Interactive Tools for Biomechanical Modeling and Realistic Animation

by

Andrew Kaufman

B.Sc., Northwestern University, 2006

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE STUDIES
(Computer Science)

The University Of British Columbia
August 2008

Abstract

We describe a semi-automatic technique for modeling and animating complex musculoskeletal systems using a strand based muscle model. Using our interactive tools, we are able to generate 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 biomechanical simulator capable of dynamic simulation with complex routing constraints on muscles and tendons. We integrate our musculoskeletal modeling and animation toolkit into a professional 3D production environment, thereby enabling artists and scientists to create complex musculoskeletal systems that were previously inaccessible to them. We demonstrate the applications of our tools to the visual effects industry with several animations of the human hand and applications to the biomechanics community with a novel model of the human shoulder.

Table of Contents

List of Tables

List of Figures

Acknowledgments

First and foremost, I thank my supervisor, Dinesh K. Pai, without whom this research would not have been possible. He has provided me with guidance, support, and encouragement throughout my time at UBC. Perhaps most importantly, he has provided the freedom to conduct my research independently, enabling me to learn and grow into an accomplished researcher.

I am also thankful to Michiel van de Panne for his helpful comments and inspirational teaching style. I thank my partner in publication, Shinjiro Sueda, for showing me the ropes, or strands as it may be, and my fellow Sensorimotor students David Levin, Danny Kaufman, Qi Wei, and Sang Hoon Yeo for their suggestions, motivation, and friendship along the way.

Finally, I would like to thank my family for their constant encouragement and support. It has meant the world to me.

Flying through your powdered trees,
Swiftly floating on your seas,
Atop your peaks, my spirits lift,
My mind flowing in your breeze.

Rainy days have come and gone,
Yet summer skies still shine on,
And when my mind begins to drift
I dream of you, my sweet Dione.

Statement of Co-Authorship

Several sections of this thesis were adapted from a paper describing our modified animation pipeline and the results of our hand model [57]. Portions of these sections were written by Shinjiro Sueda and Dr. Dinesh K. Pai. All textual descriptions of this nature have been credited as such at the beginning of the relevant section. All reproduced figures have been noted in the corresponding captions. The sections and figures have been adapted and reproduced here since they are directly connected to the design decisions and results presented in this thesis.

Chapter 1

Introduction

1.1 Overview

There are several purposes for developing musculoskeletal models. These models could be used by animators to generate realistic animation for the visual effects industry. Even though there has been some work on incorporating muscles into animations (Sec. 2.1.2), 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.

Musculoskeletal models are also important to biologists and biomechanists, who use them to analyze the anatomy and physiology of living creatures. Doctors can use these models for medical evaluation, by providing accurate biomechanical analysis of the muscles and joints, or for educational

purposes, to help patients and future practitioners visualize what is actually happening in the muscular system.

We will first discuss the specific contributions of this thesis (Sec. 1.2), followed by background information on musculoskeletal modeling and simulation (Sec. 2.1), strand based simulation (Sec. 2.2), and prominent commercial modeling tools (Sec. 2.3). Next we will discuss the design decisions that were made in the development of our musculoskeletal modeling tools (Sec. 3). We will then demonstrate the effectiveness of our tools by discussing the results of two case studies: the first on the human hand (Sec. 4.1) and the second on the human shoulder (Sec. 4.2). Finally, we will summarize the outcome of this work and discuss the possibilities for future development of our musculoskeletal modeling and animation tools (Sec. 5).

1.2 Contributions

We have developed a set of interactive tools designed for the modeling and animation of musculoskeletal systems. The intention of these tools is to make the modeling of these complex systems accessible to artists, biologists, biomechanists, and other technical and non-technical users. These tools were designed specifically to be used in conjunction with the strand based musculoskeletal simulation software being developed by the Sensorimotor Systems Lab at the University of British Columbia.

Our musculoskeletal modeling and animation tools have been implemented as a plug-in embedded in a professional 3D production environment in order to leverage existing generic modeling, animation, and user interface tools. We have defined several new primitives in order to properly repre-

sent musculoskeletal systems. We have included tools for automating the musculoskeletal modeling and animation processes, and for interfacing with our external simulation software. Most importantly, we describe a modified character animation pipeline that semi-automatically generates the motion of tendons and muscles under the skin of a traditionally animated character.

Our musculoskeletal modeling and animation tools have applications in the both the animation and biomechanical communities. We demonstrate the visual effects applications of our tools with several animations of the human hand (Sec. 4.1) and the biomechanical applications with the details of a novel model of the human shoulder (Sec. 4.2). These case studies demonstrate that our tools have made musculoskeletal modeling accessible to artists and scientists alike, and that it is now possible for both communities to adopt a strand based muscle model, such as the model introduced by Sueda et al. [57].

Chapter 2

Background

We will first discuss the key challenges involved in musculoskeletal modeling (Sec. 2.1.1), followed by a review of several common muscle models used in both computer graphics and biomechanics (Sec. 2.1.2). Next we will explain the dynamics of strand based simulation (Sec. 2.2.1) and briefly cover work related to strand simulation as it applies to various natural phenomena (Sec. 2.2.2). We will take an in depth look at strand simulation for muscles and tendons, since it is critical for this work (Sec. 2.2.3). Finally, we will discuss various commercial modeling tools which are prominent in the animation and biomechanics communities (Sec. 2.3).

2.1 Musculoskeletal Modeling and Simulation

2.1.1 Key Challenges

In all aspects of biomechanical modeling, simplifications have to be made in order to form a controllable representation of the full system. A holistic

model needs representations of geometry, activations, and forces. Ideally these simplifications will not sacrifice the validity of the system. In order to ensure that a biomechanical model upholds these principles, it is necessary to study anatomy as well as the work of current biomechanical researchers.

The first significant challenge in the development of functional musculoskeletal models, and the focus of this work, is the development of usable tools that are capable of representing the complex systems we are trying to model. See Sec. 2.3 for a description of current commercial modeling tools, some of which have a musculoskeletal focus.

Static models may help for educational purposes, but they do little to help with animation and biomechanical analysis. For this reason, we also need to be concerned with the challenges of musculoskeletal simulators. These simulators have several key challenges that they need to account for. First, they need appropriate methods of bone articulation, such as the general joint component framework suggested by Shao and Ng-Thow-Hing [49], the joint-sinus cones suggested by Maurel and Thalmann [35], or the spline joints recently developed by Lee and Terzopoulos [30].

Second, each simulator needs to implement an appropriate model of muscle tissue. Most researchers model muscle forces using lines of action or centroidal lines. However, recent studies criticize this simplification, and suggest that it may be a significant source of error [72]. Charlton and Johnson [13] were along the right lines when they recently suggested a diversion from straight line muscle forces.

Finally, musculoskeletal simulators also have to handle the complicated topic of muscle control. Due to the incredible complexity of musculoskele-

tal systems, it is nearly impossible to manually control a simulated system. Several researchers have made attempts at automated control, using regression models compiled from experimental data [15, 22], or using optimization techniques to approximate muscle synergies [13, 21, 29, 70].

2.1.2 Existing Approaches

This section is adapted from [57] and portions of this description were written by Sueda and Pai. It is reproduced here since it is essential for the remainder of this thesis. Much of the work in musculoskeletal modeling and simulation has its roots in the biomechanics community [9, 12, 13, 16, 17, 21, 22, 36, 44, 70, 77]. With a few exceptions, muscles are modeled by simple lines of force that can bend around kinematic wrapping surfaces. Few have been able to represent the complex muscle and tendon routing constraints that are demonstrated by Sueda et al. [57].

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 [1, 48, 75]. Ng-Thow-Hing [40] and Teran et al. [58, 59] developed volumetric muscle models to simulate muscles with both active and passive components. Zhu et al. [80] use a linear elastic muscle model along with finite elements for muscle volume deformation. These finite element approaches are far more computationally intensive than our strand based approach. Maurel et al. [36] and Aubel and Thalmann [2] constructed muscle-based virtual human characters using line of force muscle models. Musculoskeletal models have also been used extensively for facial animation [31, 51, 73]. Lee and Terzopoulos [29] used a neuromuscular

control model with line of force muscles for the simulation of a human neck. Zordan et al. [81] developed muscle elements based on springs for simulated respiration.

There has been significant work in developing algorithmic controllers for physically based animation, based on using joint torques in place of muscle activation [18, 42]. Some controllers are able to achieve high realism by incorporating motion and force capture data [28, 78, 82].

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 approximations of distributed muscle forces around joints [63, 64, 67]. Sifakis et al. [51] determine activations of a detailed non-linear, but quasistatic, finite element (FEM) muscle model. Weinstein et al. [74] use a novel approach to proportional-derivative (PD) control of muscles to track an animation. Hoegfors et al. [21] solve for muscle activations using a minimization of the sum of squared muscle stresses. van der Helm [70] developed a finite element muscle model, using inverse dynamics, given bone motion and external forces, to calculate optimized muscle forces. Charlton and Johnson [13] use a least-squares optimization, along with upper and lower bounds on muscle stress, in order to solve the muscle load-sharing problem. Lee and Terzopoulos [29] use neural networks to learn to control the complex musculature of the neck.

2.2 Strand Based Simulation

2.2.1 Strand Dynamics

This section is adapted from [57] and portions of this description were written by Sueda. It is reproduced here since it is essential for the remainder of this thesis. Sueda et al. [57] simulate thin structures they call strands. The path of a strand is described by a cubic B-spline curve,

$$\boldsymbol{p}(s,t) = \sum_{i=0}^{3} b_i(s)\boldsymbol{q}_i(t), \tag{2.1}$$

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.

$$\begin{aligned} \dot{\boldsymbol{p}}(s,t) &\equiv \frac{d\boldsymbol{p}}{dt} = \sum_{i=0}^{3} b_i(s)\dot{\boldsymbol{q}}_i(t) \\ \boldsymbol{p}'(s,t) &\equiv \frac{\partial \boldsymbol{p}}{\partial s} = \sum_{i=0}^{3} b_i'(s)\boldsymbol{q}_i(t). \end{aligned} \tag{2.2}$$

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 [57].

2.2.2 Applications for Various Phenomena

Many simulations of natural phenomena rely on simulating thin strand-like structures. There have been several papers in the graphics literature investigating the use of thin physical structures for simulation of various phenomena.

Qin and Terzopoulos [43] were the first to propose using B-splines for physically-based geometric design and their idea has since been extended by Remion et al. [45] and Pai [41]. Lenoir et al. [32] and Coleman and Singh [14] have developed methods for spline contact and constraints. Recently, Kass and Anderson [26] have used splines to animate oscillatory motion in an animation production environment.

Aside from general physical spline models, several recent researchers have been developing specialized physical models in order to simulate specific natural phenomena. Bertails et al. [8] have used strand-like objects called Super-helices for predicting the dynamics of natural hair. Spillmann and Teschner [56] introduced "CoRdEs" as the first detailed strand model of thread and have extended their work by including a contact model for the simulation of knots [55]. Their idea has recently been expanded upon by Bergou et al. [7], who developed their own thread model called Discrete Elastic Rods, and by Kaldor et al. [25] who have simulated knitted cloth at the yarn level.

2.2.3 Applications for Muscles and Tendons

This section is adapted from [57] and portions of this description were written by Sueda and Pai. It is reproduced here since it is essential for the

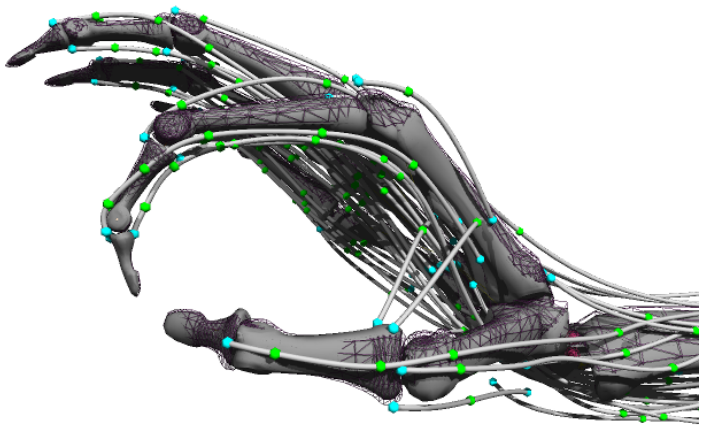


Figure 2.1: A screenshot of Sueda et al.'s simulator. Fixed constraints are shown in cyan, sliding constraints in green, and surface constraints in maroon. Surface constraints allow the strands to move axially as well as laterally. The input animation target is shown in wireframe. Figure reproduced from [57].

remainder of this thesis. Recently, Sueda et al. [57] have introduced strand-based muscle and tendon simulation (Fig. 2.1). Their simulator is built on two primitives: rigid bodies for bones and spline-based strands for tendons and muscles. Their 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 their muscle simulation, they are able to directly model this behavior. Strands allow them 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 [57].

They extend the physically-based spline models previously used in computer graphics (Sec. 2.2.2) to include muscle activation, simple yet robust sliding and surface constraint models, and implicit integration with Rayleigh damping [57].

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:

$$\begin{aligned} \chi &= \begin{bmatrix} \cdots & \mathsf{E}_i & \cdots & \boldsymbol{q}_j & \cdots \end{bmatrix}^T \\ \Phi &= \begin{bmatrix} \cdots & \phi_i & \cdots & \dot{\boldsymbol{q}}_j & \cdots \end{bmatrix}^T. \end{aligned} \tag{2.3}$$

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 [57].

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

$$M\Phi^{(k+1)} = M\Phi^{(k)} + hf - G^T\lambda, \tag{2.4}$$

where M is the block-diagonal generalized mass matrix of rigid bodies and strand control points [45], h is the step size, f is the generalized force (body forces for rigid bodies, elastic and damping forces for strands, etc.), and $G^T\lambda$ is the constraint force [57].

