Complex Deformable Objects in Virtual Reality

Young-Min Kang[*]
Department of Computer Science
Pusan National University
ymkang@pearl.cs.pusan.ac.kr

Hwan-Gue Cho
Department of Computer Science
Pusan National University
hgcho@pearl.cs.pusan.ac.kr

ABSTRACT

In this paper, we present a real-time animation technique for deformable objects based on mass-spring models in virtual reality environments. Many researchers have proposed various techniques for representing the motion and the appearance of deformable objects. However, the animation of deformable objects in virtual reality environments is still a hard problem. One of the most intensively studied deformable objects is virtual cloth. The difficulties in cloth animation mainly lie in the fact that cloth simulation easily tends to become unstable. Although the implicit method can make the simulation stable [1], it is still impossible to generate interactive animation when the number of mass-points is sufficiently large enough to represent realistic appearance. There have been a few efficient solutions for real-time animation [5, 7, 10]. However, the previous techniques for real-time cloth animation are not capable of generating the plausible motion and appearance because the methods are based on excessive approximations. This paper proposes an efficient technique that can generate realistic animation of complex cloth objects, and the technique makes it possible to integrate realistic deformable objects into the virtual reality systems without violating the interactivity of the system. The proposed method is based on the implicit integration in order to keep the system stable, and the solution of the linear system involved in the implicit integration is efficiently approximated in real-time.

Categories and Subject Descriptors

I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*virtual reality*

General Terms

Algorithms, Performance

Keywords

cloth animation, real-time animation, virtual reality

[*]GALab, Department of Computer Science, Pusan National University, Jangjeon-dong, Geumjeong-gu, Pusan, 609-735, KOREA

1. INTRODUCTION

Many problems in computer graphics are finally reduced to trade-offs between accuracy and efficiency, and the real-time animation of flexible object is also the arbitration of the accuracy and the efficiency. Many researchers have proposed various methods for reproducing the accurate behavior of deformable objects, and it is possible to generate realistic reproduction of the appearance and the movement of the cloth on the computer displays, with the noble contributions by various researchers [2, 3, 6, 13, 12, 14, 16, 17]. However, the realistic simulation of cloth requires a large amount of computations because the numerical integration easily tends to become unstable [1, 18]. Therefore, it is very difficult to produce the animation of deformable objects in real-time environments such as VR or interactive game systems. Although there have been various models and techniques for the animation of deformable objects, the real-time animation of realistic cloth is still extremely difficult because the accurate method cannot achieve real-time performance if the structure of virtual model is complex enough for the realistic appearance of cloth.

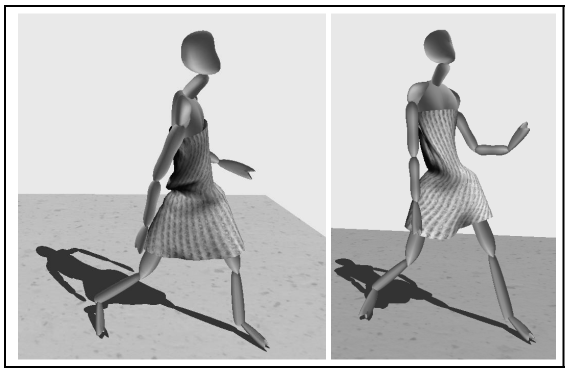

Figure 1: Dressed virtual character model

Many researchers have endeavored to devise efficient methods for cloth simulation or animation, and one of the most important advances in recent years is the use of implicit integration with large steps [1]. The implicit integration is unconditionally stable during the simulation, and the stability makes it possible to efficiently generate the cloth animation by using large time-steps. However, the implicit integration method requires the solution of a linear system, which involves a large matrix with a size proportional to the square of the number of mass-points [1, 5, 9]. Therefore, it is impossible to generate real-time animation of cloth model even with the implicit method when the number of mass-points is sufficiently large enough to represent realistic wrinkles. Some researchers have tried to devise efficient methods that can interactively manipulate the deformable objects in virtual environments, and a few works have been proposed to surmount the limitation of the implicit method [5, 7]. The previous techniques for real-time animation sacrificed the accuracy in order to achieve real-time or interactive performance. Desbrun et al. proposed an efficient technique that approximates the matrix involved in the linear system as a constant matrix and pre-computes the inverse matrix [5]. They applied the pre-computed inverse matrix as a force filter at every time-step in order to calculate the states of the cloth model at interactive rates. Although this method is more efficient than the original implicit method, it also involves O(n^2)-sized matrix where n is the number of mass-points in the mass-spring model, and the pre-computed inverse matrix is usually a dense matrix. Therefore, it is impossible to generate real-time cloth animation when the number of mass-points is large enough to represent the appearance of the real cloth.

A stable method that approximates the solution with a direct update formula has been also introduced. No matrix operations are involved in the method so that the method can update the state of the deformable object models in O(n^2) time with time-steps of arbitrary sizes [7]. However, the method excessively damps the motion in order to maintain the stability and efficiency of the computation. Therefore, the method generates slower motion than the real motion of cloth when high stiffness or large time-step is used.

Oshita and Makinouchi have proposed a geometric approach that generates wrinkly details on cloth models composed of sparse particles [10]. The method produces smooth cloth-like surfaces based on PN triangulation proposed in [15]. However, the geometric approach cannot produce the realistic appearance of cloth because the geometric interpolation of the initial coarse mesh does not take the physical reality into account.

Recently, Cordier and Magnenat-Thalmann proposed real-time animation techniques for fully dressed virtual human [4]. The fundamental idea of their method is the employment of different methods for different parts of the dress. The hybrid technique uses geometric deformation for the cloth that is tightly attached to the skin, and catenary curve for efficient animation of the sleeves. However, the method used conventional simulation method for the floating cloth like long skirt, and the animation of the floating cloth is the bottleneck of the performance.

Immersive virtual reality environments require realistic scene that can interact with user behaviors in real-time. Therefore, the deformable objects in virtual reality systems must be animated in an efficient way, while keeping the structure of the model complex enough to guarantee the plausible appearance. In this paper, we propose a real-time animation technique that can generate the motion of complex deformable objects such as the model shown in Fig. 1. The proposed method exploits the stability of the implicit integration and efficiently approximates the solution of the linear system. The proposed method can be successfully integrated into virtual reality systems in order to increase the realism of virtual environments without violating the interactivity of the system.

