To appear in the ACM SIGGRAPH conference proceedings

Musculotendon Simulation for Hand Animation

Shinjiro Sueda Andrew Kaufman Dinesh K. Pai

Sensorimotor Systems Laboratory
University of British Columbia*

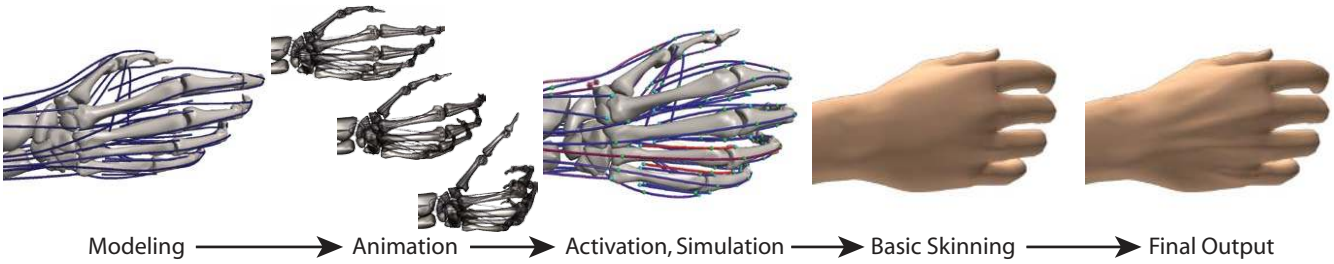


Figure 1: *Pipeline: The user specifies the model and its corresponding animation. Our system computes the required activations, and simulates the muscles, tendons, and bones. The skin is then attached to the skeleton, and the subcutaneous deformation from tendon motion is added as a post-process.*

Abstract

We describe an automatic technique for generating the motion of tendons and muscles under the skin of a traditionally animated character. This is achieved by integrating the traditional animation pipeline with a novel biomechanical simulator capable of dynamic simulation with complex routing constraints on muscles and tendons. We also describe an algorithm for computing the activation levels of muscles required to track the input animation. We demonstrate the results with several animations of the human hand.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation I.6.8 [Simulation and Modeling]: Types of Simulation—Animation

Keywords: musculoskeletal simulation, character animation, secondary motion

1 Introduction

Human bodies are more than skin and bones. When the body moves, tendons and muscles move under the skin in visually important ways that are correlated with both the movement and the internal forces. For example, the appearance of tendons on the back of the hand is related to how the hand is moving and how much force it is exerting. Even though there has been some work on incorporating muscles into animations (see Sec. 2 for a review) it has been very difficult to incorporate biomechanically realistic subcutaneous movements into a traditional animation pipeline. Integration with a traditional character animation pipeline is important since, unlike secondary motion of inanimate objects, the movements of characters are of central importance to the story and are typically hand crafted by expert animators.

In this paper we describe an automatic technique for generating biomechanically realistic secondary motion that fits into a traditional animation pipeline. In our system, animators are given a rigged character, and animate it as they normally would, for instance, using key frame techniques or motion capture. The animation is then exported to our efficient biomechanics simulator, based on Lagrangian dynamics. We constrain the animated bones to follow the animator's chosen motion path. We perform an optimization to determine the appropriate activation levels for each muscle throughout the animation. We then export the resulting physically based animations of musculotendons from our simulator and import them back into the rigged animation scene. The tendons are automatically skinned to the character's surface skin, providing complex secondary motion of the skin as a result of biomechanically realistic tendon motion.

Our method works well with areas where muscles and tendons are near the surface with little subcutaneous fat. We show the effectiveness of the method by simulating the musculotendons of the human hand and forearm.

Our main contributions are:

- A novel, efficient biomechanical simulator that can simulate thin strands including tendons and muscles, with complex routing constraints.
- Seamless integration of biomechanically realistic secondary animation into a traditional animation pipeline.
- An incremental algorithmic controller that determines the muscle activation levels required to dynamically track a target animation.

2 Related Work

We will first review several common muscle models used in both the computer graphics and biomechanics communities; we also briefly cover work related to strand simulation. Then we will discuss various methods for musculoskeletal control.

*e-mail:{sueda, akaufman, pai}@cs.ubc.ca

2.1 Muscle Simulation

Much of the work in musculoskeletal models has its roots in the biomechanics community [Delp and Loan 1995; Hogfors et al. 1995; Delp and Loan 2000; Pandy 2001; Maural et al. 1996; Chao 2003; Blemker and Delp 2005; Rasmussen et al. 2005; Wu et al. 2007]. With a few exceptions, muscles are modeled by simple lines of force that can bend around kinematic wrapping surfaces. None have been able to represent the complex tendon routing constraints that we demonstrate in this paper.

There has also been significant development of musculoskeletal models in the graphics community. Some have focused only on the muscle anatomy, and not on dynamic simulation [Scheepers et al. 1997; Wilhelms and Gelder 1997; Albrecht et al. 2003]. Ng-Thow-Hing [2001] and Teran et al. [2003; 2005] developed volumetric muscle models to simulate muscles with both active and passive components. Zhu et al. [1998] use a linear elastic muscle model along with finite elements for muscle volume deformation. Maural et al. [1996] and Aubel and Thalmann [2001] constructed muscle-based virtual human characters. Musculoskeletal models have also been used extensively for facial animation [Waters 1987; Lee et al. 1995; Sifakis et al. 2005]. Shao and Ng-Thow-Hing [2003] presented a general joint component framework designed to improve the realism of joint articulation in virtual humans. Lee and Terzopoulos [2006] used a neuromuscular control model for the simulation of a human neck. Zordan et al. [2004] developed muscle elements based on springs for simulated respiration.

