Example-based Dynamic Skinning in Real Time

Xiaohan Shi* Kun Zhou† Yiying Tong‡ Mathieu Desbrun§ Hujun Bao* Baining Guo†

*Zhejiang University †Microsoft Research Asia ‡Michigan State University §Caltech

Abstract

In this paper we present an approach to enrich skeleton-driven animations with physically-based secondary deformation in real time. To achieve this goal, we propose a novel, surface-based deformable model that can interactively emulate the dynamics of both low- and high-frequency volumetric effects. Given a surface mesh and a few sample sequences of its physical behavior, a set of motion parameters of the material are learned during an off-line preprocessing step. The deformable model is then applicable to any given skeleton-driven animation of the surface mesh. Additionally, our dynamic skinning technique can be entirely implemented on GPUs and executed with great efficiency. Thus, with minimal changes to the conventional graphics pipeline, our approach can drastically enhance the visual experience of skeleton-driven animations by adding secondary deformation in real time.

Keywords: skeleton-driven mesh deformation, secondary motion, physically-based animation, finite element method

1 Introduction

Skeleton-driven animation is widely used in the computer graphics industry, especially for interactive applications such as video games. However, such animations are often devoid of intricate physical motion details (*i.e.*, secondary deformation [O'Brien et al. 2000]) due to the small number of degrees of freedom (DoFs) in typical skeletons and the time-consuming manual work required to add physically-realistic secondary motions. With physically-based simulators, fine physical details up to the spatial and temporal resolution can be simulated. Unfortunately, simulating high frequency motions on top of an arbitrary skeleton-driven animation is rarely achievable in real time. In this paper we present an approach for adding physically-realistic secondary deformation to skeleton-driven animations at real-time rates.

To this end, we propose a simplified deformable model which is surface-based, interactive, and capable of exhibiting both low- and high-frequency volumetric effects. This interactive model contains only surface vertices—no interior vertices are necessary. It is capable of generating low-frequency phenomena such as muscle bulging, as well as high-frequency vibrations like the jiggling of fat tissues. While only surface vertices are required, the model mimics much of the behavior of a real volumetric material.

* This work was done while Xiaohan Shi was an intern at MSRA.
† E-mail: kunzhou@acm.org, bainguo@microsoft.com
‡ E-mail: ytong@msu.edu
§ E-mail: mathieu@caltech.edu

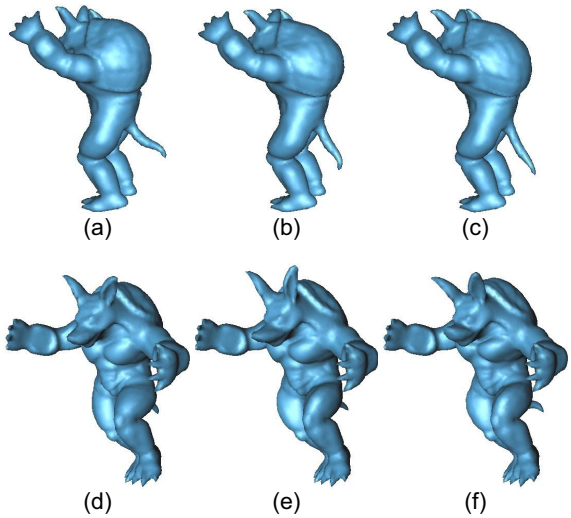

Figure 1: *Given the skeleton-driven animation of an Armadillo jumping (a), we fit the parameters of our model based on a physical simulation (b), and reproduce the secondary deformation (c). Given another animation of an Armadillo running (d), our model adds secondary deformation in real time (e). Note that even though* physical accuracy *is not our primary concern, our result (e) bears a strong resemblance to the actual physical simulation (f). See the companion video for a live demo.*

Technically, the proposed method is a surface-based approximation to a linear elasticity simulation. The skeleton animation drives the motion of vertices, while elastic forces from both the surface neighbors and the interior of the mesh as well as damping are taken into account. Since there are no interior vertices, elastic forces from the interior of the mesh are approximated by adding length-maintaining spokes between the surface vertices and the corresponding bones. The material properties of the mesh are defined by a set of per-surface-vertex parameters representing the strength of the above forces (*i.e.*, neighborhood stiffness, *etc.*). We propose a fitting algorithm to automatically learn these material parameters from user-provided examples of skeleton-driven animations, as an alternative/supplement to tuning the parameters manually. The examples can be acquired by physically-based simulation. Notice that the problem we are addressing here is challenging due to the highly nonlinear effects of (volumetric) elasticity on the surface mesh. We resolve this issue through an iterative optimization approach.

After the fitting stage, our model is ready to be applied to any given skeleton-driven animation of the surface mesh. More precisely, the model takes a skeleton-driven animation as input and determines the motion of the mesh using the set of fitted parameters and local (1-ring) vertex coordinates information from consecutive frames. Since only local information is used at runtime, commodity graphics hardware can be used to exploit vertex-level parallelism. This produces real-time performance on moderately large meshes, which is crucial for interactive applications like video games. Implemented as an additional rendering stage, secondary deformation can thus be added to skeleton-driven animations in real time, with minimal changes to the conventional graphics pipeline.

1.1 Related Work

Our work is closely related to skeleton-driven deformation techniques, elastic deformable models and parameter fitting algorithms. We only cover the most relevant references below, as the literature covering these topics is vast.