Rigid Bodies and Joints

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

$$\begin{aligned} \boldsymbol{x} &= \mathsf{E}\,\boldsymbol{r} \\ \dot{\boldsymbol{x}} &= \mathsf{E}^{-T}\,\Gamma(\boldsymbol{r})\,\phi, \end{aligned} \tag{2.5}$$

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. Joint constraints are implemented using the adjoint formulation [38], with which they easily derive different types of joints by simply dropping rows in the 6×6 adjoint matrix. For example, they drop the top three rows (the three rotational DoFs) for a ball joint, or the third row (the rotation about the z-axis) for a hinge joint [57].

Muscle Strand Dynamics

Based on the strand dynamic quantities mentioned in Sec. 2.2.1, they compute the passive and active elastic forces in the strand, which contribute to f in Eq. (2.4). Their simulator has the ability to use an arbitrary Force-Length (FL) relationship, which can be obtained from a standard Hill-type model [79] or from physiological experiments. However, the constitutive properties of muscles are still not well established and are the subject of intense ongoing research. For graphics applications, they 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 [57].

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

$$M\dot{\boldsymbol{q}}^{(k+1)} = M\dot{\boldsymbol{q}}^{(k)} + h(\boldsymbol{f}_d + \boldsymbol{f}_g + \boldsymbol{f}_p + \boldsymbol{f}_a) - G^T\lambda, \tag{2.6}$$

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, which converts the activations of the muscles into the corresponding forces on the strand DoFs. This is accomplished by scaling the activations as a function of the local strain and spline blending functions. The last term, $G^T\lambda$, is the constraint force term [57].

They use Rayleigh damping given by

$$\boldsymbol{f}_d = \left(\alpha M + \beta\frac{\partial \boldsymbol{f}^T}{\partial \boldsymbol{q}}\right)\dot{\boldsymbol{q}}, \tag{2.7}$$

where $\boldsymbol{f}$ is the cumulative force (excluding $\boldsymbol{f}_d$) from Eq. (2.6) and α and β are positive damping parameters [57].

Constraints

Constraints are required for musculotendon origins/insertions and for tendon routing. Although wrapping surfaces implemented in biomechanical simulators [16, 19] are effective for kinematic constraints, they do not work for dynamic constraints, and are also limited to simplified geometries, such as spheres and cylinders [57].

Tendon routing is particularly difficult, and ignored by other existing biomechanical simulators. They 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 [57].

Although their simulator is a general multi-body simulator with deformable strands, there is one assumption in their application that simplifies the constraint formulation. Since tendons and muscles stay in contact with surrounding tissue and do not come apart, they 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 and tracking the contact on this surface [57].

The constraints in their 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, they obtain a corresponding velocity-level constraint function that is consistent with their discretization.

$$\frac{d\,g(\chi)}{dt} = \frac{\partial\,g(\chi)}{\partial\chi}\Phi = 0. \tag{2.8}$$

Denoting the gradient of g by the constraint matrix G, they obtain the constraint equation $G\Phi = 0$ [57].

This constraint equation may allow the system to drift away from the constraint manifold because it is formulated at the velocity, not position level. They add a stabilization term [6] to help correct this drift by pushing the system back toward the constraint manifold. The stabilized velocity constraint equation is then

$$G\Phi = -\mu g, \tag{2.9}$$

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, $-\mu g$, is added to push the system back to the constraint manifold. For critical damping, they set $\mu = 1/h$, to minimize unnecessary oscillations [57].

Fixed Constraints: They use fixed constraints for strand origins and insertions, as well as for attaching several strands to form branching structures. For example, if they want to constrain a point on a strand, $\boldsymbol{p}$, to a point on

a rigid body, $\boldsymbol{p}_0$, they set the relative velocities to be equal.

$$\dot{\boldsymbol{g}} = \dot{\boldsymbol{p}} - \dot{\boldsymbol{p}}_0, \tag{2.10}$$

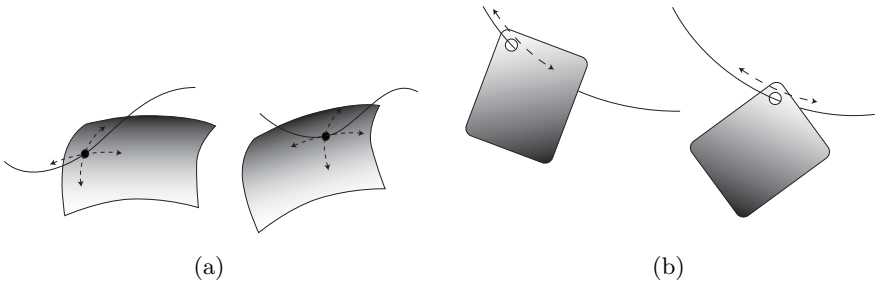
where both $\dot{\boldsymbol{p}}$ and $\dot{\boldsymbol{p}}_0$ are linear with respect to the DoFs of the system, as given by Eqs. (2.5) and (2.2) [57].

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

$$\dot{g} = \boldsymbol{n}^T(\dot{\boldsymbol{p}} - \dot{\boldsymbol{p}}_0). \tag{2.11}$$

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. 2.2(a)) [57].

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



(a) (b)

Figure 2.2: (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. Figure reproduced from [57].

to both of these vectors.

$$\begin{aligned} \dot{g}_1 &= \boldsymbol{n}_1^T(\dot{\boldsymbol{p}} - \dot{\boldsymbol{p}}_0) \\ \dot{g}_2 &= \boldsymbol{n}_2^T(\dot{\boldsymbol{p}} - \dot{\boldsymbol{p}}_0). \end{aligned} \tag{2.12}$$

Unlike the surface constraint, the constraint point on the rigid body is fixed, whereas it is allowed to slide on the strand (Fig. 2.2(b)) [57].

2.3 Commercial Modeling Tools

2.3.1 Animation

There is a wide range of modeling toolkits which have become prominent in the animation and visual effects industry. Several of these tools are parts of production level environments, offering not only modeling tools, but other advanced features such as animation, texturing, rendering, physical simulation for rigid and deformable bodies, and even complex tool sets for specific simulations, including cloth, fluids, and hair/fur simulations.

Autodesk's Maya [4] is a leader among the elite production environments of this nature. It is prominent in the film and television industries and has become a favored choice among many studios for 3D modeling and animation. It contains toolkits for all prominent aspects of visual effects development. Maya is extensible through both the Maya Embedded Language (MEL) and Python scripting. Additionally, Autodesk has released an OpenMaya API, so that Maya can be fully extended using C.

One of Maya's top competitors, 3ds Max [3], also known as 3D Studio Max, is also developed by Autodesk. Like Maya, 3ds Max is a huge 3D development environment containing all of the toolkits relevant to visual effects production. 3ds Max has been used most prominently for modeling in the video game and film industries. 3ds Max can also be extended using Python scripting.

Another prominent competitor is Avid's Softimage XSI [5]. It has a similar tool set to both Maya and 3ds Max. Softimage XSI has cemented its place in the video game industry and has recently integrated mental ray rendering into its pipeline.

Blender [10] is the only open-source 3D production environment that has received major acclaim. It contains all of the major classes of toolkits and is the only free animation suite available under the GNU General Public License. It is extensible using Python scripting, though as a publicly developed software suite, its interface and documentation are somewhat inferior to its major competitors.

SolidWorks [54] is another prominent modeling suite, though it is mainly used as a virtual testing product for engineering design work. Common ap-

plications include modeling, assembly, metal work, and free form surfacing. SolidWorks is extensible using Visual Basic and C.

In addition to these large suites designed for complete functionality of the visual effects pipeline, there are several smaller competitors that choose to focus on a limited, yet arguably more powerful, set of functionality within the visual effects industry.

Side Effects Software has developed Houdini [50], which has become a leader among visual effects and character animation suites. Houdini is a node based system with built in support for many prominent commercial renderers. Massive [34] is well known for its crowd related effects work, developed specifically for the "Lord of the Rings" trilogy. Smith Micro's Poser 3D [53] is a rendering and character animation suite based on character adjustment using predefined parameters.

There is a surprising lack of commercial musculoskeletal modeling tools in the animation industry. As with many types of simulations that are just beginning to make their way into the animation industry, many companies have developed their own in house systems, the details of which are kept secret from competitors and the research community. Some of these systems are quite intricate production quality muscle systems [46, 65], though to our knowledge, none of them use a strand based muscle approach.

The only prominent, commercially available musculoskeletal animation and simulation suite is by cgCharacter [11], which specializes in 3D character setup for film and television. They have designed male and female human characters with full muscle setups which are functional using their Absolute Character Tools (ACT) software. While the muscles in their system are

considered functional, they rely on line of action muscle models, which are no longer considered accurate enough (Sec. 2.1.1).

2.3.2 Biomechanics

The needs of the biomechanics community differs drastically from the needs of the animation and visual effects industry. In biomechanics, the primary focus is on physiologically accurate representations of musculoskeletal systems. It is much more important to have justifiable medical results than it is to have accurate looking results.

One of the leaders in musculoskeletal simulation is Musculographics Inc.'s Software for Interactive Muskuloskeletal Modeling (SIMM) [39]. This is a toolkit designed for modeling, animation, and biomechanical analysis of 3D musculoskeletal systems. Like most of its competitors, it is based on line of action muscle models and joint moments (Sec. 2.1.2). SIMM has found widespread use by biomechanists and biologists researching the functionality of a variety of musculoskeletal systems.

SimTk's OpenSim [52] software is an open source competitor to SIMM. OpenSim is designed for neuromusculoskeletal modeling and simulation for biomechanical analysis. It is geared toward multi-institutional collaboration with a variety of research communities. The software uses a combination of C++ and Java and can be extended to include custom contact models and controllers.

The MusculoSkeletal Modeling Software (MSMS) [61] released by The Alfred Mann Institute is another freely distributed package, helping researchers to build and analyze musculoskeletal models. It is implemented

in Java, Simulink and C. MSMS has a distributed block architecture, allowing each stage of development, from modeling, to simulation, to interface control, to be removed and replaced with custom plug-ins.

Portions of MSMS are based on The Alfred Mann Institute's Musculoskeletal Modeling in Simulink (MMS) [60]. The MMS package converts SIMM models into Simulink blocks. This functionality makes complex musculoskeletal modeling more accessible to novice programmers, since they no longer need C proficiency to build their models. Both MSMS and MMS allow for interchangeable Simulink muscle models, such as the Virtual Muscle [62]. This is a custom muscle package which uses line of action muscles with variable fiber types and force-length relationships.

The Visible Human Project® [68] is an ongoing research project conducted by the United States National Library of Medicine. Its goal is the creation of anatomically accurate 3D representations of both male and female human bodies. The Visible Human Project® is a joint venture among many independent research centers and is based on musculoskeletal modeling using medical imaging techniques.

Chapter 3

Design Decisions

There are several important design decisions that were made during the development of these interactive tools. These decisions were made in order to improve the development of musculoskeletal models, to easily interface with our muscle simulation software, and to provide animators with direct access to a novel biomechanical simulator.

We first discuss the overall flow of our modified animation pipeline (Sec 3.1), followed by decisions relating to the general platform that the tools are implemented in (Sec 3.2). Next we discuss the various object primitives that were specifically created for musculoskeletal modeling (Sec 3.3). We then discuss the semi-automated processes within the tool set (Sec 3.4). Finally, we discuss the tools for interfacing the modeling environment with the simulator (Sec 3.5).

3.1 Pipeline

This section is adapted from [57] and portions of this description were written by Pai. It is reproduced here since it is essential for the remainder of this thesis. We have integrated a novel biomechanical simulator into a traditional animation pipeline (Fig. 3.1). This allows artists and scientists to easily model musculoskeletal systems. Moreover, it enables artists to produce secondary motion of tendons and muscles under the skin of a traditionally animated character. To accomplish this task, our implementation makes some important pipeline design choices.

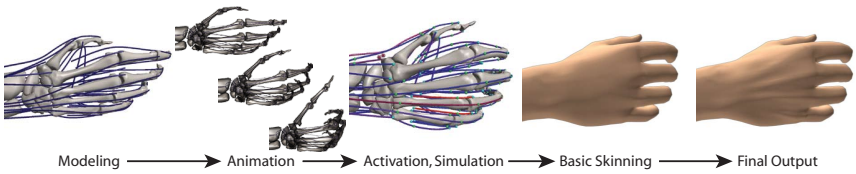


Figure 3.1: Pipeline: The user specifies the model and its corresponding animation. Our system computes the required activations, and simulates the muscles, tendons, and bones. The skin is then attached to the skeleton, and the subcutaneous deformation from tendon motion is added as a post-process. Figure reproduced from [57].

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 Character Technical Director (TD), 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 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. First, an animation rig is constructed in the usual way, but in addition, strands are placed within the character. The strands can be built manually using the standard spline tools available in Maya. In addition, the strand construction process can be semi-automated using the AutoStrands Tool provided by our plug-in (Sec. 3.4.1). 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 (for a complete list of parameters, see Table 3.1). 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 Character TD 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 (Sec. 3.4.3). 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 [47].

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 (Sec. 3.5). The simulator calculates the dynamic activation levels for the muscles that best match the animation keyframes provided (Sec. 2.2.3). 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 (Sec. 3.4.2). 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 (Sec. 3.4.3).

3.2 Platform

3.2.1 Embedded Plug-in

The interactive tools that we have developed are implemented as a plug-in for Autodesk's Maya [4]. The design principles used in the development of these tools should easily lend themselves to implementation and further development using any professional 3D modeling program.