Our muscle-tendon model relies on simulating thin structures we call strands. There have been several papers in the graphics literature investigating the use of thin physical structures for simulation of various phenomena [Qin and Terzopoulos 1996; Remion et al. 1999; Pai 2002; Grinspun et al. 2003; Lenoir et al. 2004; Coleman and Singh 2006; Bertails et al. 2006; Spillmann and Teschner 2007]. Our model extends these with new constraints and activation.

2.2 Muscle Control

There has been significant work in developing algorithmic controllers for physically based animation, without taking muscles into account [Faloutsos et al. 2001; Pollard and Zordan 2005]. Some controllers are able to achieve high realism by incorporating motion and force capture data [Yin et al. 2003; Zordan et al. 2005; Kry and Pai 2006].

Among the papers that take muscles explicitly into account and solve for their control signals, many use joint moment-arms, which are commonly used biomechanical approximation of distributed muscle forces around joints [Thelen et al. 2002; Tsang et al. 2005; Thelen and Anderson 2006]. Sifakis et al. [2005] determine activations of a detailed non-linear, but quasistatic, FEM muscle model. Weinstein et al. [2008] use a novel approach to PD control of muscles to track an animation. Lee and Terzopoulos [2006] use neural networks to learn to control the complex musculature of the neck.

3 Strand Based Musculoskeletal Simulation

Our simulator is built on two primitives: rigid bodies for bones and spline-based strands for tendons and muscles. Our decision to use strands was motivated by the anatomical structure of real muscle tissue. Muscles consist of fibers, curved in space, which are bundled into groups called fascicles. When a muscle is activated, the fibers contract, which transmits a contractile force directly along each fiber. By using strands in our muscle simulation, we are able to directly model this behavior. Strands allow us to define smooth curves to represent tendons, muscles, or the individual fascicles of each muscle. The forces applied to the strands can be transmitted directly along the curve, and also laterally through constraints.

We extend the physically-based spline models previously used in computer graphics [Qin and Terzopoulos 1996; Remion et al. 1999; Lenoir et al. 2004] to include activation (Sec. 3.2), simple yet robust sliding and surface constraint models (Sec. 3.3), and implicit integration with Rayleigh damping (Sec. 3.4).

A strand, containing $n \geq 4$ control points, has $3n$ degrees of freedom, corresponding to the x, y, and z coordinates of the n control points. The state of the system is given by the stacked positions and velocities of the rigid bodies and strand control points. These are the generalized coordinates and velocities of the system, respectively:

$$\begin{aligned} \chi &= \begin{bmatrix} \cdots & \mathsf{E}_i & \cdots & \boldsymbol{q}_j & \cdots \end{bmatrix}^T \\ \Phi &= \begin{bmatrix} \cdots & \phi_i & \cdots & \dot{\boldsymbol{q}}_j & \cdots \end{bmatrix}^T . \end{aligned} \tag{1}$$

Here, $\mathsf{E}_i \in SE(3)$ and $\phi_i \in se(3)$ are the configuration and the spatial velocity, respectively, of the i^{th} rigid body ($SE(3)$ is the space of 3D positions and orientations, and $se(3)$ is the space of translational and rotational velocities), and $\boldsymbol{q}_j \in \mathbb{R}^3$ and $\dot{\boldsymbol{q}}_j \in \mathbb{R}^3$ are the position and the velocity of the j^{th} spline control point.

For each generalized coordinate, we construct an impulse-momentum equation which, when discretized at the velocity level, is

$$M\Phi^{(k+1)} = M\Phi^{(k)} + hf - G^T\lambda, \tag{2}$$

where M is the block-diagonal generalized mass matrix of rigid bodies and strand control points [Remion et al. 1999], h is the step size, f is the generalized force (body forces for rigid bodies, elastic and damping forces for strands, etc., Sec. 3.2), and $G^T\lambda$ is the constraint force (Sec. 3.3).

3.1 Rigid Bodies and Joints

We can treat bones as rigid, since their deformation is not important for our purposes. The world position and velocity of a point on a rigid body at a local coordinate $\boldsymbol{r}$ are given by

$$\begin{aligned} \boldsymbol{x} &= \mathsf{E}\,\boldsymbol{r} \\ \dot{\boldsymbol{x}} &= \mathsf{E}^{-T}\,\Gamma(\boldsymbol{r})\,\phi, \end{aligned} \tag{3}$$

where E is the usual coordinate transformation matrix and the 3×6 matrix, $\Gamma = (-[\boldsymbol{r}]\ I)$, transforms the local spatial velocity of the rigid body, ϕ, into the velocity of a local point, $\boldsymbol{r}$, on the rigid body in local coordinates. The 3×3 matrix, $[\boldsymbol{r}]$, is the cross-product matrix such that $[\boldsymbol{r}]\boldsymbol{v} = \boldsymbol{r} \times \boldsymbol{v}$. Joint constraints are implemented using the adjoint formulation [Murray et al. 1994], with which we can easily derive different types of joints by simply dropping rows in the 6×6 adjoint matrix. For example, we drop the top three rows (corresponding to the three rotational DoFs) for a ball joint, or the third row (corresponding to the rotation about the z-axis) for a hinge joint.

