 소켓 관절은 목에, 경첩 관절은 무릎과 발목에, 유니버설 관절은 어깨와 엉덩이에 쓰였다.

[그림1]의 노란 동그라미는 스트링(string)이 마리오넷 몸체를 연결하고 있는 지점을 나타낸다. 스트링은 머리, 몸통, 팔 아랫부분과 다리 아랫부분에 연결된다. 실이 각 부분을 당기는 힘은 다음과 같이 계산된다:

$$f_{body} = -k(x_{body} - x_{string}) - k_d(\dot{x}_{body} - \dot{x}_{string}) \quad (2)$$

x_{string}는 스트링 끝 부분(마지막 파티클)의 위치, x_{body}는 실과 연결된 각 몸체의 위치, k는 스트링의 탄성계수, 그리고 k_d는 댐핑 상수이다. 몸체로부터 스트링에 다시 되돌아오는 힘은 $f = -f_{body}$가 된다.

2.2 스트링 모델링

사용자는 스트링을 통해 마리오넷을 움직일 수 있다. 사용자가 조작하는 조종막대가 스트링을 움직이게 하는데, 스트링이 움직이면서 연결된 마리오넷 몸의 각 부분을 당기게 되고, 결국에는 마리오넷이 특정한 자세를 취하게 된다.

스트링은 변형체(deformable body)로 모델링 되었다. 스트링과 같은 종류의 변형체를 모델링하는 방법은 여러가지가 있지만[7], 우리는 높은 정확성보다는 빠른 속도에 비중을 더 두었기 때문에 스프링-매스 방법을 선택하였다. 이 방법은 실행 속도가 빠르기 뿐만 아니라, 충분히 정확한 결과를 낼 수 있다.

우리의 시스템에서는, 6개의 스트링이 마리오넷의 몸을 매달고 있다. 머리 스트링, 엉덩이 스트링, 2개의 손 스트링, 2개의 다리 스트링이다. 스트링을 구성하는 파티클 중 제일 위에 있는 파티클은 시뮬레이션 스텝마다 위치가 업데이트 된다. 이 위치는 조종 막대의 위치를 따르게 된다. 조종 막대에 연결된 첫 번째 파티클을 제외하고 나머지 파티클들은 다음과 같은 힘을 적용함으로써 움직임이 결정된다.

1. 인접한 두 파티클 사이의 스프링 힘: $f_s = -k_s(x_i - d)$, k_s 는 스프링의 탄성계수, x_i는 두 파티클 사이의 거리, 그리고 d는 일정하게 유지되는 각 스프링의 길이이다.

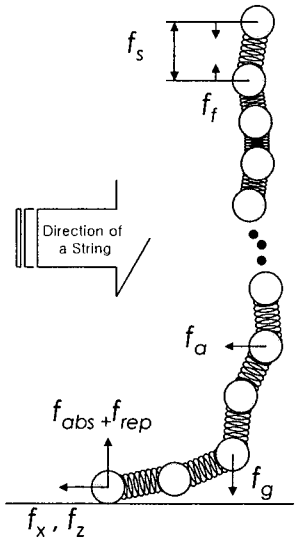

그림 3: 스트링 모델링. 스트링은 파티클들을 연결하는 체인 형태로 모델링 되었다. 각 파티클에는 스프링 힘 f_s, 내부 마찰력 f_f, 중력 f_g, 공기 저항 f_a, 지면과의 마찰력 f_x, f_z, 지면 흡수력 f_{abs} 그리고 지면의 반발력 f_{rep} 등의 힘이 적용될 수 있다.

