Strand-based Musculotendon Simulation of the Hand

by

Shinjiro Sueda

B.Sc., The University of British Columbia, 2002

M.Sc., Rutgers University, 2006

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

Doctor of Philosophy

in

THE FACULTY OF GRADUATE STUDIES

(Computer Science)

The University of British Columbia

(Vancouver)

December 2010

Abstract

This dissertation develops a framework for modelling biomechanical systems, with special focus on the muscles, tendons, and bones of the human hand. Two complementary approaches for understanding the functions of the hand are developed: the strand simulator for computer modelling, and an imaging apparatus for acquiring a rich data set from cadaver hands.

Previous biomechanical simulation approaches, based on either lines-of-force or solid mechanics models, are not well-suited for the hand, where multiple contact constraints make it difficult to route muscles and tendons effectively. In lines-of-force models, wrapping surfaces are used to approximate the curved paths of tendons and muscles near joints. These surfaces affect only the kinematics, and not the dynamics, of musculotendons. In solid mechanics models, the 3D deformation of muscles can be fully accounted for, but these models are difficult to create and expensive to simulate; moreover, the fibre-like properties of muscles are not directly represented and must be added on as auxiliary functions. Neither of these approaches properly handles both the *dynamics* of the musculotendons and the complex routing *constraints*. We present a new, strand-based approach, capable of handling the coupled dynamics of muscles, tendons, and bones through various

types of routing constraints.

The functions of the hand can also be studied from the analysis of data obtained from a cadaver hand. We present a hardware and software setup for scanning a cadaver hand that is capable of *simultaneously* obtaining the skeletal trajectory, tendon tension and excursion, and tendon marker motion. We finish with a preliminary qualitative comparison of a simulation model of the index finger with real world data acquired from *ex vivo* specimen, using the strands framework.

Preface

Some portions of this dissertation are taken from papers that have already been published. In addition to the listing here, each section of the dissertation that is taken from other manuscripts is indicated by a footnote with a bibliographic reference at the beginning of the section.

Chapter 4 on the Cosserat strand is from (D. K. Pai, S. Sueda, and Q. Wei. Fast physically based musculoskeletal simulation. In ACM SIGGRAPH '05: Sketches, page 25, New York, NY, USA, 2005. ACM.) [71], and from the present author's master's essay. The text in the introductory paragraphs of the chapter and in Section 4.3 is based largely on Pai et al. [71], portions of which were written by Dinesh Pai.

The text in Chapter 5 on the Spline strand is based on (S. Sueda, A. Kaufman, and D. K. Pai. Musculotendon simulation for hand animation. ACM Trans. Graph. (Proc. SIGGRAPH), 27(3), 2008.) [88], which was written by the present author, and edited by Dinesh Pai. Some sections are modified to better fit this dissertation.

The text in Chapter 6 on the Reduced strand is written by the present author, and does not appear in any other manuscript.

Chapter 7 on the controller is also from (S. Sueda, A. Kaufman, and D. K.

Pai. Musculotendon simulation for hand animation. ACM Trans. Graph. (Proc. SIGGRAPH), 27(3), 2008.) [88], written by the present author. The section on saccades (Section 7.1) is taken almost verbatim with minor edits from (Q. Wei, S. Sueda, and D. K. Pai. Physically-based modeling and simulation of extraocular muscles. Progress in Biophysics and Molecular Biology, 103 (2-3):273-283, 2010.) [102], written by Qi Wei, who conducted the experiment using the Spline simulator from Chapter 5 and the controller from Chapter 7.

The cadaver hand imaging work in Chapter 8 was performed in collaboration with Mitsunori Tada (National Institute of Advanced Industrial Science and Technology, Tokyo), and Yusaku Kamata (Keio University, Tokyo). The dissections were performed by Yusaku Kamata, and the preliminary data analyses were performed by Mitsunori Tada. The text in this chapter is written by the present author, and does not appear in any other manuscript.

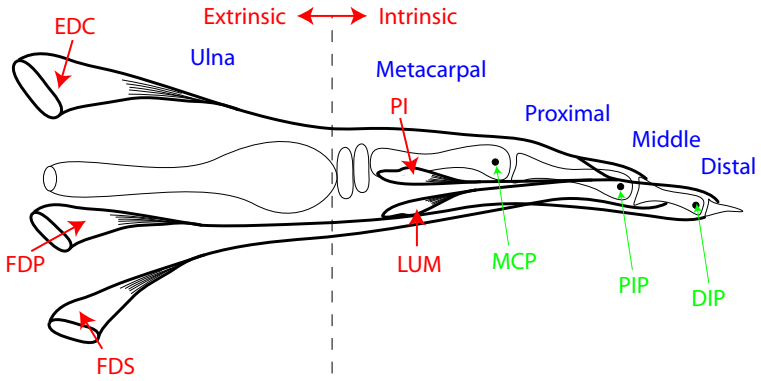
The UBC Clinical Research Ethics Board has reviewed the research on cadavers described in Chapter 8, and has found the research project acceptable on ethical grounds for research involving human subjects (UBC CREB number H10-00222).

Table of Contents

List of Tables

List of Figures

Glossary



Simplified anatomy of the finger. Musculotendons are labelled in red. (The EDI, which runs parallel to the EDC, and DI, which is on the other side of the bone, are not shown.) Bones are labelled in blue, and the joints are labelled in green.

EOM Extraocular Muscles, the muscles that control the globe (eyeball) in the orbit (eye socket).

LR Lateral Rectus, the EOM that abducts the eye (away from the nose).

MR Medial Rectus, the EOM that adducts the eye (towards the nose).

SR Superior Rectus, the EOM that elevates the eye.

ABN Abducens Neurons, the neuron that controls the lateral rectus muscle.

FDP Flexor Digitorum Profundus, the deep flexor tendon that inserts into the distal phalanx.

FDS Flexor Digitorum Superficialis, the superficial flexor tendon that inserts into the middle phalanx.

EDC Extensor Digitorum Communis, the extensor tendon that is present in all four fingers.

EDI Extensor Digitorum Indicis, also known as Extensor Indicis Proprius, the extensor tendon that is present only in the index finger, and runs parallel to the EDC.

LUM Lumbrical, the intrinsic muscle that originates from the FDP and inserts into the extensor hood structure.

DI Dorsal Interosseus, the intrinsic muscle located between the thumb and the index finger that abducts the index finger (away from the middle finger).

PI Palmar Interosseus, the intrinsic muscle located between the index and the middle fingers that adducts the index finger (towards the middle finger).

MCP Metacarpophalangeal Joint, the joint between the proximal and metacarpal bones, located at the base of the finger.

PIP Proximal Interphalangeal Joint, the joint between the proximal and middle phalangeal bones.

DIP Distal Interphalangeal Joint, the joint between the middle and distal phalangeal bones.

Acknowledgments

I would like to thank my supervisor, Dinesh Pai, for his guidance and kindness during this work, and for having the patience to allow me to work on multiple iterations of the strand simulator.

I would also like to thank the members of the examination committee, Professors Uri Ascher, Peter Cripton, Antony Hodgson, Nancy Pollard, and Michiel van de Panne, for their helpful comments.

I am very grateful to have been blessed with such wonderful lab mates and collaborators. Special thanks to Qi, Danny, Dave, Garrett, and Andrew, for working with me on various papers. I would be hard pressed to find a better lab than the Sensorimotor Lab. Tim, Sang Hoon, Martin, Mahk, Benjamin, Paul, and all other lab mates past and present, thank you for all your help and support.

I am also grateful to Mitsunori Tada, Yusaku Kamata, and the people at the Digital Human Research Center and Keio University, for their collaboration on cadaver data collection and analysis.

Lastly, I would like to thank my mom. Thank you for having the courage to cross the Pacific all those years ago with three kids.

Shinjiro Sueda

Chapter 1

Introduction

Musculoskeletal simulations provide insight into the biomechanical functions of anatomical structures, such as the extraocular muscles of the eye [26], and the extensor mechanism of the finger [97]. With the steady increase in computational power, large-scale simulation of the musculoskeletal system is becoming more practical, and has been successfully applied in a wide range of disciplines. Computational simulations are indispensable in the field of motor control, serving as a test bed for evaluating different control strategies [93]. They are also important for surgical planning [16] and ergonomics studies [83]. For researchers in functional electrical stimulation (FES), having a robust simulator is critical for evaluating prototype models before implanting them into patients [23], as well as for understanding proprioception [59]. In human kinetics, effects of injuries, diseases, and aging can be studied using simulation [105, 106]. Computer simulations are also used in biopaleontology, for recovering the gait patterns of extinct animals, such as the Tyrannosaurus rex [38].

The strand simulator also has applications in computer graphics. In Section

5.4, we show how a strand simulator can be used to automatically produce subcutaneous motion of tendons and muscles underneath the skin, adding realism to an input animation. The tendons and muscles are automatically skinned to the character's surface skin, providing complex secondary motion of the skin as a result of biomechanically realistic tendon motion.

Simulation of musculoskeletal dynamics with tendons and complex routing constraints is one of the most difficult challenges in computational biomechanics. Examples of these complex routing constraints include the tendinous hood of the extensor mechanism of the finger and the tendons of the carpal tunnel [27, 97]. In most situations, tendons move freely in the axial direction, even in highly constraining configurations. In some cases, however, they can also move in the lateral direction, such as the finger extensor tendons during extension. Moreover, a tendon that wraps around bones can exert forces on the bones not only at the origin and insertion, but also along its length.

In both the biomechanics and graphics communities, there have been a number of computational models proposed for the simulation of musculotendons, hair, wire, and other thin structures. (See Chapter 2 for more details.) All of the models based on lines-of-force muscles (e.g., [24, 41, 50, 51, 56, 74]) work well on musculotendons whose routing constraints are not too stringent. However, these models do not account for the mass of the muscles, which may cause errors that can be non-negligible even on a simple mono-articular model [70]. Various dynamic models [8, 9, 35, 86] could potentially be used for musculotendon simulation, but these models were designed for use in free-floating configurations, and do not work well in highly-constraining situations.

The challenges discussed above motivated us to develop a novel and efficient

biomechanical *strand simulator*, which can simulate thin, strand-like soft tissues with complex routing constraints, such as tendons, muscles, and ligaments (Chapters 4-6). Our focus is on the *large-scale, hyper-elastic behaviour* of muscles responsible for skeletal movement, rather than on the low-level details of the muscle mechanics, such as hysteresis and residual force enhancement. Fibre level muscle mechanics is an ongoing work amongst our collaborators [107]; in this dissertation, we simply treat it as a black-box that takes as input the current state (position, velocity, activation, etc.) of the muscles and returns as output the computed contractile and passive elastic forces.

1.1 Thesis Contributions

This thesis presents two complementary approaches for understanding the functions of the hand: strand-based musculotendon simulation and acquisition real world data from medical images of a cadaver hand.

The first contribution is the strand simulation framework. Using the terminology of Pai [69], *strands* are space curves with mass, elasticity, and other physical properties that influence their dynamics. Our decision to use strands was motivated by the anatomical structure of real muscle tissue. Muscles consist of fibres, curved in space, which are bundled into groups called fascicles. When a muscle is activated, the fibres contract and transmit a contractile force directly along each fibre. By using strands in our muscle simulation, we are able to directly model this behaviour. Strands allow us to define smooth curves to represent tendons, muscles, or even the individual fascicles of each muscle. Unlike the lines-of-force models, the strands are always simulated *along with* the rigid bodies—they have mass, inertia, and momentum, and interact with the rigid bodies through constraint forces.

An important modelling decision we face is on the choice of representation of the strand. With the Euler-Lagrange equation, we can describe the motion of a strand using any reasonable degrees of freedom, and choosing the most appropriate generalized coordinates for the application is key to the robustness, efficiency, and overall usability of the system. One can decide to use, for example, a chain of rigid bodies [35], splines [42, 73, 88], Super-Helices (curvature and twist) [9, 10], quasistatic frames [7], quaternion segments [86], or mass-springs [79, 80].

The main goal of this thesis is to develop a musculoskeletal simulator for the hand that satisfies the following two requirements: (1) ability to handle complex constraints, and (2) ability to properly transmit force around these constraints. None of the lines-of-force models [25, 74], solid-mechanics models [11, 18, 45, 90], or other strand models from graphics [7, 10, 35, 54, 69, 79, 86] satisfy these criteria completely. The strand framework provides both of these important advantages. It is well suited for the hand and other parts of the body with complex routing constraints, but, just like lines-of-force simulators, it can easily be used for many musculoskeletal systems in the body. The strand framework gives the added benefit of the coupling of dynamics between bones, muscles, and tendons, as well as the ability to deal with complex routing constraints. These two advantages will be illustrated with simple examples in Section 9.1.1.

After reviewing related work (Chapter 2), and some background material (Chapter 3), we will describe the evolution of the hand simulator (Chapters 4-6).

- Cosserat Strand, parameterized by a series of rigid frames (Chapter 4).
- Spline Strand, parameterized by the control points of a B-spline curve (Chapter 5).
- Reduced Strand, parameterized by a series of constrained nodes (Chapter 6).

The evolution of the strand simulator hinges on one important observation: *Handling of complex constraints is the most important aspect of hand simulation.* There are two subsequent observations that stem from the main observation.

The first observation is that, for hand simulation, we know a-priori where the routing constraints are going to be. The main strength of the Cosserat Strand is in the generality of collision detection and handling, but for hand simulation, we do not need such a general formulation of contact and its associated cost in simulation and modelling time. With the Spline Strand, we can have smooth routing constraints at various predefined locations along the strand.

The second observation is that, for hand simulation, we know a-priori the regions on the strand that are going to need degrees of freedom, and the regions that don't need degrees of freedom. Therefore, we can reduce the degrees of freedom as much as possible while still maintaining mass, inertia, and other material properties of the hand, by building the constraints into the degrees of freedom themselves, rather than applying external constraints. With the Reduced Strand, we can selectively choose to work in reduced coordinates in some regions of the strand, rather than in maximal coordinates.

Muscle control computation—the estimation of the activation levels necessary to produce desired skeletal motion through muscle contractions—is an important aspect of musculoskeletal simulation. Because of the complexity of the routing of tendons and muscles and the large number of degrees of freedom of the hand, even simple tasks, such as moving a finger from one position to another, are virtually impossible without an algorithmic controller. Matters are further complicated by the fact that muscles rarely contract as a single unit; rather, they are activated as synergistic groups or antagonistic pairs. In Chapter 7, we describe an incremental

Simulator	Energy modes	Contact model
Cosserat	Stretching, bending, & twisting	Explicit with mesh
Spline*	Stretching & bending†	Predefined constraint locations
Reduced	Stretching‡	Implicit with constrained DoFs

Table 1.1: Strand simulator comparison. *The Spline Strand can be formulated as a special case of the Reduced Strand. See Section 9.3. †These two modes are coupled in an unintuitive manner. See Section 5.5. ‡Bending energy can be added using the formulation described in Discrete Elastic Rods [7].

algorithmic controller that determines the muscle activation levels required for the skeleton to dynamically follow a kinematic trajectory. Our controller has been used successfully for controlling the hand (Section 5.4), eye (Section 7.1), and jaw [87].

Finally, in Chapter 8, we describe our cadaver scanning system, with which we are able to *simultaneously* acquire a rich data set consisting of: skeletal trajectory, tendon excursion, tendon tension, and tendon motion within the finger. This approach offers a different avenue for understanding the functions of the hand, and is especially useful for the hand, because the complex routing of the tendons makes it difficult to model. We finish with a preliminary simulation of an index finger model using the strands framework and the data acquired from *ex vivo* specimen.

We conclude, in Chapter 9, by summarizing the thesis, discussing and expanding on some of the key points, and by outlining some of the future work.

Table 1.1 shows a comparison of the strengths and limitations of the three strand simulators. In Section 9.3, we discuss some future work for the Reduced Strand, which would make it a superset of the Spline Strand. Once these modifications are made, we are still left with two choices: The Cosserat Strand and the Reduced Strand. For modelling some parts of the body that have a large number of volumetric muscles in close contact, the Cosserat Strand, with its more general

collision detection and handling scheme, may be more appropriate. However, if we want to simulate the hand, the Reduced Strand is clearly the best choice.

Chapter 2

Related Work

We will first review several muscle models used widely in both computer graphics and biomechanics. We will then cover work related to strand simulation. We will briefly discuss various methods for musculoskeletal control, and end with a review of related work on hand imaging.

2.1 Musculoskeletal Simulation

Biomechanics

Much of the work on musculoskeletal models has its roots in the biomechanics community, and can be broadly categorized into two types: lines-of-force and solid mechanics. These approaches have different strengths and weaknesses, depending on the application. Lines-of-force models are most useful for simulations with a large number of muscles, because of their robustness and simplicity, and solid mechanics models are useful for simulating a smaller number of large, volumetric muscles in high detail. Our aim is to strike a good balance between the two.

Lines-of-force models

The first type is based on lines-of-force approaches, such as OpenSim [24, 25] and AnyBody [74], where muscles and tendons are represented as abstract lines with no mass. These approaches are popular because they are simple to implement, robust, and computationally inexpensive. Although these approaches simulate the dynamics of the bones, they only consider the kinematics, or the simplified paths, of the muscles and tendons. At each simulation step, the strain, the strain rate, and the moment-arm of the musculotendon are calculated using the current joint configuration, and then these values are used in turn to compute the joint torques, resulting in skeletal movement. Although this method is simple and efficient, it cannot capture the full dynamics of bones and musculotendons, such as surface contact forces between bones and tendons, as well as tendon bifurcation. Another important detail not captured by these models is the non-uniform strain along muscles and tendons. Uniform strain is a reasonable assumption under static or simple routing conditions, but during fast dynamic motions, some parts of the tendon may stretch more than others. Furthermore, if a muscle has complex fibre organizations or cross sectional area varying along its length, then the strain also varies along the length.

The static simulator based on relaxation by Lipson [56] is an interesting approach for modelling tendons. Although this approach is computationally expensive, it works well for static simulations of isometric experiments, and can reproduce a wide range of complex behaviours, such as the nonlinear tension propagation in the extensor tendon network of the fingers [97]. The strand model of Johnson et al. [41] uses weightless expandable threads to simulate the statics of the

finger muscles and tendons. Like OpenSim and Lipson [56]'s model, their strands are massless, and cannot simulate the dynamics of the muscles and tendons. Even on a simple mono-articular model, the error from ignoring the mass of the strand can be non-negligible [70].

Lee and Kamper [51] simulated the coupled joint dynamics of the finger by deriving the equations of motion for the index finger movement generated by the flexor tendon that inserts into the distal phalanx. They then compared their simulation result to the moment-arm approach, and found that their method produced a more realistic joint coordination pattern. Similarly to other models, their work does not take into account the mass and inertia of the tendons. Also, their approach is very specific to the index finger, and each anatomical structure needs to be hard-coded in, whereas our approach is general, and can be used for any simulation scene with rigid bodies and strands.

Solid mechanics models

The second category of models is based on solid mechanics. These models are more accurate than lines-of-force models, especially for non-fusiform muscles, because they treat muscles as deformable visco-elastic 3D solids. The finite element method [11, 18, 45], finite volume method [90], and B-spline solids [66] all belong to this category. These methods are well suited for simulating large volumetric muscles, and are useful for applications where a few number of muscles are to be simulated in high detail. However, they are inefficient at simulating thin, strand-like muscles and tendons. Discretizing a thin tendon into volumes, for example, is highly inefficient and redundant. By representing soft tissues as volumes, solid mechanics models do not naturally capture one of the important characteristics of

musculotendons—the primary direction of force generation and propagation. For the applications that we are interested in, we believe that the appropriate modelling primitive should represent the curved axis of the basic unit of a musculotendon, rather than its volume. Furthermore, these models are not as robust as the lines-of-force models under large skeletal motions, and obtaining anatomically-correct parameters for these models is extremely difficult.

Graphics

There has also been significant development of musculoskeletal models in the graphics community. Several models incorporate realistic muscle anatomy but are restricted to static simulations [1, 77, 104]. Ng-Thow-Hing [66] and Teran et al. [90, 91] developed volumetric muscle models to simulate muscles with both active and passive components in their muscle mechanics (constitutive models). Zhu et al. [110] used a linear elastic muscle model along with finite elements for muscle volume deformation. Maural et al. [58] and Aubel and Thalmann [4] constructed muscle-based virtual human characters. Musculoskeletal models have also been used extensively for facial animation [52, 84, 100]. Shao and Ng-Thow-Hing [82] presented a general joint component framework designed to improve the realism of joint articulation in virtual humans. Lee and Terzopoulos [49] used a neuromuscular control model for the simulation of a human neck. Zordan et al. [111] developed muscle elements based on springs for simulated respiration. Lee et al. [50] combined line-of-force muscles with a volumetric simulation of the soft tissue.

Robotics

Although not directly related to our work on software simulations, there has been numerous work on robotic models of the hand. The Utah-MIT hand [39] was one of the earliest tendon-driven robotic hands. It was anatomically motivated, but did not try to be anatomically correct, and included many pulleys and tendons that do not correspond to actual tendons in the human hand. The Shadow robot hand [57] and the NAIST Hand 2 [47] are also anatomically motivated, the tendons are not anatomically-based, with each joint driven by a belt connected to a motor. This allows them to control each joint independently, making the control simpler. The ACT hand, a physical robotic hand developed at the University of Washington [27], is more anatomically correct, and even incorporates the model of the extensor mechanism. It is able to produce many of the finger poses, but does not have a proper model of the lumbrical, and therefore cannot produce some finger poses such as the hook-flexion.

2.2 Strand Simulation

Our muscle-tendon model relies on simulating thin structures, which we call *strands* using the terminology of Pai [69], as the basic modelling primitive. During the last decade, many efficient methods have been proposed for spatial discretization of strands for efficient dynamic simulation of various phenomena, such as hair, yarn, and cables.

The Cosserat strand was introduced to graphics by Pai [69]. Originally, these strands were limited to static simulations and only supported constraints at the ends. In Chapter 4, we extend these strands to make them more suitable for musculoskeletal simulation.

Mass-spring models have been used for its simplicity, but its main drawback is that the a mass-spring based strand cannot bend freely away from the mass points. Selle et al. [79] successfully applied the mass-spring model to hair animation, by leveraging the simplicity of the approach and scaling their system to tens of thousands of strands. They also added stable auxiliary springs between every other node and every three nodes for bending and twisting forces. These models work robustly and scales well, but mass lumping causes undesirable effects.

Bergou et al. [7] added the twist energy to a mass-spring system by treating the material frame quasistatically. The twist energy at each frame was formulated as the deviation from the natural (Bishop) frame, starting from the root of the strand. These discrete elastic rods work well for free strands with few constraints, and they validate their approach by simulating Michell's buckling instability. They later extended the approach to include viscous threads [8]. This approach has also been used for the simulation of yarn-based cloth [43].

The CORDE strand by Spillmann and Teschner [85] is another discrete rod approach that, unlike the discrete elastic rod [7], is fully dynamic. Originally, their weakness was in contact handling, but they later improved the model by adding adaptive resampling based on energy-change minimization [86]. Adaptive sampling is a good approach for adding extra degrees of freedom so that the strand can bend more freely, but the force computation and integration steps and the adaptive sampling step are carried forward in a staggered fashion. In contrast, with the virtual node formulation, the adaptivity of the degree of freedom is built into the system, eliminating the need for the staggered stepping scheme.

Hadap [35] used a chain of rigid bodies to simulate a strand. Such a method is attractive because all three energy modes can be modelled, and also because there

is a wide range of linear-time implementations already present in the literature [30]. This approach is suitable for free-floating strands, where only the root and the tip of the strands are fixed, with a small number of constraints along the way. For muscles and tendons, we need strands that can more easily be constrained along (possibly) their whole lengths.

Spline models, which were introduced by Qin and Terzopoulos [73], have been used for cable [54], yarn [42], and muscle simulation [88]. These models have the advantage over mass-spring models in that they are inherently smooth (C^2 continuous, if using cubic B-splines), and a sliding constraint can be applied efficiently with the "smooth" constraints of Lenoir et al. [54]. However, the stretching and bending modes become coupled with this approach, which can be difficult or impossible to deal with, if accurate stretching energy is required.

Bertails et al. [10] introduced Super-Helices, a strand simulator designed specifically for hair. Their formulation has the advantage over other models in that inextensibility is built into the system, since the degrees of freedom of each Super-Helix segment are its curvature and twist. The nodal positions are not explicitly computed, but are calculated from the hair root by traversing the strand based on the computed curvature and twist values. Bertails [9] later formulated a linear-time algorithm similar to the rigid body chain algorithm that allowed the simulation of hundreds of Super-Helix elements at interactive rates. Super-Helices can be used for hair, trees, and plants where the connectivity is a directed-acyclic-graph; it cannot model muscles and tendons, which contain cycles.

CORDS by Coleman and Singh [20] is useful for modelling static strands that wrap around scene geometry. It is not directly applicable to our work, since their strands are purely kinematic, but could be an interesting alternative for modelling

the "curve nodes" in our framework (Section 6.3.2).

One of the most serious disadvantages of all the strand simulation frameworks described above is that the strand cannot bend at an arbitrary point along its length. This is especially important when we are trying to model the tendons of the hand with strand-like primitives with complex routing constraints. To overcome this difficulty, Servin et al. [81] introduced the "virtual node" to the mass-spring model, and used their simulation framework for modelling cables in a virtual reality application. Our strand simulator (Chapter 6) extends their approach, and has two important advantages—more accurate computation of energies and more general handling of constraints. We integrate the kinetic and potential energies of a strand along its length, rather than lumping them at the nodes. By integrating these quantities, we can obtain the proper inertia of the strand, which can make a significant difference in a simulation, as shown in Section 9.1.1. In addition, with our simulator, we can mix and match different types of node (Section 6.3), whereas in their simulator, they are limited to a small subset of node types. In fact, the reduced strand framework is a generalization of the massless contact node framework. The additional types of nodes give us the flexibility required to robustly model the complex routing constraints of the tendons in the hand.

2.3 Muscle Control

There has been significant work in developing algorithmic controllers for joint-torque based simulators that do not take muscles into account [29, 72]. Some controllers are able to achieve high realism by incorporating motion and force capture data [46, 108, 112].

Among the approaches that model muscles explicitly and solve for their control

signals, many use joint moment-arms, which are a commonly used biomechanical approximation of distributed muscle forces around joints [92, 93, 95]. Sifakis et al. [84] determined activations of a detailed non-linear, but quasistatic, FEM muscle model. Weinstein et al. [103] used a novel approach to PD control to compute the joint torques required to follow an input target trajectory. Lee and Terzopoulos [49] used neural networks to learn the control of the complex musculature of the neck.

2.4 Hand Imaging

There has been a number of studies on obtaining the skeletal configuration, the tendon excursion, and/or the moment arm of a finger. To the best of our knowledge, no other group has used a medical imaging technique to capture the kinematic trajectory of the skeleton and the tendons of the hand.

Valero-Cuevas et al. [97] used a cadaver to measure the tensions in the proximal and terminal slips of the middle finger. The finger tip was rigidly attached to a 6-DoF force/torque sensor, and then tension was applied to the extensor and the interossei muscles. The resulting finger-tip force and the tendon insertion tensions were measured.

Nimbarte et al. [68] studied the effects of pulling on the extrinsic muscles of the hand, by using a motion-capture system and a cadaver. The finger was positioned at its rest state, and the tendons were pulled at a constant rate for 10 seconds. They then measured the resulting joint angles of the finger and showed that there was strong inter-joint coordination even when a single tendon was pulled in isolation. In our isolated lumbrical experiment, we augment this approach with the addition of variable weights on the lumbrical (Chapter 8).

It is also possible to obtain some of these measurements from *in vivo* experi-

ments. Mitton et al. [61] showed that the accuracy of reconstruction from biplanar X-rays was comparable to that from CT, which is important when the amount of radiation needs to be kept low. Nikanjam et al. [67] inserted buckle transducers into the tendons of patients during an open carpal tunnel release surgery to record the tendon tensions across multiple joint angles. Chao et al. [17, §7] measured finger tip force and the electromyographic (EMG) activity in the hand muscles during isometric contractions, and analytically determined the force distribution in the muscles. van den Doel et al. [98] used computer myography to estimate the muscle activation levels from surface electromyography data of the upper arm, with the goal of extending this technique to measure activity in hand muscles. Chang and Pollard [15] solved for the skeletal joint parameters, such as joint axis orientation and location, from a sparse set of markers on the skin. These methods have the advantage over cadaver experiments in that they are performed *in vivo*, but they can only acquire a subset of the data that we desire.

2.5 Summary

None of the aforementioned simulation techniques work well for the hand. Lines-of-force models [25, 74] have difficulties dealing with kinematic loops and cannot properly account for the dynamic interactions between bones, muscles, and tendons. Solid-mechanics models [11, 18, 45, 90] work well for volumetric muscles in isolation, but have difficulties with thin muscles and tendons, and with the complex routing constraints of the hand. Various techniques from the graphics community [7, 10, 35, 54, 69, 79, 86] work well for free-floating strands, but are not efficient at modelling the axial transmission of force across complex routing constraints.

Thus, there is a need for a robust musculoskeletal simulator that can handle

both the coupling of dynamics between bones, muscles, and tendons, as well as complex routing constraints. The strand framework provides both of these important advantages. It is well suited for the hand and other parts of the body with complex routing constraints, but, just like lines-of-force simulators, it can easily be used for the whole body, with the added benefit of the two advantages listed above.

Chapter 3

Background

In all three of the strand simulators, we use the rigid body simulation framework of Cline and Pai [19] and Kaufman et al. [44]. Following their notation, we will use bold letters, $\boldsymbol{x}$, to denote vectors and points in $\mathbb{R}^3$, and sans serif letters, q, to denote generalized coordinates and related quantities. Everywhere possible, q will refer to the generalized coordinates of a body, and $\boldsymbol{x}$ will refer to a point on the body in $\mathbb{R}^3$. Time derivative will be indicated with a dot, $\dot{\boldsymbol{x}} \equiv d\boldsymbol{x}/dt$, and the material derivative will be indicated with a prime, $\boldsymbol{x}' \equiv d\boldsymbol{x}/du$.

3.1 Rigid Bodies

The configuration of a rigid body is represented by the usual 4×4 transformation matrix consisting of rotational and translational components,

$$ {}^{0}_{i}\mathsf{E} = \begin{pmatrix} {}^{0}_{i}\Theta & {}^{0}_{i}\boldsymbol{p} \\ 0 & 1 \end{pmatrix}. \tag{3.1} $$

The leading subscripts and superscripts indicate that the coordinates of rigid body (or frame) i are defined with respect to the world frame, 0. Thus each column of ${}^0_i\Theta$ corresponds to the frame's basis vectors, $\boldsymbol{e}_k$, expressed in world coordinates, and ${}^0_i\boldsymbol{p}$ is the position of the frame's origin expressed in world coordinates. Given a local position ${}^i\boldsymbol{x}$ on a rigid body, its world position is

$$ {}^0\boldsymbol{x} = {}^0_i\mathsf{E}\,{}^i\boldsymbol{x}. \tag{3.2} $$

Unless otherwise stated, we will assume that the reference frame is the world frame, and use a trailing subscript to indicate the local frame, as in E_i. With this notation, E_i transforms a position from the local space of the i^{th} rigid body to world space. There are other parameterizations for the rotational component of the rigid body, such as unit quaternions and Euler angles, but we chose to use rotation matrices because they allow us to transform vectors from one coordinate frame to another very efficiently.