3.2 Strand Dynamics

The path of a strand is described by a cubic B-spline curve,

$$\boldsymbol{p}(s,t) = \sum_{i=0}^{3} b_i(s)\boldsymbol{q}_i(t), \tag{4}$$

where $\boldsymbol{q}_i(t)$ denote the control points of the strand, with velocities $\dot{\boldsymbol{q}}_i(t)$. The cubic B-spline basis functions, $b_i(s)$, depend on where

the point is along the spline. Although a strand can have an arbitrary number of control points, a point on a strand only depends on four control points, due to the local support of the B-spline basis. The velocity and the tangent vectors of a point $\boldsymbol{p}(s,t)$ can be obtained in a similar manner.

$$\begin{aligned}\dot{\boldsymbol{p}}(s,t) &\equiv \frac{d\boldsymbol{p}}{dt} = \sum_{i=0}^{3} b_i(s)\dot{\boldsymbol{q}}_i(t) \\ \boldsymbol{p}'(s,t) &\equiv \frac{\partial \boldsymbol{p}}{\partial s} = \sum_{i=0}^{3} b_i'(s)\boldsymbol{q}_i(t).\end{aligned} \tag{5}$$

Based on these quantities, we can compute the passive and active elastic forces in the strand (which contribute to f in Eq. (2)). Our simulator has the ability to use an arbitrary Force-Length (FL) relationship, which can be obtained from a standard Hill-type model [Zajac 1989] or from physiological experiments. However, constitutive properties of muscles are still not well established and are the subject of intense ongoing research. For graphics applications, we use linear FL curves for both the passive and active forces, since they work just as well for producing realistic animations. The active force of a muscle is modeled by shifting the FL curve upwards, so that the resulting stress is higher at each length and becomes zero at a shorter length.

The dynamics equation for the control points of the strands is given by

$$M\dot{\boldsymbol{q}}^{(k+1)} = M\dot{\boldsymbol{q}}^{(k)} + h(\boldsymbol{f}_d + \boldsymbol{f}_g + \boldsymbol{f}_p + \boldsymbol{f}_a) - G^T\lambda, \tag{6}$$

where M is the mass matrix, $\boldsymbol{f}_d$ is the Rayleigh damping force, $\boldsymbol{f}_g$ is the gravity force, and $\boldsymbol{f}_p$ is the passive elastic force. The active force, $\boldsymbol{f}_a$, is linear in the activation levels, and can be expressed as a matrix-vector product, Aa, where a is the vector of muscle activation levels between 0 (no activation) and 1 (full activation), and A, which is of size (#DoF $\times$ #muscles), is the "activation transport" matrix that takes as input the activations of the muscles and outputs the corresponding forces on the strand DoFs, by scaling the activations as a function of the local strain and spline blending functions. This separation of force and activation will help us later when deriving the controller in Sec. 4. The last term, $G^T\lambda$, is the constraint force term, which will be discussed in Sec. 3.3.

We use Rayleigh damping given by

$$\boldsymbol{f}_d = \left(\alpha M + \beta\frac{\partial \boldsymbol{f}^T}{\partial \boldsymbol{q}}\right)\dot{\boldsymbol{q}}, \tag{7}$$

where $\boldsymbol{f}$ is the cumulative force (excluding $\boldsymbol{f}_d$) from Eq. (6) and α and β are positive damping parameters.

3.3 Constraints

Constraints are required for musculotendon origins/insertions and for tendon routing. Although wrapping surfaces implemented in biomechanical simulators [Delp and Loan 2000; Garner and Pandy 2000] are effective for kinematic constraints, they do not work for dynamic constraints, and are also limited to simplified geometries, such as spheres and cylinders.

Tendon routing is particularly difficult, and ignored by existing biomechanical simulators. We have two types of constraints for tendon routing: sliding and surface constraints. A sliding constraint is useful when the strand is to pass through a specific point in space. Surface constraints are used to allow the strand to slide laterally on the surface as well.

Although our simulator is a general multi-body simulator with deformable strands, there is one assumption in our application that simplifies the constraint formulation. Since tendons and muscles stay in contact with surrounding tissue and do not come apart, we deal only with equality constraints; inequality constraints, which are more difficult to solve numerically, do not need to be modeled. In addition, no general-purpose proximity detection is required. Because strands are based on spline curves, keeping track of contacting points is computationally inexpensive. Potential contact points are first predetermined along each strand. After each time step the closest points are updated using Newton-Raphson search. Contact points on rigid bodies for surface constraints are tracked in a similar manner, by first wrapping each rigid body with a cubic tensor-product surface.

The constraints in our system are formulated at the velocity level. Let $g(\chi)$ be a vector of position-level equality constraint functions, such that when each constraint i is satisfied by the generalized coordinates, χ, $g_i(\chi) = 0$. By differentiating g with respect to time, we obtain a corresponding velocity-level constraint function that is consistent with our discretization.

$$\frac{d\,g(\chi)}{dt} = \frac{\partial\, g(\chi)}{\partial \chi}\Phi = 0. \tag{8}$$

Denoting the gradient of g by the constraint matrix G, we obtain the constraint equation $G\Phi = 0$.

This constraint equation may allow the system to drift away from the constraint manifold because it is formulated at the velocity, not position level. We add a stabilization term [Baumgarte 1972] to help correct this drift by pushing the system back towards the constraint manifold. The stabilized velocity constraint equation is then

$$G\Phi = -\mu g, \tag{9}$$

where μ is the stabilizer weight. If there is no positional error ($g = 0$), then the constraint equation is exactly $G\Phi = 0$. On the other hand, if there is a small positional error ($g \neq 0$), then a non-zero stabilization force, $-\mu g$, will be added to push the system back to the constraint manifold. For critical damping, we set $\mu = 1/h$, so that unnecessary oscillations are minimized.

Fixed Constraints: We use fixed constraints for strand origins and insertions, as well as for attaching several strands to form branching structures. For example, if we want to constrain a point on a strand, $\boldsymbol{p}$, to a point on a rigid body, $\boldsymbol{p}_0$, we can set their relative velocities to be equal.