2. PROBLEM FORMULATION

One of the major difficulties in cloth animation lies in the stability of the system. When a simple numerical integration scheme like explicit Euler integration is employed, a small time interval must be used in order to keep the system stable. The fact that cloth animation requires very small time intervals makes it impossible to generate the motion of deformable objects in real-time. The stability problem can be solved with implicit methods. However, the implicit integration scheme is finally reduced to a linear system solving problem that involves O(n^2)-sized matrix when a cloth model with n mass-points is considered. Fortunately, the matrix is in general so sparse that we can efficiently solve the linear system with iterative methods. Although the iterative schemes enable us to solve the linear system in an efficient way, the exact solution requires heavy computations when the cloth model is complex enough to represent the realistic details.

In this paper, we consider a mass-spring based cloth model, which is the most intuitive and simple model for efficient animation of cloth. The mass-spring model assumes that an object is composed of mass-points and spring edges, where each spring connects two mass-points. The force exerted by the springs attached to each mass-point determines the motion of the mass-points. The spring force can be easily computed with Hooke's law, and the movement of a particle can be described with Newton's second law. Therefore, the cloth animation is the numerical integration of the force for computing the new velocity and the integration of the new velocity for computing the new position of each mass-point.

However, the instability problem disables us to employ a simple explicit integration method. If the animation system has time constraints like in VR environments, the instability problem must be solved in order to use large time steps. The instability can be avoided with implicit methods. Therefore, we can stably animate the mass-spring based objects by integrating the spring forces with the following backward Euler method [5, 8]:

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

where h denotes the time interval, and $\mathbf{v}$ denotes the vector of velocities of the mass-points. Similarly, $\mathbf{f}$ and $\mathbf{x}$ are the vectors of forces and locations of the mass-points respectively. The matrix M is the mass matrix. The superscripts t and $t+h$ denote time, and semantically mean the current state and the next state respectively.

The vectors ($\mathbf{v}$, $\mathbf{f}$, and $\mathbf{x}$) and the matrix M can be described as follows:

$$\mathbf{f} = [\mathbf{f}_1, \mathbf{f}_2, \cdots, \mathbf{f}_n]^{\mathbf{T}}, \qquad \mathbf{x} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n]^{\mathbf{T}}$$

$$\mathbf{v} = [\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_n]^{\mathbf{T}}, \quad \mathrm{M}_i = \begin{bmatrix} m_i & 0 & 0 \\ 0 & m_i & 0 \\ 0 & 0 & m_i \end{bmatrix}$$

$$\mathrm{M} = \begin{bmatrix} \mathrm{M}_1 & 0 & \cdots & 0 \\ 0 & \mathrm{M}_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & \mathrm{M}_n \end{bmatrix} \tag{2}$$

where $\mathbf{f}_i$, $\mathbf{x}_i$, $\mathbf{v}_i$, and m_i denote the force, location, velocity, and mass of the mass-point i respectively.

The objective of computation is to find $\mathbf{x}^{t+h}$ (i.e., the new positions of the mass-points at the next step), and the vector $\mathbf{x}^{t+h}$ can be easily determined when we know $\mathbf{v}^{t+h}$ (the next velocities of the mass-points). The next velocity vector can be computed if we find the velocity change vector $\Delta\mathbf{v}^{t+h}$ that denotes $\mathbf{v}^{t+h} - \mathbf{v}^{t}$. Therefore, the cloth animation with Eq. 1 is eventually reduced to finding $\Delta\mathbf{v}^{t+h}$ as follows:

$$\Delta\mathbf{v}^{t+h} = h\mathrm{M}^{-1}\mathbf{f}^{t+h} \tag{3}$$

Although the mass-spring based animation can be described as a simple equation like Eq. 3, the real-time animation is not an easy problem. The difficulty arises from the unknown force vector at time $t + h$. Since Hooke's law can compute only $\mathbf{f}^t$, the force at the next time step should be approximated as follows:

$$\mathbf{f}^{t+h} = \mathbf{f}^t + \frac{\partial \mathbf{f}}{\partial \mathbf{x}} \Delta \mathbf{x}^{t+h} = \mathbf{f}^t + \mathbf{J} \Delta \mathbf{x}^{t+h} \tag{4}$$

where $\mathbf{J}$ denotes the Jacobian matrix of the force vector with respect to the position vector, and $\Delta \mathbf{x}^{t+h}$ is the positional change from time t to $t + h$. The size of the matrix $\mathbf{J}$ is $n \times n$, and each component of $\mathbf{J}$ is a 3×3 submatrix.

Because the positional change $\Delta \mathbf{x}^{t+h}$ can be rewritten as $(\mathbf{v}^t + \Delta \mathbf{v}^{t+h})h$, the force at the next state can be expressed as follows:

$$\mathbf{f}^{t+h} = \mathbf{f}^t + \mathbf{J}(\mathbf{v}^t + \Delta \mathbf{v}^{t+h})h \tag{5}$$

With the force approximation shown in Eq. 5, the animation of the mass-spring model is finally reduced to a linear system solving as follows:

$$\Delta \mathbf{v}^{t+h} = h\mathrm{M}^{-1}\mathbf{f}^t + h^2\mathrm{M}^{-1}\mathbf{J}\mathbf{v}^t + h^2\mathrm{M}^{-1}\mathbf{J}\Delta \mathbf{v}^{t+h} \tag{6}$$

Rearrangement of Eq. 6 to the linear system yields the linear system in the standard form as follows:

$$(\mathbf{I} - h^2\mathrm{M}^{-1}\mathbf{J})\Delta \mathbf{v}^{t+h} = h\mathrm{M}^{-1}\mathbf{f}^t + h^2\mathrm{M}^{-1}\mathbf{J}\mathbf{v}^t \tag{7}$$

where $\mathbf{I}$ is the identity matrix.

By multiplying the mass-matrix M to the both side, the problem can be finally expressed as follows:

$$(\mathrm{M} - h^2\mathbf{J})\Delta \mathbf{v}^{t+h} = h\mathbf{f}^t + h^2\mathbf{J}\mathbf{v}^t \tag{8}$$

The stable animation of the mass-spring model can be achieved by solving the linear system shown in Eq. 8. However, the stability is not the complete solution to the real-time animation. Since the matrix $\mathrm{M} - h^2\mathbf{J}$ is sparse and symmetric, the linear system can be solved efficiently. However, the size of the matrix rapidly increases when the number of mass-points increases. In order to guarantee realistic virtual environments, sufficiently large number of mass-points must be used. However, the exact solution of Eq. 8 requires too heavy computation if the system is supposed to satisfy certain time constraints.