Skeleton-driven deformation techniques are extensively used to drive realistic animated characters. The most pervasive technique among them is linear blend skinning, also known as skeleton subspace deformation (SSD) (*i.e.*, [Magnenat-Thalmann et al. 1988] and several variants), which was later implemented efficiently on modern graphics hardware in the form of matrix-palette skinning [Lee 2006]. Significant improvements [Lewis et al. 2000; Mohr and Gleicher 2003] are proposed to address well-known artifacts like collapsing joints. Recently, a rotational regression model was proposed to capture common skinning deformation such as muscle bulging, twisting, and challenging regions such as the shoulders [Wang et al. 2007]. Shi *et al.* [2007] propose to optimize bone transformation and skin-weight with surface-detail-preserving inverse-kinematics constraints during deformation. All these techniques focus on the primary deformation of the surface mesh, but do not address secondary deformation (i.e., small idiosyncracies or detailed motions) despite its significant impact on realism [von Funck et al. 2007]. Our method adds secondary deformation to skeleton-driven animations generated by these techniques.

Volumetric deformable models can be incorporated with skeleton-driven animations via first constructing (possibly coarse) volumetric elements inside/around the surface mesh (using volumetric meshing algorithms, *i.e.*, the red-green strategies [Molino et al. 2003], variational tetrahedral meshing [Alliez et al. 2005], *etc.*), then simulating continuum elasticity in the volume, and finally reconstructing the (finely detailed) surface mesh from the volume elements through interpolation. Methods in this category include (but are not limited to) physically-based approaches such as the basic finite element method [Capell et al. 2002; Müller and Gross 2004], modal analysis [Pentland and Williams 1989; Hauser et al. 2003], mass-spring systems [Bourguignon and Cani 2000; Teschner et al. 2004], and non-physical approaches such as Fast Lattice Shape Matching [Rivers and James 2007]. Volumetric deformable models usually achieve high-quality simulation results, at the cost of having to simulate all volumetric nodes during online computation. Compared to existing volumetric approaches, the advantages of our method include: (1) efficiency, as only surface nodes are involved, a more finely detailed surface can be simulated in real time; (2) compatibility with the existing skinning framework and the traditional graphics hardware pipeline.

Surface-based deformable models drastically increase efficiency by reducing the DoFs to only surface nodes. Our deformable model falls into this category. ArtDefo [James and Pai 1999] introduced the boundary element method for static linear elasticity into graphics, and achieved through quasi-static simulation one of the earliest interactive-rate elastic simulations. Precomputed multizone elastokinematic models were introduced to simulate multibody kinematic systems which include elastostatic deformation [James and Pai 2002b]. To extend to real dynamical simulation, DyRT [James and Pai 2002a], based on modal analysis, discards all interior nodes after the preprocessing stage and excites the low-frequency modes by rigid body motions. Since the algorithm was implemented on GPUs and the number of modes in use was limited by hardware, very few (*e.g.*, five) low-frequency modes were simulated, and thus high frequency dynamics (*e.g.*, the jiggling of fat tissues) was not captured. Wilhelms [1995] modeled and animated animals using simulated individual bones and muscles, soft tissues, and skin. Larboulette *et al.* [2005] added local dynamic effects to classical animations of characters by specifying flesh elements. The use of muscles, soft tissues and flesh elements makes it hard to fit the skinning framework. Von Funck *et al.* [2007] added elastic secondary deformation to a given primary deformation by the simulation of a low number of user-placed mass-spring sets. To simulate high-frequency dynamics realistically, many mass-spring sets need to be placed and tuned, requiring extensive manual work. Bergou *et al.* [2007] introduced an animation method that uses computational physics to generate fine motion details while constraining the general motion to still follow a coarse (user-defined or precomputed) simulation. Their shell model is high resolution and can produce excellent surface detail. Since full-blown physics equations are used to generate high-quality final results, it does not allow interactive rates.

Parameter fitting algorithms learn model parameters from user-provided examples. Mesh-based inverse kinematics can learn a space of natural deformations from example meshes [Sumner et al. 2005]. Wang *et al.* propose learning from examples the parameters of their rotational regression model [Wang et al. 2007]. These methods fit parameters from static poses, while our fitting algorithm learns the dynamical properties of materials from animation sequences, rendering our task more challenging as the problem becomes highly non-linear. Bianchi *et al.* [2004] proposed simultaneous identification of the topology and stiffness of mass-spring models based on finite element method (FEM) reference deformations. Our deformable model inherits the topology of the user-provided surface mesh to best fit the traditional graphics hardware pipeline. Bhat *et al.* [2003] estimated the parameters of a cloth simulation from a video sequence of a real fabric. Our approach instead learns parameters from a volumetric elastic material.

1.2 Our Contributions

We enhance purely geometric, skeleton-driven mesh animations with physically-realistic secondary motions. As the coarse motion we start from is generally not physical to begin with, we do not pursue physical accuracy. To improve efficiency for interactive applications like video games, we use a surface-based deformable model trained by volumetric models (or example sequences) to further reduce computational complexity. We can also extend our method to incorporate better control of the low-frequency accuracy as we will discuss briefly in the final section. Our main contributions include:

- ◇ An interactive deformable model which is surface-based, yet capable of exhibiting natural volumetric behavior.
- ◇ An iterative fitting algorithm that overcomes the high nonlinearity of learning the (locally optimal) motion parameters.
- ◇ A fully GPU-based implementation of the on-the-fly dynamic skinning stage.

In Section 2, we present the surface-based deformable model whose simulation is parallelizable. Section 3 introduces an iterative fitting algorithm to automatically generate the parameters used in the deformable model. The on-the-fly GPU-based dynamic skinning is briefly discussed in Section 4. Finally, we conclude with a discussion of our results and possible extensions.