The decision to build an embedded plug-in within an existing modeling and animation environment was made in order to leverage the production quality animation tools as well as the professional interactive interface already familiar to many artists. We chose to use Maya as our modeling and animation environment because it is a well known and commonly used program within the animation industry. In addition to Maya's reputation as a top level 3D environment, Autodesk, Inc. has a substantial history supporting animation research within the educational community and has generously donated Maya licenses for our continued research and development of musculoskeletal software and other graphics related projects.

3.2.2 Scripting

The options for plug-in implementation in Maya, and many other 3D modeling programs, include both scripted commands and compiled code using an API. In Maya, scripts can either be written in MEL or in Python. Compiled code can be written in C and C++ using the OpenMaya API.

Our plug-in is implemented as a combination of MEL and Python scripts. This decision was made primarily due to the significantly faster speed of

development that is made possible by scripting languages. Secondary and tertiary reasoning for the choice to use a scripted plug-in include prior familiarity with both MEL and Python and to leverage Python's ability to easily incorporate external code as modules. See Sec. 3.2.4 and Sec. 3.5 for specific examples of the advantages that Python's modularization capabilities can provide.

3.2.3 User Interface

There are several user interface possibilities within Maya. Interfaces can involve text menus, mouse context menus, pop-up windows and dialogs, and shelves. The first three choices are common interfacing techniques, while the final choice, a shelf, is a specific Maya implementation of button menus. Maya has organized all of its button menus into separate tabs called shelves. Any Maya setup contains several shelves, organized into categories such as "Polygons", "Surfaces", "Rendering", "Physics", "Cloth", etc.

We have chosen to design our interface by utilizing the shelving feature along with a few pop-up windows and dialogs. The majority of the functionality of our plug-in is accessible through our custom shelf, the "sslMuscles" shelf (Fig. 3.2). This shelf contains buttons for the creation of our muscle primitives (Sec. 3.3), for exporting data to our muscle simulator (Sec 3.5), importing data from the muscle simulator (Sec. 3.4.2), and for automating skin deformation (Sec. 3.4.3).

In addition to our custom shelf, we make use of several pop-up windows and dialogs. The simplest pop-up dialog that we use consists of some instructions along with a single field to be filled in by the user. We have two

Figure 3.2: Our custom shelf in Maya.

more complicated windows that are integral to the automation processes in our plug-in.

The Skin Strands Window (Fig. 3.3) is the simpler of the two windows. It allows users to automatically deform a mesh based on the proximity to a set of strands (Sec. 3.4.3). There are several parameters that are tunable with fields, sliders, and check boxes in the window. Finally, there is a button to begin the deformation process using the current parameter values. See Appendix A.9 for complete instructions on how to use the Skin Strands Window.

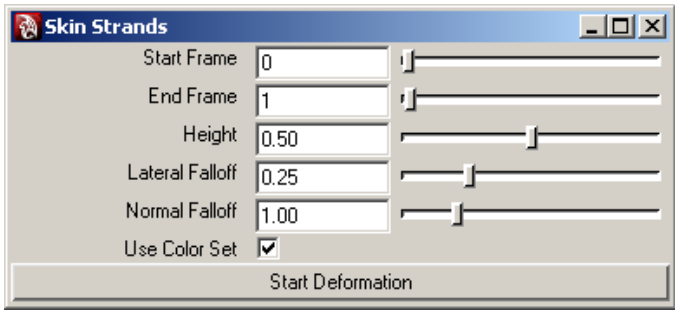


Figure 3.3: The window to control the deformation effects available for skinning strands.

The AutoStrands Window (Fig. 3.4) is considerably more complex than the Skin Strands Window. It allows users to semi-automatically fill muscle meshes with strands, based on external simulation code (Sec. 3.2.4). The key difference making the AutoStrands Window more complicated is the

presence of an external simulation running in a separate thread. The window therefore contains several buttons that control the AutoStrand simulation threads (Sec. 3.4.1). It also contains fields, sliders, and radio buttons that tune the simulation parameters. However, several of these parameters can continue to be tuned after the simulation has been started, interactively affecting the outcome. See Appendix A.4 for complete instructions on how to use the AutoStrands Window.

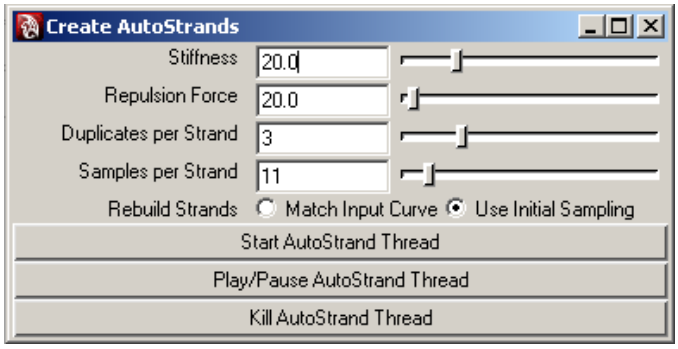


Figure 3.4: The window to control AutoStrand simulations.

3.2.4 Modularization and External Code

The decision to use Python as a primary development language for our plug-in allows us to easily incorporate external code as modules. The primary advantage that this modularization provides is drastically increased functionality requiring minimal development time. We have made use of python modularization in several significant areas throughout our plug-in.

The built in Python module for XML querying and development allows us to transport data between Maya and our simulation software in an efficient and reliable manner (Sec. 3.5). The Python XML module is integrated

into any standard Python implementation, and its functionality is easily accessible to developers.

We have also made use of the Python Threading module in a unique way. Our AutoStrand Tool relies on an external simulation developed in C/C++, but not using the OpenMaya API (Sec. 3.4.1). We compile the simulation code as a Python extension and run it in a separate thread using the Threading module. This process (Fig. 3.5) has two key benefits. First, it allows us to easily incorporate external code created by developers with no knowledge of the OpenMaya API. Second, by running a simulation in a separate thread we can allow the user to interactively control the simulation from within Maya. The current state of the simulation is automatically displayed within the Maya viewport and the user can tweak the parameters as they see fit. This functionality is usually not available during a standard Maya simulation, which temporarily freezes most user controls.

Figure 3.5: Block diagram of the AutoStrand modularization process.

An added benefit of this threading process is that users can continue modeling in Maya while the simulation runs. Since our AutoStrand simulation is a background process designed to produce a final static output, the actual simulation animation is irrelevant. The user can safely continue their work while keeping an eye on the progress of the AutoStrand simulation.

3.3 Primitives

We have developed a variety of new primitives for the construction of musculoskeletal models. The five types of primitives are rigid bodies, joints, strands, muscle groups, and constraints. Each of these primitives represents a key building block required by our simulation software.

All of our musculoskeletal primitives are based on existing Maya primitives. They are implemented as pseudo-primitives, adding parameters (Table 3.1) to each of their respective Maya primitives, so that the maximal set of existing tools can be leveraged for the creation of musculoskeletal models.

This pseudo-primitive architecture also allows us to easily convert between Maya primitives and our musculoskeletal primitives. A musculoskeletal primitive can be deleted, leaving behind the valuable control point data contained within the original primitive. Any musculoskeletal primitive can be reverted back to its original Maya primitive using our Remove Attributes Tool (Appendix A.5).

Rigid Bodies

Rigid bodies represent bones. They can be built on top of any frame based structure in Maya, though the conventions set by our simulation software dictate that they be created from Group nodes. This allows multiple meshes and surfaces to be bundled into one rigid body. When a user creates a rigid body, a center of mass locator will automatically be generated and placed in a best guess location. This center of mass is critical for determining the bone's inertia and other physical properties at simulation time.

Joints

Joints represent anatomic joints connecting bones. We allow for several types of joints, including ball, hinge, cylindrical, universal, prismatic, and rigid joints. As with rigid bodies, joints can be built on any frame based structure in Maya. It is reasonably irrelevant which structure is chosen, though locators or Maya joints make the most sense, as they make good visual representations of single points in space. In order to properly communicate with our simulation software, all joints in a scene should be placed in a single, world level group named "Joints."

Strands

A strand is our basic muscle tissue primitive. Strands must be built on top of NURBS curves, since strand simulations are based on a spline representation (Sec. 2.2.1). There are several methods for strand creation, including manual creation using Maya's built in NURBS curve tools, automated creation by scripting the strand paths, or semi-automated creation using our AutoStrands Tool (Sec. 3.4.1). One interesting aspect of our simulation software is that each strand is allowed a unique passive and active force-length (FL) curve. We use pop-up dialogs at strand creation time to allow the user to specify custom FL curves for each strand, or to stick with the default curves used in the simulation software.

Table 3.1: Attributes available on our musculoskeletal primitives.

Name	Type	Default Value	Summary
Rigid Body	sslRigidBody	Compound	A container for all other rigid body attributes.
Density	Float	1.0	The density of the rigid body.
Pinned	Boolean	False	A flag pinning the rigid body in place during simulation.
Radius	Float	1.0	The average radius of the entire rigid body (assuming a cylindrical bounding shape).
Length	Float	1.0	The length of the entire rigid body (assuming a cylindrical bounding shape).
Enabled	Boolean	True	A flag to enable the rigid body in the simulation.
Ssl Constraint	Message	Locator	A pointer to any constraints attached to the rigid body.
Joint	sslJoint	Compound	A container for all other joint attributes.
Type	Enumeration	Ball	The type of joint this object represents. Options are Ball, Hinge, Rigid, Cylindrical, Universal, or Prismatic.
Rigid Body A	Message	1st object	The 1st of 2 sslRigidBodies that this joint connects.
Rigid Body B	Message	2nd object	The 2nd of 2 sslRigidBodies that this joint connects.
Strand	sslStrand	Compound	A container for all other strand attributes.
Density	Float	1.0	The density of the strand.
Damping	Float	0.0	The damping parameter used during simulation.
Passive Scale	Float	1.0	The passive scale used during simulation.
Passive FL	String	“”	The passive FL curve associated with this strand.
Active Scale	Float	1.0	The active scale used during simulation.
Active FL	String	“”	The active FL curve associated with this strand.
Radius	Float	1.0	The radius of the entire strand (assuming uniform thickness across the strand).
Enabled	Boolean	True	A flag to enable the strand in the simulation.
Ssl Constraint	Message	Locator	A pointer to any constraints attached to the strand.

Table 3.1: – continued from previous page.

Name	Type	Default Value	Summary
Muscle Group	sslStrandGroup	Compound	A container for all other muscle group attributes.
Type	Enumeration	Volumetric	The type of muscle this group represents. Options are Volumetric or Sheet muscle.
Divisions	Int	5	The number of divisions that the simulation volume preservation function should use.
Start Point	Float	0.1	The parameter point which begins the first division.
End Point	Float	0.9	The parameter point which ends the last division.
Kp	Float	1e2	The spring parameter for volume preservation.
Kd	Float	1e-3	The damping parameter for volume preservation.
Constraint	sslConstraint	Compound	A container for all other constraint attributes.
Position	Float	Input	The parameter point on the strand that this constraint is attached to. The actual attribute name varies with constraint type.
Cnst Attachment	Message	Object	A pointer to the object constraining a strand.
Equality	Enumeration	Equal	Only available on surface constraints. Determines whether this is an equality or inequality constraint.
Side	Enumeration	Above	Only available on surface constraints. Constrains a strand to lie above or below the surface.

Muscle Groups

A muscle group is a muscle structure that consists of a collection of strands. Muscle groups are built on top of Group nodes. Any strands contained in the Group node will be considered to be a part of the muscle group. There are two primary functions of muscle groups. The first is to specify which strands belong to one muscle, and should thus co-activate during a simulation. The second function is to designate volumetric and sheet muscles. By making this key distinction, we have the freedom to process volumetric muscles differently in our simulation software.

Constraints

Constraints are the least intuitive of our musculoskeletal primitives. They are used for routing strands and attaching them to other primitives (Sec. 2.2.3). A constraint is the only musculoskeletal primitive that is created before its Maya primitive exists in the scene. Constraints are built on top of locators, which are automatically generated and attached to strands upon creation. The constraint locators can slide around on the strand using the position attribute, but will always be attached directly to the strand. There are several types of constraints, each with its own specific purpose:

Fixed Constraints: Fixed constraints are used to define strand origin and insertion points, to allow a single strand to split off in multiple directions, or multiple strands to merge into one strand.

Surface Constraints: Surface constraints are used to constrain a strand on, above, or below a given surface.

Sliding Constraints: Sliding constraints are used to constrain a strand to slide through a specific point in space, relative to a rigid body.

Strut Force Constraints: Strut force constraints are used to mark points between neighboring strands, in sheet muscles, that should be concerned with volume preservation.

3.4 Automation

We have developed several tools for automating the musculoskeletal modeling and animation processes. Some of these tools are fully automatic, while others are semi-automatic. The key difference is that semi-automatic tools allow for user influence. This decision was made in order to automate as much of the process as possible, while leaving a pipe open for a user to manually manipulate the outcome.

3.4.1 AutoStrands

The most significant automated modeling tool that we have developed is the AutoStrands Tool, which is designed for semi-automatic strand placement. This tool will generate several new strands based on each input strand and attempt to evenly disperse them within a given polygon mesh. Each new autostrand will copy the origin and insertion points from its input strand and will be placed at the same level in the scene hierarchy.

The AutoStrands Tool is able to disperse the autostrands using external simulation software by Levin et al. [33], developed with no relation to Maya (Sec. 3.2.4). This simulation is run in a separate thread, which must be run and killed from the AutoStrands Window (Fig. 3.4). When the user starts an AutoStrand simulation, the thread will be launched and the simulation will be initialized using the current parameters along with the selected mesh and strands.