The spatial velocity ϕ of a frame E describes the motion of the frame at time t. The spatial velocity is composed of the angular component, $\boldsymbol{\omega}$, and the linear component, $\boldsymbol{v}$, both expressed in body coordinates, and is defined by

$$ \mathsf{E}^{-1}\dot{\mathsf{E}} = \begin{pmatrix} [\boldsymbol{\omega}] & \boldsymbol{v} \\ 0 & 0 \end{pmatrix}, \tag{3.3} $$

where the "dot" indicates time-derivative, and the 3×3 matrix, $[\boldsymbol{a}]$, is the cross-product matrix such that $[\boldsymbol{a}]\boldsymbol{b} = \boldsymbol{a} \times \boldsymbol{b}$. $\boldsymbol{\omega}$ and $\boldsymbol{v}$ are repackaged into the 6-vector

ϕ.

$$ {}^{i}\phi_i = \begin{pmatrix} {}^{i}\boldsymbol{\omega}_i \\ {}^{i}\boldsymbol{v}_i \end{pmatrix}, \tag{3.4} $$

which is the angular and linear velocity of the rigid body in its own body coordinates with respect to the world frame. The world space velocity, ${}^{0}\dot{\boldsymbol{x}}$, of a point in local coordinates ${}^{i}\boldsymbol{x}$ of a rigid body moving with spatial velocity ${}^{i}\phi_i$ is

$$ \begin{aligned} {}^{0}\dot{\boldsymbol{x}} &= {}^{0}_{i}\Theta \left(-[{}^{i}\boldsymbol{x}]\ I\right) {}^{i}\phi_i \\ &= \Gamma({}^{0}_{i}\mathsf{E}, {}^{i}\boldsymbol{x})\, {}^{i}\phi_i \end{aligned} \tag{3.5} $$

where the 3×6 matrix, Γ, called the material Jacobian matrix, transforms the local spatial velocity of the rigid body, ${}^{i}\phi_i$, into the velocity of a local point, ${}^{0}\dot{\boldsymbol{x}}$, on the rigid body in world coordinates. Its transpose, a 6×3 matrix, transforms a point force, $\boldsymbol{f}_i$ into a world wrench,

$$ \mathsf{f}_o = \Gamma^T \boldsymbol{f}_i. \tag{3.6} $$

The spatial velocity transform from one frame to another according to the adjoint of the coordinate transform, and is easy to define from a rigid transform ${}^{0}_{i}\mathsf{E}$.

$$ {}^{0}_{i}Ad = \begin{pmatrix} {}^{0}_{i}\Theta & 0 \\ [{}^{0}_{i}\boldsymbol{p}]\,{}^{0}_{i}\Theta & {}^{0}_{i}\Theta \end{pmatrix}. \tag{3.7} $$

The twist of frame i in world coordinates is then

$$ {}^{0}\phi_i = {}^{0}_{i}Ad\, {}^{i}\phi_i. \tag{3.8} $$

If we ignore the angular component of a twist, it is easy to see that a 3D velocity is transformed simply with the rotation matrix,

$$ {}^0\dot{\boldsymbol{x}} = {}^0_i\Theta\, {}^i\boldsymbol{v}. \tag{3.9} $$

Again, we will suppress the leading superscript for clarity and write ϕ_i, assuming that all twist quantities are expressed in local coordinates.

3.2 Constrained Dynamics

Expressing the twist of a frame in local coordinates has the advantage that the mass matrix becomes diagonal in the dynamics equations. The Newton-Euler equations of motion of a single frame can be written in a compact form as

$$ \begin{aligned} \mathsf{M}\dot{\phi} &= [\text{coriolis forces}] + [\text{body forces (e.g., gravity)}] \\ &= \phi_\times^T \mathsf{M}\phi + B(\mathsf{E}). \end{aligned} \tag{3.10} $$

Here, M is the spatial inertia of the frame, and $\phi_\times$ is the spatial cross product matrix,

$$ \phi_\times = \begin{pmatrix} [\boldsymbol{\omega}] & 0 \\ [\boldsymbol{v}] & [\boldsymbol{\omega}] \end{pmatrix}. \tag{3.11} $$

We use the following time-stepping discretization at the velocity level to obtain the discrete impulse-momentum equations at time $t^{(k)}$.

$$ \mathsf{M}\phi^{(k+1)} = \mathsf{M}\phi^{(k)} + h\left(({\phi_\times}^{(k)})^T \mathsf{M}\phi^{(k)} + B(\mathsf{E}^{(k)}) \right). \tag{3.12} $$

Joint constraints between rigid bodies are implemented using the adjoint formulation [65], with which we can easily derive different types of joints simply by dropping rows in the 6×6 adjoint matrix. Given two frames, i and k, and a joint frame defined with respect to the first frame, ${}^{i}_{j}\mathsf{E}$, we constrain the frames' spatial velocities, ${}^{i}\phi_{i}$ and ${}^{k}\phi_{k}$, with respect to the joint frame. Using Equation 3.8, the relative velocity at joint j is given by

$$\begin{aligned} \delta\phi_{j} &= {}^{j}_{i}Ad\,{}^{i}\phi_{i} - {}^{j}_{k}Ad\,{}^{k}\phi_{k} \\ &= \begin{pmatrix} {}^{j}_{i}Ad & -{}^{j}_{i}Ad\,{}^{i}_{k}Ad \end{pmatrix} \begin{pmatrix} {}^{i}\phi_{i} \\ {}^{k}\phi_{k} \end{pmatrix}. \end{aligned} \tag{3.13}$$

For a rigid joint, we want the relative velocities to be zero, so we set $\delta\phi = 0$. From this, we can derive different types of joints, by dropping various rows of the constraint equation: the top three rows (corresponding to the three rotational DoFs) for a ball joint, or the third row (corresponding to the rotation about the z-axis) for a hinge joint.

Writing these joint constraint equations in the form $G\phi = 0$, we simultaneously obtain the velocity that satisfies the constraints at the time $t^{(k+1)}$ and the contact impulse $G^{T}\lambda$ by solving the KKT system ([12])

$$\begin{pmatrix} M & G^{T} \\ G & 0 \end{pmatrix} \begin{pmatrix} \phi^{(k+1)} \\ \lambda \end{pmatrix} = \begin{pmatrix} \mathsf{f}^{(k)} \\ 0 \end{pmatrix}, \tag{3.14}$$

where λ is the vector of Lagrange multipliers for the constraints.

The frame $\mathsf{E}^{(k+1)}$ can be obtained by integrating $\phi^{(k+1)}$. We must be careful here, because E belongs to a non-Euclidean space (SE(3), the Special Euclidean

group in 3 dimensions). We use a first order implicit discretization, with the time step $h = t^{(k+1)} - t^{(k)}$:

$$\mathsf{E}^{(k+1)} = \mathsf{E}^{(k)} \exp\left(h \begin{pmatrix} [\boldsymbol{\omega}^{(k+1)}] & \boldsymbol{v}^{(k+1)} \\ 0 & 0 \end{pmatrix} \right). \tag{3.15}$$

The matrix exponential can be computed efficiently using Rodriguez' formula [65].

Chapter 4

Cosserat Strands

Overview: Cosserat strands are parameterized by a sequence of discrete frames, each associated with a 3D coordinate frame and an inertial mass. Since the strand mesh (i.e., muscle geometry) is used for collision detection and contact, Cosserat strands and are well-suited for volumetric, fusiform muscles.

- *Energy modes*: Stretching, bending, and twisting.
- *Contact handling*: Uses muscle and bone mesh.

The Cosserat strand is an extension of the formulation by Pai [69] for thin solid simulation, specifically designed for musculoskeletal simulation. A Cosserat strand, (just "strand" for the rest of this section), is composed of a series of 3D material frames arranged along the length of the muscle, and the state of the strand's dynamics at time t is described by the frames $\mathsf{E}_i \in SE(3)$ and their spatial velocities $\phi_i \in se(3)$ for all i. These "material" frames of the strands are embedded in the material represented by the strand, and are not defined based on the geometry

This chapter is based on Pai, Sueda, and Wei [71].

of the strand, like Frenet frames. A segment, the portion of the strand between two consecutive frames, is defined from these primary variables, and represent the geometry and the viscoelastic connection between the frames, such as strain and strain rate, from which the elastic energies are defined (Section 4.1). Intuitively, a frame is a small rigid body with mass, and the geometry is represented by the segment between two frames, as shown in Figure 4.1. The embedded mesh of the strand is deformed according to the configuration of the frames, and is used for collision detection and resolution (Section 4.3).

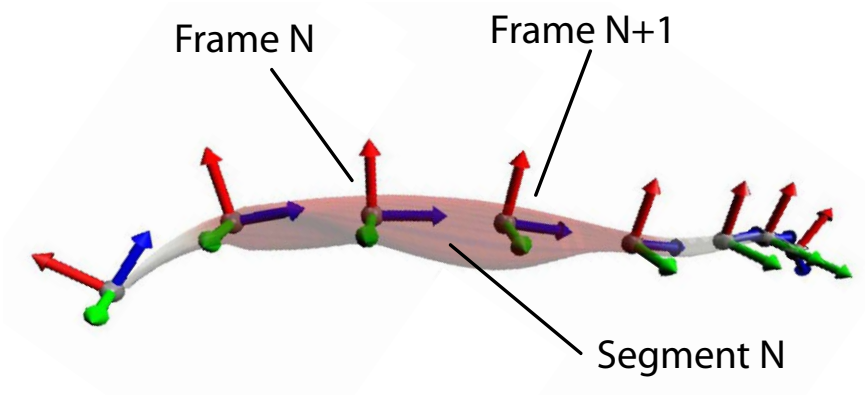


Figure 4.1: Discrete Cosserat strand. A muscle is discretized into frames, with the Z-axis aligned along the axis of the muscle.

Because Cosserat strands are composed of rigid frames, a rigid body can be treated as a special case of a Cosserat strand, with a single frame. This allows us to simulate the musculotendons and bones in a unified framework, with rigid frames as the one and only primitive in the system. To compute the mass matrix of bones, we align the Z-axis of each frame to the principal axis of each rigid body. We do the same for strand frames as well; each strand frame is aligned to match the musculotendon's axial direction. This has the added bonus that the nodal frames

are always aligned to the embedded material in the most natural way. Because the strand frames can be treated as rigid bodies, all the equations from Section 3.1 apply to strands—individual frame inertia, forces, and constraint terms are added to the appropriate submatrices in the KKT matrix (Equation 3.14).

4.1 Strand Forces

We account for anisotropy by modelling behaviour in the fibre direction separately from that in the transverse direction. This is the main motivation for our strand-based muscle model, and is the reason why lines-of-force models are very useful in biomechanics despite their limitations. A strand has three elastic modes: stretching, bending, and twisting, with the most important mode for muscle simulation being the stretching mode. Given the frames of a segment, E_i and E_j, the normal strain is computed trivially as $\varepsilon = \|\boldsymbol{x}_j - \boldsymbol{x}_i\|/L - 1$, where L is the rest length of the segment. These values are then fed into a black-box muscle model function that returns the forces to be applied to the two frames.

The other two modes are based on the relative rotations of the two frames. The bending energy is defined as a function of the difference between the displacement vector of the frame origins, $\Delta\boldsymbol{x} = \boldsymbol{x}_{i+1} - \boldsymbol{x}_i$, and the direction of the Z-axis at rest. Similarly, the twist energy is defined as a function of the difference between the X-axes of the frames.

4.2 Muscle Mechanics

As an elastic solid, muscle is made of a complex nonlinear viscoelastic material [109], and modelling and understanding its properties is an active area of research with divided views [28, 64, 107]. The main contentious issue is the behaviour of

the muscle in the "descending limb" of the force-length curve.

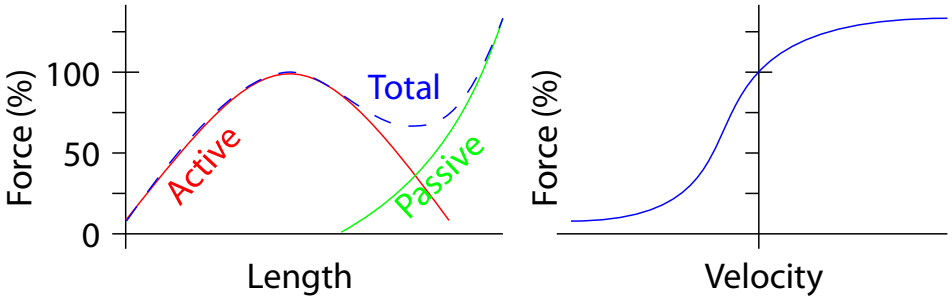


Figure 4.2: Typical Force-Length and Force-Velocity curves. As a muscle is stretched, the total force increases, then decreases, and then increases again. If the muscle is being shortened, its force decreases, and if the muscle is lengthened, its force increases.

The force exerted by a muscle is the sum of the passive and active forces. In the Hill-Zajac model [109], the passive force is assumed to be due to a hyperelastic material model, while the active force is computed as a function of the muscle's activation level, a, maximal (isometric) force, f_0, current length l, and current stretching speed, $\dot{l}$.

$$f_a(a, f_0, l, \dot{l}) = a \cdot f_0 \cdot FL(l) \cdot FV(\dot{l}). \tag{4.1}$$

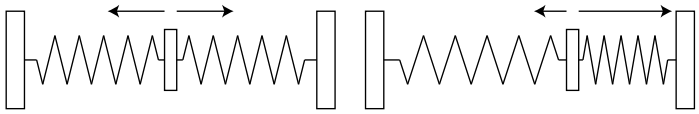
The Force-Length (FL) and Force-Velocity (FV) curves are measured empirically from isolated muscle experiments. First, the muscle's passive elasticity is measured by gradually increasing its length and tabulating the generated tension. Then the experiment is repeated with the muscle activated. The muscle at rest is stretched to the desired length, then maximally activated, and then the tension is measured. Depending on the muscle specimen, the resulting FL relationship often shows an area of negative slope—as the length is increased, the force decreases. The order

of operation here is critical, and is the source of the contention. If the muscle is activated first and then stretched to the desired length, it results in a significantly different FL relationship and does not show any areas of negative stiffness [28, p. 92]. This "history dependency" is also present in the ascending arm and the plateau of the FL curve, but is most prominent in the descending arm.

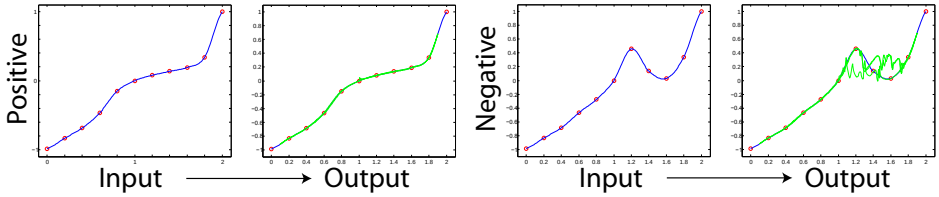
The slope is usually interpreted as stiffness, and this can cause instability in the muscle response. The reason for this instability can be demonstrated by imagining the muscle as being composed of two springs in series (Figure 4.3a). If the stiffness is positive, then any perturbation away from equilibrium causes the springs to exert restoring forces. If the stiffness is negative, however, the resulting springs forces cause further perturbation away from equilibrium, since the lengthened spring becomes weaker and the shortened spring becomes stronger.

Incorporating proper muscle mechanics is a key component of building an accurate musculoskeletal simulator. However, understanding and modelling a robust and accurate muscle model is out of scope of this thesis. One approach is to remove the area with negative slope with a straight horizontal line [71]. Instead we treat it as a black-box that returns the force as a function of the arguments in Equation 4.1.

Using the strand simulator, we can illustrate the impact of negative stiffness in a dynamic simulation. In a line-based approach such as OpenSim, each musculotendon can have only one strain value, and therefore does not suffer from these instabilities. With strands, on the other hand, a musculotendon has a continuous distribution of strain along its length, and the negative stiffness in muscle mechanics based on the Hill-Zajac model can sometimes cause severe instabilities, as shown in Figure 4.3b. In the left figure (labelled "Positive"), an FL curve with no negative slope is fed in as the black box muscle model. A single strand with



(a) Serial springs with negative stiffness

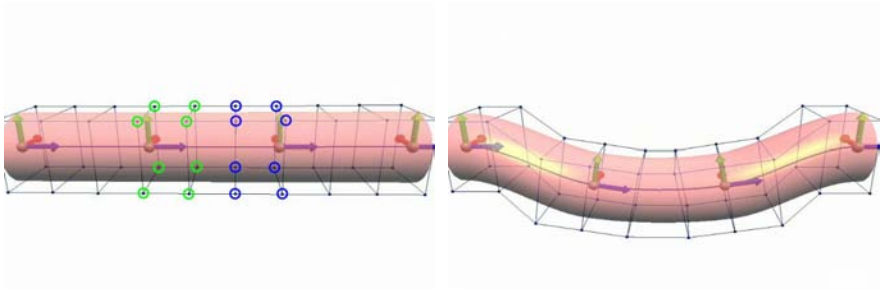


(b) Strand simulator result with negative stiffness

Figure 4.3: (a) If the springs have negative stiffness, then any perturbation away from the equilibrium results in forces that push the system further away from equilibrium. (b) Strand simulation with and without negative stiffness in the FL curve. The X-axis is the strain, and the Y-axis is the normalized force. In the left plots, the input FL curve (blue) contains no negative stiffness, and the resulting simulated strain (green) shows no instabilities. In the right plots, the input FL curve contains an area of negative stiffness, and the resulting simulated strain shows instabilities near the area.

the specified FL curve is stretched slowly (to suppress dynamic effects) and the resulting tension is plotted as the green curve, which shows a close match to the input FL curve. If there is an area with negative slope as in the right figure (labelled "Negative"), the resulting tension is unstable.

With the customizable FL and FV curves, we are able to insert any muscle mechanics model into a strand simulation, as long as it fits Equation 4.1. Depending on the application, we can choose different models. When simulating the subcutaneous motion of the hand for a graphics application, we can use a linear model (Section 5.4), and when simulating the saccade of an eye and comparing the results to analytical models, we can use an empirical muscle dynamics model taken from



(a) Undeformed FFD mesh **(b)** Deformed FFD mesh

Figure 4.4: (a) FFD mesh with control points. The green control points are rigidly attached to the 2nd frame, and the blue control points are rigidly attached to the 3rd frame. (b) When the frames move, the mesh is deformed accordingly.

experiments (Section 7.1).

4.3 Contact Detection and Resolution

Collision detection and resolution is one of the most difficult parts of a physical simulation. Simulators used in biomechanics partially address this problem by adding kinematic constraints corresponding to wrapping on spheres and cylinders [24, 31]. With these approaches, however, it is not possible to model muscles and tendons that wrap over the geometry of the bones and other deformed muscles. In general, proximity detection algorithms are expensive, and can quickly become the bottleneck of a simulation. In our case, however, the key observation is that we do not need the exact contact configuration. Rather, we just need to compute the local constraint directions for each frame. We use a fast, approximate collision detection algorithm that generates many, potentially noisy, contact points from the real geometry of the embedded mesh, and then find the least-constraining directions for each frame.

We use free-form deformation (FFD) [78]) to smoothly deform the embedded mesh (vertices and normals) as the frames move. In our implementation, the FFD grid on a segment is defined by a set of control points kinematically attached to the two frames—eight control points per frame. Four control points are arranged in a square around the X-Y plane of a frame, and four more are arranged in another square offset along the Z-axis, to achieve C^1 geometric continuity. Every mesh vertex is deformed by only taking into account the positions of these sixteen control points (Figure 4.4). If u, v, and w are the parameters of a vertex $\boldsymbol{x}$ in the grid, its position is interpolated from the control points $\boldsymbol{c}_{i,j,k}$ using trivariate Bernstein polynomials:

$$\boldsymbol{x}(u,v,w) = \sum_{i=0}^{l}\sum_{j=0}^{m}\sum_{k=0}^{n} B_i^l(u)B_i^m(v)B_k^l(w)\boldsymbol{c}_{i,j,k}. \tag{4.2}$$

We take advantage of the fact that muscles and tendons are always in close contact with the surrounding tissues, by using a voxel-based approach for collision detection, and equality (rather than inequality) constraints for contact resolution. General purpose proximity detection algorithms [40, 48] are not appropriate for objects in close contact such as muscles; they are designed to quickly prune away candidate contact points when objects are well separated. At each time step, vertices (positions and normals) of the deformed meshes are bucketed into voxels on the fixed grid, and then scanned to detect pairs of contacting vertices from different muscles and bones (See Figure 4.6). The voxel size is chosen so that many more contact points are detected than needed to determine a local linear constraint on the frame.

From these redundant and noisy contact points, the constraint on a frame is

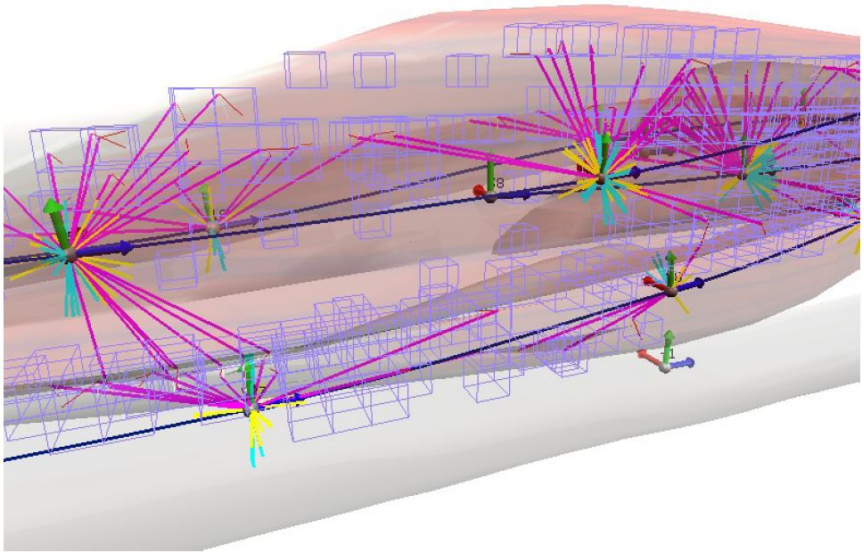


Figure 4.5: Contact detection on voxel grid. Voxels with vertices from two different meshes are shown. At these voxels, if collision is detected between the vertices, the normal of the offending vertex, shown in red, is lifted to the parent frame, shown by the thick magenta lines. Each detected normal of the deformed mesh applies a small wrench on the parent frame, shown in yellow and cyan.

estimated robustly using the singular value decomposition. The normals of each detected contact vertex can be viewed as a small abstract "force" on the frame. We then convert these 3D forces into 6D wrenches acting at the origin of the frame, using the Jacobian transpose from Equation 3.5. After traversing through all of the detected contacts, each frame will have a list of wrenches, which forms a matrix of size $c \times 6$, where $c > 6$. Then we perform an SVD on the matrix to extract the most constraining wrench directions. These directions form a submatrix of G for this frame, in Equation 3.14. A 2D illustration of the collision detection/handling process is shown in Figure 4.5. Two meshes (magenta and cyan) are voxelized and a pair-wise normal check is performed on all the vertices in a voxel. The offend-

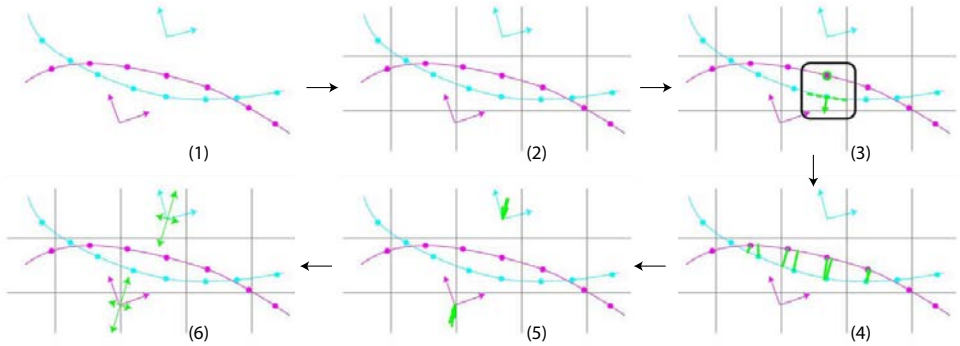


Figure 4.6: 2D illustration of collision detection and resolution. (1) The cyan mesh and magenta mesh come in contact. (2) The mesh vertices are bucketed into voxels, and (3-5) each offending pair in a voxel is added as a wrench, using the Jacobian transpose from Equation 3.5, to the parent rigid bodies. (6) Then SVD is applied at the parent frames to compute the least constraining directions.

ing normals are lifted to the parent frames, and the least constraining directions are computed with the SVD. Performing multiple SVDs per time step may seem expensive, but because the SVDs are performed per frame, with a relatively small number of wrenches, the cost is negligible compared to the KKT matrix solve. Also, because the SVD effectively estimates the active set from noisy contact data, we can treat the constraints as equality constraints, eliminating the need for QP or LCP solvers.

Having computed the least constraining directions for each frame with the SVD, a frame is no longer a free floating frame in $SE(3)$ but acts essentially as a jointed frame. Unlike a mechanical joint, however, these contact constraint matrices are recomputed at every time step, and their sizes can be different depending on the nature of the contacts detected at the time step.

4.4 Integration

We must be careful when choosing a numerical integrator to step the system forward in time. Our choice will have an impact on the efficiency, stability, and accuracy of the system. We used a linearly-implicit Euler stepping scheme [5, 19], because of its efficiency and stability. Although our system is stable for large step sizes, we must restrict the step size to avoid unacceptable numerical damping introduced by the implicit method. We currently use the same integration scheme in all three of our strand simulators (Chapters 5 & 6). It is important to note, however, that our system is not limited to this integrator.

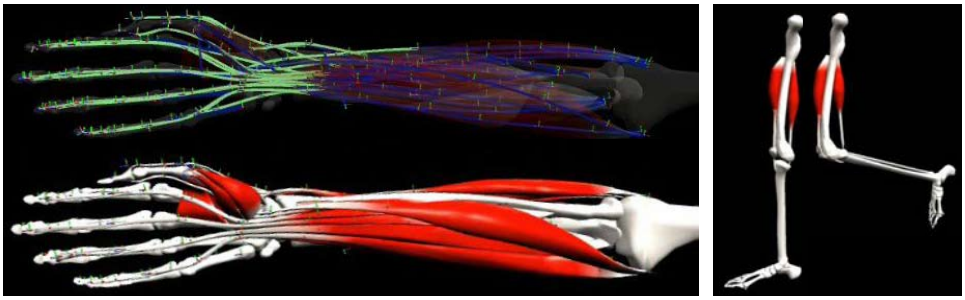
There are, however, some factors that contribute to the difficulty in incorporating other integrators. First, we are solving a differential algebraic equation, due to the presence of the constraints, rather than an ordinary differential equation. There are integrators that honour constraints, such as RATTLE [2], but because our rigid bodies are expressed in maximal, 6D-twist coordinates (rather than using joint angles, for example) live on a curved space, we cannot simply plug in RATTLE to our system and expect a guarantee of attractive properties, such as energy-preservation. Also, since muscles are very viscous, energy-preservation is not as critical as in other physical simulations like molecular dynamics or planetary simulation.

The linearly-implicit Euler scheme for non-linear systems, used extensively in computer graphics [3, 5, 19], is a variant of the fully implicit Euler scheme, where only one iteration of Newton search is taken per time step, rather than iterating until convergence. The rationale is that for small enough step sizes, the integrator will converge to the true solution quickly over multiple steps. This amounts to adding a linear Taylor expansion term about the current term and augmenting the

mass matrix with the gradients of stiff elastic and damping forces. Combining the dynamics equation with the augmented mass matrix, and the constraint equation, we obtain a linear system that we solve at each time step to obtain the generalized velocities at the next step (Equation 3.14), which we solve with a direct sparse method based on Gaussian Elimination [22].

4.5 Summary

The Cosserat strand is well-suited for volumetric muscles because it supports all three deformation energies, and also takes into account the mesh of the bones and muscles for computing the constraints. Some simulation examples of muscles in close contact are shown in Figure 4.7.



(a) Forearm strands **(b)** Sliding knee cap

Figure 4.7: (a) Forearm showing strands in the muscles. (b) The knee cap slides smoothly on top of the bone geometry.

Using the mesh itself for collision detection and response is a double-edged sword, however, since mesh becomes *necessary* for simulation. Segmenting and extracting muscles from medical images is an on-going research area, and by no means trivial [33], and segmenting tendons is even harder than muscles. (In Chapter 8, we will be describing our setup for measuring the motion of tendons *ex vitro*.)

In the simulations shown, we used the meshes that were purchased from cgCharacter [13] and cleaned up in Maya. We found that having enough mesh vertices is critical for collision detection and response, especially in the fingers, and adding extra vertices became a major performance bottleneck for simulating *thin* muscles and tendons. A mesh-based contact model is good for its generality, but for muscles and tendons, since we know *a priori* where the contact areas are going to be, we can use a simpler approach with pre-determined contact areas.

Chapter 5

Spline Strands

Overview: Spline strands are parameterized by the control points of dynamic spline curves with inertia, and are well-suited for thin muscles and tendons with complex routing constraints.

- *Energy modes*: Stretching and bending (coupled)
- *Contact handling*: Sliding and surface constraints at predefined contact locations

Although Cosserat strands work quite well for volumetric fusiform muscles, they are not well-suited for simulating thin muscles and tendons of the hand and fingers. For this, we use a new primitive which we call the *Spline* strand, an extension of the physically-based spline models previously used in computer graphics [53, 54, 73, 75] that includes activation (Section 5.1), and simple yet robust sliding and surface constraint models (Section 5.2).

The spline strand, or just "strand", for the rest of this chapter, unless otherwise

This chapter is based on Sueda, Kaufman, and Pai [88].

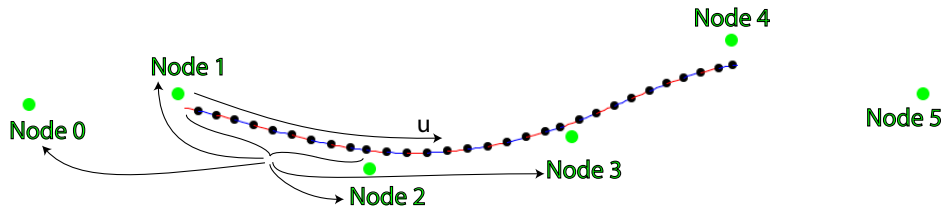


Figure 5.1: 2D spline strand, with the control points shown in green. Any point on the strand is defined by four control points and a scalar parameter u. The black circles illustrate the material points sampled at $u = 0.1, 0.2, 0.3, \ldots$. Note that for the computation of the dynamics, we integrate along u, rather than taking discrete samples.

stated, is a cubic spline curve with mass, elasticity, and other physical properties that influence its dynamics. It is parameterized by $n \geq 4$ control points, has $3n$ degrees of freedom, corresponding to the x, y, and z coordinates of the n control points. The state of the system is given by the stacked positions and velocities of the rigid bodies and strand control points. These are the generalized coordinates and velocities of the system, respectively:

$$\begin{aligned} \mathsf{q} &= \begin{bmatrix} \cdots & \mathsf{E}_i & \cdots & \boldsymbol{q}_j & \cdots \end{bmatrix}^T \\ \dot{\mathsf{q}} &= \begin{bmatrix} \cdots & \phi_i & \cdots & \dot{\boldsymbol{q}}_j & \cdots \end{bmatrix}^T. \end{aligned} \tag{5.1}$$

Here, $\mathsf{E}_i \in SE(3)$ and $\phi_i \in se(3)$ are the configuration and the spatial velocity, respectively, of the i^{th} rigid body ($SE(3)$ is the space of 3D positions and orientations, and $se(3)$ is the space of translational and rotational velocities. See Section 3.1), and $\boldsymbol{q}_j \in \mathbb{R}^3$ and $\dot{\boldsymbol{q}}_j \in \mathbb{R}^3$ are the position and the velocity of the j^{th} spline control point.