2. 내부 마찰력: $f_f = -k_f \times (\dot{x}_i - \dot{x}_{i+1})$, k_f는 마찰계수이다.
3. 중력: $f_g = mg$, m은 파티클의 질량이고 g는 중력 가속도이다.
4. 공기 저항: $f_a = -k_{air}\dot{x}_i$, k_{air}는 공기 저항 계수이다.
5. 지면 마찰력 (파티클이 땅에 닿아있을 때): 지면 마찰력은 x와 z 방향으로만 적용된다; $f_x = -v_i^x k_{ground}$, $f_z = -v_i^z k_{ground}$, v_i^x, v_i^z는 파티클 i의 y방향과 z방향으로의 속도, k_{ground}는 지면 마찰 계수이다.
6. 파티클이 계속 땅에 닿아있게 될 경우 지면 반발력과 지면 흡수력이 적용된다: 지면 흡수력 $f_{abs} = v_i^y k_{abs}$, 지면 반발력 $f_{rep} = v_i^y(h_{ground} - p_i^y)$, k_{abs}는 지면 흡수 계수, p_i^y는 파티클 i의 y축 위치이고 h_{ground}는 지면의 높이이다.

위에 나열된 힘(F)을 모든 파티클에 대해 적용한 후, 뉴턴 2차 상미분 방정식(Ordinary Differential Equation)을 푼다.(오일러의 음함수 방법으로 $\ddot{\mathbf{x}} = \frac{\mathbf{F}}{m}$를 푼다.)[12]

스트링의 충돌 검사와 충돌 반응은 지면과의 충돌에 대해서만 구현하였다. 이 과정은 스트링을 구성하는 파티클이 지면과 충돌하게 되는 경우, 지면 반발력과 지면 흡수력을 그 파티클에 대해 적용(5번째와 6번째 과정)함으로써 수행된다. 그러나 스트링과 스트링 사이의 충돌이나, 스트링과 마리오넷 몸체와의 충돌에 대해서는 고려하지 않았다. 그 이유는 다음과 같다:

- 사용자와 캐릭터간의 상호작용을 가능하게 하기 위해서는 시뮬레이션이 매우 빠르게 진행될 수 있어야 한다. 스트링과 같은 변형체의 충돌 검사와 충돌 반응은 계산이 많이 필요하므로 실행 속도를 저하시키게 된다.
- 이 시스템의 궁극적인 목적은 스트링의 세밀한 묘사보다도 캐릭터의 모션을 제어하는 것이다.
- 실제 마리오넷 조종 방법에서도, 스트링을 겹치거나 꼬는 것은 마리오넷을 조종하는 기술로는 사용되지 않는다.

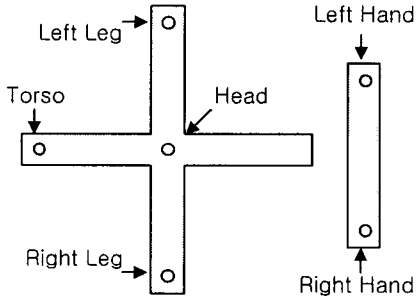

그림 4: 주 조종막대와 손 막대

2.3 조종막대 모델링

조종막대는 시스템 내에서 물리적으로 시뮬레이션이 되는 대상은 아니다. 이것은 단지 사용자로부터의 인풋(위치와 기울기)을 시스템에 전달해주는 도구 또는 인터페이스의 역할을 한다. 즉, 조종막대는 가상 현실상의 다른 물체들(예를 들어, 스트링이나 마리오넷의 몸)을 사용자의 의도대로 움직이게 한다. 조종막대는 햅틱 인터페이스를 통해 사용자로부터 직접 조종된다. 이 시스템에서는, 주 조종막대와 손 막대를 이용하여 마리오넷을 조종한다. 주 조종막대는 마리오넷의 머리, 몸통, 다리를 연결하고 있고, 몸 전체의 움직임을 제어한다. 손 막대는 양손을 조종하기 위한 것이다. 시스템에서 구현된 조종막대는 실제 마리오넷 조종에 쓰이는 조종막대([그림4])와 매우 유사한 생김새를 가진다.

햅틱 디바이스는 디바이스만의 좌표계를 따로 가지고 있으므로, 인풋으로 들어온 디바이스의 위치($\mathbf{P}$)를 마리오넷 시스템이 사용하고 있는 월드 좌표($\mathbf{P}'$)로 변환하는 과정이 필요하다. 좌표계의 변환은 다음과 같이 이루어진다:

$$\mathbf{P}' = \mathcal{S}\mathcal{W}\mathbf{P} \tag{3}$$

$\mathcal{S}$와 $\mathcal{W}$는 4×4 호모지니어스(homogenous) 행렬로 각각 햅틱 워크스페이스($\mathcal{S}$)를 스케일링하거나, 햅틱 좌표계를 월드 좌표계($\mathcal{W}$)로 변환하는 일을 한다.