The simulation will update the Maya viewport at every time step and will continue running until it is either paused or killed. Several of the parameters can be interactively manipulated while the simulation is running (Appendix A.4). Once a final desired state has been reached, the user must kill the AutoStrand thread. At this point the autostrands will behave like normal strands and the control points can be manually manipulated to finalize the strand paths.

3.4.2 Baking Simulations

Since we are generating strand motion in our external musculoskeletal simulator, we have provided a method for using that automated strand animation within Maya. Baking a simulation refers to the process of saving simulation results by adding keyframes, based on the simulation, to the objects in the scene [47]. Once a simulation has been baked, it exists as a keyframed animation and there is no need to recompute the simulation. The user is then free to manually alter the baked keyframes.

We are concerned with baking keyframes to the strand control points in our scenes. To accomplish this task, we provide a Bake Simulation Tool

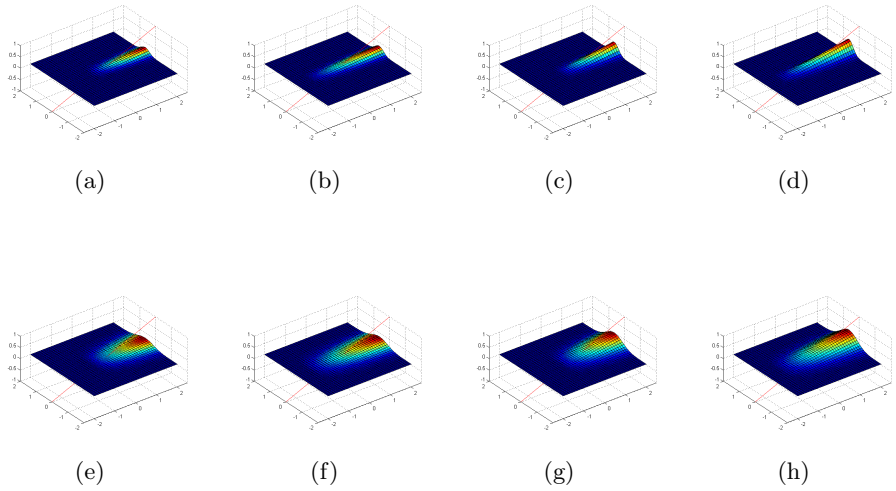
that uses a pop-op dialog to locate a saved Replay XML file created by the simulation software (Sec. 3.5). Our tool then queries the XML file using the Python XML module (Sec. 3.2.4) and adds the necessary keyframes to the strand control points in order to reproduce the simulation. See Appendix A.9 for specific details on using the Bake Simulation Tool.

3.4.3 Secondary Deformation

This section is adapted from [57] and portions of this description were written by Pai. It is reproduced here since it is essential for the remainder of this thesis. We have provided a method for automatically generating secondary deformation of a character's skin. The tendons are automatically skinned to the characters surface mesh, providing complex secondary motion of the skin as a result of biomechanically realistic tendon motion.

We automatically determine the skin mesh vertices that are affected by each strand using proximity computations. 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 (Fig. 3.6).

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 (Fig. 3.7). Only mesh vertices with positive influence weights are deformed.



(a) (b) (c) (d)

(e) (f) (g) (h)

Figure 3.6: Varying effects of the skin deformation algorithm. Lateral falloff is 0.25 for (a) – (d) and 0.5 for (e) – (h). Normal falloff is 0.25 for (a), (c), (e), and (g) and is 0.5 for the rest. Height is 0.25 for (a), (b), (e), and (f) and is 0.5 for the rest.

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:

$$\begin{aligned} \boldsymbol{d} &= \boldsymbol{p}_s - \boldsymbol{p}_v \\ h &= \max(\boldsymbol{d} \cdot \boldsymbol{n} + c, 0) \\ f &= a \exp\left(\frac{-\|\boldsymbol{d} - (\boldsymbol{d} \cdot \boldsymbol{n})\boldsymbol{n}\|^2}{2b^2}\right) \\ \boldsymbol{p}_v &= \boldsymbol{p}_v + (w\, h\, f)\boldsymbol{n}, \end{aligned} \tag{3.1}$$

where $\boldsymbol{n}$ is the outward vertex normal and w is the vertex influence weight. The parameter a controls the height, h, of the offset, b controls the falloff factor, f, in the lateral direction, and c controls the falloff factor in the

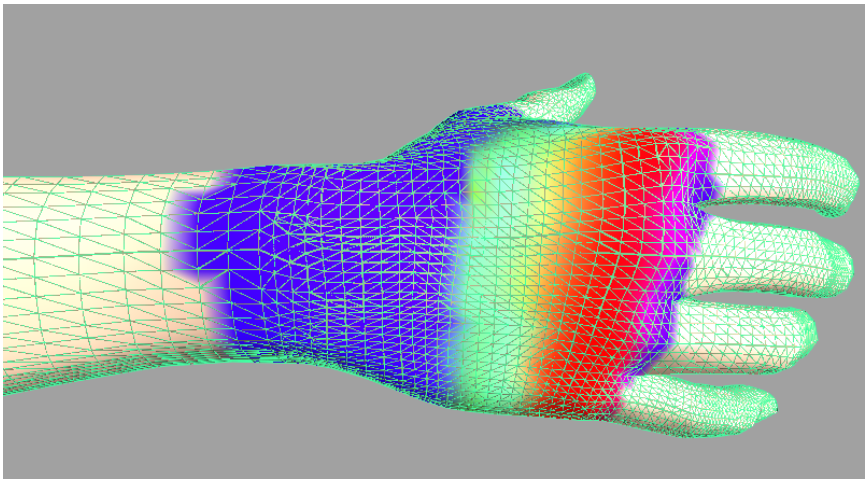


Figure 3.7: Vertex influence weights painted on the base mesh of a human hand. The weights go from 0 to 1 using only the R channel of the color to determine the weight.

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.

3.5 Interfacing with the Simulator

It is of utmost importance that we effectively interface with the musculoskeletal simulation software. Without these interfacing tools, we would not be able to use our musculoskeletal models in biomechanical simulations, nor would we be able to create realistic animation based on our simulations.

There are several key decisions that we made in the development of the interfacing tools. The first decision was to develop a new Java class that

is designed to create all of the elements in a 3D scene. We developed the ModelerScene class as a generic 3D scene class, with no elements specific to Maya or any other program. We have chosen to keep the ModelerScene class completely distinct from our musculoskeletal simulator as well. This allows the ModelerScene class to remain a generic 3D scene class that can be used by any Java graphics application.

We decided that the best way to interface with this class (Fig. 3.8) is by following the JavaBeans specifications [20]. These specifications describe a format for Java classes that allows many objects to be encapsulated into a single object, called a bean. JavaBeans can be serialized and can be easily encoded into, and decoded from, XML files. A fully functional instance of the ModelerScene class can be created given a ModelerScene XML file.

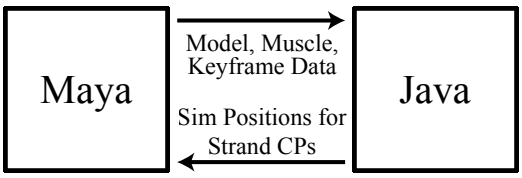


Figure 3.8: Block diagram of the simulation interfacing process.

The next step in the interfacing process is to create the ModelerScene XML beans. We have developed a ModelerScene XML exporter in Maya that will export the selected objects to a properly formatted ModelerScene XML bean (Appendix A.6).

Since the ModelerScene class has no knowledge of musculoskeletal objects, it is also necessary to export the musculoskeletal data. We have created a tool to export this muscle data in a separate Muscle XML file (Appendix

A.7) and have implemented a ModelerSceneLoader class within our musculoskeletal simulation software that uses both the ModelerScene XML and the Muscle XML files.

In order to track an animation using the strand controller [57], we need to first create that animation. Keyframed animations can be created in Maya and exported in a Control XML file (Appendix A.8). The Control XML file can easily be loaded in our simulator, since it follows the specifications of our simulator's Replay XML files. Our Replay XML files contain keyframe data for rigid body positions and orientations along with strand control points. They can be used to replay simulations within the simulator or to bake simulations onto strands in Maya (Sec. 3.4.2).

Chapter 4

Results

We have conducted two case studies in order to demonstrated the effectiveness of our musculoskeletal modeling and animation tools. The first case study focuses on development and animation of a realistic model of the human hand (Sec. 4.1). This model constitutes an important part of the results of our recent SIGGRAPH publication [57]. The second case study focuses on development and biomechanical analysis of a novel model of the human shoulder (Sec. 4.2). This shoulder model was recently included in the Proceedings of the International Shoulder Group [27].

4.1 Case Study: The Human Hand

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. The core goal of this

case study is to look into the anatomy of the hand, and the current state of hand simulation and animation techniques, in order to generate more realistic human hand animation.

4.1.1 Anatomic Description

The human hand is comprised of carpal bones, metacarpal bones and phalanges, while the forearm includes the radius and ulna (Fig. 4.1). Carpal bones are a series of eight small bones, joined by ligaments, and articulated with the radius, ulna, and each other by intercarpal joints. The hand also has five metacarpal bones, which make up the base of the palm. Each metacarpal bone articulates with a carpal bone and the distal bone of its corresponding digit. The distal head of each metacarpal bone is a knuckle. Phalanges are the bones of the digits, or fingers. The thumb has two phalanges, while each finger has three phalanges. Proximal phalanges articulate with the metacarpal bones and the middle phalanges, which in turn articulate with the distal phalanges [66].

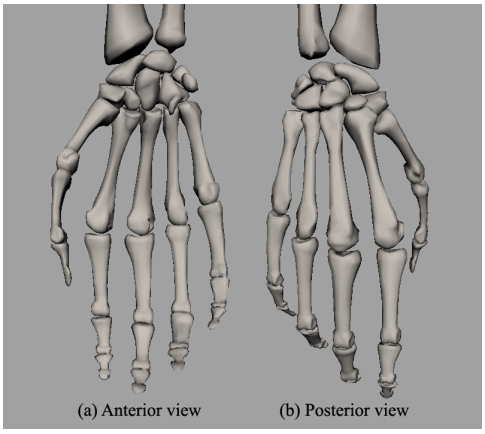


Figure 4.1: Anterior and posterior views of the bones of the hand.

There are two major compartments of muscles that move the wrist, hand, and digits. The anterior compartment muscles are flexor muscles that originate on the humerus and insert on the carpal bones, metacarpal bones, and phalanges. The posterior compartment muscles are extensors which originate on the humerus and insert on the metacarpal bones and phalanges. Each compartment contains both superficial and deep groups of muscles [66].

Anterior Compartment

There are four muscles in the superficial anterior compartment and two muscles in the deep anterior compartment (Fig. 4.2). The deep anterior compartment consists of flexor pollicis longus, which flexes the distal phalanx of the thumb, and flexor digitorum profundus, which flexes the distal, middle, and proximal phalanges and the hand at the wrist joint [66].

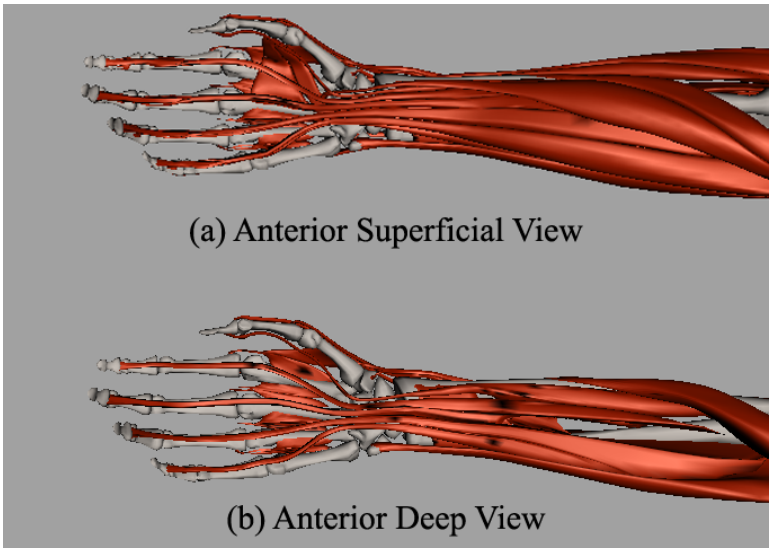


Figure 4.2: Superficial and deep anterior compartments for the muscles of the hand.

The largest superficial muscle is the flexor digitorum superficialis, which flexes the middle and proximal phalanges along with the hand at the wrist joint. It is assisted in felxion by palmaris longus, flexor carpi radialis, and flexor carpi ulnaris. Flexor carpi radialis also abducts the hand, while flexor carpi ulnaris adducts it [66].

Posterior Compartment

The posterior compartment consists of five superficial muscles and four deep muscles (Fig. 4.3). In the deep posterior compartment, abductor pollicis longus and extensor pollicis longus are responsible for abducting and extending the thumb and wrist. Extensor pollicis brevis assists in extending the thumb. The final muscle in the deep compartment, extensor indicis, extends all phalanges of the the index finger and wrist [66].

The primary muscle of the superficial posterior compartment is extensor digitorum. This muscle comprises most of the posterior surface of the forearm and splits into four tendons inserting onto each finger. It extends all of the phalanges of each finger, along with the wrist. Extensor carpi radialis and extensor carpi radialis brevis extend and abduct the hand. Extensor carpi ulnaris extends and adducts the wrist. The final muscle of the superficial compartment, extensor digiti minimi, helps extensor digitorum to extend the little finger [66].