For each generalized coordinate, we construct an impulse-momentum equation

which, when discretized at the velocity level, is

$$\mathsf{M}\dot{\mathsf{q}}^{(k+1)} = \mathsf{M}\dot{\mathsf{q}}^{(k)} + h\mathsf{f} - G^T\lambda, \tag{5.2}$$

where M is the block-diagonal generalized mass matrix of rigid bodies and strand control points (derived from the kinetic energy of the spline [75]), h is the step size, f is the generalized force (body forces for rigid bodies, elastic and damping forces for strands, etc., Section 5.1), and $G^T\lambda$ is the constraint force (Section 5.2).

5.1 Strand Dynamics

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

$$\boldsymbol{x}(s,t) = \sum_{i=0}^{3} b_i(s)\boldsymbol{q}_i(t), \tag{5.3}$$

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 $\boldsymbol{x}(s,t)$ can be obtained in a similar manner.

$$\begin{aligned} \dot{\boldsymbol{x}}(s,t) &\equiv \frac{d\boldsymbol{x}}{dt} = \sum_{i=0}^{3} b_i(s)\dot{\boldsymbol{q}}_i(t) \\ \boldsymbol{x}'(s,t) &\equiv \frac{\partial \boldsymbol{x}}{\partial s} = \sum_{i=0}^{3} b_i'(s)\boldsymbol{q}_i(t). \end{aligned} \tag{5.4}$$

Based on these quantities, we can compute the passive and active elastic forces in the strand (which contribute to f in Equation 5.2). Our simulator has the ability

to use an arbitrary Force-Length (FL) relationship, which can be obtained from a standard Hill-type model [109] or from physiological experiments [107] (Section 4.2). Constitutive properties of muscles are, however, still not well established and are the subject of intense ongoing research. For graphics applications in Section 5.4, we use linear FL curves for both the passive and active forces, since they work well enough for producing realistic animations. The active force of a muscle is modelled as a linear FL curve with zero rest length.

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

$$M\dot{\boldsymbol{q}}^{(k+1)} = M\dot{\boldsymbol{q}}^{(k)} + h(\boldsymbol{f}_d + \boldsymbol{f}_g + \boldsymbol{f}_p + \boldsymbol{f}_a) - G^T\boldsymbol{\lambda}, \tag{5.5}$$

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, $\boldsymbol{f}_a = Aa$, where a is the vector of muscle activation levels between 0 (no activation) and 1 (full activation). The matrix A, which is of size (#DoF $\times$ #muscles), is the "activation transport" matrix that takes as input the activations of the muscles and outputs the corresponding forces on the strand DoFs. This separation of force and activation will help us later when deriving the controller in Chapter 7. The last term, $G^T\boldsymbol{\lambda}$, is the constraint force term, which will be discussed in Section 5.2.

The activation transport matrix is easy to compute for linear forces. Using the tangent vector function from Equation 5.4, the elastic potential energy at a point on the spline strand is

$$V(s) = \frac{1}{2}K\|\boldsymbol{x}'(s,t)\|^2 a \tag{5.6}$$

where a is the activation level, and K is the spring constant, which includes both the stiffness and the physiological cross sectional area. The total stretching potential energy of a strand is computed by integrating along the spline, and the force on the i^{th} control point is computed by taking the derivative with respect to $\boldsymbol{q}_i$.

$$\boldsymbol{f}_i(t) = \left(- \sum_{j=i-3}^{i} K \int_0^1 b'(s) \boldsymbol{x}'(s,t) \|\boldsymbol{x}'(s,0)\| ds \right) a. \tag{5.7}$$

The last term in the integral, $\|\boldsymbol{x}'(s,0)\|$, comes from the change of variables from arc-length parameter to the spline parameter, s. The resulting force is linear in a, and the quantity enclosed in the brackets forms an element of the activation transport matrix. The integral can be evaluated accurately using a numerical integration scheme. In our implementation, we use the midpoint rule with 20 partitions.

We use Rayleigh damping given by

$$\boldsymbol{f}_d = \left(\alpha M + \beta \frac{\partial \boldsymbol{f}^T}{\partial \boldsymbol{q}} \right) \dot{\boldsymbol{q}}, \tag{5.8}$$

where $\boldsymbol{f}$ is the cumulative force (excluding $\boldsymbol{f}_d$) from Equation 5.5 and α and β are positive damping parameters.

5.2 Constraints

Constraints are required for musculotendon origins/insertions and for tendon routing. Although wrapping surfaces implemented in biomechanical simulators [24, 31] are effective for kinematic constraints, they do not work for dynamic constraints, and are also limited to simplified geometries, such as spheres and cylinders.

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

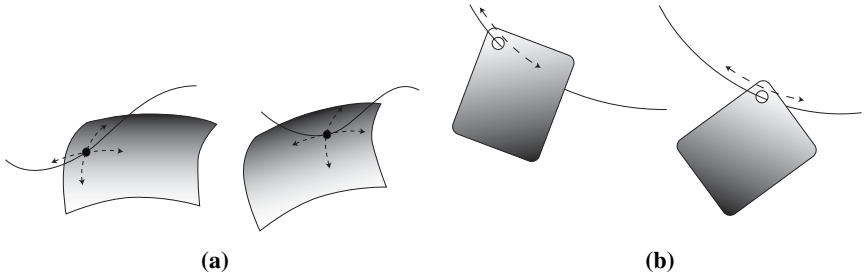


Figure 5.2: (a) Surface constraint and (b) sliding constraint. With a surface constraint, the constraint point moves on the rigid body surface, whereas with a sliding constraint, the constraint point moves along the strand.

Although our simulator is a general multi-body simulator with deformable strands, there are two key assumptions in our application that simplifies the constraint formulation. Since tendons and muscles stay in contact with surrounding tissue and do not come apart, *we deal only with equality constraints*; inequality constraints, which are more difficult to solve numerically, do not need to be modelled. In addition, because we know where the contact regions are going to be, *no general-purpose collision 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. First, each

rigid body is wrapped with a cubic tensor-product surface, and the contact point is tracked on the surface at each time step using the Newton-Raphson method.

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

$$\frac{d\,g(\mathsf{q})}{dt} = \frac{\partial\,g(\mathsf{q})}{\partial \mathsf{q}}\dot{\mathsf{q}} = 0. \tag{5.9}$$

Denoting the gradient of g by the constraint matrix G, we obtain the constraint equation $G\dot{\mathsf{q}} = 0$.

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

$$G\dot{\mathsf{q}} = -\mu g, \tag{5.10}$$

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

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

$$\dot{\boldsymbol{g}} = \dot{\boldsymbol{x}} - \dot{\boldsymbol{x}}_0, \tag{5.11}$$

where both $\dot{\boldsymbol{x}}$ and $\dot{\boldsymbol{x}}_0$ are linear with respect to the rigid body and spline DoFs of the system, as given by Equation 3.5 and Equation 5.4.

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 $\boldsymbol{x}$ denote the 3D position of the strand point to constrain and $\boldsymbol{x}_0$ and $\boldsymbol{n}$ its corresponding contact point and normal on the rigid body. Equating the relative velocities along the normal gives

$$\dot{g} = \boldsymbol{n}^T(\dot{\boldsymbol{x}} - \dot{\boldsymbol{x}}_0). \tag{5.12}$$

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 (Figure 5.2a).

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

tors, and apply the constraint with respect to both of these vectors.

$$\begin{aligned} \dot{g}_1 &= \boldsymbol{n}_1^T(\dot{\boldsymbol{x}} - \dot{\boldsymbol{x}}_0) \\ \dot{g}_2 &= \boldsymbol{n}_2^T(\dot{\boldsymbol{x}} - \dot{\boldsymbol{x}}_0). \end{aligned} \tag{5.13}$$

Unlike the surface constraint, the constraint point on the rigid body is fixed, whereas it is allowed to slide on the strand (Figure 5.2b).

5.3 Integration

We use the linearly-implicit integrator [5, 19] from Section 4.4, by combining the dynamics equation and the constraint equation (Equations 5.5 & 5.10) to form the KKT equation (3.14), which we solve for the new rigid body and spline control point velocities, $\phi^{(k+1)}$ and $\dot{q}^{(k+1)}$. The positions are then updated trivially using the new velocities.

5.4 Subcutaneous Motion

In this section, we show how the strand simulation framework can be used for automatically generating the motion of tendons and muscles under the skin of a traditionally animated character.

Human bodies are more than skin and bones. When the body moves, tendons and muscles move under the skin in visually important ways that are correlated with both the movement and the internal forces. For example, the appearance of tendons on the back of the hand is related to how the hand is moving and how much force it is exerting. Even though there has been some work on incorporating muscles into animations, it has been very difficult to incorporate biomechanically realistic subcutaneous movements into a traditional animation pipeline. Integration

(a) (b) (c)

Figure 5.3: Stillshots from an animation showing pronation/supination of the forearm, as well as abduction/adduction of the wrist, computed at interactive rates using our controller.

(a) (b) (c)

Figure 5.4: 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".

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.

The hand model used in our examples contains 54 musculotendons and 17

(a) **(b)**

Figure 5.5: (a) Base skin. (b) With our method applied, showing tendons on the back of the hand realistically deforming the skin.

bones. The bone and muscle meshes were purchased from Snoswell Design in Adelaide, and the musculotendon paths were constructed based on standard textbook models in the literature [63]. The skeleton was rigged and animated by an artist, and the resulting animations were imported into our simulator, implemented in Java. The activations for an animation sequence of several seconds were computed within a few minutes. Our method works well with areas where muscles and tendons are near the surface with little subcutaneous fat. We show the effectiveness of the method by simulating the musculotendons of the human hand and forearm.

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 (Chapter 7) was able to produce motion involving all of these ranges of motion (Figure 5.3). Subcutaneous motions are most pronounced for the extensor and abductor tendons of the thumb (Figure 5.4), and the extensor digitorum tendons on the back of the palm (Figure 5.5). Our technique correctly captures the deformation of the skin on the back of the hand during finger extension (Fig-

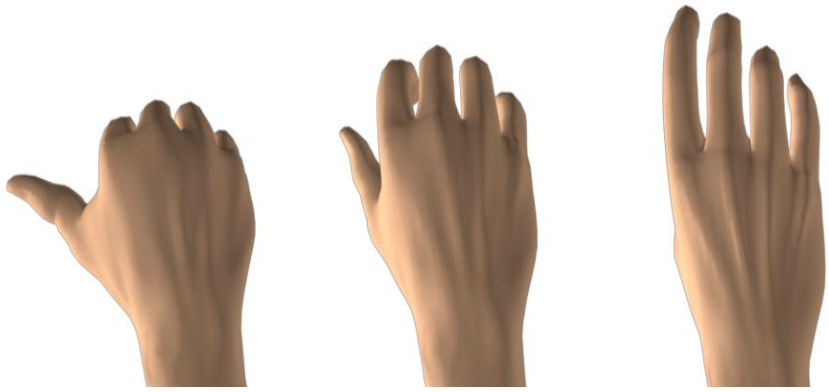


Figure 5.6: Animation showing tendons deforming the skin during extension.

Figure 5.7: We compare the simulated tendons of the thumb to several real thumb photographs.

ure 5.6). We can generate more subtle deformation by varying the skinning parameters (Figure 5.4c). Finally, we show some real thumb photographs as a reference comparison (Figure 5.7).

5.5 Summary

The spline strand is an improvement over the Cosserat strand in that constraints can be applied smoothly on the strand due to the inherent continuity of the spline basis.

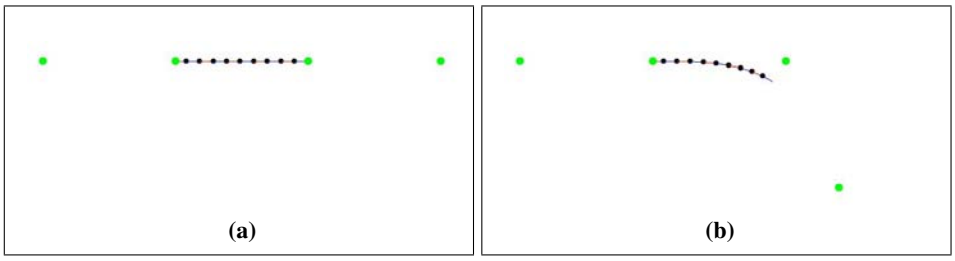


Figure 5.8: Illustration of the coupling between the stretching and bending modes. In (a), the strand is straight, with zero strain. In (b), the same strand is bent slightly. It is not possible to construct the spline with the shape in (b) while maintaining the strain from (a) at all points along the strand.

It works very well as long as the curvature of the strand is not too large, and scales much better for the hand, with its many thin tendons and complex routing constraints. We demonstrated its effectiveness in a graphics application for rendering subcutaneous motion.

The biggest drawback of the spline strand, however, is that it loses degrees of freedom in unintuitive and unexpected ways. This is true in general for all implicitly parameterized models. Because geometry of the strand is constrained to be a cubic spline, there is unavoidable coupling between the stretching and bending modes. If a constraint is applied to make a strand inextensible, it could lose its bending degree of freedom, thereby reducing the strand into a rigid body. There is no way to bend the strand without changing the strain somewhere along the strand. As an example, consider the simple strand with four control points shown in Figure 5.8. It is originally straight, with zero strain. When one of its control points is moved, it bends accordingly, but the strain is no longer zero. In fact, there is no way to deform the strand to this curved configuration in (b) while maintaining the strain from (a). This implies that a change in the bending energy causes a change

in the stretching energy.

In other words, *increasing the stretching stiffness also increases the bending stiffness, and vice-versa*. As another illustration, consider a simple mass-spring strand, consisting of a sequence of particles in series connected by springs. To this, add springs connecting every other particle, which cause the strand to resist bending. In fact, any spring between two non-neighbouring nodes causes the strand to resist bending. In the same vein, the coupling between the stretching and bending energies in a spline strand comes from the fact that the stretching energy depends on four, rather than two, neighbouring nodes along the strand, making any nodal displacement affect both types of energies. In a spline strand, this problem manifests itself even more vividly if the stiff stretching energy is replaced by an inextensible constraint. If a spline strand is constrained to be inextensible, it completely loses its bending degree of freedom, making the strand into a rigid body. This means that we cannot constrain the strand to be inextensible, without over-constraining it to be a rigid body.

This coupling causes another problem: the force propagation direction along a strand can often be unintuitive. For example, when one end of a strand placed in an S-shaped configuration with constraints is pulled, the other end of the strand may not follow the trajectory along the shape of the strand. In fact, depending on where the constraints are placed, the strand may even become rigid. This problem can be overcome by increasing the number of nodes and carefully placing the constraints, but this requires considerable manual tuning.

This leads to the next disadvantage. As more nodes are inserted, corresponding constraints need to be added, increasing the total system size. In order to properly constrain a spline strand to lie on a curve or a surface, every single node along

the strand has to be constrained. On the other hand, if we work on a reduced space defined by a curve or a surface, the constraints will be enforced implicitly, eliminating them from the system equation (i.e., "curve" nodes in Chapter 6).

Since there is no analytical expression for the some of the integrals in the force terms, numerical integration techniques must be used, reducing the advantage of using a higher order primitive (cubic B-splines) in the first place. Although there are highly accurate Gaussian quadrature based techniques for numerical integration, since we need to also compute the gradient of the forces for implicit time integration, only simple techniques can be used, such as mid-point or Simpson's rule, which require many subdivisions to be accurate. The advantage of using a higher-order basis is not as clear, since for a constant or linear basis, no numerical integration is necessary for computing the forces.

For these reasons, rather than using a cubic-spline basis for computing the dynamics in $\mathbb{R}^3$ and constraining the resulting strand to lie on a surface or a curve, we are going to use a linear basis for computing the dynamics of the strand directly in the subspace defined by a surface or a curve. This way, we have exactly the right degrees of freedom, without the unwanted mode coupling, numerical integration of energies and forces, and unintuitive force propagation direction. We call these new strands, "Reduced Strands" (Chapter 6). The key observation here is that for musculotendon simulation, *robust handling of constraints is much more important than higher-order accuracy in computing the shape of the strand*.

Chapter 6

Reduced Strands

Overview: Reduced strands are parameterized by a series of constrained nodes and virtual nodes, which describe the strand with the minimal number of degrees of freedom.

- *Energy modes*: Stretching
- *Contact handling*: Various types of constrained nodes.

The degrees of freedom of a reduced strand are a series of nodes, of which some are free to move in space, some are attached to a rigid body, and some are constrained to move along a kinematic path. We must emphasize here that nodal formulation does *not* imply mass-lumping. This is important when using strands as muscles, since the mass of the muscle accounts for a significant portion of the musculoskeletal system. Later, we will show the difference between our simulator, with continuous mass distribution, and a simulator with constant lumped-mass particles (Section 9.1.1).

Nodal formulation trivially supports stretching energy, but if required, bending

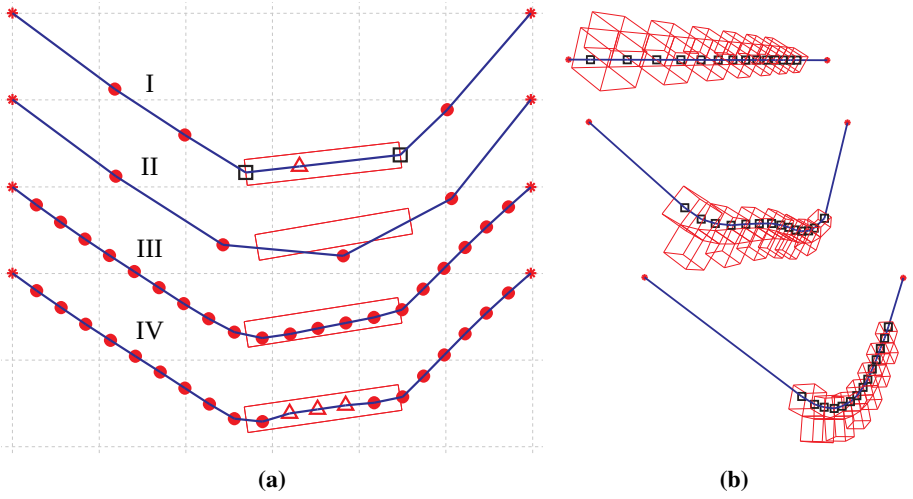


Figure 6.1: (a) Simple hanging strand with a rigid body. (b) Extreme application of virtual nodes. Maximal nodes are indicated by circles, curve nodes by triangles, and rigid virtual nodes by black squares.

forces can be added by measuring the discrete curvature of the strand, and twisting forces can be added with the quasistatic formulation [80] or with auxiliary springs [79].

By itself, a strand parameterized by a series of nodes cannot be constrained robustly, since the segments between nodes are rigid rods and cannot bend. Therefore, we insert "virtual" nodes into a strand to allow it to bend at an arbitrary location along the length of the strand. Figure 6.1a shows the effect of adding virtual nodes to a strand. In (I), the strand has two "virtual" nodes that allow the strand to bend as it comes out of the rigid body, and one "curve" node that is constrained to lie on the line inside the rigid body. In (II), the virtual nodes are removed and replaced with the sliding constraints. The resulting strand loses its bending degrees of freedom, and become over-constrained. Adding additional nodes (III and IV)

does not completely remove the problem. Figure 6.1b shows a strand with no internal nodes passed through a series of virtual nodes attached to rigid bodies with hinge joints. The strand itself has no degrees of freedom, but is properly coupled to the dynamics of the rigid bodies. It may be possible to add virtual nodes to other formulations, but the resulting Lagrangian would be very complicated. Virtual nodes were originally introduced by Servin and Lacoursiere [80] for simulating cables in a virtual reality environment. Our approach has the advantage in that the virtual nodes are fully dynamic, and their velocities and positions are computed along with all the other bodies in the system. Furthermore, in our approach, virtual nodes are accounted for in the computation of the inertia of the strand, rather than being a mass-less particle. We will describe this difference in more detail later.

6.1 Equations of Motion with Maximal Nodes

To derive the equations of motion, we need to express the kinetic and potential energies (T and V) as a function of the generalized coordinates (q). The kinetic energy gives us the generalized mass matrix and the quadratic velocity vector (M and f_q), and the potential energy gives us the generalized force (f).

$$\begin{aligned} \mathsf{M}\ddot{\mathsf{q}} &= \mathsf{f} + \mathsf{f}_q, \\ \mathsf{f}_q &= \left(\frac{\partial T}{\partial \mathsf{q}} - \dot{\mathsf{M}}\dot{\mathsf{q}} \right). \end{aligned} \tag{6.1}$$

In the derivation of these quantities, we take advantage of the fact that with the nodal formulation, a strand is locally supported, meaning these energy quantities

can be computed segment by segment. In other words,

$$T = \sum_{i=0}^{n-1} T_i(\mathsf{q}_i, \mathsf{q}_{i+1}), \quad V = \sum_{i=0}^{n-1} V_i(\mathsf{q}_i, \mathsf{q}_{i+1}), \tag{6.2}$$

where a segment is the portion of the strand between nodes q_i and q_{i+1}. For brevity, we will drop the subscript i from T and V, and assume that we are working with segment energies that will be summed to form the strand energy.

We will start the derivation with the simplest case involving two unconstrained nodes (we call these "maximal" nodes), and build up to the full system by adding different types of nodes and virtual nodes. Once the core framework is in place, adding new types of nodes is easy; as long as each type of node implements a certain set of interface methods, such as `getGeneralizedVelocity()`, `getWorldVelocity()`, or `getJacobian()`, all different types of nodes can be treated the same way in the rest of the code.

6.1.1 Mass Matrix

We start with a segment between two maximal nodes with no virtual node. A segment then is simply the line connecting the two nodes, and can be parameterized by a scalar s. With maximal nodes, the generalized coordinates and velocities *are* the world positions and velocities of the nodes, and so $\mathsf{q} = \boldsymbol{x}$. And for the material point at s along the segment, its world position and velocity are convex combinations of the two maximal nodes, $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$.

$$\begin{aligned} \boldsymbol{x}(s) &= (1-s)\boldsymbol{x}_a + s\boldsymbol{x}_b \\ \dot{\boldsymbol{x}}(s) &= (1-s)\dot{\boldsymbol{x}}_a + s\dot{\boldsymbol{x}}_b. \end{aligned} \tag{6.3}$$

The kinetic energy of a segment is obtained by integrating the velocities of the material points along the length of the segment.

$$T = \frac{1}{2} m \int_0^1 \dot{\boldsymbol{x}}(s)^T \dot{\boldsymbol{x}}(s) ds, \tag{6.4}$$

where m is the mass of the segment, which is calculated once from the density of the strand and assumed to be fixed. By substituting Equation 6.3 into Equation 6.4 and integrating out s, we get

$$T = \frac{1}{2} \begin{pmatrix} \dot{\boldsymbol{x}}_a^T & \dot{\boldsymbol{x}}_b^T \end{pmatrix} \underbrace{\frac{m}{6} \begin{pmatrix} 2I & I \\ I & 2I \end{pmatrix}}_{\boldsymbol{M}} \begin{pmatrix} \dot{\boldsymbol{x}}_a \\ \dot{\boldsymbol{x}}_b \end{pmatrix}. \tag{6.5}$$

The quantity sandwiched between the velocity vector is the mass matrix of the segment, which in this case is not a function of the generalized coordinates, making the quadratic velocity vector zero. As a comparison, if we were to lump the segment mass onto the nodes by assigning half of the mass to $\boldsymbol{x}_a$ and the other half to $\boldsymbol{x}_b$, we would get

$$T = \frac{1}{2} \begin{pmatrix} \dot{\boldsymbol{x}}_a^T & \dot{\boldsymbol{x}}_b^T \end{pmatrix} \frac{m}{2} \begin{pmatrix} I & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} \dot{\boldsymbol{x}}_a \\ \dot{\boldsymbol{x}}_b \end{pmatrix}. \tag{6.6}$$

Even this small difference can manifest itself in simple tests, as shown by Pai [70].

6.1.2 Forces

It is trivial to derive the forces for maximal nodes. We first define the potential and then take the derivative with respect to the generalized coordinates. Here, we

derive the gravitational and stretching forces, but other forces can be added, as long as they are derived from a potential.

Gravity The gravitational potential of a segment is the integral of the contributions from all the material points along the segment.

$$V = -m\boldsymbol{g}^T \int_0^1 \boldsymbol{x}(s)\, ds, \tag{6.7}$$

where $\boldsymbol{g}$ (typically $(0,0,-9.81)^T$) is the gravity vector. Integrating out s and taking the derivative with respect to the degrees of freedom,

$$\mathsf{f} = -\frac{\partial V}{\partial \mathsf{q}} = \frac{m}{2}\begin{pmatrix} I \\ I \end{pmatrix}\boldsymbol{g}, \tag{6.8}$$

where the two rows correspond to the forces acting on $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$. Note that even though the differentiation of the potential with respect to the generalized coordinates gives a generalized force for each of the generalized coordinates, we only show the forces for the two nodes of the segment, since the other forces are all zero.

Stretching If there is no virtual node, the stretching force is a simple spring between the two maximal nodes. Let $l = \|\boldsymbol{x}_b - \boldsymbol{x}_a\|$ be the length of the vector between the two nodes, and $\boldsymbol{d} = (\boldsymbol{x}_b - \boldsymbol{x}_a)/l$ be the corresponding unit vector. Using L to denote the rest length of the segment, the spring potential is

$$V = \frac{1}{2}K_s\,(l-L)^2, \tag{6.9}$$

where K_s is the spring constant. Taking the derivative, the corresponding forces acting on the two nodes are

$$\mathsf{f} = K_s\,(l - L)\begin{pmatrix} \boldsymbol{d} \\ -\boldsymbol{d} \end{pmatrix}. \tag{6.10}$$

If the stiffness of the spring is prohibitively high, as is often the case with tendons, we can replace the spring force with the inextensibility constraint. Letting g denote the positional constraint function, we want to maintain the length of the segment by setting $g = l - L = 0$. Taking the time derivative and using the chain rule, we get the velocity-level constraint, $\dot{g} = G\dot{\mathsf{q}} = 0$.

$$\dot{g} = \begin{pmatrix} -\boldsymbol{d}^T & \boldsymbol{d}^T \end{pmatrix}\begin{pmatrix} \dot{\boldsymbol{x}}_a \\ \dot{\boldsymbol{x}}_b \end{pmatrix} = 0. \tag{6.11}$$

6.2 Equations of Motion with a Virtual Node

A virtual node added to a segment allows it to bend at an arbitrary location along the length of the strand. This is critical for avoiding locking in highly constrained scenarios, and additionally acts as a smooth transition interface between constrained and unconstrained portions of the strand. A virtual node inserts an Eulerian degree of freedom to the segment, and affects the entire formulation of the system, including the inertia, force transmission, and constraint handling. This is in contrast to the virtual nodes found in the work of Servin et al. [81], which only affect the constraints and not inertia.

The term "virtual" comes from the fact that the virtual node does not have its

own degree of freedom. The position and velocity of the virtual node can always be derived from other true degrees of freedom within the system. The simplest example is when a virtual node is defined to be a particle. The degrees of freedom of the virtual node is then tied to the degrees of freedom of the particle. A slightly more complex example is when a virtual node is defined to be a local point on a rigid body. Both the position and the velocity of the virtual node is now tied to the degrees of freedom of the rigid body.

Intuitively, a virtual node acts like a ring passed through a frictionless elastic string. Assuming that the rest length of the string is zero, and ignoring mass effects for now, the geometry of the elastic string is wholly determined by the locations of the two ends and the ring. If the initial configuration of the string is horizontal, with the ring in the middle, then raising the two ends up while holding the ring still makes a "V", while raising the ring along with the ends makes the whole string move up. Holding the two ends fixed and moving the ring up and down produces similar effects. If the ring is moved sideways, however, the material points on the string slides through the ring. Similarly, if the ring is fixed but the two ends are moved horizontally in the same direction, material flows through the ring. The kinetic energy of the string depends on all three nodes: two end nodes and the virtual node.

One disadvantage of the virtual node is that if the virtual node happens to lie right on top of one of the end nodes, we get a singularity. Servin et al. [81] solved this problem by adaptively resampling the strand around the virtual node, using the segment tension and length as the criteria. Currently, we circumvent this problem by simply nudging the offending node away from the virtual node.

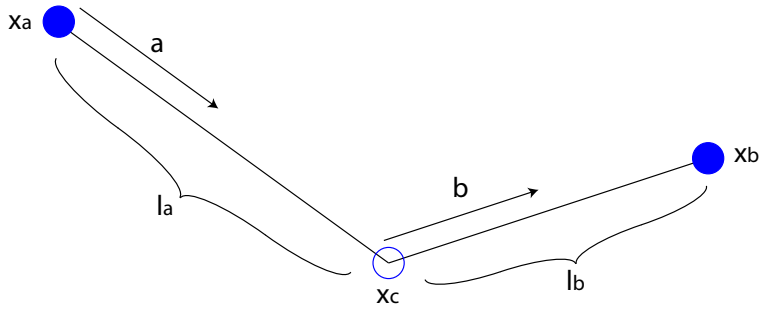


Figure 6.2: Segment with a virtual node, $\boldsymbol{x}_c$. The first subsegment $\boldsymbol{x}_a \to \boldsymbol{x}_c$ is of length l_a, and the second subsegment $\boldsymbol{x}_c \to \boldsymbol{x}_b$ is of length l_b. The corresponding unit vectors are denoted $\boldsymbol{a}$ and $\boldsymbol{b}$.

6.2.1 Mass Matrix

As before, we start by parameterizing the material point on a strand segment with a scalar s. We will continue to call the end nodes $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$, and will call the virtual node $\boldsymbol{x}_c$. As we increase s, the material point, $\boldsymbol{x}(s)$, travels along the subsegment, $\boldsymbol{x}_a \to \boldsymbol{x}_c$, followed by the subsegment $\boldsymbol{x}_c \to \boldsymbol{x}_b$. Let $l_a = \|\boldsymbol{x}_c - \boldsymbol{x}_a\|$ and $l_b = \|\boldsymbol{x}_b - \boldsymbol{x}_c\|$ be the lengths of the two subsegments, and $\alpha = l_a/(l_a + l_b)$ be the relative length of the first subsegment (see Figure 6.2). If $s < \alpha$, $\boldsymbol{x}(s)$ is on the first subsegment, and otherwise, it is on the second subsegment.

if $s < \alpha$ **then**

$s_a = \frac{s}{\alpha}$ // Scalar fraction along $\boldsymbol{x}_a \to \boldsymbol{x}_c$

$\boldsymbol{x}(s_a) = (1 - s_a)\boldsymbol{x}_a + s_a\boldsymbol{x}_c$

else

$s_b = \frac{s-\alpha}{1-\alpha}$ // Scalar fraction along $\boldsymbol{x}_c \to \boldsymbol{x}_b$

$\boldsymbol{x}(s_b) = (1 - s_b)\boldsymbol{x}_c + s_b\boldsymbol{x}_b$

end if

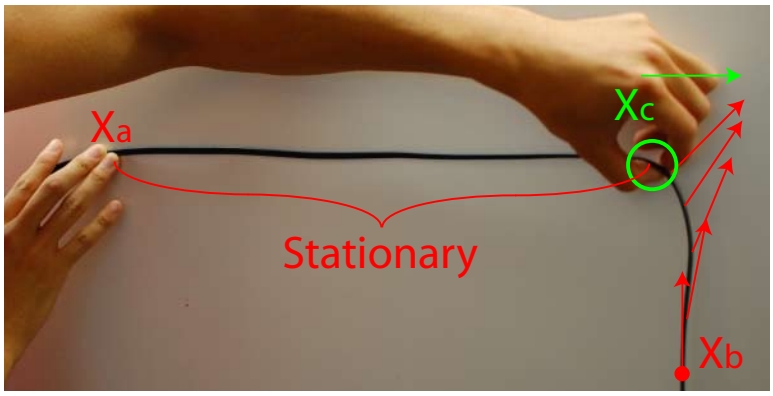


Figure 6.3: Demonstration of the effect of a virtual node. The string is fixed at the left end, goes over the thumb, and down toward the floor.

