안정적이고 병렬화가 용이한 옷감 애니메이션 기법

강영민*

Stable and Easily Parallizable Cloth Animation Method

Young-Min Kang*

요 약

효율적인 옷감 애니메이션 생성을 위해 암시적 적분법이 표준적 접근법으로 자리 잡았다. 이 기법은 시스템의 안정성이 보장되므로 큰 시간 간격을 사용할 수 있으며, 옷감 모델의 실시간 혹은 상호작용적 애니메이션을 위한 최선의 선택으로 받아들여지고 있다. 암시적 방법이 옷감 애니메이션에 도입된 이후 다양한 종류의 암시적 적분 기반 옷감 애니메이션 기법들이 제안되었으며 일반적인 PC 환경에서 수천 개의 질점을 가진 옷감 모델을 실시간에 시뮬레이션 할 수 있는 수준에 도달해 있다. 암시적 적분법이 안정성을 보장한다는 장점을 가지지만 명시적 기법에 비해 구현이 조금 더 복잡하며 병렬화가 어렵다는 문제를 가지고 있다. 암시적 적분법을 통해 옷감 애니메이션을 생성하는 것은 대규모 희소행렬을 가진 선형 시스템을 푸는 것으로 정형화된다. 본 논문에서는 암시적 적분법의 특성을 이용하여 안정적으로 복잡한 옷감 모델의 동작을 생성하면서도 매우 쉽게 병렬화가 가능한 기법을 제안한다. 옷감 애니메이션은 동작 시뮬레이션과 함께 자체충돌을 고려해야만 사실적인 결과를 얻을 수 있다. 그런데 자체충돌 감지 역시 중요한 계산상의 병목으로 작용한다. 본 논문에서는 효율적인 자체 충돌 처리 기법도 같이 제안한다.

ABSTRACT

Implicit Integration has become a standard approach to efficient cloth animation, and it guarantees the stability of the system so that large steps can be used. Therefore, it is regarded as the best method for the real-time or interactive animation of cloth. Since the implicit method was introduced for stable cloth animation, various cloth animation techniques based on the method have been proposed. It is now possible to generate the real-time animation of cloth model with thousands of mass-point in general PC environments. Although the implicit method guarantees the stability, the implementation of the implicit method is generally more difficult than that of the explicit method. Even worse, it is very difficult to parallelize the computation process of the implicit method. The cloth animation with implicit method can be formalized as a linear system solving. In this paper, we propose an stable and efficient cloth animation techniques based on the implicit method. The proposed method can be easily parallelized. Self-collision is another important issue in cloth animation, we also propose an efficient self-collision avoidance techniques.

키워드

옷감 애니메이션, 실시간 애니메이션, 암시적 적분, 병렬화, 충돌 감지

Ⅰ. 서 론

옷감 애니메이션을 실시간에 처리할 때 발생하는 어려움을 해결하기 위한 다양한 연구들 가운데 가장 중요한 성과는 암시적 적분법(implicit integration)을 이용하여 안정성을 보장하고 큰 시간 간격을 사용할 수

* 동명정보대학교

접수일자 : 2005. 5. 27

있게 한 것이다. 암시적 적분법을 옷감 애니메이션에 적용하면 선형 시스템 풀이 문제가 된다[1]. 몇몇 연구자들이 암시적 적분법의 안정성을 이용하면서도 선형 시스템 풀이 과정의 계산 부담을 줄일 수 있는 효율적인 기법들을 제안하여 실시간 옷감 애니메이션에 이용하려고 하였지만[2,5,6,8], 이러한 기법들은 실시간 성능을 얻기 위해 정확성을 희생하였고 질점의 수가 많은 복잡한 모델에는 적용이 힘들었다. 최근에는 코르디에(Cordier)에 의해 옷감을 완전히 갖춰 입은 캐릭터 모델의 실시간 애니메이션 기법이 제안되었다[4]. 그러나 이 기법은 물리 시뮬레이션 자체의 성능을 개선하지는 못했다. 암시적 적분법을 이용한 옷감 애니메이션은 명시적 기법에 비해 구현이 어려우며 시뮬레이션 계산의 병렬화 역시 쉽지 않다. 본 논문에서는 안정정인 결과를 얻을 수 있으면서도 병렬처리에 적합한 형태로 각 질점의 상태 변화를 계산할 수 있는 기법을 제안한다.

