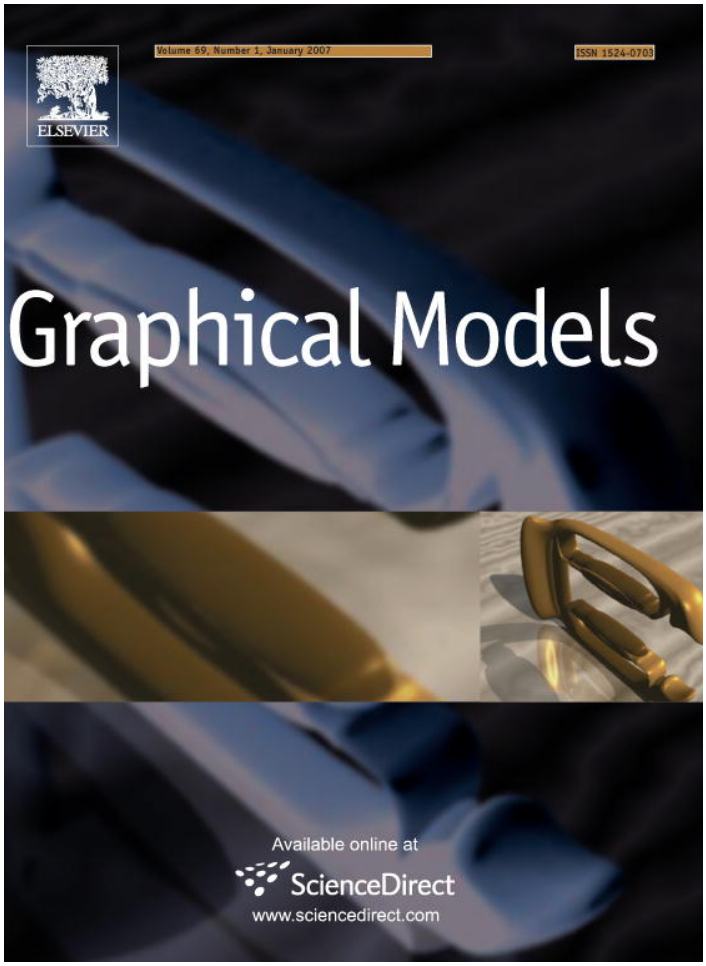

Available online at www.sciencedirect.com
ScienceDirect
Graphical Models 69 (2007) 71–87
Graphical Models
www.elsevier.com/locate/gmod

Physically based rigging for deformable characters

Steve Capell [a,*,1], Matthew Burkhart [b], Brian Curless [b], Tom Duchamp [b], Zoran Popović [b]

[a] *Havok, 4/2 Middleton Street, Suite 802, Kolkata 700071, India*
[b] *University of Washington, USA*

Received 23 February 2006; received in revised form 19 August 2006; accepted 4 September 2006

Abstract

In this paper, we introduce a framework for instrumenting (*rigging*) characters that are modeled as dynamic elastic bodies, so that their shapes can be controlled by an animator. Because the shape of such a character is determined by physical dynamics, the rigging system cannot simply dictate the shape as in traditional animation. For this reason, we introduce forces as the building blocks of rigging. Rigging forces guide the shape of the character, but are combined with other forces during simulation. Forces have other desirable features: they can be combined easily and simulated at any resolution, and since they are not tightly coupled with the surface geometry, they can be more easily transferred from one model to another. Our framework includes a new pose-dependent linearization scheme for elastic dynamics, which ensures a correspondence between forces and deformations, and at the same time produces plausible results at interactive speeds. We also introduce a novel method of handling collisions around creases.

Keywords: Character animation; Animation; Simulation; Rigging; Physically based animation

1. Introduction

Believable computer animation requires that virtual *characters* such as humans and animals be produced with a high degree of realism. Faces should express emotions through mouth and eyebrow movements, limbs should bend at joints, muscles should bulge when in use, and soft tissue such as fat should bounce and vibrate when the character walks.

To create such realism, the traditional animation pipeline requires that each character be *rigged*, a process that is analogous to setting up a puppet to be controlled by strings. After having been rigged, a character's shape can be controlled through a set of abstract parameters with meaningful names, like "lift left eyebrow" or "bend right knee." For each keyframe, instead of having to position each vertex of the surface mesh, the animator need only set the values of the control parameters. To include fleshy bounces and vibrations, the animator must either create them by

* Corresponding author. Tel.: +91 33 3042 1542.
E-mail address: steve.capell@havok.com (S. Capell).
[1] This work was done while the first author was at the University of Washington.

1524-0703/$ - see front matter © 2006 Elsevier Inc. All rights reserved.
doi:10.1016/j.gmod.2006.09.001

Glossary of Symbols

Ω	A domain in $\mathbb{R}^3$ representing the points in a body.
Γ	The boundary of Ω.
C	A cell complex used to parameterize a body.
K	An embedding of C in $\mathbb{R}^3$ that contains Ω.
$\mathbf{r}(u)$	A homeomorphism that maps C to K.
S	A set of edges from K that form the skeleton of the body.
$\mathbf{x}$	The Euclidean coordinates of a body at rest.
$\mathbf{d}(\mathbf{x},t)$	The displacement of a moving body.
$\mathbf{p}(\mathbf{x},t)$	The position of a moving body.
$\alpha(t)$	A vector of parameters that control the pose of the body.
$\beta(t)$	A vector of parameters that influence the shape of the body.
$\Theta(t)$	A vector of control parameters concatenating $\alpha(t)$ and $\beta(t)$.
$\mathbf{f}(\mathbf{x},\Theta)$	A force field acting on the body to influence its shape.
$\mathscr{B}$	A set of basis functions.
$\phi_i(\mathbf{x})$	A basis function from the set $\mathscr{B}$.
$\mathbf{q}_i(t)$	A generalized coordinate used to describe $\mathbf{d}(\mathbf{x},t)$.
$\mathbf{q}(t)$	A vector concatenating all $\mathbf{q}_i$ that do not lie on the skeleton.
$\mathbf{q}_S(t)$	A vector concatenating all $\mathbf{q}_i$ that lie on the skeleton.
$\mathbf{Q}_i(\Theta)$	A generalized force due to $\mathbf{f}$ acting on a body.
$\mathbf{Q}_i^e(t)$	A generalized force due to U.
$T(\dot{\mathbf{q}},\dot{\mathbf{q}}_S)$	The kinetic energy of a body.
$U(\mathbf{q},\mathbf{q}_S)$	The potential energy of a body.
$\mathbf{M}$	The generalized mass matrix associated with $\mathbf{q}$.
$\mathbf{N}$	The generalized mass matrix associated with $\mathbf{q}_S$.
$\mathbf{S}$	The stiffness matrix that results from linearizing the elastic force.
$\mathbf{R}_i(\alpha)$	An estimate of the rotation of the ith vertex, based on the pose.
$\mathbf{c}_i(\alpha)$	An estimate of the translation of the ith vertex, based on the pose.
$\mathbf{A}(\alpha)$	The constant matrix term in the linear static equilibrium equation for a body.
$\mathbf{b}(\alpha)$	The constant vector term in the linear static equilibrium equation for a body.
η_a	A vector of parameters that control the ath rig.
$\mathbf{f}_a(\mathbf{x},\eta_a)$	The force field due to the ath rig.
γ_a	A parameter from Θ that controls the ath rig.
$\tau_a(\gamma_a)$	A mapping from γ_a to η_a.
$\mathbf{h}$	A mapping from one body domain to another.

hand or tack a physical simulation onto the geometrically crafted deformations. In interactive settings, such as video games, these additional motions are not always possible to anticipate, necessitating some form of physical simulation.

Elastic simulation has proved to be a powerful method both for automatically creating plausible skeleton-dependent deformations and for introducing secondary motions. These simulations can be performed quickly with techniques like the finite element method for linearized elastic dynamics, allowing for real-time simulation suitable for interactive video games and rapid prototyping for film production.

A significant shortcoming of these approaches is that the simulations do not provide a way for the animator to control the shape, other than by posing the skeleton. In this paper we address this limitation by introducing force-based rigging. In our system, deformations are created indirectly through the use of forces.

Section 3 describes our physical and numeric framework. Our system is built on a new method of pose-dependent linearization of elasticity, which ensures a correspondence between deformations and forces, while producing plausible results (both static and dynamic equilibria) at the speed of linear dynamics (Section 3.3). To increase realism, we introduce a new technique for handling collisions near creases (Section 3.4). Section 4 describes our rigging framework, which includes interactive optimization-based rig configuration (Section 4.2), computing rig forces from sculpted or measured surface deformations (Section 4.3), transferring rigging

between characters (Section 4.4), and adaptive rigging (Section 4.5).

2. Background

Many methods have been devised for the geometric deformation of surfaces, including free-form deformation [26,17], shape interpolation [16,29,2], and wire deformation [28]. Such methods form the building blocks for rigging in modern animation programs such as Maya. These methods are not based on physical simulation, so the animator must hand craft physically realistic motions. The work of Kry et al. uses simulated input deformations to produce very realistic (but not dynamic) deformations [15].

Dynamic deformation based on continuum elasticity was introduced to computer graphics by Terzopoulos et al. [31]. Terzopoulos and Witkin showed that the equations of motion can be linearized about a moving frame of reference, as long as the deformations are modest [32]. The linearized equations are more stable and can be solved quickly. We will discuss more recent quasi-linear schemes in Section 3.3.

Physical constraints have been used to make elastic models more controllable for animation [23,34,5], but constraints have not been shown to be useful for providing detailed shape control.