Taking the derivative with respect to time (See Appendix A), we can compute the material velocity of the segment at the virtual node. There are two discontinuous velocities, one toward the left node, and another toward the right node.

$$\begin{aligned} \dot{\boldsymbol{x}}_{c-} &= \beta \boldsymbol{a}\boldsymbol{a}^T \dot{\boldsymbol{x}}_a + \alpha \boldsymbol{a}\boldsymbol{b}^T \dot{\boldsymbol{x}}_b + (I - \beta \boldsymbol{a}\boldsymbol{a}^T - \alpha \boldsymbol{a}\boldsymbol{b}^T)\dot{\boldsymbol{x}}_c \\ \dot{\boldsymbol{x}}_{c+} &= \beta \boldsymbol{b}\boldsymbol{a}^T \dot{\boldsymbol{x}}_a + \alpha \boldsymbol{b}\boldsymbol{b}^T \dot{\boldsymbol{x}}_b + (I - \beta \boldsymbol{b}\boldsymbol{a}^T - \alpha \boldsymbol{b}\boldsymbol{b}^T)\dot{\boldsymbol{x}}_c, \end{aligned} \tag{6.12}$$

where $\beta = 1 - \alpha$, and $\boldsymbol{a} = \frac{\boldsymbol{d}_a}{\|\boldsymbol{d}_a\|}$ and $\boldsymbol{b} = \frac{\boldsymbol{d}_b}{\|\boldsymbol{d}_b\|}$ are the unit vectors along the two subsegments (Figure 6.2). This discontinuity is not a problem, because we will be integrating the velocities along the two subsegments separately.

A real-world example to illustrate the virtual node is shown in Figure 6.3. The left end of the cable is fixed by the right hand to the wall, while the right end of the cable passes over the left thumb and droops down toward the floor. If the left hand is moved horizontally away from the right hand (green arrow), then the velocity of the material points is zero in the portion of the string between the hands, and non-zero in the other portion, as shown in the figure overlay. This can be

shown numerically if we plug in the values corresponding to this scenario into Equation 6.12. Let $\boldsymbol{x}_a$ be the left node and $\boldsymbol{x}_b$ be the bottom node. Then, without loss of generality,

$$\alpha = \beta = 0.5, \quad \boldsymbol{a} = \begin{pmatrix} 1 & 0 \end{pmatrix}^T, \quad \boldsymbol{b} = \begin{pmatrix} 0 & -1 \end{pmatrix}^T$$
$$\boldsymbol{a}\boldsymbol{a}^T = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, \quad \boldsymbol{a}\boldsymbol{b}^T = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}, \quad \boldsymbol{b}\boldsymbol{a}^T = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}, \quad \boldsymbol{b}\boldsymbol{b}^T = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \tag{6.13}$$

The left node is fixed, while the right node moves toward the virtual node with the same magnitude as the horizontal velocity of the virtual node.

$$\dot{\boldsymbol{x}}_a = \begin{pmatrix} 0 & 0 \end{pmatrix}^T, \quad \dot{\boldsymbol{x}}_b = \begin{pmatrix} 0 & 1 \end{pmatrix}^T, \quad \dot{\boldsymbol{x}}_c = \begin{pmatrix} 1 & 0 \end{pmatrix}^T. \tag{6.14}$$

Plugging these values into Equation 6.12, we get

$$\dot{\boldsymbol{x}}_{c-} = \begin{pmatrix} 0 & 0 \end{pmatrix}^T, \quad \dot{\boldsymbol{x}}_{c+} = \begin{pmatrix} 1 & 1 \end{pmatrix}^T, \tag{6.15}$$

which is shown in red in 6.3.

For further intuition on these two velocity quantities, it may be helpful to pause and think about what the instantaneous velocities are at the virtual node. Let us first assume that the virtual node stays fixed, $\dot{\boldsymbol{x}}_c = 0$. As material flows through the virtual node, the direction of the material velocity vectors are along the two subsegment vectors. Furthermore, the magnitude of the material velocity vectors

are equal, since the amount of material going in must equal the amount going out.

$$\dot{\boldsymbol{x}}_{c-} = c\boldsymbol{a}, \quad \dot{\boldsymbol{x}}_{c+} = c\boldsymbol{b}. \tag{6.16}$$

The shared magnitude, c, is a linear combination of the projections of the nodal velocities onto the subsegment vectors. For example, if $\dot{\boldsymbol{x}}_a$ is perpendicular to $\boldsymbol{a}$, then $\dot{\boldsymbol{x}}_a$ does not contribute to c, but if $\dot{\boldsymbol{x}}_a$ is parallel to $\boldsymbol{a}$, then it fully contributes to c. The contributions are further weighted by the relative distances of the virtual node to $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$. If the virtual node is close to $\boldsymbol{x}_a$, then $\dot{\boldsymbol{x}}_a$ should be weighed more, and if it is close to $\boldsymbol{x}_b$, then $\dot{\boldsymbol{x}}_b$ should we weighed more. Using the convex weights α and β from before,

$$c = \beta \boldsymbol{a}^T \dot{\boldsymbol{x}}_a + \alpha \boldsymbol{b}^T \dot{\boldsymbol{x}}_b, \tag{6.17}$$

which, if substituted into Equation 6.16, we arrive at Equation 6.12, with $\dot{\boldsymbol{x}}_c = 0$.

If we remove the assumption that the virtual node stays fixed, then the two discontinuous material velocities are also affected by the velocity of the virtual node itself.

$$\dot{\boldsymbol{x}}_{c-} = \dot{\boldsymbol{x}}_c + c\boldsymbol{a}, \quad \dot{\boldsymbol{x}}_{c+} = \dot{\boldsymbol{x}}_c + c\boldsymbol{b}. \tag{6.18}$$

This time, the magnitude uses the projection of the *relative* velocities of the end nodes and the virtual node.

$$c = \beta \boldsymbol{a}^T (\dot{\boldsymbol{x}}_a - \dot{\boldsymbol{x}}_c) + \alpha \boldsymbol{b}^T (\dot{\boldsymbol{x}}_b - \dot{\boldsymbol{x}}_c). \tag{6.19}$$

If we substitute this new c into Equation 6.18, we get the original Equation 6.12.

Armed with the two material velocities at $\boldsymbol{x}_c$, we can express the material velocity of any point along the two subsegments.

$$\begin{aligned} \dot{\boldsymbol{x}}_-(s_a) &= (1-s_a)\dot{\boldsymbol{x}}_a + s_a\dot{\boldsymbol{x}}_{c-} \\ \dot{\boldsymbol{x}}_+(s_b) &= (1-s_b)\dot{\boldsymbol{x}}_{c+} + s_b\dot{\boldsymbol{x}}_b. \end{aligned} \tag{6.20}$$

The kinetic energy of the segment is obtained by integrating the material-point kinetic energies along the two subsegments.

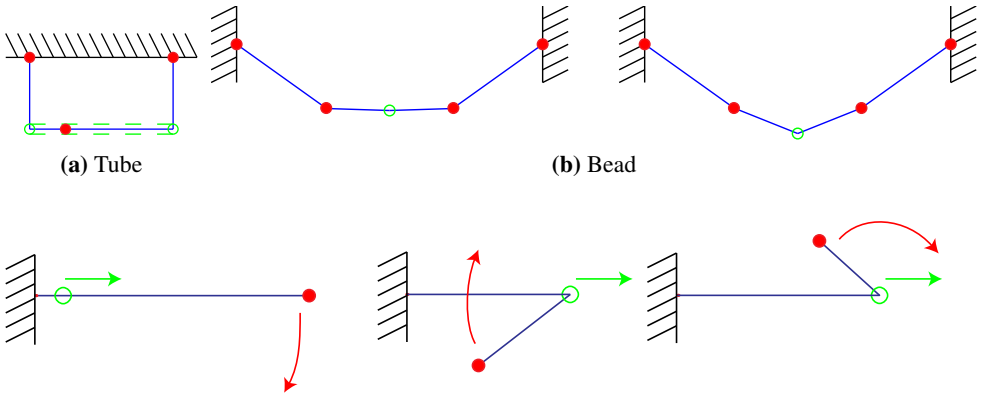
$$\begin{aligned} T_a &= \frac{1}{2}\alpha m \int_0^1 \dot{\boldsymbol{x}}_-(s_a)^T \dot{\boldsymbol{x}}_-(s_a)\, ds_a \\ T_b &= \frac{1}{2}\beta m \int_0^1 \dot{\boldsymbol{x}}_+(s_b)^T \dot{\boldsymbol{x}}_+(s_b)\, ds_b \end{aligned} \tag{6.21}$$

Here, the mass of the segment is appropriately distributed using the ratios α and β. As before, we integrate out s_a and s_b, and rearrange to get the symmetric positive definite mass matrix of the segment (only upper portion shown).

$$\mathsf{M} = \frac{m}{6}\begin{pmatrix} 2(\alpha I + \beta \boldsymbol{a}\boldsymbol{a}^T) & \boldsymbol{a}\boldsymbol{b}^T & \alpha(I+2C_{a-}^T)C_{c-} + 2\beta C_{a+}^T C_{c+} \\ \cdot & 2(\beta I + \alpha \boldsymbol{b}\boldsymbol{b}^T) & 2\alpha C_{b-}^T C_{c-}^T + \beta(I+2C_{b+}^T)C_{c+} \\ \cdot & \cdot & 2(\alpha C_{c-}^T C_{c-} + \beta C_{c+}^T C_{c+}) \end{pmatrix}, \tag{6.22}$$

where

$$\begin{aligned} C_{a-} &= \beta \boldsymbol{a}\boldsymbol{a}^T, \quad C_{b-} = \alpha \boldsymbol{a}\boldsymbol{b}^T, \quad C_{c-} = I - C_{a-} - C_{b-} \\ C_{a+} &= \beta \boldsymbol{b}\boldsymbol{a}^T, \quad C_{b+} = \alpha \boldsymbol{b}\boldsymbol{b}^T, \quad C_{c+} = I - C_{a+} - C_{b+}. \end{aligned} \tag{6.23}$$



(a) Tube **(b)** Bead

(c) Flail

Figure 6.4: Three simple examples for showing the difference between lumped and non-lumped mass matrices.

For the derivation of the quadratic velocity vector, see Appendix B

Three simple test cases show the difference in behaviour between lumped and non-lumped mass matrices. The hanging strand in Figure 6.4a goes through a tube, and has a single off-centred internal curve node. If the mass is lumped onto the node, then the tube will start to tilt due to the non-symmetric forces acting on the node. If the mass is continuously distributed, however, the tube and the strand stays in equilibrium.

The strand in Figure 6.4b, with two fixed nodes, two free nodes, and a *light* virtual node, shows different static behaviours depending on whether the mass is lumped or not. In both cases, the strand has a bending degree of freedom at the virtual node. However, if the mass is lumped, the static configuration would resemble the left figure, because all of the mass of the strand is lumped onto the material (red) nodes. On the other hand, if the mass is distributed equally along the strand, the static configuration would properly resemble the catenary shape shown in the

right figure.

In Figure 6.4c, a strand with its left node fixed and right node free is shown. The virtual node is scripted to move at a constant speed toward the free node. (This is the same scenario as in Figure 6.3.) If the mass is lumped, then the mass of the right node stays constant, making this into a "flail-like" system, and causes the right node to swing faster as the radius around the virtual node gets smaller. If, on the other hand, the mass is not lumped, then the mass of the right node decreases since the length of the right subsegment decreases, and so the right node does not swing uncontrollably as the virtual node moves to the right.

6.2.2 Forces

Gravity

We must divide the potential energy into two subsegments. $V = V_a + V_b$.

$$\begin{aligned} V_a &= \alpha m \boldsymbol{g}^T \int_0^1 \boldsymbol{x}(s_a)\, ds_a \\ V_b &= \beta m \boldsymbol{g}^T \int_0^1 \boldsymbol{x}(s_b)\, ds_b, \end{aligned} \tag{6.24}$$

where $\boldsymbol{x}_-(s_a)$ and $\boldsymbol{x}_+(s_b)$ are positional equivalent of Equation 6.20. If we integrate out s_a and s_b, and take the derivative with respect to $\boldsymbol{x}_a$, $\boldsymbol{x}_b$, and $\boldsymbol{x}_c$, we get the generalized gravity force acting on the two end nodes and the virtual node. The local support property of the segment still holds, and thus the generalized force is zero for all other nodes. After some rearranging, the resulting force looks similar

to the case with no virtual node (Equation 6.8), with some additional terms.

$$\mathsf{f} = -\frac{\partial V}{\partial \mathsf{q}} = \frac{m}{2} \begin{pmatrix} \alpha I + \frac{1}{l}\beta \boldsymbol{a}(\boldsymbol{x}_b - \boldsymbol{x}_a)^T \\ \beta I + \frac{1}{l}\alpha \boldsymbol{b}(\boldsymbol{x}_b - \boldsymbol{x}_a)^T \\ I - \frac{1}{l}(\beta \boldsymbol{a} + \alpha \boldsymbol{b})(\boldsymbol{x}_b - \boldsymbol{x}_a)^T \end{pmatrix} \boldsymbol{g}, \tag{6.25}$$

where $l = l_a + l_b$ is the current total length of the segment. The three block rows correspond to the generalized forces acting on the two end nodes, $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$, and the virtual node $\boldsymbol{x}_c$, respectively.

Stretching

Since the virtual node applies no friction, we make use of the assumption that the strain values on the two subsegments are equal. Servin et al. [81] applies a pseudo-friction at the virtual node by limiting the positional update of the virtual node to be a set fraction of the computed value. If we want to add proper friction, however, we would have to add a new degree of freedom at the virtual node, the material flux, that keeps track of the amount of material flowing from one side to the other.

The spring potential of a segment is the same with or without a virtual node.

$$V = \frac{1}{2} K_s (l - L)^2. \tag{6.26}$$

But with a virtual node, the segment length is the sum of the lengths of the two subsegments, $l = l_a + l_b$. Taking the derivative with respect to the degrees of freedom

yields

$$\mathsf{f} = K_s(l-L)\begin{pmatrix} \boldsymbol{a} \\ -\boldsymbol{b} \\ \boldsymbol{b}-\boldsymbol{a} \end{pmatrix}. \tag{6.27}$$

If the segment is stretched relative to the rest length, the generalized spring force pulls both $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$ toward $\boldsymbol{x}_c$, and vice-versa. Again, this force bares resemblance to the stretching force with no virtual node (Equation 6.10).

Sometimes it is advantageous to replace a stiff spring with an inextensibility constraint. Using $g = l - L = 0$ to denote the positional constraint function, the velocity-level constraint is

$$\dot{g} = \begin{pmatrix} -\boldsymbol{a}^T & \boldsymbol{b}^T & (\boldsymbol{a}-\boldsymbol{b})^T \end{pmatrix} \begin{pmatrix} \dot{\boldsymbol{x}}_a \\ \dot{\boldsymbol{x}}_b \\ \dot{\boldsymbol{x}}_c \end{pmatrix} = 0. \tag{6.28}$$

6.3 Adding Other Node Types

So far, our exposition has been limited to maximal nodes—all strand nodes, as well as the virtual node, were assumed to be unconstrained and free to move in space. One of the strengths of our system is that other types of nodes can be added, without changing the core framework. To do so, we require that each node type implements the following interface methods.

- `getPosW()` (3×1) Gets the world position vector.
- `getVelG()` $(n \times 1)$ Gets the generalized velocity.

- `intPosG()` (void) Integrates the generalized position using the current generalized velocity.
- `getJac()` $(3 \times n)$ Gets the material Jacobian matrix, which gives the material point velocity from a generalized velocity ($\dot{\boldsymbol{x}} \equiv \mathsf{J}\dot{\mathsf{q}}$).
- `getJacT()` (Optional) $(3 \times n)$ Gets the time derivative of the material Jacobian matrix, $\dot{\mathsf{J}}$.
- `getJacQ()` (Optional) $(3 \times n)$ Gets the product of the derivative of the material Jacobian matrix and the generalized velocity vector, $\frac{\partial \mathsf{J}}{\partial \mathsf{q}}\dot{\mathsf{q}}$.

The last two methods are required for the computation of the quadratic velocity vector. If the mass of the strand is small relative to the rest of the bodies in the scene, these methods can be left unimplemented to turn off the quadratic velocity vector.

A maximal node, whose generalized position and velocity are $\boldsymbol{x}$ and $\dot{\boldsymbol{x}}$, trivially implements these interface methods.

- `getPosW()` The generalized position, $\boldsymbol{x}$, *is* the world position.
- `getVelG()` The generalized velocity, $\dot{\boldsymbol{x}}$, *is* the world velocity.
- `intPosG()` This depends on the integrator used, but in our case (see Section 4.4) it is $\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + h\dot{\boldsymbol{x}}^{(k+1)}$.
- `getJac()` Because the generalized velocity is the world velocity, $\mathsf{J} = I$.
- `getJacT()` $\dot{\mathsf{J}} = 0$.
- `getJacQ()` $\frac{\partial \mathsf{J}}{\partial \boldsymbol{x}}\dot{\boldsymbol{x}} = 0$.

Once we have the quantities returned from the interface methods for all the nodes in the segment, we can update the generalized mass matrix, generalized forces, and constraints by modifying the original matrices with a simple multiplication by the material Jacobian matrix, J, returned from the `getJac()` method. Since the material Jacobian transforms the node's generalized velocity into point velocity, we can rewrite the kinetic energy as

$$T = \frac{1}{2}\begin{pmatrix} \dot{\mathsf{q}}_a^T \mathsf{J}_a^T & \dot{\mathsf{q}}_b^T \mathsf{J}_b^T & \dot{\mathsf{q}}_c^T \mathsf{J}_c^T \end{pmatrix} \begin{pmatrix} \mathsf{M}_{aa} & \mathsf{M}_{ab} & \mathsf{M}_{ac} \\ \mathsf{M}_{ba} & \mathsf{M}_{bb} & \mathsf{M}_{bc} \\ \mathsf{M}_{ca} & \mathsf{M}_{cb} & \mathsf{M}_{cc} \end{pmatrix} \begin{pmatrix} \mathsf{J}_a \dot{\mathsf{q}}_a \\ \mathsf{J}_b \dot{\mathsf{q}}_b \\ \mathsf{J}_c \dot{\mathsf{q}}_c \end{pmatrix}, \tag{6.29}$$

where a, b, and c refer to the two end nodes and the virtual node of a segment. (For a segment with no virtual node, we just take the upper 2×2 portion of the matrix.) By multiplying the Jacobian matrices into the mass matrix, we get the generalized mass matrix for the segment.

$$\mathsf{M} = \begin{pmatrix} \mathsf{J}_a^T \mathsf{M}_{aa} \mathsf{J}_a & \mathsf{J}_a^T \mathsf{M}_{ab} \mathsf{J}_b & \mathsf{J}_a^T \mathsf{M}_{ac} \mathsf{J}_c \\ \mathsf{J}_b^T \mathsf{M}_{ba} \mathsf{J}_a & \mathsf{J}_b^T \mathsf{M}_{bb} \mathsf{J}_b & \mathsf{J}_b^T \mathsf{M}_{bc} \mathsf{J}_c \\ \mathsf{J}_c^T \mathsf{M}_{ca} \mathsf{J}_a & \mathsf{J}_c^T \mathsf{M}_{cb} \mathsf{J}_b & \mathsf{J}_c^T \mathsf{M}_{cc} \mathsf{J}_c \end{pmatrix}. \tag{6.30}$$

If all three nodes are maximal, then the Jacobian matrix is the identity, giving us back the original mass matrix.

Since transpose of the Jacobian matrix transforms a point force into a generalized force,

$$\mathsf{f} = \begin{pmatrix} \mathsf{J}_a^T \boldsymbol{f}_a \\ \mathsf{J}_b^T \boldsymbol{f}_b \\ \mathsf{J}_c^T \boldsymbol{f}_c \end{pmatrix}, \tag{6.31}$$

for all the forces derived for maximal nodes.

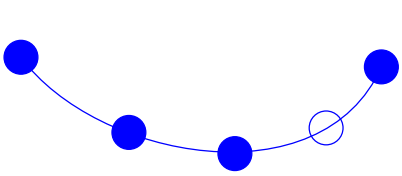
Finally, the constraint matrix is modified similarly with the Jacobian matrix.

$$\dot{g} = \begin{pmatrix} G_a \mathsf{J}_a & G_b \mathsf{J}_b & G_c \mathsf{J}_c \end{pmatrix} \begin{pmatrix} \dot{\mathsf{q}}_a \\ \dot{\mathsf{q}}_b \\ \dot{\mathsf{q}}_c \end{pmatrix}. \tag{6.32}$$

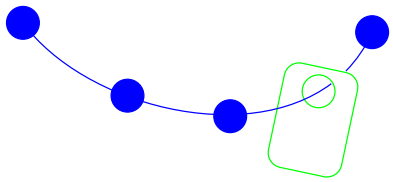
The quadratic velocity vectors are similarly modified with the Jacobian and its derivatives (see Appendix B).

We have implemented four types of nodes in the system so far:

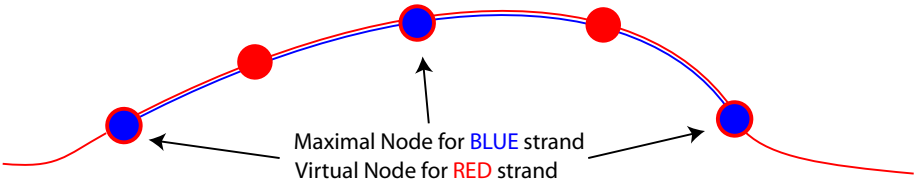
- *Maximal* nodes $\in \mathbb{R}^3$, which are the normal nodes that are allowed to move about freely in space.
- *Rigid* nodes, which have no degree of freedom, and are kinematically attached to rigid bodies. These nodes are used to model origins and insertions.
- *Curve* nodes $\in \mathbb{R}$ for nodes constrained to lie on a 3D space curve.
- *Blend* nodes, which is a convex combination of multiple nodes, analogous to the way vertices are blended in linear blend skinning. The parent nodes can be a combination of different types of nodes listed above.



(a) A maximal virtual node (hollow blue circle) on a strand.



(b) A rigid virtual node on a strand (hollow green circle).



(c) A strand (red) passed through another strand (blue). To make sure that the red strand does not deviate from the blue strand, each blue node is added to the red strand as virtual nodes.

Figure 6.5: Various applications of the virtual node

Furthermore, when constructing a strand from a set of nodes, any node can be defined as virtual or non-virtual. For example, a strand with a single maximal virtual node models a bead on a string (Figure 6.5a), and a strand with a single rigid virtual node models a swinging object on a string (Figure 6.5b). As a more complicated example, a node can be a virtual node for one strand and a non-virtual node for another strand, which is useful when routing a strand inside another strand (Figure 6.5c).

6.3.1 Rigid Node

Next, let us look at a "rigid" node, a node that is fixed kinematically to a rigid body. The node is defined in the local coordinate frame of the parent rigid body, and the world position and velocity of the node is completely determined by the

rigid body's $SE(3)$ configuration, E, and $se(3)$ twist, ϕ.

- `getPosW()` Given the current configuration of the rigid body, we want to return the world position of the node. This is just a simple rigid transformation from local to world space.

$$\boldsymbol{x} = \mathsf{E}\boldsymbol{r}. \tag{6.33}$$

- `getVelG()` The generalized velocity of a rigid node is the parent rigid body's twist, ϕ.

- `intPosG()` In this method, we want to update the generalized position of this node, but since a rigid node has zero degrees of freedom, this method does nothing. Note that the underlying rigid body does not need to be integrated in this method, since its own `intPosG()` will be called separately. This makes sense, because a rigid body's twist should be integrated just once per time step, even if the rigid body contains multiple rigid nodes.

- `getJac()` Given the current configuration and twist of the rigid body, we want to return the material Jacobian of the node. Here we assume that the rigid body's twist, ϕ, is stored as a local twist (i.e., the relative twist of the rigid body with respect to the world, expressed in the body's local frame). Since the world velocity of a local point, $\boldsymbol{r}$ is $\dot{\boldsymbol{x}} = \Theta\left([\boldsymbol{r}]^T I\right)\phi$ (Equation 3.5), the Jacobian matrix is

$$\mathsf{J} = \Theta\left([\boldsymbol{r}]^T I\right). \tag{6.34}$$

- `getJacT()` The time derivative of J is

$$\dot{\mathsf{J}} = \Theta[\boldsymbol{\omega}] \left([\boldsymbol{r}]^T I \right), \tag{6.35}$$

  where $[\boldsymbol{\omega}]$ is the cross product matrix of the angular velocity.

- `getJacQ()` The product of the material derivative and the generalized velocity is trickier to derive (See Appendix B).

$$\frac{\partial \mathsf{J}}{\partial \mathsf{q}} \dot{\mathsf{q}} = \begin{pmatrix} \Theta[\boldsymbol{e}_1]\dot{\boldsymbol{x}} & \Theta[\boldsymbol{e}_2]\dot{\boldsymbol{x}} & \Theta[\boldsymbol{e}_3]\dot{\boldsymbol{x}} & 0 & 0 & 0 \end{pmatrix}, \tag{6.36}$$

  where $\boldsymbol{e}_1 = (1,0,0)^T$ is the 1st basis vector, etc., and $\dot{\boldsymbol{x}}$ is the world velocity of the rigid node.

6.3.2 Curve Node

A curve node is a node constrained to lie on a 3D space curve, and is parameterized by a scalar u. The constraining curve must be explicit, in that we need an analytical expression for the world position of the curve node as a function of the parameter u. In our current implementation, we have support for piece-wise linear curves and cubic B-spline curves, but other curve types may be useful, such as circles and ellipses. In all cases, the curve must be parameterized by the parameter u and a set of control points. These control points are themselves nodes—maximal, rigid, or even other curve nodes (see Figure 6.6). The hierarchy of node dependence can be of any depth, as long as there are no cycles.

Both piece-wise linear and cubic B-spline curves have local support. The position of a curve node is a function of two control points for a linear curve, and

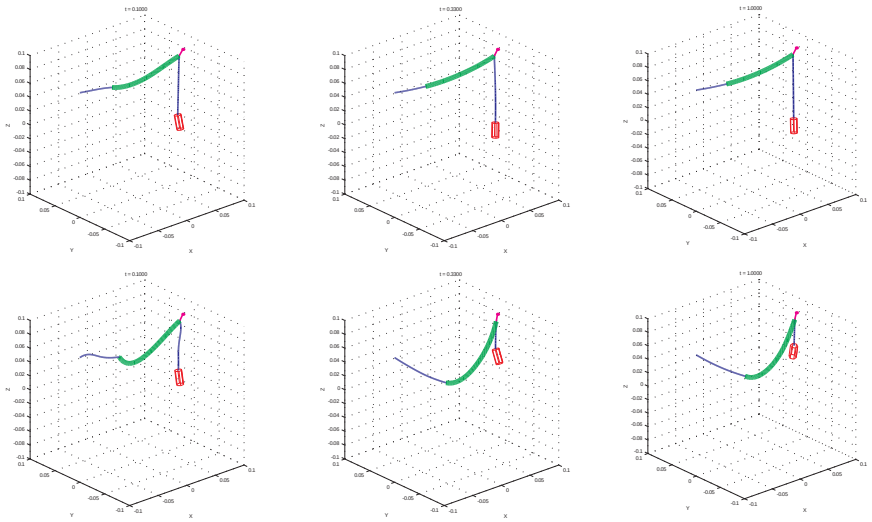


Figure 6.6: Strand inside another strand. Compared to the longer blue string, the green tube is lighter in the top row, and heavier in the bottom row.

four control points for a cubic B-spline curve. In both cases, we parameterize the curve so that the integer part of u is the index of the first supporting control point, and the fractional part is the interpolation parameter along the portion of the curve supported by the control points. For a linear curve, for example, $u = 1.5$ gives the point midway through the second and third control points, $(1-0.5)\boldsymbol{x}_2 + 0.5\boldsymbol{x}_3$, and for a B-spline curve, it gives the point defined by the four control points $\sum_{k=1}^{4}(b_k(0.5)\boldsymbol{x}_{k+1})$, where $b_i(s)$ is the i^{th} cubic B-spline blending function.

One typical use for the curve node is for passing a strand through a tube fixed to a rigid body. This can be used to model a tendon passing through a rigid sheath, such as the flexor tendon sheath in the finger (see Figure 6.7). To do this, the control points of the curve are constructed from a series of rigid nodes. (These rigid nodes can be parented to different rigid bodies, but if we want the tube to be rigid, they must be attached to a single rigid body.) Once the curve is constructed,

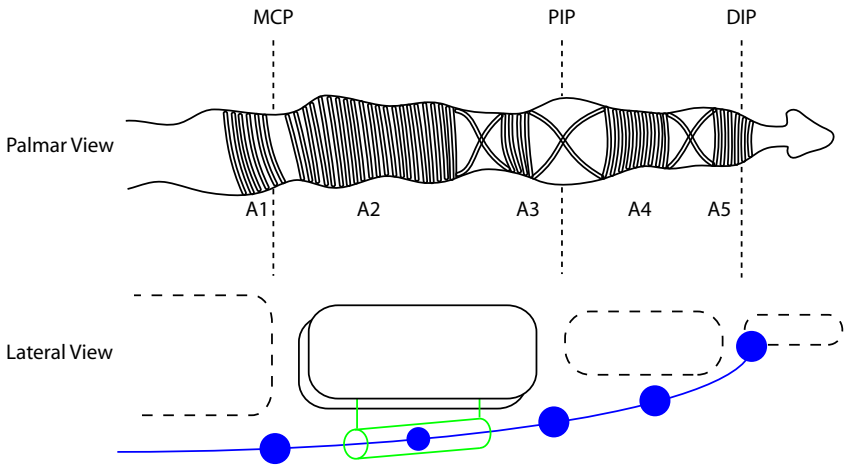


Figure 6.7: Tendon sheath for holding the flexor in place. The flexor tendon shown in blue has a curve node through the A2 sheath shown in green.

a strand can be routed through the curve by creating a series of curve nodes inside the curve.