Ⅱ. 애니메이션 모델

옷감 모델의 안정적인 애니메이션을 생성하기 위해서는 암시적 수치 적분법을 사용해야 한다. 따라서 다음 상태의 힘 f^{t+h}를 포함하는 역 오일러 기법을 다음과 같이 적용하여 안정적인 옷감 애니메이션을 생성할 수 있다[7]:

$$\begin{pmatrix} \mathrm{v}^{t+h} \\ \mathrm{x}^{t+h} \end{pmatrix} = \begin{pmatrix} \mathrm{v}^{t} + h\mathrm{M}^{-1}\mathrm{f}^{t+h} \\ \mathrm{x}^{t} + h\mathrm{v}^{t+h} \end{pmatrix} \tag{1}$$

이때 h는 시간 간격이며, 질점의 수를 n이라고 할 때, v는 각 질점의 속도를 원소로 하는 $3n$ 차원의 벡터로 $[\mathrm{v}_1, \mathrm{v}_2, \cdots, \mathrm{v}_n]^{\mathrm{T}}$를 의미한다. f와 x 역시 각 질점의 힘과 위치를 원소로 하는 $3n$ 차원 벡터이다. 행렬 M은 $3n \times 3n$의 질량 행렬이다. 암시적 적분법에 의한 옷감 애니메이션은 질점의 속도 변화 벡터인 $\Delta\mathrm{v}^{t+h} = h\mathrm{M}^{-1}\mathrm{f}^{t+h}$를 계산하는 문제인데, 문제는 안정성을 위해 도입된 다음 상태의 힘 f^{t+h}를 현재 스프링 상태로 계산할 수 없다는 것이며, 이 값은 다음과 같이 근사를 통해 얻어야 한다.

$$\mathrm{f}^{t+h} = \mathrm{f}^{t} + \frac{\partial \mathrm{f}}{\partial \mathrm{x}} \Delta\mathrm{x}^{t+h} = \mathrm{f}^{t} + \mathrm{J}\Delta\mathrm{x}^{t+h} \tag{2}$$

이때 J는 힘 벡터를 각 정점의 위치로 이루어진 벡터 x로 미분한 자코비안(Jacobian) 행렬이며, 질량 스프링 모델의 애니메이션은 결국 다음과 같은 $\mathrm{Ax} = \mathrm{b}$ 형태의 선형 시스템으로 표현할 수 있다:

$$(\mathrm{M} - h^2\mathrm{J})\Delta\mathrm{v}^{t+h} = h\mathrm{f}^{t} + h^2\mathrm{J}\mathrm{v}^{t} \tag{3}$$

옷감 애니메이션은 식 3에 나타난 선형 시스템의 해를 구하는 문제가 되며 행렬 $\mathrm{M} - h^2\mathrm{J}$는 매우 큰 행렬이다. 다행히 이 행렬이 희소 행렬이며 대칭 행렬이기 때문에 효율적으로 풀 수 있다. $\mathrm{M} - h^2\mathrm{J}$를 간단히 W로 표현하자. 이 행렬 W는 매우 중요한 여러 특성을 가지며, 이러한 특성들을 적절히 이용함으로써 실시간 애니메이션이 가능하다. 식 3의 우변은 스프링에서 발생하는 힘에 점성력 $h\mathrm{J}\mathrm{v}^{t}$이 추가된 것으로 다음과 같이 연결된 질점만을 고려함으로써 쉽게 계산할 수 있다[5]:

$$\tilde{\mathrm{f}}_i^{t} = \mathrm{f}_i^{t} + h\sum_{(i,j)\in E} \mathrm{J}_{ij}(\mathrm{v}_j^{t} - \mathrm{v}_i^{t}) \tag{4}$$

이렇게 구한 내부힘을 이용하여 식 3의 선형 시스템은 다음과 같이 간단히 표현될 수 있다:

$$\mathrm{W}\Delta\mathrm{v}^{t+h} = h\tilde{\mathrm{f}}^{t} \tag{5}$$

W 행렬의 특성을 고려하여 질점 i의 속도변화를 중심으로 다시 정리하면 식 5는 다음과 같은 n 개의 방정식으로 이루어진 선형 시스템이다.