2 Deformable Model

To provide some background knowledge for our deformable model, we give a brief overview of FEM-style linear elasticity simulation. We list all the degrees of freedom in a single time-varying vector $\mathbf{x}$, seen as a point in a high-dimensional configuration space parameterized by a generalized coordinate system. The equations of motion (Newton's second law, or equivalently, the Euler-Lagrange

equations) for $\mathbf{x}(t)$ are differential equations describing the dynamical behavior of the system:

$$\mathbf{M}\ddot{\mathbf{x}} + \mathbf{C}\dot{\mathbf{x}} + \mathbf{K}(\mathbf{x} - \mathbf{x}_0) = \mathbf{f}_{ext}, \tag{1}$$

where $\mathbf{M}$ is the mass matrix, $\mathbf{C}$ is the damping coefficient matrix, $\mathbf{K}$ is the stiffness matrix, $\mathbf{x}_0$ is the rest pose, and $\mathbf{f}_{ext}$ is the generalized external force. Small displacements are usually assumed for linear models to work, while simple generalizations to large deformation through corotational frames exist, see *e.g.* [Müller and Gross 2004].

In our real-time deformable model, we simulate the deformable object using an elastic thin shell supported by elastic spokes emanating from the bones in the skeleton. As we target secondary motions caused by the primary motion, we add, on top of the elastic forces, a restoration force to each vertex towards its position in the primary motion $\hat{\mathbf{x}}$:

$$\mathbf{M}\ddot{\mathbf{x}} - \mathbf{F}(\mathbf{x}, \mathbf{x}_0, \dot{\mathbf{x}}, \hat{\mathbf{x}}) = \mathbf{f}_{ext}, \tag{2}$$

where $\mathbf{F}$ is the *total* internal force. This is the governing equation we use for our surface-based model. Note that the volumetric elastic force acting on surface nodes consists of two parts: those coming from surface neighbors, and those coming from interior neighbors. Since our model is surface-based, the degrees of freedom representing interior neighbors are not present. We detail how we approximate the elastic force from interior neighbors using spokes in Section 2.1.

In FEM, the state of the object is usually evolved in time using implicit integration when large time steps are used for improved stability. We choose, instead, to use a simple, GPU-friendly explicit time integration since we seek parallelism of the process instead of high accuracy. Forces will be carefully implemented so that they do not overshoot when our explicit integration scheme is applied.

We call the original surface mesh the *rest pose*, and the skeleton-driven poses the *goal poses* (we employ SSD [Magnenat-Thalmann et al. 1988] to generate goal poses in our paper). For the i-th vertex the surface mesh, we denote its position in the rest pose by $\mathbf{x}_i^0$, its position in goal poses (*i.e.*, the *goal positions*) by $\hat{\mathbf{x}}_i(t)$, its dynamic position by $\mathbf{x}_i(t)$ and its velocity by $\mathbf{v}_i(t)$, where $t \in [0,T]$ is time and T is the duration of the animation. In the following sections, we first elaborate on each of the simulation forces, then on time integration.

2.1 Per-Vertex Forces

For each vertex of the mesh, four forces besides external force are evaluated and applied at each frame of the animation. They are: restoration force from goal position $\mathbf{G}_i(t)$, elastic force from surface neighbors $\mathbf{L}_i(t)$, volumetric elastic force $\mathbf{B}_i(t)$, and damping $\mathbf{D}_i(t)$ (see Figure 2). When summed up to get the total force, each force is scaled by a factor denoting the strength of the force (*e.g.*, neighborhood stiffness, strength of the spoke attaching that vertex to the bone, *etc.*) which best approximates the desired volumetric behavior. In other words, these factors are the material properties of the surface mesh. We follow the common practice of *mass lumping* in elasticity simulation. Thus, without loss of generality, we can use the terms force and acceleration interchangeably, assuming the mass associated to the node is taken into account.

$$\mathbf{F}_i(\Pi_i, t) = \alpha_i \mathbf{G}_i(t) + \beta_i \mathbf{L}_i(t) + \gamma_i \mathbf{B}_i(t) + \lambda_i \mathbf{D}_i(t) + \mathbf{F}_{ext}(t). \tag{3}$$

These per-vertex parameters $\Pi = \{\alpha_i, \beta_i, \gamma_i, \lambda_i\}_{i \in vertices}$ allow the model to achieve different dynamical behaviors. The user may tune the parameters via a painting or curve-based editing interface, or use the fitting algorithm described in Section 3 to learn parameters from example animations.

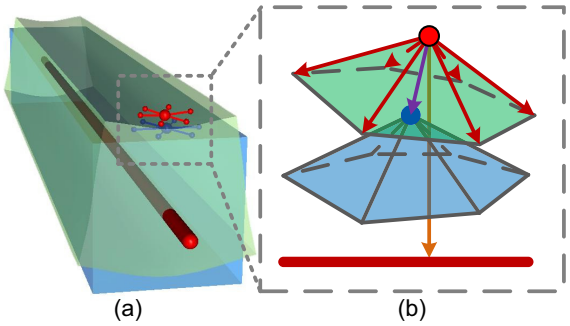

Figure 2: *(a) The dynamics of our model (green) is guided by the skeleton and the* goal pose *(blue). (b) Per-vertex Forces: A vertex (red dot) experiences four forces besides external force: restoration force (purple arrow) to goal position (blue dot), elastic force (red arrow) from surface neighbor (green umbrella), elastic force (brown arrow) from bone (red segment) and damping force (not drawn here).*

Restoration Force from Goal Position Each vertex is pushed towards its goal position so that the simulated mesh can achieve equilibrium near the goal pose and converge if the goal pose stops moving.

$$\mathbf{G}_i(t) = \frac{\hat{\mathbf{x}}_i(t) - \mathbf{x}_i(t)}{h^2}. \tag{4}$$

This term follows the formulation of [Müller et al. 2005], as it is the largest possible acceleration that does not produce overshoot over one time step h. The next two terms are also treated in the same fashion.

