Fast Simulation of Skeleton-Driven Deformable Body Characters

JUNGGON KIM and NANCY S. POLLARD
Carnegie Mellon University

We propose a fast physically-based simulation system for skeleton-driven deformable body characters. Our system can generate realistic motions of self-propelled deformable body characters by considering the two-way interactions among the skeleton, the deformable body, and the environment in the dynamic simulation. It can also compute the passive jiggling behavior of a deformable body driven by a kinematic skeletal motion. We show that a well-coordinated combination of: (1) a reduced deformable body model with nonlinear finite elements, (2) a linear-time algorithm for skeleton dynamics, and (3) explicit integration can boost simulation speed to orders of magnitude faster than existing methods, while preserving modeling accuracy as much as possible. Parallel computation on the GPU has also been implemented to obtain an additional speedup for complicated characters. Detailed discussions of our engineering decisions for speed and accuracy of the simulation system are presented in the article. We tested our approach with a variety of skeleton-driven deformable body characters, and the tested characters were simulated in real time or near real time.

Categories and Subject Descriptors: I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*Physically based modeling*; I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*Animation*

General Terms: Algorithms

Additional Key Words and Phrases: Physics simulation, deformable body, skeleton, finite element method, mesh embedding, hybrid dynamics, parallel computing, physically-based simulation, skeleton-driven deformable body

ACM Reference Format:

Kim, J. and Pollard, N. S. 2011. Fast simulation of skeleton-driven deformable body characters. ACM Trans. Graph. 30, 5, Article 121 (October 2011), 19 pages.
DOI = 10.1145/2019627.2019640
http://doi.acm.org/10.1145/2019627.2019640

This research was partially supported by NSF award CCF-0702443.

Authors' addresses: J. Kim (corresponding author), N. S. Pollard, Department of Computer Science, Carnegie Mellon University, Robotics Institute, 5000 Forbes Avenue, Pittsburgh, PA; email: junggon@cs.cmu.edu.

DOI 10.1145/2019627.2019640
http://doi.acm.org/10.1145/2019627.2019640

1. INTRODUCTION

Many researchers are interested in fast simulation of elastically deformable bodies and many approximate techniques have been proposed, along with fixes to those techniques. However, it is not clear how to make the best choices to have a fast simulation that is also as accurate and realistic as possible. In addition, although skeleton-driven deformable bodies are very important, they have been little studied in previous literature. This article looks at skeleton-driven deformable bodies, discusses design choices for mathematical modeling, and details a complete implementation of a simulation system.

Producing character motion through physics simulation usually takes a long time not only because the simulation itself is computationally expensive, but also because many trials must be run with different settings of simulation parameters to reach a desired result. Therefore, in order to be most useful to animators, a simulation system should be fast enough to run at interactive rates. Speed, however, is not the only virtue required of a simulation algorithm. The accuracy of the simulation is also very important, because obtaining better, or realistic, animation is the very reason for using an expensive physics-based technique. Because speed and accuracy usually cannot be pursued at the same time, most existing techniques find their own points of compromise between the two ultimate goals. This article considers in detail which design tradeoffs should be made when real-time or near real-time simulation of characters such as that shown in Figure 1 is a requirement.

We focus on physics-based simulation for elastically deformable body characters which have their own skeletons inside the soft bodies, and the deformable bodies are assumed to be driven by their skeletons. The types of simulation may be classified into two groups: the so-called one-way and two-way simulations. In *one-way* simulation, the skeleton is driven kinematically so that the global skeletal motion is not affected by the secondary motions of the passive deformable bodies and the environment. Such one-way simulation systems have been implemented in many commercial animation tools for postprocessing the behavior of passive elements such as hair and clothing. In *two-way* simulation, on the contrary, the character is usually actuated by internal forces only; in this case, forces or torques applied at the joints of the skeleton. The two physical systems, the skeleton and the deformable body, interact with each other so that the global motion of the skeleton is affected by the secondary motions of the soft body as well as the internal actuation. The character and the environment can also be affected by each other through contact and collisions, and this often leads to complicated behaviors, especially when the environment is changeable, for example, by having movable obstacles as shown in Figure 2.

In this article we present a fast physics simulation system for skeleton-driven deformable body characters. Our system can handle both one-way and two-way simulations for skeleton-driven deformable body characters in a unified framework. Moreover, the simulation speed is fast enough to be applied to an interactive character animation system. For example, the jiggling in the deformable

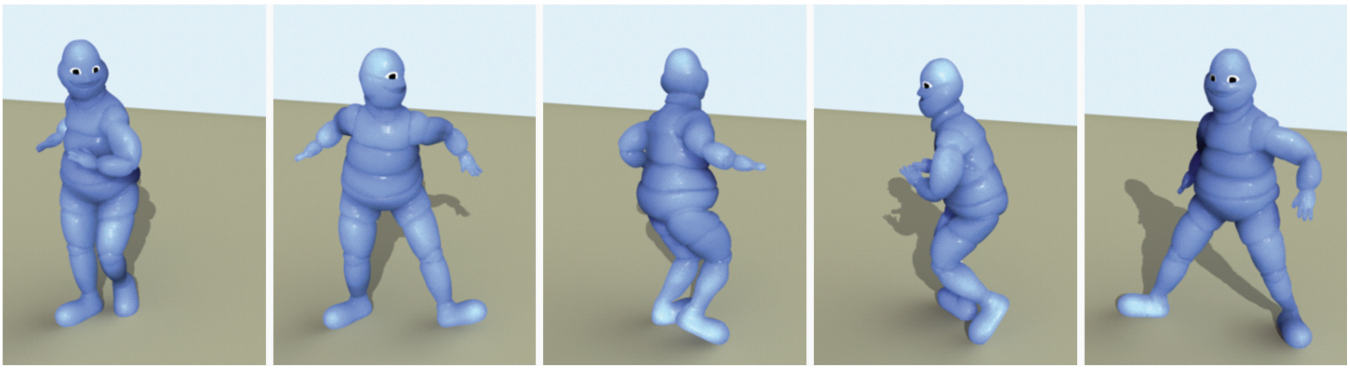

Fig. 1. A skeleton-driven deformable body character (Fatman): A realistic jiggling behavior of the dancing character's deformable body can be captured in near real time with our physically-based simulation technique. See Figure 15 and Table IV for its simulation model and computation time.

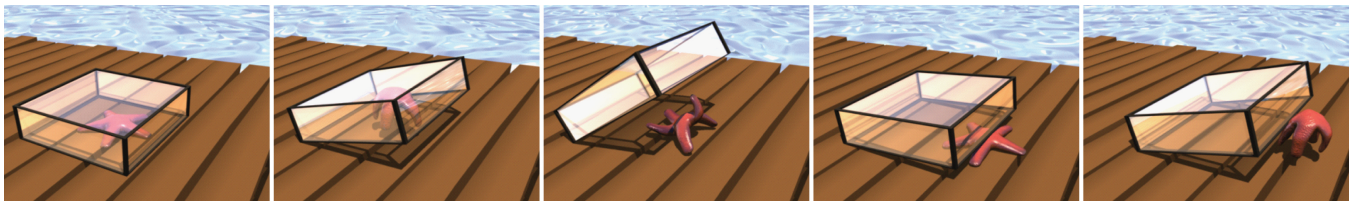

Fig. 2. A starfish escaping from a glass cage: A starfish has been trapped in a glass cage and is trying to escape by attempting a few trials, for example, hitting the obstacle with a jump (the second frame) or lifting up the obstacle's edge using its arm (the third frame). Our two-way simulation handles such complicated interaction between the character and the changeable environment automatically, and results in a physically plausible animation.

body of the complex skeleton-driven character, Fatman, shown in Figure 1 can be generated in almost real time in our system (Table IV). Our system for skeleton-driven deformable body characters has the following features, and to our knowledge, no previous work shows all of them (see Figure 3).

(1) Speed. The simulation speed is fast enough to be applied to an interactive animation control system while preserving modeling accuracy as much as possible.
(2) Unity. Our system handles both the one-way and two-way simulations in a unified formulation.
(3) Scalability. The cost of all runtime computations for solving the equations of motion for a single simulation time-step is linear in the model complexity, which is desirable for good scalability to complicated characters.

We investigate existing techniques, review their strengths and weaknesses, and take some of their advantages while addressing their drawbacks by modifying them or by hiring other techniques. More specifically, a nonlinear finite element method is chosen to handle largely deformed elements effectively, a reduced system for deformable bodies obtained by applying mesh embedding and mass lumping is used to speed up the simulation, and a linear-time algorithm is used to solve the fully nonlinear dynamics of the skeletons. We present detailed discussions on our engineering decisions in putting such technical pieces together to achieve a fast simulation speed while preserving modeling accuracy as much as possible.

After reviewing related previous work in Section 2, we will explain our choices in building an approximate mathematical model for a skeleton-driven deformable body system and solving the equations of motion of the dynamical system efficiently in Sections 3, 4, and 5. Parallelizing the computation can lead to additional speedup, and this will be addressed in Section 6. Finally, we will show the results in our experiments in Section 7, and conclude this article in Section 8.

2. RELATED WORK

Generating realistic behaviors of deformable bodies has been an active research topic in computer graphics. After the pioneering work by Terzopoulos et al. [1987], physics-based methods have been applied successfully to the simulation of various phenomena in deformable objects such as cloth [Baraff and Witkin 1998; Bridson et al. 2002; Choi and Ko 2002], elasticity [O'Brien and Hodgins 1999; Müller et al. 2002], and plasticity [O'Brien et al. 2002; Bargteil et al. 2007]. An excellent survey on the methods for deformable bodies in computer graphics can be seen in Nealen et al. [2006], and here we briefly summarize the previous work most closely related to ours, that is, the techniques for physically-based simulation of soft elastic bodies with skeletons.

Shinar et al. [2008] presented a time integration scheme for solving dynamic elastic deformations in soft bodies interacting with rigid bodies. Their framework can handle an articulated rigid body skeleton embedded in a soft body using prestabilization and post-stabilization to enforce joint constraints, and capture the two-way coupling between the skeleton and the deformable body. However, their method does not facilitate development of an interactive animation system because of the massive computation required for the finite elements representing the deformable body. In their experiment, for example, it took about 30 min for a 1s simulation of a flopping fish motion. In our approach we obtain a similar two-way coupling of dynamic motions in real time with a reduced model while preserving modeling accuracy as much as possible.

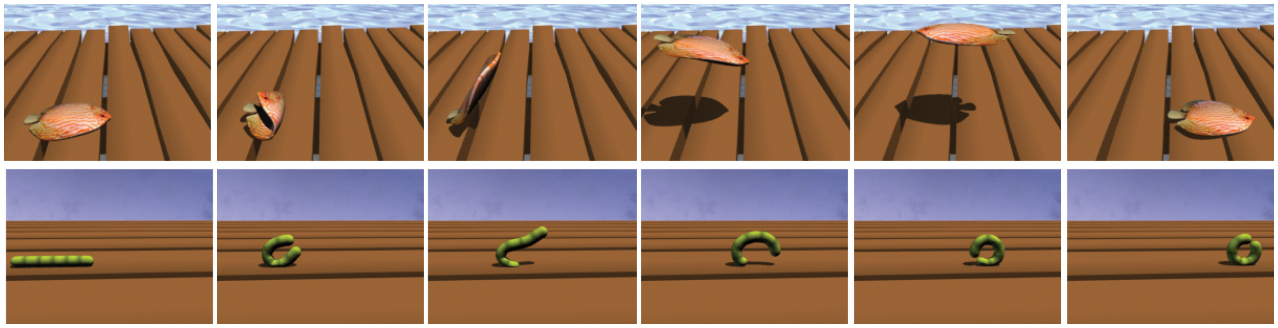

Fig. 3. Simulated motions of self-propelled characters: The fish and worm characters have deformable bodies attached to their skeletons, and the jumping and rolling motions were actuated only by internal motors (i.e., the active joints in the skeleton). The only external forces are those due to gravity and contact forces computed using a penalty-based contact mechanism. This physically plausible model of forces was used for the two-way simulation in order to mimic lifelike self-propelled motions.

Focusing on the surface rather than the volume is one possible approach to a fast solution for physically-based deformations of soft bodies [Turner and Thalmann 1993; Bro-nielsen and Cotin 1996; James and Pai 1999; Shi et al. 2008]. Galoppo et al. [2006] presented a fast method to capture dynamic deformations on the surface of a soft body including a rigid core, and they extended their method to apply to soft body characters with multiple rigid bones in Galoppo et al. [2007]. Their formulation, however, only considers the elastic energy from skin-layer deformation and does not include the deformation inside the volume, so it does not capture pose-dependent deformations correctly.

Another possible approach to fast, physically-based volumetric deformations in the soft body is using a quasistatic approximation. Teran et al. [2005] presented a quasistatic solution for flesh deformation driven by a skeleton. Lee et al. [2009] used a similar method to compute the deformation of the soft tissue in their biomechanical model of the human upper body. Though such quasistatic solutions are much faster than a fully dynamic simulation with the same model size, they do not capture the dynamic behaviors of the soft body such as jiggling.

Reducing the mathematical model size would be a practical choice to speed the full dynamics simulation to an interactive rate. Mesh embedding, which is also called free-form deformation [Sederberg and Parry 1986], uses a low-dimensional coarse volumetric mesh to represent the behavior of a deformable body. The embedding mesh representation has been successfully applied to physics-based simulations of deformable objects such as elastic soft bodies [Faloutsos et al. 1997, Capell et al. 2002b, Nesme et al. 2006, 2009; Kharevych et al. 2009] and viscoplastic material flow with thin features [Wojtan and Turk 2008]. Particularly, Capell et al. [2002a] employed a skeleton to control the coarse mesh enclosing a soft elastic body and they extended their method to include rigging forces which guide the deformation to a desired shape [Capell et al. 2005]. They handled the skeletal constraints as a set of prescribed trajectories for the nodes located on the bones, so that the trajectories must be given before simulation. Though their method can effectively handle one-way simulation with a given global skeletal motion, it does not consider two-way coupling among the skeleton, the soft body, and the environment which is needed to generate physically plausible motions of lifelike characters. Our approach can handle two-way interactions in the simulation to create realistic motions of self-propelled characters like the starfish shown in Figure 2, in addition to one-way simulation. In this animation, two-way coupling gives us, for example, physically plausible ballistic trajectories, and realistic vibration of the entire character body, including the skeleton, when the starfish drops onto the pier, while handling complicated interaction between the character and the movable obstacle.