Except for `intPosG()`, the interface methods of the curve node are based on the underlying curve. For instance, to get the world position of a curve node, the node just calls the `getPosW()` method on the curve with the parameter u. (I.e., `node.getPosW()` calls `curve.getPosW(u)`.) Therefore, each curve (linear, cubic b-spline, etc.) must in turn implement the corresponding interface methods. At the beginning of each method, the index of the first supporting control point, i, and the interpolation parameter, s, are computed from the current value of u by separating u into integer and fractional parts.

- `getPosW(`u`)` The world position of the node is, for piece-wise linear and

cubic B-spline curves,

$$\begin{aligned} \boldsymbol{x}(u) &= (1-s)\boldsymbol{x}_i + s\boldsymbol{x}_{i+1} \\ \boldsymbol{x}(u) &= \sum_{k=1}^{4} b_k(s)\,\boldsymbol{x}_{k+i}. \end{aligned} \tag{6.37}$$

Because the control points are themselves nodes, for both cases, the world positions of the control points are computed by calling the control point's `getPosW()` method.

- `getVelG()` The generalized velocity of a curve node is the concatenation of the generalized velocities of the supporting control points and the speed of the interpolation parameter, $\dot{u}$.

- `getJac(`u`)`, `getJacT(`$u,\dot{u}$`)`, `getJacQ(`$u,\dot{u}$`)` The material Jacobian and the derivatives are weighted combinations of the control points' corresponding matrices, as in Equation 6.37.

The curve node's `intPosG()` method updates the node's curve parameter, u, using the parameter speed, $\dot{u}$. If the curve is parameterized consistently, (i.e., $\|\boldsymbol{x}'(u)\| \equiv \|d\boldsymbol{x}/du\|$ is constant, as in an arc-length parameterized curve), we can integrate the speed as usual: $u^{(k+1)} = u^{(k)} + h\dot{u}^{(k+1)}$. However, in most cases, the parameterization isn't consistent. A typical piece-wise linear curve would consist of line segments of different lengths, and so $\|\boldsymbol{x}'(u)\| = \|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\|$ varies from segment to segment (see Figure 6.9), and similarly, the length of the tangent vector of a cubic B-spline curve, $\|\boldsymbol{x}'(u)\| = \|\sum_{i=1}^{4} b_i'(s)\boldsymbol{x}_{k+i}\|$, is, in general, never constant for a non-straight curve. To compute $u^{(k+1)}$, what we need to do instead is to `traverse` along the curve from $u^{(k)}$ for the distance $h\,\dot{u}^{(k+1)}$. We therefore add

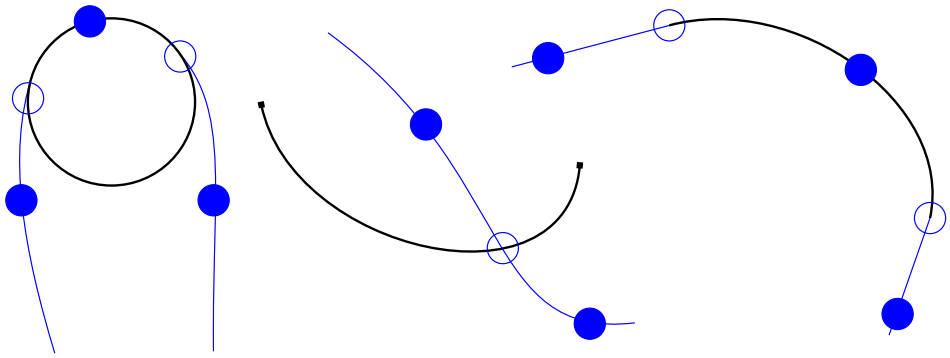


Figure 6.8: Wrapped, capped, and plain curves. A wrapped curve can be used for a pulley, and a plain curve can be used for a tube. A capped curve can be used to constrain a virtual node to stay within a portion of a curve.

one more interface method to curves, `traverse`$(u_0, dist)$, that returns the curve parameter u on the curve that is the result of traversing along the curve starting at u_0 for distance $dist$.

Curve End Constraint

A curve can be wrapped, capped, or plain (neither wrapped nor capped). A wrapped curve is useful for modelling pulley paths, so that the pulley curve forms a closed path. In a capped curve, the nodes inside the curve are confined to stay inside the curve. A plain curve is used most often for modelling simple tendon sheaths, where the strand passes through the sheath and comes out of the other end (see Figure 6.8).

The cap constraint can be implemented in a similar fashion to joint limits. If a node is detected to be at the end of a capped curve, an equality constraint, (for example, $g(u) = u = 0$ for capping a node to the beginning of a curve), is added to the system to keep the node fixed at the end. The constraint remains in the system

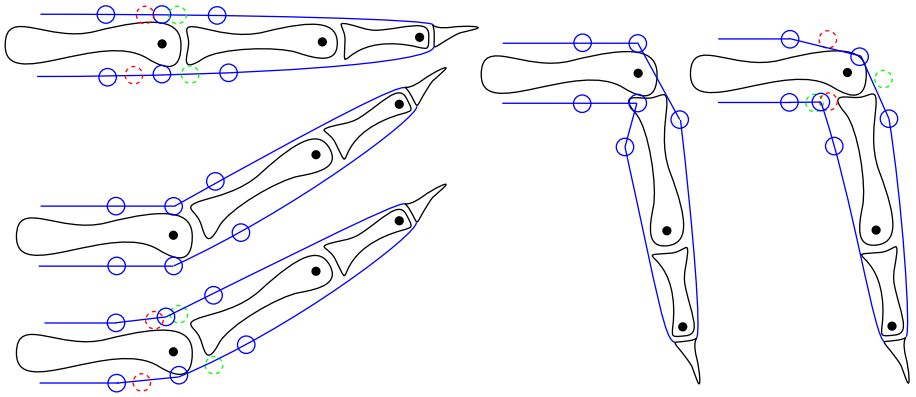


Figure 6.9: Blend node at the metacarpal phalangeal joint (MCP), is set to be half-way between two rigid nodes. The red rigid nodes are parented to the metacarpal bone, and the green rigid nodes are parented to the proximal bone.

until the sign of the corresponding Lagrange multiplier becomes negative, at which point the constraint is removed. If the minor amount of "sticking" becomes an issue, we can replace the bilateral constraint with a proper unilateral constraint, at an added cost in computation time.

6.3.3 Blend Node

Finally, a node can be constructed from a convex combination of other nodes. This is useful especially when a strand has to be routed across a joint. In Figure 6.9, a virtual blend node is constructed from two rigid nodes. As the joint bends, the virtual node's position is a linear blend of the two rigid node positions, allowing the cable to be routed across the joint without going into the rigid body. Without the blend node, the tendon sinks into the bone when the joint angle deviates too much from the rest angle.

The interface methods for the blend node are simply the weighted combination

of the parent interface methods.

6.4 Results

We implemented our simulator in MATLAB ™ and tested its scalability and robustness with mechanical examples. Because of the reduced formulation of the strand with virtual nodes, the system matrix size is quite small, and therefore the velocity solve is inexpensive compared to other simulators. In fact for all the simulations that we ran, the velocity solve accounted for less than 1% of the total cost. The bottleneck was in node adaptive resampling (50%) and force evaluation (30%). Since both of these steps involve memory-intensive for-loops, they can be performed much more quickly in a faster language, and we expect that porting the code would give us a considerable speed-up.

In our examples, the physical dimensions of the rigid bodies range from about 3 cm to 20 cm, and the strand radius from 1 mm to 3 mm. The simulation cost for a large scene (Fig. 6.13) was about 30 minutes for a 30 second simulation, with a 20 ms time step.

Pulleys

These simple pulley examples (Fig. 6.10) show that with the virtual node formulation, we can smoothly handle contact between a strand and a rigid body without having to resort to a system with unilateral constraints, which can be relatively expensive to solve compared to a bilateral system. The eccentric pulley shows a simple example of a non-circular/non-elliptical wrapped curve constraint. We also show that we get proper coupling of forces between hinged rigid bodies as well as free-floating rigid bodies, with simulations of multi-pulley systems.

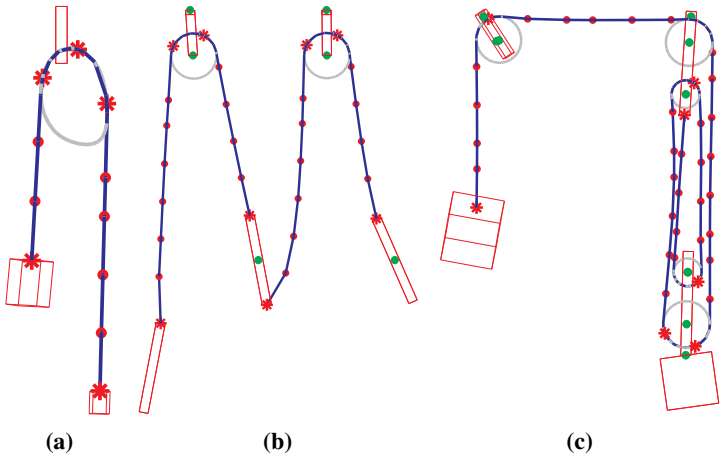


Figure 6.10: (a) Eccentric pulley, (b) Two single pulleys in series, and (c) Double pulley.

Elevator

The constraint curves through which we pass reduced nodes can be any space curve, and are not limited to circles and lines. To illustrate this point, we modelled a simple cable-driven elevator with the cable passing through the floor of the platform using a spline, as shown in Fig. 6.11. As it passes through the platform, the strand bends at an acute angle, but it is still able to transmit force properly around the corner. By applying a torque to the bottom wheel, the platform can be pulled up or down; the tension is transmitted properly even though there are two sharp kinks as the strand goes through the platform. Additionally, because of the way the cable crosses over, the platform naturally comes to a stop when approaching the top or bottom wheel. The transmission of force works just as well even if the line through the floor is replaced by a more complicated spline path.

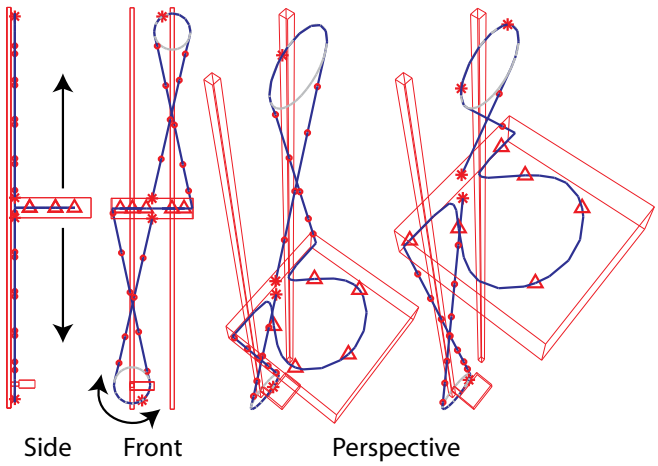


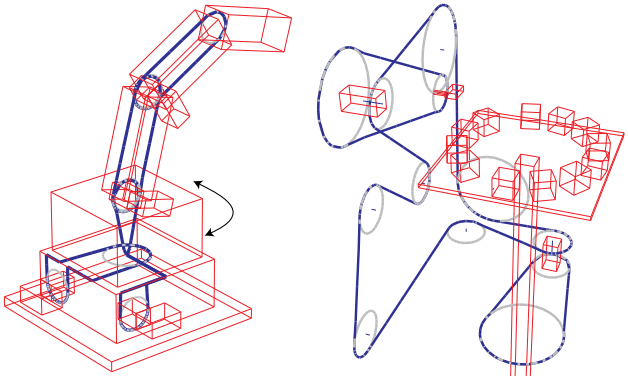
Figure 6.11: Cable-powered elevator: The nodes of the strand are shown as red circles (maximal), triangles (reduced), and asterisks (rigid). The platform is powered up and down by the crank at the bottom. The cable is routed through a spline curve inside the platform. The tension is not lost at the sharp corners at the sides of the platform.

Gears

In this example (Fig. 6.12b), a system involving two 2:1 gear boxes are simulated, resulting in a gear ratio of 4:1 between the crank and the platform. Midway through the simulation, the direction is reversed to show how the momentum of the platform causes the driving cable to react dynamically.

Robotic Arm

The robotic arm in Fig. 6.12a shows a mechanism by which torque can be transmitted across a rotating body, using a small aperture for routing the cable. The steering wheel to the left controls the rotation of the whole robot, and the wheel to the right controls the proximal joint. Even when the body is rotated, the cable responsible for controlling the proximal arm does not change its length, because



(a) Cable-driven robotic arm

(b) Cables and gears

Figure 6.12: (a) Robot arm: the cable for controlling the proximal arm is routed through a small aperture so that the arm can be controlled even when the robot body is rotated. (b) A cabled system with two 2:1 gear boxes.

the cable goes through the centre of rotation.

Large-scale Compound Pulleys

A distinct feature of pulley systems is that the force generated is linearly proportional to the number of pulleys implemented. This is referred to as the mechanical advantage of the system. We demonstrate an example illustrating how our framework can reproduce this relationship. An alternating set of hanging and fixed pulleys is constructed, and a weight is rigidly attached to the hanging pulleys, with a *single* strand, routed around all of the pulleys. One end of the strand remains fixed while its other end is attached to a counter weight, acting as an input force. The weight required to keep the system in equilibrium is calculated for a varying number of pulleys. The system is frictionless, with the strand and pulley masses set small as to be negligible. A near-perfect linear relationship is reproduced through

# Lower Pulleys	Weight (kg)
1	500.3
2	250.1
4	124.8
8	62.4
16	31.2
32	15.7
64	7.8

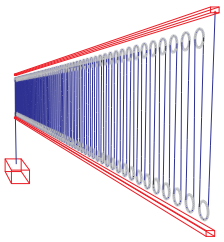


Table 6.1: Compound pulley test: Pulleys are inserted between two horizontal bars. A single strand is passed between all the pulleys, attached at one end to a hanging weight (box) and at the other end to the fixed upper bar. The lower bar weighs 1000 kg. The table shows the weight of the box required to hold in equilibrium the lower bar, with varying number of pulleys.

our framework, as shown in Table 6.1, for up to 64 pulleys in series connected by a single strand.

Line Shaft

The final example is a large-scale simulation of a line shaft, driven with a simple feedback controller (Fig. 6.13). Power is transmitted from the overhead line shaft to various components, such as an eccentric pulley, a double-pulley, an elevator, a robotic arm, etc., using crank-rocker linkages with different link ratios to produce oscillatory motion. Since there are no spurious bending and frictional effects in the system, the torque from the shaft is transmitted properly across numerous constraints. All of the submachines are dynamically linked, with no "islands" and kinematically scripted objects. All objects are simulated together and are dynamically coupled; if we forcefully jam any one of the components, the whole system will come to a halt.

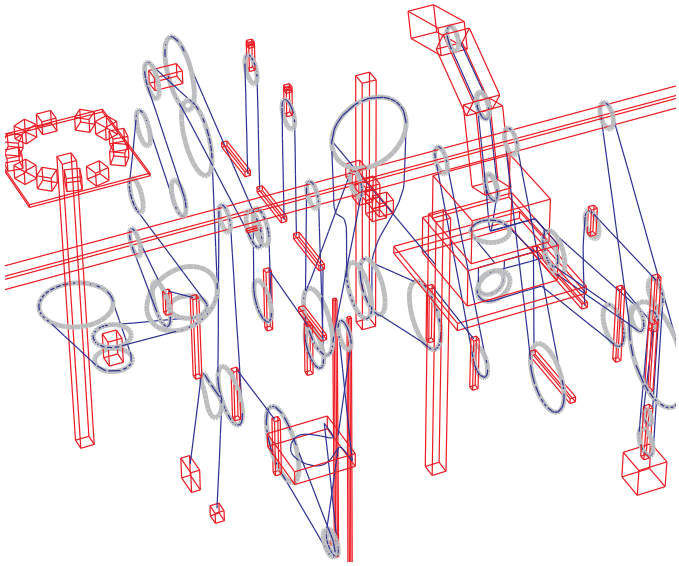


Figure 6.13: Large-scale simulation—subcomponents connected by an overhead line shaft.

6.5 Summary

The Reduced Strand is an improvement on the Spline Strand in both force transmission and routing constraint handling. It does not suffer from undesirable coupling between the stretching and bending modes, and the addition of a virtual node allows it to bend sharply at any location along the length of the strand. The simpler linear basis also allows us to introduce different types of constrained nodes, making the system size minimal, with just the right degrees of freedom.

Since it does not take into account the shape of the strand, it cannot efficiently model volumetric muscles. There are several ways to overcome this problem. One option is to add transverse springs (or even strands) that apply volume-preserving forces. This approach is appealing because of its simplicity, but would suffer from the usual problems with mass-spring simulators. Penalty-force based methods are

not stable when there are many objects stacked on top of each other, and in some areas of the body, such as the thigh, there are many muscles layered on top of each other. Another approach would be to mix reduced strands and Cosserat strands. Reduced strands could be used for tendons and thin muscles, and Cosserat strands can be used for volumetric muscles. An embedding approach is also possible, where reduced strands are embedded inside a volumetric simulation.

Chapter 7

A Controller for Musculoskeletal Systems

Because of the complexity of the routing of tendons and muscles, even simple tasks, such as moving a finger from one position to another, requires the coordinated activation of several muscles, acting as synergists and antagonists. It is a virtually hopeless task to attempt control of such a system manually, for example with a GUI—an algorithmic controller is needed.

For the purpose of our simulation, the job of the controller is to compute the activation levels of the muscles, which innervates the muscles to derive the skeleton to follow some target trajectory. We assume that the simulator has the following form, such that any strand formulation should work, as long as the contractile force

This chapter is based on Sueda, Kaufman, and Pai [88].

is linear in the activation level, a:

$$\mathsf{M}\dot{\mathsf{q}} = \mathsf{f} + Aa, \tag{7.1}$$

where M is the system mass matrix, $\dot{\mathsf{q}}$ is the system velocity vector, f is the system force vector, and A is a binary selection matrix that maps the activation vector, a, to the appropriate DoFs. The KKT system from Equation 3.14 can be converted to this form, by first extracting out the active force, $f_a = Aa$, from the rest of the forces, assuming that the active force is linear in the activation levels, and then combining the velocities and Lagrange multipliers into a single vector, $\dot{\mathsf{q}}$.

$$\underbrace{\begin{pmatrix} M & G^T \\ G & 0 \end{pmatrix}}_{\mathsf{M}} \underbrace{\begin{pmatrix} \phi^{(k+1)} \\ \lambda \end{pmatrix}}_{\dot{\mathsf{q}}} = \underbrace{\begin{pmatrix} \mathsf{f} + Aa \\ 0 + 0 \end{pmatrix}}_{\mathsf{f}+Aa}. \tag{7.2}$$

The length of the vector $\dot{\mathsf{q}}$ is the number of DoFs in the system *plus* the number of constraints. For a system with rigid bodies and spline strands we have,

$$\dot{\mathsf{q}} = \begin{pmatrix} \phi_0 & \dots & \phi_n & \dot{\boldsymbol{q}}_0 & \dots & \dot{\boldsymbol{q}}_m & \lambda_0 & \dots & \lambda_l \end{pmatrix}^T. \tag{7.3}$$

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

If the input target is a sequence of rigid body configurations rather than velocities, it can be converted into the required form by computing the spatial velocities needed to move the bodies from their current configurations to the target configu-

rations in a time step, using the matrix logarithm.

$$\phi =]\log(\mathsf{E}^{-1}\,\mathsf{E}^{target})[\,, \tag{7.4}$$

where E is the current rigid body configuration, and E^{target} is the target rigid body configuration. The unbracket operator, $]A[$, extracts the spatial velocity vector from the argument matrix (See Equation 3.3).

The reference animation specifies the target velocities of rigid bodies, v_x. This can be either a 3D point velocity or a 6D spatial velocity, and the total size of v_x is (3 $\times$ #point targets + 6 $\times$ #spatial targets). We consider the following two examples:

1. If there are n rigid bodies in the simulator, and if all of them have spatial velocity targets, then v_x would be a vector of size $6n$ composed of n target spatial velocities stacked on top of each other: $v_x = (\phi_0^{target} \ldots \phi_n^{target})^T$.

2. If there is a single 3D point target velocity (say at the tip of the end effector), then v_x would be a vector of size 3 consisting of that target velocity vector: $v_x = (\boldsymbol{v}_n^{target})$.

We then require that the controller computes the muscle activations to match the resulting system velocities to the desired velocities. That is,

$$\Gamma_x \dot{\mathsf{q}} = v_x, \tag{7.5}$$

where the entries of the Jacobian Γ_x contain the 6×6 identity matrix for spatial velocity targets and the 3×6 matrix Γ from Equation 3.5 for point velocity targets.

For Examples (1) and (2) from above, we have

$$\begin{pmatrix} I & \dots & 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & I & 0 & \dots & 0 & 0 & \dots & 0 \end{pmatrix} \dot{\mathsf{q}} = \begin{pmatrix} \phi_0^{target} \\ \vdots \\ \phi_n^{target} \end{pmatrix}, \qquad (7.6)$$
$$\begin{pmatrix} 0 & \dots & \Gamma & 0 & \dots & 0 & 0 & \dots & 0 \end{pmatrix} \dot{\mathsf{q}} = \begin{pmatrix} \boldsymbol{v}_n^{target} \end{pmatrix}.$$

Substituting Equation 7.1 into Equation 7.5 and eliminating $\dot{\mathsf{q}}$, we arrive at the *targeting constraint equation*

$$H_x a + v_f = v_x, \qquad (7.7)$$

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

In some cases, it is possible to solve for the activations using the targeting Equation 7.7 as a hard constraint. In most cases, however, the dynamics of the system cannot *exactly* follow the requested targets. For example, the bone joint constraints may prevent certain poses, or muscles may not be strong enough to produce the required force. Instead, we convert the targeting equation into a quadratic minimization problem.

To make the dynamics follow the target trajectory as closely as possible, the

controller may sometimes return activations that switch spastically. In order to prevent this, we add a damping term to the objective function, using the linear approximation of the derivative of the activation. (In a real muscle, this is automatically taken care of, because activation, or excitation, is not a linear function of innervation, but is better approximated as a first order ODE [109].) In addition, to minimize the total activation, we add an "activation energy" term to the objective. Putting this all together, the activation optimization problem is

$$\begin{aligned} \min_{a} \quad & w_a\|a\|^2 + w_x\|(H_x a + v_f) - v_x\|^2 + w_d\|a - a^{(0)}\|^2 \\ \text{s.t.} \quad & 0 \le a \le 1, \end{aligned} \tag{7.8}$$

where w_a, w_x, and w_d are the blending weights, and $a^{(0)}$ is the activation from the previous time step. The first term minimizes the total activation and also adds regularization to the quadratic problem, whereas the second and third terms function as a spring-and-damper controller to guide the dynamics of the system toward the target motion. For easy motions that are not over-constrained, w_a and w_d can be set to zero to achieve close tracking of the target. However, for some motions, especially those that involve configurations that are almost singular, such as full flexion or extension, regularization (w_a) and damping (w_d) help stabilize the solution. For the animations in Section 5.4, we used $(w_a, w_x, w_d) = (1, 1, 0.01)$.

The weight for the first term, w_a, need not be a scalar; each muscle can have its own weight, and can include an additional weighting term such as the muscle's physiological cross sectional area. Herzog [37] takes a similar approach, and additionally includes an inverse factor for the corresponding moment arm of the muscle so that muscles with large moment arms are recruited preferentially. Other

weighting terms can added, such as average muscle stress [21].

Although the original dynamics equation can be large (albeit very sparse), the dimensionality of the activation quadratic problem is the number of muscle strands, which in general is small (< 100). The main computational cost is in the construction of the quadratic matrix, which requires $min(\#targets, \#muscles)$ sparse solves on the KKT matrix from Equation 3.14. However, since the LU factors of the KKT matrix are required for solving the system dynamics anyway, the only additional cost is the backsolve, which is relatively cheap. In practice, we note that the controller adds no significant computational cost to the overall system.

7.1 Control of Human Eye Movement

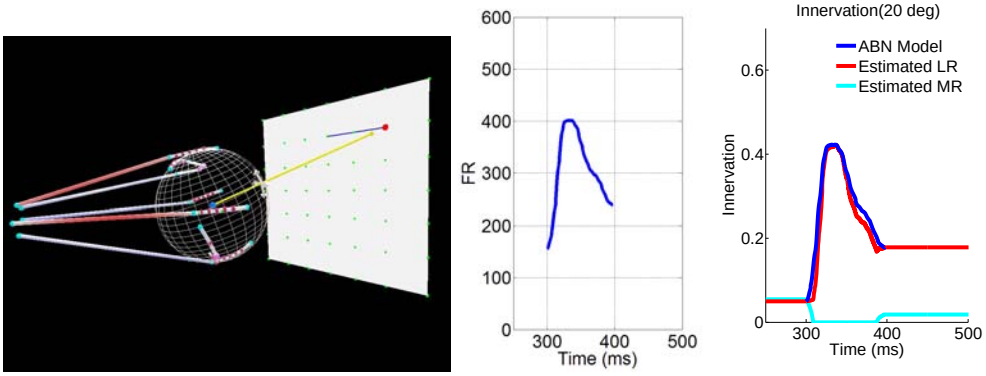
We have used our controller for computing the activation levels for the motions of the hand in Section 5.4 and for the eye (rest of this section). The controller has also been used by Stavness et al. [87].

The oculomotor plant presents an interesting application of the strands framework, because the geometries of the six Extraocular Muscles (EOM)s are relatively simple and are well isolated from the rest of the musculature. We used strands to simulate various eye movements and the associated defects, such as smooth pursuits, saccades, and acute superior oblique palsy [102]. In particular, we simulated saccadic movement using experimental EOM neural activity data as the neural input of the Lateral Rectus (LR) muscle to drive the orbital plant. We then compared the simulated motion to the desired motion to evaluate the system behaviour.

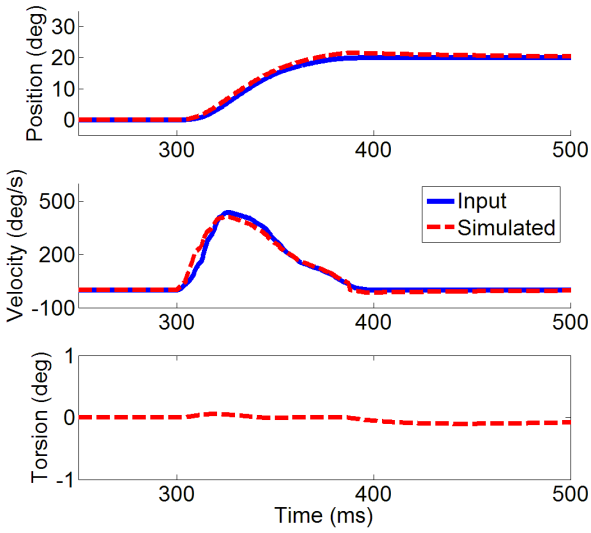
We used the controller to compute the activation levels of the six EOMs from the velocity profile taken from a saccadic experiment data set of a monkey from

This section is taken from Wei, Sueda, and Pai [102].

Sylvestre and Cullen [89]. We then compared the computed activation levels, as well as the resulting forward simulation, to their experimental model.



(a) 20 degree saccade **(b)** Model-based firing rate **(c)** Innervation



(d) Simulation validation

Figure 7.1: Simulated 20 degree abduction saccade in 20 degree elevation.

The neural drive, or the activation level, of a saccadic eye movement can be

characterized by a *pulse* component to overcome the viscoelasticity of the orbital plant, a *step* component to stabilize the eye in the new position, and a *slide* component that models the gradual transition between the pulse and step [76]. The neural control of saccades has been studied extensively [76, 89]. The activities of neurons are characterized by the discharge rates recorded at the neuronal sites. The movement of the LR muscle is controlled by Abducens Neurons (ABN) whose discharges during saccades can be expressed as a first order equation [89]:

$$\text{Firing Rate} = b + kE + r\dot{E}. \tag{7.9}$$

This model approximates the ABN firing rates as a linear function of the eye position E and velocity $\dot{E}$. b is a bias constant, which is the neural firing rate at the stationary central eye position. k and r are constants and their optimal values have been estimated by fitting the above model to the actual neuron discharge recordings from monkeys [89].

Figure 7.1 shows the analysis of a 20 degree abduction saccade while the eye is elevated by 20 degrees. Here we experiment with a saccade from a secondary position to a tertiary position, which is more interesting and challenging than a purely horizontal saccade. The torsional component in the tertiary gaze is a standard test to evaluate the biomechanics of eye movement. This movement is generated by contraction of the LR muscle, relaxation of the Medial Rectus (MR) muscle, and co-contraction of other EOMs.

Figure 7.1a is a snapshot of the simulated gaze trajectory and the orbit configuration when saccade is completed. Muscle activation levels computed from the controller are shown in red on the EOM strands. The LR and Superior Rectus (SR)

muscles are the two most active EOMs. Figure 7.1b shows the approximated ABN firing rate profile for a 20 degree saccade based on the model in Equation 7.9. The model coefficient values are $b = 156$, $k = 4.2$, and $r = 0.42$. Figure 7.1c plots our estimated innervations of the LR and MR muscles—the primary rectus muscles contributing to horizontal saccades in red and cyan respectively. The other EOMs maintain the vertical elevation of the eye throughout this movement. Note that the computed LR innervation has the pulse-slide-step characteristics of saccadic neural control. Also note that the computed MR innervation realistically models the behaviour of antagonist motoneurons during OFF direction saccades—most antagonist motoneurons completely cease firing after saccade onsets [89]. The model-based firing rate profile in Figure 7.1b is linearly scaled such that the starting and ending values match the computed innervation. The scaled profile is plotted in blue in Figure 7.1c, overlaid on our estimated activations for comparison. We observe that the computed LR activation dynamics (in red) agrees with the experimental ABNs discharge dynamics (in blue), which shows that our controller realistically captures the biomechanics of the EOMs.

Saccade Amplitude (°)	Position RMSE (°)	Velocity RMSE (°/sec)	Max Velocity Difference (°/sec)	Max Torsional Error (°)
10	1.27	34.31	6.31	0.16
20	1.27	32.93	24.00	0.10
30	1.78	41.27	12.22	0.09

Table 7.1: Root mean squared errors (RMSEs) of the simulated saccadic positions and velocities, and maximum torsional errors.

We then replace the computed activation of LR with the scaled empirical neural discharge profile and re-simulate the 20 degree abduction saccade. The blue curves

in Figure 7.1d are the desired position and velocity profiles, based on saccade traces recorded from monkeys [89]. The superimposed red curves are the simulation results. The simulated saccade reasonably follows the desired saccadic trajectory. The root mean square errors (RMSEs) are analyzed quantitatively in Table 7.1. The associated torsion is very small, which further demonstrates the simulation accuracy and realism of the nonlinear EOM model in reproducing saccades.

Chapter 8

Hand Imaging

In the previous chapters, the focus was on understanding the functions of the hand through biomechanical modelling using *computer simulations*. In this chapter, we describe a complementary approach; we start from *medical images* and glean insight from real world data. In Sections 8.1-8.4, we will go over the cadaver hand imaging apparatus, including the hardware, software, and scanning protocol used, for acquiring the rich data set consisting of skeletal trajectory, tendon tensions and excursions, and tendon marker trajectories. Then in Section 8.5, we will describe our first attempt at simulating a simple model of the index finger with real world data acquired from *ex vivo* specimen, using the strands framework developed in the previous chapters.