각 스트링이 연결되어 있는 자리도 조종막대의 바디 프레임에 정의되어 있기 때문에 매 스텝마다 변환이 되어야한다. 스트링의 위치는 [그림1]에 동그라미로 표시되어 있다.

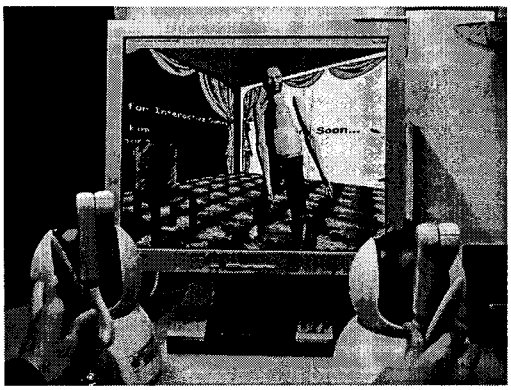

그림 5: 시스템 셋업

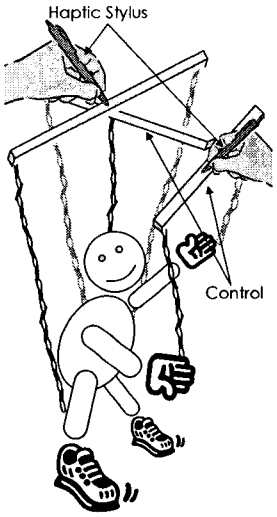

그림 6: 햅틱 디바이스와 조종막대의 연결

3. 햅틱 인터페이스

마리오넷 시스템에서 주목할만한 점은 가상환경 상의 마리오넷을 조종하는 데 햅틱 인터페이스를 사용하였다는 점이다. 햅틱 인터페이스는 [그림5]와 [그림6]에 보이는 것과 같이, 바로 조종막대 그 자체와 대응된다. 사용자는 복잡한 키 입력을 통해 조종하는 것 대신 햅틱 인터페이스를 사용함으로써 마리오넷을 더욱 직관적으로, 쉽게 조작할 수 있다.

3.1 인터페이스 설계

센서블(Sensable) 사에서 나온 두 대의 햅틱 옴니 디바이스([그림5])를 이용하여 각각 주 조종막대와 손 막대를 조종할 수 있도록 하였다.

햅틱 옴니 디바이스는 6자유도(Degree Of Freedom)의 인풋(햅틱 디바이스의 위치와 오리엔테이션)을 받고, 3자유도의 아웃풋(변환된 힘)을 줄 수 있다. 각 디바이스의 위치와 기울기는 햅틱 디바이스의 업데이트와 같이 이루어진다 ($1KHz$). 각 햅틱 디바이스의 끝부분은 [그림6]에서 보이는 것과 같이 주 조종막대와 손 막대의 무게중심에 위치한다.

3.2 햅틱 피드백 계산

스트링의 장력을 느낄 수 있도록 하기 위해, 스트링의 상태를 팽팽한 상태와 느슨한 상태로 구분을 하였다. 스트링의 첫 파티클과 마지막 파티클 사이의 거리를 구하여 주어진 한계값 이상인 경우에는 팽팽한 상태, 더 짧을 경우에는 느슨한 상태로 지정하였다.

i번째 스트링이 팽팽한 경우, 사용자에게 전달되는 피드백 힘 F_i는 다음의 식에 의해 결정된다:

$$F_i = -k(x_{proxy} - x_{string}) - mg - k_d(\dot{x}_{proxy} - \dot{x}_{string}) \quad (4)$$

k는 탄성계수, m은 마리오넷의 총 질량, k_d는 햅틱 디바이스의 댐핑 계수, g는 중력 가속도, x_{string}은 스트링의 마지막 파티클, 그리고 x_{proxy}는 햅틱 디바이스의 위치이다.