4.1.2 Previous Hand Models

There have been several avenues of research into the development of biomechanically accurate hand models. Hands are an important topic in the

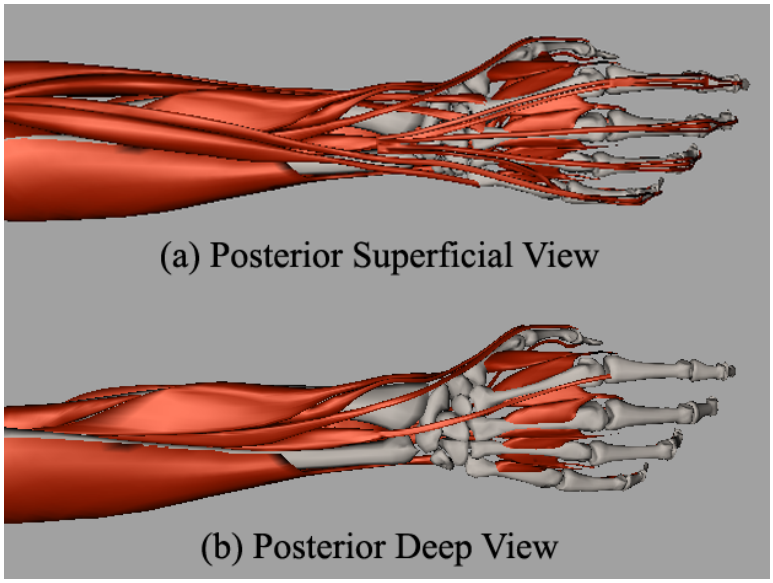


Figure 4.3: Superficial and deep posterior compartments for the muscles of the hand.

robotics community, since grasping and manipulation of objects is a key issue in the creation of humanoid robots. Along this line, Wilkinson et al. [76] have developed an extensor mechanism for an anatomical robotic hand.

As with any important musculoskeletal system, biologists, biomechanists, and biomedical engineers are interested in studying hand models. A recent example of research in this area is that of Valero-Cuevas et al. [69], who looked into the ability of the tendon network of the fingers to perform anatomical computations at a macroscopic scale.

In recent years, there has been significant research into the generation of realistic hand animation for the visual effects industry. The likely reason for this recent emergence of hand animation research is that character animation is of central importance to the industry and hands are often a key focal point

on animated characters. Albrecht et al. [1] were the first, in this recent surge, to tackle construction and animation of anatomically based human hand models.

Later, Tsang et al. [67] developed an anatomically accurate inverse dynamics solution for unconstrained hand motion. Around the same time, Pollard and Zordan [42] created animations from physically-based grasping control, based on examples from hand related motion capture data. Kry and Pai [28] have also worked with motion captured hand data. They used motion capture, along with captured contact forces, to synthesize physically-based hand animations.

4.1.3 Creating a New Hand Model

We have developed a musculoskeletal model of the human hand using our musculoskeletal modeling tools and our simulation software. Our hand model includes all of the muscles that move the wrist, hand, and digits. It can be animated using keyframed bone animation paths (Sec. 3.5) and the novel muscle controller built into our simulation software [57].

The major differences between our hand model and the previous hand models are that we use no motion or force capture techniques. Our model does not incorporate any external motion databases and the motion of the muscles are generated based on the input animation. Additionally, our muscles are simulated using forward dynamics, allowing for realistic muscle deformation and reactions to prior motion and contact.

Our hand model (Fig. 4.4) contains 54 musculotendons and 17 bones. The bone and muscle meshes were purchased from Snoswell Design in Ade-

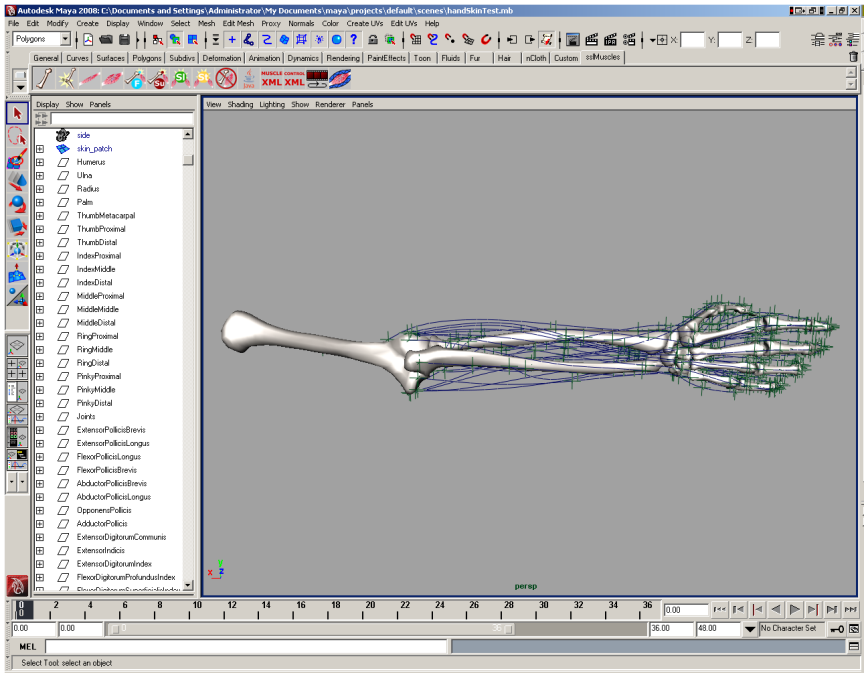


Figure 4.4: Posterior view of our hand model. Strands are shown in blue, constraints in green, and bones in gray. The top level scene hierarchy is displayed on the left.

laide, and the musculotendon paths were manually constructed based on standard textbook models in the literature [37]. We use fixed constraints to define origin and insertion points. In our hand model, surface constraints keep extensor pollicis brevis and extensor pollicis longus near the superficial layer of the thumb, while sliding constraints help route the extensor digitorum tendons over the knuckles of each digit. We did not need to use struts, since most of the hand muscles are thin fusiform muscles that can be fully simulated using a single strand. The model took an expert user approximately two days to build and debug, with the primary building time corresponding to manipulation of constraint locations.

4.1.4 Animation from Simulation

This section is adapted from [57] and portions of this description were written by Sueda. It is reproduced here since it is essential for the remainder of this thesis. In order to generate hand animations with our model, we had an artist rig and animate the hand skeleton, and the resulting animations were imported into our simulation software. The activations for an animation sequence of several seconds were computed within a few minutes. These sequences were baked onto the strand control points in Maya, and the final animations were rendered using Maya's internal software renderer.

(a) (b) (c)

Figure 4.5: Still shots from an animation showing pronation/supination of the forearm, and abduction/adduction of the wrist, computed at interactive rates using our controller. Figure reproduced from [57].

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. 4.5). Subcutaneous motions are most pronounced for the extensor and abductor tendons of the thumb (Fig. 4.6), and the extensor digitorum tendons on the back of the palm (Fig. 4.7). Our tech-

(a) (b) (c)

Figure 4.6: 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". Figure reproduced from [57].

(a) (b)

Figure 4.7: (a) Base skin. (b) With our method applied, showing tendons on the back of the hand realistically deforming the skin. Figure reproduced from [57].

nique correctly captures the deformation of the skin on the back of the hand during finger extension (Fig. 4.8). We can generate more subtle deformation by varying the skinning parameters (Fig. 4.6(c)). Finally, we validate our technique by comparing the simulated tendons of the thumb to several real thumb photographs (Fig. 4.9).

(a) (b) (c)

Figure 4.8: Animation showing tendons deforming the skin during extension of the digits. Figure reproduced from [57].

Figure 4.9: We compare the simulated tendons of the thumb to several real thumb photographs. Figure reproduced from [57].

4.2 Case Study: The Human Shoulder

The human shoulder is one of the more complicated musculoskeletal structures to simulate accurately. The core goal of this case study is to look into the anatomy of the shoulder, and the current state of biomechanical shoulder models, in order to gain a better understanding of how a shoulder model would ideally function. The chief purpose of this research is the development of a new biomechanical shoulder model which may have benefits in both the medical and animation industries.

4.2.1 Anatomic Description

There are three bones that comprise the shoulder apparatus: the clavicle, scapula, and humerus (Fig. 4.10). The medial end of the clavicle articulates with the sternum, superior to the first rib of the thorax, forming the sternoclavicular joint. The lateral end of the clavicle articulates with the acromion of the scapula to form the acromioclavicular joint. The proximal head of the humerus articulates with the glenoid cavity of the scapula forming the glenohumeral joint [66].

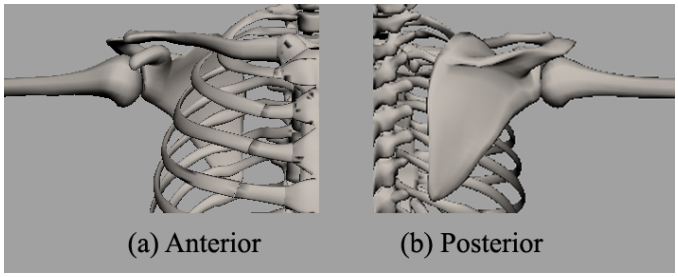


Figure 4.10: Anterior and posterior views of the shoulder bones.

The system of muscles that are involved with the shoulder can be described in two major groups, based on function. The muscles of the pectoral girdle stabilize the scapula in order to provide a steady point of origin for the muscles that move the humerus. These muscles, along with the tendons that form the rotator cuff, are responsible for the stabilization and movement of the humerus [66].

Pectoral Girdle

While the main function of the muscles that move the pectoral girdle is to stabilize the scapula for humeral movement, these muscles also increase

the range of motion of the humerus by moving the scapula. There are seven muscles that move the pectoral girdle, three of which originate on the anterior thorax (Fig. 4.11), and four of which originate on the posterior thorax (Fig. 4.12) [66].

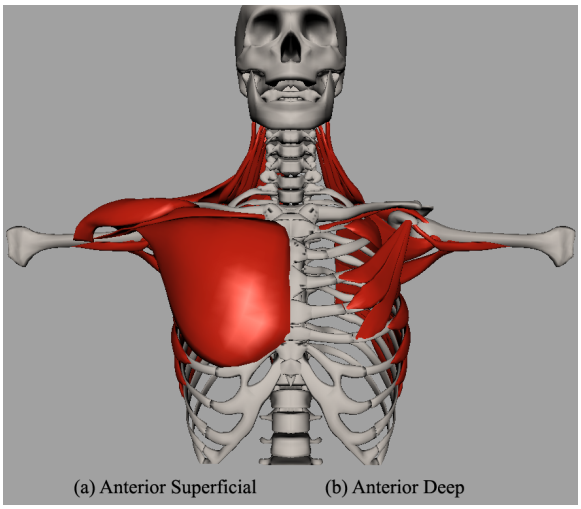


Figure 4.11: (a) Anterior superficial view of the shoulder and (b) anterior deep view of the pectoral girdle.

The subclavius is small and cylindrical, and connects the first rib to the clavicle. Pectoralis minor is a flat triangular muscle which inserts on the coracoid process of the scapula, and is integral in the movement of the scapula. Serratus anterior is a large fan-shaped muscle connecting the superior eight or nine ribs to the vertebral border and inferior angle of the scapula [66].

The trapezius is the largest muscle involved in the pectoral girdle. It is a large flat sheet of muscle spanning from the superior nuchal line of the occipital bone, down over the spines of all the thoracic vertebrae, and

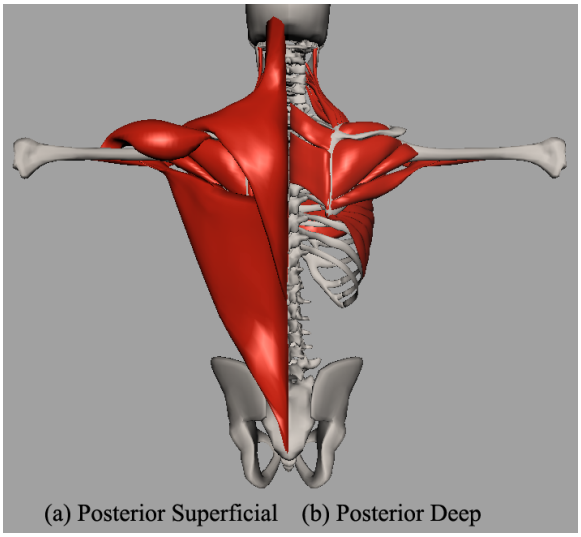


Figure 4.12: (a) Posterior superficial view and (b) posterior deep view of the shoulder and thorax.

laterally inserting on the clavicle, acromion and spine of the scapula. The fibers of trapezius perform several different functions, based on their location within the massive superficial muscle [66].

The levator scapulae is a long, narrow muscle originating at the cervical vertebrae, which functions to elevate the scapula. Rhomboideus major and rhomboideus minor are parallel bands which span inferiolaterally from the thoracic vertebrae to the vertebral boarder of the scapula. These rhomboid shaped muscles play a large roll in forcibly lowering raised upper limbs [66].

Rotator Cuff

There are nine muscles that are responsible for the movement of the humerus. Two of these muscles originate on the axial skeleton and the remaining seven muscles originate on the scapula. The humerus is held in place largely due

to the four muscles that make up the rotator cuff (Fig. 4.13). Their tendons fuse together to form a circle around the shoulder joint, providing much of the strength and stability of the shoulder [66].

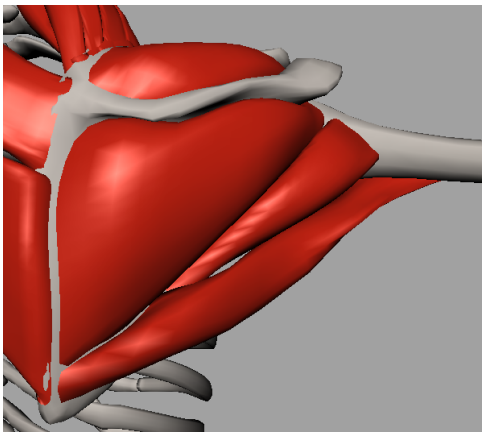


Figure 4.13: Posterior deep view of the rotator cuff.