$$\Delta\mathrm{v}_i^{t+h} = \mathrm{W}_{ii}^{-1}h\tilde{\mathrm{f}}_i^{t} + h^2\sum_{(i,j)\in E} \mathrm{J}_{ij}\Delta\mathrm{v}_j^{t+h} \tag{6}$$

그러나 이 값은 다른 방정식에 의존적이기 때문에 각 질점의 속도 변화가 식 6에 의해 독립적으로 계산될 수 없으며, 자코비(Jacobi) 반복법을 k 번 수행해 얻는 값을 $\Delta\mathrm{v}_i^{t+h^{(k)}}$라고 하면 다음과 같이 해를 구할 수 있다.

$$\Delta v_i^{t+h^{(0)}} = W_{ii}^{-1} h \tilde{f}_i^t$$
$$\Delta v_i^{t+h^{(k+1)}} = W_{ii}^{-1} (h \tilde{f}_i^t + h^2 \sum_{(i,j) \in E} J_{ij} \Delta v_j^{t+h^{(k)}}) \quad (7)$$

Ⅲ. 병렬처리에 적합한 속도변화 계산식

병렬 처리를 통해 옷감 애니메이션의 효율성을 높이려면, 질점 상호간의 의존이 없을수록 좋다. 명시적 기법의 경우 각 질점에 가해지는 힘을 계산하면 모든 질점에 대해 독립적인 상태 갱신이 가능하므로 병렬화에 적합하다. 그러나 앞에서 언급한 바와 같이 명시적 기법은 안정성 문제 때문에 매우 짧은 시간 간격을 사용해야 하며 병렬 처리를 하더라도 좋은 성능을 얻기 힘들다.

식 7의 속도 변화 계산식을 보면 각 질점 i의 속도 변화를 계산하기 위해 연결된 다른 질점 j의 속도 변화를 고려해야 하므로 어떤 옷감 모델의 질점을 몇 개의 구역으로 나누고 각각의 구역에서 독립적으로 속도 변화를 계산할 수가 없는 것이다. 하지만 식 7에 나타나 있는 속도변화 계산식은 선형 시스템 전체를 푸는 것 보다는 훨씬 더 병렬화에 유리한 형태로 표현되어 있으며 이를 더욱 병렬화에 적합한 형태로 변경할 수 있다. 힘 f와 점성력이 포함된 힘 $\tilde{f}$가 구해져 있고, 힘의 미분치 J가 구해져 있다고 가정하면 W_{ii}를 계산하는 것은 모든 질점에 대해 독립적으로 계산될 수 있다. 또한 각 질점의 속도변화 초기값 $\Delta v_i^{t+h^{(0)}}$를 간단히 p_i라고 할 때, 이 초기값도 W_{ii}를 이용하여 각 질점별로 독립적으로 계산할 수 으며 각 질점 i에 대해 이 두 값을 구하는 작업은 매우 쉽게 병렬화할 수 있다.

$$W_{ii} = M_i - h^2 J_{ii} \in R^{3\times 3}$$
$$p_i = h W_i^{-1} \tilde{f}_i \in R^3 \quad (8)$$

실험을 통해 단 한 번의 반복을 통해 얻은 결과도 안정적이며 사실적인 동작을 보여주었기 때문에 다음과 같은 속도 변화의 근사해를 사용하여 옷감 애니메이션을 생성할 수 있다.

$$\Delta v_i^{t+h} \simeq \Delta v_i^{t+h^{(1)}} = p_i + h^2 W_i^{-1} (\sum_{(i,j) \in E} J_{ij} p_j) \quad (9)$$