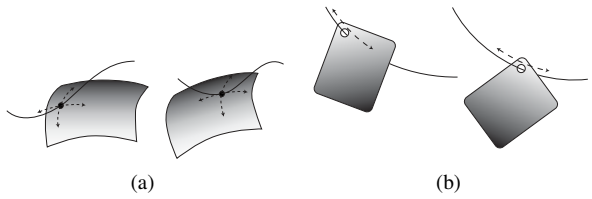
$$\dot{\boldsymbol{g}} = \dot{\boldsymbol{p}} - \dot{\boldsymbol{p}}_0, \tag{10}$$

where both $\dot{\boldsymbol{p}}$ and $\dot{\boldsymbol{p}}_0$ are linear with respect to the DoFs of the system, as given by Eqs. (3) and (5).

Surface Constraints: Surface constraints are similar to the usual rigid body contact constraints. A point on a strand is constrained to lie on a point on the surface of a rigid body. Let $\boldsymbol{p}$ denote the 3D position of the strand point to constrain and $\boldsymbol{p}_0$ and $\boldsymbol{n}$ its corresponding contact point and normal on the rigid body. Equating the relative velocities along the normal gives

$$\dot{g} = \boldsymbol{n}^T(\dot{\boldsymbol{p}} - \dot{\boldsymbol{p}}_0). \tag{11}$$

The constraint point on the strand is fixed, whereas the point on the surface is updated before each step by finding the closest point on the tensor-product surface attached to the rigid body (Fig. 2(a)).



(a) (b)

Figure 2: *(a) Surface constraint and (b) sliding constraint. With a surface constraint, the constraint point moves on the rigid body surface, whereas with a sliding constraint, the constraint point moves along the strand.*

Sliding Constraints: In most situations, such as in the carpal tunnel, tendons are confined to slide axially but not laterally. We can achieve this behavior by adding an additional dimension to the surface constraint. Given the tangent vector of the point to constrain on a strand, we generate the normal, $\boldsymbol{n}_1$, and binormal, $\boldsymbol{n}_2$, vectors, and apply the constraint with respect to both of these vectors.

$$\begin{aligned} \dot{g}_1 &= \boldsymbol{n}_1^T(\dot{\boldsymbol{p}} - \dot{\boldsymbol{p}}_0) \\ \dot{g}_2 &= \boldsymbol{n}_2^T(\dot{\boldsymbol{p}} - \dot{\boldsymbol{p}}_0). \end{aligned} \tag{12}$$

Unlike the surface constraint, the constraint point on the rigid body is fixed, whereas it is allowed to slide on the strand (Fig. 2(b)).

3.4 Integration

We must be careful when choosing a numerical integrator to step the system forward in time. Our choice will have an impact on the efficiency, stability, and accuracy of the system. We currently use a semi-implicit Euler stepping scheme, because of its efficiency and stability. Although our system is stable for large step sizes, we must restrict the step size to avoid unacceptable numerical damping introduced by the implicit method. In the rest of this section, we describe our stepping scheme in detail. It is important to note, however, that our system is not limited to this integrator.

The semi-implicit Euler scheme [Hairer and Wanner 2004] for nonlinear systems is a variant on the fully implicit Euler scheme, where only one iteration of Newton search is taken per time step, rather than iterating until convergence. The rationale is that for small enough step sizes, the integrator will converge to the true solution quickly over multiple steps. This amounts to adding a linear Taylor expansion term about the current term and augmenting the mass matrix with the gradients of implicit elastic and damping forces. Combining the dynamics equation (Eq. (2) with the modified mass matrix, Eq. (7)) and the constraint equation (Eq. (9)), we obtain a linear system (called a KKT system, [Boyd and Vandenberghe 2004]) that we solve at each time step to obtain the generalized velocities at the next step.

$$\begin{pmatrix} M & G^T \\ G & 0 \end{pmatrix} \begin{pmatrix} \Phi^{(k+1)} \\ \lambda \end{pmatrix} = \begin{pmatrix} M\Phi^{(k)} + hf \\ -\mu g \end{pmatrix}, \tag{13}$$

where λ is the vector of Lagrange multipliers for the constraints. We use a direct method based on Gaussian Elimination [Davis 2006] to solve this matrix.

Once the new velocities, $\Phi^{(k+1)}$, are computed, the new positions, $\chi^{(k+1)}$, are trivially computed, for example, using Rodrigues' formula.

4 Controller

The subcutaneous movement of tendons and muscles depends on how those muscles were activated to achieve the desired movement of the character. Because of the complexity of the routing of tendons and muscles, even simple tasks, such as moving a finger from one position to another, requires the coordinated activation of several muscles, acting as synergists and antagonists. It is a virtually hopeless task to attempt control of such a system by hand (for example with a GUI) — an algorithmic controller is needed.

For the purpose of our simulation, a controller's job is to compute the activation levels of the muscles, given some target movement of parts of the skeleton. The simulation loop can be summarized by this simple pseudocode:

REPEAT
Compute activation levels, a.
Compute new velocities, Φ.
Update positions, χ. (14)

The controller computes the activation levels, a, from a set of target velocities specified by the user, for example, from motion-capture data, key-framed animations, or even from other skeletal controllers.

For brevity, we will use the following notation for the constrained dynamics equation:

$$\underbrace{\begin{pmatrix} M & G^T \\ G & 0 \end{pmatrix}}_{\tilde{M}} \underbrace{\begin{pmatrix} \Phi^{(k+1)} \\ \lambda \end{pmatrix}}_{\tilde{\Phi}} = \underbrace{\begin{pmatrix} (M\Phi^{(k)} + hf) + hAa \\ -\mu g \qquad\qquad +0 \end{pmatrix}}_{\tilde{f}+\tilde{A}a}. \tag{15}$$

The dimensionality of $\tilde{\Phi}$ is the number of DoFs in the system *plus* the number of constraints. Note that here, we have extracted out the active force, $f_a = Aa$, which contains the activation levels, a, that we are trying to solve for.