Elastic Force from Surface Neighbors In FEM, adjacent nodes exert elastic forces on each other when the element between them changes shape, inducing a locally non-zero strain tensor. We mimic this effect by maintaining the surface details: the position which best preserves the Laplacian coordinates is calculated, and the vertex is moved towards this position.

More precisely, let us consider a single vertex i and its local neighborhood $nbr(i)$. First, the optimal rotation R from the rest pose (of the neighborhood) to the current pose is calculated via a polar decomposition of

$$\mathbf{A} = \sum_{j \in nbr(i)} \left(\mathbf{x}_j(t) - \mathbf{x}_i(t)\right) \left(\mathbf{x}_j^0 - \mathbf{x}_i^0\right)^T. \tag{5}$$

In terms of the singular value decomposition of $\mathbf{A}$, $\mathbf{A} = \mathbf{UDV}$, one has $\mathbf{R} = \mathbf{UV}$ [Haralick et al. 1989]. Then, the position which best preserves the Laplacian coordinates is calculated as

$$\mathbf{c}_i(t) = \frac{1}{|nbr(i)|} \sum_{j \in nbr(i)} \left(\mathbf{R}(\mathbf{x}_i^0 - \mathbf{x}_j^0) + \mathbf{x}_j(t)\right). \tag{6}$$

Thus the surface detail preserving force is

$$\mathbf{L}_i(t) = \frac{\mathbf{c}_i(t) - \mathbf{x}_i(t)}{h^2}. \tag{7}$$

Here we choose the graph Laplacian over other forms of Laplacian operators (*e.g.*, cotangent form) mainly due to its simple uniform weights, which means no extra texture storage or tex-read operations needed for GPU implementation. The same reasoning also leads us to use the graph Laplacian in the damping term discussed below.

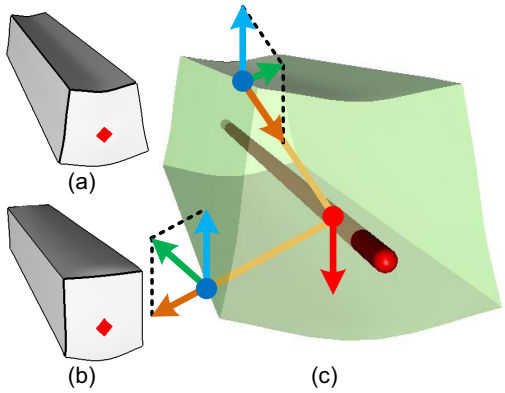

Figure 3: *Volumetric Behavior. The bone (red) accelerates downward, causing the rubber tube to deform. With the help of the spokes (yellow in (c)), our model (a) exhibits volumetric behavior, in contrast to the result generated without spokes (b). In the illustration (c), the D'Alembert force (cyan) resulting from the acceleration of the bone and the elastic force from the bone (brown) together drive the two vertices (blue) to accelerate in their respective directions (green). Without the support from the bone, such volumetric behavior would be hard to emulate.*

Volumetric Elastic Force In FEM, surface vertices experience elastic forces not only from surface neighbors, but also from the inside of the material. Since we reduce our model to be purely boundary-based for efficiency, the positions and momenta usually stored at interior vertices are unavailable. It is impossible in theory to use only a thin shell model to simulate volumetric elasticity with full accuracy. However, for models with a skeleton, we observe that visually one of the most important volumetric behaviors not already captured by the restoration force to goal positions is the tendency to restore the distance between the boundary vertex and the bone supporting it. Modeling this behavior alone obviously omits the subtle interaction between boundary vertices and interior vertices, and the interaction between nonadjacent boundary vertices through internal elements. However, these detailed effects are not compatible with interactive-rate requirements. Nevertheless, it is conceivable to extend our basic model to include a proper treatment of the low-frequency modes if the application requires so. We delay the discussion of this extension to Section 6.

For the sake of performance, we take a simplistic model where the surface vertices are linked directly to the bones by pseudo-springs that can slide along the bones. (Adding a stick of specific mass and moment of inertia would be more desirable, but we choose to use this simple model since the restoration force to goal position has partially taken the other volumetric effects into account, and in practice it serves its purpose well.) During the simulation, a force along the spoke tries to maintain the length of the spring through the following term:

$$\mathbf{B}_i(t) = \frac{\left(|\mathbf{x}_{ib}^0|/|\mathbf{x}_{ib}(t)| - 1\right)\mathbf{x}_{ib}(t)}{h^2}, \tag{8}$$

where $\mathbf{x}_{ib}$ is the difference between a vertex and its projection on the bone. For joint regions, a surface vertex is linked to several bones. In this case, the forces from each bone are averaged using the usual skin weights. With this term, our model is capable of reproducing the visually important part of the volumetric behavior, *i.e.*, it can mimic the resistance to local volume change (see Figure 3).

Damping With the commonly used Rayleigh damping, a vertex experiences resistance when its velocity differs from its neighbors. To approximate the effect of such damping forces, the velocity of a vertex is smoothed based on its 1-ring neighbors, again using the graph Laplacian:

$$\mathbf{D}_i(t) = \frac{1}{h\,|nbr(i)|}\sum_{j\in nbr(i)} \mathbf{v}_j(t) - \mathbf{v}_i(t). \tag{9}$$

Ranges of the Parameters To simulate different material properties, the terms above are scaled by corresponding parameters before being employed in the dynamics described in the next section (see Figure 4). Since we normalized the largest possible forces without causing instability to be 1 (the parameters actually correspond to F/m instead of F), the natural limits to these parameters are $[0,1]$, with the exception of λ which should be within $[0,0.5]$ to ensure the stability of the system.