3. REAL-TIME CLOTH ANIMATION

Although the size of the matrix $\mathrm{M} - h^2\mathbf{J}$ in Eq 8 rapidly increases when the number of mass-points increases, the linear system can be efficiently solved because of the properties of the matrix. In general, the matrix is so sparse that iterative methods methods can be successfully employed. Although the sparseness of the matrix does not guarantee the real-time performance of the conventional linear system solvers, it is the most important property. Our real-time animation method also exploits the sparseness of the matrix and other properties in order to compute the next state as fast as possible.

3.1 Iterative Approximation of the Solution

Let $\mathbf{f}_i^j$ denote the force on the mass-point i caused by the spring between the mass-points i and j. If the spring force is simply modeled with Hooke's law and κ_{ij} denotes the spring constants between the mass-points, $\mathbf{f}_i^j$ can be described as follows:

$$\mathbf{f}_i^j = \kappa_{ij}(|\mathbf{x}_j - \mathbf{x}_i| - l_{ij}^0)\frac{(\mathbf{x}_j - \mathbf{x}_i)}{|\mathbf{x}_j - \mathbf{x}_i|} \tag{9}$$

The total spring force on the i-th mass-point is then the sum of all the spring forces caused by all the springs linked to the mass-point. Let l_{ij}^0 denote the rest length of the spring between the i-th and j-th mass-points. The derivative of force on the mass-point i with respect to the location of the mass-point j can then be expressed as follows:

$$\frac{\partial \mathbf{f}_i}{\partial \mathbf{x}_j} = \kappa_{ij}\frac{|\mathbf{x}_j - \mathbf{x}_i| - l_{ij}^0}{|\mathbf{x}_j - \mathbf{x}_i|}\mathbf{I}_3 + \kappa_{ij}\frac{l_{ij}^0}{|\mathbf{x}_j - \mathbf{x}_i|}\mathbf{I}_3\left(\frac{(\mathbf{x}_j - \mathbf{x}_i)(\mathbf{x}_j - \mathbf{x}_i)^{\mathrm{T}}}{|\mathbf{x}_j - \mathbf{x}_i|^2}\right) \tag{10}$$

where $\mathbf{I}_3$ denotes the 3×3 identity matrix.

However, the force derivative in Eq. 10 has serious problem when the distance between the mass-points i and j is smaller than l_{ij}^0. When the distance between two mass-points approaches to 0, the derivative of the force contains extremely large values as components, and it makes the system fail. In order to avoid the undesirable situations, we approximated the force derivative by assuming l_{ij}^0 has the same value as $|\mathbf{x}_j - \mathbf{x}_i|$. Then the components of the force derivative are guaranteed to have smaller values than the spring constant of the spring. We found this assumption enables the system to avoid the failure, and the result animation is plausible enough. Therefore, the force derivative used for our method was approximated as follows:

$$\frac{\partial \mathbf{f}_i}{\partial \mathbf{x}_j} = \mathbf{J}_{ij} = \kappa_{ij}\left(\frac{(\mathbf{x}_j - \mathbf{x}_i)(\mathbf{x}_j - \mathbf{x}_i)^{\mathrm{T}}}{|\mathbf{x}_j - \mathbf{x}_i|^2}\right) \tag{11}$$

Henceforth, $\mathbf{J}_{ij}$ denotes the approximate force derivative shown in Eq. 11, and $\mathbf{J}_{ii}$ denotes the negated sum of the approximate force derivatives (i.e., $\mathbf{J}_{ii} = -\sum_{(i,j)\in E}\mathbf{J}_{ij}$). Since M is a diagonal matrix and the Jacobian matrix $\mathbf{J}$ is sparse, it is obvious that $\mathrm{M} - h^2\mathbf{J}$ is also sparse. For the simplicity, let us define a matrix $\mathbf{W}$ as follows:

$$\mathbf{W} = \mathrm{M} - h^2\mathbf{J} \tag{12}$$

The components of the matrix $\mathbf{W}$ are 3×3 submatrices, which can be easily computed. The component in the i-th row and j-th column, $\mathbf{W}_{ij}$, is $-h^2\mathbf{J}_{ij}$ if i and j are different from each other. The diagonal components, $\mathbf{W}_{ii}$, can be computed as $\mathrm{M}_i - h^2\mathbf{J}_{ii}$. In other words, the i-th diagonal component is the sum of the off-diagonal components in the row and mass submatrix of the i-th mass-point. Therefore, the diagonal components can be expressed as $\mathrm{M}_i + h^2\sum_{(i,j)\in E}\mathbf{J}_{ij}$ where E is the set of springs. The matrix $\mathbf{W}$ in the linear system has many important properties that can be exploited for better performance. The major properties of the matrix $\mathbf{W}$ are as follows:

- Block Matrix: The components vectors $\mathbf{f}$, $\mathbf{v}$, and $\mathbf{x}$ are all 3-dimensional vectors as described in Eq. 2. Therefore, the components of the Jacobian matrix $\mathbf{J}$ must be 3×3 matrices. We can consider the matrix in the linear system as a block matrix composed of 3×3 submatrices. The previous approximate methods [5, 7] approximated these submatrices as scalar values, and the result is stable enough to use arbitrarily large time steps. However, the excessive approximation introduces unnecessary damping in every direction so that the animation results by these methods are not so plausible. Our technique does not employ the approximation of these submatrices into scalar values in order to maintain the plausibility of the cloth animation.

- Symmetric Matrix: It is obvious that the matrix is symmetric because $\mathbf{J}_{ij}$ and $\mathbf{J}_{ji}$ are identical. One additional property is that each submatrix is also a symmetric matrix. The symmetry of the whole matrix and submatrices enables us to efficiently store the matrix, and the inverses of the submatrices can be computed in a more efficient way because they are 3×3 symmetric matrices.

- Sparse Matrix: The matrix $\mathbf{W}$ is sparse because M is diagonal matrix and a component of the Jacobian matrix $\mathbf{J}_{ij}$ has a non-zero matrix only when the mass-point i and j are linked with a spring. Therefore, the number of non-zero components (3×3 submatrices) in the i-th row of the matrix is simply one larger than the number of springs linked to the mass-point i. In general, only a small number of springs are linked to a specific mass-point, and it is not affected by the complexity of the model, i.e., the number of total mass-points in the cloth model. Therefore, each row in the matrix has a limited number of effective components, and the sparseness of the matrix increases as the number of total mass-points increases.