The reference animation specifies the target velocities of rigid bodies, v_x. This can be either a 3D point velocity or a 6D spatial velocity, and the total size of v_x is (6 × #spatial targets + 3 × #point targets). If the input target is a sequence of rigid body configurations rather than velocities, it can be converted into the required form by computing the spatial velocities needed to move the bodies from their current configurations to the target configurations in a time step, using the matrix logarithm.

We then require that the controller computes the activations of the muscles such that the resulting system velocities match the desired velocities. That is,

$$\Gamma_x \tilde{\Phi} = v_x, \tag{16}$$

where the entries of the Jacobian Γ_x contain the 6×6 identity matrix for spatial velocity targets and the 3×6 matrix Γ from Eq. (3) for point velocity targets. Substituting Eq. (15) into Eq. (16), we arrive at the *targeting constraint equation*

$$H_x a + v_f = v_x, \tag{17}$$

where $H_x = \Gamma_x \tilde{M}^{-1} \tilde{A}$ and $v_f = \Gamma_x \tilde{M}^{-1} \tilde{f}$. The matrix H_x can be thought of as the effective inverse inertia experienced by the muscle activation levels in order to produce the target motion. The "free velocity" vector, v_f, is the velocity of the targets due to the non-active forces acting on the system. Thus, we are looking for the activation levels, a, that zero out the difference between the desired target velocities and the sum of active and passive velocities.

In some cases, it is possible to solve for the activations using the targeting equation (17) as a hard constraint. However, in most cases,

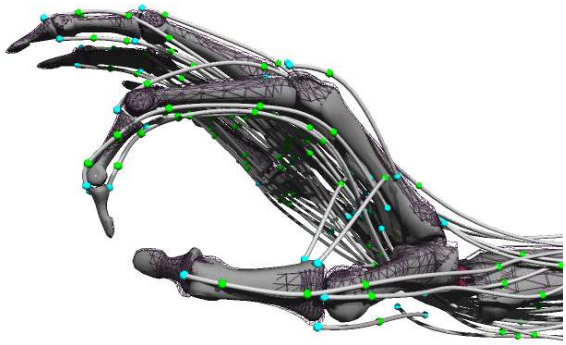


Figure 3: *A screenshot of the simulator. Fixed constraints are shown in cyan, sliding constraints in green, and surface constraints in maroon. Surface constraints allow the strands to move axially as well as laterally. The input animation target is shown in wireframe.*

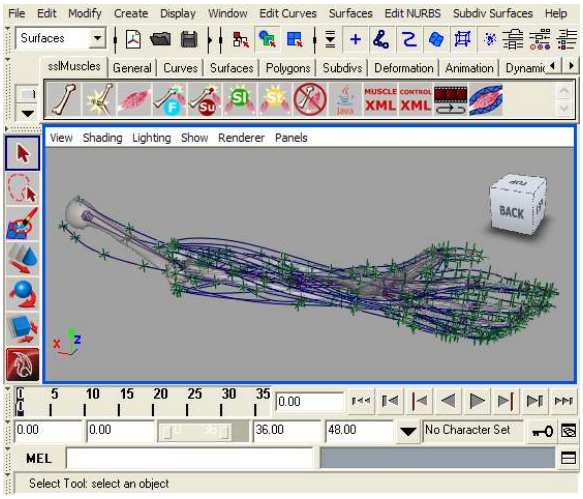


Figure 4: *Maya interface with our custom shelf. Strands are shown in blue, and constraints are shown in green.*

the dynamics of the system cannot *exactly* follow the requested targets. For example, the bone joint constraints may prevent certain poses, or muscles may not be strong enough to produce the required force. Instead, we convert the targeting equation into a quadratic minimization problem.

To make the dynamics follow the target trajectory as closely as possible, the controller may sometimes return activations that switch spastically. In order to prevent this, we add a damping term to the objective function, using the linear approximation of the derivative of the activation. In addition, to minimize the total activation, we add an "activation energy" term to the objective. Putting this all together, the activation optimization problem is

$$\begin{aligned} \min_{a} \quad & w_a\|a\|^2 + w_x\|(H_x a + v_f) - v_x\|^2 + w_d\|a - a^0\|^2 \\ \text{s.t.} \quad & 0 \le a \le 1, \end{aligned} \tag{18}$$

where w_a, w_x, and w_d are the blending weights, and a^0 is the activation from the previous time step. The first term minimizes the total activation and also adds regularization to the quadratic problem, whereas the second and third terms function as a spring-and-damper controller to guide the dynamics of the system towards the target motion. For easy motions that are not over-constrained, w_a and w_d can be set to zero to achieve close tracking of the target. However, for some motions, especially those that involve configurations that are almost singular, such as full flexion or extension, regularization (w_a) and damping (w_d) help stabilize the solution. For the animations in this paper, we used $(w_a, w_x, w_d) = (1, 1, 0.01)$.

Although the original dynamics equation can be large (albeit very sparse), the dimensionality of the quadratic minimization is the number of muscle strands, which in general is small (< 100). The main computational cost is in the construction of the quadratic matrix, which requires $min(\#targets, \#muscles)$ sparse solves on the KKT matrix from Eq. (13). However, since the LU factors of the KKT matrix are required for solving the system dynamics anyway, the only additional cost is the backsolve, which is relatively cheap. In practice, we note that the controller adds no significant computational cost to the overall system.