특정 질점 i의 속도 변화를 구하는 식 9에서 병렬화를 방해하는 요소는 질점 i에 연결된 모든 j의 속도변화 초기값 p_j를 고려해야 한다는 것이다. 이를 병렬화하기 위해서는 p 벡터를 모든 병렬 작업에서 공유하여야 하며, J_{ij}와 p_j의 곱을 병렬적으로 계산하여 i 질점에 반영할 수 있는 기법이 필요하다. 유효한 J_{ij}의 수는 스프링 에지의 수와 정확히 일치하므로 이 작업은 에지를 몇 개의 그룹으로 나눠 병렬적으로 처리할 수 있다. $\Sigma_{(i,j) \in E} J_{ij} p_j$를 q_i라고 하면 질점 i의 속도 변화는 다음과 같이 계산할 수 있다.

$$\Delta v_i^{t+h} \simeq \Delta v_i^{t+h^{(1)}} = p_i + h^2 W_i^{-1} q_i \quad (10)$$

옷감을 구성하는 질점을 여러 개의 그룹으로 나눈다고 하자. 어떤 질점 i가 그룹 G에 속한다고 할 때, 그룹 G를 처리하는 프로세스는 질점 i와 관련된 W_{ii}와 J_{ii}, 그리고 $\tilde{f}_i$를 가지고 있기 때문에 그룹 내의 정보만으로 독립적으로 계산이 가능하다. 따라서 p_i를 계산하는 작업은 각각의 그룹 G에 대해 코드 1과 같은 작업을 수행하도록 병렬화 할 수 있다.

코드 1. 그룹 G에 속한 질점의 p_i 계산

```
for all mass-point i ∈ G {
        compute p_i;
}
```

q_i를 계산하는 방법은 모든 스프링 에지에 대해 수행한 결과를 q 벡터에 누적하는 방식으로 다음 코드 2와 같이 계산할 수 있다.

코드 2. q 벡터 계산 방법

```
q←0
for all edge E_{i,j} {
        q_i←q_i + J_ij p_j
        q_j←q_j + J_ij p_i
}
```

q 벡터의 값은 모든 스프링 에지에 대해 이루어져야 하며, 어떤 스프링 에지 $E_{i,j}$에 의해 계산된 결과는 q_i와 q_j에 누적된다. 스프링 에지 집합을 질점 그룹을 고려하여 그룹화 함으로써 공유 데이터에 대한 동시 접근을 최소화할 수 있다. 어떤 질점 i와 j는 병렬화를 위해 각각 특정한 그룹에 속해 있다. 이 두 질점이 모두 같은 그룹에 속할 수도 있고, 서로 다른 그룹에 속할 수도 있다. 만약 두 질점이 같은 그룹에 속한다면 에지 E_{ij}역시 동일한 그룹에 속하며, i와 j가 서로 다른 그룹에 있다면 E_{ij}는 i의 그룹에 속하게 하는 것이다.

그림 1에서 정점 1과 3은 그룹 G_0에 속하고, 정점 2, 4는 G_1에 속한다. 각각의 그룹은 독립적으로 시뮬레이션을 수행하게 되는데, 이때 스프링의 수만큼 q_i를 계산하는 작업이 필요한데, 이 작업은 스프링을 적절히 그룹화 하여 병렬 처리를 할 수 있다. 스프링을 병렬화 할 때, 그림 2와 같이 스프링에 연결된 질점을 고려하여 그룹을 결정한다.

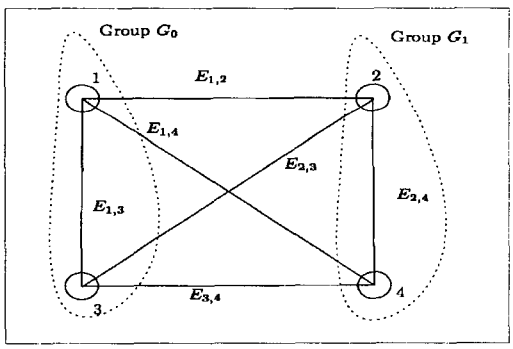

그림 1. 질점의 그룹화
Fig. 1 Grouping Mass-points

이러한 방식으로 그룹화 하여 각 그룹별로 p_i와 q_i를 계산하면 그룹 내의 데이터만을 이용하여 그룹 내의 값을 갱신하는 비율을 높일 수가 있다. 질점 i와 관련된 속성이라고 할 수 있는 W_{ii}^{-1}, J_{ii} 역시 질점 i와 동일한 그룹에 저장되며, 스프링 E_{ij}에 관련된 속성이라 할 수 있는 J_{ij}는 스프링 E_{ij}와 동일한 그룹에 저장된다. 따라서 각각의 그룹 G는 자신에 속한 질점과 스프링만을 이용하여 코드 3과 같은 작업을 수행한다. 이때 p 벡터와 q 벡터는 모든 작업 그룹에 공유되는 데이터이다.