The vector $(h\mathbf{f}^t + h^2\mathbf{J}\mathbf{v}^t)$ in the right side of the Eq. 8 can be described as $h(\mathbf{f}^t + h\mathbf{J}\mathbf{v}^t)$ and the additional forces $h\mathbf{J}\mathbf{v}^t$ can be considered as viscosity forces [5]. Because of the sparseness of the Jacobian matrix, we can efficiently compute the vector $h\mathbf{J}\mathbf{v}^t$. If we denote the sum of spring forces and viscosity forces $(\mathbf{f}^t + h\mathbf{J}\mathbf{v}^t)$ as internal forces $\tilde{\mathbf{f}}$, each component (3-dimensional vector) of internal force vector can be calculated with the consideration of linked mass-points as follows:

$$\begin{aligned} \tilde{\mathbf{f}}_i^t &= \mathbf{f}_i^t + h \sum_{(i,j)\in E} \mathbf{J}_{ij}\mathbf{v}_j^t + h\mathbf{J}_{ii}\mathbf{v}_i^t \\ &= \mathbf{f}_i^t + h \sum_{(i,j)\in E} \mathbf{J}_{ij}(\mathbf{v}_j^t - \mathbf{v}_i^t) \end{aligned} \tag{13}$$

Then, the linear system in Eq. 8 can be rewritten as follows:

$$\mathbf{W}\Delta\mathbf{v}^{t+h} = h\tilde{\mathbf{f}}^t \tag{14}$$

Due to the properties of the matrix W, the linear system of Eq. 14 can be expressed as following n equations:

$$\begin{aligned} \mathbf{W}_{11}\Delta\mathbf{v}_1^{t+h} &= h\tilde{\mathbf{f}}_1^t + h^2 \sum_{(1,j)\in E} \mathbf{J}_{1j}\Delta\mathbf{v}_j^{t+h} \\ \mathbf{W}_{22}\Delta\mathbf{v}_2^{t+h} &= h\tilde{\mathbf{f}}_2^t + h^2 \sum_{(2,j)\in E} \mathbf{J}_{2j}\Delta\mathbf{v}_j^{t+h} \\ &\vdots \\ \mathbf{W}_{nn}\Delta\mathbf{v}_n^{t+h} &= h\tilde{\mathbf{f}}_n^t + h^2 \sum_{(n,j)\in E} \mathbf{J}_{nj}\Delta\mathbf{v}_j^{t+h} \end{aligned} \tag{15}$$

All the equations in Eq. 15 have a similar form, and the i-th equation can be rearranged to highlight the velocity change of the i-th mass-point as follows:

$$\Delta\mathbf{v}_i^{t+h} = \mathbf{W}_{ii}^{-1}(h\tilde{\mathbf{f}}_i^t + h^2 \sum_{(i,j)\in E} \mathbf{J}_{ij}\Delta\mathbf{v}_j^{t+h}) \tag{16}$$

The matrices ($\mathbf{M}_i$, $\mathbf{J}_{ii}$, and $\mathbf{J}_{ij}$) involved in Eq. 16 are 3×3 symmetric matrices, and the computation of the vector $\sum \mathbf{J}_{ij}\Delta\mathbf{v}_j^{t+h}$ requires a small amount of computations if we know the velocity change of linked mass-points at the next time step $\Delta\mathbf{v}_j^{t+h}$. Therefore, the motion of the cloth model can be computed in O($n + e$) time if the velocity change of each mass-point can be computed independently with Eq. 16. However, each equation, unfortunately, cannot be computed independently because it is related with the result of some other equations ($\Delta\mathbf{v}_j^{t+h}$). Our method approximates the solution of the linear system by iteratively computing the velocity change of each mass-point with Eq. 16. In fact, Eq. 16 is the Jacobi iteration scheme if we introduce another superscript k as the number of iterations. Let $\Delta\mathbf{v}_j^{t+h^{(k)}}$ be the result after the k iterations, and suppose that the initial guess of velocity change of each mass-point is simply $\Delta\mathbf{v}_i^{t+h} = \mathbf{W}_{ii}^{-1}h\tilde{\mathbf{f}}_i^t$ by ignoring the velocity changes of linked mass-points. The iterative scheme can then be described as follows:

$$\begin{aligned} \Delta\mathbf{v}_i^{t+h^{(0)}} &= \mathbf{W}_{ii}^{-1}h\tilde{\mathbf{f}}_i^t \\ \Delta\mathbf{v}_i^{t+h^{(k+1)}} &= \mathbf{W}_{ii}^{-1}(h\tilde{\mathbf{f}}_i^t + h^2 \sum_{(i,j)\ inE} \mathbf{J}_{ij}\Delta\mathbf{v}_j^{t+h^{(k)}}) \end{aligned} \tag{17}$$

With a small number of iterations, we can obtain the approximate solution of the linear system in real-time. Since the iterative scheme is based on the implicit method, the animation result is sufficiently stable enough to be used in real-time system.

During the iterative update of $\Delta\mathbf{v}_i^{t+h}$, the matrix $\mathrm{M}_i - h^2\mathbf{J}_{ii}$ (i.e., $\mathbf{W}_{ii}$) and the vector $h\tilde{\mathbf{f}}_i$ remain constant. Therefore, these matrix and vector need to be computed only once during the iterations for one time-step. Let us denote the constant matrix and vector as $\mathbf{W}_i$ and $\mathbf{p}_i$ as follows:

$$\begin{aligned} \mathbf{W}_i = &\ \mathrm{M}_i - h^2\mathbf{J}_{ii} &\in \mathbf{R}^{3\times3} \\ \mathbf{p}_i = &\ h\mathbf{W}_i^{-1}\tilde{\mathbf{f}}_i &\in \mathbf{R}^3 \end{aligned} \tag{18}$$

The iterative scheme in Eq. 17, then, can be more compactly expressed as follows:

$$\begin{aligned} \Delta\mathbf{v}_i^{t+h^{(0)}} &= \mathbf{p}_i \\ \Delta\mathbf{v}_i^{t+h^{(k+1)}} &= \mathbf{p}_i + h^2\mathbf{W}_i^{-1} \sum_{(i,j)\ inE} \mathbf{J}_{ij}\Delta\mathbf{v}_j^{t+h^{(k)}} \end{aligned} \tag{19}$$

In most cases, the approximate solution after only one iterative update ($\Delta\mathbf{v}_j^{t+h^{(1)}}$) was stable and plausible enough to be used in real-time system. Therefore, we approximated the velocity change of each mass-point as follows:

$$\Delta\mathbf{v}_i^{t+h} \simeq \Delta\mathbf{v}_i^{t+h^{(1)}} = \mathbf{p}_i + h^2\mathbf{W}_i^{-1} \sum_{(i,j)\in E} \mathbf{J}_{ij}\mathbf{p}_j \tag{20}$$

The fast approximation of the velocity change of each mass-point with Eq. 20 enables real-time cloth animation.