만약 스트링이 느슨한 상태라면, 사용자에게는 아무 힘도 전달되지 않는다.

4. 결과와 분석

4.1 구현

마리오넷 시스템은 Visual C++와 OpenGL 그래픽스 라이브러리, 그리고 햅틱 API를 위해서는 OpenHaptics 라이브러리[1]를 사용하였다.

마리오넷 시뮬레이션이 빠른 속도로 이루어지도록 시뮬레이션 과정에서는 마리오넷의 각 몸체를 박스나 구와 같은 간단한 바운딩 볼륨으로 경계를 삼고, 렌더링시에는 원래의 모양을 그대로 보여주도록 하였다. Open Dynamics Engine (ODE) [11]은 유관절체 시뮬레이션을 위한 물리 엔진으로 이용이 되었고, 우리의 시스템의 목적에 맞도록 약간 수정되었다. 마리오넷 관절을 모델링하기 위해서 ODE의 볼 소켓 관절, 경첩 관절, 유니버설 관절을 이용하였다.

[그림7]에서와 같이 마리오넷 시스템은 4개의 프로세스로 구성이 된다. 스트링 시뮬레이션, 마리오넷 시뮬레이션, 햅틱 렌더링, 그리고 그래픽스 렌더링 과정이다. 각 과정은 다음과 같이 연결이 되어 진행된다:

1. 스트링 시뮬레이션은 햅틱 디바이스의 위치를 받아서 스트링의 모양을 결정한다.
2. 스트링의 위치와 모양이 바뀌면서 마리오넷을 당기게 되고 마리오넷 시뮬레이션 과정에서 마리오넷의 움직임을 결정한다.
3. 마리오넷과 주변 환경의 현재 상태가 시각적으로 렌더링된다.
4. 마리오넷의 움직임으로 인해 연결된 스트링의 마지막 부분의 위치가 변하게 되고, 다시 전체 스트링의 모양이 변하게 된다.
5. 스트링과 마리오넷 시뮬레이션의 결과는 힘의 형태로 변환이 되어 햅틱 디바이스에 전달된다.

스트링 시뮬레이션과 햅틱 렌더링은 $1KHz$의 주기로 업데이트되고, 마리오넷 시뮬레이션과 그래픽스 렌더링은 $30Hz$의 주기로 업데이트 된다.

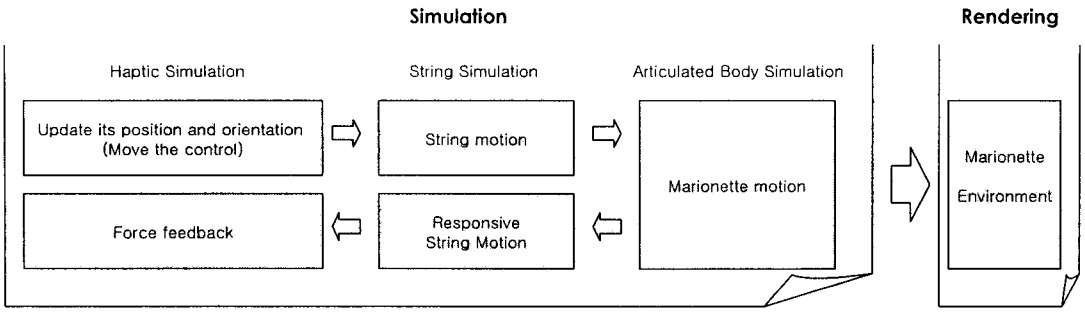

그림 7: 시스템 다이어그램. 붉은 색은 비동기화된 프로세스 과정이고, 푸른 색은 서로 동기화되어 진행되는 프로세스 과정이다.

스트링은 평균 50개의 파티클로 구성이 되어 있고, [10]와 같은 방법을 사용한다면 더욱 빠르게 시뮬레이션이 이루어질 수 있을 것이다.