Using a linear subspace spanned by a small number of basis vectors to represent the global deformation of a deformable body is one of the most popular techniques for reducing the dimension of complexity in a huge physics model. Such a technique, which is also called modal analysis, has been successfully applied to the analysis of mechanical structures in engineering, because usually very small deformation occurs in such systems. After the pioneering work by Pentland and Williams [1989], modal analysis has also been studied by many researchers in the graphics community. James and Pai [2002] used precomputed modal vibration models excited by rigid body motions to produce secondary motions of the soft tissues attached to the character bones in real time. More recently, many researchers have tried to expand the applications of modal analysis by effectively handling nonlinear deformation and geometric constraints. For example, Hauser et al. [2003] presented a method for real-time manipulation of positional constraints, Choi and Ko [2005] presented a modal warping technique which handles large rotational displacements and orientational constraints, and Barbič and James [2005] used additional basis vectors obtained from either modal derivatives or user sketches to capture the nonlinear deformation effectively. Kim and James [2009] introduced an online model reduction technique, which incrementally builds a reduced model as the simulation progresses, to skip the computationally expensive precomputation stage and not to be restricted to any initial basis. Though exploiting the modal technique is complementary to the approach described in this article and may potentially give our simulation system further speedup, it is still inherently difficult to handle the nonlinear kinematic constraints caused by a complex internal skeleton within the linear subspace framework.

In this article, we reduce the mathematical model size through the use of mesh embedding, but in a form that allows for two-way coupling. We discuss the implications of using nonlinear finite elements in such an implementation, and present a novel technique to speed up treatment of largely deformed elements. We discuss the advantages of the lumped mass model and detail proper treatment of mechanical properties near the boundary of the body in order to preserve modeling accuracy as much as possible even in such a reduced system. We describe how to treat skeletal dynamics so that the entire system scales in a linear way with character complexity. Finally, we discuss issues that arise in parallel implementation of

such a system. Throughout the article, we explain our engineering decisions and compare to alternatives.

3. NONLINEAR FINITE ELEMENTS

The first decision that must be made is to choose how to model the deformable body for simulation. In this section we argue for the choice of nonlinear finite elements, discuss the importance of handling largely deformed elements properly, and present a novel algorithm for fast detection of largely deformed elements. For additional background information to supplement this section, please see the textbooks Bonet and Wood [1997] and Basar and Weichert [2000], as well as the overview by Nealen et al. [2006].

3.1 Nonlinear Deformation

Strain is a geometric measure of deformation such as stretching, compression, and shearing in a deformable body. Most previous work pursuing fast simulation relies on a linearized strain, or infinitesimal strain under the assumption of small deformation, because of its simplicity in computation. The linearized strain, however, can cause serious problems such as inflation of the body especially when the deformation contains rotational modes as illustrated in Müller and Gross [2004]. This is because the linearized strain cannot cleanly filter out the rigid rotational modes. Many techniques, called corotational methods, have been suggested to remove as much of the rigid rotation as possible by using local coordinate frames following the global motion of the body [Terzopoulos and Witkin 1988; Capell et al. 2002a; Müller et al. 2002; Müller and Gross 2004]. Though corotated linearized strain has been widely used in interactive graphics applications, it is still valid only when the deformation is very small.

Skeleton-driven deformable bodies, however, are likely to undergo large deformation, especially near the skeletal joints (e.g., see Figure 4). When deformations are large, a linearized method can generate highly unrealistic deformations as pointed out in Capell et al. [2005]. In order to handle large deformation in the elements effectively, we use the nonlinear Green-Lagrangian strain tensor because it can express large deformations correctly regardless of the rigid modes. From the viewpoint of simulation speed, however, using the nonlinear strain could be risky because of the expensive computation required. We overcome this problem by reducing the number of mesh elements with a coarse mesh (Section 4) and by parallelizing the computation (Section 6). This combination of decisions makes it possible to have convincing large deformations near the joints at real-time or near real-time simulation speeds.

Nonlinear Strain Details. The nonlinear Green-Lagrangian strain is defined as

$$E = \frac{1}{2}(F^T F - I), \tag{1}$$

where I is the identity tensor and F is the deformation gradient tensor $F = \frac{\partial x}{\partial X}$ where x and X denote the positions of a point in the material after and before deformation, respectively.

In a finite element method, the strain E for each element is computed from the nodal positions in the currently deformed state and the initially undeformed state. In the case of a tetrahedral mesh with a linear shape function, E becomes a constant 3×3 matrix for each element. One can see, for example, O'Brien and Hodgins [1999], for an explicit formula in such a case.

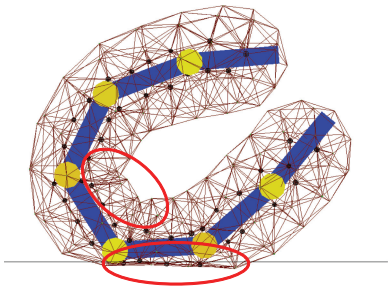

Fig. 4. A largely deformed mesh: Large deformations may easily occur during the simulation of skeleton-driven deformable body systems. This figure shows an example of such deformations in the worm mesh. The red ovals indicate highly deformed regions due to the underlying skeleton motion and the contacts between the character and the ground (black horizontal line). Handling such large deformations effectively is important to obtain a pleasing result and to achieve a stable simulation system. The reddish grid lines represent the volumetric finite element mesh. This mesh is attached to the skeleton (blue links and yellow joints) by fixing some of the the mesh nodes (black spheres) to the links. See Section 5 for detail on our skeleton-driven deformable body model.

3.2 Elastic Forces

To compute elastic forces from deformation, or strain, we need to know how internal stress is distributed across the entire soft body. Because the internal stress depends on both the amount of deformation and the material's mechanical properties, we need a mathematical material model, which is also called as a constitutive equation, to represent the relationship. Once a proper material model is chosen, we can compute the stress from the strain already obtained before.

St. Venant-Kirchhoff (or StVK) material is one of the most popular models for the purpose of computer animation because it is simple to compute and applicable to geometric nonlinear analysis. The material is defined by

$$S = \lambda(\mathrm{tr}E)I + 2\mu E, \tag{2}$$

where S is the second Piola stress tensor, and λ and μ are Lamé parameters which determine the material's elastic properties. Though our simulation system does not require the use of a particular material model, we chose the StVK model for our examples because of its simple implementation. The StVK model, however, has a fatal drawback when the element undergoes large compression. During large compression the stiffness of the material gets weaker as compression increases so that the internal stress finally becomes zero when the element is flat. Moreover, if the element gets inverted, the stress acts in the opposite direction so that the inverted element cannot be naturally restored after the inversion [Irving et al. 2004]. To handle the problem, Irving et al. [2004] suggested a correction mechanism for largely deformed elements, which will be briefly reviewed in Section 3.3 along with our suggested improvement for fast computation.

The elastic forces acting on the element nodes can be obtained from the second Piola stress tensor S in (2). We follow the formula in Teran et al. [2003] because the formula fits well with the diagonalization technique of Irving et al. [2004]. The elastic forces are obtained by

$$f_i = Pb_i, \tag{3}$$

where $P = FS$ denotes the first Piola-Kirchhoff stress tensor and $b_i = -\frac{1}{3}(A_1N_1 + A_2N_2 + A_3N_3)$ where A_j are the areas of the undeformed faces incident to the vertex and N_j denotes the normal vectors to those undeformed faces. Because A_j and N_j do not change during simulation, b_i can be precomputed. One can also save multiplications by using the relationship $f_0 = -(f_1 + f_2 + f_3)$. The total elastic force acting on a node is obtained by summing the forces exerted by all elements incident to the node. Note that (3) is valid only when the mesh element is fully filled with elastic material. We will discuss how to handle partially filled elements, which are commonly found when modeling with mesh embedding, in Section 4.3. Treating partially filled elements properly is critical for achieving accuracy in modeling and for improving stability of the simulation, but to our knowledge is not addressed in prior publications.

3.3 Largely Deformed Elements

As mentioned earlier, the StVK material becomes softer as the material gets compressed and this can lead elements to become inverted quite easily. More seriously, the material is getting rapidly stiffer as it gets stretched and this may cause the simulation to diverge. In fact, many other mathematical material models have similar problems. For example, Neo-Hookean material has behavior opposite to StVK material in large compression, becoming rapidly stiffened as it is compressed.

To overcome such a problem Irving et al. [2004] presented a technique for modifying the material model in the range of large deformation. Suppose that the deformation gradient F can be decomposed as

$$F = U\hat{F}V^T, \tag{4}$$

where $\hat{F}$ is a diagonal matrix, and U and V are pure rotations, that is, $U^TU = V^TV = I$ and $\det U = \det V = 1$. Note that such a decomposition is slightly different from the traditional singular value decomposition in the sense that U and V must be pure rotation matrices and the diagonal entries in $\hat{F}$ can be negative. The diagonal entries of $\hat{F}$, which are called the principal stretches, represent how much the element has been stretched or compressed along principal axes. Therefore, through the diagonalization of the deformation gradient, we can determine if an element undergoes large deformation or not. For example, if one of the diagonal entries is negative, it means that the element has been inverted along the corresponding principal axis. If the element has been stretched or compressed too much in a certain principal direction, a different material model such as a linear model is applied in that direction to obtain reasonable stress and stiffness. In our implementation the StVK model is replaced with rotated linear models when the deformation exceeds given lower or upper limits as shown in Figure 5. Once the stresses along the principal axes are determined from the principal stretches, the first Piola-Kirchhoff stress tensor can be obtained as

$$P = U\hat{P}V^T, \tag{5}$$

where $\hat{P} = P(\hat{F})$ is a diagonal matrix whose diagonal entries are the stresses along the principal axes, and U and V have been obtained in (4).

In order to decompose the deformation gradient, we need to perform singular value decomposition on F (or solve the eigenproblem $F^TFv = \lambda v$) and postprocess to ensure U and V are pure rotations [Irving et al. 2004]. Though F is just a 3×3 matrix, executing the process for every element is computationally expensive. We suggest an efficient way to address this problem by diagonalizing the deformation selectively for largely deformed elements only.

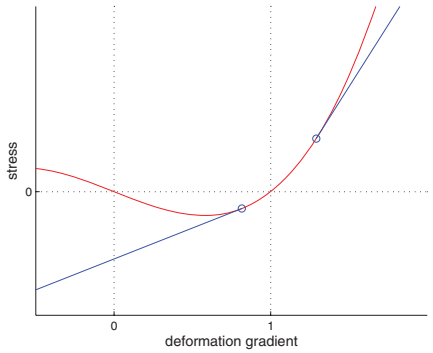

Fig. 5. The relationship between the deformation gradient ($\hat{F}$) and the first Piola-Kirchhoff stress ($\hat{P}$) along a principal axis: (Red) St. Venant-Kirchhoff model (blue) Rotated linear models with the same stiffness as the StVK model at the boundaries (blue circles). Note that the element is undeformed, compressed, and stretched when $\hat{F} = 1$, $\hat{F} < 1$ and $\hat{F} > 1$, respectively.

Selective Diagonalization

The key for selective diagonalization is to sort out largely deformed elements in a short time. To see whether the element deformation exceeds a given range of small deformation, checking the range of the principal stretches, rather than obtaining the exact values of them, is enough. If $\det F \leq 0$ for an element, this means that the element has been inverted so that we need the diagonalization process on that element. On the other hand, if $\det F > 0$, we need an additional step to decide whether the element undergoes large deformation. Let $[\delta_L, \delta_U]$ be a given small deformation range where we do not need the diagonalization process. First note that the solutions of the characteristic equation, $\det(\lambda I - F^TF) = 0$, correspond to the square of the principal stretches. For elements with small deformation, solutions to this equation must lie between δ_L^2 and δ_U^2. Now, observe that the left-hand side of this equation is a cubic polynomial in λ

$$f(\lambda) = \lambda^3 + a\lambda^2 + b\lambda + c, \tag{6}$$

where a, b, and c are the coefficients of this polynomial. This cubic polynomial will have three zeros, which all must lie between δ_L^2 and δ_U^2 for the element to have small deformation. Finally, consider the diagram in Figure 6. We can conclude that the element has a small deformation, that is, that all of the solutions are in the range $\left[\delta_L^2, \delta_U^2\right]$ by satisfying

$$f\left(\delta_L^2\right) < 0, \quad f\left(\delta_U^2\right) > 0, \quad \delta_L^2 < \alpha, \beta < \delta_U^2, \tag{7}$$

where α and β are the solutions of $f'(\lambda) = 0$. If some of (7) are not satisfied for an element, this means that the element has been largely deformed and we need the diagonalization process for the element. With given F and $C = F^TF$, the check process requires up to 37 multiplications per element in our implementation.

For every element we execute the determination process, and then we follow the diagonalization technique by Irving et al. [2004] for the largely deformed elements only. In the worst case, that is, when every element is largely deformed, our approach could be

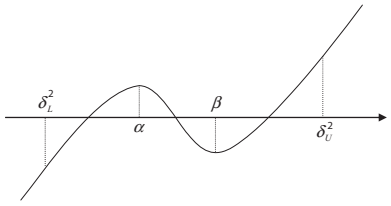

Fig. 6. A cubic polynomial: If a cubic polynomial, $f(\lambda) = \lambda^3 + a\lambda^2 + b\lambda + c$, satisfies (7), the zeros of the polynomial must be located between δ_L^2 and δ_U^2 where $0 < \delta_L^2 < \delta_U^2$.

inefficient for its added determination process. However, in many character motions only some part of the deformable body undergoes large deformation.