2.2 Dynamical Simulation

The dynamical simulation of our model is straightforward. A symplectic Euler integration scheme similar to [Müller et al. 2005] is used for its simplicity and efficiency. At each time step, the velocity of the vertices are first updated explicitly based on the above terms, then the positions are updated explicitly using the latest velocity:

$$\mathbf{v}_i(t+h) = \mathbf{v}_i(t) + h\mathbf{F}_i(\Pi, t), \tag{10}$$

$$\mathbf{x}_i(t+h) = \mathbf{x}_i(t) + h\mathbf{v}_i(t+h). \tag{11}$$

3 Iterative Fitting Algorithm

The model we described above contains four parameters for each vertex. Although manual tuning via painting or a spline-based editing interface can be applied, the user may have already obtained satisfactory examples for the behavior of the given mesh. Thus we propose an iterative fitting algorithm to learn the material properties directly from example animations.

The input of the fitting algorithm includes a surface mesh with a skeleton structure, and two skeleton-driven animations - one with physical details (the *example pose*) and one without (the *goal pose*). The fitting process optimizes the set of per-vertex parameters Π so that when driven by the goal pose, the surface mesh follows the example pose as closely as possible:

$$\Pi = \arg\min_{\Pi} E(\Pi). \tag{12}$$

We measure the similarity of the two animations by the difference between the positions of vertices computed as:

$$E(\Pi) = \sum_t \sum_i \|\mathbf{x}_i(t) - \mathbf{p}_i(t)\|^2, \tag{13}$$

where $\mathbf{p}_i(t)$ is the position of the i-th vertex in the example animation (assuming regular time steps). The above measure can also

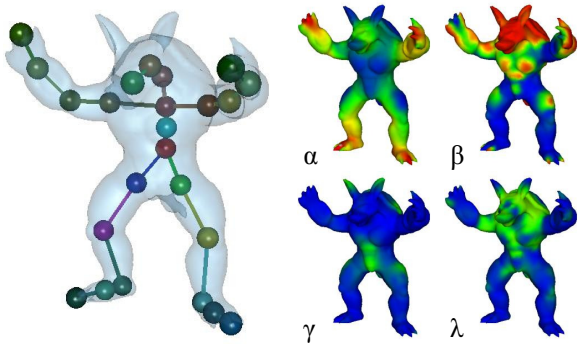

Figure 4: *Armadillo's skeleton and an example of the four per-vertex parameters. Blue and red correspond to 0 and 1, respectively.*

be weighted to adapt to irregularly sampled meshes, or to stress certain segments containing particularly desirable behaviors in the sequences.

Given such a non-linear optimization problem, gradient descent or Newton's method [Avriel 2003] could be applied. However, such algorithms require the gradient (and possibly the Hessian) of the function to be evaluated at each iteration, which is both costly and difficult to compute even with the adjoint method [McNamara et al. 2004] (as the adjoint matrices involve taking derivatives of the forces based on positions of the neighbors and the polar decomposition).

We propose instead a customized iterative optimization procedure. Suppose we have an initialization of the parameters Π^0. Let us assume we are now in the k-th iteration ($k \geq 0$) and we already have the parameters Π^k. The simulated sequence can be constructed according to Π^k and the dynamics. Denote its position and velocity by $\mathbf{x}_i^k(t)$ and $\mathbf{v}_i^k(t)$ respectively. We obtain the parameters of the next iteration Π^{k+1} via:

$$\begin{aligned} \Pi^{k+1} &= \arg\min_{\Pi} E^k(\Pi) \\ \text{s.t. } &\alpha,\beta,\gamma \in [0,1] \text{ and } \lambda \in [0,0.5], \end{aligned} \tag{14}$$

where the evaluation of $E^k(\Pi)$ will be explained below. After Π^{k+1} is solved, the algorithm advances to the $(k+1)$-th iteration, and the dynamics is simulated again using Π^{k+1} to obtain the sequence $\mathbf{x}_i^{k+1}(t)$ and $\mathbf{v}_i^{k+1}(t)$. The above iterative procedure loops until convergence.

We evaluate $E^k(\Pi)$ in Eq. (14) by expanding Eq. (13) using Eq. (10) and (11):

$$\begin{aligned} E^k(\Pi) &= \sum_t \sum_i \|\mathbf{x}_i(t+h) - \mathbf{p}_i(t+h)\|^2 \\ &= \sum_t \sum_i \left\| \mathbf{x}_i^k(t) + h\mathbf{v}_i^k(t) + h^2 \mathbf{F}_i^k(\Pi_i, t) - \mathbf{p}_i(t+h) \right\|^2, \end{aligned} \tag{15}$$

where $\mathbf{F}_i^k(\Pi_i, t)$ is the force term generated using $\mathbf{x}_i^k(t)$, $\mathbf{v}_i^k(t)$ and Π. Inspired by the expectation-maximization (EM) algorithm [Dempster et al. 1977], we consider all positions and velocities constant in the above equation (basically to treat each step of the motion as a separate short sequence). Consequently E^k depends only on $\mathbf{F}$, thus E^k can be regarded as a function of variable Π only.