This complementary approach is useful because the hand is one of the most challenging parts of the human body to model due to the complex routing of tendons. In particular, the extensor mechanism of the finger, composed of the extensor hood and a network of tendons and ligaments, is extremely difficult to model (Figure 8.1). The biomechanical structures of the digits are stacked on top of each

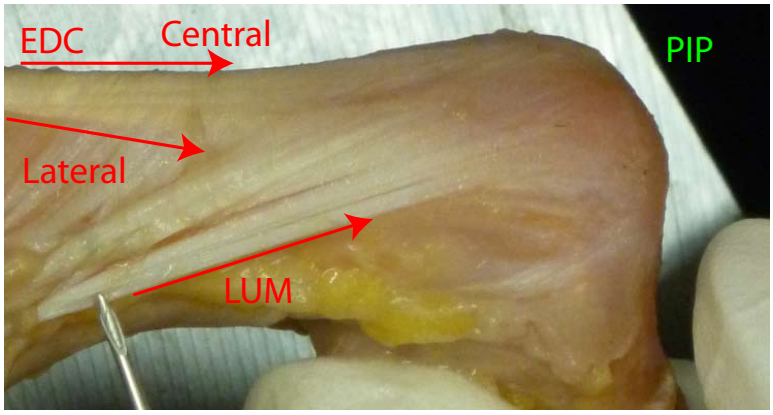


Figure 8.1: Radial view of the extensor hood. The central and extrinsic lateral bands of the EDC can be seen below the semi-transparent extensor hood. The lumbrical inserts into the bottom of the hood, near the needle point.

other in multiple layers, with each structure having an important mechanical function, such as supporting the joint, applying a torque, or holding a tendon in place.

As mentioned in the introduction and related work, we can get significant amount of data from *in vivo* experiments, such as electromyography [17, 98], biplanar X-rays [61], motion capture [15], or surgical implants [67]. However, with *ex vivo* cadaver imaging apparatus, we are able to acquire these time-varying measurements simultaneously, and have more control over the acquisition setup.

This experiment was a collaborative effort between the Digital Human Research Center (DHRC), Keio University, and the University of British Columbia. Dr. Mitsunori Tada (DHRC) designed and built the hardware, and Dr. Toshiyasu Nakamura and Dr. Yusaku Kamata (Keio) provided the cadavers and performed the dissections.

The strands framework can be used for most of the biomechanical structures of the hand, such as muscles, tendons, ligaments, and bones. However, volumetric

(a) Flexors **(b)** Suturing the EDC **(c)** Suturing the DI

Figure 8.2: Cadaver dissection, showing extrinsic and intrinsic musculotendons.

tissues, such as cartilage, will require another modelling primitive. Because of its complexity, it is difficult to fully model the functions of all the biomechanical structures in the hand. Instead, we concentrate on the seven musculotendons of the index finger[1]: Flexor Digitorum Profundus (FDP), Flexor Digitorum Superficialis (FDS), Extensor Digitorum Communis (EDC), Extensor Digitorum Indicis (EDI), Lumbrical (LUM), Dorsal Interosseus (DI), and Palmar Interosseus (PI). Of these, FDP, FDS, EDC, and EDI are called "extrinsic", since their muscles are located outside the hand in the forearm, and DI, PI, and LUM are called "intrinsic" because the muscles are located in the hand itself.

Although the extrinsic muscles of the hand are readily identifiable after some of the most superficial structures are removed, the intrinsic muscles are extremely difficult to separate and identify. The structures shown in anatomy textbooks [63] look clean and well-separated, but in a real cadaver hand, these intrinsic structures can be exposed and identified only by a trained surgeon.

An important step toward understanding the structure and function of these muscles is to measure how they move and transfer force during finger movement.

[1] Not all fingers have the same set of musculotendons.

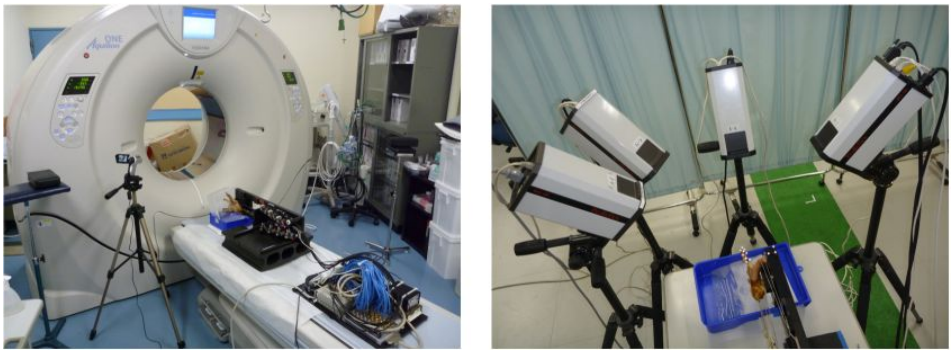


Figure 8.3: Experimental setup with CT scanner (left) and motion capture (right)

We built a cadaver scanning apparatus (Figure 8.4) that is capable of simultaneously measuring the skeletal trajectory, tendon excursion and tension, and tendon marker locations.

We performed two experiments with the apparatus (Section 8.3):

- *Isolated lumbrical experiment*: Measuring the effect of the lumbrical on the functions of an isolated extrinsic musculotendon (FDP, EDC, FDS, and EDI).
- *Combined kinematic experiment*: Measuring the combined kinematics of all the musculotendons during a complex finger motion, such as hook flexion or claw finger.

8.1 Hardware Setup

Each tendon was actuated by a servo-controlled DC motor (Dynamixel RX-28) with a maximum rated force output of $\sim 10N$ and stroke of $\sim 10cm$, which provided ample operation space, since the maximum tendon excursion in the hand is about $5cm$. The wire from the motor was routed through a strain gauge, shown in dotted yellow box in Figure 8.4, then to a suture, which was tied to the tendon. A

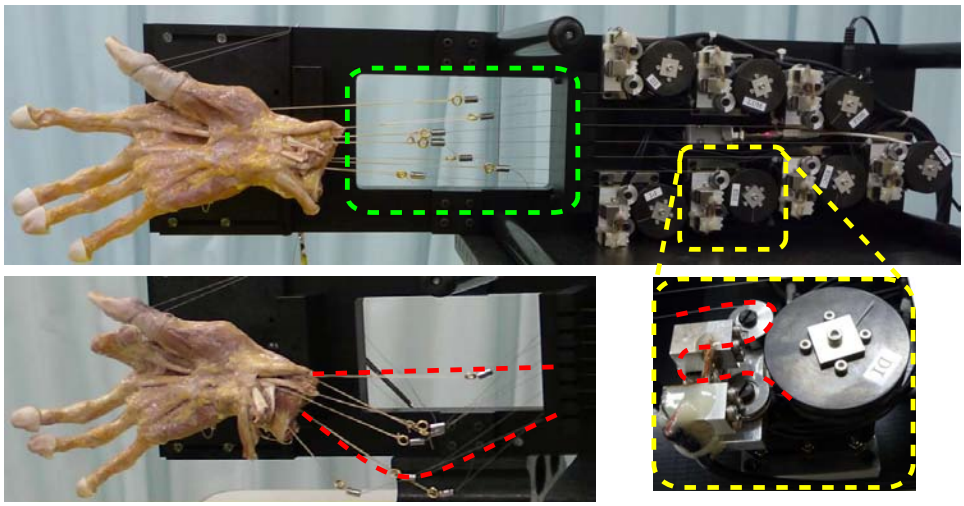


Figure 8.4: Wire routing in the scanning apparatus. The wire from the motor is routed through a strain gauge and attaches to the suture tied to the tendon. A weight is attached to each wire so that any slack is passively removed.

weight was added to the wire, as shown in the green box in the figure, so that any slack could be removed passively during the strain gauge taring process, and also so that a tendon could be better isolated in the lumbrical experiment. The cadaver hand was placed with the palm in the vertical plane, in an attempt to minimize the effects of gravity during flexion and extension.

Tendon markers were inserted into the muscles and tendons using a custom injection device. The locations of the markers are shown in Figure 8.5. After some trial-and-error, we determined that stainless steel markers with a diameter of $0.5mm$ provided optimal scanner visibility and handling. Smaller markers were hard to segment in the scans, and larger markers hindered proper movement of the tendons.

The motion of the finger was recorded using a Toshiba Aquilion ONE ® CT

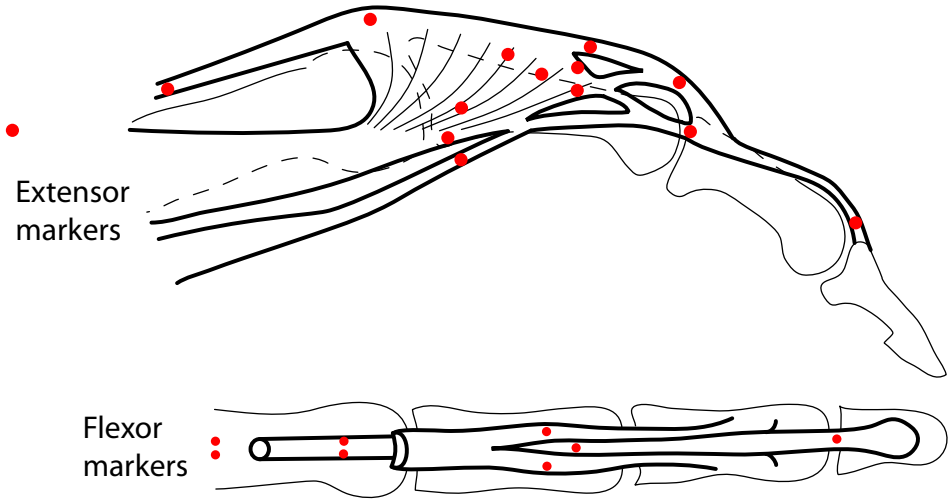


Figure 8.5: Locations of inserted tendon markers.

scanner with a maximum field-of-view of $160mm \times 244mm \times 244mm$ and matrix size of $320 \times 512 \times 512$ voxels. This scanner was capable of capturing dynamic image volumes at $3Hz$. A Qualisys motion capture system (Figure 8.3) was also used. The skeletal trajectory can be obtained from either system, but the tendon marker trajectory can only be obtained from the video CT scanner. However, the operational cost of a CT machine is very high (thousands of dollars), and the machine tends to be back-logged for other uses. Motion capture, on the other hand, does not give us the tendon marker locations, since the markers inside the muscles or tendons are not reflective, but the speed and ease of capture as well as post-processing is a major advantage over using the video CT scanner.

Considering the advantages and disadvantages of each system, we designed the apparatus so that it could be used for both the CT scanner and for motion capture. Because ferromagnetic materials cause significant noise in the CT scans, we used metallic support structures only when necessary. For motion capture, we rigidly

Figure 8.6: Bone motion capture markers

attached three markers to each bone, as shown in Figure 8.6, and took a single static CT scan of the cadaver hand with the markers to get the relative positions of the markers and bones. Then any subsequent skeleton configuration was reconstructed from these marker positions.

8.2 Cadaver Preparation

The first step in the scanning process was to prepare the cadaver. The extrinsic tendons were easily identified, and the sutures were stitched onto their ends (Figure 8.2b). The placement of sutures for intrinsic muscles were not as clear. The origin of an interosseous muscle, for example, is quite wide, and the placement of the suture can make the tendon into a flexor or an extensor (Figure 8.2c). Again, it is worth noting that these structures are not very well separated, with everything seemingly attached to everything else. We attached the sutures to the middle of the insertion area of the intrinsic muscles. The lumbrical must be treated with extra care, as its routing in the hand is very complex—its origin is not on a bone but on the FDP tendon. In order to preserve the force direction of the lumbrical, we

attached a small plastic tube to the FDP and routed the suture from the lumbrical through it. In this way, whenever the lumbrical was pulled, the direction of the force was aligned with the FDP.

Two screws were inserted into the metacarpal for attaching the cadaver to the apparatus base. Although non-ferromagnetic materials are preferred for these screws, metal screws were necessary to keep the cadaver from moving when the tendons were pulled. The screws were placed so that they provided sufficient strength while minimizing contact with the tendons. The attachment of the screws to the apparatus base was designed so that the angle of the hand could be adjusted slightly if necessary. After each scanning session, the cadaver hand was placed back in the flash freezer with the mounting screws and sutures attached. This ensured that the relative position and orientation of the hand to the base and the length of the wires remained constant across multiple scanning experiments on different days.

The postmortem properties of tissues are affected by both mechanical loading and freeze-thaw cycles. The stiffness can change by as much as $\sim$10% for bones [55], $\sim$40% for skeletal muscles [99], and $\sim$35% for cartilages [94], while the change is negligible for ligaments [62]. Although some of these changes are significant, they are not important for our experiment because we are interested in the kinematics of the skeleton and the tendons in this cadaver study. Giannini et al. [32] report that, for the human posterior tibial tendons, even though the Young's modulus is unaffected by freeze-thaw cycles, the cross-sectional area is increased by over 17%. This difference is significant, but should not play an important role in the qualitative analysis in Section 8.5. In the future, however, the corresponding increase in tendon stiffness should be taken into account for a quantitative study.

Once the cadaver was attached to the base, the sutures were tied to the wires from the motors (green box in Figure 8.4). We used a split-ring washer for this connection, which allowed us to attach and remove the wires to the sutures quickly, and also served as the predefined breaking point, in case the tension in the tendon became too large. The relative lengths of the sutures to the wires were also important, since the washers needed sufficient headroom in the area between the plate and the hand for all possible joint angles of the finger (green box in Figure 8.4). To do so, the finger was held in its neutral, straight position (neither flexed nor hyper-extended), and the washers were positioned closer to the hand for the FDP and FDS, closer to the plate for the EDC and EDI, and in the middle for the DI, PI, and LUM. With these starting washer positions, the FDP and FDS washers had enough headroom toward the hand (to the left in Figure 8.4) when the finger was flexed, and likewise, the EDC and EDI washers had enough headroom toward the plate when the finger was hyper-extended.

8.3 Software and Protocol

A custom software package, *Kyadabah* ("Cadaver" in Japanese), shown in Figure 8.7, was created in order to view the tension and length data in real-time, as well as to edit, import, and export tendon excursion profiles. Having a visual editor was indispensable when designing and adjusting the experiments, and being able to see the tension in real-time was especially appreciated by the surgeons. We divided our experiments into two main categories: the isolated lumbrical experiment and the combined kinematic experiment.

In the isolated lumbrical experiment, a single extrinsic tendon (FDP, EDC, FDS, and EDI) was pulled at a constant speed, with different constant loads on the lum-

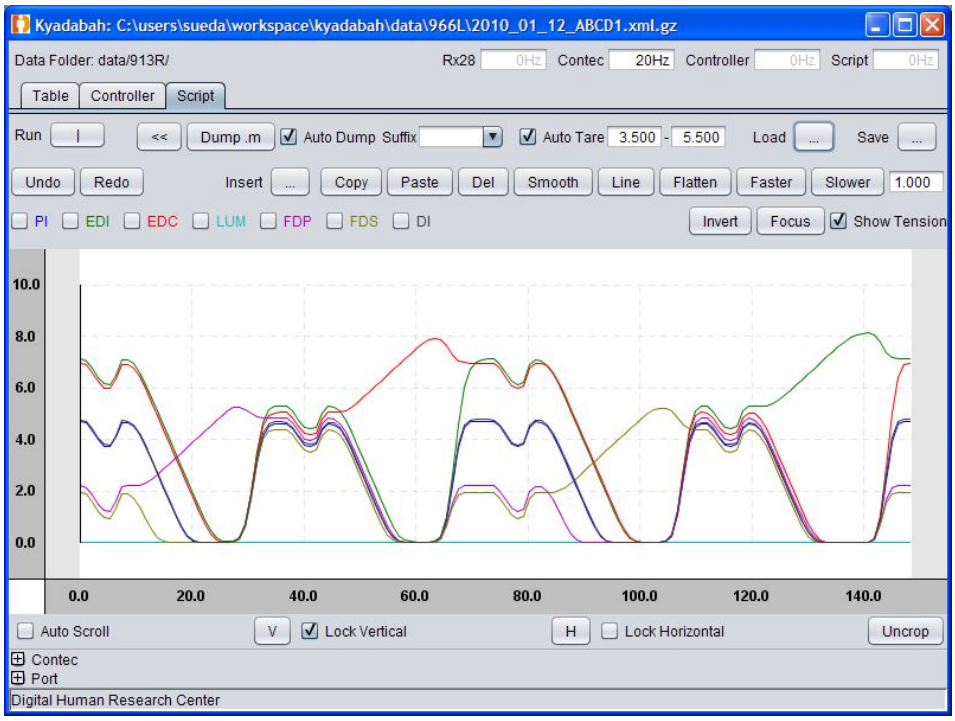


Figure 8.7: Our custom software, *Kyadabah*, for creating, editing, and playback of tendon excursion sequences.

brical. For a schematic diagram of the FDP experiment, see Figure 8.12. The lumbrical can act as an agonist or antagonist, depending on the finger pose, and has an interesting effect on the FDP and FDS. A data profile from the experiment in Figure 8.7 is shown in Figure 8.8, in which the FDP, EDC, FDS, and EDI were pulled in isolation. Note that because we want to be able to concatenate any number of these experiments together, we designed the scripts so that each experiment started and ended with predefined tendon excursions, labelled (A) and (B) in blue. (A) corresponded to the canonical extended position, and (B) corresponded to the canonical flexed position. This ability to concatenate experiments, and to scan multiple experiments in sequence with minimal delay in between, was crucial when taking

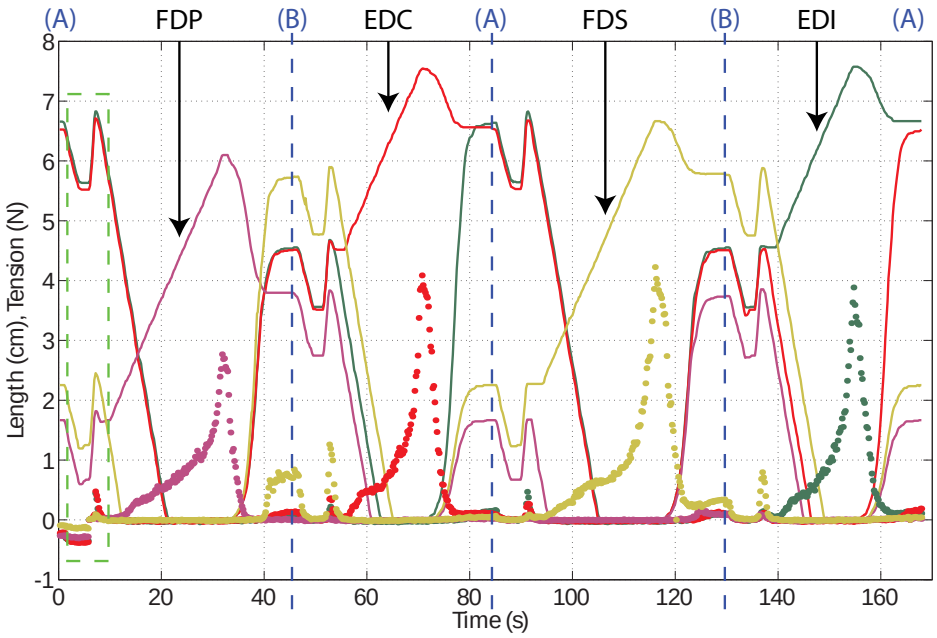


Figure 8.8: Length output (lines) and force output (dots) from an isolated lumbrical experiment (Specimen #913R).

multiple scans in the CT machine, because of the resource and scheduling cost with the scanner. At the beginning of each experiment, the tensions in all tendons were released by adding 1cm of slack to the wires for a few seconds to tare the tension sensors (dotted green box in Figures 8.8 & 8.9). Then each extrinsic tendon was pulled in isolation, while the others were released simultaneously. The experiment was then repeated with a different weight on the lumbrical (0g, 50g, 100g, 250g, 200g).

In the combined kinematic experiment (Figure 8.9), the finger was moved by the experimenter and the resulting tendon excursions were recorded for later playback. A PD controller was used to keep the tensions in the tendons at $1N$. The finger was moved by the experimenter while the tendon excursion lengths were

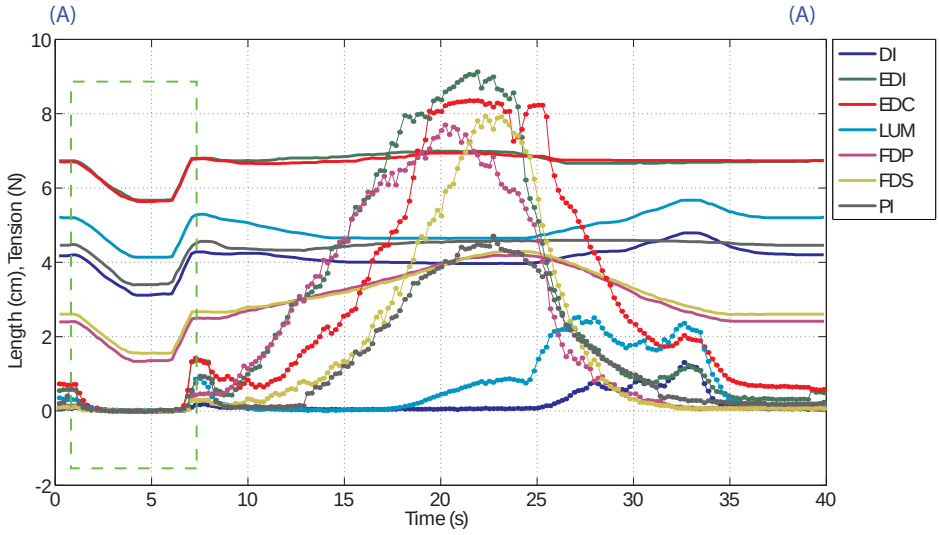
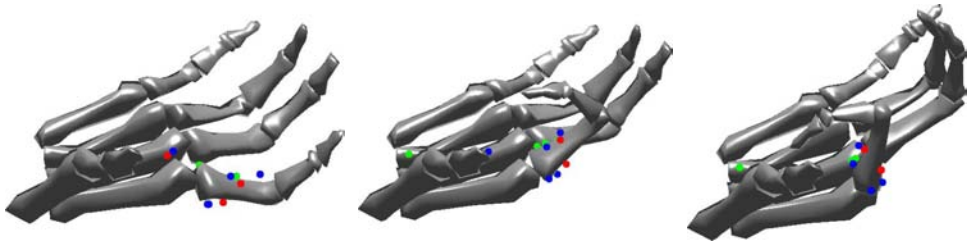


Figure 8.9: Length output (lines) and force output (dots) from a combined kinematic experiment (Specimen #913R).

recorded. These recorded tendon excursions were then used to drive the cadaver hand automatically at a later time inside the CT scanner. The resulting motion varied from the original motion because of the non-zero force that was applied to the finger by the experimenter during the recording process. Therefore, the recorded lengths were tweaked using the editing capabilities of *Kyadabah* to make the finger motion match the desired motion as closely as possible. These experiments were also scripted to have the tension taring period as before (dotted green box), and to have the same starting and ending finger poses so that they could be concatenated together.

8.4 Segmentation

Reconstruction of skeletal and musculotendon data from raw medical data is in its own right a challenging problem. We used the shape-matching registration frame-



(a) (b) (c)

Figure 8.10: Segmented bones and tendon markers

work of Gilles and Pai [34] and Gilles and Magnenat-Thalmann [33] to segment the volumetric data.

With the CT video sequences, the amount of raw data captured is extremely large. We performed about 120 scans of lumbrical and kinematic experiments, each of which is roughly 30 seconds long, taken at $3Hz$. Each one of these static scans has 320 slices of 512×512 pixels. Taken together, this results in a data set of *over 1TB*.

The segmentation software [33, 34] is a template-based approach. First, a template bone mesh hierarchy (DAG) was created, and the meshes were elastically aligned to the first CT volume to produce subject-specific meshes. Then for each subsequent CT video volumes, the meshes were aligned to the image data as closely as possible using rigid joint constraints. This gave us the subject-specific meshes for the skeleton from the first frame, and a sequence of transforms for each bone in the skeleton for each of the subsequent CT video frames (Figure 8.10). The tendon markers were segmented similarly. The markers were placed manually on the template meshes for the first frame. Each marker was assumed to be rigidly attached to a bone, and then in the next frame, they were first transformed rigidly

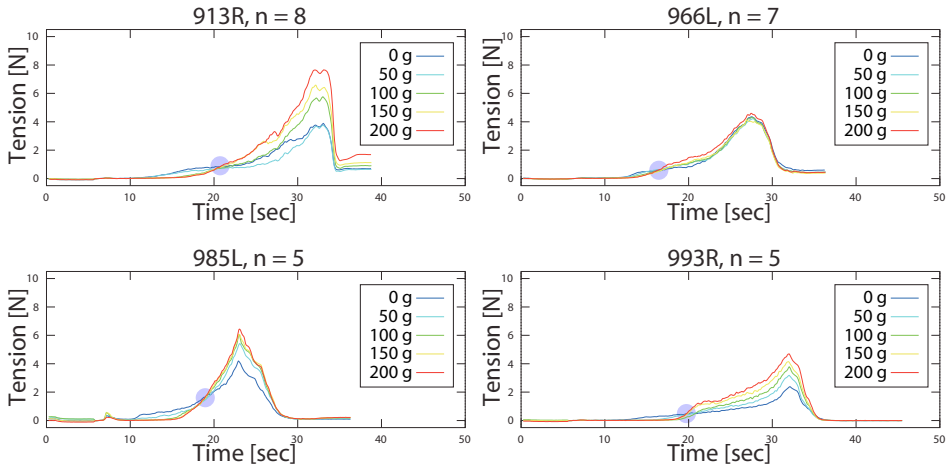


Figure 8.11: FDP tensions with different LUM weights. Each of the four graphs corresponds to one specimen, and the title indicates the specimen name and the number of trials run. The FDP was pulled at a constant rate of 2mm per second, with different constant weights on the LUM (0g - 200g). The graphs show the mean of n trials for each weight.

with the bones. Then their exact locations were searched in the volume using these transformed positions as the initial guess.

8.5 Preliminary Analysis of the Isolated Lumbrical Experiment

In this section, we will first describe the patterns that were observed in the cadaver data. We will then go over the parameters used in the simulation, and show the resulting qualitative match between the cadaver data and simulation output.

8.5.1 Observed Pattern

In Figure 8.11, the FDP tension output of the isolated lumbrical experiment is shown. Each of the four graphs corresponds to a cadaver specimen, with the spec-

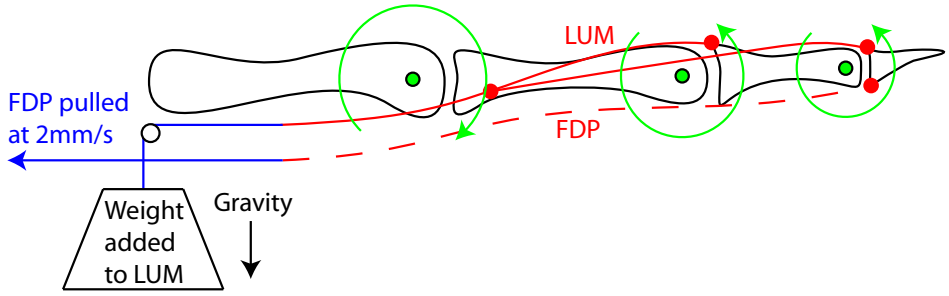


Figure 8.12: A schematic diagram of the lumbrical (solid) and FDP (dotted). The lumbrical flexes the MCP joint, and extends the PIP and DIP joints. A weight is attached to the lumbrical, and the FDP is pulled at 2mm per second.

imen number indicated in the figure titles. Starting at the neutrally extended pose, the FDP was pulled at a constant speed of 2mm per second to flex the finger. All other tendons were slack, except for the LUM, which had various weights attached to it (0g, 50g, 100g, 150g, 200g). For each LUM weight, the trial was repeated $n = 5..8$ times, as indicated in the figure titles. The plotted tensions are the means of the n trials for each LUM weight.

In each of the four specimens, there is a balancing point at which the FDP tensions cross, indicated by the blue circles in Figure 8.11. Initially, when the finger is extended, the FDP tension with a smaller LUM weight is lower than the tension with a larger weight. At around 20 seconds, these tensions are roughly equal, and as the finger is further flexed, the FDP tension with a larger LUM weight becomes higher than the tension with a smaller weight. Although the location of the balancing point is different in each specimen, they all exhibit this behaviour.

This behaviour is most likely due to the mechanics of the FDP and the lumbrical. Unlike the extrinsic muscles, the lumbrical can act both as a flexor and an extensor; the primary function of the lumbrical is to flex the MCP joint, and to

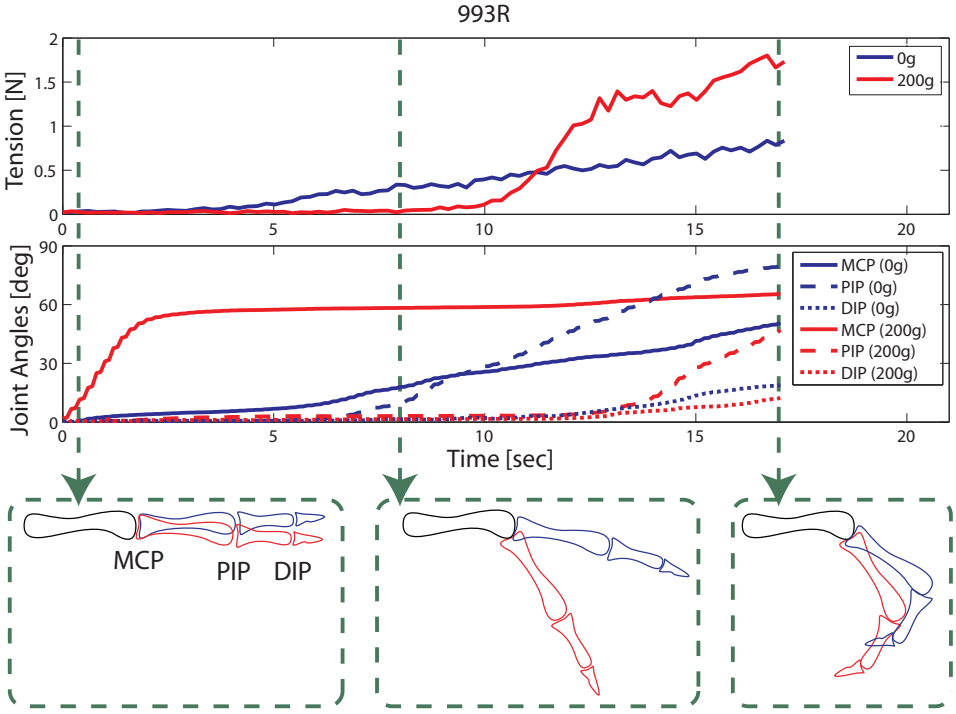


Figure 8.13: Tension and joint angles from specimen 993R, with lumbrical weights of 0g and 200g. The joint angles are relative to the neutral, straight position. In both cases (0g and 200g), the FDP is being pulled at a constant rate of $2mm/sec$.

extend the PIP and DIP joints. The rough schematic of its routing is shown in Figure 8.12. If there is a large weight attached to the lumbrical, the tension is initially low because the MCP flexes quickly, and slack develops in the FDP. As the FDP is pulled slowly, it becomes taut again and tension is restored, but now, the resulting tension is higher than before, since the lumbrical is acting against the FDP as an extensor of the DIP and PIP joints.