The speedup by the selective diagonalization technique varies depending on many factors such as the type of motions to be created, the number of the finite elements, and the choice of SVD algorithm for the diagonalization. It is clear that the technique can be more effective when the number of finite elements is large so that most of the computation time is spent in computing the elastic forces inside the deformable bodies. Indeed, this technique results in a speedup of a factor of 3 or more in the time required for the entire physics simulation in some of our tests (Table V). Even for our low-dimensional models, we found that the technique still works to some extent (speedup from 10% to 70% depending on the simulation setup). The effect of the technique will be discussed in detail in Section 7.

4. REDUCED SYSTEM

In this section we explain our reduced model for deformable bodies. We use a mesh embedding technique to reduce the complexity of the mathematical model so that we can perform a fast low-resolution volumetric simulation. We also consider a high-resolution character surface in the formulation to handle the contact between the character and the environment, and to capture correct mechanical properties of the deformable body such as mass and elastic forces.

4.1 Mesh Embedding

In order to capture the detailed physics of the deformable body with a finite element method, a fine mesh structure fitting to the body's volumetric geometry can be used as in Shinar et al. [2008]. However, though the fine mesh gives high accuracy to the simulation, the computation is too expensive to be applied to a fast simulation system. Therefore we need to reduce the complexity of the mathematical model to speed up the simulation.

Most techniques to reduce the system's DOF within a finite element framework can be roughly classified into two groups: modal reduction and mesh embedding. *Modal reduction* is a very popular method to reduce the complexity of a finite element system. It uses a linear subspace spanned by a small number of displacement basis vectors to represent the deformation in the body. The eigenmodes obtained from linear modal analysis would be the best basis vectors for small deformation. For large deformation, however, they are not sufficient to capture the nonlinear deformation, so many techniques have been suggested to choose a good deformation basis set. See, for example, Barbič and James [2005] for their choice of the basis vectors based on either modal derivatives or user sketches. Though modal techniques have been widely used in many real-time applications handling soft bodies such as surgery simulators, it is still an open problem to handle the nonlinear constraints caused by a skeleton in the soft body within the linear framework.

Mesh embedding, which is also called free-form deformation in the literature, uses a relatively low-dimensional coarse volumetric mesh enclosing the entire deformable body in order to represent the behavior of the body. The location of every material point inside the deformable body is determined by interpolating the positions of the neighboring nodes in the mesh. Since the work by Faloutsos et al.[1997], the mesh embedding technique has been widely used to simulate soft bodies in graphics literature [Capell et al. 2002a; Kharevych et al. 2009; Nesme et al. 2009].

We chose the mesh embedding to reduce the DOF of the deformable bodies in our simulation system not only because the technique can reduce the system DOF without losing the fine geometry of the characters but also because the internal skeleton can be handled more easily in the embedding mesh system compared to the modal reduction. Capell et al.[2002a] presented an embedding mesh framework to control soft body characters which have an underlying skeleton made up of mesh nodes with known global trajectories. In our formulation, however, the skeleton is considered as an articulated rigid body system and the dynamics of the skeletal system is fully considered in solving the equations of motion of the whole system. Therefore, our method can simulate not only the deformable body motions that result from a kinematically moving skeleton, but also the motions of lifelike characters that are actuated with internal motors in the skeleton. The complete system consisting of a deformable body and a skeleton will be addressed later in Section 5.

The position of a material point in the deformable body is determined from the nodal positions of the coarse mesh through interpolation. We assume a linear relationship between the body point and the nodes defined as

$$y = \sum_i \phi_i \, x_i, \tag{8}$$

where $y \in \Re^3$ denotes the position of a body point, $x_i \in \Re^3$ represents the i-th nodal position, and $\phi_i = \phi_i(y) \in \Re$ is a coefficient which is associated to the i-th node at the point y and satisfies $\sum_i \phi_i(y) = 1$. Only neighboring nodes of the body point have nonzero coefficients, and in the case of tetrahedral mesh, we choose the four nodes of the tetrahedral element enclosing the material point as the neighbors. In this case, $(\phi_1(y), \ldots, \phi_4(y))$ is the barycentric coordinates of the point y with respect to the nodal position $(x_1, \ldots, x_4)$.

4.2 Mass Lumping

The mass matrix for the deformable body can be obtained from its kinetic energy formula

$$T = \frac{1}{2} \int_{\mathrm{V}} \rho \; \dot{y}^T \dot{y} \; d\mathrm{V}, \tag{9}$$

where ρ denotes the material density at a body point y, and $\dot{y}$ is the velocity of the point. By substituting (8) to (9), one can get $T = \frac{1}{2} \dot{x}^T M \dot{x}$ where M denotes the mass matrix of the deformable body and x is a concatenated position vector of the coarse mesh nodes. The mass matrix can be obtained by $m_{ij} = \int_{\mathrm{V}} \rho \; \phi_i \; \phi_j \; d\mathrm{V}$ where $m_{ij} \cdot \mathrm{I}$ denotes the 3 by 3 matrix component of M representing the mass coupling between the i-th and j-th mesh nodes, and ϕ_i and ϕ_j are the coefficients associated with nodes i and j as defined

in (8). Though the mass matrix for the deformable body is sparse, the cost for solving the equations of motion of the whole system, including the deformable body and the skeleton, increases rapidly as the number of mesh nodes increases. This is not only because the mass matrix for the skeleton is always dense and dependent on the pose but also because mass coupling between the soft body and the skeleton also occurs when the soft body is attached to the skeleton by fixing some nodes to the bones as illustrated in Figure 9. No efficient linear-time solution for such a system is yet known.[1]

Mass lumping is a common way to simplify the dynamics equations and speed up the computation by approximating the mass matrix with a diagonal matrix. Using mass lumping for the soft body allows a linear-time solution to the dynamics equations of the entire system and this will be described in Section 5. Through mass lumping, the continuously distributed mass in the deformable body is modeled as a set of point masses located at the nodes. Moreover, by merging the point masses of the fixed nodes into the bones, the mass coupling between the soft body and skeleton can also be simply treated within the skeletal dynamics without any additional cost. Note that we will still fully consider the dense mass matrix of the skeleton because it can be efficiently handled by existing linear-time algorithms such as Featherstone [1983] and Baraff [1996].

The point mass of the i-th node is obtained by integrating an effective material density over the mesh volume

$$m_i = \int_{\mathrm{V}} \rho\ \phi_i\ d\mathrm{V}, \tag{10}$$

and it is same as the sum of the i-th row, or column, of the full mass matrix, that is, $m_i = \sum_j m_{ij}$.[2] In the case of tetrahedral mesh, the mass contribution of an element fully filled with a homogeneous material to its four nodes is simply $\rho\ \mathrm{v}/4$ where v is the undeformed volume of the element [O'Brien and Hodgins 1999]. Because we use a coarse mesh enclosing the deformable body, the mesh elements near the boundary of the body may not be fully filled with a material. We obtain the mass contribution of such elements to their nodes by computing (10), which will be explained in Section 4.3. By lumping the mass of the soft body, the whole system including the deformable body and its skeleton can be treated as a mechanical system consisting of point mass particles and articulated rigid bodies, which could be solved by any existing rigid-body dynamics engines. We, however, use our own implementation of a recursive dynamics algorithm in our simulation system for better performance and a more flexible simulation setup, and this will be described in Section 5.

[1]Let M_s be the mass matrix of the skeleton and $[m_{ij}] = \left[\begin{smallmatrix} M_d^{aa} & M_d^{ab} \\ M_d^{ab} & M_d^{bb} \end{smallmatrix}\right]$ be the mass matrix of the deformable body where the superscripts a and b denote the terms corresponding to the free and fixed nodes, respectively. When the fixed nodes are attached to the skeleton with hard constraints, that is, $x^b = f(q)$ where q denotes joint coordinates of the skeleton and $f(q)$ is forward kinematics to the nodes, the mass matrix of the whole system can be written as $\left[\begin{smallmatrix} M_d^{aa} & M_d^{ab}J \\ J^T M_d^{ab} & J^T M_d^{bb} J + M_s \end{smallmatrix}\right]$ where $J = \frac{\partial f}{\partial q}$. Because the mass matrix is not constant (M_s and J are dependent on the skeleton pose), no efficient linear-time solution exists yet.

[2]The lumped mass formulation can also be obtained from momentum as in Sifakis et al. [2007]. Let P be the momentum of the deformable body and m_i be an effective lumped mass at the i-th node. Then, $P = \int_{\mathrm{V}} \rho\ \dot{y}\ d\mathrm{V} = \sum_i m_i \dot{x}_i$, and substituting (8) for $\dot{y}$ and differentiating both sides of the equation with respect to $\dot{x}_i$ will lead to (10).

4.3 Mechanical Properties near Boundary

Usually the coarse mesh elements near the fine surface of the soft body are partially filled with a material. In most previous work using mesh embedding, however, such elements are assumed fully filled for simplicity at the cost of decreased accuracy. With the assumption of fully filled elements, the lumped mass can be easily computed from the undeformed volume of the elements, and the elastic force can be obtained as explained in Section 3. Though such an approach could be thought as a practical choice for graphics applications that are generous with regard to the simulation accuracy, sometimes it could lead to visible artifacts such as overly stiff behaviors of soft thin bodies surrounded by coarse control lattices as pointed out in Nesme et al. [2009].

Capturing the mechanical properties of a soft body as correctly as possible in a coarse embedding mesh system has been explored recently. Nesme et al. [2006] suggested spatial averaging of mass and stiffness for coarse linear finite element models. They recently extended it to consider material inhomogeneity through obtaining a displacement map between coarse nodes and fine-level nodes by solving a static equilibrium equation on the fine-level model, and finally obtaining the stiffness matrices of the coarse elements [Nesme et al. 2009]. Partially filled coarse elements are handled in their method by introducing hard constraints rigidly linking the disconnected coarse nodes to their neighboring fine nodes when solving the equilibrium equation. Kharevych et al. [2009] also presented a similar method for capturing material inhomogeneity within a coarse mesh system, but in a different way, by finding an effective linear elasticity tensor per each coarse element and a displacement mapping between coarse nodes and fine nodes. However, because the previous methods are based on a linear finite element framework, they do not fit well in our simulation system, which uses nonlinear finite elements. Wojtan and Turk [2008] presented a method for capturing the correct mass of thin homogeneous materials within a time-varying coarse mesh system, but they did not discuss elasticity in the partially filled elements.

We have already discussed how to obtain the mass matrix of the deformable body from its kinetic energy and how to effectively approximate the mass property with a set of lumped nodal masses in Section 4.2. To evaluate the volume integration in (10) numerically, we discretize each element volume near the body surface by randomly sampling points. We follow Rocchini and Cignoni [2000] for generating random points in tetrahedral elements. For each sample point, we check whether it is located inside of the body or not, and if it is, the mass contributions of that point, $\rho\phi_i \mathrm{v}/n_s$, are added to the neighboring nodes where ϕ_i denotes the nodal coordinates of the sample point, v is the volume of the element, and n_s is the number of sample points used for the element. We also refer the reader to Rathod et al. [2005] for Gauss Legendre quadrature formulas for computing numerical integration over a tetrahedron.

Obtaining the particle masses correctly is not sufficient for both accuracy and stability of the simulation. In fact, without considering partially filled elements in calculating the elastic forces, the correct mass computation could do more harm than good, especially with regards to stability. Indeed, as shown in Figure 7 (right), the outermost nodes may have very small masses so that the elastic force computed from an assumption of fully filled elements, which is much larger than the force that should be exerted on the node, could lead to an unstable simulation. For example, a Fatman model with only the mass treatment required a simulation step size nearly 10 times smaller for the simulation of the dancing motion shown in Figure 1.

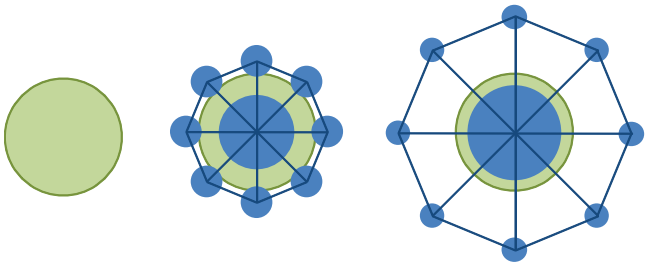

Fig. 7. Mesh embedding and mass lumping: This 2D illustration shows two different examples (middle, right) of coarse meshes enclosing a deformable body (left) depicted as the solid green circle. The meshes are depicted as the blue lines, and their corresponding lumped mass particles located at the nodes are shown as the solid blue circles. The area of the circles represents the mass of the particles and the deformable body, and the lumped masses were computed by (10). Note that the sum of the lumped masses must be equal to the mass of the deformable body.

Computing elastic forces while taking care of partially filled nonlinear elements is surprisingly simple, though, as far as we know, it has not been clearly discussed in the literature. We assume that each element has constant material properties such as Lamé parameters for StVK material. The nonlinear strain tensor is constant for each element as mentioned in Section 3.1, and the stress tensor is also constant due to the constant material properties. Therefore, the elastic potential for each element is proportional to the material volume of the element. Because the elastic force is defined as the negative partial derivative of the potential with respect to nodal position, the forces acting on the nodes due to the partially filled element can be obtained by

$$\tilde{f}_i = w f_i, \tag{11}$$

where f_i is the elastic force computed by (3) with the assumption of a fully filled element, and w denotes the ratio of the filled region in the element to its volume. The volume ratio for each element can be obtained as $w = n_s^*/n_s$ where n_s^* is the number of sample points located inside the soft body and it can be counted during the nodal mass calculation described before. Note that the nodal masses and the volume ratio for each element can be precomputed because they do not vary during simulation. We show the effectiveness of our treatment of partially filled elements in Figure 8 where different soft body models with the same volumetric mesh are compared. Without the treatment, all three models in the figure would have had the same deformation which is not physically plausible.