Further, notice that the terms involving the parameters of any individual vertex use *only* its neighbors' positions and velocities (which are considered constant). The energy minimization problem, Eq. (14), can thus be split into a per-vertex quadratic programming (QP) problem, which can be solved efficiently:

$$\begin{aligned} \Pi_i^{k+1} = &\arg\min_{\Pi_i} \sum_t \left\| \mathbf{x}_i^k(t) + h\mathbf{v}_i^k(t) + h^2 \mathbf{F}_i^k(\Pi_i, t) - \mathbf{p}_i(t+h) \right\|^2 \\ &\text{s.t. } \alpha_i,\beta_i,\gamma_i \in [0,1] \text{ and } \lambda_i \in [0,0.5]. \end{aligned} \tag{16}$$

We point out here that we have no formal proof of global convergence, as we designed this scheme for our specific problem. However, our numerical tests always converged to a same local minimum from all the initial conditions we tried. An example of the fitted parameters is shown in Figure 4. Note that even if the physical material is uniform, the surface-based parameters learned from volumetric material can still be non-uniform. An important issue is that the parameter values of consecutive iterations (say, Π^k and Π^{k+1}) might differ greatly. Direct update of the value of Π from Π^k to Π^{k+1} may cause the optimization process to diverge. In order to make the process more stable, a step-ratio coefficient μ can be defined so that whenever the value of a new iteration is calculated, the actual solution is moved only partially towards the new value.

$k = 0$
repeat
  // SIMULATE THE SEQUENCE
  Simulate the dynamics using Π^k to get $\mathbf{x}_i^k(t)$ and $\mathbf{v}_i^k(t)$

  // PERFORM ONE STEP OF THE OPTIMIZATION
  for $i = 1$ to N **do**
    // *For each vertex*
    Solve Eq. (16) via QP, and obtain Π_i^{k+1}

  // ADVANCE TO THE NEXT ITERATION
  $\Pi^{k+1} = (1-\mu)\Pi^k + \mu\Pi^{k+1}$
  $k = k+1$
until $|E^k - E^{k-1}|/E^0 < \varepsilon$

Figure 5: *Pseudocode of our fitting algorithm.*

Implementation Details From our numerical tests, a step-ratio μ within the range of $[1/T, 0.1]$ is usually sufficient for stability, where $T > 10$ is the duration of the given sequences.

Having no prior knowledge of the optimal solution, we use $\alpha = 1, \beta = \gamma = \lambda = 0$ as an educated guess for the initial values of all vertices. The initial sequence thus follows the goal poses, which is reasonable because the goal poses match the example poses up to secondary deformations.

Discussion In the definition of the distance between two sequences, it is plausible to include the difference between the velocities of the two animations. This corresponds to a Sobolev norm, with the two sequences regarded as two functions in time. In our tests, this norm only provided negligible improvement.

From Eq. (10) and (11), we can see that $\mathbf{x}_i(t+h)$ depends recursively on $\mathbf{x}_i(t)$, and each recursion involves increasing the orders of the terms containing Π by one (actually, nonlinear dependence on Π is also involved). Thus, $\mathbf{x}_i(t+h)$ is more complicated than a high degree polynomial of Π. This is why we resort to sequential quadratic programming, so that each step can be dealt with as a linear problem. Another benefit is that our algorithm is easy to implement, since the calculation of the gradient is avoided.

4 Dynamic Skinning on GPUs

After the per-vertex parameters are obtained, we are ready to simulate the additional physically-based secondary motion on any given skeleton-driven animation. The algorithm is implemented as an additional pass in the graphics pipeline before the rendering pass. Indexed matrix palette skinning [Lindholm et al. 2001] is implemented in this pass. Instead of rendering the vertices, we use them as the goal pose. After the calculation of our algorithm, a typical rendering pass is invoked to render the mesh onto the screen.

Our GPU implementation of the above algorithm uses the NVIDIA CUDA framework [NVIDIA 2007]. CUDA provides a general-purpose C language interface for GPU programming, in which the algorithm is straightforwardly implemented in one CUDA kernel. All our data are organized as linear lists, and stored in CUDA global memory. Skeleton and one-ring neighborhood information are accessed via linear texture fetches while other per-vertex data are directly accessed using global memory operations.

model	# vertices	# bones	frame time
Toy	494	3	0.50ms
Tube	4430	2	0.57ms
Armadillo	6969	24	0.67ms
Armadillo	25001	19	1.64ms

Table 1: *Performance statistics, including size of the meshes and the time spent per frame for the calculation of secondary motion. All timings were measured on a 3.7GHz Intel Xeon workstation with 3GB RAM and an nVidia GeForce 8800GTX graphics card.*

The implementation described above can simulate dynamics with great efficiency. In our test, a mesh with 25001 vertices can be simulated at the speed of 610 frames-per-second. This means that we can add secondary deformation to animations and interactive applications with negligible additional time. See Table 1 for detailed statistics.

Discussion The implementation would be even faster if the per-vertex polar decomposition were omitted and the rotation calculated from the bone matrices. Considering that the implementation with polar decomposition is already faster than real-time on moderately large meshes, we never resorted to this approximation.

5 Applications and Results

A direct application of our approach is enriching skeleton-driven animations with secondary effects. In the accompanying video, the Armadillo model was trained using a "jump" animation simulated with non-uniform material. Note that the vertex parameters (see Figure 4) reflect the non-uniform material of the example model. After the model is trained, it is applied to another "run-and-jump" animation, and generates plausible results (see Figure 1). It also suggests that a simple animation like "jump" is enough to properly train our model.

Our per-vertex model can express high-frequency deformations, both in animation and user interaction (see Figure 6). On the contrary, DyRT [James and Pai 2002a] simulates only low-frequency modes obtained via modal analysis, thus high-frequency vibrations are not present. The reason for DyRT not using more high-frequency modes is that the number of modes in use is limited by graphics hardware. In our model, a fixed number of textures is used, and thus we do not suffer from this issue. With the help of spokes, our model can exhibit volumetric behavior (see Figure 3). We also show in the video that the per-vertex parameters *do* emulate intrinsic material properties well. The parameters of the bar model used are fitted to the physical simulations of the elastic bar induced by three different rigid motions of a bone segment (translation, axial rotation and lateral rotation), and the resulting model reproduces the sequences quite well. Figure 1 shows that fitted models can also be re-used on other animations and produce desirable results.