3.2 Damping

Although the implicit method is unconditionally stable during the simulation and the proposed approximate method shows a good stability property, the approximate solution obtained by Eq. 20 does not always guarantee the stability of the system. In order to increase the stability of the system and the realism of the motion, damping was adopted. The simplest damping model is to employ damping force vector $\mathbf{f}_d$ as follows:

$$\mathbf{f}_d = -C_d\mathbf{v} \tag{21}$$

where C_d is damping coefficient, which is a non-negative scalar value.

In our implementation, no damping force was explicitly applied to the mass-points. Instead, we introduce the damping effect when we approximate the force vector at the next time step. Therefore, the force at the next time step can be expressed as follows:

$$\begin{aligned}\mathbf{f}^{t+h} &= \mathbf{f}^t + \frac{\partial \mathbf{f}}{\partial \mathbf{x}}\Delta\mathbf{x}^{t+h} + \frac{\partial \mathbf{f}_d}{\partial \mathbf{v}}\Delta\mathbf{v}^{t+h} \\ &= \mathbf{f}^t + \mathbf{J}\Delta\mathbf{x}^{t+h} - C_d\mathbf{I}\Delta\mathbf{v}^{t+h} \\ &= \mathbf{f}^t + \mathbf{J}(\mathbf{v}^t + \Delta\mathbf{v}^{t+h})h - C_d\mathbf{I}\Delta\mathbf{v}^{t+h}\end{aligned} \quad (22)$$

Note that the damping affects only the diagonal components of the matrix. Therefore, the velocity change in Eq. 8 can be rewritten as follows:

$$(\mathbf{W} + hC_d\mathbf{I})\Delta\mathbf{v}^{t+h} = h\mathbf{f}^t + h^2\mathbf{J}\mathbf{v}^t \quad (23)$$

Eq. 23 implies that a simple addition of hC_d to the diagonal components of the 3×3 submatrix $\mathbf{W}_i$ generates damping effects. The simple damping model dissipates the energy and improves the stability of the system. While the damping effect can improve the stability, the plausibility of the motion can be impaired because the simple model slows the motion in any cases. In order not to damp the motion of a mass-point when its movement has the same direction and the same velocity as those of the linked mass-points, the damping model can be modified as follows:

$$\mathbf{f}_d = -C_d \sum_{(i,j)\in E} (\mathbf{v} - \mathbf{v}) \quad (24)$$

The sparseness of the matrix $\partial\mathbf{f}_d/\partial\mathbf{v}$ is exactly the same as the sparseness of $\mathbf{J}$ because the 3×3 submatrix $(\partial\mathbf{f}_d/\partial\mathbf{v})_{ij}$ is a non-zero matrix only when the mass-points i and j are linked by a spring edge. If two mass-points are linked, $(\partial\mathbf{f}_d/\partial\mathbf{v})_{ij}$ becomes $C_d\mathbf{I}_3$.

The diagonal submatrix $(\partial\mathbf{f}_d/\partial\mathbf{v})_{ii}$ can also be simply computed as $-n_iC_d\mathbf{I}_3$ where n_i is the number of mass-points linked to the mass-point i. We can clearly see that this damping model only affects the diagonal components of the non-zero submatrices. Therefore, the velocity change in Eq. 16 can be modified to include the damping effects without introducing any significant additional computations. For the simplicity, let us denote $(\mathbf{W}_i + n_ihC_d\mathbf{I}_3)^{-1}$ and $\mathbf{J}_{ij} + hC_d\mathbf{I}_3$ as $\mathbf{W}'_i$ and $\mathbf{J}'_{ij}$ respectively. Then, the velocity change can be computed as follows:

$$\Delta\mathbf{v}_i^{t+h} = \mathbf{W}'^{-1}_i(h\tilde{\mathbf{f}}_i^t + h^2\sum\mathbf{J}'_{ij} + \Delta\mathbf{v}_j^{t+h}) \quad (25)$$

The iterative update scheme can be constructed with a simple modification of the definition of $\mathbf{W}_i^{-1}$ and $\mathbf{p}_i$ in Eq. 18.

4. IMPLEMENTATION DETAILS

Iterative update or approximation with Eq. 19 and 20 enables us to efficiently manipulate deformable objects of high complexity that have not been available in current virtual reality systems. However, in the actual implementations, more things should be considered. In this section, some important issues for the implementation will be explained.

4.1 Jacobian Matrix and Storage

Although the size of the Jacobian matrix $\mathbf{J}$ is huge, it is very sparse. We can efficiently store the matrix by taking advantage of the sparseness with general techniques for sparse matrices. However, sparseness of the matrix is not the only property that can be exploited for efficient storage.

As mentioned early, the matrix is symmetric so that $\mathbf{J}_{ij}$ and $\mathbf{J}_{ji}$ are identical and have effective values only when the mass-point i and j are linked. Therefore, each effective submatrix of $\mathbf{J}$ except for the diagonal components ($\mathbf{J}_{ii}$) can be considered as an attribute of each spring. The number of diagonal submatrices $\mathbf{J}_{ii}$ is exactly the same as the number of mass-points. Therefore, each diagonal submatrix can then be considered as an attribute of each mass-point. Since the effective submatrices of the Jacobian matrix $\mathbf{J}$ are considered as attributes of either mass-points or spring edges, they can be easily stored and retrieved as member data of the mass-point and spring objects. The lower triangular components in submatrices are not stored because all the submatrices in our problem are symmetric.

```
Begin ComputeVelocityChange

    1. Compute $\mathbf{J}$
    2. Compute $\mathbf{W}_i^{-1}$ for each vertex
    3. Compute $\mathbf{p}_i$ for each vertex
    4. Compute $\Delta\mathbf{v}_i$ for each vertex

End ComputeVelocityChange
```

Table 1: The computation of velocity change

```
Begin ComputeJacobian
  for ( $i$ : from 1 to $n$) $\mathbf{J}_{ii} \leftarrow \begin{bmatrix} 0&0&0\\0&0&0\\0&0&0\end{bmatrix}$
  {$n(E)$ is the number of total spring edges}
  for ( $e$ : from 1 to $n(E)$ ) [
    {$v_1$ is the first mass-points of the spring edge $E(e)$}
    {$v_2$ is the second mass-points of the spring edge $E(e)$}
    $i \leftarrow E(e).v_1$
    $j \leftarrow E(e).v_2$
    $\mathbf{J}_{ij} \leftarrow \partial\mathbf{f}_i/\partial\mathbf{x}_j$
    $\mathbf{J}_{ii} \leftarrow \mathbf{J}_{ii} - \mathbf{J}_{ij}$
    $\mathbf{J}_{jj} \leftarrow \mathbf{J}_{jj} - \mathbf{J}_{ij}$
  ]
End ComputeJacobian
```