5. SKELETON-DRIVEN DEFORMABLE BODY

So far we have chosen a coarse mesh structure with lumped mass particles to reduce the model size. We also selected nonlinear finite elements with the selective diagonalization technique for better handling of large deformation, and considered the fine geometry of the body in calculating the particle masses and elastic forces for better modeling accuracy. In this section, we complete our modeling and mathematical formulations for skeleton-driven deformable body systems.

The skeleton consists of arbitrarily shaped rigid bones connected with joints, and one of the bones is virtually connected to the ground with a joint which is called the *root joint*. We assume that the skeleton has a tree topology, and there is no other restriction in

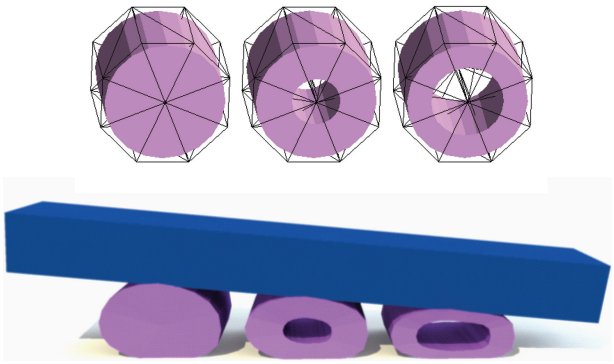

Fig. 8. Effect of the treatment of partially filled elements on the simulation: All three models have the same volumetric mesh (upper), but they deform differently under the heavy load pressure due to our treatment of partially filled elements (lower).

modeling of the articulated rigid body system such as the mass properties of the bones and the types of joints. In this section we will explain how to attach the deformable body to the skeleton, consider damping and contact, and efficiently solve the equations of motion of the entire system.

Fixed Nodes

One possible way to attach the deformable body to the skeleton would be to use soft positional constraints as done by Lee et al. [2009] in their quasistatic soft tissue simulation. Zero-length springs are used to connect points in the soft body to their target position on the bones. Such an approach allows flexibility in choosing the attachment points, but the stiff constraint springs could cause a severe time step restriction in the simulation. Though implicit integration could alleviate the small step size problem, it requires building and solving a huge matrix system which is neither positive-definite nor symmetric, and this decreases the overall simulation speed.

Instead, we attach the deformable body to the skeleton by directly fixing some of the coarse mesh nodes to the bones (Figure 9). The simplest way to select the nodes to be fixed would be to choose nodes located near the bones. However, if there are specific points in the soft body to be fixed to bones, we must generate the embedding coarse mesh in such a way that its nodes are placed on those locations. In such a case, we first set the positions of nodes and then generate mesh elements using a 3D Delaunay triangulation. Because the Delaunay triangulation always results in a convex mesh, we usually cut out some of the outer elements which do not contain the soft body material. This cutting-out process, however, is not really necessary because the empty elements will be automatically excluded from the simulation by (11). The mass of each fixed node is merged into the bone to which the node is attached, in order to consider the mass coupling between the deformable body and the skeleton in the skeleton dynamics. All the forces acting on the fixed nodes such as the elastic forces are transmitted to the bones and they affect the solution of the skeletal dynamics. The resulting skeletal motions will also affect the secondary motions of the deformable body in the next time step through the updated fixed nodal positions.

Damping

We need damping to attenuate jiggling in the deformable bodies. The most common approach for this in the literature would

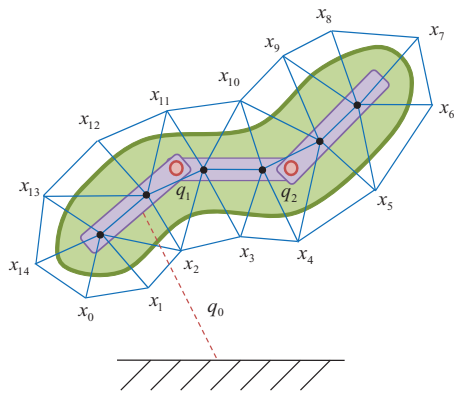

Fig. 9. A 2D illustration for skeleton-driven deformable body systems: The shape of the deformable body is defined by the fine surface, but the motions of it are represented by the coarse volumetric mesh enclosing the surface. The deformable body is attached to the skeleton by fixing some of the mesh nodes (black dots) to the bones, and overall motions are driven by the skeleton. In this illustration, $\{q_0, q_1, q_2\}$ is the joint coordinates of the skeleton where q_0 denotes the coordinates of the virtual root joint, and $\{x_0, \ldots, x_{14}\}$ represents the coordinates for the deformable body.

be to use Rayleigh damping, which assumes that the damping matrix is proportional to the mass and stiffness matrices. Applying Ralyleigh damping in our simulation system, however, is not straightforward because the stiffness matrix for nonlinear finite elements, which is not needed in our system, would have to be constructed, and this would decrease the simulation speed. Perhaps, it would be most elegant to consider the viscous term along with the elastic forces in the finite element formulation as done by O'Brien and Hodgins [1999], but this method requires to additionally compute the strain rate and the viscous stress in each element.

We want a simple and effective way of dissipating vibrations in the deformable bodies. In addition, the damping forces must be dependent on the relative velocity of the deformable body to the underlying skeleton, rather than the global velocity, so that the forces dampen the jiggling only and do not drag down the whole body motion.

With these considerations in mind, we connect a damper to each node, or mass particle, in the deformable body and attach the other end of the damper to the particle's reference bone which is usually set to be the nearest bone from the undeformed position of the node. The damping force f_d acting on the particle is obtained by

$$f_d = -c\,(\dot{x} - \dot{p}), \tag{12}$$

where c is a damping coefficient proportional to the particle's mass, $\dot{x}$ is the global velocity of the particle, and $\dot{p}$ denotes the global velocity of the point to which the damper is connected to the bone. The body velocity $\dot{p}$ is a nonlinear function of the skeletal pose and the skeletal joint velocities, that is, it can be written as $\dot{p} = J\dot{q}$ where $J = J(q)$ is a pose-dependent Jacobian matrix and $\dot{q}$ denotes the joint velocities. In our implementation, $\dot{p}$ is obtained efficiently when we update kinematic information of the skeleton which will be discussed later in this section. Our damping model fits well to our skeleton-driven deformable body systems, and as far as we know, it has not been explicitly mentioned in the literature. Note that the opposite force must also be transmitted to the reference bone because the other end of the damper is connected to the bone.

Frictional Contact

We apply penalty-based contact normal forces and Coulomb friction forces to the fine surface of the deformable body, not to the coarse mesh. Because we use the coarse mesh in the simulation, we must transform the contact forces acting on the fine surface to equivalent forces acting on the coarse mesh nodes. The force transformation can be obtained easily from the principle of virtual work as done in Faloutsos et al. [1997]. Let X and Y be the vectors representing all of the nodal positions of the coarse mesh and all of the vertex positions on the surface, respectively, and they have a linear relationship $\delta Y = J\ \delta X$ where J is a Jacobian matrix. Let F_s be the contact forces acting on the surface vertices and F_v be the nodal forces equivalent to the contact forces. From the principle of virtual work, we have $\delta Y^T F_s = \delta X^T J^T F_s = \delta X^T F_v$ and this leads to $F_v = J^T F_s$. More specifically to our formulation, because we use a linear embedding defined in (8), for each contact force acting on a surface vertex, f^s, we can obtain its equivalent nodal forces acting on its neighboring nodes with

$$f_i^v = \phi_i f^s, \tag{13}$$

where f_i^v denotes the equivalent force acting on the i-th neighbor node of the surface vertex, and ϕ_i is the corresponding coefficient defined in (8). The total forces acting on the coarse mesh nodes are obtained by summing the forces transformed from all contact forces on the surface, and the computational cost for the force transformation is linear to the number of contact points on the surface. Note that the transformation can be applied to transform any forces acting on the deformable body into equivalent nodal forces.

Though we compute contact forces with respect to the vertices on the fine surface of the soft body, penetration of the surface into an obstacle will still occur because of the underlying penalty-based contact mechanism. If nonpenetration is of utmost importance, a projected vertex approach, in which the contact vertices are projected onto the obstacle's surface and velocity impulses are applied to those vertices, could also be used within our framework. However, because we use a low-dimensional coarse mesh to control the surface, multiple contact vertices in an element could cause conflicting constraints. Applying the method to only the most deeply penetrating vertex in each element would be a practical way to avoid such a problem as is done in Wojtan and Turk [2008].

Equations of Motion

The dynamics of the free mass particles, the nodes which are not fixed to the bones, can be simply written as

$$m_i \ddot{x}_i = f_i, \tag{14}$$

where m_i is the mass of the i-th free particle, x_i denotes the global position of the i-th free particle, and f_i represents all forces acting on that particle which includes the elastic force, the damping force, the equivalent contact force, and the gravity. Once all the forces have been computed from the current state, the equation of motion for each free particle can be solved independently from the other particles and the skeleton.

The skeleton is an articulated rigid body system with a tree topology. We use the relative joint coordinate system to represent the degree of freedom of the skeleton system (Figure 9). The equations

of motion of the skeleton can be written as

$$M\ddot{q} + b = \tau, \tag{15}$$

where q denotes the joint coordinates including the root joint coordinates, τ is the torques, or forces, acting on the joints, $M = M(q)$ is the mass matrix of the skeleton, and $b = b(q, \dot{q})$ denotes all the other terms including the Coriolis and centrifugal forces, the force by gravity, and the forces transmitted from the mass particles in the deformable body. As we mentioned before, the rigid bodies, or bones, in the skeleton contain the masses of the fixed particles. The elastic forces and contact forces are directly transmitted to the bones through the fixed nodes, and the damping forces acting on the free nodes are also transmitted to the bones. We obtain the system state at the next time step by solving the equations of motion, (14) and (15), for any accelerations that are unknown and by integrating the solution. The skeleton dynamics will affect the solution of the deformable body at the next time step through the updated states of the bones, more specifically, the updated positions of the fixed nodes and the updated velocities of the points to which the dampers are connected to the bones.

Solution of the Skeleton Dynamics

The dynamics equations for the skeleton shown in (15) can be solved in various ways. We briefly discuss the following three common scenarios for animated characters.

—*Forward simulation with joint torques:* If the joint torque τ in (15) is known, we can compute the acceleration $\ddot{q}$ from the equations of motion and this is called *forward dynamics*. To simulate lifelike character motion driven by internal motor actuation only, we can set the torque acting on the root joint to be always zero. This choice leads to the following equations of motion

$$M \begin{pmatrix} \ddot{\boldsymbol{q}}_{\boldsymbol{a}} \\ \ddot{\boldsymbol{q}}_{\boldsymbol{r}} \end{pmatrix} + b = \begin{pmatrix} \tau_a \\ 0 \end{pmatrix}, \tag{16}$$

where the subscripts "a" and "r" denote the active internal skeletal joints and the root joint respectively, and the boldfaced symbols, here, the accelerations of the internal and root joints (1) represent the unknown variables to be solved in the equations. To execute the simulation by providing torques at the active joints, we usually need a servo controller for each actuator to generate a torque compensating for error in tracking a given desired joint trajectory or to achieve other goals. Most of the physically-based character simulations in the literature lie in this category, and a freefall simulation can also be regarded as its special case when $\tau_a = 0$ or $\tau_a = Kq_a$ with a spring element in the joint. We refer readers to Shinar et al. [2008] as an example of such simulation for a skeleton-driven deformable body character. In our examples with the fish and starfish characters shown in Figures 2, 3, and 10, we generated the landing motions with the free-falling simulation.

—*Forward simulation with prescribed joint trajectories:* We may also define a simulation by prescribing the acceleration of the active joints in some cases, for example, when the reference joint trajectory for the active joints has been given.

$$M \begin{pmatrix} \ddot{q}_a \\ \ddot{\boldsymbol{q}}_{\boldsymbol{r}} \end{pmatrix} + b = \begin{pmatrix} \boldsymbol{\tau_a} \\ 0 \end{pmatrix} \tag{17}$$

In this case, the acceleration of the active joints becomes the command input of the simulation, and the acceleration of the passive root joint (and, if needed, the torques acting on the active joints as well) will be obtained by solving the equations of motion.

Fig. 10. A jump turn of a starfish character: In order to mimic a lifelike self-propelling motion, we ran the two-way simulation by giving joint commands on the active skeletal joints only. The root joint was set to be passive and no external forces except contact forces were applied to the simulation. See Section 7 for how to control the starfish character to make the jump turn.

Because the prescribed joints follow the given trajectory kinematically or exactly, such joints do not need servo controllers, but they will lose compliance. Note that, though the active joints in the skeleton are kinematically following the reference trajectory, the global skeletal motion is obtained after integrating the root joint acceleration which is computed from the skeletal dynamics. We mainly used this type of simulation to create realistic motions for self-propelled characters. Examples can be seen in Figures 2, 3, and 10. (See Section 7 for a description of how we set the acceleration of the active joints.) Because the resulting global motion can be regarded as a function of a given active joint trajectory, this approach has also been applied to find optimal joint trajectories for ballistic characters [Albro et al. 2000; Sulejmanpašić and Popović 2005].

—*Simulation of secondary motion of the passive deformable body with a fully prescribed skeletal motion:* Another common scenario for physically-based animation would be obtaining the secondary motion of a soft elastic body that is driven by a given global skeletal motion as done in Capell et al. [2002a]. In this case, we do not need to solve the equations of motion for the skeleton (15) because we already know the global motion of the skeleton. (If needed, we can compute the joint torques with the given joint acceleration from the equations and this is called *inverse dynamics*.) Instead, at every time step, the global positions and velocities of the fixed nodes attached to the bones must be updated using kinematics. The state of the deformable body at the next time step is then obtained by integrating the solution of the particle dynamics (14). Note that, in this case, the secondary motions of the soft body do not affect skeletal motions (one-way simulation). We obtained the jiggling soft body motion of Fatman shown in Figure 1 in this way.