4.2 결과

[그림8]은 마리오넷 시스템을 이용하여 생성된 캐릭터의 모션을 보여주고 있다. 이와 같이 사용자는 유관절체에 대해서 쉽게 모션을 생성할 수 있다. 우리가 제안한 마리오넷 시스템은 신속한 상호작용을 가능하게 하고(1KHz 이상의 빠른 시뮬레이션을 제공), 마리오넷과 가상 환경에 대해 물리적으로도 충분히 사실적인 반응을 제공한다. [그림8]에서 가상 마리오넷은 의자에서 일어나려고 하는 모습, 공을 차고 쫓아가는 모습을 보여주고 있다.

5. 결론과 향후 연구

우리는 본 논문에서 사용자와 유관절체 캐릭터의 상호작용을 제공하고, 캐릭터의 움직임을 물리적으로 타당하게 모델링할 수 있는 마리오넷 시스템을 제안하였다. 이 시스템은 또한 햅틱 디바이스를 사용하여 복잡한 움직임도 쉽고 직관적으로 조종할 수 있는 인터페이스를 제공한다. 사용자는 두 대의 햅틱 디바이스를 이용하여 다양한 마리오넷 모션을 생성할 수 있으며, 가상 환경과의 상호작용 또한 관찰할 수 있다.

향후 연구로, 마리오넷 시스템의 성능 향상을 위하여, 마리오넷과 스트링 시뮬레이션에 [10]과 같은 방법을 적용해 보고자 한다. 현재의 마리오넷 시스템은 스트링 사이의 충돌이나 스트링과 마리오넷 몸체의 충돌에 대해서는 검사를 하고 있지 않은데, 스트링과 마리오넷 사이의 충돌을 추가적으로 구현을 한다면 좀 더 복잡한 형태의 마리오넷을 시도해볼 수 있을 것이다. 마지막으로, 더욱 향상된 인터페이스 제공을 위하여 더 높은 자유도의 아웃풋을 내는 햅틱 디바이스에 응용을 해 볼 것이다.

6. 감사의 글

본 연구는 부분적으로 교육인적자원부의 재원으로 한국 학술진흥재단(과제번호 R08-2004-000-10406-0)과 정보통신부 ITRC 프로그램의 지원으로 수행되었음.

참고 문헌

[1] 3D Touch SDK OpenHaptics™ toolkit version 1.02 API reference. http://www.sensable.com.

[2] Ragdoll Kungfu. *http://www.ragdollkungfu.com/.*

[3] A. M. B. A. Bar-Lev and G. Elber. Virtual marionettes: A system and paradigm for real-time 3D animation. Technical report, Technion, I.I.T., Israel, May 2004.

[4] R. Boulic, P. Fua, L. Herda, M. Silaghi, J. Monzani, L. Nedel, and D. Thalmann. An anatomic human body for motion capture. In *Proceedings of EMMSEC*, 1998.

[5] D. Currell. *Making and Manipulating Marionettes.* The Crowood Press, 2004.

[6] J. Davis, M. Agrawal, E. Chuang, Z. Popovi, and D. Salesin. A sketching interface for articulated figure animation. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, 2003.

[7] K. Erleben, J. Sporring, K. Henriksen, and H. Dohlmann. *Physics Based Animation.* Charles River Media, 2005.

[8] L. Kovar, M. Gleicher, and F. Pighin. Motion graphs. In *ACM SIGGRAPH*, 2002.

[9] J. Laszlo, M. Panne, and E. Fiume. Interactive control for physically-based animation. In *Proceedings of SIGGRAPH 2000*, pages 201–209, 2000.

[10] S. Redon, N. Galoppo, and M. C. Lin. Adaptive dynamics of articulated bodies. In *Proceedings of SIGGRAPH 2005*, 2005.

[11] R. Smith. Open Dynamics Engine user guide, May 2004.

[12] A. Witkin and D. Baraff. Physically based modeling: Principles and practice. In *SIGGRAPH Course Note*, 1997.

그림 8: 가상 마리오넷 시스템을 이용하여 생성된 애니메이션. 가상 공간의 마리오넷이 의자에서 일어나려는 모습, 그리고 공을 차고 쫓아가는 모습을 보여주고 있다.