In our tests, the traditional SSD [Magnenat-Thalmann et al. 1988] is employed to generate goal poses. Interactive Virtual Material [Müller and Gross 2004] is used to generate example animation sequences.

Dynamic Skinning in Interactive Environments Our algorithm can enhance various interactive experiences. In interactive manipulation algorithms such as Mesh Puppetry [Shi et al. 2007], the user manipulates a mesh, and its skeleton deforms accordingly. Our algorithm can then add secondary deformation to the mesh according to the deformation of both the mesh and the skeleton in real time.

In computer games, characters may not only be manipulated directly through user interaction (*e.g.*, change of gaze direction by mouse click), but also can be partially influenced by the virtual environment as an indirect result of user interaction (*e.g.*, being hit by a ball with random momentum at an unpredictable contact position during a dodgeball game). Various articulated rigid body simulation algorithms [Hadap and Kokkevis 2005] can be used to simulate the primary motion of the character, while our algorithm supplements it with secondary deformation to enhance realism, thus improving the resulting gaming experience.

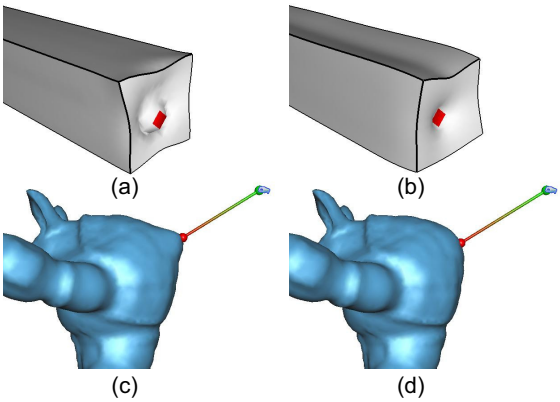

Figure 6: *Our per-vertex model can express high-frequency vibrations. When driven by a skeleton animation, our model is capable of generating wrinkles (a), while DyRT only generates low-frequency deformation (b). When dragged by the user, a soft material deforms locally using our model (c), while it deforms globally using DyRT (d), as low temporal frequencies are generally associated to low spatial frequencies. Here for DyRT, 15 modes are used in (b), and 5 in (d).*

Limitations Since we aim to enrich skeleton-driven animations and base our formulation on the existence of a skeleton, our approach can obviously not be applied to arbitrary mesh sequences. Even for skeleton-based animations, our approach will most often not precisely reproduce example animations. Additionally, our approach can lead to issues due to explicit integration when applied to very stiff objects. In such a case, a damping term to absolute velocity can be applied and be trained as another parameter. Our fitting algorithm also inherits the limitation of all example-based approaches. However, in our tests, a simple sequence like the "jump" animation provides enough material information to reproduce the desired high-frequency effects. Finally, overfitting can occur during the fitting procedure. Additional techniques such as cross-validation can be applied to circumvent this issue.

6 Conclusion and Future Work

We have presented a novel example-based approach to motion detail enrichment in real time. A surface-based deformable model that can interactively emulate both low- and high-frequency volumetric dynamical effects is introduced. To generate the set of per-vertex parameters used in our model, an iterative fitting algorithm is proposed to handle the high nonlinearity of the optimization problem. The simplicity and parallelism of our model lead naturally to a fully GPU-based implementation that runs in real time on moderately large meshes. The direct applications include adding secondary deformation to existing animation sequences, as well as to animated characters in interactive environments such as video games.

For future work, we first wish to test our model on secondary effects captured by recent motion capture methods (*e.g.*, [Sand et al.

2003; Park and Hodgins 2006]). More realistic results could likely be achieved if our model were able to match such motion captured skin data. In addition, it would be beneficial to find a better measurement of motion errors and to compress the parameter space of our model under the assumption that the physical material varies smoothly over the volume (in the current model, we learn per-vertex parameters and make no assumptions about the smoothness of the physical material). We also plan to explore the possibility of replacing spokes with more elaborate models. For example, in the realm of linear elasticity, the mesh can be divided into parts and multi-zone elastokinematic models may be simulated using precomputed Green's functions according to skeleton motion, as done in [James and Pai 2002b]. To improve the accuracy of the low frequency motion, we can incorporate the low-frequency modes of DyRT into $\hat{\mathbf{x}}$. Furthermore, forces due to deviation from the goal position can be used to change the excitation level of the low-frequency modes through projection onto the chosen modes to provide better coupling between low- and high-frequency components—offering more physical accuracy while retaining the interactive rates, and providing more options to enhance visual appearance when sufficient computational power is available.

Acknowledgements

We would like to thank the anonymous reviewers for their helpful comments. Special thanks to Eric Stollnitz, Jianwei Han and Qiming Hou for their help during video production. Hujun Bao was partially supported by the 973 Program of China (No. 2002CB312102).

References

ALLIEZ, P., COHEN-STEINER, D., YVINEC, M., AND DESBRUN, M. 2005. Variational tetrahedral meshing. *ACM Trans. Graph. 24*, 3, 617–625.

AVRIEL, M. 2003. *Nonlinear programming: Analysis and Methods*. Dover Publications.

BERGOU, M., MATHUR, S., WARDETZKY, M., AND GRINSPUN, E. 2007. Tracks: toward directable thin shells. *ACM Trans. Graph. 26*, 3, 50.

BHAT, K. S., TWIGG, C. D., HODGINS, J. K., KHOSLA, P. K., POPOVIĆ, Z., AND SEITZ, S. M. 2003. Estimating cloth simulation parameters from video. In *Proceedings of SCA'03*, 37–51.