In general, the command input on a joint can be either torque or acceleration during the simulation and the command type does not have to be same for all joints. The equations of motion can be rewritten as

$$M \begin{pmatrix} \ddot{q}_u \\ \ddot{\boldsymbol{q}}_{\boldsymbol{v}} \end{pmatrix} + b = \begin{pmatrix} \boldsymbol{\tau_u} \\ \tau_v \end{pmatrix}, \tag{18}$$

where the subscript "u" is for the acceleration-prescribed joints and the subscript "v" is for the joints with given, or known, torques. We can compute $(\tau_u, \ddot{q}_v)$ with known $(\ddot{q}_u, \tau_v)$ from the equations and we call this *hybrid dynamics*.[3] The second scenario with prescribed

[3]We follow Featherstone [1987] for the terminology. The term "hybrid dynamics" used in this article differs from the concepts with similar names such as the hybrid control for nonsmooth dynamical systems in control community [Grossman et al. 1993], the hybrid control by mixing kinematic and dynamic controls in computer graphics such as Zordan and Hodgins

Table I. Recursive Hybrid Dynamics

initialization {
$\dot{V}_{\text{ground}} = \begin{pmatrix} 0 \\ -g \end{pmatrix}$
}
while (forward recursion) {
$T_{\lambda(i),i} = \text{function of } q_i$
$V_i = \text{Ad}_{T^{-1}_{\lambda(i),i}} V_{\lambda(i)} + S_i \dot{q}_i$
$\eta_i = \text{ad}_{V_i} S_i \dot{q}_i + \dot{S}_i \dot{q}_i$
}
while (backward recursion) {
$\hat{J}_i = J_i + \sum_{k \in \mu(i)} \text{Ad}^*_{T^{-1}_{i,k}} \Pi_k \text{Ad}_{T^{-1}_{i,k}}$
$B_i = -\text{ad}^*_{V_i} J_i V_i - F^{\text{ext}}_i + \sum_{k \in \mu(i)} \text{Ad}^*_{T^{-1}_{i,k}} \beta_k$
if ($i \in \mathcal{P}$) {
$\Pi_i = \hat{J}_i$
$\beta_i = B_i + \hat{J}_i (\eta_i + S_i \ddot{q}_i)$
} else {
$\Psi_i = (S_i^{\text{T}} \hat{J}_i S_i)^{-1}$
$\Pi_i = \hat{J}_i - \hat{J}_i S_i \Psi_i S_i^{\text{T}} \hat{J}_i$
$\beta_i = B_i + \hat{J}_i \left\{ \eta_i + S_i \Psi_i \left(\tau_i - S_i^{\text{T}} \left(\hat{J}_i \eta_i + B_i \right) \right) \right\}$
}
}
while (forward recursion) {
if ($i \in \mathcal{P}$) {
$\dot{V}_i = \text{Ad}_{T^{-1}_{\lambda(i),i}} \dot{V}_{\lambda(i)} + S_i \ddot{q}_i + \eta_i$
$F_i = \hat{J}_i \dot{V}_i + B_i$
$\tau_i = S_i^{\text{T}} F_i$
} else {
$\ddot{q}_i = \Psi_i \left\{ \tau_i - S_i^{\text{T}} \hat{J}_i \left(\text{Ad}_{T^{-1}_{\lambda(i),i}} \dot{V}_{\lambda(i)} + \eta_i \right) - S_i^{\text{T}} B_i \right\}$
$\dot{V}_i = \text{Ad}_{T^{-1}_{\lambda(i),i}} \dot{V}_{\lambda(i)} + S_i \ddot{q}_i + \eta_i$
$F_i = \hat{J}_i \dot{V}_i + B_i$
}
}

joint trajectories described earlier is one common application of hybrid dynamics and see also, for example, Lee et al. [2009] for applying hybrid dynamics to a biomechanical upper body simulation. Note that hybrid dynamics is a generalization of traditional forward and inverse dynamics, that is, they can be regarded as the extreme cases of hybrid dynamics when all of the joints have given, or known, torques and when all of the joints are acceleration-prescribed respectively.

One possible solution for hybrid dynamics is to rearrange (15) and solve it with a direct matrix inversion. For example, the accelerations of the unprescribed joints can be obtained by $\ddot{q}_v = M_{vv}^{-1}(\tau_v - b_v - M_{vu}\ddot{q}_u)$ where $M = \begin{bmatrix} M_{uu} & M_{uv} \\ M_{vu} & M_{vv} \end{bmatrix}$, $b = \begin{pmatrix} b_u \\ b_v \end{pmatrix}$, and $q = \begin{pmatrix} q_u \\ q_v \end{pmatrix}$. The method, however, is not efficient for a complex system because it requires building the mass matrix and inverting the submatrix corresponding to the unprescribed joints, which leads to an $O(n^2) + O(n_v^3)$ algorithm where n and n_v denote the number of all coordinates and the number of unprescribed coordinates, respectively. When the DOF of the deformable body are small due to use of a coarse embedding mesh, the relative cost of the skeletal dynamics could be especially high. In this situation in particular, an efficient algorithm for the skeletal dynamics is required for a fast simulation system.

We use a linear-time hybrid dynamics algorithm which was originally introduced by Featherstone [1987]. We follow a geometric formulation for robot dynamics by Park et al. [1995] to obtain a clean and efficient algorithm which is shown in Table I for completeness. The notation used in the algorithm is summarized in Table II and we also refer the reader to Murray et al. [1994] for a detailed mathematical background. Our hybrid dynamics algorithm is so general that it can consider a range of single and multiple DOF joints efficiently, and it can handle any arbitrary set of active and passive joints in a unified manner. The hybrid dynamics algorithm consists of three recursions: two forward and one backward recursions where "forward" means the computation is repeated for each bone in the skeleton from the base bone connected to the ground by a root joint, to the end bones, and "backward" means the opposite direction.

(1) Forward recursion. Updates kinematic information such as the global position and velocity of each bone.

(2) Backward recursion. Updates the articulated body inertia and bias force of each bone.

(3) Forward recursion. Calculates either the acceleration or torque on each joint depending on whether the joint is torque-specified or acceleration-prescribed.

Because the hybrid dynamics algorithm is a generalized version of the traditional forward and inverse dynamics algorithms, we can apply it to various kinds of simulation scenarios by properly setting the joints to be either acceleration-prescribed or torque-specified according to the problem.

The relative computational cost of the skeletal dynamics depends on the complexity of the character model. For example, the cost of the skeletal dynamics accounts for about 40% of the total simulation cost when we use a coarse volumetric mesh for the deformable starfish body (Figure 13, upper left), but it decreases to less than 1% in case of a very fine mesh model (Figure 13, lower right). See Table III for the costs of the various simulation components.

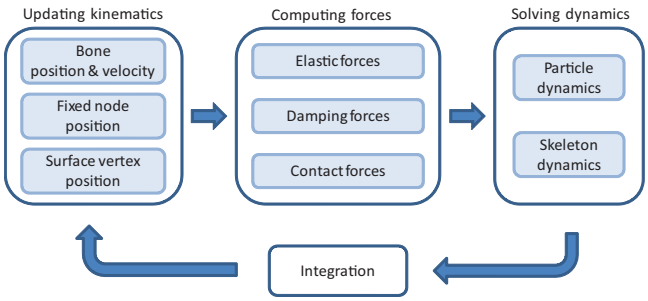

Fig. 11. Simulation flow.

Simulation

The simulation flow for a skeleton-driven deformable body system consists of four major steps (Figure 11).

(1) Updating kinematics. The global positions and velocities of the bones are obtained by executing the first forward recursion step of the hybrid dynamics algorithm in Table I. From the positions and velocities of the bones, we compute the positions and velocities of the points located on the bones such as the fixed nodes and the points where the dampers are connected to the bones. Finally, the positions and velocities of the surface vertices on the deformable body are updated by using (8).

[2002], and the hybrid simulation for deformable solids combining a mesh-based method and a point-based method such as Sifakis et al. [2007].

Table II. Notation in Table I

Symbol	Meaning	Type
$\lambda(i)$	Index of the parent body of body i	
$\mu(i)$	Index set of the child bodies of body i	
$\{i\}$	Coordinate frame attached to body i	
joint i	Joint connecting body i and its parent body	
$\mathcal{P}$	Index set of prescribed joints	
q_i	Coordinates of joint i	$\Re^{n_i}$
τ_i	Torque(or force) acting on joint i	$\Re^{n_i}$
$T_{\lambda(i),i}$	Homogeneous transform from $\{\lambda(i)\}$ to $\{i\}$	$SE(3)$
V_i	Generalized velocity of body i, viewed in $\{i\}$	$se(3)$ or $\Re^6$
$\dot{V}_i$	Component-wise time derivative of V_i	$se(3)$ or $\Re^6$
S_i	Jacobian of $T_{\lambda(i),i}$, i.e., $T^{-1}_{\lambda(i),i}\dot{T}_{\lambda(i),i} = S_i\dot{q}_i$	$\Re^{6\times n_i}$
J_i	Generalized inertia of body i, viewed in $\{i\}$	$\Re^{6\times 6}$
F_i	Generalized force transmitted to body i from its parent through the connecting joint i, viewed in $\{i\}$	$dse(3)$ or $\Re^6$
F_i^{ext}	Generalized external force acting on body i from environment, viewed in $\{i\}$	$dse(3)$ or $\Re^6$
$\hat{J}_i$, B_i	Articulated inertia of body i and corresponding bias force	$\Re^{6\times 6}$, $\Re^6$
η_i, β_i	Temporary variables for efficient computation	$\Re^6$, $\Re^6$
g	Magnitude and direction of gravity	$\Re^3$
Ad_T	$\text{Ad}_T = \begin{bmatrix} R & 0 \\ [p]R & R \end{bmatrix}$ where $T = (R, p) \in SE(3)$, $R \in SO(3)$, $p \in \Re^3$ and $[\cdot]$ denotes the skew-symmetric matrix representation of a 3-dimensional vector.	$\Re^{6\times 6}$
ad_V	$\text{ad}_V = \begin{bmatrix} [w] & 0 \\ [v] & [w] \end{bmatrix}$ where $V = (w, v) \in se(3)$ or $\Re^6$	$\Re^{6\times 6}$
Ad^*_T	$\text{Ad}^*_T = (\text{Ad}_T)^{\text{T}}$	$\Re^{6\times 6}$
ad^*_V	$\text{ad}^*_V = (\text{ad}_V)^{\text{T}}$	$\Re^{6\times 6}$

(2) Computing forces. After collision detection, penalty forces and Coulomb friction forces are applied to the penetrating vertices on the surface. Then, the frictional contact forces are transformed into equivalent nodal forces by using (13). Elastic forces acting on the nodes are obtained as explained in Sections 3 and 4.3. Damping forces acting on the free nodes are computed by using (12).

(3) Solving the equations of motion. The accelerations of the free nodes are computed from (14). The accelerations of the torque-specified joints in the skeleton are obtained by executing the second (backward) and third (forward) recursions in the hybrid dynamics algorithm. The forces transmitted from the nodes to the bones are handled as the external forces in the backward recursion step, and the joint input such as torques or accelerations, are considered in the last forward recursion step.

(4) Integration. The system state, which consists of the position and velocity of the free nodes and the displacement and velocity of the skeletal joints, at the next time step is computed by integrating the accelerations obtained by solving the equations of motion.

We used the mixed Euler integrator, which is also known as the symplectic Euler method [Stern and Desbrun 2006], for our experiments. The integrator updates the system velocity with the acceleration obtained by solving the equations of motion, and then updates the system displacement using the updated velocity. The mixed Euler integrator is as easy to implement as the explicit Euler integrator, but it has much better stability so that we can choose a reasonable step size for integration. See Section 8 for a discussion on the choice of the integrator.

In our simulation system, there are a few important factors which help the entire simulation to be fast and to have better stability. First of all, using the coarse mesh significantly decreases the computational cost per each time step by reducing the DOF of the physics model and allows larger step size because of the enlarged element size. The mesh embedding leads to speedup factors of up to orders of magnitude. Second, the stiffness of each nonlinear element is always maintained within given boundaries due to the diagonalization technique, and this makes the simulation stable even when the skeletal motion causes large element deformation. Third, by using the hybrid dynamics algorithm, the skeleton-driven deformable body character can trace a given reference skeletal motion without relying on a servo controller which could make the whole system very stiff and the simulation unstable. Finally, the computational cost per step size increases linearly as the model size grows, which is favorable for complex characters such as Fatman in Figure 1.

As explained before, our skeleton-driven deformable body system uses fixed nodes to attach the soft body to the skeleton. Because the fixed nodes in the volumetric mesh are firmly attached to the skeletal links, the finite elements located near the joints may experience unrealistically large deformation, such as element inversion, depending on the underlying skeletal motion, especially when the mesh is very coarse (e.g., Figure 4). Though our simulation system handles such elements well so that they usually do not cause critical problems such as instability, the character surface obtained by interpolating the mesh nodes may have undesirable scars at moments of large deformation, and this could degrade the overall animation quality. If such an artifact is problematic, the user may wish to use a denser volumetric mesh even with the additional computational cost required due to the increased model complexity. For example, if we wanted to make the creases shown in Figure 4 smaller, we would have to use a denser volumetric mesh for the worm character. Actually, for the Fatman model (Figure 15), we have already used a relatively complex volumetric mesh compared to the other models because the human skeleton is more complicated than the others, and the output quality demanded for such a human-like character is generally very high. In such cases with a relatively large number of finite elements, we need an additional speedup for achieving fast simulation at interactive rates. We address this issue by implementing parallel computation on the GPU, which will be discussed in the next section.

6. PARALLEL COMPUTING

The simulation speed can be increased further by carrying out many calculations simultaneously. However, there are limits in the amount of speedup that can be obtained using parallelization. In this section we explain our parallel implementation of the algorithms presented in this article and discuss the issues limiting computational performance.