Detailed geometric deformation was combined with dynamic deformation in an anatomical modeling framework by Wilhelms and Van Gelder [33] (a similar approach by Scheepers et al. did not include a dynamic layer [25]). The muscle shapes are a function of the configuration of the skeleton, and a dynamic fatty layer is modeled to connect the statically deforming muscles to dynamic skin. In recent years the use of anatomical models in animation has continued to be an active area of research. An interactive system for building anatomical models was presented by Aubel and Thalmann [4]. B-spline muscle models were fit to muscle data points by Ng-Thow-Hing and Fiume [21]. A sophisticated simulation for complex muscle groups has been developed by Teran et al. [30]. Our framework differs from anatomical modeling in that it allows general shape control in unison with dynamic deformation of the entire character, without explicitly representing internal anatomical structure.

Our framework is based on linear dynamics and can produce interactive animations, as in previous work on speeding up dynamic deformable models [11,14,19]. In particular, we build on the framework presented by Capell et al., where skeletal control for dynamic deformable models was introduced [10].

Singh et al. used forces to effect deformations [27]. Our system is distinguished by the computation of such forces from desired deformations, and the overall rigging framework based on forces.

3. Deformable character formulation

Our modeling and simulation of deformable characters builds on the framework of Capell et al. [10], augmented with rigging forces, a new pose-dependent linearization scheme, and collision handling near creases.

The basic data defining a *deformable character* are: (i) an elastic domain $\Omega \subset \mathbb{R}^3$ (the interior of a triangular mesh Γ), with specified mass density, Poisson ratio, and Young's modulus, (ii) an embedded cell complex (or *control lattice*) $K \subset \mathbb{R}^3$ containing Ω, (iii) a skeleton $S \subset \Omega$ consisting of prescribed edges of K, and (iv) a parameterized family $\mathbf{f}(\mathbf{x}, \Theta(t))$ of force fields (rigs) that can be used to deform the character into a desired shape. Fig. 1 demonstrates these concepts.

The state of the character at time t is represented by the function

$$\mathbf{p} : \Omega \times \mathbb{R} \to \mathbb{R}^3 : (\mathbf{x}, t) \mapsto \mathbf{p}(\mathbf{x}, t) = \mathbf{x} + \mathbf{d}(\mathbf{x}, t), \quad (1)$$

our animation framework gives a method for interactively determining the *displacement vector* $\mathbf{d}(\mathbf{x}, t)$ from a time-dependent family $\Theta(t)$ of control parameters that determine the pose of the skeleton and the force field $\mathbf{f}(\mathbf{x}, \Theta(t))$ by solving a system of linear differential equations (Eq. (9)).

Animation is controlled by a time-dependent vector of *pose parameters* $\alpha(t) \in \mathbb{R}^{N_\alpha}$, and another of

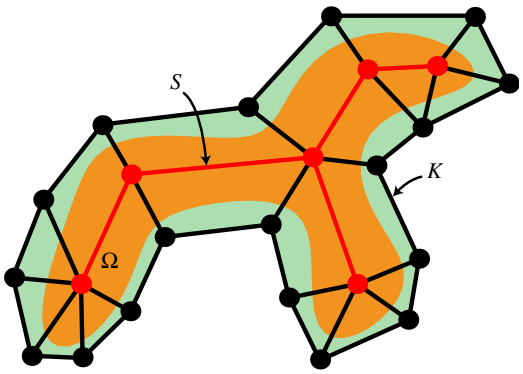

Fig. 1. An object $\Omega \subset \mathbb{R}^3$ instrumented with a control lattice K and skeleton S.

abstract parameters $\beta(t) \in \mathbb{R}^{N_\beta}$. The pose parameters and abstract parameters are combined into a single vector of *control parameters*

$$\begin{aligned}\Theta &= (\alpha_1, \ldots, \alpha_{N_\alpha}, \beta_1, \ldots, \beta_{N_\beta}) \\ &= (\theta_1, \ldots, \theta_{N_\theta}) \in \mathbb{R}^{N_\theta}. \end{aligned} \tag{2}$$

The components of α determine the configuration of the skeleton. The components of β serve as general-purpose shape controllers. For example, one such parameter value could be named *raised left eyebrow* and be expected to change the shape of the character appropriately. The animator may then implement the raising of the left eyebrow via a curve $\beta(t)$ in parameter space. An animation of the entire character is created by varying the control parameters $\Theta(t)$ over time.

The parameters Θ affect the shape of the character through objects that we call rigs (a variation on standard animation terminology). Each rig maps the control parameters to a force field acting on the body. The collection of all rigs on a character is thus modeled by a continuous family $\mathbf{f}(\mathbf{x}, \Theta)$ of force fields on Ω. Notice that we allow all of the elements of Θ to affect the shape, not just the elements of β. This allows the pose to affect the deformation, which is useful paradigm in animation. In particular, anatomical features of the shape such as muscle bulges are often pose-dependent.

3.1. Finite element approximation

To apply the finite element method, we need a suitable basis $\mathscr{B} = \{\phi_i(\mathbf{x})\}$ of functions on the domain Ω. To build such a basis, we parameterize a region $K \subseteq \Omega$ by a cell complex C using a piecewise smooth homeomorphism:

$$\mathbf{r} : C \to K : u \mapsto \mathbf{r}(u) \tag{3}$$

the mapping $\mathbf{r}$ is obtained by placing each of the vertices u_i of C into $\mathbb{R}^3$ and then applying trilinear subdivision. We refer to K, the image of C in $\mathbb{R}^3$, as a *control lattice*. Each basis function $\phi_i \in \mathscr{B}$ is centered at a vertex $\mathbf{x}_i$ of K, and is the restriction to Ω of a piecewise trilinear function on K, with $\phi_i(\mathbf{x}_i) = 1$ and with support contained in the union of cells of K containing $\mathbf{x}_i$. We write the displacement in the form

$$\mathbf{d}(\mathbf{x}, t) = \sum_i \mathbf{q}_i(t)\phi_i(\mathbf{x}), \quad \mathbf{q}_i(t) \in \mathbb{R}^3. \tag{4}$$

By construction, $\mathbf{d}(\mathbf{x}_i, t) = \mathbf{q}_i(t)$ for each vertex $\mathbf{x}_i$.

By definition, the *skeleton* of the character is a set S of edges of K contained in Ω. By construction, $\mathbf{p}(\mathbf{x}, t)$ is linear on each edge of S, and the coefficients $\mathbf{q}_{i(t)}$ at vertices of S are constrained so that the length of each edge of S is preserved by the mapping $\mathbf{p}$. By requiring the set S to be a tree that mirrors the joint hierarchy, we ensure that the skeletal parameters $(\alpha_1, \alpha_2, \ldots, \alpha_{m_\alpha})$ uniquely determine the value of the control coefficients $\mathbf{q}_i$ at the vertices of S. To simplify notation, we assemble the coefficients into two column vectors: a vector $\mathbf{q}_S$ whose i-component is $\mathbf{q}_i$, for i a vertex of S, and a vector $\mathbf{q}$ whose ith component is $\mathbf{q}_i$, for i vertex of $K\backslash S$. As we already observed, $\mathbf{q}_S$ is determined by the control parameters Θ.

3.2. Dynamics

The motion of the character is determined by specifying the control parameters $\Theta(t)$ and solving the *Euler–Lagrange* equations

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial T}{\partial \dot{\mathbf{q}}_i}\right) + \frac{\partial U}{\partial \mathbf{q}_i} = \mathbf{Q}_i \tag{5}$$

for the unconstrained coefficients $\mathbf{q}_i$, where $T = T(\dot{\mathbf{q}}, \dot{\mathbf{q}}_S)$ is the kinetic energy, the potential energy $U = U(\mathbf{q}, \mathbf{q}_S)$ is the sum of the elastic energy (a fourth degree polynomial in $\mathbf{q}$ and $\mathbf{q}_S$) and a twist energy that penalizes displacement near the skeleton and has the effect of simulating a thickened skeleton (see [10]), and $\mathbf{Q} = \mathbf{Q}(\Theta(t))$ is the vector of generalized forces determined by $\Theta(t)$. Each component

$$\mathbf{Q}_i(\Theta) = \int_\Omega \mathbf{f}(\mathbf{x}, \Theta)\phi_i(\mathbf{x})\,\mathrm{d}V \tag{6}$$

of $\mathbf{Q}$ represents the force acting on one control vertex at a point in $K\backslash S$. Later in the paper we will describe the creation and placement of specific force fields.

Substituting the expansions for T, U and $\mathbf{Q}$ in terms of the basis functions into Eq. (5) yields the system of non-linear, ordinary differential equations

$$\sum_j \mathbf{M}_{ij}\ddot{\mathbf{q}}_j = \mathbf{Q}^e_i + \mathbf{Q}_i(\Theta) - \sum_k \mathbf{N}_{ik}\ddot{\mathbf{q}}_{S,k}(t), \tag{7}$$

where $\mathbf{M}_{i,j}$ and $\mathbf{N}_{i,k}$ are constant 3×3 matrices. The function $\mathbf{Q}^e{}_i = -\partial U/\partial \mathbf{q}_i$ measures the internal elastic force acting on the body, and $\mathbf{Q}_i(\Theta)$ is the generalized rigging force centered at $\mathbf{x}_i$. The indices i and j range over vertices in $K\backslash S$ and k ranges over the vertices of S.

3.3. Skeletally warped linearization

The non-linear system in Eq. (7) is computationally expensive to solve, and potentially unstable, so various methods have been devised to approximate the equations [32,10,19,13,20]. These methods can efficiently produce plausible deformations, but each of them has shortcomings in the context of the present application. In particular, we require the following additional features that are not present in previous approaches:

- Because our framework uses forces to effect deformations, it is important to be able to compute forces that will consistently produce a given deformation for a given pose. The system must be invertible.
- When computing the rig forces for a given surface deformation at a given pose, we rely on the static equilibrium solution to the equations of motion. Computing static equilibrium must be efficient.

A pose-dependent linear system satisfies these goals.