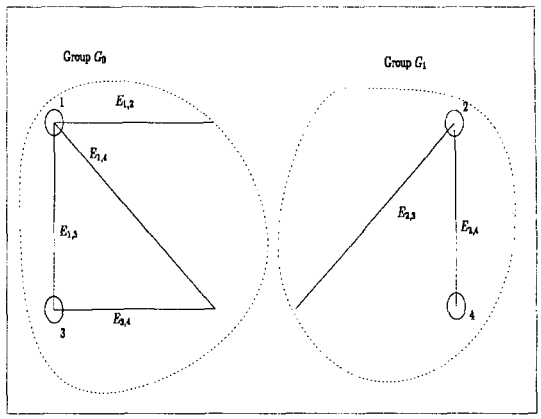

그림 2. 스프링 에지의 그룹화
Fig. 1 Grouping Spring Edges

코드 3. 그룹 G가 담당하는 p, q 벡터 계산

```
Task_For_Group_G {
        for all mass-point i ∈ G {
                compute p_i
        }
        for all edge E_ij ∈ G {
                q_i ← q_i + J_ij p_j
                q_j ← q_j + J_ij p_i
        }
}
```

코드 4. 힘, 점성력, 힘의 미분치 병렬 계산

```
Task_For_Group_G {
        for all edge E_ij ∈ G {
                f~_i ← f~_i + computeForce(E_ij);
                f~_j ← f~_j − computeForce(E_ij);
                J_ij ← ∂f_i/∂x_j
                J_ii ← J_ii + J_ij
                J_jj ← J_jj + J_ij
                f~_i ← f~_i + h J_ij (v_j − v_i)
                f~_j ← f~_j + h J_ij (v_i − v_j)
        }
}
```

앞서 우리는 J와 $\tilde{f}$가 미리 계산되어 있는 것으로 가정했다. 이 값들은 각 질점별로 계산되는 것이 아니라 q 벡터를 계산하는 것과 같이 각 스프링에 대해서 얻은 값들을 누적해서 얻게 된다. 각 질점에 적용되는

힘, 힘의 미분치 등을 계산하는 작업 역시 q를 구하는 방법과 유사하게 병렬화될 수 있다. 이 작업은 코드 4와 같은 방식으로 진행된다.

스프링 힘을 사용할 경우 J_{ij}는 다음과 같이 얻을 수 있다.

$$\frac{\partial f_i}{\partial x_j} = \kappa_{ij}\frac{|x_j - x_i| - l_{ij}^0}{|x_j - x_i|} I_3 + \kappa_{ij}\frac{l_{ij}^0}{|x_j - x_i|}\frac{(x_j - x_i)(x_j - x_i)^T}{|x_j - x_i|^2} \quad (11)$$

그런데 이 미분치는 두 질점 i와 j가 서로 가까워질 때 원소 값들은 무한히 커질 수 있어 시스템의 안정성이 깨어질 수 있다[3]. 이러한 상황을 피하기 위해서는 스프링 수축했을 때에 다음 값을 사용하여 시스템 안정성을 유지한다.:

$$\frac{\partial f_i}{\partial x_j} = \kappa_{ij}\frac{(x_j - x_i)(x_j - x_i)^T}{|x_j - x_i|^2} \quad (12)$$

Ⅳ. 실시간 충돌 처리

옷감과 같이 변형이 일어나는 객체에서 충돌 감지와 처리 문제는 객체 운동을 시뮬레이션하는 문제만큼이나 중요한 주제로 다루어졌다. 변형 가능한 객체의 동작을 생성할 때, 가장 큰 문제가 되는 충돌은 객체의 일부가 자기 자신의 다른 부분에 부딪히는 자체충돌(self-collision)이다. 자체충돌에 관해 다양한 연구가 수행되었는데, 자체 충돌을 효과적으로 처리하기 위해서는 강체들 사이의 충돌을 감지하는데 사용되는 일반적 계층구조를 적용하는 것이 큰 도움이 되지 않는다는 것이 알려져 있다. 본 논문에서는 이러한 자체충돌 감지를 수행할 때 발생하는 문제들을 실시간 환경이라는 제한조건을 만족하면서 해결할 수 있는 기법을 제시한다.