This effect can be seen clearly if we plot the three joint angles along with the tendon tensions. The data from specimen 993R with lumbrical weights of 0g and

200g are shown in Figure 8.13. The top graph shows the tensions and the bottom graph shows the three joint angles, relative to the neutral, straight position. When there is no weight attached to the lumbrical, the FDP flexes the MCP at a constant rate, as shown by the solid blue line in the bottom graph. The resulting tension on the FDP is also linear. However, when there is a 200g weight on the lumbrical, the MCP flexes quickly due to the weight, and reaches equilibrium, as shown by the solid red line. Although the FDP is being pulled at a constant rate, this quick change in the MCP joint angle builds slack in the FDP. Until this slack is removed, there is no tension in the FDP and the joint angles remain constant. At around 12 seconds, the FDP becomes taut again, and the PIP and DIP start to flex.

The interesting observation is that in each cadaver, this tension-crossing pattern happens at the same time for all LUM weights. In the four graphs in Figure 8.11, these crossings occur roughly 20 seconds from the start of the experiment. With the strand simulator, we are able to reproduce this pattern.

8.5.2 Qualitative Simulation Comparison

In order to make a meaningful qualitative comparison, we need to find reasonable parameters for the simulation model. In this section, we describe the parameters that were extracted from the cadaver data, and also some parameters that were chosen manually. These parameters are listed in Table 8.1. For a quantitative analysis, all of the parameters would have to be obtained from the cadaver data or other external references.

The inertia of the bones were obtained from the mesh geometry segmented from the static CT scans. The obtained values were smaller than the values reported in the literature [101] by about 50%. Unfortunately, we did not dissect the cadaver

Parameter	Value	Source
distal inertia	$0.8g^*$	cadaver data
middle inertia	$1.7g^*$	cadaver data
proximal inertia	$5.0g^*$	cadaver data
MCP axis	-	cadaver data
PIP axis	-	cadaver data
DIP axis	-	cadaver data
MCP stiffness	$0.01Nm/rad$	manual
PIP stiffness	$0.01Nm/rad$	manual
DIP stiffness	$0.01Nm/rad$	manual
MCP damping	$0.8N/rad/s$	manual
PIP damping	$2.0N/rad/s$	manual
DIP damping	$5.0N/rad/s$	manual
FDP Young's modulus	$0.7GPa$	Harris et al. [36]
FDP area	$7mm^2$	Harris et al. [36]
FDP route	-	manual
LUM Young's modulus	∞	manual
LUM area	$3mm^2$	manual
LUM route	-	manual

Table 8.1: Simulation parameters. The dashes in the value column indicate that the values are geometric and are shown in Figs. 8.14 and 8.15. (*) Only the total mass is shown.

to obtain anatomically accurate mass properties. However, these values are not critical for the purposes of the qualitative analysis, because the FDP was pulled slowly at $2mm/s$, and the resulting motion is essentially quasistatic.

The joint axis positions and orientations were extracted from the sequence of bone transforms, which were obtained from MoCap or CT video. First, a point cloud was formed from the positions of the centre of mass of the child bone with respect to the parent bone. Then a best fitting plane was fitted onto this point cloud using the singular value decomposition. Finally, a 2D circle fitting algorithm was run on this plane to find the axis of rotation of the joint.

The extracted joint axes are shown in Figure 8.14. The DIP joint of this particu-

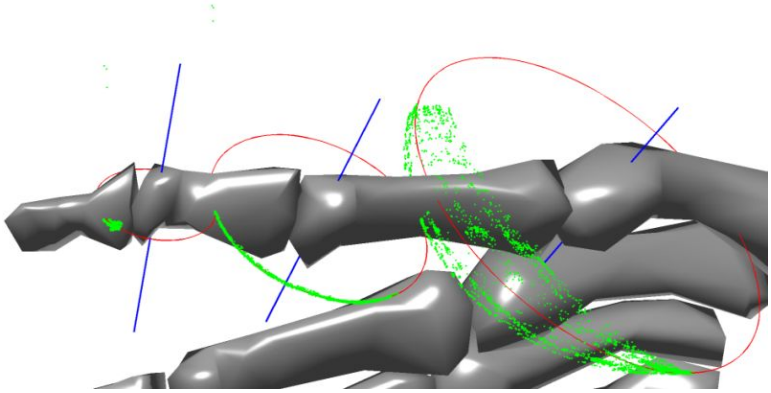
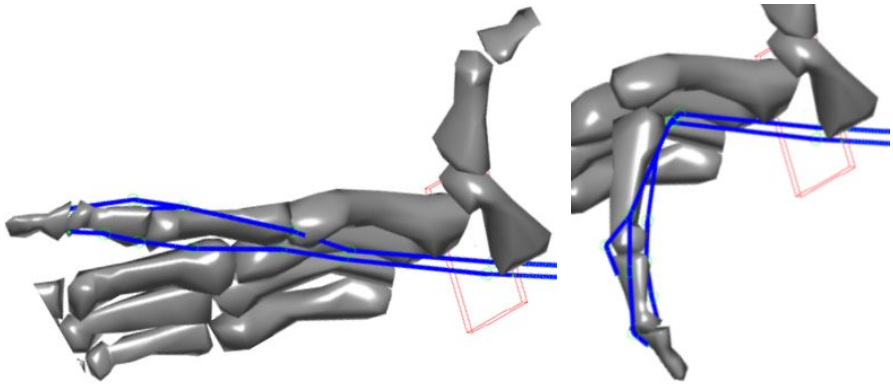


Figure 8.14: Extracted joint angles

lar cadaver specimen was very stiff, resulting in a very small range of motion. This necessitated manual adjustment of the joint axis, since the circle fitting algorithm failed to find a reasonable fit. Using the robust estimation method of Chang and Pollard [14] would yield better results, and should be investigated further, once we acquire reasonable values for the other manually estimated parameters. Also note that the MCP joint is better approximated as a universal joint, as it is also able to abduct and adduct the finger. For our simulation, however, we approximated this as a hinge joint.

In an intact, healthy hand, the joints are very well lubricated, and are essentially frictionless, when the finger is not fully extended or flexed. However, in the cadaver hand, this was not the case, as shown by the blue lines in all four graphs in Figure 8.11 and in the top graph in Figure 8.13. A small tension was present in the FDP, even when there was no weight on the lumbrical. There is significant joint stiffness near the joint limits, due to bone structure, but this aforementioned joint stiffness was evident away from the joint limits, at around $10^\circ - 50^\circ$ MCP flexion. There are multiple possible sources of this stiffness. First, the supporting



(a) **(b)**

Figure 8.15: Tendon paths in simulation

ligaments of the joints add some stiffness, due to their lengthening during joint movement [17, §8]. Second, even though all tendons other than the FDP are slack, there may still be small tension in them, due to the small weights attached to the wires for slack removal (Figure 8.4). Third, despite the care taken to keep the cadaver watered and well-lubricated, the repeated freezing and thawing may have introduced unnatural stiffness. Indeed, we did notice that the first scan of the day always resulted in higher than average tensions.

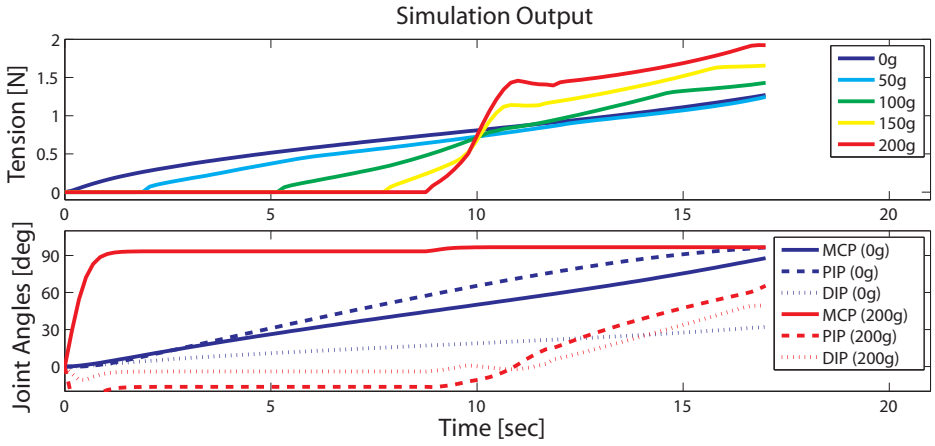
Although many studies have been performed to obtain finger joint stiffness [17, 60], there is a distinctive lack of data on the stiffness of the joints of a *relaxed* finger. Using the terminology of Chao et al. [17], the stiffness of a joint can be approximated by three linear segments. The slope of the line at the low stiffness area away from the joint limits is defined as "neutral stiffness", and the slope at the high stiffness areas near the joint limits is defined as "terminal stiffness". Although they reported the values of the MCP terminal stiffness ($\sim 2Nm/rad$), they did not report the values of the neutral stiffness. Milner and Franklin [60] reported the

stiffness of the three joints when the hand muscles are activated, and remarked that for a relaxed finger, the stiffness should be less than 20% of the activated case. Taking the mean of the reported values, this would result in the stiffness values of 0.76, 0.28, and $0.04 Nm/rad$ for the MCP, PIP, and DIP respectively. These values, however, are over-estimates, since these values were measured when the finger was flexed or extended, and are closer to terminal stiffness than neutral stiffness. Because the neutral stiffness is lower than the terminal stiffness, we chose to use $0.01 Nm/rad$ in all three joints in our simulation. The damping parameters were chosen manually to approximate critical damping for each of the joints. In the cadaver hand, there are multiple possible sources of damping, such as fascia, ligaments, and cartilage. We approximated the net effect of these structures with with damped joints, but for a quantitative analysis, these values will need to be carefully obtained from the cadaver.

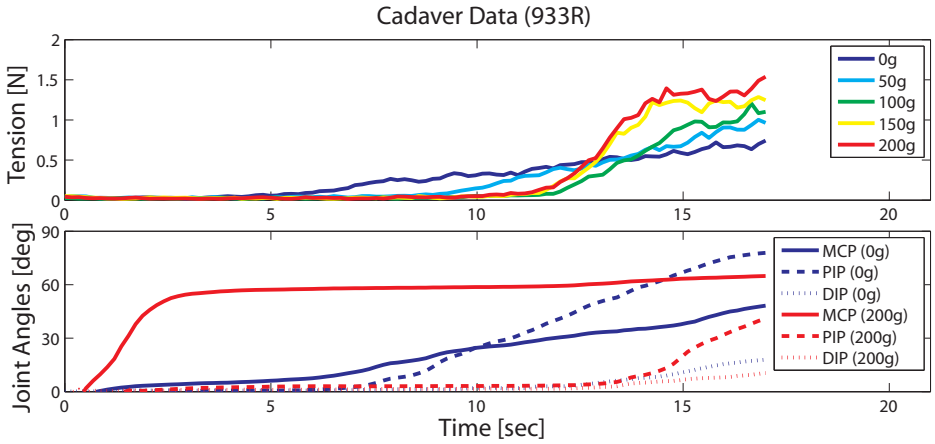
The tensile stiffness and the physiological cross-sectional area of the FDP were taken from Harris et al. [36]. The routing of the FDP was chosen manually following a standard anatomy textbook [63]. The parameters for the lumbrical were not so straightforward to obtain, since its insertion is a hood consisting of many fibres (Figure 8.1). We modelled this hood using two strands, shown in Figure 8.15, and assumed inextensibility.

Using these parameters, we attempted to replicate the experiment with the strand simulator. The resulting tensions and joint angles from the simulation are plotted in Figure 8.16a, along with the data from the experiment in Figure 8.16b for comparison. We were able to qualitatively reproduce the crossing pattern observed in the cadaver data.

One noticeable error was the equilibrium MCP joint angle, which was $\sim 90^\circ$



(a) Simulation output



(b) Data from 993R

Figure 8.16: FDP data from (a) simulation and (b) experiment. The data in (b) is taken from specimen 993R. The joint angles are relative to the neutral, straight position.

in the simulation and $\sim 60^\circ$ in the experiment (red solid lines). Also, the PIP was hyper-extended to around -20° in the simulation, whereas it stayed constant at 0° (red dashed lines) in the experiment.

The most obvious sources of error are the manually chosen parameter values

listed in Table 8.1. Ideally, we want to obtain all of these parameters from the cadaver. (In retrospect, using simpler experimental protocols would be better, such as isolating a single joint and measuring its stiffness and damping across its range of motion.) Although repeating the cadaver experiments is infeasible at this time, we can recover some of the parameters from the data we have already obtained. For example, the routing of the FDP can be solved with an optimization, since we have the FDP excursion and the joint angles data for many sequences. The variables would be the positions of the virtual nodes, and the objective function to be minimized would be the sum of the errors in the FDP length in all sequences using these search variables.

The data obtained through the combined kinematic experiment shown in Figure 8.9 have not been used. The joint motion and tendon tension data from these experiments are more complex and varied than the data from the isolated lumbrical experiments. Although we do not currently plan on simulating the hand using these data, they may be useful for the recovery of some of the manually chosen parameters.

Chapter 9

Conclusions and Future Work

9.1 Summary

This dissertation developed two complementary approaches for understanding the functions of the hand: strand-based musculoskeletal simulation, and acquisition of a rich data set from cadaver hands.

In designing the musculotendon strand simulator, we had two main goals in mind: (1) coupled dynamics of bones, muscles, and tendons, and (2) robust handling of routing constraints. In Chapter 2, we discussed the related work, and the shortcomings of each model with respect to these goals. With line-of-force models, such as OpenSim, the constraints are handled robustly (although they support only simple routing constraints), but the coupled dynamics of musculotendons and bones cannot be simulated, since the musculotendons are abstract massless forces. On the other hand, with dynamic strand simulators from graphics, the coupled dynamics are handled well, but complex routing constraints are difficult to apply.

This led us to design three types of strand simulators, each with its own ad-

vantages and disadvantages. The Cosserat strand (Chapter 4) works well for volumetric muscles, but cannot deal with thin muscles and tendons efficiently. The spline strand (Chapter 5) is very efficient at applying smooth sliding constraints, but suffers from unintuitive force coupling between stretching and bending modes. The reduced strand (Chapter 6) circumvents this problem by adding virtual nodes to add bending degrees of freedom at arbitrary locations along the strand.

In Chapter 7, we described an incremental algorithmic controller that determines the muscle activation levels required for the skeleton to dynamically follow a kinematic trajectory. Our controller has been used successfully for controlling the hand (Section 5.4), the eye (Section 7.1), as well as for the jaw [87].

Obtaining the motion of tendons from a real hand is an ongoing work. In Chapter 8, we showed that we are able to obtain the 3D skeletal trajectory, tendon excursion, tendon tension, and the tendon marker locations during a cadaver experiment. As far as we know, this is the first time that these measurements have been acquired simultaneously. We ended this chapter with a qualitative comparison of the cadaver data and the strand simulator.

9.1.1 Simple Illustrative Examples

We conclude the summary with two simple examples that illustrate the strengths of the strand simulation framework: inclusion of mass and inertia, and the ability to add complex routing constraints.

Mass and Inertia

In a recent article, Pai [70] showed that even for a simple mono-articular limb, "errors in inertia due to [mass] lumping can be quite large and variable, changing

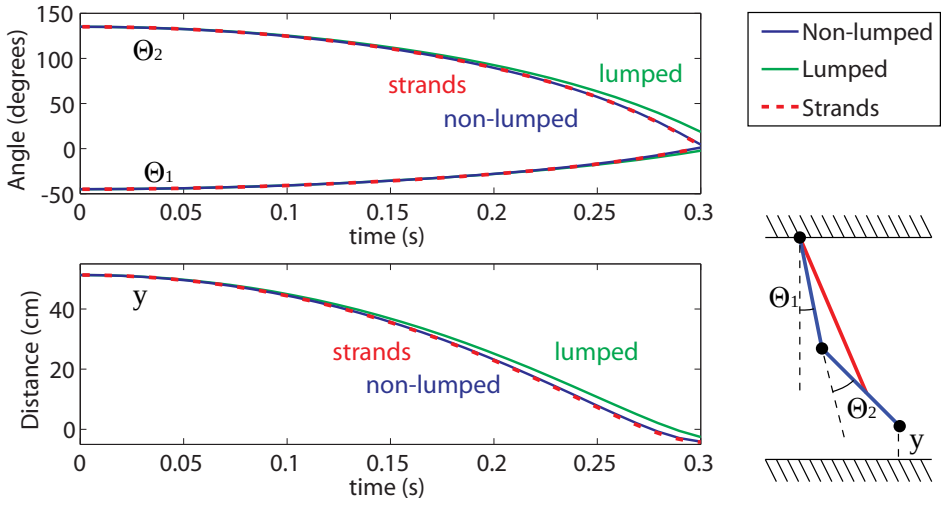


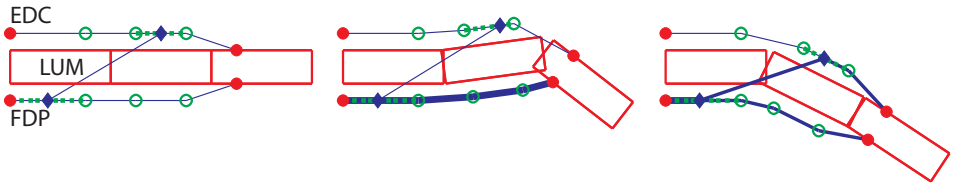
Figure 9.1: Comparison of movement with the data from Pai [70, Fig. 2]. A mono-articular limb was simulated for 300 ms with and without mass lumping. The general strands framework closely matches the output of the simpler model.

with body posture and coupling the velocities of different joints." This was shown with a simple Lagrangian simulator based on joint angles with and without mass-lumping. Here, we show a close match between the result of Pai [70] with our more general simulator. In Figure 9.1, the simulation result of a mono-articular limb with and without mass lumping is shown. The weight and length of the two segments are $4.2kg$ and $43cm$. With mass-lumping, the mass of the muscle, shown in red in the bottom right figure, is lumped to the two bones, shown in blue, at some default pose, and the system mass is assumed to be fixed over time, even though it changes due to the movement of the muscle with respect to the degrees of freedom of the system. Without mass-lumping, the continuous change in the effective mass of the degrees of freedom of the system is taken into account. In the case of Pai [70]'s simulator, the 2×2 effective mass matrix for Θ_1 and Θ_2 is

recomputed at every time step. In the case of the strand simulator, the two bones are modelled as rigid bodies, and the muscle strand is modelled with two rigid nodes, which have zero degrees of freedom; and the 12×12 mass matrix of the two rigid bodies are recomputed at every time step to reflect the movement of the strand with respect to the rigid bodies. In the figure, the limb, starting from the initial pose ($\Theta_1 = -45^\circ, \Theta_2 = 135^\circ$), falls due to gravity. In the top figure, the two joint angles are plotted over time. The solid blue (non-lumped) and green (lumped) curves show the results from Pai [70], and the overlaid, dotted red curve shows the results from with the strand simulator. The strand simulator closely matches the non-lumped model of Pai [70] with the final mean error of 0.7° for Θ_1 and Θ_2, compared to the error of 14° between the lumped and non-lumped cases. Similarly, if the height of the end point (foot) is plotted over time, the strand simulator matches the result from Pai [70] with the final error of $1.4mm$, compared to the error of $29mm$ that would result from mass-lumping. The small discrepancies in the non-lumped models are likely due to the different integrators used; Pai [70] used a higher order integrator (RK45), whereas we used the linearly-implicit Euler integrator.

Complex Routing Constraints

Branching, kinematic loops, and slack are difficult to model using lines-of-force models, but can be modelled naturally with the strands framework. With a simple simulation example of the lumbrical in Figure 9.2, we show how the lumbrical can change the transmission of force of the flexor muscle. The simulation model consists of three rigid bodies and three strands. Both the extensor (EDC) and the flexor (FDP) insert into the distal bone. The lumbrical (LUM) originates from the



(a) All muscles inactive **(b)** FDP active **(c)** FDP and LUM active

Figure 9.2: Simplified lumbrical simulation model, with a kinematic loop. The effect of activating the FDP differs depending on whether the lumbrical is coactivated.

FDP and inserts into the EDC. From the finger pose shown in Figure 9.2a, the simulation is run with the lumbrical inactive (Figure 9.2b), and with the lumbrical coactivated (Figure 9.2c). When the FDP is activated in isolation, the distal joint is flexed, and the proximal joint is slightly hyper-extended. If the lumbrical is coactivated, then the tension in the FDP is distributed to the distal part of EDC, and only the proximal joint flexes.

9.2 Discussion

At its core, the differences in various strand simulators boils down to the choice of representation of the degrees of freedom. For musculotendon strands, we want to choose the degrees of freedom that allow us to express the important deformation modes of muscles and tendon strands, while limiting the total degrees of freedom to a reasonably low number. In general, a 3D curve has three deformation energies—stretching, bending, and twisting. Which of these are important for musculotendon simulation? Depending on the application, surgery simulation for example, a simulator with volumetric primitives with all three deformation energies, such as Cosserat strands, is most suitable. But such volumetric models may

not be necessary for other applications. In order for the muscles to generate torques on the bones through tendons, which of these modes are most used? The obvious answer is the stretching energy. The predominant mode of force generation and transmission in the musculature is through stretching along its axis. This observation manifests itself when the material properties of a tendon is examined. A tendon is sufficiently strong to transmit force axially, but is very flexible in other directions and hardly resists bending or twisting. We therefore chose the nodal formulation that allows us to efficiently model the stretching mode (Chapter 6).

The choice of representation also has its effect on how easily routing constraints can be applied. Many of the aforementioned techniques perform well when the strands are relatively unconstrained (e.g., hair strands fixed at one end and free at the other). However, robust simulation of highly constrained strands is very difficult using these methods. Interactions between the discretization of the strand and constraints on the strand path can result in unexpected force coupling and other unintuitive behaviours. With the nodal formulation and virtual nodes, we were able to model many different types of routing constraints, as shown in the mechanical pulley examples in Section 6.4.

OpenSim [25] and other lines-of-force models offer an attractive alternative to musculoskeletal modelling and simulation. Unlike our strands, which are fully dynamic, OpenSim models the muscles and tendons quasistatically, with no mass and inertia, which results in an inherently more stable and robust simulation. Another advantage with OpenSim is that because the musculotendons are purely kinematic, their paths can be solved kinematically from the configuration of the skeleton. This ability to pose the skeleton and musculotendons kinematically is very useful, especially during the modelling phase. On the other hand, with our simulator, the

strands need to be dynamically simulated whenever the skeletal pose changes.

One disadvantage of using lines-of-force models is the lack of dynamics and mass of the musculotendons [70]. In these models, the mass of the muscle is lumped onto the bones in a canonical skeletal configuration, and are assumed to be fixed during skeletal movement. This may be a reasonable assumption depending on the question being asked. Important biomechanical properties, such as moment arm and isometric forces, are of this type. However, some questions, such as those concerning the force exerted during a walk cycle, or the pursuit and saccade of the eye, are better answered with a dynamic simulator, since in these situations, the errors in inertia due to lumping can be large, and can vary with body posture and joint configuration. One may argue that for the hand, inertia and the effects of dynamics may not be very important. This criticism may have some weight, but we note that in the future, if we want to simulate other parts of the body, such differences will become important.

Another disadvantage is the fact that branching and other routing constraints cannot be modelled naturally. Musculotendons are idealized as abstract forces acting on the bones, and each muscle-tendon pair is represented with a single entity. For instance, the Extensor Digitorum Communis (EDC) of the four fingers must be modelled with four separate musculotendons, even though the muscles of the EDC are fused together in the forearm. With the strands framework, the fused muscles can be modelled with one strand, which forks into four separate tendon strands for each of the fingers.

9.3 Future Work

There are still some outstanding issues to be resolved in both the simulator and data acquisition, which we describe below.

For the simulator, the Surface Node, a 2D extension of the Curve Node, would be an essential ingredient for properly modelling the extensor mechanism of the finger. A better integrator should also be investigated. Since the simulator is not real-time, variable time-stepping schemes should work quite well. Support for other file formats, such as the OpenSim format, must be added, to enable fair comparison with other simulators. A new software implementation in a faster language could be beneficial for both the development and application of the simulator. Finally, a robust modelling interface that allows biomechanists and other researchers to build models for simulation is desirable.

The strand currently supports only the stretching energy. An interesting research direction would be to bind three or more strands together to form an elastic solid with support for both bending and twisting energies. Such an approach could be useful for modelling anisotropic solids with predefined fibre directions.

From the cadaver data, we plan to recover more of the parameters listed in Table 8.1 that were obtained manually. But first, it is important to note the coupling between the simulator and parameter estimation. The relative importance of these parameters depends on the nature of the simulator and the choices that were made in the design of the simulator, because sensitivity to variation in the parameters is dependent on the simulator itself. Also, since a simulator is useful only if the parameters can be obtained or approximated from the real world, the ease of parameter acquisition should be taken into account. Some of the important questions

are

- Tendon network topology modelling. How reasonable is the assumption that the various sheet-like structures can be well modelled by 1D strands? Does shearing of the tendinous hood come into play? The connectivity of the strand network is more important than the geometry of the network [96].
- Viscoelasticity of the musculotendons. Is there hysteresis? Are there frictional effects? Are volumetric forces important, especially for the intrinsic muscles?
- Joint stiffness. Is the stiffness position-dependent? How much does the elasticity of the skin affect the joint stiffness? Can the joint supporting ligaments Chao et al. [17] be modelled effectively with strands?

9.4 Conclusion

We have taken large strides in the development of both the simulation framework and data acquisition protocol for the hand. The proposed strand simulator robustly models the coupled dynamics of bones, muscles, and tendons with various types of routing constraints. The preliminary hand simulation uses only a small subset of the data acquired with the cadaver imaging apparatus; there is still a wealth of information to be extracted and analyzed. In the future, with the strand simulation of subject specific models, we hope to shed further light on how we move and control the hand, and ultimately the rest of the body.

Bibliography

[1] I. Albrecht, J. Haber, and H.-P. Seidel. Construction and animation of anatomically based human hand models. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 98–109, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association. ISBN 1-58113-659-5. → pages 11

[2] H. C. Andersen. Rattle: A "velocity" version of the shake algorithm for molecular dynamics calculations. *Journal of Computational Physics*, 52(1): 24 – 34, 1983. ISSN 0021-9991. doi:DOI:10.1016/0021-9991(83)90014-1. URL http://www.sciencedirect.com/science/article/B6WHY-4DD1MVH-6R/2/e7d2ac9700e994cee6eb0cbc18698e5d. → pages 35

[3] U. M. Ascher and E. Boxerman. On the modified conjugate gradient method in cloth simulation. *The Visual Computer*, 19:526–531, 2003. ISSN 0178-2789. URL http://dx.doi.org/10.1007/s00371-003-0220-4. 10.1007/s00371-003-0220-4. → pages 35

[4] A. Aubel and D. Thalmann. Interactive modeling of the human musculature. In *Computer Animation*, pages 167–255, 2001. → pages 11

[5] D. Baraff and A. Witkin. Large steps in cloth simulation. *ACM SIGGRAPH '98*, 32(Annual Conference Series):43–54, 1998. → pages 35, 46

[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. → pages 44

[7] M. Bergou, M. Wardetzky, S. Robinson, B. Audoly, and E. Grinspun. Discrete elastic rods. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 27(3), 2008. doi:http://doi.acm.org/10.1145/1399504.1360662. → pages 4, 6, 13, 17

[8] M. Bergou, B. Audoly, E. Vouga, M. Wardetzky, and E. Grinspun. Discrete viscous threads. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 28(3), 2010. → pages 2, 13

[9] F. Bertails. Linear time super-helices. *Comput. Graph. Forum*, 28(2): 417–426, 2009. → pages 2, 4, 14

[10] 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 Trans. Graph. (Proc. SIGGRAPH)*, 25(3):1180–1187, 2006. → pages 4, 14, 17

[11] 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. → pages 4, 10, 17

[12] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004. → pages 23

[13] cgCharacter, 2006. URL http://cgCharacter.com. → pages 37

[14] L. Y. Chang and N. Pollard. Robust estimation of dominant axis of rotation. *Journal of Biomechanics*, 40(1):2707 – 2715, 2007. → pages 116

[15] L. Y. Chang and N. Pollard. Method for determining kinematic parameters of the in vivo thumb carpometacarpal joint. *IEEE Transactions on Biomedical Engineering*, 55(1):1897–1906, July 2008. → pages 17, 99

[16] E. Y. Chao, P. Barrance, E. Genda, N. Iwasaki, S. Kato, and A. Faust. *Medicine Meets Virtual Reality: Global Healthcare Grid*, chapter Virtual reality techniques in orthopaedic research and practice, pages 107–114. IOS Press, Amsterdam, Netherlands, 1997. → pages 1

[17] E. Y. S. Chao, K.-N. An, W. P. C. III, and R. L. Linscheid. *Biomechanics of the Hand*. World Scientific, 1989. → pages 17, 99, 117, 129

[18] D. T. Chen and D. Zeltzer. Pump it up: computer animation of a biomechanically based model of muscle using the finite element method. In *ACM SIGGRAPH '92*, pages 89–98, 1992. ISBN 0-89791-479-1. doi:http://doi.acm.org/10.1145/133994.134016. → pages 4, 10, 17

[19] M. B. Cline and D. K. Pai. Post-stabilization for rigid body simulation with contact and constraints. *IEEE International Conference on Robotics and Automation (Proceedings of ICRA 2003)*, 3:3744–3751 vol.3, 2003. ISSN 1050-4729. doi:10.1109/ROBOT.2003.1242171. → pages 19, 35, 46

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

[21] R. D. Crowninshield and R. A. Brand. A physiologically based criterion of muscle force prediction in locomotion. *Journal of Biomechanics*, 14(11): 793 – 801, 1981. ISSN 0021-9290. doi:DOI:10.1016/0021-9290(81)90035-X. → pages 93

[22] T. A. Davis. *Direct Methods for Sparse Linear Systems*. SIAM Book Series on the Fundamentals of Algorithms. SIAM, 2006. ISBN ISBN-13: 978-0-898716-13-9. → pages 36

[23] R. Davoodi, I. E. Brown, and G. E. Loeb. Advanced modeling environment for developing and testing FES control systems. *Medical Engineering & Physics*, 25(1):3–9, Jan 2003. → pages 1

[24] 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. → pages 2, 9, 31, 42

[25] S. L. Delp, F. C. Anderson, A. S. Arnold, P. Loan, A. Habib, T. John, E. Guendelman, and D. G. Thelen. Opensim: Open-source software to create and analyze dynamic simulations of movement, 2007. → pages 4, 9, 17, 126

[26] J. L. Demer. The orbital pulley system: A revolution in concepts of orbital anatomy. *Annals of the New York Academy of Sciences*, 956:17–32, 2002. → pages 1

[27] A. D. Deshpande, R. Balasubramanian, R. Lin, B. T. Dellon, and Y. Matsuoka. Understanding variable moment arms for the index finger mcp joints through the act hand. In *The IEEE/RAS-EMBS International Conference on Biomedical Robotics and Biomechatronics*, Scottsdale, AZ, 2008. → pages 2, 12

[28] M. Epstein and W. Herzog. *Theoretical Models of Skeletal Muscle*. John Wiley and Sibs, Chichester, 1998. → pages 27, 29

[29] P. Faloutsos, M. van de Panne, and D. Terzopoulos. Composable controllers for physics-based character animation. In *ACM SIGGRAPH '01*, pages 251–260, 2001. URL citeseer.ist.psu.edu/faloutsos01composable.html. → pages 15