These four muscles are subscapularis, supraspinatus, infraspinatus, and teres minor. Subscapularis is a large triangular muscle in the subscapular fossa. Supraspinatus is rounded and lays in the supraspinous fossa. Likewise, infraspinatus is located in the infraspinous fossa. Teres minor is a cylindrical muscle and is commonly connected to the infraspinatus. Supraspinatus is the primary muscle involved in shoulder dislocation [66].

The two axial muscles are pectoralis major and latisimus dorsi. Pectoralis major is a large fan shaped muscle covering the superior thorax. It has two origins, each with different functions. The clavicular head is responsible for flexion and the sternocoastal head is responsible for extension. Latissimus dorsi is a broad triangular muscle covering the inferior portion of the back.

The remaining scapular muscles that move the humerus are teres major, coracobrachialis, and the deltoid. Teres major is a thick flat muscle originating at the inferior angle of the scapula and inserting on the intertubercular sulcus of the humerus. Coracobrachialis is an elongated narrow muscle originating on the coracoid process of the scapula, traveling down the humerus and inserting on the medial shaft. The deltoid is the large, thick muscle that forms the rounder contour of the shoulder. As with the trapezius, the fibers of the deltoid perform several different functions. However, in the deltoid, these fibers can be identified by their differing points of origin [66].

4.2.2 Previous Shoulder Models

There has been a considerable amount of investigation into the inner workings of the shoulder. Researchers across several different fields of study, from biomechanical engineering, to orthopedic surgery, to computer science, have attempted to accurately model the shoulder. The scale of these models is highly variable. Some researchers choose to model the entire upper limb, while others may focus on just a single joint or muscle. While experimentally obtained shoulder models attempt to generalize shoulder function based on human subjects, some researchers create biomechanical shoulder models which exist entirely as computer simulations. These biomechanical models also vary in scope, though most go beyond the single joint or muscle. It is often the case that the simulations are limited intentionally in order to provide accurate approximations to the full system.

One of the most common simplifications among the biomechanical shoulder models is the inclusion of a scapulothoracic constraint. The purpose of

a scapulothoracic constraint is to restrict the motion of the scapula so that it glides along the thorax [35]. In the anatomic human, the muscles that insert on the scapula, along with the acromioclavicular and glenohumeral joints, determine the scapula's range of motion. Maurel et al. [36] move the scapula by defining a 5 degree-of-freedom gliding dot contact on an ellipsoidal model of the thorax. Shao and Ng-Thow-Hing [49] model the thorax as two ellipsoids, one centered on each half of the rib cage. They then define three pairs of reference points on the scapula, mandating that at least one reference pair be constrained to the ellipsoidal surface at all times. Charlton and Johnson [13] constrain the scapula using a pair of prismatic joints at the inferior and superior angles of the scapula.

Several models automatically determine the position and orientation of the clavicle and scapula by including an explicit model of the shoulder rhythm. When the body and arm are held stationary, the positions of the clavicle and scapula are under-constrained [21]. Some models choose to make use of the regression model by de Groot and Brand [15], while others implement their own versions of the shoulder rhythm. de Groot and Brand's model statistically predicts the orientation of the scapula and clavicle, based on the orientation of the humerus. Holzbaur et al. [22] use this regression model, along with experimentally measured moment arms, to define the path of each muscle or muscle part.

Muscles with large attachments are commonly simplified by treating different sections as separate muscles. There are several approaches to segmenting large muscles, stemming from both robotic and physiological justifications [13]. Johnson et al. [24] have defined the most physiologically

accurate fascicle divisions for all the muscles of the shoulder. The deltoid is often split into three main sections based on point of origin, which is in keeping with medical descriptions (Sec. 4.2.1). Pectoralis major, trapezius, and latissimus dorsi are also commonly split muscles.

4.2.3 Creating a New Shoulder Model

We have developed a novel musculoskeletal model of the human shoulder using our musculoskeletal modeling tools and our simulation software. Our shoulder model includes all of the muscles involved in scapulo-humeral operation, and functions with both forward and inverse dynamics. In forward dynamics, muscles may be activated individually or by anatomic function. Furthermore, we have eliminated the need for an explicit scapulothoracic constraint, allowing the muscles to fully control the motion of the scapula.

The most significant difference between our model and any of the existing shoulder models is the method of muscle simulation. Rather than using straight line muscle forces (Sec. 2.1.1), our model uses strands (Sec. 2.2.3). The complex routing constraints in the strand based muscle model allow us to eliminate the need for an explicit scapulothoracic constraint. Rather than creating an artificial constraint between bones, scapular placement is automatically determined by muscle configuration.

Our shoulder model (Fig. 4.14) simulates the motion of 16 shoulder muscles using 120 strands, 3 ligaments using 12 strands, and 3 ball joints. The motion of these muscle parts determines the position and orientation of the clavicle, scapula, and humerus. We use fixed constraints to define origin and insertion points. In our shoulder model, surface constraints keep

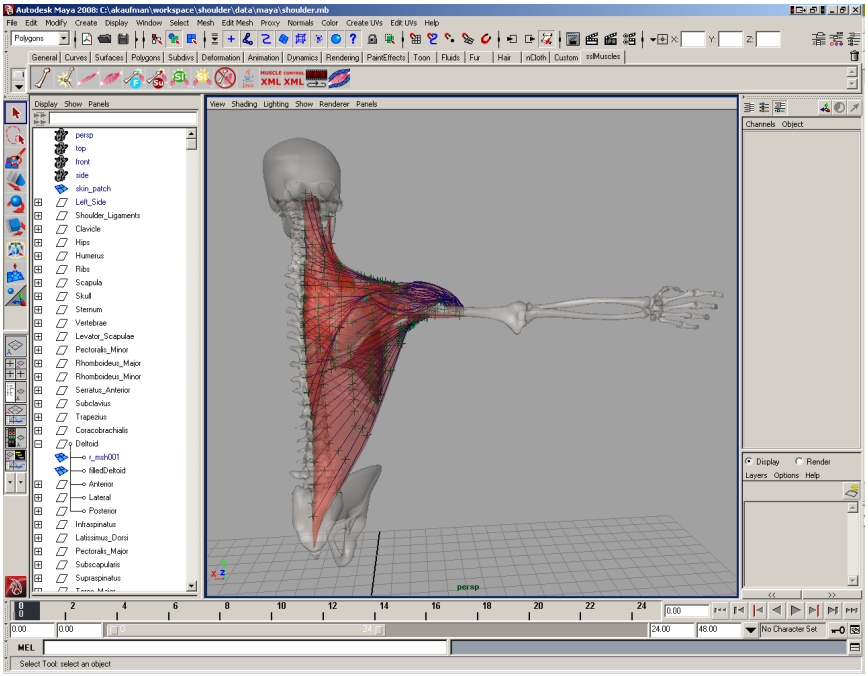


Figure 4.14: Posterior view of our shoulder model. Muscles are shown in red, strands in blue, constraints in green, and bones in gray. The top level scene hierarchy is displayed on the left.

the deep muscles above the thorax and below the scapula, and keep the superficial muscles above the scapula and outside the humerus. We mainly use struts on the superficial muscles, as we have found that most of the deep muscles in the shoulder are flat sheets that hold volume reasonably well on their own. The deltoid was created using our AutoStrands Tool (Sec. 3.4.1), while all other muscle strands were manually placed.

A user of our shoulder simulation has several options for controlling the muscle dynamics. Motion paths can be specified for any of the bones, in which case we perform an optimization to determine the dynamic activation levels for each of the muscle strands [57]. Forward dynamic simulations can

be performed either with pre-programmed activation levels, or interactively using a GUI. Muscles can be activated on an individual basis, or they can be co-activated by anatomic function.

4.2.4 Simulation for Biomechanical Analysis

We are able to successfully simulate several anatomic functions of the shoulder using forward dynamics. The anatomic functions of the scapula that our model supports include elevation, depression, abduction, adduction, upward rotation, and downward rotation. For the humerus, our model supports flexion, extension, abduction, adduction, medial rotation, and lateral rotation. The strand paths were built based on our existing muscle meshes and the detailed descriptions of Johnson et al. [23, 24]. The muscle parameters were tuned using experimental cadaver data [23, 24, 71] and a Java applet to determine damping forces.

Our simulator can be used for visualization as well as biomechanical analysis. Fig. 4.15 demonstrates visualization of muscle activations and Fig. 4.16 shows the forces exerted at the insertion points of the trapezius fascicles and their length change over time. This type of biomechanical analysis can be useful to understand the roles of individual sections of muscles during a given movement. The force plot tells us that the superior and middle fascicles of the trapezius exert the most force during the movement shown in Fig. 4.15, while the length plot tells us that during this simulation, all trapezius fascicles underwent a comparable length change. The total time to simulate the 330 milliseconds of motion in Fig. 4.15 was 66.2 seconds.

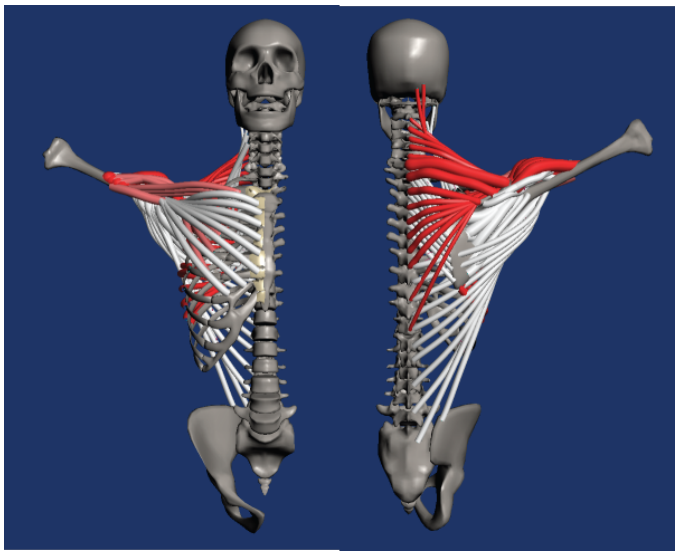


Figure 4.15: Anterior and posterior views of our shoulder model during 50% abduction and 80% upward rotation of the scapula, and 50% flexion and 80% abduction of the humerus. Activation levels are shown in red. Figure reproduced from [27].

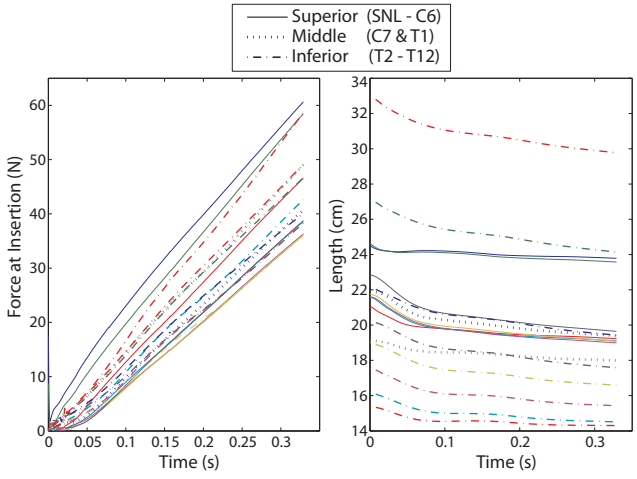


Figure 4.16: Forces and lengths in the trapezius during the simulation shown in Fig. 4.15. Figure reproduced from [27].

Chapter 5

Conclusions

We developed an interactive toolkit for the modeling and animation of musculoskeletal systems. Our tools make modeling these complex systems accessible to artists, biologists, biomechanists, and medical practitioners. We automate the entire animation process and portions of the modeling process. Most importantly, we enabled the seamless integration of biomechanically realistic secondary animation into a traditional animation pipeline.

5.1 Discussion

Portions of this section are adapted from [57] and were written by Sueda and Pai. It is reproduced here since it is essential for the remainder of this thesis. We demonstrated the effectiveness of our approach to realistic secondary animation by simulating the musculotendons of the human hand. Our hand model is able to accurately reproduce the visual effect of tendon motion on the back of the hand. Other than the hands, our method should work well with areas of flesh 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 the skinning process can be applied to areas of the body with large volumetric muscles, such as the shoulder or the torso.

Regardless of skinning, our modeling tools effectively enable scientists to create novel musculoskeletal models, such as our model of the human shoulder, which is able to simulate a wide range of motion of the shoulder using either forward or inverse dynamics. We have eliminated the need for a scapulothoracic constraint, giving the muscles total responsibility for scapular motion. In addition to transmitting muscle forces in a continuous manner, our simulation times on a single core are an order of magnitude faster than solid mechanics approaches running on multiple cores. The addition of multi-core processing into our simulation should make our computation times considerably faster.

These two case studies demonstrated that our musculoskeletal modeling and animation tools provide both artists and scientists with easy access to the development of complex musculoskeletal systems. We have shown that there is little reason, aside from the complexity of implementation, for either community to continue using simple line of action muscle models and that strand based approaches can become the standard for musculoskeletal simulation.

5.2 Future Work

Clearly, there are still challenges to be met, both on the graphics side and in terms of biomechanical construction and validation. We would like to

provide tools which are appropriate for animating large volumetric muscles. This will require further development of our volumetric muscle simulation techniques. The current strut forces work well for flat sheet muscles, such as the trapezius, but are not robust enough to support large volumetric muscles, such as the deltoid. We will need to incorporate the effects of these layered and bulging muscles. This will also require a new skinning algorithm that evenly distributes the effects of muscle bulging across a character's skin.