We combine the benefits of the methods of Capell et al. [10] and Müller et al. [19], while eliminating some of their disadvantages. Like Capell et al., we use the configuration of the skeleton to predict the shape of the deformed object. But our method produces much more plausible deformations near joints. Like Müller et al., the linearization is performed independently at each vertex. But solving for static equilibrium using their method is a non-linear problem. Our method is fast, stable, and free of gross distortions.

Similar to [19], we linearize the elastic force by estimating the local transformation at each control vertex, resulting in the following approximation:

$$\mathbf{Q}_i^e = \frac{\partial U}{\partial \mathbf{q}_i} \approx \sum_j \mathbf{R}_i \mathbf{S}_{ij} \left(\mathbf{R}_i^T (\mathbf{x}_j + \mathbf{q}_j - \mathbf{c}_i) - \mathbf{x}_j \right), \qquad (8)$$

where the terms $\mathbf{S}_{ij}$ are constant 3×3 *stiffness matrices*, and $\mathbf{R}_i$ and $\mathbf{c}_i$ are the estimated rotation matrix and translation vector, respectively, at node i. Note that our formulation includes an estimate of the translation in order to accommodate the translation-dependent twist energy used in [10].

Rather than measuring $\mathbf{R}_i$ from the current state of the body, as in [19], we estimate $\mathbf{R}_i$ and $\mathbf{c}_i$ from the pose of the skeleton using a *skinning* methodology (e.g., [2,18]). The pose parameters $\alpha(t)$ determine a Euclidean motion of the form $\mathbf{R}_b(\mathbf{x} + \mathbf{c}_b)$, sending each edge or "bone" $b \subset S$ to the proper position in $\mathbb{R}^3$. We associate each vertex $\mathbf{x}_i$ of K with two bones of S, denoted b_{i1} and b_{i2} and we let $\mathbf{R}_i = \omega_{i1}\mathbf{R}_{i1} \oplus \omega_{i2}\mathbf{R}_{i2}$ and $\mathbf{c}_i = \omega_{i1}\mathbf{c}_{i1} + \omega_{i2}\mathbf{c}_{i2}$. The user-defined weights ω_{ik} dictate how much each bone affects each point. The symbol $\oplus$ in the formula for $\mathbf{R}_i$ represents interpolation of rotations. In our current implementation, we allow only two nonzero weights so that spherical linear interpolation can be used. More general methods are available for blending more than two rotations (e.g. [1]), at higher computational cost.

It is also necessary to transform the force fields to be aligned with the skeleton. For instance, a force field creating a bump might change to a dimple when the body is rotated 180°, if the forces are applied in the global frame. We therefore transform the forces according to the transformation of the body. For simplicity and stability, we use the transformation predicted by the skinning procedure. Hence, given a particular skeletal configuration Θ at time t, the generalized force at vertex i is of the form $\mathbf{R}_i\mathbf{Q}_i$. We note that our approach to interpolation by warping the force fields according to the skeleton is analogous to the interpolation of displacements used by Lewis et al. [16].

With these changes the system (Eq. (7)) assumes its final form

$$\sum_j \mathbf{M}_{i,j}\ddot{\mathbf{q}}_j = \sum_j \mathbf{R}_i(\Theta)\mathbf{S}_{ij}\left(\mathbf{R}_i^T(\Theta)(\mathbf{x}_j + \mathbf{q}_j - \mathbf{c}_i(\Theta)) - \mathbf{x}_j\right) + \mathbf{R}_i(\Theta)\mathbf{Q}_i(\Theta) - \sum_k \mathbf{N}_{i,k}\ddot{\mathbf{q}}_{S,k}(\Theta). \qquad (9)$$

Notice that Eq. (9) is a pose-dependent linear system. A comparison between our new method of linearization and the one presented in [10] (blended local linearization) is shown in Fig. 2.

It is noteworthy that Müller and Gross, in [20], present an improvement over [19]. To eliminate ghost forces, they estimate the rotation for each cell rather than at each vertex. Since our system is highly constrained by the skeleton, ghost forces have not been a problem. However, it would not be difficult to combine our approach with that of [20], predicting the rotation of each cell from the nearby skeleton. The result would offer the advantages of pose-dependent linearity without ghost forces.

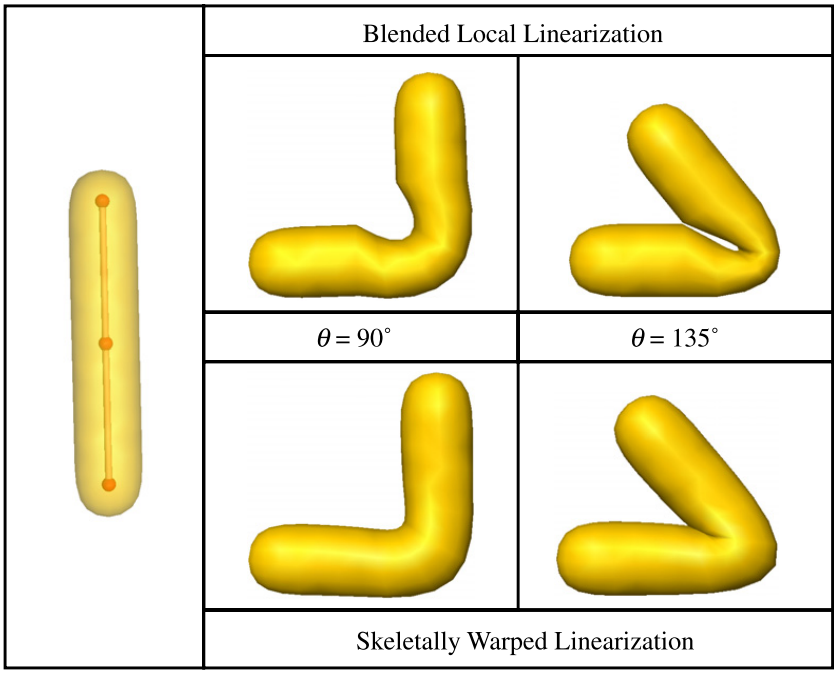

Fig. 2. Comparison between our skeletally warped linearization method and the blended local linearization scheme from [10]. The object shown on the left, instrumented with a simple skeleton, is put into two poses. On the top row are the results using blended local linearization. Notice how the region near the bend unrealistically contracts, similar to traditional skinning methods used in animation. On the bottom row are the results using our new method, which look much more plausible.

3.4. Self-collision detection and response

To realistically model human dynamics it is essential to take self-collisions of the surface of the character into account. When the human arm bends, for example, skin creases and folds, forcing muscles and fat to bulge outward around the elbow. We model such behavior within our framework by introducing constraints that prohibit surface interpenetration.

Interactive rates are achieved through a preprocessing step that determines the locations of likely collisions for a given animation. A deterministic propagating-front collision scheme, partially inspired by [22], is used to incrementally add constraints to the system as the collision surface expands. Raghupathi et al. have employed a similar technique in that they check the neighborhoods of regions in contact to find all colliding pairs in the area [24]. We expand on these ideas by introducing a preprocessing step that reduces the number of collision tests needed at runtime.

The image shown in Fig. 3b was generated using the collision response method described in this section, absent of any rigging forces. Our solution satisfies our requirement of efficiency. As Fig. 3 illustrates, the dynamical equations then produce the desired bulging effects as an artifact of the equations of elasticity.

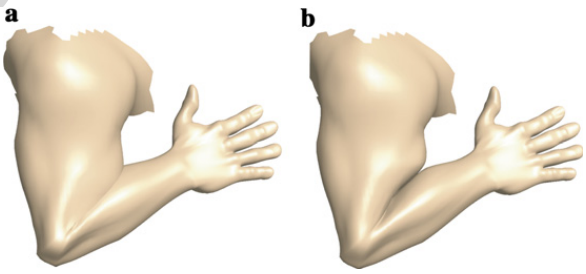

Fig. 3. Without (a) and with (b) collision detection.

3.4.1. Preprocessing

To efficiently detect self-collisions of the surface Γ as a simulation progresses, we determine approximate collisions using a preprocessing step. Consider a *target pose*, such as a bent elbow, in which self-collisions of the surface Γ occur. By choosing a family of control parameters $\Theta(t)$ that slowly deforms the skeleton from its rest state to the desired pose and solving the dynamic equations, we obtain a family of near equilibrium states deforming the character into the target pose. As this "slow motion" deformation evolves, we track the position of each vertex of the triangle mesh Γ.

By definition, a *collision* occurs when a vertex of Γ first penetrates a triangle of Γ. When a collision occurs, we activate a constraint that prevents further penetration (see below) and continue the dynamic simulation. The *seed set* D is the sequence of vertex–vertex pairs obtained by pairing each penetrating vertex with one of the vertices of the triangle that it penetrates. We make the *a priori* assumption that vertices outside the seed set will only collide after one of their neighbors has collided. If this assumption holds, the result is identical to that of general collision detection.

3.4.2. Run-time behavior

During a dynamic simulation, collisions not included in the seed set (generated in the preprocessing step) may occur. To maintain interactive rates, it is essential that we maintain the list of current collisions and a list of locations at which future collisions are likely to occur.

The *collision list* is the ordered list

$$C = \{(i_1, b_1), (i_2, b_2), \ldots, (i_{n_c}, b_{n_c})\} \tag{10}$$

of all pairs of the form (i,b) where i is a vertex of Γ and b are the barycentric coordinates of the point at which the vertex penetrates a triangle. The elements of C are ordered according to the time when each vertex enters the region Ω.

If a collision (i,b) occurs in the time interval $[t_0 - \Delta t, t_0]$, we impose the inequality constraint on (i,b) crafted to prevent the vertex from moving deeper into the interior of Ω

$$\chi_{i,b}(t) := (\dot{\mathbf{r}}_i(t) - \dot{\mathbf{r}}_b(t)) \cdot \mathbf{n}(t) \geqslant 0 \quad \text{for all } t \geqslant t_0, \tag{11}$$

where $\mathbf{r}_i(t)$ and $\mathbf{r}_b(t)$ denote spatial locations of i and b at time t, and $\mathbf{n}(t)$ is the normal to the triangle T at time t.