5 Pipeline

We have integrated our muscle simulator into a standard animation pipeline in order to demonstrate its effectiveness in a production environment (Fig. 4). Although the simulator could be used in conjunction with any 3D animation software, we have chosen to implement a plug-in for Maya (Autodesk, Inc., San Rafael, CA).

Our implementation makes some important design choices. First, we clearly separate the secondary motion simulation from the character animation, so that the animators can perform most of their work in the usual way. Only the rigger (as opposed to the animator) needs to be aware of the presence of a muscle simulation underneath the hood. Second, we provide tools to make it easy to add strands and edit their properties using GUIs in Maya. Finally, the effect of the subcutaneous strand motion on the skin is performed as a post-process to the normal skinning, and can be considered as an animation pass to add secondary motion.

From an animator's point of view, there are three main differences with a standard animation pipeline. Please see the accompanying video to see how this works (Fig. 1).

First, an animation rig is constructed in the usual way, but in addition, strands are placed within the character. The strands are built using the standard spline tools available in Maya. Though not a requirement, it is much easier to place the strands if the bone and muscle meshes are available. A GUI allows easy setting and editing of physical parameters such as mass, density, and dimensions. In addition to these simple scalar parameters, the proper constraints must be added to ensure that the strands function appropriately. Constraints can be easily added through our plug-in by selecting the appropriate strands and specifying the normalized parameter point to which the constraint should be added. After the strands have been created, they never need to be manually manipulated again.

The rigger also skins the standard rig in the normal way. This will allow the animators to continue their work unaffected by the addition of muscle strands. Along with this skinned base mesh, there is another version of the skin mesh, which will be deformed by the muscle strands. We will call this additional mesh the deformed mesh. The deformation process is completely automated in our plug-in. We automatically determine the skin mesh vertices that are affected by each strand by using proximity computations. Although we have added an additional skin mesh to each character, we do not foresee any problems since it is already common practice to include several different resolution meshes with each character.

Second, the animators construct a reference animation as they normally would, adding life to the characters in their scenes. Once

the character has been animated to their liking, the next step is to add the secondary motion provided by the muscles. The plug-in exports the models, muscle data, and animation keyframes to XML files. The muscle simulator then uses these XML files to recreate the Maya scene within the simulation software. The simulator calculates the dynamic activation levels for the muscles that best match the animation keyframes provided (Sec. 4). The simulation is then saved as keyframes in an XML file, which are loaded from our Maya plug-in and baked onto the strand control points. Now the Maya strands will move in a biomechanically accurate way.

The simulated rigid body motion is similar, but not identical, to the input skeletal motion. In the example animations, the average and worst per-vertex reconstruction errors are around 1mm and 5mm respectively. These reconstructions errors are actually automatic corrections of physically unattainable motions and configurations. Nevertheless, once the simulation is completed, the animator can choose to use the new skeletal animation or to stick with the original input animation of the rigid bodies, and remap the tendon motion back to the original motion.

Finally, once a strand animation has been imported, the animator can use the plug-in to calculate the skin deformation that corresponds to the given strand motion. Skin deformation due to subcutaneous strands is implemented as a post-process that offsets the skin mesh based on the proximity to strands. The degree of influence of the strands, and hence their visual prominence, can be controlled by the animator to produce a range of effects.

The skin deformation algorithm is as follows. Every base mesh vertex is given a scalar influence weight. This allows us to control the amount of deformation that each deformed mesh vertex undergoes. In our system, these influence weights are painted onto the base mesh using a color set and Maya's Paint Vertex Color Tool. Only mesh vertices with positive influence weights are deformed.

At each frame, the closest strand point ($\boldsymbol{p}_s$) to each base mesh vertex ($\boldsymbol{p}_v$) is determined. Then the deformed mesh vertex is modified as follows:

$$\begin{aligned} \boldsymbol{d} &= \boldsymbol{p}_s - \boldsymbol{p}_v \\ h &= \max(\boldsymbol{d} \cdot \boldsymbol{n} + c, 0) \\ f &= a \, \exp\left(\frac{-\|\boldsymbol{d} - (\boldsymbol{d} \cdot \boldsymbol{n})\boldsymbol{n}\|^2}{2b^2}\right) \\ \boldsymbol{p}_v &= \boldsymbol{p}_v + (w \, h \, f)\boldsymbol{n}, \end{aligned} \tag{19}$$

where $\boldsymbol{n}$ is the outward vertex normal and w is the vertex influence weight. The parameter a controls the height of the offset, b controls the falloff factor in the lateral direction, and c controls the falloff factor in the normal direction. Thus, when strands protrude above (or lie just beneath) the base skin, each vertex of the deformed skin is moved along its normal by an amount proportional to its distance from the strand. The level of deformation is controllable by the influence weights, and by tweaking the height and falloff parameters.

6 Results

The hand model used in our examples contains 54 musculotendons and 17 bones. The bone and muscle meshes were purchased from Snoswell Design in Adelaide, and the musculotendon paths were constructed based on standard textbook models in the literature [Moore and Dalley 1999]. The skeleton was rigged and animated by an artist, and the resulting animations were imported into our simulator, implemented in Java. The activations for an animation sequence of several seconds were computed within a few minutes.

The degrees of freedom of the skeleton are flexion/extension and abduction/adduction of the fingers, thumb, and wrist, and pronation/supination of the forearm. Our controller was able to produce motion involving all of these ranges of motion (Fig. 5). Subcutaneous motions are most pronounced for the extensor and abductor tendons of the thumb (Fig. 6), and the extensor digitorum tendons on the back of the palm (Fig. 7). Our technique correctly captures the deformation of the skin on the back of the hand during finger extension (Fig. 8). We can generate more subtle deformation by varying the skinning parameters (Fig. 6(c)). Finally, we validate our technique by comparing the simulated tendons of the thumb to several real thumb photographs (Fig. 9).