We would also like to extend our tools to incorporate joint dislocation as well as muscle tearing. Our muscle simulation framework should allow us to incorporate these principles. We will have to develop new tools for simulation baking that can handle the occurrence of muscle tearing and we will have to relax the rigging constraints to allow bone separation.

We are working with other researchers to develop methods of model fitting from medical imaging. Ideally, these techniques will allow us to automate much of the modeling process, including strand paths, origin and insertion points, and several of the muscle parameters. This will allow us to build parameterized models using our tools, which can be fitted to patient specific data. Such models would greatly increase the accuracy of our simulations, allowing our models to be used directly in clinical procedures. It may even become possible to mimic muscle injuries and neural damage on a patient to patient level.

Bibliography

[1] I. Albrecht, J. Haber, and H.-P. Seidel. Construction and animation of anatomically based human hand models. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 98–109, Aire-la-Ville, Switzerland, 2003. Eurographics Association.

[2] A. Aubel and D. Thalmann. Interactive modeling of the human musculature. In *Proceedings of Computer Animation 2001*, pages 167–255, 2001.

[3] Autodesk. 3ds Max, 2008. URL http://autodesk.com.

[4] Autodesk. Maya, 2008. URL http://autodesk.com.

[5] Avid. Softimage XSI, 2008. URL http://www.softimage.com/.

[6] J. Baumgarte. Stabilization of constraints and integrals of motion in dynamical systems. *Computer Methods in Applied Mechanics and Engineering*, 1:1–16, June 1972.

[7] M. Bergou, M. Wardetzky, S. Robinson, B. Audoly, and E. Grinspun. Discrete Elastic Rods. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2008)*, 27(3):63:1–63:12, Aug. 2008.

[8] F. Bertails, B. Audoly, M.-P. Cani, B. Querleux, F. Leroy, and J.-L. Lévque. Super-helices for predicting the dynamics of natural hair. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2006)*, 25(3):1180–1187, 2006.

[9] S. S. Blemker and S. L. Delp. Three-dimensional representation of complex muscle architectures and geometries. *Annals of Biomedical Engineering*, 33(5):661–673, May 2005.

[10] Blender Foundation. Blender, 2008. URL http://www.blender.org/.

[11] cgCharacter. Absolute Character Tools, 2006. URL http://cgCharacter.com.

[12] E. Y. S. Chao. Graphic-based musculoskeletal model for biomechanical analysis and animation. *Medical Engineering & Physics*, 25(3):201–212, April 2003.

[13] I. Charlton and G. Johnson. A model for the prediction of the forces at the glenohumeral joint. In *Proceedings of the Institute of Mechanical Engineering*, pages 801–812, Nov. 2006.

[14] P. Coleman and K. Singh. Cords: Geometric curve primitives for modeling contact. *IEEE Computer Graphics and Applications*, 26(3): 72–79, 2006.

[15] J. de Groot and R. Brand. A three-dimensional regression model of the shoulder rhythm. *Clinical Biomechanics*, pages 735–743, Nov. 2001.

[16] S. L. Delp and J. P. Loan. A computational framework for simulating and analyzing human and animal movement. *Computing in Science & Engineering*, 2(5):46–55, 2000.

[17] S. L. Delp and J. P. Loan. A graphics-based software system to develop and analyze models of musculoskeletal structures. *Computing in Biology & Medicine*, 25(1):21–34, 1995.

[18] P. Faloutsos, M. van de Panne, and D. Terzopoulos. Composable controllers for physics-based character animation. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2001)*, pages 251–260, 2001.

[19] B. Garner and M. Pandy. The obstacle-set method for representing muscle paths in musculoskeletal models. *Computational Methods for Biomechanics & Biomedical Engineering*, 3(1):1–30, 2000.

[20] G. Hamilton, editor. *JavaBeans^{TM} API Specification*. Sun Microsystems Inc., California, 1.01-a edition, Aug. 1997.

[21] C. Hoegfors, D. Karlsson, and B. Petersen. Structure and internal consistency of a shoulder model. *Journal of Biomechanics*, 28(7), 1995.

[22] K. Holzbaur, W. Murray, and S. Delp. A model of the upper extremity for simulating musculoskeletal surgery and analyzing neuromuscular control. *Annals of Biomedical Engineering*, pages 829–840, June 2005.

[23] G. Johnson, N. Bogduk, A. Nowitzke, and D. House. Anatomy and actions of the trapezius muscle. *Clinical Biomechanics*, 9:44–50, 1994.

[24] G. Johnson, D. Spalding, A. Nowitzke, and N. Bogduk. Modelling the muscles of the scapula: Morphometric and coordinate data and functional implications. *Journal of Biomechanics*, 29(8):1039–1051, 1996.

[25] J. Kaldor, D. James, and S. Marschner. Simulating knitted cloth at the yarn level. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2008)*, 27(3):65:1–65:9, Aug. 2008.

[26] M. Kass and J. Anderson. Animating oscillatory motion with overlap: Wiggly Splines. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2008)*, 27(3):28:1–28:8, Aug. 2008.

[27] A. Kaufman, S. Sueda, and D. K. Pai. Dynamic shoulder simulation using musculoskeletal strands. In *Proceedings of the seventh conference of the International Shoulder Group*, June 2008.

[28] P. G. Kry and D. K. Pai. Interaction capture and synthesis. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2006)*, 25(3): 872–880, August 2006.

[29] S.-H. Lee and D. Terzopoulos. Heads up!: biomechanical modeling and neuromuscular control of the neck. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2006)*, 25(3):1188–1198, 2006.

[30] S.-H. Lee and D. Terzopoulos. Spline joints for multibody dynamics. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2008)*, 27(3):22:1–22:8, Aug. 2008.

[31] Y. Lee, D. Terzopoulos, and K. Walters. Realistic modeling for facial animation. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 1995)*, pages 55–62, 1995.

[32] J. Lenoir, L. Grisoni, P. Meseure, Y. Rémion, and C. Chaillou. Smooth constraints for spline variational modeling. In *Proceedings of GRAPHITE 2004*, pages 58–64, 2004.

[33] D. I. W. Levin, B. Gilles, B. Madler, and D. K. Pai. A fiber tracking method for building patient specific dynamic musculoskeletal models from diffusion tensor data. In *MICCAI Workshop on Computational Diffusion MRI*, page (to appear), 2008.

[34] Massive Software. Massive, 2008. URL http://www.massivesoftware.com/.

[35] W. Maurel and D. Thalmann. Human shoulder modeling including scapulo-thoracic constraint and joint sinus cones. *Computers & Graphics*, pages 203–218, 2000.

[36] W. Maurel, D. Thalmann, P. Hoffmeyer, P. Beylot, P. Gingins, P. Kalra, and N. Magnenat-Thalmann. A biomechanical musculoskeletal model of human upper limb for dynamic simulation. *Biomedical Imaging*, 2000.

[37] K. L. Moore and A. F. Dalley. *Clinically oriented anatomy.* Lippincott Williams & Wilkins, fourth edition, 1999.

[38] R. M. Murray, Z. Li, and S. S. Sastry. *A Mathematical Introduction to Robotic Manipulation.* CRC Press, 1994.

[39] Musculographics Inc. SIMM: Software for Interactive Muskuloskeletal Modeling, 2007. URL http://www.musculographics.com/.

[40] V. Ng-Thow-Hing. *Anatomically-based models for physical and geometric reconstruction of humans and other animals.* PhD thesis, The University of Toronto, 2001.

[41] D. K. Pai. STRANDS: Interactive simulation of thin solids using Cosserat models. In *Proceedings of Eurographics 2002*, pages 347–352, 2002.

[42] N. S. Pollard and V. B. Zordan. Physically based grasping control from example. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 311–318, New York, NY, USA, 2005. ACM Press.

[43] H. Qin and D. Terzopoulos. D-NURBS: A Physics-Based Framework for Geometric Design. *IEEE Transactions on Visualization and Computer Graphics*, 2(1):85–96, 1996.

[44] J. Rasmussen, M. Damsgaard, S. T. Christensen, and M. de Zee. Anybody - decoding the human musculoskeletal system by computational mechanics. *Konferanse i beregningsorientert mekanikk (invited paper)*, 2005.

[45] Y. Remion, J. Nourrit, and D. Gillard. Dynamic animation of spline like objects. In *Proceedings of the 1999 WSCG Conference*, pages 426–432, 1999.

[46] Rhythm & Hues Studios, 2006. URL http://www.rhythm.com/.

[47] K. Ritchie, J. Callery, and K. Biri. *The Art of Rigging: A definitive guide to character technical direction with Alias Maya*, volume 1. CG Toolkit, California, 2005.

[48] F. Scheepers, R. E. Parent, W. E. Carlson, and S. F. May. Anatomy-based modeling of the human musculature. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 1997)*, pages 163–172, 1997.

[49] W. Shao and V. Ng-Thow-Hing. A general joint component framework for realistic articulation in human characters. In *Proceedings of the 2003 Symposium on Interactive 3D Graphics*, Apr. 2003.

[50] Side Effects Software. Houdini, 2008. URL http://www.sidefx.com/.

[51] E. Sifakis, I. Neverov, and R. Fedkiw. Automatic determination of facial muscle activations from sparse motion capture marker data. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2005)*, 24(3):417–425, 2005.

[52] SimTk. OpenSim, 2008. URL https://simtk.org/home/opensim.

[53] Smith Micro. Poser, 2008. URL http://my.smithmicro.com/.

[54] SolidWorks Corporation. SolidWorks, 2008. URL http://www.solidworks.com/.

[55] J. Spillmann and M. Teschner. An adaptive contact model for the robust simulation of knots. *Computer Graphics Forum (Proceedings of Eurographics 2008)*, 27(2):497–506, Apr. 2008.

[56] J. Spillmann and M. Teschner. 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, pages 63–72, 2007.

[57] S. Sueda, A. Kaufman, and D. K. Pai. Musculotendon simulation for hand animation. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2008)*, 27(3):83:1–83:8, Aug. 2008.

[58] J. Teran, S. Blemker, V. N. T. Hing, and R. Fedkiw. Finite volume methods for the simulation of skeletal muscle. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 68–74, 2003.

[59] J. Teran, E. Sifakis, S. S. Blemker, V. Ng-Thow-Hing, C. Lau, and R. Fedkiw. Creating and simulating skeletal muscle from the visible human data set. *IEEE Transactions on Visualization and Computer Graphics*, 11(3):317–328, 2005.

[60] The Alfred Mann Institute. MMS: Musculoskeletal Modeling in Simulink, 2008. URL http://ami.usc.edu/.

[61] The Alfred Mann Institute. MSMS: MusculoSkeletal Modeling Software, 2008. URL http://ami.usc.edu/.

[62] The Alfred Mann Institute. Virtual Muscle, 2008. URL http://ami.usc.edu/.

[63] D. G. Thelen and F. C. Anderson. Using computed muscle control to generate forward dynamic simulations of human walking from experimental data. *Journal of Biomechanics*, 39:1107–1115, 2006.

[64] D. G. Thelen, F. C. Anderson, and S. L. Delp. Generating dynamic simulations of movement using computed muscle control. *Journal of Biomechanics*, 36:321–328, 2002.

[65] Tippett Studios, 2000. URL http://www.tippett.com/.

[66] G. Tortora and S. Grabowski. *Principles of Anatomy & Physiology*. John Wiley & Sons Inc, New York; Toronto, ninth edition, Jan. 2000.

[67] W. Tsang, K. Singh, and E. Fiume. Helping hand: an anatomically accurate inverse dynamics solution for unconstrained hand motion. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 319–328. ACM Press, 2005.

[68] United States National Library of Medicine. The Visible Human Project®, 2008. URL http://www.nlm.nih.gov/research/visible/.

[69] F. Valero-Cuevas, J.-W. Yi, D. Brown, R. McNamara, C. Paul, and H. Lipson. The tendon network of the fingers performs anatomical computation at a macroscopic scale. *IEEE Transactions on Biomedical Engineering*, 54(6):1161–1166, June 2007.

[70] F. C. T. van der Helm. A finite musculoskeletal model of the shoulder mechanism. *Journal of Biomechanics*, 27(5):551–569, 1994.

[71] H. Veeger, F. C. T. van der Helm, L. van der Woude, G. Pronk, and R. Rozendal. Inertia and muscle contraction parameters for musculoskeletal modelling of the shoulder mechanism. *Journal of Biomechanics*, 24(7):615–629, 1991.

[72] M. Viceconti, D. Testi, F. Taddei, S. Martelli, G. Clapworthy, and S. Jan. Biomechanics modeling of the musculoskeletal apparatus: status and key issues. *Proceedings of the IEEE*, 94(4):725–739, Apr. 2006.

[73] K. Waters. A muscle model for animation three-dimensional facial expression. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 1987)*, pages 17–24, 1987.

[74] R. Weinstein, E. Guendelman, and R. Fedkiw. Impulse-based control of joints and muscles. *IEEE Transactions on Visualization and Computer Graphics*, 14(1):37–46, 2008.

[75] J. Wilhelms and A. V. Gelder. Anatomically based modeling. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 1997)*, pages 173–180, 1997.

[76] D. D. Wilkinson, M. V. Weghe, and Y. Matsuoka. An extensor mechanism for an anatomical robotic hand. In *Proceedings of the 2003 IEEE International Conference on Robotics & Automation*, 2003.

[77] F. T. H. Wu, V. Ng-Thow-Hing, K. Singh, A. M. Agur, and N. H. McKee. Computational representation of the aponeuroses as NURBS surfaces in 3D musculoskeletal models. *Computer Methods & Programs in Biomedicine*, 88(2):112–122, 2007.