충돌 처리는 사실적인 애니메이션에 필수적인 요소로 컴퓨터 그래픽스에서 다루어지는 주요 문제들 가운데 하나이다. 특히 변형체에서 발생하는 자체 충돌을 감지하는 일은 높은 계산 복잡도를 요구한다. 자체 충돌에 관한 훌륭한 처리 기법은 볼리노(Volino)에 의해 제안되었고[10], 이후 프로보트(Provot)에 의해 개선되었다[9].

이 논문에서는 실시간 환경에서 자체 충돌을 처리할 수 있는 기법을 제안한다. 이 기법은 해쉬(hash) 함수를 이용하여 옷감을 구성하는 n 개의 질점을 $O(n)$ 개의 버켓에 저장한다. 이 해쉬 기반 기법은 옷감 모델의 바운딩 박스를 우선 생성한다. 이 바운딩 박스는 좌표 축을 따라 분할되어 다수의 육면체 셀로 나뉜다. 모든 질점은 각각 하나의 셀에 놓여 있게 된다. 이 셀은 세 개의 정수를 인덱스로 하여 (i_x, i_y, i_z)로 표현할 수 있다. 계산 효율을 높이고 저장 공간을 최소화하기 위해 해쉬 함수 $h(i_x, i_y, i_z)$를 이용하여 이 인덱스를 1차원 공간으로 변환한다. 이 해쉬 함수는 셀 인덱스를 n 보다 작은 양의 정수 k로 변환하며 이 k 값이 바로 질점이 저장될 버켓의 인덱스가 된다. 따라서 하나의 버켓 내에 존재하는 질점들이 서로 충돌 후보가 된다.

그림 3은 이렇게 질점이 놓인 셀의 인덱스를 기준으로 해쉬 함수를 수행하여 버켓에 저장하는 방법을 보이고 있다. 이 방법을 사용할 경우 서로 인접한 질점들은 서로 충돌하지 않으면서 동일한 셀에 놓이게 될 가능성이 높다. 이 문제는 볼리노(Volino)가 제안한 자체충돌 감지 기법에서 사용하는 충돌없는 영역을 이용하여 계산 속도를 높일 수 있다. 옷감 모델 전체를 m 개의 충돌없는 영역으로 나눌 수 있으며, 이때 m 개의 버켓 집합을 생성한다. 어떤 질점 i가 버켓 인덱스 k를 가지고, 충돌 없는 영역 l에 속한다고 하면, l 번째 버켓 집합의 k 번 버켓에 저장하는 것이다. 동일 버켓 집합에 있는 질점은 충돌 후보에서 제외함으로써 인접한 질점을 충돌 후보로 검사하는 작업을 제거할 수 있다.

충돌하는 두 정점을 찾은 뒤에는 적절한 방법으로 이 두 정점의 충돌을 회피하기 위해 두 질점의 위치와 속도를 변경해야 한다. 충돌 임계치 θ_c가 두 정점이 가질 수 있는 최소 거리라고 하자. 충돌하는 두 정점 i, j가 결정되면 충돌 예상 지점 C_{ij}와 충돌 지점의 속도 v_c, 그리고 충돌 법선 벡터 N_c를 구할 수 있다. 위치의 수정은 두 정점이 충돌 임계치 이내로 접근했을 때는 충돌 법선 방향으로 이동시켜 임계치 밖으로 나가게 하여 i의 위치를 $C_{ij} + (1/2)\theta_c N_c$로 설정한다. 충돌하려는 정점의 위치를 강제로 수정함으로써, 현재 프레임에서의 충돌은 막을 수 있지만, 속도가 그대로 유지되