We classify elements of C into *active* and *inactive* elements. More precisely, let $n_a \leqslant n_c$ be the smallest non-negative integer such that if we impose the equality constraint

$$\chi_{i_k,b_k}(t) = 0 \quad \text{for all } k < n_a \tag{12}$$

then $\chi_{i_k,b_k} > 0$ for all $n_a \leqslant k \leqslant n_c$.

We assume that future collisions will either occur near one of the collision seeds or in a neighborhood of the current collision set. Specifically, the *neighborhood* of a pair $(i,p) \in C$ is the set $N(i,b)$ of all pairs (i',j') of vertices of Γ such that i' is in a neighborhood of i and j' is in a neighborhood of p.

At each time step we perform the following operations:

(1) For each pair $(i,j) \in P$, check for penetration of i into each triangle in a neighborhood of j as well as for penetration of j into each triangle in a neighborhood of i. When a penetration is detected, add the corresponding pair (i,b) or (j,b) to C.
(2) Remove any pair (i,b) from C for which the point $\mathbf{r}_i(t)$ is exterior to Ω. The inequality
$$(\mathbf{r}_i(t) - \mathbf{r}_b(t)) \cdot \mathbf{n}(t) > 0$$
provides an easy test.
(3) Update P to be the union $P = D \cup \bigcup_{(i,b)\in C} N(i,b)$.
Find the number n_a that determines the set of active elements of C.

Note that because we detect collisions after they have occurred, small interpenetrations will occur. However, our constraints prevent any further incursion once collisions are detected. The result is visually appealing, leaving no visible gaps between regions in contact.

To maintain the constraints $\chi_{i,b} = 0$ we use the modified conjugate gradient method presented by Baraff and Witkin [6] and extended to FEM by Capell et al. [10].

The set of all colliding points and active constraints $\chi_{i,b} = 0$ in this formulation will generally be significantly higher than the number of degrees of freedom provided by the control lattice. We build an orthogonal basis in constraint space from the list of constraints using a QR-decomposition. This basis spanning the active constraint space is used to project out components of the solution that violate the constraints within the conjugate gradient (CG) solver. The use of such orthogonalization of constraints to project the null space of the constraint basis has been shown to still lead to guaranteed convergence of the CG [3].

4. Rigging—shape control using forces

Shape control is important because it enables the animator to create more realistic deformations and express the emotion and intent of a character. For example, skeletal controls will not effectively help the animator to make a character appear to breathe, produce a muscle bulge, or smile. To address this

shortcoming, we introduce an additional control mechanism.

Since our framework is based on physical simulation, two natural mechanisms exist for influencing the shape: hard constraints and forces. Forces (or force fields) are better suited to our framework for a variety of reasons: (1) in contrast with hard constraints, which are rigid and therefore not a realistic model for animal tissues other than bones, forces influence the shape without destroying its dynamic nature. (2) Forces can coexist without troublesome compatibility issues—if two force fields overlap, their combined effect is obtained by vector addition. (3) Finally, force fields need not line up exactly with the character's surface geometry. Our framework already handles the embedding of the character; if the force field extends beyond the interior of the character it can easily be restricted to the interior.

To use forces to effect deformations, we exploit an important feature of pose-dependent linearization. For a fixed skeletal pose, deformations and generalized forces are in correspondence. Consider the body in static equilibrium ($\dot{\mathbf{q}} = \mathbf{0}$ and $\ddot{\mathbf{q}} = \mathbf{0}$). Eq. (9) reduces to the form

$$\mathbf{A}(\alpha)\mathbf{q} = \mathbf{b}(\alpha) + \mathbf{Q}, \tag{13}$$

where $\mathbf{A}(\alpha)$ is a matrix (containing 3×3 blocks), and $\mathbf{b}(\alpha)$ is a vector (with elements that are 3-vectors). Since $\mathbf{A}(\alpha)$ is invertible, there is a bijection between generalized forces $\mathbf{Q}$ and displacements $\mathbf{q}$. This relationship allows us to think about shape in terms of forces. While our simulations are dynamic, the rigging is configured while running a static equilibrium solver (i.e., solving Eq. (13)), where forces and displacements are in correspondence.

The rigging process, by which the mapping $\mathbf{f}$ is constructed is based on two independent paradigms, based on two kinds of components: *force field templates* and *surface deformation rigs*. Force field templates are reusable components representing parameterized force fields in a canonical coordinate frame. We will discuss them in Section 4.2. Surface deformation rigs are rigs constructed directly from surface deformations. This gives the user more control by leveraging existing technologies such as surface scanners and geometric modeling software. Surface deformation rigs are presented in Section 4.3.

4.1. The rigging process

To rig a character, individual rigs must be configured and combined to form the rigging force $\mathbf{f}(\mathbf{x}, \Theta)$. Each rig is described as a force $\mathbf{f}_a(\mathbf{x}, \eta_a)$, where η_a is a vector of *rig parameters* that determine the force produced by the ath rig. The rigging process consists of deciding on a set of rigs and creating a mapping from Θ to η_a for each rig. For simplicity, we impose the restriction that the ath rig depends on a single control parameter $\gamma_a \in \Theta$. The rig force is then formed by combining the force contributions from the rigs:

$$\mathbf{f}(\mathbf{x}, \Theta) = \sum_{a=1}^{N_{\text{rigs}}} \mathbf{f}_a(\mathbf{x}, \tau_a(\gamma_a)), \tag{14}$$

where the function $\tau_a : \gamma_a \mapsto \eta_a$ maps the control parameter associated with rig a to its rig parameters. It is also convenient to express the rig force purely in terms of generalized forces:

$$\mathbf{Q}_i(\Theta) = \mathbf{R}_i(\alpha) \sum_{a=1}^{N_{\text{rigs}}} \mathbf{Q}_i^a(\gamma_a), \tag{15}$$

where $\mathbf{Q}_i^a$ is the generalized force (on the free variables) contributed by the ath rig.

We represent the function τ_a as the linear interpolation of a set of sample points $\{(\gamma_{a,k}, \eta_{a,k}) : k = 1, 2, \ldots, N_a^\tau, \gamma_{a,k} < \gamma_{a,k+1}\}$ that are created by the user during an interactive session:

$$\tau_a(\gamma_a) = \begin{cases} \eta_{a,0} & \text{for } \gamma_a \leqslant \gamma_{a,0} \\ \eta_{a,k} + (\eta_{a,k+1} - \eta_{a,k})\left(\frac{\gamma_a - \gamma_{a,k}}{\gamma_{a,k+1} - \gamma_{a,k}}\right) & \\ \quad \text{for } \gamma_{a,k} < \gamma_a \leqslant \gamma_{a,k+1}, 1 \leqslant k < N_a^\tau & \\ \eta_{a,N_a^\tau} & \text{for } \gamma_{a,N_a^\tau} < \gamma_a. \end{cases} \tag{16}$$

The process by which the user interactively rigs a character can be described algorithmically as follows:

```
a ← 1
while the character is not fully rigged do
    instantiate rig a
    choose a control parameter γ_a ∈ Θ for the rig
    k ← 1
    while the user desires more samples do
        configure the character by setting Θ
        set the control parameter γ_a ← γ_{a,k}
        configure the rig by setting η_a ← η_{a,k}
        record the sample (γ_{a,k}, η_{a,k})
        k ← k + 1
    end
    a ← a + 1
end
```

4.2. Rigging with force field templates

Our first approach to rigging is to rely on a library of predefined force field templates. Each template defines a parameterized force field in a canonical coordinate frame. The following design goals guided our selection of templates:

- A force template should produce an effect that is easy for the user to understand.
- It should have few control parameters so that the user and optimizer (Section 4.2.2) can easily configure it. The parameters must be intuitively meaningful to the user.
- It should be continuous and have continuous first derivatives so that changes in the parameters produce smooth changes in the forces.
- It should be spatially localized and have a simple closed form so that it can be evaluated and integrated efficiently to compute generalized forces.

We have designed four force field templates with the above goals in mind: *bump*, *bulge*, *radial*, and *torus*. The bump template is supported on a ball, with forces aligned in a single direction. The bulge template is inspired by the behavior of a muscle. Roughly speaking, when a muscle contracts in one direction it expands in the other two directions (see, e.g. [25]). The bulge behaves similarly. The radial template is supported on a ball with forces pointing away from the center. The torus template is supported on the interior of a torus. Each circular cross section of the torus contains forces radiating outward from the center of the cross section.

4.2.1. Template guides

When a template is applied to a character, it must be positioned and oriented. To simplify this task we introduce template guides. Each template guide supports a unique user interface that allows the user to easily position and orient the template. The six degrees of freedom of position and orientation are reparameterized and divided into three categories by the template guide. Some of them are configured automatically as the user drag-and-drops the template. Some are eliminated. The remaining DOFs can be configured using sliders if desired. By reducing the degrees of freedom of position and orientation, we also ease the workload on the optimizer during optimization-driven rigging (Section 4.2.2).

Bone guides position and orient the template relative to a bone of the character. *Skin guides* position and orient the template with respect to the skin of the character. *Free guides* can be used when the other two template guides are too restrictive. See [9] for more a detailed description of force field templates and template guides.

When a force field template is combined with a template guide, together they form a rig, producing a force of the form given in Eq. (14). The rig parameters η are a concatenation of the template parameters and the template guide parameters. The rig is configured by manually setting the template parameters, dragging-and-dropping the template according to the chosen template guide, and manually adjusting the remaining position and orientation DOFs if desired.

4.2.2. Optimization-based template configuration

In order to make it easier for the user to configure a rig, we support an interface wherein the user directly manipulates the surface of the character with the mouse. The system automatically optimizes the parameters of the rig so that the forces cause the surface of the character to conform to the user's input.