[78] K. Yin, M. B. Cline, and D. K. Pai. Motion perturbation based on simple neuromotor control models. In *Proceedings of Pacific Graphics 2003*, pages 445–449, 2003.

[79] F. Zajac. Muscle and tendon: properties, models, scaling, and application to biomechanics and motor control. *Critical Reviews in Biomedical Engineering*, 17(4):359–411, 1989.

[80] Q.-h. Zhu, Y. Chen, and A. Kaufman. Real-time biomechanically-based muscle volume deformation using fem. *Computer Graphics Forum*, 17(3):275–284, 1998.

[81] V. B. Zordan, B. Celly, B. Chiu, and P. C. DiLorenzo. Breathe easy: model and control of simulated respiration for animation. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 29–37, 2004.

[82] V. B. Zordan, A. Majkowska, B. Chiu, and M. Fast. Dynamic response for motion capture animation. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2005)*, 24(3):697–701, 2005.

Appendix A

SMUSCLES User Guide

The primary function of this document is to provide a comprehensive user guide for the SMUSCLES plug-in. The SMUSCLES plug-in is a set of tools designed for modeling musculoskeletal systems. Its intention is to make the modeling of these complex systems accessible to artists, biologists, biomechanists, and other less-technical users. It was designed specifically to be used in conjunction with the musculoskeletal simulation software being developed by the Sensorimotor Systems Lab at the University of British Columbia.

The SMUSLCES plug-in has been developed for Autodesk's Maya [4] in order to leverage the production quality animation tools already familiar to many artists. The design principles used in the development of the SMUSCLES plug-in should easily lend themselves to further development using any professional 3D modeling program.

In this document I will explain how to use the SMUSCLES plug-in for Maya. This user guide will assume a basic knowledge of the Maya interface and built-in tools. If at any time you do not understand a reference to a Maya tool, I encourage you to learn about it in the Maya help files. Maya has several very good documentation and help files, which can be found under the Help menu in Maya. I find the following help files the most useful:

- Maya Help – general information and tutorials on everything in Maya
- MEL Command Reference – explanation of MEL script commands
- Python Command Reference – explanation of Python script commands
- Node and Attribute Reference – explanation of built-in object types and attributes

The most recent SMUSCLES release is supported by Maya 2008. If you have a different version of Maya, be sure to use the appropriate plug-in release.

A.1 Loading and Reloading the Plug-in

You must follow the procedure below to install the plug-in when you first begin using it or when new versions of the plug-in are released.

1. Download the version of the plug-in corresponding to your version of Maya (i.e. Maya 8.5 or Maya 2008)
2. Extract smusclesMaya2008.zip to the Maya user path (i.e. C:/Documents and Settings/User Name/My Documents/maya/)
3. Copy sslMuscles.py and paste it in the Maya Plug-in path (i.e. C:/Program Files/Autodesk/Maya2008/bin/plug-ins/)
4. In Maya, select Window–>Settings/Preferences–>Plug-in Manager
5. Find sslMuscles.py in the list of plug-ins. If it is not there, try refreshing
6. Select the "Load" and "Auto Load" options

When you exit Maya, your shelf will be saved. The sslMuscles tab will exist until you unload the plug-in or delete it manually.

Sometimes when Maya crashes, it fails to save the proper system state. This may cause problems with the plug-in functionality. If you experience problems with the plug-in procedures, or if you have made changes to any of the plug-in script files, then you should try reloading the plug-in using the following steps:

1. Select Window–>Settings/Preferences–>Plug-in Manager
2. Find sslMuscles.py in the list of plug-ins
3. Uncheck the "Loaded" option

4. Allow Maya to delete the sslMuscles shelf
5. Recheck the "Loaded" option
6. Make sure the "Auto Load" option is checked

A.2 sslMuscles Shelf Tab

The majority of the functionality of the plug-in can be accessed using the buttons on the sslMuscles shelf tab (Fig. 3.2). If you do not have an sslMuscles shelf tab, refer to Appendix A.1 in order to create one.

Many of the functions within the shelf have relaxed error checking. This allows a script to run on a selected list of objects, automatically ignoring any incompatible selections rather then issuing a formal error or warning. If nothing appears to happen when you use one of the functions, it is most likely that you have incompatible items selected. Check the Script Editor for important feedback details.

A.3 Primitives

There are five types of primitives in the SMUSCLES plug-in: rigid bodies, joints, strands, muscle groups, and constraints.

Rigid Bodies

To create an sslRigidBody, push the Rigid Body Button (picture of a single bone). This will add sslRigidBody attributes to all of the selected objects. These attributes can be edited from the channel editor. For a list of sslRigidBody attributes, see Table 3.1.

It will also automatically generate a centerOfMass locator, which will be placed at a best guess location. This center of mass can be moved manually after creation.

While any object can become an sslRigidBody, I strongly encourage following the conventions defined in Sec. 3.3 in order to ensure compatibility with the simulation software.

Joints

To create an sslJoint, select two sslRigidBodies, followed by the object which should become an sslJoint. Push the Joint Button (picture of two connected bones). This will add sslJoint attributes to the last selected object. These

attributes can be edited from the channel editor. For a list of sslJoint attributes, see Table 3.1.

While any object can become an sslJoint, I strongly encourage following the conventions defined in Sec. 3.3 in order to ensure compatibility with the simulation software.

Strands

A strand is the basic muscle structure used in this plug-in. Build your strands using any of Maya's built in NURBS curve creation tools, or if you know the strand paths, with the scripting commands. You can also fill in a muscle mesh using the Auto Strands Tool (Appendix A.4). I recommend using the EP Curve Tool and manually tweaking the control points if you are creating paths by hand.

To convert your NURBS curves to sslStrands, select the curves and push the Strand Button (picture of thin muscle tissue). Provide Passive and Active FL curve XML files to the dialog prompts, or hit Cancel to use the default values.

This will add sslStrand attributes to all of the selected NURBS curves (and sslMuscleGroup attributes to all of the selected groups). These attributes can be edited from the channel editor. For a list of sslStrand attributes, see Table 3.1.

Muscle Groups

A muscle group is a muscle structure that consists of a collection of strands. To create an sslMuscleGroup, push the Strand Button (picture of thin muscle tissue). This will add sslMuscleGroup attributes to all of the selected groups (and sslStrand attributes to all of the selected NURBS curves). These attributes can be edited from the channel editor. For a list of sslMuscleGroup attributes, see Table 3.1.

Constraints

A constraint is a locator. Constraints are used to mark locations along the strand that should be constrained at simulation time. There are several types of constraints, each with its own button (marked by colored circles and letters). For an explanation of the purpose and functionality of each constraint, see Sec. 3.3.

Fixed Constraints: The F Button adds fixed constraints between all of the selected sslStrands and the last selected object.

Surface Constraints: The Su Button adds surface constraints between all of the selected sslStrands and the last selected NURBS surface.

Sliding Constraints: The Sl Button adds sliding constraints between all of the selected sslStrands and the last selected sslRigidBody.

Strut Constraints: The St Button adds strut constraints between all of the selected sslStrands.

A.4 AutoStrands

To use the AutoStrands Tool (Sec. 3.4.1), select one, or several, input strands, along with an enclosing mesh and push the AutoStrands Button (picture of thick muscle tissue). This will launch the Create AutoStrands Window (Fig. 3.4).

There are several simulation parameters to tweak before starting an autostrand simulation. The Stiffness controls each autostrand's "desire" to become a straight line. The Repulsion Force controls its "desire" to move away from all other strands in the simulation.

Duplicates per Strand determines the number of autostrands that will be created for each input strand, while Samples per Strand determines how many points along the input strand will be sampled for the creation of each autostrand. Rebuild Strands controls the number of control points (cps) that each autostrand contains. An autostrand can either match the number of cps from its input curve, or it can base its cps on the initial sampling.

Once all of the parameters have been set, press the Start AutoStrand Thread Button. This will launch a new thread that the simulation can run in (Sec. 3.2.4). The simulation will update the autostrands in the Maya viewport at every timestep. It can be paused/continued using the Play/Pause AutoStrands Thread Button. The Stiffness, Repulsion Force, and Rebuild Strands parameters can all be modified on the fly while the simulation is running. Once the autostrands have moved to a final position that you are comfortable with, press the Kill AutoStrands Thread Button. The autostrands are now essentially identical to regular strands and can be manually manipulated to further refine their shape.

A.5 Removing SSL Attributes

The Remove Attributes Tool is used to remove all SSL attributes from objects that no longer need them, or that have outdated versions of SSL attributes and need to be updated. Essentially, it takes any of the SMUSCLES primitives and converts them back into the corresponding original Maya primitive.

To remove SSL attributes, select the desired objects and press the Remove Attributes Button (picture of a bone and muscle under a "no smoking" symbol). Please note that removing SSL attributes commonly results in broken constraints. These broken constraints are automatically deleted by the Remove Attributes Tool. Once you have removed SSL attributes, you may have to have recreate some constraints.

A.6 Exporting a ModelerScene

In order to interface with the Java simulation software, you will need to export your Maya scene as a ModelerScene XML file. The ModelerScene XML Exporter currently supports the following objects:

- Polygons
- NURBS curves
- NURBS surfaces
- Locators
- Object Hierarchy (parents and groups)

The following steps explain how to export a ModelerScene XML file:

1. Open a Maya scene
2. Select the objects that you want to export
   - NOTE: All visible objects within a selected object's hierarchy will be exported. Hide objects that you don't want to export.
   - NOTE: All polygons will be triangulated if they are not already triangular.
   - NOTE: You should delete history on the objects that you export because extra polyShapes often cause export problems.
3. Click the JavaBean XML Button (picture of Java) in the sslMuscles shelf tab
4. Use the file dialog pop-up to choose a directory and name for your file

- NOTE: If you are exporting many objects, it may take some time once you hit save. To monitor your progress, open the Script Editor before Step 3.

A.7 Exporting sslMuscle Data

In addition to the ModelerScene XML file mentioned in Appendix A.6, you will need to export the sslMuscle data to a separate XML file. The reasoning behind this second file is explained in Sec. 3.5. Follow these steps in order to export the sslMuscle data:

1. Open a Maya scene
2. Add the relevant sslMuscle data using the plug-in buttons
3. Select the objects that you want to export
   - NOTE: All visible objects within a selected object's hierarchy with sslMuscle data will be exported. Hide objects that you don't want to export.
   - NOTE: You should delete history on the objects that you export as extra polyShapes often cause export problems.
4. Click your "Muscle XML" Button in the sslMuscles shelf tab
5. Use the file dialog pop-up to choose a directory and name for your file
6. Use the dialog pop-up to choose the ModelerScene XML file that this new XML file corresponds to.
7. Use the next dialog pop-up to type the name of the Activation class. If there is no Activation class, press Cancel.
   - NOTE: Depending on how you call this file in Java, you may have to hand edit the Scene filename and Activation class.

A.8 Exporting Control Data

If you want to track an animation during a simulation, you can first create the animation in Maya and export it to an XML file with the plug-in by following these steps:

1. Open a Maya scene

2. Select the objects that you want to export keyframe data for
    - NOTE: Position and orientation data will be exported for the selected visible sslRigidBodies only.
    - NOTE: This data will be exported at every frame between the first and last keyframe found on the selected visible objects
3. Click your "Control XML" Button in the sslMuscles shelf tab
4. Use the file dialog pop-up to choose a directory and name for your file

A.9 Baking Simulations

Once you have run and saved a Java simulation that you like using the simulation software, you can bake that simulation onto the strands in your Maya scene (Sec. 3.4.2). To bake the simulation data as keyframes on your Maya strand control points, push the Bake Simulation Button (picture of looping film reel).

Use the windows file dialog pop-up to locate a Replay XML file (note that this XML file must have been created in a Java simulation using the ModelerScene class). Once you choose a file, the plug-in will add keyframes to the cps of the visible strands with names matching those in the Replay XML file. See Sec. 3.5 for an explanation of Replay XML files.

Be aware that baking simulations disables the undo function and that Maya often crashes when trying to bake too much simulation data at once. For best results, break up your simulation data into several smaller Replay XML files and save your Maya scene after you bake each one.

A.10 Skinning Strands

Skinning strands is a complicated process. The current implementation is only acceptable for skinning thin muscles and tendons close to the surface of the skin.

The effects of the skinning process can be controlled from the Skin Strands Window (Fig 3.3). The deformation will take place for every frame starting with the Start Frame and ending before the End Frame. The Height parameter controls the height of the offset deformation, Lateral Falloff controls the falloff factor in the lateral direction, and Normal Falloff controls the falloff factor in the normal direction. For a more technical explanation of the skinning process see Sec. 3.4.3.

If you wish to attach skin to your strands, the following steps will guide you through the skinning process:

1. Rig and skin a base mesh to your animated rigid bodies
2. Duplicate this mesh
3. Rename both meshes. The names must be the same, but the base mesh name must end in "Base"
4. Add a new color set to the base mesh using "Create Empty Color Set" from the Polygons–>Color Menu
5. Paint vertex influence weights using Maya's Paint Vertex Color Tool
    - NOTE: Only the R channel of the color set affects deformation
6. Select the sslStrands that will drive the deformation, followed by the new mesh that will be deformed
    - NOTE: The sslStrands must already have the animation motion keyframed (Appendix A.9) before this step
7. Click your Skin Strands Button (picture of thick muscle tissue surrounded by a blue mesh) in the sslMuscles shelf tab
8. Set the appropriate parameters in the Skin Strands Window as explained above
9. Click the "Start Deformation" Button
    - NOTE: It may take some time once you begin the deformation process. To monitor your progress, open the Script Editor before Step 9

The deformed skin mesh will need to have its normals recalculated for every frame. To do this, use "Set To Face" and "Average Normals" from the Polygons–>Normals Menu at the current viewing frame.