어 수정 이후에도 계속해서 충돌하는 방향으로 움직이게 된다. 따라서 충돌하는 객체가 충돌지점으로 다가가고 있는 경우에는 속도를 감속하여 충돌이 일어나지 않도록 해야만 한다. 충돌 객체 i와 충돌 지점의 상대속도 v_{rel}는 $v_i - v_c$로 간단히 구할 수 있다. 이 상대속도와 충돌 법선 벡터 N_c의 내적이 음수가 되면 충돌 객체 i가 충돌 지점에 다가가고 있으므로 충돌 지점으로 다가가는 성분을 제거하는 작업을 다음과 같이 수행한다.

$$v_i^{adjusted} = v_i - (v_{rel} \cdot N_c)N_c \quad (13)$$

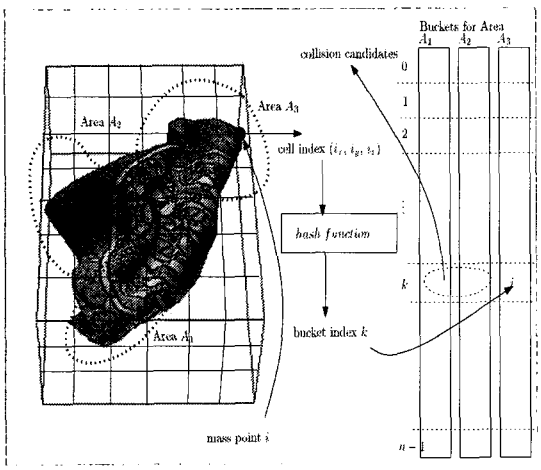

그림 3. 특정 점점을 버켓에 할당하는 방법
Fig. 3 Assigning Mass-points into Buckets

V. 실험 결과

본 논문의 기법은 일반적인 PC 환경에서 실시간 옷감 애니메이션을 구현하기 위해 연구되었다. 그림 4는 본 논문의 기법을 이용하여 생성한 깃발을 실제 게임 프로그램에 통합한 결과로 게임의 상호작용성을 전혀 해치지 않으면서 깃발 모델을 생성할 수 있었다. 그림에 나타난 깃발 모델은 사용자의 동작에 따라 이동하는 차량에 부착되어 있는 것으로 다양한 사용자의 조작에 부합하는 자연스러운 깃발 동작을 생성할 수 있었다.

그림 5는 자체 충돌을 처리하면서 바람에 날리는 옷감 애니메이션을 생성한 결과이다. 그림에서 볼 수 있는 바와 같이, 매우 사실적인 결과를 생성할 수 있었으며 이 결과는 일반적 PC 환경에서 실시간에 생성될 수 있다.