(a) (b) (c)

Figure 5: *Stillshots from an animation showing pronation/supination of the forearm, as well as abduction/adduction of the wrist, computed at interactive rates using our controller.*

(a) (b) (c)

Figure 6: *A frame from the thumb animation, before applying our method (a), and with varying levels of skin deformation (b and c), showing the "anatomical snuffbox".*

(a) (b)

Figure 7: *(a) Base skin. (b) With our method applied, showing tendons on the back of the hand realistically deforming the skin.*

7 Conclusion and Future Work

We have developed a method for efficient biomechanical simulation of subcutaneous tendons and muscles. The simulation is incorporated into a traditional animation pipeline and can automatically

(a) (b) (c)

Figure 8: *Animation showing tendons deforming the skin during extension.*

Figure 9: *We compare the simulated tendons of the thumb to several real thumb photographs.*

generate secondary motion of the skin. We are able to produce dynamic simulations with complex routing constraints that were previously not tackled in either the graphics or the biomechanics communities. We have also developed a novel controller that automatically computes the muscle activation levels, given some target movement of the skeleton. We demonstrated the effectiveness of our approach by simulating the musculotendons of the human hand.

Other than the hands, our method should work well with areas where muscles and tendons are near the surface with little subcutaneous fat, such as the feet, neck, forearms, and hamstrings. However, more work needs to be done before it can be applied to areas of the body with volumetric muscles, such as the shoulder or the spine. We also plan to extend the controller to take into account neurally controlled muscle and joint impedances that are essential for manipulation [Kry and Pai 2006]. Control of a body's tone and stiffness is also an important feature of a physiologically-based animation system [Neff and Fiume 2002; Lee and Terzopoulos 2006].

Acknowledgements. We would like to thank Qi Wei, Danny Kaufman, and the anonymous reviewers for their helpful comments, Stelian Coros and David Levin for the use of their thumbs, and Karan Singh for advice on writing Maya plug-ins. This work was supported in part by Canada Research Chairs Program, Peter Wall Institute for Advanced Studies, NSERC, Canada Foundation for Innovation, BC KDF, MITACS, and Maya licences donated by Autodesk.

References

ALBRECHT, I., HABER, J., AND SEIDEL, H.-P. 2003. Construction and animation of anatomically based human hand models. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 98–109.

AUBEL, A., AND THALMANN, D. 2001. Interactive modeling of the human musculature. In *Proceedings of Computer Animation 2001*, 167–255.

BAUMGARTE, J. 1972. Stabilization of constraints and integrals of motion in dynamical systems. *Computer Methods in Applied Mechanics and Engineering 1* (June), 1–16.

BERTAILS, F., AUDOLY, B., CANI, M.-P., QUERLEUX, B., LEROY, F., AND LÉVQUE, J.-L. 2006. Super-helices for predicting the dynamics of natural hair. *ACM Trans. Graph. (Proc. SIGGRAPH) 25*, 3, 1180–1187.

BLEMKER, S. S., AND DELP, S. L. 2005. Three-dimensional representation of complex muscle architectures and geometries. *Annals of Biomedical Engineering 33*, 5 (May), 661–673.

BOYD, S., AND VANDENBERGHE, L. 2004. *Convex Optimization*. Cambridge University Press.

CHAO, E. Y. S. 2003. Graphic-based musculoskeletal model for biomechanical analyses and animation. *Medical Engineering & Physics 25*, 3 (April), 201–212.

COLEMAN, P., AND SINGH, K. 2006. Cords: Geometric curve primitives for modeling contact. *IEEE Computer Graphics and Applications 26*, 3, 72–79.

DAVIS, T. A. 2006. *Direct Methods for Sparse Linear Systems*. SIAM Book Series on the Fundamentals of Algorithms. SIAM.

DELP, S. L., AND LOAN, J. P. 1995. A graphics-based software system to develop and analyze models of musculoskeletal structures. *Comput. Biol. Med. 25*, 1, 21–34.

DELP, S. L., AND LOAN, J. P. 2000. A computational framework for simulating and analyzing human and animal movement. *Computing in Science & Engineering 2*, 5, 46–55.

FALOUTSOS, P., VAN DE PANNE, M., AND TERZOPOULOS, D. 2001. Composable controllers for physics-based character animation. In *Proceedings of ACM SIGGRAPH 2001*, 251–260.

GARNER, B., AND PANDY, M. 2000. The obstacle-set method for representing muscle paths in musculoskeletal models. *Comput Methods Biomech Biomed Engin 3*, 1, 1–30.

GRINSPUN, E., HIRANI, A. N., DESBRUN, M., AND SCHRĆDER, P. 2003. Discrete shells. In *Proceedings of 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 62–67.

HAIRER, E., AND WANNER, G. 2004. *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*, 3 ed., vol. 2. Springer.

HOGFORS, C., KARLSSON, D., AND PETERSON, B. 1995. Structure and internal consistency of a shoulder model. *Journal of Biomechanics 28*, 7 (July), 767–777.

KRY, P. G., AND PAI, D. K. 2006. Interaction capture and synthesis. *ACM Trans. Graph. (Proc. SIGGRAPH) 25*, 3, 872–880.

LEE, S.-H., AND TERZOPOULOS, D. 2006. Heads up!: biomechanical modeling and neuromuscular control of the neck. *ACM Trans. Graph. (Proc. SIGGRAPH) 25*, 3, 1188–1198.

LEE, Y., TERZOPOULOS, D., AND WALTERS, K. 1995. Realistic modeling for facial animation. In *Proceedings of ACM SIGGRAPH 1995*, 55–62.