Consider a scenario in which a single rig, with parameter vector η is being configured. By Eq. (13), $\mathbf{q} = \mathbf{q}(\eta)$. Using a drag-and-drop interface, the user indicates target positions $\bar{\mathbf{p}}_1, \bar{\mathbf{p}}_2, \ldots, \bar{\mathbf{p}}_{N^{\mathbf{p}}}$ for the points $\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_{N^{\mathbf{p}}}$ on the input surface Γ. Our goal is to choose η so that these constraints are met. This is accomplished by minimizing the cost function

$$C(\eta, \mathbf{q}) = C_{\text{points}}(\mathbf{q}) + C_{\text{params}}(\eta), \tag{17}$$

where the term

$$C_{\text{points}}(\mathbf{q}) = \sum_{j=1}^{N^{\mathbf{p}}} |\mathbf{p}_j - \bar{\mathbf{p}}_j|^2 \tag{18}$$

enforces the constraints, and

$$C_{\text{params}}(\eta) = \sum_{j=1}^{N^{\eta}} \xi_j (\eta_{0,j} - \eta_j)^2 \tag{19}$$

penalizes deviation of η from its preferred value η_0. The weight ξ_j determines the magnitude of the penalty associated with the jth parameter.

We optimize the cost C using the L-BFGS-B algorithm [8], a quasi-Newtonian solver with limited memory usage. Standard optimization methods for smooth cost functions, such as L-BFGS-B, require knowledge of the gradient $\frac{dC}{d\eta}$. To compute it efficiently, we apply the chain rule as follows:

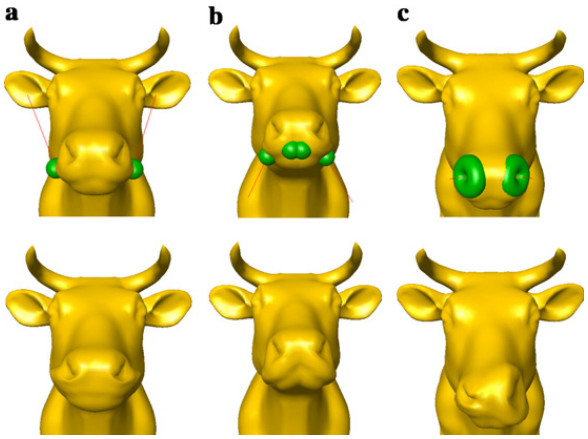

Fig. 4. A cow head interactively rigged using force field templates to create simple facial expressions. The templates are visualized in the top row. The bottom row shows the resulting deformations. (a) Two bump templates create a smile. (b) Four bump templates effect a frown. (c) Two torus templates produce dilation and contraction of the nostrils.

$$\begin{aligned}\frac{dC}{d\eta} &= \frac{\partial C}{\partial \eta} + \frac{\partial C}{\partial \mathbf{q}}\frac{d\mathbf{q}}{d\eta} \\ &= \frac{\partial C}{\partial \eta} + \frac{\partial C}{\partial \mathbf{q}}\left(\mathbf{A}^{-1}\frac{d\mathbf{Q}}{d\eta}\right) \\ &= \frac{\partial C}{\partial \eta} + \left(\frac{\partial C}{\partial \mathbf{q}}\mathbf{A}^{-1}\right)\frac{d\mathbf{Q}}{d\eta}.\end{aligned} \tag{20}$$

Both $\frac{\partial C}{\partial \eta}$ and $\frac{\partial C}{\partial \mathbf{q}}$ have simple analytic forms, but $\frac{d\mathbf{Q}}{d\eta}$ requires integration over the body (Eq. (6)) and is difficult to compute analytically. We therefore use finite differences to compute $\frac{d\mathbf{Q}}{d\eta}$, which requires one integration over the body per element of η. The important gain in efficiency comes from the regrouping involving $\mathbf{A}^{-1}$. After regrouping, rather than solving a linear system for each element of η, we need only solve one. This results in much faster optimization. Using this method we achieve interactive optimizations in the case where a single rig is being configured. Fig. 4 shows the result of interactively rigging the head of a cow with force field templates.

4.3. Deriving rigs from surface deformations

In addition to predefined force field templates, our framework supports surface deformation rigs, which are constructed directly from surface deformations. A surface deformation rig uses an arbitrary generalized force to effect the desired shape change. We express the force of a surface deformation rig directly in the generalized form of Eq. (15), dropping the index a since we will only be discussing a single rig:

$$\mathbf{Q}(\gamma) = \gamma\eta. \tag{21}$$

For surface deformation rigs, the rig parameters η determine the direction of the generalized force and the rig control γ determines its magnitude. The rig is configured by computing the optimal parameters η so that a target surface is matched.

Given a target surface $\bar{\Gamma}$ represented as a triangle mesh, we want to compute a generalized force such that $\bar{\Gamma} \approx \Gamma$, where Γ is the surface of the simulated body at static equilibrium. This computation replaces the user interaction in the rigging process. Because Γ also depends on the skeletal configuration parameters α, which are unknown for the target surface, we must also compute an optimal skeletal pose. Our approach to optimizing the skeletal pose is based on the work of Allen et al. [2].

4.3.1. Pose optimization

Our procedure for pose optimization is based on matching a set of user-selected feature points $\{\mathbf{p}_i\}$ on Γ and $\{\bar{\mathbf{p}}_i\}$ on $\bar{\Gamma}$ using the following cost function:

$$C_{\text{features}}(\alpha) = \frac{1}{N_{\text{features}}}\sum_{k=1}^{N_{\text{features}}} |\mathbf{p}_k - \bar{\mathbf{p}}_k|^2, \tag{22}$$

where N_{features} is the number of feature points. Although it requires user interaction, the selection of feature points by the user has the advantage that points can be chosen that are representative of the configuration of the underlying skeleton, such as places where the bone is close to the surface. The L-BFGS-B algorithm is used to solve for the optimal pose parameters α. Fig. 5 shows the results of optimizing the pose to fit a target surface.

4.3.2. Force optimization

After finding the optimal skeletal pose associated with $\bar{\Gamma}$, and setting the value of γ, we compute the optimal generalized force (via η) that aligns the deformed surface Γ with the target surface $\bar{\Gamma}$. Our goal is then to find a vector η that minimizes the following cost function that penalizes mismatch between Γ and $\bar{\Gamma}$:

$$\begin{aligned}C(\eta) &= C_{\text{surf}}(\eta) + \omega C_{\text{smooth}}(\eta) \\ &= \frac{1}{N_{\text{verts}}}\sum_{k=1}^{N_{\text{verts}}} \text{dist}(\mathbf{v}_k, \bar{\Gamma})^2 \\ &\quad + \frac{\omega}{2}\sum_{k=1}^{3}\int_{\Omega} |\nabla d_k|^2 dV,\end{aligned} \tag{23}$$

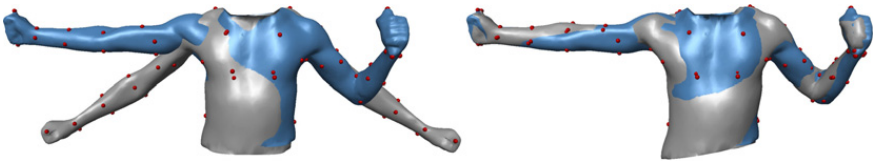

Fig. 5. In this example, a skeletal pose is computed that deforms one surface to fit the other. The spheres indicate user-selected feature points that guide the optimization.

where ω is a user-specified smoothing parameter. Here N_{verts} is the number of vertices in Γ, $\mathbf{v}_k$ are the vertices of Γ, dist measures the minimum distance between a point and a surface, and d_k is the kth component of $\mathbf{d}(\mathbf{x})$. The term C_{smooth} is a smoothness penalty. If the surfaces are in correspondence the expression $\text{dist}(\mathbf{v}_k, \bar{\Gamma})$ can be replaced by $|\mathbf{v}_k - \bar{\mathbf{v}}_k|$.

In some cases, we would like to match surfaces that are not expected to match at every point. For example, we may want to match an isolated arm to an arm belonging to a full human body. In cases like this, a lasso tool is used to select the vertices on Γ that are expected to correspond with $\bar{\Gamma}$. In this case we modify the cost function to only include the selected vertices.

Fig. 6 shows the results of creating a surface deformation rig to match a bent arm. Because the deformation is associated with the arm in a bent pose, we chose the angle of the elbow joint as the control parameter γ associated with the rig. As γ varies from 0 to 1 the force that effects the deformation is gradually introduced. In this manner, intermediate poses of the elbow produce intermediate deformations. Fig. 7 shows the effect of the rig when the pose of the arm differs from the pose of the target data.

The optimization was broken into two stages to simplify the problem. Although it worked well in our examples, it is conceivable that it could fail in cases where the deformation is larger. In such cases it might be necessary to optimize simultaneously for the pose and forces.