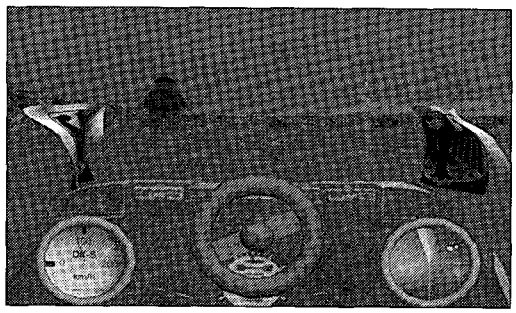

그림 4. 게임환경에 적용된 옷감 애니메이션
Fig. 4 Cloth Animation in Game Environments

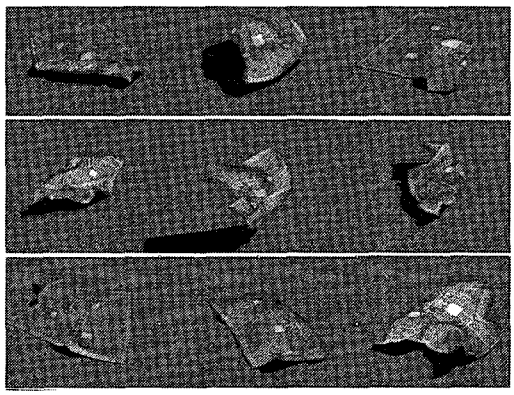

그림 5. 충돌 처리를 고려한 옷감 애니메이션
Fig. 5 Cloth Animation with Collision Handling

VI. 결론

본 논문에서는 옷감 객체의 움직임을 가상현실 환경에서 실시간에 생성할 수 있는 효율적인 기법을 제안하였다. 이전에 제안된 여러 가지 실시간 기법들은 힘의 미분치를 지나치게 근사하여 안정성은 보장하는 반면 동작의 사실성이 크게 떨어지는 단점을 가지고 있었다. 이러한 한계를 극복하기 위해 본 논문에서 제안된 기법은 옷감 애니메이션의 사실성과 시간 효율성을 높이기 위해 정확성을 최대한 유지하면서 수치적분의 해를 효과적으로 근사하고 있다. 또한, 본 연구에서 제안된 기법이 사용하는 힘의 미분치는 이전의 근사

기법과 달리 가능한 정확한 값을 갖도록 하여 움직임의 사실성을 더욱 높였으며, 정확한 힘 미분치를 사용할 경우 스프링 수축시 발생하는 바람직하지 못한 상황을 피하기 위해 수축시 미분치를 계산할 때 언제나 현재의 스프링 길이가 스프링의 휴지 상태 길이가 되도록 하는 방법을 채택하여 자연스러운 동작을 유지하면서도 시스템의 실패를 회피할 수 있도록 하였다.

본 논문에서 제안된 기법은 사실적인 옷감 애니메이션을 매유 효율적으로 생성할 수 있도록 암시적 적분법에 기반하고 있지만 명시적 기법처럼 쉽게 병렬화할 수 있는 특징을 갖는다. 따라서 제안된 기법 자체의 효율성과 함께 병렬화를 통해 더욱 높은 성능을 얻을 수 있다.

옷감의 동작을 효율적으로 생성하기 위한 최적화 및 병렬화 기법과 함께 본 논문에서는 옷감의 자체 충돌을 빠르게 감지하고 해결하기 위한 기법도 제안하였다.

참고문헌

[1] David Baraff and Andrew Witkin. Large steps in cloth simulation. *Proceedings of SIGGRAPH 98*, pp. 43-54, 1998.

[2] A. Vlachos, J. Peters, C. Boyd, and J. Mitchell. Curved PN triangles. *Symposium on Interactive 3D Graphics 2001*, pp. 159-166, 2001.

[3] Kwang-Jin Choi and Hyeong-Seok Ko. Stable but responsive cloth. *ACM Transactions on Graphics: Proceedings of SIGGRAPH 2002*. pp. 604-611. 2002.

[4] Frederic Cordier and Nadia Magnenat- Thalmann. Realtime animation of dressed virtual humans. *Proceedings of Eurographics 2002*, 2002.

[5] Mathieu Desbrun, Peter Schroder, and Alan Barr. Interactive animation of structured deformable objects. *Graphics Interface '99*, pp. 1-8, 1999.

[6] Young-Min Kang, Jeong-Hyeon Choi, Hwan-Gue Cho, and Chan-Jong Park. An efficient animation of wrinkled cloth with approximate implicit integration. *The Visual Computer Journal*, 17(3):147-157, 2001.

[7] M. Kass. An introduction to continuum dynamics for computer graphics. *In SIGGRAPH Course Note: Physically-based Modelling.* ACM SIGGRAPH, 1995.

[8] Masaki Oshita and Akifumi Makinouchi. Real-time cloth simulation with sparse particles and curved faces. *Proc. of Computer Animation 2001*, pp. 220-227, November 2001.

[9] Xavier Provot. Collision and self-collision handling in cloth model dedicated to design. *Computer Animation and Simulation '97*, pp. 177-190, September 1997.

[10] Pascal Volino and Nadia Magnenat-Thalmann. Efficient self-collision detection on smoothly discretized surface animations using geometrical shape regularity. *Computer Graphics Forum*, 13(3):155-166, 1994.

저자약력

강영민(Young-Min Kang)

1996 부산대학교 전산학과 이학사
1999 부산대학교 전산학과 이학석사
2003 부산대학교 전산학과 이학박사
2002년~2002년 제네바대학 MIRALab
2003년~2005년 한국전자통신연구원
2005년~ 동명정보대학교 게임공학과
※관심분야 : 컴퓨터 그래픽스, 물리기반 애니메이션, 게임 프로그래밍