LENOIR, J., GRISONI, L., MESEURE, P., RÉMION, Y., AND CHAILLOU, C. 2004. Smooth constraints for spline variational modeling. In *Proceedings of GRAPHITE 2004*, 58–64.

MAURAL, W., THALMANN, D., HOFFMEYER, P., BEYLOT, P., GINGINS, P., KALRA, P., AND THALMANN, N. M. 1996. A biomechanical musculoskeletal model of human upper limb for dynamic simulation. In *Proceedings of the 1996 Eurographics Workshop on Computer Animation and Simulation*, 121–136.

MOORE, K. L., AND DALLEY, A. F. 1999. *Clinically oriented anatomy*, 4 ed. Lippincott Williams & Wilkins.

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

NEFF, M., AND FIUME, E. 2002. Modeling tension and relaxation for computer animation. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 81–88.

NG-THOW-HING, V. 2001. *Anatomically-based models for physical and geometric reconstruction of humans and other animals*. PhD thesis, The University of Toronto.

PAI, D. K. 2002. STRANDS: Interactive simulation of thin solids using Cosserat models. In *Proceedings of Eurographics 2002*, 347–352.

PANDY, M. G. 2001. Computer modeling and simulation of human movement. *Annual Review of Biomedical Engineering*, 3, 245–273.

POLLARD, N. S., AND ZORDAN, V. B. 2005. Physically based grasping control from example. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 311–318.

QIN, H., AND TERZOPOULOS, D. 1996. D-NURBS: A Physics-Based Framework for Geometric Design. *IEEE Transactions on Visualization and Computer Graphics 2*, 1, 85–96.

RASMUSSEN, J., DAMSGAARD, M., CHRISTENSEN, S. T., AND DE ZEE, M. 2005. Anybody - decoding the human musculoskeletal system by computational mechanics. *Konferanse i beregningsorientert mekanikk (invited paper)*.

REMION, Y., NOURRIT, J., AND GILLARD, D. 1999. Dynamic animation of spline like objects. In *Proceedings of the 1999 WSCG Conference*, 426–432.

SCHEEPERS, F., PARENT, R. E., CARLSON, W. E., AND MAY, S. F. 1997. Anatomy-based modeling of the human musculature. In *Proceedings of ACM SIGGRAPH 1997*, 163–172.

SHAO, W., AND NG-THOW-HING, V. 2003. A general joint component framework for realistic articulation in human characters. In *Proceedings of the 2003 Symposium on Interactive 3D Graphics*, 11–18.

SIFAKIS, E., NEVEROV, I., AND FEDKIW, R. 2005. Automatic determination of facial muscle activations from sparse motion capture marker data. *ACM Trans. Graph. (Proc. SIGGRAPH) 24*, 3, 417–425.

SPILLMANN, J., AND TESCHNER, M. 2007. CoRdE: Cosserat rod elements for the dynamic simulation of one-dimensional elastic objects. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 63–72.

TERAN, J., BLEMKER, S., HING, V. N. T., AND FEDKIW, R. 2003. Finite volume methods for the simulation of skeletal muscle. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 68–74.

TERAN, J., SIFAKIS, E., BLEMKER, S. S., NG-THOW-HING, V., LAU, C., AND FEDKIW, R. 2005. Creating and simulating skeletal muscle from the visible human data set. *IEEE Transactions on Visualization and Computer Graphics 11*, 3, 317–328.

THELEN, D. G., AND ANDERSON, F. C. 2006. Using computed muscle control to generate forward dynamic simulations of human walking from experimental data. *Journal of Biomechanics*, 39, 1107–1115.

THELEN, D. G., ANDERSON, F. C., AND DELP, S. L. 2002. Generating dynamic simulations of movement using computed muscle control. *Journal of Biomechanics*, 36, 321–328.

TSANG, W., SINGH, K., AND FIUME, E. 2005. Helping hand: an anatomically accurate inverse dynamics solution for unconstrained hand motion. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 319–328.

WATERS, K. 1987. A muscle model for animation three-dimensional facial expression. In *Proceedings of ACM SIGGRAPH 1987*, 17–24.

WEINSTEIN, R., GUENDELMAN, E., AND FEDKIW, R. 2008. Impulse-based control of joints and muscles. *IEEE Transactions on Visualization and Computer Graphics 14*, 1, 37–46.

WILHELMS, J., AND GELDER, A. V. 1997. Anatomically based modeling. In *Proceedings of ACM SIGGRAPH 1997*, 173–180.

WU, F. T. H., NG-THOW-HING, V., SINGH, K., AGUR, A. M., AND MCKEE, N. H. 2007. Computational representation of the aponeuroses as nurbs surfaces in 3d musculoskeletal models. *Comput. Methods Prog. Biomed. 88*, 2, 112–122.

YIN, K., CLINE, M. B., AND PAI, D. K. 2003. Motion perturbation based on simple neuromotor control models. In *Proceedings of Pacific Graphics 2003*, 445–449.

ZAJAC, F. 1989. Muscle and tendon: properties, models, scaling, and application to biomechanics and motor control. *Crit Rev Biomed Eng. 17*, 4, 359–411.

ZHU, Q.-H., CHEN, Y., AND KAUFMAN, A. 1998. Real-time biomechanically-based muscle volume deformation using fem. *Computer Graphics Forum 17*, 3, 275–284.

ZORDAN, V. B., CELLY, B., CHIU, B., AND DILORENZO, P. C. 2004. Breathe easy: model and control of simulated respiration for animation. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 29–37.

ZORDAN, V. B., MAJKOWSKA, A., CHIU, B., AND FAST, M. 2005. Dynamic response for motion capture animation. *ACM Trans. Graph. (Proc. SIGGRAPH) 24*, 3, 697–701.