4.4. Retargeting surface deformation rigs

Deformed surfaces can be difficult to construct, and sometimes a deformation is available for one model but not for another similar model. For example, we may want to apply the deformations from a large person to a smaller person, or from a man to a woman. For these reasons, we would like to transfer

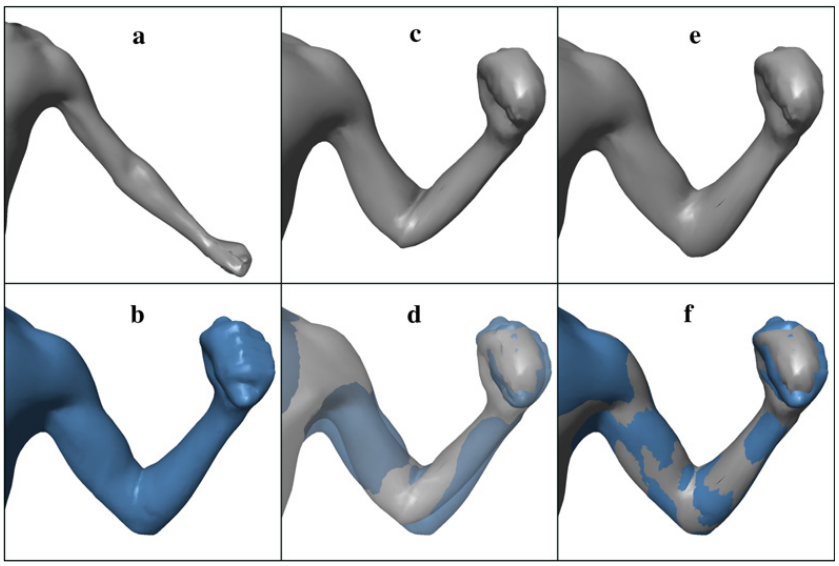

Fig. 6. A surface deformation rig for a bent arm. (a) A scanned arm in its rest configuration. The arm has been instrumented with a skeleton and control lattice for elastic simulation. (b) The target surface, which was also acquired by scanning. We want to compute the forces that approximately produce the target surface when applied to the simulated arm shown in (a). (c) The static equilibrium shape of the arm in (a) posed to match (b) as well as possible. (d) A comparison between (b) and (c). (e) The bent arm after applying an optimized surface deformation rig that uses forces to produce the target shape. (f) A comparison between (b) and (e).

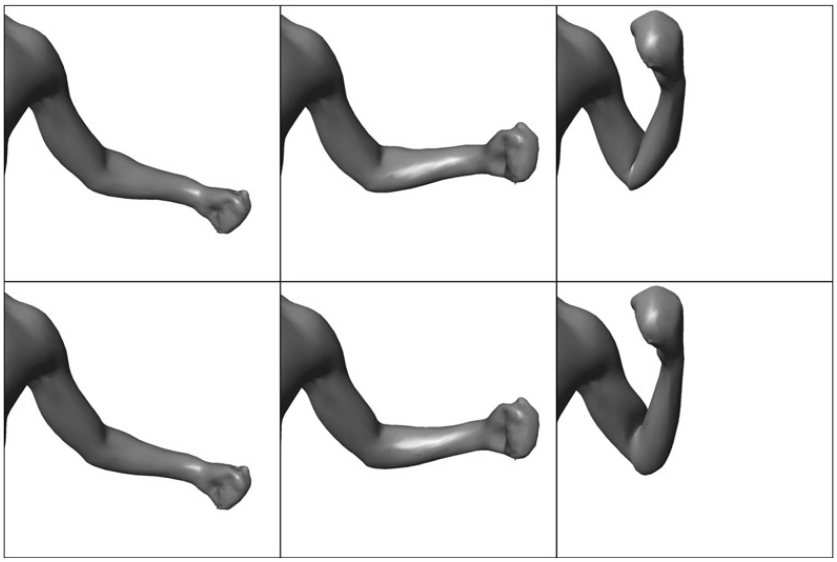

Fig. 7. The rig from Fig. 6 used at different poses. The top row shows the simulation without rigging. The bottom row shows the same poses simulated with rig forces in effect. Notice that the biceps and the area around the crease of the elbow are much more realistic looking when the rig is used.

rigs between characters. In this section, we discuss the transfer of surface deformation rigs from one character to another. We accomplish this by creating a mapping between the two characters and then using the map to transfer forces.

4.4.1. Mapping between characters

Suppose that we have a character whose domain is Ω with surface $\Gamma = \partial\Omega$ and skeleton S, and that the character has been parameterized by a cell complex C in the manner described in Section 3.1, i.e., there is a homeomorphism $\mathbf{r}: C \to K \subseteq \Omega$. Our goal is to create a mapping from Ω to a new character $\bar{\Omega}$ with surface $\bar{\Gamma} = \partial\bar{\Omega}$ and skeleton $\bar{S}$. We approximate such a mapping by repositioning the vertices of the control lattice to form a homeomorphism $\bar{\mathbf{r}} : C \to \bar{K} \supset \bar{\Omega}$ using the trilinear basis. We can then form a trilinear mapping between the characters as follows:

$$\mathbf{h} : K \to \bar{K} : \mathbf{x} \mapsto \bar{\mathbf{r}}(\mathbf{r}^{-1}(\mathbf{x})). \tag{24}$$

Our goal is to choose $\bar{\mathbf{r}}$ so that $\mathbf{h}(\Omega) \approx \bar{\Omega}$.

We reposition the vertices of K using an optimization procedure that minimizes a cost function of the form

$$C(\mathbf{h}) = \omega_{\text{skel}} C_{\text{skel}}(\mathbf{h}) + \omega_{\text{surf}} C_{\text{surf}}(\mathbf{h}) + C_{\text{distort}}(\mathbf{h}). \tag{25}$$

The first term encourages the skeletons to match by penalizing the difference $\bar{S} - \mathbf{h}(S)$. The second term (defined in Section 4.3.2) matches the surfaces by penalizing the difference $\bar{\Gamma} - \mathbf{h}(\Gamma)$. The last term penalizes distortion in the mapping. The user defined weights ω_{skel} and ω_{surf} determine how much each term contributes to the total cost. For efficiency, and to avoid local minima, we perform the optimization in four stages.

In the first stage, we attempt to find a good starting guess for our (non-linear) optimization procedure. We grossly align the models by minimizing the cost function

$$C_1(\mathbf{h}) = C_{\text{skel}}(\mathbf{h}) = \frac{1}{N_{\mathbf{b}}} \sum_{k=1}^{N_{\mathbf{b}}} |\mathbf{b}_i - \bar{\mathbf{b}}_i|^2 \tag{26}$$

while allowing the lattice only seven degrees of freedom for translation, rotation and scale. Here, $\mathbf{b}_k$ and $\bar{\mathbf{b}}_k$, $1 \leqslant k \leqslant N_{\mathbf{b}}$, denote the vertices of S and $\bar{S}$, respectively.

In the second stage, we allow deformation to occur and attempt to match up the skeletons while minimizing distortion of the mapping. We accomplish this by minimizing the cost function

$$C_2(\mathbf{h}) = C_{\text{skel}}(\mathbf{h}) + C_{\text{distort}}(\mathbf{h}). \tag{27}$$

Our distortion penalty was chosen to be scale, rotation, and translation invariant:

$$C_{\text{distort}}(\mathbf{h}) = \frac{1}{N_a} \sum_{k=1}^{N_a} (a_k - \bar{a}_k)^2, \tag{28}$$

where a_k and $\bar{a}_k$, $1 \leqslant k \leqslant N_a$, are the set of face angles in the cell complexes, $\bar{K}$ and K, respectively, as embedded in $\mathbb{R}^3$ by the mappings $\mathbf{r}$ and $\bar{\mathbf{r}}$. Each face

angle measures the angle between two edges of the lattice that belong to the same cell and have a vertex in common.

In the third stage, in addition to the skeleton, we attempt to match the surface. Instead of matching the whole surface, we begin by matching user-selected feature points using the cost function $C_3(\mathbf{h}) = C_2(\mathbf{h}) + \omega_f C_{\text{features}}(\mathbf{h})$. This stage, the only one that requires user interaction (to select the feature points), is optional but drastically speeds up the optimization.

In the fourth and final stage, we match all of the surface points, instead of only the feature points, using the cost function of Eq. (25).

Due to the coarseness of the control lattice K, the above procedure does not usually produce a perfect answer. In difficult areas, such as underarm creases, it is necessary to manually adjust a few control points to ensure that the surface is embedded in K.

The weights used to scale each of the cost terms were chosen by trial and error. The term C_{skel} was weighted very high because matching the skeleton is paramount; after optimization, the control lattice vertices are moved to the closest skeleton vertex. We weighted the term C_{distort} as low as possible. It was increased as needed to remedy problems with the cells in the control lattice becoming overly deformed or inverting.

4.4.2. Force transfer

Once the mapping $\mathbf{h}$ from Ω to $\bar{\Omega}$ has been established, we can use it to transfer a generalized force $\mathbf{Q}$ from Ω to $\bar{\Omega}$. Transferring the generalized forces directly between different physical systems produced unintuitive results, so instead we convert the generalized forces to displacements, transfer the displacements, and then convert back to generalized forces. In doing so we take advantage of the fact that our pose-dependent linear system draws a correspondence between generalized forces and displacements.

Consider a generalized force $\mathbf{Q}$ acting on the character whose domain is Ω. We can convert it to a displacement $\mathbf{d}$ by first applying Eq. (13) to compute the generalized coordinates $\mathbf{q}$ and then applying the basis expansion in Eq. (4). The laws of vector transformation dictate that the transformed displacement $\bar{\mathbf{d}}$ satisfies the equation $\bar{\mathbf{d}}(\bar{\mathbf{x}}) = \mathbf{J}(\mathbf{x})\mathbf{d}(\mathbf{x})$, where $\mathbf{J}(\mathbf{x}) = \frac{\partial \mathbf{h}}{\partial \mathbf{x}}(\mathbf{x})$. We approximate this transformation by transforming the coefficients: $\bar{\mathbf{q}}_i = \mathbf{J}(\mathbf{x}_i)\mathbf{q}_i$.

Although $\mathbf{h}$ is not differentiable at $\mathbf{x}_i$, we can estimate $\mathbf{J}_i$ at the vertex $\mathbf{x}_i$ of the lattice by computing the affine map that minimizes the energy function

$$E(\mathbf{J}_i) = \sum_{j=1}^{N} (\mathbf{J}_i \mathbf{e}_{ij} - \bar{\mathbf{e}}_{ij})^2, \tag{29}$$

where $\mathbf{e}_{ij} = \mathbf{r}_j - \mathbf{r}_i$ and $\bar{\mathbf{e}}_{ij} = \bar{\mathbf{r}}_j - \bar{\mathbf{r}}_i$ are edge vectors surrounding vertex i in K and $\bar{K}$, respectively. We minimize E by the Procrustes algorithm [7]. Once we have computed $\bar{\mathbf{q}}$ as above, Eq. (13) can be applied again to produce the generalized force $\bar{\mathbf{Q}}$ for the target character.