Table 2: Jacobian matrix computation

4.2 Computation of $\mathbf{W}_i^{-1}$

The proposed method is efficient because it computes the velocity change of each mass-point by considering only the linked mass-points. However, the method requires the system to compute the inverse matrix ($\mathbf{W}_i^{-1}$) of diagonal submatrices $\mathbf{W}_i$. Compared to the whole linear system solving, the burden of calculating the 3×3 inverse matrix is trivial. However, the implementation of real-time environments is often a struggle to obtain the efficiency with every possible optimization. Since $\mathbf{W}_i$ is a 3×3 symmetric matrix, the adjoint matrix and the determinant can be easily computed and the inverse matrix can be also efficiently computed.

4.3 Computation of Velocity Change

The animation technique proposed in this paper is finally reduced to finding the velocity change of each mass-point. The velocity change computation is composed of four modules as shown in Ta-

```
Begin Compute_$\mathbf{p}_i$

    for ( $i$ : from 1 to $n$)    $\mathbf{p}_i \leftarrow [0,0,0]^{\mathbf{T}}$
    {$n(E)$ is the number of total spring edges}
    for ( $e$ : from 1 to $n(E)$ ) [
      {$v_1$ is the first mass-points of the spring edge $E(e)$}
      {$v_2$ is the second mass-points of the spring edge $E(e)$}
      $i \leftarrow E(e).v_1$
      $j \leftarrow E(e).v_2$
      $\mathbf{p}_i \leftarrow \mathbf{p}_i + h^2\mathbf{J}_{ij}(\mathbf{v}_i - \mathbf{v}_j)$
      $\mathbf{p}_j \leftarrow \mathbf{p}_j + h^2\mathbf{J}_{ij}(\mathbf{v}_j - \mathbf{v}_i)$
    ]
    for ( $i$ : from 1 to $n$) $\mathbf{p}_i \leftarrow \mathbf{W}_i^{-1}(\mathbf{p}_i + h\mathbf{f}_i)$

End Compute_$\mathbf{p}_i$
```

Table 3: Computation of $\mathbf{p}_i$ vectors

number of vertices	148	405	1566	2128
number of edges	404	1161	4590	6231
number of triangles	257	757	3025	4104
update time (sec)	0.0008	0.0023	0.0108	0.0158

Table 4: Performance of the method on medium level PC

ble 1.

The first module computes the Jacobian matrix $\mathbf{J}$. Although the number of total 3×3 submatrices in the Jacobian matrix is n^2, we can compute the effective submatrices by considering the spring edges only. Therefore, the computation of Jacobian can be done as shown in Table 2. In order to simplify, the damping was not considered in the pseudo-code.

The second module for computing the inverse of the matrix $\mathbf{W}_i$ is straightforward and can be performed in O(n) time.

The third module computes the $\mathbf{p}_i$ that is defined in Eq. 18. The computation of $\mathbf{p}_i$ for each mass-point requires the consideration of the velocities of the linked mass-points. Therefore, the module can be implemented as shown in Table 3. Damping is ignored again in this pseudo-code. The fourth module that computes the velocity change of each mass-point can be simply implemented with the already obtained $\mathbf{W}_i$ and $\mathbf{p}_i$.

5. EXPERIMENTS

The proposed method has been successfully implemented for an interactive cloth animation system. The cloth animation system has been developed on PC environments. Even with the medium performance level PC, the method could generate real-time animation of virtual cloth with thousands of polygons.

For the animation of the deformable objects, collision resolution is also an important issue because deformable objects may encounter special collision cases where some parts of one object collide with each other. A few efficient methods have been already proposed for handling the self-collision [11, 16]. Usually, accurate collision handling methods require heavy computations. The collision handling was not a major subject of our research, and we employed a simplified collision resolution. During the experiments, we found that the cloth model dressed on a virtual human seldom experiences self-collision situation, and the self-collision is usually resolved immediately. Therefore, we did not consider the self-collision, and approximated the geometry of virtual human for the real-time collision handling.

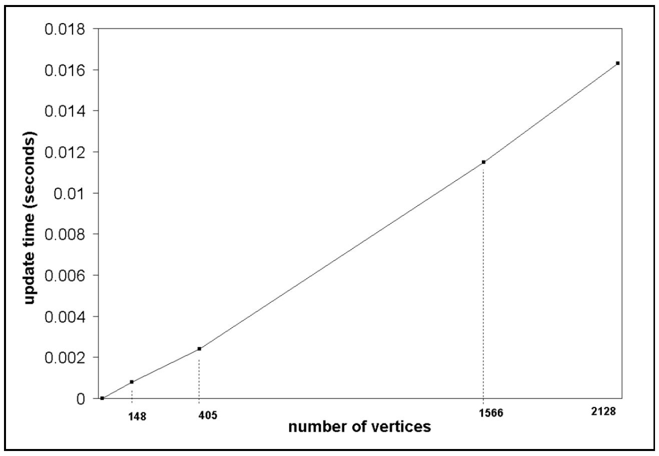

Figure 2: Update time according to the number of mass-points

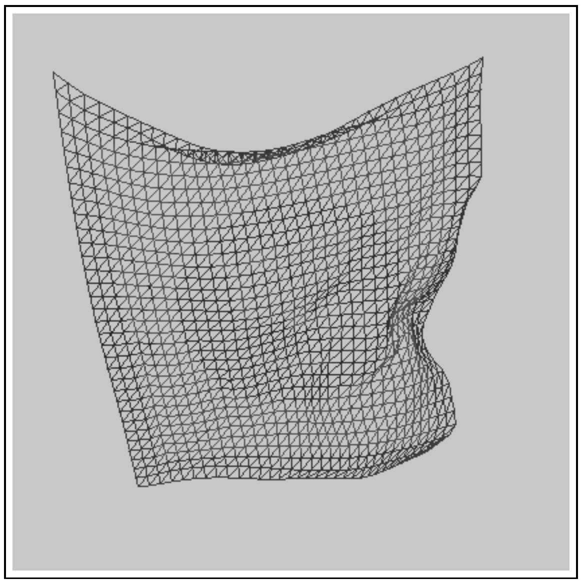

Figure 3: Complex mass-spring model with 2048 triangles

Table 4 shows the computational performance of the proposed method. We have tested the method in medium level PC environments (Pentium II 600 MHz), and the proposed method showed real-time performance for the animation of complex models with thousands of polygons. As shown in Table 4, it was possible to update the state of a complex cloth model with 3025 polygons (the third column) in 0.0108 seconds. Since the proposed method is stable enough to use large steps, it was also possible to generate the real-time animation of the complex model with 3025 polygons by using time-steps with the size of 1/120 second.