In the skeleton-driven deformable body system, many parts of the computation are parallelizable such as the elastic force computation and solving the equations of motion of the particle system. The overall performance, however, significantly depends on the

Table III. Speedup by Parallel Computing

Complexity of vol. mesh (Starfish)		coarse		moderate		fine		very fine	
		GPU	CPU	**GPU**	CPU	**GPU**	CPU	**GPU**	CPU
Calc. time per time step (msec)	Update fixed nodes and surf.	**0.117**	0.094	**0.130**	0.103	**0.130**	0.117	**0.159**	0.175
	Transform contact forces[a]	**0.300**	0.047	**0.106**	0.048	**0.075**	0.053	**0.061**	0.075
	Compute elastic forces	**0.123**	0.088	**0.252**	1.558	**0.355**	4.105	**0.945**	17.611
	Compute damping forces	**0.011**	0.002	**0.016**	0.042	**0.019**	0.088	**0.044**	0.319
	Solve particle dynamics	**0.009**	0.003	**0.013**	0.031	**0.014**	0.069	**0.030**	0.248
	Transmit nodal forces to bones	**0.061**	0.003	**0.067**	0.063	**0.072**	0.134	**0.106**	0.503
	Solve skeleton dynamics (CPU)	0.118							
	Update skeleton kinematics (CPU)	0.049							
	Collision detection (CPU)	0.034							
	Total computation time (msec)	**0.823**	0.437	**0.784**	2.047	**0.865**	4.767	**1.547**	19.132
Speedup by parallel computing on GPU		0.5×		2.6×		5.5×		12.4×	
Simulation (free falling)	Step size for integration[b] (msec)	3.0		1.0		0.5		0.2	
	Calc. time for 1 sec simul. (sec)	**0.27**	0.15	**0.78**	2.05	**1.73**	9.53	**7.73**	95.66
Model info.	Total DOF	260		3365		7262		26741	
	Num. of nodes in vol. mesh	78		1113		2412		8905	
	Num. of elements in vol. mesh	200		3428		9014		37573	
	Num. of nodes in surf. mesh	1162							
	DOF of skeleton (num. of bones)	26 (16)							

The speedup gained from parallel computation depends on the complexity of the model. This table shows the relationship between the speedup ratio and the complexity of the volumetric mesh in the starfish character. We used four volumetric mesh models (coarse, moderate, fine, very fine) shown in Figure 13 for the comparison.

[a]The calculation time for contact force transformation on our GPU implementation decreases as volume mesh complexity increases, and this is because we used the same surface model in the experiment. More specifically, the average number of surface nodes corresponding to each volumetric node decreases as volume mesh complexity increases in our experimental setup.

[b]For comparison purpose, the step sizes for integration shown here are maximum values with which a free-fall simulation becomes stable. The test was performed on a desktop machine with a 2.8GHz Intel Core2 Quad CPU and NVIDIA GeForce GTX 280 GPU.

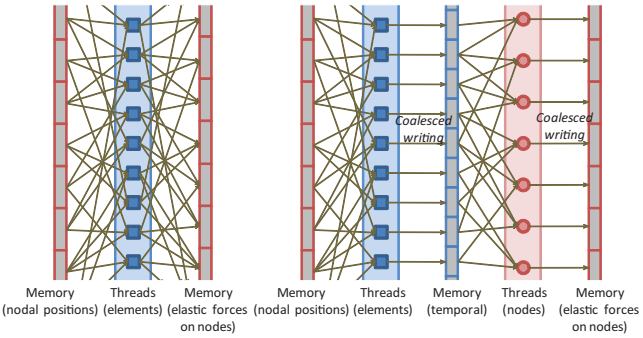

Fig. 12. Parallel computation of elastic forces: In the task, the input is the nodal positions and the output is the elastic forces acting on the nodes. (Left) An ideal parallel structure for computing the elastic forces acting the nodes. However, this cannot be realized because of concurrent memory access during writing. (Right) Our implementation consists of two parallel processes to avoid the concurrent memory accessing problem and to achieve coalesced memory access in writing.

computational structure, so we need to design the parallel structure carefully to get a maximum speedup. Figure 12 shows an example of our computational structure for elastic forces where the inputs are the nodal positions and the outputs are the elastic forces acting on the nodes. Because we compute the nonlinear strain and stress per element as explained in Section 3, the calculation can be parallelized at the element level (Figure 12, left). In this case, however, because we need to sum up the elastic forces exerted by adjacent elements to obtain the net elastic force acting on each node, multiple threads running in parallel at the element level may access the same memory address assigned to nodal force for summation of the elastic forces. Such concurrent memory access for writing may cause unexpected problems such as data loss. In our implementation we avoided the problem by splitting the task into two parallel processes (the elastic force computation at the element level and the force summation at the node level) and using a temporary memory space for the data flow between the two processes (Figure 12, right). There is another issue related to the speed of memory access that must be considered when designing the structure. Because we need a large memory space for saving information related to the elements and the nodes, we use a global memory region in which large chunks of memory can be allocated but the memory accessing is slow in general. The global memory can be accessed most efficiently when the access pattern has been specially designed in a *coalesced* way, and if it is not coalesced, the memory access could be extremely slow especially for writing. It would be most desirable to design processes to always have coalesced memory access for both reading and writing, but this is impossible because the sizes of input and output data are different so that at least one of the reading and writing accesses cannot be coalesced. Because the noncoalesced memory access is more critical in writing than reading, we designed the structure of our parallel computation to have a coalesced memory access for writing to increase the overall performance (Figure 12, right). Using this methodology we parallelize the following.

—Updating the positions of the surface vertices. We use a single parallel process running at the vertex level to obtain the position of each vertex from neighboring nodal positions.

—Computing the nodal forces equivalent to the surface contact forces acting on the vertices. We use two parallel processes for this task. The first process runs at the vertex level, and for each

vertex, it computes forces acting on the neighboring nodes that are equivalent to the contact force at the vertex. The second process runs at the node level, and for each node, it sums up the forces acting on the node which have been computed by the first process.

—Computing the elastic forces acting on the nodes (described before and shown in Figure 12).

—Computing the damping forces acting on the nodes, and calculating the forces and moments to be transmitted to the bones in the skeleton. We designed two parallel processes to accomplish the task. The first process runs at the node level. If the node is a free particle, it first computes the damping force acting on the node. Then, the process computes the force and moment to be transmitted to the skeleton for each node. We transmit the damping forces acting on the free particles to their reference bones, and the contact and elastic forces on the fixed particles to the bones to which the nodes are attached. The forces and moments to be transmitted to the bones are computed with respect to a local frame attached to each bone. The second process runs at the bone level, and for each bone, it sums up the forces and moments transmitted from the neighboring free nodes and the fixed nodes attached to the bone. The net force and moment acting on each bone will be sent to the CPU memory for the solution of the skeleton dynamics.

—Solving the equations of motion for the particle system. A single process runs at the node level for the task. For each node, it calculates the nodal acceleration using the forces (contact, elastic, damping, and the gravitational forces) and obtains the nodal velocity and position in the next time-step with the mixed Euler integration.

In our implementation, we chose the CUDA architecture from NVIDIA to execute the massive computation for the deformable body in parallel on the GPU (Graphics Processing Unit) because of its easy integration with other nonparallelizable C/C++ modules. In some parallelized tasks with noncoalesced memory reading, we texture a specific global memory range for caching to reduce the memory access time for reading. We use traditional serial computing on CPU (Central Processing Unit) for the other computations such as solving the skeleton dynamics, where the equations of motion for the bones are highly coupled by the joints. A point-triangle collision detection algorithm running on CPU is used in our current simulation system. Though the simple collision detection works well for our test models, it could become a critical computational bottleneck when the number of vertices on the surface model is very large. Using parallelized collision detection algorithms, such as Govindaraju et al. [2003] and Zhang and Kim [2007], would be a good choice for such a case.

In the case of the dancing Fatmat example shown in Figure 1, the parallel computation speeded up the simulation about 10 times faster than the CPU-only implementation, and in our experiments we were able to achieve a speedup of a factor of up to 12 through parallel computation. The speedup ratio depends on the complexity of the model such as the number of elements, nodes, and bones, and for comparison purpose, we show the computation times for four different models of a starfish character in Table III, where we used different model complexities for the volumetric mesh while using the same surface and skeleton models (Figure 13). If the volumetric mesh is too coarse, the parallel computation may decrease the simulation speed because the profits gained from the parallelism are less than the overhead such as relatively low clock speed and slow memory access in the GPU. The speedup gained from parallel computation increases as the complexity of the volumetric mesh grows, which is expected because a model with more nodes and elements can have more room to be accelerated by parallel computation. However, increasing model complexity, especially in the volumetric mesh of the deformable body, requires us to use a smaller step size for integration due to the small element size, and this leads the overall simulation speed to become slow. Therefore, the model reduction as well as our engineering decisions to effectively support such a reduced model (Sections 3, 4, and 5) are still necessary to achieve a fast simulation speed even in the parallel computing case.

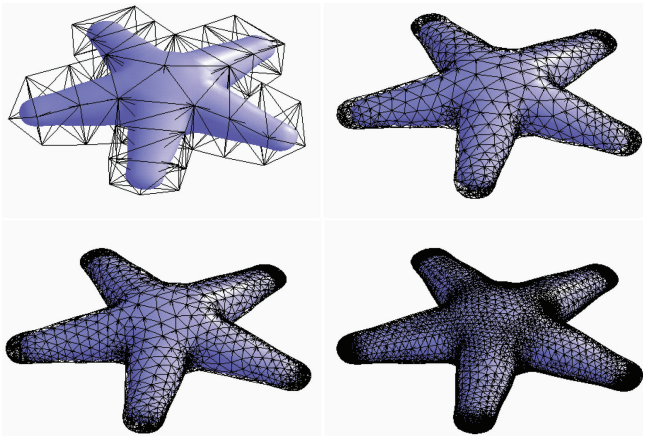

Fig. 13. Starfish models with different complexity of the volumetric mesh used in Table III (coarse, moderate, fine, very fine): The coarse mesh (upper left) was designed manually while the others (moderate, fine, very fine) were generated using NETGEN [1997]. In the coarse mesh model, we assumed that the elastic material fills only the inside of the surface so the nodal mass and the element volume ratio were obtained as described in Section 4.3. For the other models, however, their elements were assumed fully filled with the material for convenience sake in modeling. See Figure 15 for more information on the surface mesh and the skeleton of the starfish character.

It might be possible to additionally increase the speed by optimizing the parallel computing structure, but the gain would be limited because the data flow between the different size of spaces inherently causes inefficiency in memory access for parallel computation (Figure 14). See, for example, de Farias et al. [2008] and Comas et al. [2008] for parallel computation of deformable bodies (without skeletons) where they achieved a few tens of speedup, and as far as we know, there is no previous work on parallel computing for skeleton-driven deformable body simulation.

7. RESULTS

Various kinds of simulations have been tested in our system with the three-dimensional skeleton-driven deformable body models shown in Figure 15. The simulations with all the test models can run in real time or near real time on a desktop machine with 2.8 GHz Intel Core2 Quad CPU and NVIDIA GeForce GTX 280 GPU. See Table IV for detailed information on the simulation speed for the test models.

In order to create physically plausible motions of the self-propelling starfish, fish, and worm characters in Figures 2, 3, and 10, we used only internal forces, that is, the torque or acceleration for the active skeletal joints, to drive the overall character motions. The secondary motions of the deformable bodies and the passive root joints were obtained through two-way simulation where the soft body motion affects the global skeleton motion and vice versa, and no intentional external forces were used in the simulation. To obtain desired character motions given the underactuated nature of

Table IV. Simulation Speed

Model	Calc. time (sec) for 1sec simul.[a]	h (msec)[e]	dof	sdof	vnodes	velems	snodes
Fatman	1.33[b] (GPU)[c]	0.5	4887	60	2121	8619	34362[f]
Starfish	0.49[d] (CPU)	1.0	260	26	78	200	1162
Fish	0.50 (CPU)	1.0	258	9	107	415	958
Worm	0.57 (CPU)	1.0	543	9	224	714	262

This table shows the calculation time for running a 1 sec simulation with the test models in Figure 15, corresponding step size for integration, and model information such as total DOFs of the models (dof), the DOFs of the skeletons (sdof), the numbers of the nodes (vnodes) and elements (velems) in the coarse volumetric mesh, and the number of nodes (snodes) in the fine surface mesh. The test was performed on a desktop machine with a 2.8GHz Intel Core2 Quad CPU and NVIDIA GeForce GTX 280 GPU. (a) The averaged computation times were measured in the simulations for the dancing Fatman (Figure 1, 60 sec), the jump-turning starfish (Figure 10, 2 sec), the jumping fish (Figure 3 (upper), 4 sec) and the rolling worm (Figure 3 (lower), 4 sec) with the step sizes shown in the table, and the corresponding animations are shown in the accompanying video. The data only accounts for the computation times for the open-loop physics simulation with given input commands (e.g., the accelerations or torques of the active joints in two-way simulation), and does not include the cost for computing the commands in case of using a high level controller to guide the simulation in a desired way. High quality rendering, which was done in off-line, is not also considered in the calculation times. If needed to control the characters interactively during the simulation, we used a simple mesh grid rending, as shown in Figure 15, which is updated at about 60 frames per second. (b) The average computation time for running the same dancing Fatman simulation in our CPU only implementation was 13.24 sec per 1 sec simulation, so we got an additional speedup of a factor of 10 by using parallel computation. (c) (GPU) and (CPU) denote the elapsed times measured in our parallel implementation using GPU and our CPU only implementation respectively. (d) For reference, the average calculation time for the example of the escaping starfish (Figure 2, 17 sec) was 0.94 sec per 1 sec simulation. In the example, interaction between the character and the movable obstacle must be handled in the simulation, and in our test, more than 40% of the calculation time was spent in collision checking with a simple point-triangle collision detection algorithm, which could be improved by using an advanced collision detection algorithm. (e) The values shown here are the fixed step sizes used in the simulations. For reference, we were able to double the step size for the worm and starfish examples, but we could not double the step size for the Fatman and fish examples without driving the simulation into instability. (f) We did not perform collision checking in the dancing Fatman simulation.