4.5. Adaptive rigging

To make surface deformation rigs match the input surface better, we can increase the resolution of the control lattice. But doing so greatly increases the computational cost of the simulation. To reduce the cost we apply an adaptive simulation procedure.

During the optimization of the rig forces, we adapt the basis in order to match the simulated surface to the goal surface, as in the work of Gortler and Cohen [12]. We apply the following procedure:

```
for l ← 0 to N_levels − 1 do
    optimize the force Q as in Section 4.3.2
    for k ← 1 to N_verts do
        if dist(v_k, Γ) > ε then
            activate all level l basis functions that support v_k
        end
    end
end
```

The quantity ϵ represents a user-defined distance threshold. The result of the above procedure is that the ath surface deformation rig has an associated basis $\mathscr{B}^a$ that supports the approximation of the input surface that was used to create the rig. When animating we adapt the basis by setting $\mathscr{B}_A \leftarrow \mathscr{B}_A \cup \mathscr{B}^a$ whenever the ath rig is active ($\gamma_a \neq 0$). If the rig is not active, basis functions may be deactivated.

5. Results

Some of the results of our rigging system have already been shown in Figs. 3, 4, 6, and 7. We tested the system on two additional input surfaces:

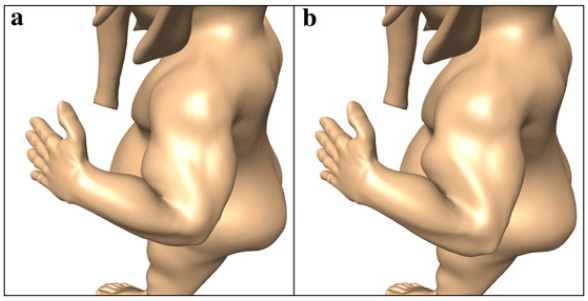

Fig. 8. (a) The unrigged Ganesh arm (in a bent pose). (b) The arm with rigging transferred from the arm scan in Fig. 6. On Ganesh the rig causes similar effects as on the original arm model: the elbow extends, becoming less rounded, and the biceps bulges.

"Ganesh" (a rotund humanoid figure with an elephant's head) designed using geometric modeling software, and a scan of a human. The Ganesh character was instrumented with three rigs which were then transferred to the human character. One of the rigs was derived from a third model, the arm scan demonstrated in Figs. 6 and 7. Another rig, representing the flexing of the chest, was derived from a surface deformation designed by an artist. A third rig, effecting a breathing motion of the torso, was created using a force field template.

Each of these examples required roughly a day to set up. Constructing the control lattice is currently a laborious process requiring manual placement of vertices and cells. Interactively placing the rigs on the cow head model was done in less than an hour. Tagging the surfaces with feature points to guide the optimization required about an hour. Most of the optimizations required less than an hour, although the high resolution version of the chest flex took several hours to compute. The mapping from one character to another required about an hour of manual tweaking to ensure that the control lattice completely contained the new character.

Figs. 8 and 9 demonstrate the transfer of rigging between characters. In Fig. 8, the rig derived from the bent arm scan (Fig. 6) has been transferred to the Ganesh character. It is noteworthy that the thin human arm on which the rig is based and the Ganesh arm behave quite differently when using the unrigged model. The Ganesh arm is short, fat, and cartoon-like, so the deficiencies of the simplified physical model are not as apparent as for the thin human arm as in Fig. 6c and d. Despite their significant differences, the rig produces a similar effect on Ganesh as on the scanned arm. The biceps bulges in

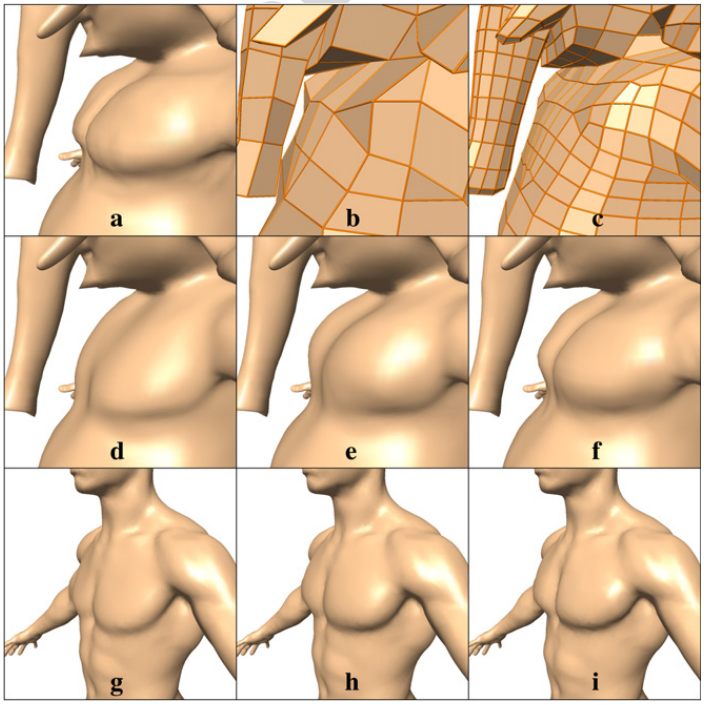

Fig. 9. A surface deformation rig effecting a chest flex. (a) A deformation representing Ganesh with a flexed chest, created using geometric modeling software. The second row shows the use of a force field rig to effect the same shape change. (d) The model at rest. (e) A coarse simulation (the surface of the volumetric lattice is shown in (b) above). (f) A finer simulation (the lattice is shown in (c) above). The bottom row shows the chest flex rig transferred to the human model: (g) at rest, (h) coarse, and (i) fine.

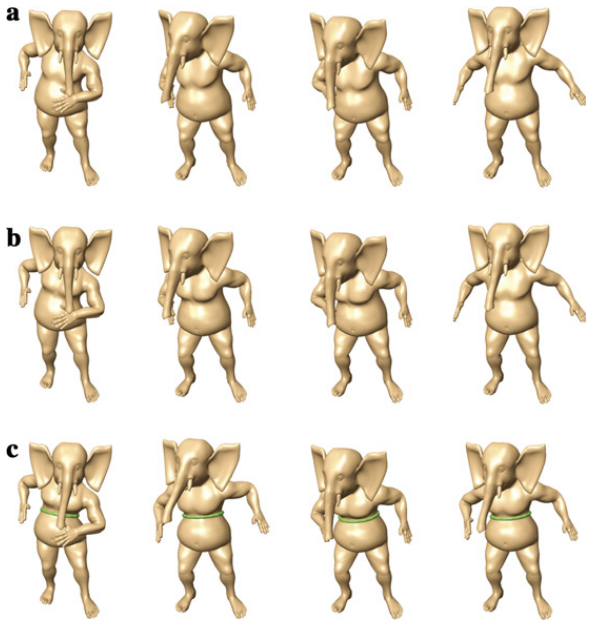

Fig. 10. Animations using a chest flex rig. (a) Animation with no force-based rigging. (b) Animation using a chest flex rig. (c) Animation using the chest flex and a belt constraint.

an appropriate place and the elbow extends, counteracting an unrealistic contraction.

The transfer of a chest flex rig from Ganesh to the human model is shown in Fig. 9. The chest flex rig was created using a target deformation created using geometric modeling software. The transferred rig produces a plausible deformation despite the fact that the characters are very different.

Fig. 10 demonstrates frames of an animation of the rigged Ganesh character. In row (c), a belt constraint has been applied, demonstrating that our system supports constraints along with rigging. Although we have not done extensive timing studies, the cost of simulation is about equally split between overhead (e.g., bookkeeping, rendering) and solving the sparse linear system at each time step.

The use of adaptive simulation for rigging is demonstrated in Fig. 11. Notice that the result of the adaptively simulated chest flex using two levels of basis functions (Fig. 11c) is visually identical to the non-adaptive simulation using a twice subdivided control lattice (Fig. 9d). But the computational cost of the adaptive simulation is significantly less. The adaptive simulation required about 0.18 s per simulation step, while the non-adaptive simulation required 1.2 s. The discrepancy is of course due to the fact that the adaptive simulation requires fewer degrees of freedom (2637 as compared to 9957 for the non-adaptive simulation).

Several shortcomings can be seen in the animations that result from our approach (see the supplementary video). The oscillations of the body do not always look realistic. Sometimes the body appears to be too jiggly, and sometimes too firm. This is not a limitation in the ability of the simulation to handle various physical parameters. It results from the fact that without an anatomical mechanical model of the character we cannot expect the nuanced dynamic behavior that results from a variety of tissues interacting. In our framework, a remedy to this problem would be to learn spatially varying material parameters from animation data captured from real creatures, or those simulated using more sophisticated anatomical models.