BIANCHI, G., SOLENTHALER, B., SZÉKELY, G., AND HARDERS, M. 2004. Simultaneous topology and stiffness identification for mass-spring models based on fem reference deformations. In *MICCAI (2)*, 293–301.

BOURGUIGNON, D., AND CANI, M.-P. 2000. Controlling anisotropy in mass-spring systems. In *Proceedings of Computer Animation and Simulation'00*, 113–123.

CAPELL, S., GREEN, S., CURLESS, B., DUCHAMP, T., AND POPOVIĆ, Z. 2002. Interactive skeleton-driven dynamic deformations. *ACM Trans. Graph. 21*, 3, 586–593.

DEMPSTER, A. P., LAIRD, N. M., AND RUBIN, D. B. 1977. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological) 39*, 1, 1–38.

HADAP, S., AND KOKKEVIS, V. 2005. Introduction to articulated rigid body dynamics. In *ACM SIGGRAPH 2005 Courses*, 1.

HARALICK, R., JOO, H., LEE, C., ZHUANG, X., VAIDYA, V., AND KIM, M. 1989. Pose estimation from corresponding point data. *IEEE Transactions on Systems, Man and Cybernetics 19*, 6, 1426–1446.

HAUSER, K. K., SHEN, C., AND O'BRIEN, J. F. 2003. Interactive deformation using modal analysis with constraints. In *Graphics Interface*, 247–256.

JAMES, D. L., AND PAI, D. K. 1999. Artdefo: accurate real time deformable objects. In *Proceedings of SIGGRAPH'99*, 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, 582–585.

JAMES, D. L., AND PAI, D. K. 2002. Real time simulation of multizone elastokinematic models. *Proceedings of ICRA'02 1*, 927–932.

LARBOULETTE, C., CANI, M.-P., AND ARNALDI, B. 2005. Dynamic skinning: adding real-time dynamic effects to an existing character animation. In *Proceedings of SCCG'05*, 87–93.

LEE, M. 2006. Seven ways to skin a mesh: Character skinning revisited for modern gpus. In *Proceedings of GameFest, Microsoft Game Technology Conference*.

LEWIS, J. P., CORDNER, M., AND FONG, N. 2000. Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation. In *Proceedings of SIGGRAPH'00*, 165–172.

LINDHOLM, E., KLIGARD, M. J., AND MORETON, H. 2001. A user-programmable vertex engine. In *Proceedings of SIGGRAPH'01*, 149–158.

MAGNENAT-THALMANN, N., LAPERRIÈRE, R., AND THALMANN, D. 1988. Joint-dependent local deformations for hand animation and object grasping. In *Proceedings of GI'88*, 26–33.

MCNAMARA, A., TREUILLE, A., POPOVIĆ, Z., AND STAM, J. 2004. Fluid control using the adjoint method. *ACM Trans. Graph. 23*, 3, 449–456.

MOHR, A., AND GLEICHER, M. 2003. Building efficient, accurate character skins from examples. *ACM Trans. Graph. 22*, 3, 562–568.

MOLINO, N., BRIDSON, R., TERAN, J., AND FEDKIW, R. 2003. A crystalline, red green strategy for meshing highly deformable objects with tetrahedra. In *IMR*, 103–114.

MÜLLER, M., AND GROSS, M. 2004. Interactive virtual materials. In *Proceedings of GI'04*, 239–246.

MÜLLER, M., HEIDELBERGER, B., TESCHNER, M., AND GROSS, M. 2005. Meshless deformations based on shape matching. *ACM Trans. Graph. 24*, 3, 471–478.

NVIDIA, 2007. CUDA homepage. http://developer.nvidia.com/object/cuda.html.

O'BRIEN, J. F., ZORDAN, V. B., AND HODGINS, J. K. 2000. Combining active and passive simulations for secondary motion. *IEEE Comput. Graph. Appl. 20*, 4, 86–96.

PARK, S. I., AND HODGINS, J. K. 2006. Capturing and animating skin deformation in human motion. *ACM Trans. Graph. 25*, 3, 881–889.

PENTLAND, A., AND WILLIAMS, J. 1989. Good vibrations: model dynamics for graphics and animation. In *Proceedings of SIGGRAPH'89*, 215–222.

RIVERS, A. R., AND JAMES, D. L. 2007. Fastlsm: fast lattice shape matching for robust real-time deformation. *ACM Trans. Graph. 26*, 3, 82.

SAND, P., MCMILLAN, L., AND POPOVIĆ, J. 2003. Continuous capture of skin deformation. *ACM Trans. Graph. 22*, 3, 578–586.

SHI, X., ZHOU, K., TONG, Y., DESBRUN, M., BAO, H., AND GUO, B. 2007. Mesh puppetry: cascading optimization of mesh deformation with inverse kinematics. *ACM Trans. Graph. 26*, 3, 81.

SUMNER, R. W., ZWICKER, M., GOTSMAN, C., AND POPOVIĆ, J. 2005. Mesh-based inverse kinematics. *ACM Trans. Graph. 24*, 3, 488–495.

TESCHNER, M., HEIDELBERGER, B., MULLER, M., AND GROSS, M. 2004. A versatile and robust model for geometrically complex deformable solids. In *Proceedings of CGI'04*, 312–319.

VON FUNCK, W., THEISEL, H., AND SEIDEL, H.-P. 2007. Elastic secondary deformations by vector field integration. In *Proceedings of SGP'07*, 99–108.

WANG, R. Y., PULLI, K., AND POPOVIĆ, J. 2007. Real-time enveloping with rotational regression. *ACM Trans. Graph. 26*, 3, 73.

WILHELMS, J. 1995. Modeling animals with bones, muscles, and skin. Technical report. University of California at Santa Cruz.