Because the update process for each mass-point considers only neighboring mass-points linked by springs, it is obvious that the time complexity of the proposed method is linearly proportional to the number of mass-points and spring edges. Fig. 2 visually shows the time required to compute the next state according to the number of mass-points. As shown in Fig. 2, the state update time of the proposed method was linearly proportional to the number of the mass-points.

Fig. 3 shows the complex mass-spring model for realistic cloth animation. The proposed method can generate real-time animation of such complex models without any significant decline of realism such as over-damping or over-stretching effects.

Fig. 4 shows the result of the real-time animation system. The proposed method was applied to a flag model composed of 2048 triangles. As shown in the Fig. 4, the proposed method generated plausible appearance of the cloth model in real-time. The size of the time step was 1/300 sec.

Fig. 5 shows the result of the interactive animation system implemented with the proposed method. The proposed method efficiently generated plausible animation of dressed human at interactive rate.

6. CONCLUSIONS

In this paper, we presented a new method to animate mass-spring models in real-time. The proposed method is stable and efficient enough to integrate realistic deformable object into virtual reality environments. The instability of a numerical integration method affects the overall performance of the cloth animation system because the unstable method requires extremely small time-steps. Therefore, real-time animation of a complex cloth model is not feasible with the unstable methods. For that reason, the stability is the first essential property of animation techniques in real-time environments such as VR systems.

The stability of the method has been obtained by employing the implicit integration method for deriving the iterative state-update scheme that computes the next state of each mass-point. Therefore, the method can produce cloth animation with large time-steps.

Although the stability is the essential property, it is not the sufficient property for the real-time animation. The stability of the implicit method requires heavy computational burdens to obtain the next state of each mass-point because it casts the original problem into a linear system problem. However, the linear system for mass-spring animation derived from the implicit method has many properties, which can be exploited for improving the performance of the animation system. The matrix involved in the linear system is a sparse, symmetric, and block matrix. The proposed method efficiently approximates the solution of the linear system by taking advantage of these properties. The proposed method independently computes the next state of each mass-point by considering only linked mass-points so that the computation of the next state of the whole cloth model can be achieved in O($n + e$) time where n is the number of the mass-points and e is the number of the spring edges in the cloth model. Therefore, the efficiency and the stability of the method enable real-time animation of deformable objects to be integrated in virtual environments.

Apart from the real-time performance, the proposed method has additional important advantages. The proposed method is intuitive and easy to implement. The method can be easily implemented with simple modules for inverse computation of 3×3 matrices and multiplication of 3×3 matrices and 3 dimensional vectors so that the method does not require developers to struggle with optimization of the storage and the performance of the linear system solver. The easiness of the implementation guarantees that the implementation of the proposed method in various real-time or interactive systems will be successfully achieved with minimum efforts.

7. ACKNOWLEDGMENTS

Authors are thankful to Frédéric Cordier and Thomas Di Giacomo for their helpful comments. This research was supported in part by the Ministry of Information and Communication of Korea: The Support Project for University Foundation Research 2002, supervised by Institute of Information Technology Assessment (IITA).

8. REFERENCES

[1] D. Baraff and A. Witkin. Large steps in cloth simulation. *Proceedings of SIGGRAPH 98*, pages 43–54, July 1998.

[2] D. E. Breen, D. H. House, and M. J. Wozny. Predicting the drape of woven cloth using interacting particles. *Proceedings of SIGGRAPH '94*, pages 365–372, July 1994.

[3] M. Carignan, Y. Yang, N. M. Thalmann, and D. Thalmann. Dressing animated synthetic actors with complex deformable clothes. *Computer Graphics (Proceedings of SIGGRAPH 92)*, 26(2):99–104, July 1992.

[4] F. Cordier and N. Magnenat-Thalmann. Real-time animation of dressed virtual humans. *Proceedings of Eurographics 2002*, 2002.

[5] M. Desbrun, P. Schröder, and A. Barr. Interactive animation of structured deformable objects. *Graphics Interface '99*, pages 1–8, June 1999.

[6] B. Eberhardt, A. Weber, and W. Strasser. A fast, flexible particle-system model for cloth draping. *IEEE Computer Graphics & Applications*, 16(5):52–59, September 1996.

[7] Y.-M. Kang, J.-H. Choi, H.-G. Cho, and C.-J. Park. An efficient animation of wrinkled cloth with approximate implicit integration. *The Visual Computer*, 17(3):147–157, 2001.

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

[9] S. Nakamura. Initial value problems of ordinary differential equations. In *Applied Numerical Methods with Software*, pages 289–350. Prentice-Hall, 1991.

[10] M. Oshita and A. Makinouchi. Real-time cloth simulation with sparse particles and curved faces. *Proc. of Computer Animation 2001*, pages 220–227, November 2001.

[11] X. Provot. Collision and self-collision handling in cloth model dedicated to design. *Computer Animation and Simulation '97*, pages 177–190, September 1997.

[12] D. Terzopoulos and K. Fleischer. Modeling inelastic deformation: Viscoelasticity, plasticity, fracture. *Computer Graphics (Proceedings of SIGGRAPH 88)*, 22(4):269–278, August 1988.

[13] D. Terzopoulos, J. Platt, A. Barr, and K. Fleischer. Elastically deformable models. *Computer Graphics (Proceedings of SIGGRAPH 87)*, 21(4):205–214, July 1987.

[14] D. Terzopoulos and A. Witkin. Physically based models with rigid and deformable components. *IEEE Computer Graphics & Applications*, 8(6):41–51, November 1988.

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

[16] P. Volino, M. Courshesnes, and N. Magnenat-Thalmann. Versatile and efficient techniques for simulating cloth and other deformable objects. *Proceedings of SIGGRAPH 95*, pages 137–144, August 1995.

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

[18] A. Witkin and D. Baraff. Differential equation basics. In *SIGGRAPH Course Note: Physically-based Modelling*. ACM SIGGRAPH, 1994.