Another shortcoming that can be seen in the supplementary video is that sometimes the chosen resolution of the simulation is not high enough to avoid

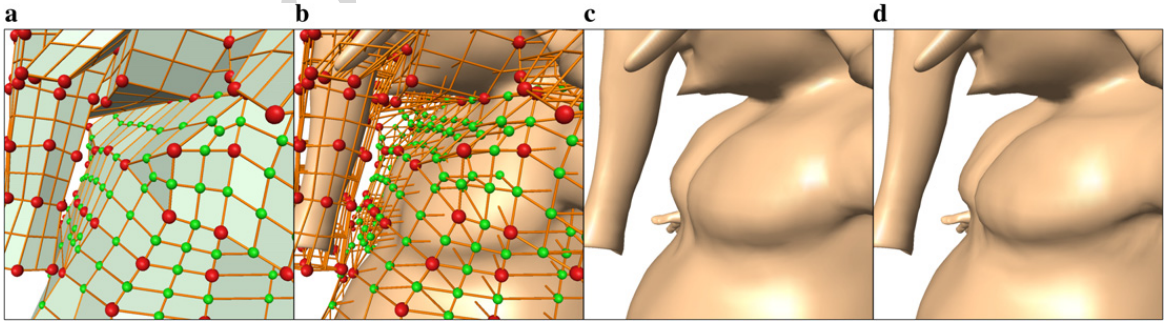

Fig. 11. Adaptive simulation of the chest flex rig. (a) The surface of the control lattice at rest. Red spheres represent active level 0 basis functions and green spheres represent active level 1 basis functions. Notice that outside the chest region the level 1 basis functions are inactive. The active basis functions are being shown on the rest configuration for clarity, although they were selected during adaptive optimization of the rig forces. (b) The volumetric control lattice. (c) The chest flex resulting from adaptive simulation using the basis functions shown in (a) and (b). (d) The target surface. (For interpretation of color mentioned in this figure the reader is referred to the web version of the article.)

undesirable artifacts. For example, in the simulation in which collision detection is applied to a bending arm, the coarsness of the control lattice causes the inner forearm to collapse somewhat. The effect is exacerbated when the collision detection is turned on, because the collision forces impact larger areas of the body than they should due to the basis functions having large support. A remedy is to increase the resolution of the simulation, but such a simulation would probably not be interactive. One could also apply a more general adaptive simulation to the problem (the adaptation presented in Section 4.5 only adapts to the rig forces).

6. Conclusion

Our system gives animators control over the shapes of elastic deformable characters by introducing force fields as the building blocks of rigging. The resulting simulations combine the guidance of the animator with other influences such as gravity, physical constraints, and user interaction. A key component is a new method of approximating non-linear elasticity via a pose-dependent linear system, which enables efficient dynamic simulation, computation of static equilibria, and optimization; our system is interactive in a wide range of circumstances. Our system supports interactive rigging, computing rigs from surface deformations, and the transfer of rigs from one character to another.

A disadvantage of using forces to effect deformations is that in order to create finely detailed deformations, the dynamic simulation must be computed at a high resolution. To reduce computation in such cases, adding displacements to the dynamic shape might be preferable to using forces. It would be interesting to further explore this tradeoff.

Another possible extension to this work would be the interpolation of physical properties as a part of the rigging. In addition to forces, it would fit nicely into our framework to adjust the stiffness of part of the character as a function of rig parameters. It would also be interesting to try to extract such physical properties from captured data.

Acknowledgments

Thanks to Brett Allen, Keith Grochow, Daichi Sasaki, Supriyo, Ernest Wu, and Yeuhi Abe. This research was supported by NSF Grants EIA-0121326, CCR-0092970, IIS-0113007, and CCR-0098005, and by UW Animation Research Labs, an Alfred P. Sloan Fellowship, Electronic Arts, Sony, Microsoft Research, Alias, and Washington Research Foundation.

Appendix A. Supplementary data

Supplementary data associated with this article can be found, in the online version, at doi:10.1016/j.gmod.2006.09.001.

References

[1] Marc Alexa, Linear combination of transformations, in: ACM Transaction Graphics, Proceedings of SIGGRAPH 2002, vol. 21, No. 3, 2002, pp. 380–387.
[2] Brett Allen, Brian Curless, Zoran Popović, Articulated body deformation from range scan data, in: ACM Transactions on Graphics, Proceedings of SIGGRAPH 2002, vol. 21, No. 3, 2002, pp. 612–19.
[3] U.M. Ascher, E. Boxerman, On the modified conjugate gradient method in cloth simulation, Visual Computer 19 (7–8) (2003) 526–531.
[4] Amaury Aubel, Daniel Thalmann, Interactive modeling of the human musculature, in: Proceedings of Computer Animation 2001, 2001.
[5] David Baraff, Andrew Witkin, Dynamic simulation of non-penetrating flexible bodies, in: Computer Graphics, Proceedings of SIGGRAPH 92, vol. 26, No. 2, 1992, pp. 303–308.
[6] David Baraff, Andrew Witkin, Large steps in cloth simulation, in: Proceedings of SIGGRAPH 98, 1998, pp. 43–54.
[7] P.J. Besl, N.D. McKay, A method for registration of 3-D shapes, IEEE Transactions on Pattern Analysis and machine Intelligence 14 (2) (1992) 239–258.
[8] Richard H. Byrd, Peihuang Lu, Jorge Nocedal, Ciyou Zhu, A limited memory algorithm for bound constrained optimization, SIAM Journal on Scientific Computing 16 (6) (1994) 1190–1208.
[9] Steve Capell, Interactive Character Animation Using Dynamic Elastic Simulation, PhD thesis, University of Washington, Department of Computer Science and Engineering, 2004.
[10] Steve Capell, Seth Green, Brian Curless, Tom Duchamp, Zoran Popović, Interactive skeleton-driven dynamic deformations, in: ACM Transaction on Graphics, Proceedings of ACM SIGGRAPH 2002, vol. 21, No. 3, 2002.
[11] Gilles Debunne, Mathieu Desbrun, Marie-Paule Cani, Alan H. Barr, Dynamic real-time deformations using space & time adaptive sampling, in: Proceedings of SIGGRAPH 2001, 2001.
[12] Steven J. Gortler, Michael F. Cohen, Hierarchical and variational geometric modeling with wavelets, in: 1995 Symposium on Interactive 3D Graphics, 1995, pp. 35–42.
[13] G. Irving, J. Teran, R. Fedkiw, Invertible finite elements for robust simulation of large deformation, in: Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, 2004, pp. 131–140.
[14] Doug L. James, Dinesh K. Pai, Dyrt: dynamic response textures for real time deformation simulation with graphics hardware, in: Proceedings of SIGGRAPH 2002, 2002, pp. 582–585.

[15] Paul G. Kry, Doug L. James, Dinesh K. Pai. Eigenskin: real time large deformation character skinning in hardware, in: Proceedings of the 2002 ACM SIGGRAPH Symposium on Computer Animation, 2002, pp. 153–159.
[16] J.P. Lewis, Matt Cordner, Nickson Fong, Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation, in: Proceedings of SIGGRAPH 2000, 2000, pp. 165–172.
[17] Ron MacCracken, Kenneth I. Joy, Free-form deformations with lattices of arbitrary topology, in: Computer Graphics, Proceedings of SIGGRAPH 96, 1996, pp. 181–188.
[18] Alex Mohr, Michael Gleicher, Building efficient, accurate character skins from examples, in: ACM Transactions on Graphics, Proceedings of SIGGRAPH 2003, 2003, pp. 562–568.
[19] Matthias Müller, Julie Dorsey, Leonard McMillan, Robert Jagnow, Barbara Cutler, Stable real-time deformations, in: Proceedings of the 2002 ACM SIGGRAPH Symposium on Computer Animation, vol. 189, 2002, pp. 49–54.
[20] Matthias Müller, Markus Gross, Interactive virtual materials, in: Proceedings of Graphics Interface 2004, 2004, pp. 239–246.
[21] Victor Ng-Thow-Hing, Eugene Fiume, Application-specific muscle representations, in: Proceedings of Graphics Interface, 2002, pp. 107–116.
[22] Mark Pauly, Dinesh K. Pai, Leonidas J. Guibas, Quasi-rigid objects in contact, in: SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, ACM Press, 2004, pp. 109–119.
[23] John C. Platt, Alan H. Barr, Constraint methods for flexible models, in: Computer Graphics, Proceedings of SIGGRAPH 88, vol. 22, No. 4, 1988, pp. 279–288.
[24] Laks Raghupathi, Laurent Grisoni, François Faure, Damien Marchall, Marie-Paule Cani, Christophe Chaillou, An intestine surgery simulator: real-time collision processing and visualization, IEEE Transactions on Visualization and Computer Graphics 10 (6) (2004) 708–718.
[25] Ferdi Scheepers, Richard E. Parent, Wayne E. Carlson, Stephen F. May, Anatomy-based modeling of the human musculature, in: Proceedings of SIGGRAPH 97, 1997, pp. 163–172.
[26] Thomas W. Sederberg, Scott R. Parry, Free-form deformation of solid geometric models, Computer Graphics 20 (4) (1986) 151–160.
[27] K. Singh, J. Ohya, R. Parent, Human figure synthesis and animation for virtual space teleconferencing, in: Proceedings of the Virtual Reality Annual International Symposium, 1995, pp. 118–126.
[28] Karen Singh, Eugene Fiume, Wires: a geometric deformation technique, in: Proceedings of ACM SIGGRAPH 98, 1998, pp. 405–414.
[29] Peter-Pike J. Sloan, Charles F. Rose III, Michael F. Cohen, Shape by example, in: Proceedings of 2001 Symposium on Interactive 3D Graphics, 2001, pp. 135–143.
[30] Joseph Teran, Eftychios Sifakis, Silvia S. Blemker, Victor Ng-Thow-Hing, Cynthia Lau, Ronald Fedkiw, Creating and simulating skeletal muscle from the visible human data set, IEEE Transactions on Visualization and Computer Graphics 11 (3) (2005) 317–328.
[31] Demetri Terzopoulos, John Platt, Alan Barr, Kurt Fleischer, Elastically deformable models, in: Computer Graphics, Proceedings of SIGGRAPH 87, vol. 21, No. 4, 1987, pp. 205–214,
[32] Demetri Terzopoulos, Andrew Witkin, Physically based models with rigid and deformable components, IEEE Computer Graphics and Applications 8 (6) (1988) 41–51.
[33] Jane Wilhelms, Allen Van Gelder, Anatomically based modeling, in: Proceedings of SIGGRAPH 97, 1997, pp. 173–180.
[34] Andrew Witkin, William Welch, Fast animation and control of nonrigid structures, in: Computer Graphics, Proceedings of SIGGRAPH 90, vol. 24, No. 4, 1990, pp. 243–252.