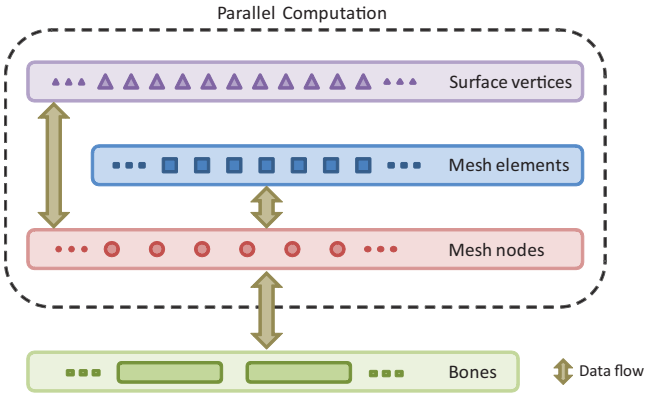

Fig. 14. Data Flow in Parallel Computation: To implement parallel computation for skeleton-driven deformable bodies, we need to handle data flow between spaces with different sizes. For example, we need to implement two tasks having data flow between the surface vertices and the mesh nodes – (a) transformation of the contact forces on the surface vertices into the equivalent nodal forces, and (b) updating the positions and velocities of the surface vertices from the nodal positions. Such a data flow between different spaces causes inefficiency in memory accessing and limits the overall performance of parallel computation in our system.

the physics model, we generated appropriate motor commands on the active skeletal joints with a combination of direct control and keyframe control as presented in Kim and Pollard [2011]. Using direct control one can interactively control the character with a mouse drag. The mouse cursor is regarded as a desired trajectory of a selected bone and, at every time step, the desired position will be transformed into an optimal joint command (acceleration or torque) which can make the character follow the user input trajectory as closely as physically possible. For keyframe control, we interpolate given keyframes, or keyposes, with a B-spline to obtain an active joint trajectory and advance the simulation using hybrid dynamics.

To make the acrobatic jump motion of the fish character in Figure 3(upper), we first guided the head with a mouse drag under direct control in order to prepare the jump by bending the body. Then, we used keyframe control to make a takeoff by giving a desired final pose and the time for transition, and finally switched into a free-fall simulation mode for landing where all of the skeletal joints are set to be passive. In Figure 3(lower) we guided the left end of the worm character with direct control and then switched into keyframe control, providing a final pose of circular shape to get a rolling motion. To make a jump turn of the starfish character in Figure 10, we set desired position and orientation for the center part of the skeleton in direct control, and switched into free-fall simulation for landing. Note that all the motions of the self-propelling characters were generated by internal forces only and no intentional external forces, or the hand of God [van de Panne and Lamouret 1995], were used in the simulations.

An example of more complicated interaction between a character and a changeable environment is shown in Figure 2 where a starfish has been trapped and is trying to escape by attempting various trials such as hitting the obstacle with a jump and lifting up the obstacle's edge with its arm. We used direct control to guide the character motion to achieve these motions. Note that all interactions between the character, the obstacle, and the ground are automatically handled through the contact mechanism in the two-way simulation, and the motor command on the character's active joints, which is generated by the direct controller, is the only input of the simulation system.

Our system can also handle one-way simulation within its unified framework by simply changing the property of the joints. For example, when a global skeletal motion is given as in the case of the dancing Fatman in Figure 1, we simply set all of the skeletal joints to be prescribed, which means all the skeletal joints including

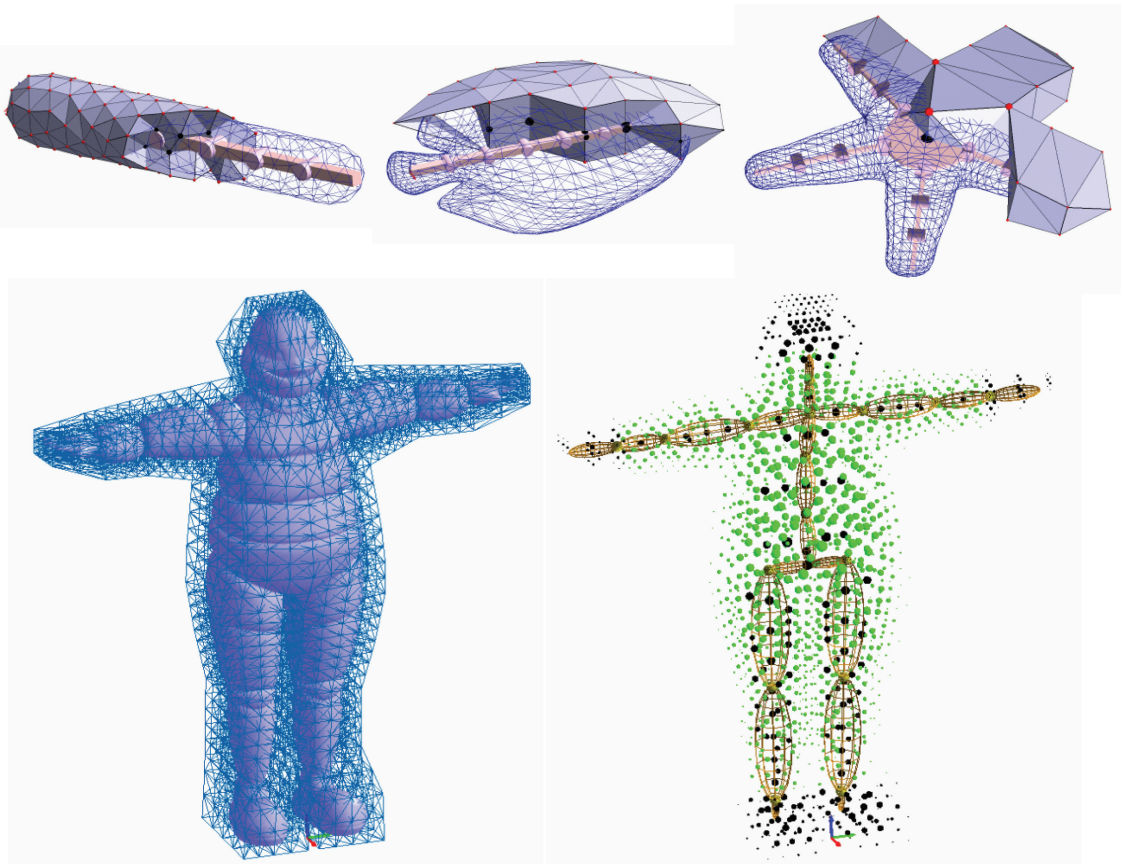

Fig. 15. Test models. (Worm, fish, and starfish) The wire frames are the fine surfaces of the deformable bodies, and the solid meshes represent the embedding coarse volumetric meshes. The nodes fixed to bones are depicted as black spheres while the others are red ones. (Fatman, left) The wire frame represents the coarse volumetric mesh embedding the fine solid surface of the character. (Fatman, right) The character's skeleton and the mass distribution in the deformable body are shown. The mass particles (spheres) are located at the coarse mesh nodes and the size of the spheres are proportional to the mass. The particles fixed to bones are colored black while the others are green. See Table IV for more information on the models.

the root joint will exactly follow the given trajectory and they will not be affected by the secondary motions of the deformable body. To create the dancing motion of the Fatman character, we drove the skeleton using a captured human motion data, which had been modified to fit the character's skeleton. The jiggling motion of the deformable body was obtained in almost real time in our system.

The effect of the selective diagonalization technique (Section 3.3) on the simulation speedup is shown in Table V. As mentioned before, the technique is more effective when the number of finite elements is large. For example, when equipped with an existing general SVD algorithm[4], the selective diagonalization technique boosted the simulation of the fine mesh starfish model shown in Figure 13 by a factor of more than 3 in the time required for the entire dynamics simulation, but resulted in a speedup of about 70% for the coarse mesh starfish model. The effectiveness of the technique highly

Table V. Speedup by Selective Diagonalization

Model (type of motion)		speedup factor with $SVD_{m\times n}$	speedup factor with $SVD_{3\times 3}$
Fatman (dancing)		1.44	1.19
Fish (escaping)		1.33	1.17
Worm (rolling)		1.43	1.23
Starfish (free-falling)	coarse	1.68	1.16
	moderate	3.31	1.67
	fine	3.60	1.78
	very fine	3.42	1.72

This table shows the effect of the selective diagonalization technique in Section 3.3 on the simulation speedup. The speedup factor represents the ratio of the elapsed simulation times without and with the technique. The effectiveness of the technique varies depending on the type of motion, complexity of the finite element model, and the choice of SVD algorithm for the diagonalization of the element deformation. For example, the technique boosted the simulation speed by a factor of more than 3 when we tested with the finely meshed starfish model (shown in Figure 13), equipped with an existing general SVD algorithm. On the other hand, the same technique could only increase the computation speed by 16% when tested with the coarsely meshed starfish model and a fast dedicated SVD algorithm. See the footnotes 4 and 5 for the SVD algorithms we tested with.

[4]We tested with an SVD algorithm for $m \times n$ matrices obtained from Numerical Recipes in C. For reference, it has been reported that an approximate estimate of the computational cost of such an iterative SVD algorithm for general $m \times n$ matrices is $4m^2n + 8mn^2 + 9n^3$ flops [Golub and Van Loan 1996].

depends on the choice of SVD algorithm for the diagonalization of the element deformation. When we tested with our own implementation of a fast dedicated SVD algorithm[5], which is about four times faster than the general one, we obtained a speedup by a factor of 15% to 80% depending on the model complexity and the type of motion.

8. CONCLUSION

We have introduced a fast physics simulation system for skeleton-driven deformable body characters. Our approach can handle both one-way and two-way simulations within a unified framework. We can create physically plausible realistic character motions through a two-way simulation where the skeleton with passive root joint, the particle system representing the deformable body, and the environment interact with each other in the solution of fully nonlinear dynamics equations. Secondary jiggling motions of the deformable body driven by a global skeletal motion can also be easily computed through a one-way simulation.

The simulation speed in our system is fast enough to be used with an interactive user interface system such as that shown in Kim and Pollard [2011]. Most of the skeleton-driven deformable body characters tested in the article can be simulated in real time. Also, our formulation, which takes the fine character surface into account, preserves the original mechanical properties of the deformable body as much as possible even in the low-dimensional representation, which will improve the physical fidelity of produced animations.

Detailed discussions on our engineering decisions for speed, modeling accuracy, and stability have been presented in the article. We chose an embedding mesh representation to reduce model size. We use mass lumping for deformable bodies and a hybrid dynamics algorithm for skeletons to make the overall computation per time step linear in the model size. We also implemented parallel computation which is quite effective for complicated models like our Fatman character. We use nonlinear finite elements for better handling of large deformation in the simulation. Though we chose a coarse volumetric mesh to represent the motions of the deformable body for speedup, we also consider the fine surface of the character to increase the modeling accuracy by capturing the mechanical properties of the deformable body as correctly as possible and by evaluating the contact forces directly at vertices of the fine surface mesh. The element stiffness is maintained within given boundaries by using our selective diagonalization technique, and this helps the simulation to remain stable. If the trajectory of some or all of the skeletal joints are given, we can drive the character using a hybrid simulation with no servo controller for the prescribed joints, and this also improves the stability of the simulation.

Our choices for fast fully dynamic simulation of skeleton-driven deformable body characters includes nonlinear material and explicit integration, and this is quite different from the usual choice for fast elastic body simulation in computer graphics, that is, the combination of linear material and implicit, or semi-implicit, integration. One of the main reasons for choosing a linear material in many existing techniques for elastic body simulation would be that it produces a constant local stiffness matrix and this can lead to a Symmetric Positive Definite (SPD) matrix system for implicit integration which can be solved efficiently by, for example, conjugate gradient iteration. However, in the case of our skeleton-driven deformable body system, because the elastic forces in the soft body are nonlinearly dependent on the joint coordinates of the skeleton through the (hard constrained) fixed nodes, the matrix system for implicit integration is not SPD any more and is expensive to solve. Moreover, in order to build the matrix system for implicit integration, we need to differentiate the fully nonlinear dynamics equations of the skeleton as well as the elastic forces at every time step, which is also computationally expensive. We chose explicit integration to avoid such problems in our skeleton-driven deformable body simulation system. Because explicit integration does not require local stiffness matrices for the finite elements, we can choose nonlinear elements to increase accuracy at very little cost. One demerit of explicit integration is that it requires very tiny step size when the finite elements are small, and this could make the simulation speed extremely slow. Mesh embedding, which is also one of our choices, addresses this problem by reducing the number of nodes and elements in the volumetric mesh of the deformable body and by making the element size large enough to be used in explicit integration with a reasonable step size.

Though the mixed Euler integrator we chose in our current implementation works well in our experiments with a reasonable step size, such an explicit integrator may limit the flexibility of our system in choosing the simulation resolution, especially when the user wants to play with a character having a small size but with a complicated geometry and skeleton. In fact, a naïve implementation of an implicit or semi-implicit integrator can be easily added to our simulation framework. However, as explained earlier, we need to find a way to efficiently solve the non-SPD matrix system for the implicit integration, which would be challenging future work. Perhaps using parallel computation could be one possible way to obtain a fast solution for such a nontrivial matrix system. Finding an approximate but efficient solution would be another direction to explore as Galoppo et al.[2007] did in their skin-layer deformation-based method. Releasing the hard constraints to produce a SPD system for implicit integration and enforcing the constraints later using pre/poststabilization as in Shinar et al. [2008] would also be an interesting extension.