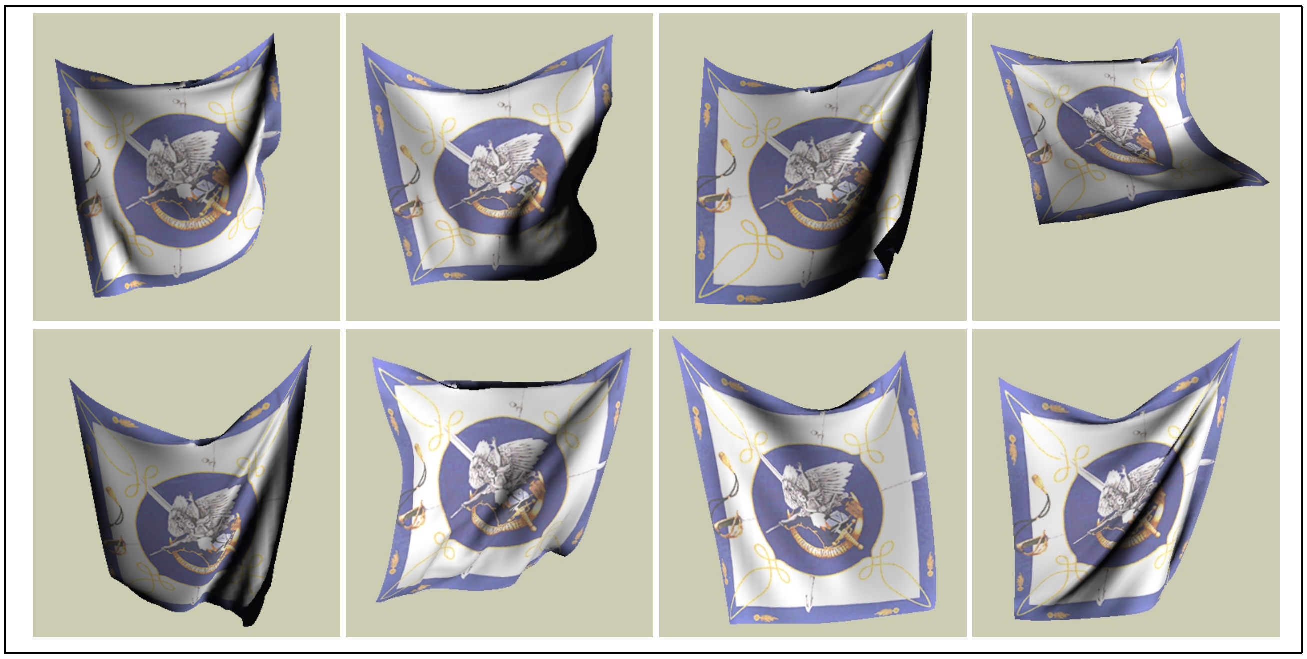

Figure 4: Real-time animation of the complex mass-spring model

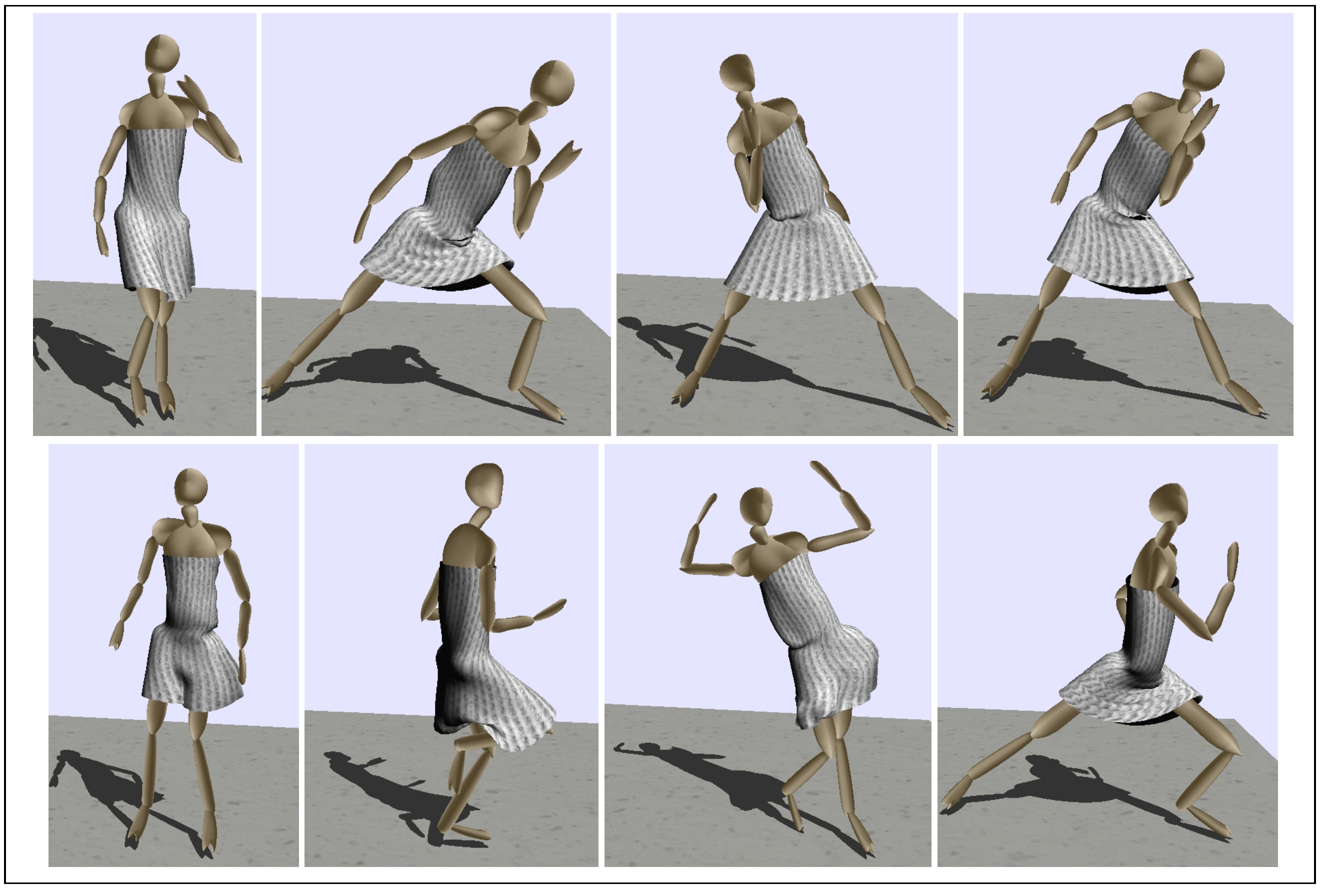

Figure 5: Interactive animation of a dressed character