In our current system we use barycentric coordinates to map the fine character surface onto the coarse volumetric mesh representation of the deformable body. Though this linear mapping is simple to implement and fits nicely in our simulation system, it creates a first-order discontinuity in the surface across the element boundary which may degrade the quality of the output animation. Currently we use a filter to smooth out the resulting artifacts when we render the video in a commercial animation tool, but this is not a fundamental solution to the problem because it will blur out fine geometric detail on the character surface. One possible way to treat the problem would be rerendering the surface for video production with an optimized mapping rule minimizing discontinuity such as the modified barycentric interpolation by Huang et al. [2008]. Such a manipulation of the simulated results, however, would break physics in the character motions and this could also degrade the animation quality. It would be interesting future work to incorporate an advanced mapping rule such as the harmonic coordinates by Joshi et al. [2007] into the physics formulation to replace the barycentric coordinates used in our current framework.

Volume preservation, which is not considered in our formulation, can also be a critical issue in some particular applications such as a surgery simulation involving highly incompressible soft bodies,

[5]We also tested with our own implementation of a fast SVD algorithm for 3×3 matrices based on an efficient solution for eigenvalues of symmetric 3×3 matrices proposed in Smith [1961]. The SVD algorithm requires up to 152 multiplications in the implementation. Though the solution for the eigenvalues was originally intended for an application requiring moderate accuracy and its numerical robustness has not been well investigated [Smith 1961], it suffices for our simulation examples. For reference, the diagonalization in (4) requires an additional 25 multiplications (in our implementation) to make sure that U and V are pure rotations.

so incorporating existing volume conservation techniques such as that of Irving et al. [2007] in our formulation would also be a good extension.

ACKNOWLEDGMENTS

The authors gratefully acknowledge Moshe Mahler for help with Maya modeling and rendering. We also thank Sangil Park for providing motion-captured data for the dancing Fatman example, Junsik Kim for advice on the implementation of the parallel computation, and the graphics group at Carnegie Mellon for their helpful comments.

REFERENCES

ALBRO, J. V., SOHL, G. A., BOBROW, J. E., AND PARK, F. C. 2000. On the computation of optimal high-dives. In *Proceedings of the IEEE International Conference on Robotics and Automation*. IEEE, 3958–3963.

BARAFF, D. 1996. Linear-time dynamics using lagrange multipliers. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH'96)*. ACM, New York, 137–146.

BARAFF, D. AND WITKIN, A. 1998. Large steps in cloth simulation. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH'98)*. ACM, New York, 43–54.

BARBIČ, J. AND JAMES, D. L. 2005. Real-time subspace integration for St. Venant-Kirchhoff deformable models. *ACM Trans. Graph 24,* 3, 982–990.

BARGTEIL, A. W., WOJTAN, C., HODGINS, J. K., AND TURK, G. 2007. A finite element method for animating large viscoplastic flow. *ACM Trans. Graphics. 26,* 3.

BASAR, Y. AND WEICHERT, D. 2000. *Nonlinear Continuum Mechanics of Solids*. Springer.

BONET, J. AND WOOD, R. D. 1997. *Nonlinear Continuum Mechanics for Finite Element Analysis*. Cambridge University Press.

BRIDSON, R., FEDKIW, R., AND ANDERSON, J. 2002. Robust treatment of collisions, contact and friction for cloth animation. *ACM Trans. Graphics. 21,* 3.

BRO-NIELSEN, M. AND COTIN, S. 1996. Real-time volumetric deformable models for surgery simulation using finite elements and condensation. *Comput. Graph. Forum*. 57–66.

CAPELL, S., BURKHART, M., CURLESS, B., DUCHAMP, T., AND POPOVIĆ, Z. 2005. Physically based rigging for deformable characters. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA'05)*. ACM, New York, 301–310.

CAPELL, S., GREEN, S., CURLESS, B., DUCHAMP, T., AND POPOVIĆ, Z. 2002a. Interactive skeleton-driven dynamic deformations. *ACM Trans. Graph. 21,* 3.

CAPELL, S., GREEN, S., CURLESS, B., DUCHAMP, T., AND POPOVIĆ, Z. 2002b. A multiresolution framework for dynamic deformations. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA'02)*. ACM, New York, 41–47.

CHOI, K.-J. AND KO, H.-S. 2002. Stable but responsive cloth. *ACM Trans. Graph. 21,* 3.

CHOI, M. G. AND KO, H.-S. 2005. Modal warping: real-time simulation of large rotational deformation and manipulation. *IEEE Trans. Visual. Comput. Graph. 11*, 91–101.

COMAS, O., TAYLOR, Z. A., ALLARD, J., OURSELIN, S., COTIN, S., AND PASSENGER, J. 2008. Efficient nonlinear fem for soft tissue modelling and its gpu implementation within the open source framework sofa. In *Proceedings of the 4th International Symposium on Biomedical Simulation (ISBMS'08)*. Springer-Verlag, Berlin, 28–39.

DE FARIAS, T. S. M., ALMEIDA, M. W. S., TEIXEIRA, J. M. X., TEICHRIEB, V., AND KELNER, J. 2008. A high performance massively parallel approach for real time deformable body physics simulation. In *Proceedings of the 20th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'08)*. 45–52.

FALOUTSOS, P., VAN DE PANNE, M., AND TERZOPOULOS, D. 1997. Dynamic free-form deformations for animation synthesis. *IEEE Trans. Visual. Comput. Graph. 3,* 3, 201–214.

FEATHERSTONE, R. 1983. The calculation of robot dynamics using articulated-body inertias. *Int. J. Robotics Res. 2,* 1, 13–30.

FEATHERSTONE, R. 1987. *Robot Dynamics Algorithms*. Kluwer.

GALOPPO, N., OTADUY, M. A., MECKLENBURG, P., GROSS, M., AND LIN, M. C. 2006. Fast simulation of deformable models in contact using dynamic deformation textures. In *ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, M.-P. Cani and J. O'Brien, Eds., Eurographics Association, 73–82.

GALOPPO, N., OTADUY, M. A., TEKIN, S., GROSS, M., AND LIN, M. C. 2007. Soft articulated characters with fast contact handling. *Comput. Graph. Forum 26,* 3.

GOLUB, G. H. AND VAN LOAN, C. F. 1996. *Matrix Computations* 3rd Ed. The Johns Hopkins University Press.

GOVINDARAJU, N. K., REDON, S., LIN, M. C., AND MANOCHA, D. 2003. Cullide: interactive collision detection between complex models in large environments using graphics hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics hardware (HWWS'03)*. Eurographics Association, 25–32.

GROSSMAN, R. L., NERODE, A., RAVN, A. P., AND RISCHEL, H., Eds. 1993. *Hybrid Systems*. Lecture Notes in Computer Science, vol. 736, Springer.

HAUSER, K. K., SHEN, C., AND O'BRIEN, J. F. 2003. Interactive deformation using modal analysis with constraints. In *Proceedings of the Graphics Interface Conference*. Canadian Human-Computer Commnication Society, 247–256.

HUANG, J., CHEN, L., LIU, X., AND BAO, H. 2008. Efficient mesh deformation using tetrahedron control mesh. In *Proceedings of the 2008 ACM Symposium on Solid and Physical Modeling (SPM'08)*. ACM, New York, 241–247.

IRVING, G., SCHROEDER, C., AND FEDKIW, R. 2007. Volume conserving finite element simulations of deformable models. *ACM Trans. Graph. 26,* 3.

IRVING, G., TERAN, J., AND FEDKIW, R. 2004. Invertible finite elements for robust simulation of large deformation. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA'04)*. Eurographics Association, 131–140.

JAMES, D. L. AND PAI, D. K. 1999. ArtDefo-accurate real time deformable objects. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH'99)*, A. Rockwood, Ed., ACM Press, N.Y., 65–72.

JAMES, D. L. AND PAI, D. K. 2002. Dyrt: dynamic response textures for real time deformation simulation with graphics hardware. *ACM Trans. Graph. 21,* 3.

JOSHI, P., MEYER, M., DEROSE, T., GREEN, B., AND SANOCKI, T. 2007. Harmonic coordinates for character articulation. *ACM Trans. Graph. 26,* 3.

KHAREVYCH, L., MULLEN, P., OWHADI, H., AND DESBRUN, M. 2009. Numerical coarsening of inhomogeneous elastic materials. *ACM Trans. Graph. 28,* 3.

KIM, J. AND POLLARD, N. S. 2011. Direct control of simulated non-human characters. *IEEE Comput. Graph. Appl. 31,* 4.

KIM, T. AND JAMES, D. 2009. Skipping steps in deformable simulation with online model reduction. *ACM Trans. Graph. 28,* 5.

LEE, S.-H., SIFAKIS, E., AND TERZOPOULOS, D. 2009. Comprehensive biomechanical modeling and simulation of the upper body. *ACM Trans. Graph. 28,* 4, 1–17.

MÜLLER, M., DORSEY, J., MCMILLAN, L., JAGNOW, R., AND CUTLER, B. 2002. Stable real-time deformations. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA'02)*. ACM, New York, 49–54.

MÜLLER, M. AND GROSS, M. 2004. Interactive virtual materials. In *Proceedings of Graphics Interface (GI'04)*. Canadian Human-Computer Communications Society, 239–246.

MURRAY, R. M., LI, Z., , AND SASTRY, S. S. 1994. *A Mathematical Introduction to Robotic Manipulation*. CRC Press.

NEALEN, A., MÜLLER, M., KEISER, R., BOXERMAN, E., AND CARLSON, M. 2006. Physically based deformable models in computer graphics. *Comput. Graph. Forum 25,* 4, 809–836.

NESME, M., KRY, P. G., JEŘÁBKOVÁ, L., AND FAURE, F. 2009. Preserving topology and elasticity for embedded deformable models. *ACM Trans. Graph. 28,* 3.

NESME, M., PAYAN, Y., AND FAURE, F. 2006. Animating shapes at arbitrary resolution with non-uniform stiffness. In *Proceedings of the Eurographics Workshop in Virtual Reality Interaction and Physical Simulation (VRIPHYS)*. Eurographics.

O'BRIEN, J. F., BARGTEIL, A. W., AND HODGINS, J. K. 2002. Graphical modeling and animation of ductile fracture. *ACM Trans. Graph. 21,* 3.

O'BRIEN, J. F. AND HODGINS, J. K. 1999. Graphical modeling and animation of brittle fracture. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH'99)*. 137–146.

PARK, F. C., BOBROW, J. E., AND PLOEN, S. R. 1995. A lie group formulation of robot dynamics. *Int. J. Robotics Res. 14,* 6, 609–618.

PENTLAND, A. AND WILLIAMS, J. 1989. Good vibrations: modal dynamics for graphics and animation. In *Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH'89)*. ACM, New York, 215–222.

RATHOD, H. T., VENKATESUDU, B., AND NAGARAJA, K. V. 2005. Gauss legendre quadrature formulas over a tetrahedron. *Numer. Math. Part. Diff. Eqs. 22,* 1, 197–219.

ROCCHINI, C. AND CIGNONI, P. 2000. Generating random points in a tetrahedron. *J. Graph. Tools 5,* 4, 9–12.

SCHÖBERL, J. 1997. NETGEN: An advancing front 2D/3D-mesh generator based on abstract rules. *Comput. Visual Sci. 1*, 41–52.

SEDERBERG, T. W. AND PARRY, S. R. 1986. Free-form deformation of solid geometric models. *SIGGRAPH Comput. Graph. 20,* 4, 151–160.

SHI, X., ZHOU, K., TONG, Y., DESBRUN, M., BAO, H., AND GUO, B. 2008. Example-based dynamic skinning in real time. *ACM Trans. Graph. 27,* 3.

SHINAR, T., SCHROEDER, C., AND FEDKIW, R. 2008. Two-way coupling of rigid and deformable bodies. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA'08)*. ACM.

SIFAKIS, E., SHINAR, T., IRVING, G., AND FEDKIW, R. 2007. Hybrid simulation of deformable solids. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA'07)*, M. Gleicher and D. Thalmann, Eds., Eurographics Association, 81–90.

SMITH, O. K. 1961. Eigenvalues of a symmetric 3 × 3 matrix. *Commu. ACM 4,* 4, 168.

STERN, A. AND DESBRUN, M. 2006. Discrete geometric mechanics for variational time integrators. In *ACM SIGGRAPH Courses*. ACM, New York, 75–80.

SULEJMANPAŠIĆ, A. AND POPOVIĆ, J. 2005. Adaptation of performed ballistic motion. *ACM Trans. Graph. 24,* 1, 165–179.

TERAN, J., BLEMKER, S., HING, V. N. T., AND FEDKIW, R. 2003. Finite volume methods for the simulation of skeletal muscle. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA'03)*. Eurographics Association, 68–74.

TERAN, J., SIFAKIS, E., IRVING, G., AND FEDKIW, R. 2005. Robust quasistatic finite elements and flesh simulation. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA'05)*. ACM Press, New York, 181–190.

TERZOPOULOS, D., PLATT, J., BARR, A., AND FLEISCHER, K. 1987. Elastically deformable models. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH'87)*. ACM, New York, 205–214.

TERZOPOULOS, D. AND WITKIN, A. 1988. Physically based models with rigid and deformable components. *IEEE Comput. Graphi. Appli. 8,* 6, 41–51.

TURNER, R. AND THALMANN, D. 1993. The elastic surface layer model for animated character construction. In *Proceedings of Computer Graphics International Conference*. Springer-Verlag, 399–412.

VAN DE PANNE, M. AND LAMOURET, A. 1995. Guided optimization for balanced locomotion. In *Proceedings of the Computer Animation and Simulation*. D. Terzopoulos and D. Thalmann, Eds., Springer-Verlag, 165–177.

WOJTAN, C. AND TURK, G. 2008. Fast viscoelastic behavior with thin features. *ACM Trans. Graph. 27,* 3.

ZHANG, X. AND KIM, Y. J. 2007. Interactive collision detection for deformable models using streaming AABBs. *IEEE Trans. Visual. Comput. Graph. 13,* 2, 318–329.

ZORDAN, V. B. AND HODGINS, J. K. 2002. Motion capture-driven simulations that hit and react. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA'02)*. ACM, New York, 89–96.

Received January 2010; revised January 2011; accepted April 2011