[30] R. Featherstone. *Robot Dynamics Algorithms*. Springer, 1 edition, 1987. → pages 14

[31] B. Garner and M. Pandy. The obstacle-set method for representing muscle paths in musculoskeletal models. *Comput Methods Biomech Biomed Engin*, 3(1):1–30, 2000. → pages 31, 42

[32] S. Giannini, R. Buda, F. Di Caprio, P. Agati, A. Bigi, V. De Pasquale, and A. Ruggeri. Effects of freezing on the biomechanical and structural properties of human posterior tibial tendons. *International Orthopaedics*, 32:145–151, 2008. ISSN 0341-2695. 10.1007/s00264-006-0297-2. → pages 105

[33] B. Gilles and N. Magnenat-Thalmann. Musculoskeletal mri segmentation using multi-resolution simplex meshes with medial representations. *Medical Image Analysis*, 14(3):291 – 302, 2010. ISSN 1361-8415. doi:DOI:10.1016/j.media.2010.01.006. URL http://www.sciencedirect.com/science/article/B6W6Y-4YH56GD-1/2/cfdc9a58c2339cdd61ffb0750a315961. → pages 36, 110

[34] B. Gilles and D. K. Pai. Fast musculoskeletal registration based on shape matching. In *MICCAI '08: Proceedings of the 11th International Conference on Medical Image Computing and Computer-Assisted Intervention, Part II*, pages 822–829, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-85989-5. doi:http://dx.doi.org/10.1007/978-3-540-85990-1_99. → pages 110

[35] S. Hadap. Oriented strands: dynamics of stiff multi-body system. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2006. → pages 2, 4, 13, 17

[36] E. Harris, L. Walker, and B. Bass. Stress-strain studies in cadaveric human tendon and an anomaly in the young's modulus thereof. *Medical and Biological Engineering and Computing*, 4:253–259, 1966. ISSN 0140-0118. URL http://dx.doi.org/10.1007/BF02474798. 10.1007/BF02474798. → pages 115, 118

[37] W. Herzog. Individual muscle force estimations using a non-linear optimal design. *Journal of Neuroscience Methods*, 21(2-4):167 – 179, 1987. ISSN 0165-0270. doi:DOI:10.1016/0165-0270(87)90114-2. → pages 92

[38] J. R. Hutchinson, F. Anderson, S. Blemker, and S. Delp. Analysis of hindlimb muscle moment arms in Tyrannosaurus rex using a

three-dimensional musculoskeletal computer model: implications for stance, gait, and speed. *Paleobiology*, 31(4):676–701, 2005. → pages 1

[39] S. Jacobsen, J. Wood, D. Knutti, and K. Biggers. The UTAH/M.I.T. Dextrous Hand: Work in Progress. *The International Journal of Robotics Research*, 3(4):21–50, 1984. doi:10.1177/027836498400300402. URL http://ijr.sagepub.com/content/3/4/21.abstract. → pages 12

[40] D. L. James and D. K. Pai. BD-tree: output-sensitive collision detection for reduced deformable models. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 23: 393–398, 2004. doi:http://doi.acm.org/10.1145/1186562.1015735. → pages 32

[41] E. R. Johnson, K. Morris, and T. D. Murphey. A variational approach to strand-based modeling of the human hand. In *WAFR*, pages 151–166, 2008. → pages 2, 9

[42] J. M. Kaldor, D. L. James, and S. Marschner. Simulating knitted cloth at the yarn level. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 27(3):1–9, 2008. doi:http://doi.acm.org/10.1145/1399504.1360664. → pages 4, 14

[43] J. M. Kaldor, D. L. James, and S. Marschner. Efficient yarn-based cloth with adaptive contact linearization. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 29(3), 2010. doi:http://doi.acm.org/10.1145/1399504.1360664. → pages 13

[44] D. M. Kaufman, S. Sueda, D. L. James, and D. K. Pai. Staggered projections for frictional contact in multibody systems. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)*, 27(5), 2008. ISSN 0730-0301. → pages 19

[45] K. R. Kaufman, D. A. Morrow, G. M. Odegard, T. L. H. Donahue, P. J. Cottler, S. Ward, and R. Lieber. 3d model of skeletal muscle to predict intramuscular pressure. In *American Society of Biomechanics Annual Conference*, 2010. → pages 4, 10, 17

[46] P. G. Kry and D. K. Pai. Interaction capture and synthesis. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 25(3):872–880, 2006. → pages 15

[47] Y. Kurita, Y. Ono, A. Ikeda, and T. Ogasawara. Human-sized anthropomorphic robot hand with detachable mechanism at the wrist. In *Proceedings of 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems (IEEE IROS 2009)*, pages 2271–2276, 2009. → pages 12

[48] E. Larsen, S. Gottschalk, M. C. Lin, and D. Manocha. Fast proximity queries with swept sphere volumes. In *Int. Conf. on Robotics and Automation*, 1999. → pages 32

[49] S.-H. Lee and D. Terzopoulos. Heads up!: biomechanical modeling and neuromuscular control of the neck. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 25(3):1188–1198, 2006. ISSN 0730-0301. doi:http://doi.acm.org/10.1145/1141911.1142013. → pages 11, 16

[50] S.-H. Lee, E. Sifakis, and D. Terzopoulos. Comprehensive biomechanical modeling and simulation of the upper body. *ACM Trans. Graph.*, 28(4): 1–17, 2009. ISSN 0730-0301. doi:http://doi.acm.org/10.1145/1559755.1559756. → pages 2, 11

[51] S. W. Lee and D. G. Kamper. Modeling of multiarticular muscles: importance of inclusion of tendon-pulley interactions in the finger. *IEEE Trans Biomed Eng*, 56(9):2253–62, 2009. ISSN 1558-2531. → pages 2, 10

[52] Y. Lee, D. Terzopoulos, and K. Walters. Realistic modeling for facial animation. In *ACM SIGGRAPH '95*, pages 55–62, 1995. ISBN 0-89791-701-4. doi:http://doi.acm.org/10.1145/218380.218407. → pages 11

[53] J. Lenoir and S. Fonteneau. Mixing deformable and rigid-body mechanics simulation. In *Computer Graphics International*, pages 327–334, Hersonissos, Crete - Greece, june 16-19 2004. → pages 38

[54] J. Lenoir, L. Grisoni, P. Meseure, Y. Rémion, and C. Chaillou. Smooth constraints for spline variational modeling. In *GRAPHITE 2004*, pages 58–64, 2004. ISBN 1-58113-883-0. doi:http://doi.acm.org/10.1145/988834.988844. → pages 4, 14, 17, 38

[55] F. Linde and H. C. F. Sorensen. The effect of different storage methods on the mechanical properties of trabecular bone. *Journal of Biomechanics*, 26 (10):1249 – 1252, 1993. ISSN 0021-9290. doi:DOI:10.1016/0021-9290(93)90072-M. → pages 105

[56] H. Lipson. A relaxation method for simulating the kinematics of compound nonlinear mechanisms. *ASME Journal of Mechanical Design*, 128:719–728, 2006. → pages 2, 9, 10

[57] S. R. C. Ltd. Shadow robot company ltd. URL http://www.shadowrobot.com. → pages 12

[58] W. Maural, D. Thalmann, P. Hoffmeyer, P. Beylot, P. Gingins, P. Kalra, and N. M. Thalmann. A biomechanical musculoskeletal model of human upper limb for dynamic simulation. In *Proceedings of the 1996 Eurographics Workshop on Computer Animation and Simulation*, pages 121–136, 1996. ISBN 3-211-82885-0. → pages 11

[59] M. P. Mileusnic, I. E. Brown, N. Lan, and G. E. Loeb. Mathematical Models of Proprioceptors. I. Control and Transduction in the Muscle Spindle. *J Neurophysiol*, 96(4):1772–1788, 2006. doi:10.1152/jn.00868.2005. → pages 1

[60] T. E. Milner and D. W. Franklin. Characterization of multijoint finger stiffness: dependence on finger posture and force direction. *IEEE Trans Biomed Eng*, 45(11):1363–75, 1998. ISSN 0018-9294. URL http://www.biomedsearch.com/nih/Characterization-multijoint-finger-stiffness-dependence/9805835.html. → pages 117

[61] D. Mitton, K. Zhao, S. Bertrand, C. Zhao, S. Laporte, C. Yang, K.-N. An, and W. Skalli. 3d reconstruction of the ribs from lateral and frontal x-rays in comparison to 3d ct-scan reconstruction. *Journal of Biomechanics*, 41 (3):706 – 710, 2008. ISSN 0021-9290. doi:DOI:10.1016/j.jbiomech.2007.09.034. URL http://www.sciencedirect.com/science/article/B6T82-4R2H1Y9-2/2/1e1cf114d51287600adff66b2bcd87ec. → pages 17, 99

[62] D. K. Moon, S. L.-Y. Woo, Y. Takakura, M. T. Gabriel, and S. D. Abramowitch. The effects of refreezing on the viscoelastic and tensile properties of ligaments. *Journal of Biomechanics*, 39(6):1153 – 1157, 2006. ISSN 0021-9290. doi:DOI:10.1016/j.jbiomech.2005.02.012. → pages 105

[63] K. L. Moore and A. F. Dalley. *Clinically oriented anatomy*. Lippincott Williams & Wilkins, 4th edition, 1999. → pages 48, 100, 118

[64] D. Morgan. New insights into the behavior of muscle during active lengthening. *Biophhys J.*, 1990. → pages 27

[65] R. M. Murray, Z. Li, and S. S. Sastry. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, 1994. → pages 23, 24

[66] V. Ng-Thow-Hing. *Anatomically-based models for physical and geometric reconstruction of humans and other animals*. PhD thesis, The University of Toronto, 2001. → pages 10, 11

[67] M. Nikanjam, K. Kursa, S. Lehman, L. Lattanza, E. Diao, and D. Rempel. Finger flexor motor control patterns during active flexion: An in vivo tendon force study. *Hum Mov Sci*, 2006. → pages 17, 99

[68] A. Nimbarte, R. Kaz, and Z.-M. Li. Finger joint motion generated by individual extrinsic muscles: A cadaveric study. *Journal of Orthopaedic Surgery and Research*, 3(1):27, 2008. ISSN 1749-799X. doi:10.1186/1749-799X-3-27. URL http://www.josr-online.com/content/3/1/27. → pages 16

[69] D. K. Pai. STRANDS: Interactive simulation of thin solids using Cosserat models. In *Proceedings of Eurographics 2002*, pages 347–352, 2002. → pages 3, 4, 12, 17, 25

[70] D. K. Pai. Muscle mass in musculoskeletal models. *Journal of Biomechanics*, 43(11):2093 – 2098, 2010. ISSN 0021-9290. doi:DOI:10.1016/j.jbiomech.2010.04.004. → pages xiii, 2, 10, 57, 122, 123, 124, 127

[71] D. K. Pai, S. Sueda, and Q. Wei. Fast physically based musculoskeletal simulation. In *ACM SIGGRAPH '05: Sketches*, page 25, New York, NY, USA, 2005. ACM. doi:http://doi.acm.org/10.1145/1187112.1187141. → pages iv, 25, 29

[72] N. S. Pollard and V. B. Zordan. Physically based grasping control from example. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 311–318, 2005. ISBN 1-7695-2270-X. doi:http://doi.acm.org/10.1145/1073368.1073413. → pages 15

[73] 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. → pages 4, 14, 38

[74] 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. → pages 2, 4, 9, 17

[75] Y. Remion, J. Nourrit, and D. Gillard. Dynamic animation of spline like objects. In *Proceedings of the 1999 WSCG Conference*, pages 426–432, 1999. → pages 38, 40

[76] D. A. Robinson. The mechanics of human saccadic eye movement. *Journal of Physiology*, 174:245–264, 1964. → pages 95

[77] F. Scheepers, R. E. Parent, W. E. Carlson, and S. F. May. Anatomy-based modeling of the human musculature. In *ACM SIGGRAPH '97*, pages 163–172, 1997. ISBN 0-89791-896-7. doi:http://doi.acm.org/10.1145/258734.258827. → pages 11

[78] T. W. Sederberg and S. R. Parry. Free-form deformation of solid geometric models. *ACM SIGGRAPH '86*, 20(4):151–160, 1986. ISSN 0097-8930. doi:http://doi.acm.org/10.1145/15886.15903. → pages 32

[79] A. Selle, M. Lentine, and R. Fedkiw. A mass spring model for hair simulation. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 27(3), 2008. doi:http://doi.acm.org/10.1145/1399504.1360663. → pages 4, 13, 17, 54

[80] M. Servin and C. Lacoursiere. Massless cable for real-time simulation. *Computer Graphics Forum*, 26(2):172–184, 2007. → pages 4, 54, 55

[81] M. Servin, C. Lacoursiere, and K. Bodin. Hybrid, multi-resolution wires with massless frictional contacts. *IEEE Transactions on Visualization and Computer Graphics*, 2010. → pages 15, 59, 60, 68

[82] 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*, pages 11–18, 2003. ISBN 1-58113-645-5. doi:http://doi.acm.org/10.1145/641480.641486. → pages 11

[83] K. Siebertz, S. T. Christensen, and J. Rasmussen. Biomechanical car driver models to analyze comfort. *5th International CTI Conference Automotive Seats*, 2005. → pages 1

[84] E. Sifakis, I. Neverov, and R. Fedkiw. Automatic determination of facial muscle activations from sparse motion capture marker data. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 24(3):417–425, 2005. ISSN 0730-0301. doi:http://doi.acm.org/10.1145/1073204.1073208. → pages 11, 16

[85] J. Spillmann and M. Teschner. CoRdE: Cosserat rod elements for the dynamic simulation of one-dimensional elastic objects. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 63–72, 2007. ISBN 978-1-59593-624-4. → pages 13

[86] J. Spillmann and M. Teschner. An adaptive contact model for the robust simulation of knots. *Computer Graphics Forum (Proc. Eurographics)*, 27 (2), 2008. → pages 2, 4, 13, 17

[87] I. Stavness, A. Hannam, J. Lloyd, and S. Fels. Predicting muscle patterns for hemimandibulectomy models. *Computer Methods in Biomechanics and Biomedical Engineering*, 13(4):483–491, 2010. URL http://hct.ece.ubc.ca/publications/pdf/stavness-hannam-2010.pdf. → pages 6, 93, 122

[88] S. Sueda, A. Kaufman, and D. K. Pai. Musculotendon simulation for hand animation. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 27(3), 2008. → pages iv, v, 4, 14, 38, 88

[89] P. Sylvestre and K. E. Cullen. Quantitative analysis of abducens neuron discharge dynamics during saccadic and slow eye movements. *Journal of Neurophysiology*, 82(5):2612–2632, November 1999. → pages 94, 95, 96, 97

[90] J. Teran, S. Blemker, V. Ng-Thow-Hing, and R. Fedkiw. Finite volume methods for the simulation of skeletal muscle. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 68–74, 2003. ISBN 1-58113-659-5. → pages 4, 10, 11, 17

[91] 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. ISSN 1077-2626. doi:http://dx.doi.org/10.1109/TVCG.2005.42. → pages 11

[92] 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(6):1107 – 1115, 2006. ISSN 0021-9290. doi:DOI:10.1016/j.jbiomech.2005.02.010. URL http://www.sciencedirect.com/science/article/B6T82-4GMGWJ8-1/2/42b980426364d21d76cb85cd957a88ce. → pages 16

[93] D. G. Thelen, F. C. Anderson, and S. L. Delp. Generating dynamic simulations of movement using computed muscle control. *Journal of Biomechanics*, 36(3):321 – 328, 2003. ISSN 0021-9290. doi:DOI:10.1016/S0021-9290(02)00432-3. URL http://www.sciencedirect.com/science/article/B6T82-47VS1SH-4/2/743a0c404c5e12a856d6b57a1751b955. → pages 1, 16

[94] D. S. Tordonato. *The effects of freezing on the mechanical properties of articular cartilage*. PhD thesis, Virginia Polytechnic Institute and State University, 2003. → pages 105

[95] W. Tsang, K. Singh, and E. Fiume. Helping hand: an anatomically accurate inverse dynamics solution for unconstrained hand motion. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 319–328, 2005. ISBN 1-7695-2270-X. doi:http://doi.acm.org/10.1145/1073368.1073414. → pages 16

[96] F. Valero-Cuevas, V. Anand, A. Saxena, and H. Lipson. Beyond parameter estimation: Extending biomechanical modeling by the explicit exploration of model topology. *Biomedical Engineering, IEEE Transactions on*, 54 (11):1951 –1964, nov. 2007. ISSN 0018-9294. doi:10.1109/TBME.2007.906494. → pages 129

[97] 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, 2007. ISSN 0018-9294. doi:10.1109/TBME.2006.889200. → pages 1, 2, 9, 16

[98] K. van den Doel, U. M. Ascher, and D. K. Pai. Computed myography: three-dimensional reconstruction of motor functions from surface emg data. *Inverse Problems*, 24(6):065010, 2008. → pages 17, 99

[99] C. A. Van Ee, A. L. Chasse, and B. S. Myers. Quantifying skeletal muscle properties in cadaveric test specimens: Effects of mechanical loading, postmortem time, and freezer storage. *Journal of Biomechanical Engineering*, 122(1):9–14, 2000. doi:10.1115/1.429621. → pages 105

[100] K. Waters. A muscle model for animation three-dimensional facial expression. In *ACM SIGGRAPH '87*, pages 17–24, 1987. ISBN 0-89791-227-6. doi:http://doi.acm.org/10.1145/37401.37405. → pages 11

[101] M. V. Weghe, M. V, E. Weghe, M. Rogers, M. Weissert, Y. Matsuoka, and A. correct Testbed. The act hand: Design of the skeletal structure, 2004. → pages 114

[102] Q. Wei, S. Sueda, and D. K. Pai. Physically-based modeling and simulation of extraocular muscles. *Progress in Biophysics and Molecular Biology*, 103 (2-3):273 – 283, 2010. ISSN 0079-6107. doi:DOI:10.1016/j.pbiomolbio.2010.09.002. Special Issue on Biomechanical Modelling of Soft Tissue Motion. → pages v, 93

[103] 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. → pages 16

[104] J. Wilhelms and A. Van Gelder. Anatomically based modeling. In *ACM SIGGRAPH '97*, pages 173–180, 1997. ISBN 0-89791-896-7. doi:http://doi.acm.org/10.1145/258734.258833. → pages 11

[105] L. A. Wojcik, D. G. Thelen, A. B. Schultz, J. A. Ashton-Miller, and N. B. Alexander. Age and gender differences in peak lower extremity joint torques and ranges of motion used during single-step balance recovery from a forward fall. *Journal of Biomechanics*, 34:67–73, 2000. → pages 1

[106] T. Yanagawa, K. Shelburne, F. Serpas, and M. Pandy. Effect of hamstrings muscle action on stability of the ACL-deficient knee in isokinetic extension exercise. *Clinical Biomechanics*, 17:705–712, 2002. → pages 1

[107] S.-H. Yeo. Analysis on isokinetic force profile of muscle mechanics. Technical report, University of British Columbia, 2009. → pages 3, 27, 41

[108] 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. ISBN 0-7695-2028-6. → pages 15

[109] F. Zajac. Muscle and tendon: properties, models, scaling, and application to biomechanics and motor control. *Crit Rev Biomed Eng.*, 17(4):359–411, 1989. → pages 27, 28, 41, 92

[110] 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. doi:10.1111/1467-8659.00274. URL http://www.blackwell-synergy.com/doi/abs/10.1111/1467-8659.00274. → pages 11

[111] V. B. Zordan, B. Celly, B. Chiu, and P. C. DiLorenzo. Breathe easy: model and control of simulated respiration for animation. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 29–37, 2004. ISBN 3-905673-14-2. doi:http://doi.acm.org/10.1145/1028523.1028528. → pages 11

[112] V. B. Zordan, A. Majkowska, B. Chiu, and M. Fast. Dynamic response for motion capture animation. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 24(3): 697–701, 2005. ISSN 0730-0301. doi:http://doi.acm.org/10.1145/1073204.1073249. → pages 15

Appendix A

Virtual Node Velocity Derivation

We want to find the two (discontinuous) velocities of the strand at the virtual node. The two (co-located) representations of the position of the virtual node are reproduced here for reference from Section 6.2.1.

$$\begin{aligned} \boldsymbol{x}(s_a) &= (1-s_a)\boldsymbol{x}_a + s_a\boldsymbol{x}_c, \qquad & s_a &= \frac{s}{\alpha} \\ \boldsymbol{x}(s_b) &= (1-s_b)\boldsymbol{x}_c + s_b\boldsymbol{x}_b, \qquad & s_b &= \frac{s-\alpha}{1-\alpha}. \end{aligned} \tag{A.1}$$

The time-varying quantities here are the nodal positions, $\boldsymbol{x}_a$, $\boldsymbol{x}_b$, and $\boldsymbol{x}_c$, and the relative lengths of the subsegments, α. Using the notation from before, the lengths of the subsegments are $l_a = \|\boldsymbol{x}_c - \boldsymbol{x}_a\|$, $l_b = \|\boldsymbol{x}_b - \boldsymbol{x}_c\|, l = l_a + l_b$, and the unit vectors for the subsegments are $\boldsymbol{a} = (\boldsymbol{x}_c - \boldsymbol{x}_a)/l_a$, $\boldsymbol{b} = (\boldsymbol{x}_b - \boldsymbol{x}_c)/l_b$. Then $\alpha = l_a/l$, and applying the chain and product rules, the velocities are

$$\begin{aligned} \dot{\boldsymbol{x}}(s_a) &= -\dot{s}_a\boldsymbol{x}_a + (1-s_a)\dot{\boldsymbol{x}}_a + \dot{s}_a\boldsymbol{x}_c + s_a\dot{\boldsymbol{x}}_c, \qquad & \dot{s}_a &= -\frac{s\dot{\alpha}}{\alpha^2} \\ \dot{\boldsymbol{x}}(s_b) &= -\dot{s}_b\boldsymbol{x}_c + (1-s_b)\dot{\boldsymbol{x}}_c + \dot{s}_b\boldsymbol{x}_b + s_b\dot{\boldsymbol{x}}_b, \qquad & \dot{s}_b &= -\frac{(1-s)\dot{\alpha}}{(1-\alpha)^2}. \end{aligned} \tag{A.2}$$

Since we want the material velocity at the virtual node, we let $s = \alpha$, and so $s_a = 1$ and $s_b = 0$, and

$$\begin{aligned} \dot{\boldsymbol{x}}_{c-} &= \dot{s}_a(\boldsymbol{x}_c - \boldsymbol{x}_a) + \dot{\boldsymbol{x}}_c \\ \dot{\boldsymbol{x}}_{c+} &= \dot{s}_b(\boldsymbol{x}_b - \boldsymbol{x}_c) + \dot{\boldsymbol{x}}_c. \end{aligned} \tag{A.3}$$

Substituting the definition of $\boldsymbol{a}, l_a, \boldsymbol{b}, l_b$, and $\dot{s}_a, \dot{s}_b$, and rearranging, we get

$$\begin{aligned} \dot{\boldsymbol{x}}_{c-} &= -\dot{\alpha}\, l\, \boldsymbol{a} + \dot{\boldsymbol{x}}_c \\ \dot{\boldsymbol{x}}_{c+} &= -\dot{\alpha}\, l\, \boldsymbol{b} + \dot{\boldsymbol{x}}_c. \end{aligned} \tag{A.4}$$

The only remaining task is to derive $\dot{\alpha}$. To do so, we must first derive $\dot{l}_a$ and $\dot{l}_b$. Let $\boldsymbol{d}_a = \boldsymbol{x}_c - \boldsymbol{x}_a$ and $\boldsymbol{d}_b = \boldsymbol{x}_b - \boldsymbol{x}_c$. Then

$$\dot{l}_a = \frac{d\|\boldsymbol{d}_a\|}{dt} = \frac{\partial\|\boldsymbol{d}_a\|}{\partial \boldsymbol{d}_a}\frac{d\boldsymbol{d}_a}{dt} = \boldsymbol{a}^T(\dot{\boldsymbol{x}}_c - \dot{\boldsymbol{x}}_a). \tag{A.5}$$

Here, we used the identity $\frac{\partial\|\boldsymbol{v}\|}{\partial \boldsymbol{v}} = \frac{\boldsymbol{v}^T}{\|\boldsymbol{v}\|}$. Similarly,

$$\begin{aligned} \dot{l}_b &= \boldsymbol{b}^T(\dot{\boldsymbol{x}}_b - \dot{\boldsymbol{x}}_c) \\ \dot{l} &= \boldsymbol{a}^T(\dot{\boldsymbol{x}}_c - \dot{\boldsymbol{x}}_a) + \boldsymbol{b}^T(\dot{\boldsymbol{x}}_b - \dot{\boldsymbol{x}}_c). \end{aligned} \tag{A.6}$$

And finally, using the quotient rule,

$$\dot{\alpha} = \frac{\dot{l}_a l - l_a \dot{l}}{l^2} = \frac{l_b}{l^2}\boldsymbol{a}^T(\dot{\boldsymbol{x}}_c - \dot{\boldsymbol{x}}_a) - \frac{l_a}{l^2}\boldsymbol{b}^T(\dot{\boldsymbol{x}}_b - \dot{\boldsymbol{x}}_c). \tag{A.7}$$

Plugging in this expression into Equation A.4 gives Equation 6.12.

Appendix B

Quadratic Velocity Vector Derivation

The quadratic velocity vector is composed of two terms, $\mathsf{f}_q = \partial T/\partial \mathsf{q} - \dot{\mathsf{M}}\dot{\mathsf{q}}$. We will look at these terms separately in reverse order.

2nd Term: $\dot{\mathsf{M}}\dot{\mathsf{q}}$

If there are no virtual nodes, then the mass matrix is

$$\mathsf{M} = \frac{m}{6}\begin{pmatrix} 2\mathsf{J}_a^T\mathsf{J}_a & \mathsf{J}_a^T\mathsf{J}_b \\ \mathsf{J}_b^T\mathsf{J}_a & 2\mathsf{J}_b^T\mathsf{J}_b \end{pmatrix}, \tag{B.1}$$

where J_i is the material Jacobian matrix returned from the `getJac()` method of the i^{th} node. Then using the product rule, the $(i,j)^{th}$ term of $\dot{\mathsf{M}}$ is $\dot{\mathsf{J}}_i\mathsf{J}_j + \mathsf{J}_i\dot{\mathsf{J}}_j$, where $\dot{\mathsf{J}}_i$ is the matrix returned from `getJacT()`. If there is a virtual node, then the

mass matrix is

$$\mathsf{M} = \begin{pmatrix} \mathsf{J}_a^T M_{aa} \mathsf{J}_a & \mathsf{J}_a^T M_{ab} \mathsf{J}_b & \mathsf{J}_a^T M_{ac} \mathsf{J}_c \\ \mathsf{J}_b^T M_{ba} \mathsf{J}_a & \mathsf{J}_b^T M_{bb} \mathsf{J}_b & \mathsf{J}_b^T M_{bc} \mathsf{J}_c \\ \mathsf{J}_c^T M_{ca} \mathsf{J}_a & \mathsf{J}_c^T M_{cb} \mathsf{J}_b & \mathsf{J}_c^T M_{cc} \mathsf{J}_c \end{pmatrix}. \tag{B.2}$$

Using the product rule again, the $(i,j)^{th}$ term of $\dot{\mathsf{M}}$ is $\dot{\mathsf{J}}_i M_{ij} \mathsf{J}_j + \mathsf{J}_i \dot{M}_{ij} \mathsf{J}_j + \mathsf{J}_i M_{ij} \dot{\mathsf{J}}_j$. Each mass term is composed of $C_{a-}, C_{b-}, C_{c-}, C_{a+}, C_{b+}, C_{c+}$ (Equation 6.22), which in turn are composed of $\alpha, \beta, \boldsymbol{a}, \boldsymbol{b}$ (Equation 6.23). So to derive $\dot{\mathsf{M}}$, we need $\dot{\alpha}, \dot{\beta}, \dot{\boldsymbol{a}}, \dot{\boldsymbol{b}}$.

Letting $\boldsymbol{d}_a = \boldsymbol{x}_c - \boldsymbol{x}_a$ denote the subsegment vector, $\boldsymbol{a} = \boldsymbol{d}_a / \|\boldsymbol{d}_a\|$ is the normalized vector, and so

$$\dot{\boldsymbol{a}} = \frac{d\boldsymbol{a}}{dt} = \frac{\partial \boldsymbol{a}}{\partial \boldsymbol{d}_a} \dot{\boldsymbol{d}}_a = \frac{1}{l_a} (I - \boldsymbol{a}\boldsymbol{a}^T)(\dot{\boldsymbol{x}}_c - \dot{\boldsymbol{x}}_a), \tag{B.3}$$

where we used the identity

$$\frac{\partial \|\boldsymbol{v}\|}{\partial \boldsymbol{v}} = \frac{\boldsymbol{v}^T \boldsymbol{v} I - \boldsymbol{v}\boldsymbol{v}^T}{\|\boldsymbol{v}\|^3}. \tag{B.4}$$

Similarly,

$$\dot{\boldsymbol{b}} = \frac{1}{l_b} (I - \boldsymbol{b}\boldsymbol{b}^T)(\dot{\boldsymbol{x}}_b - \dot{\boldsymbol{x}}_c). \tag{B.5}$$

We already have $\dot{\alpha}$ from Equation A.7, and $\dot{\beta} = -\dot{\alpha}$. From these quantities, we can construct $\dot{M}_{ij}$.

1st Term: $\partial T/\partial \mathsf{q}$

To derive the other quadratic velocity vector term, we first note that the kinetic energy can be divided up into the following terms, involving $\alpha, \boldsymbol{a}$, and $\boldsymbol{b}$.

$$T_{ij} = \frac{1}{2}\dot{\boldsymbol{x}}_i^T \left(\alpha_{00} I + \alpha_{aa}\boldsymbol{a}\boldsymbol{a}^T + \alpha_{ab}\boldsymbol{a}\boldsymbol{b}^T + \alpha_{ba}\boldsymbol{b}\boldsymbol{a}^T + \alpha_{bb}\boldsymbol{b}\boldsymbol{b}^T\right)\dot{\boldsymbol{x}}_j, \tag{B.6}$$

where each α_{ij} is a polynomial (up to cubic) function of α. For example, from Equation 6.22, the α_{ij} functions for M_{aa} are

$$\alpha_{00} = 2\alpha, \qquad \alpha_{aa} = 2(1-\alpha), \qquad \alpha_{ab} = \alpha_{ba} = \alpha_{bb} = 0. \tag{B.7}$$

The quadratic velocity vector term, $\partial T/\partial \mathsf{q}$, can then be obtained by taking the derivative of each T_{ij} with respect to q. And this involves evaluating the following: $\partial \boldsymbol{a}/\partial \mathsf{q}, \partial \boldsymbol{b}/\partial \mathsf{q}, \partial \alpha/\partial \mathsf{q}$, and $\partial \dot{\boldsymbol{x}}/\partial \mathsf{q}$.

Letting $\boldsymbol{d}_a = \boldsymbol{x}_c - \boldsymbol{x}_a$ denote the subsegment vector, $\boldsymbol{a} = \boldsymbol{d}_a/\|\boldsymbol{d}_a\|$ is the normalized vector, and so

$$\frac{\partial \boldsymbol{a}}{\partial \mathsf{q}} = \frac{\partial \boldsymbol{a}}{\partial \boldsymbol{d}_a}\frac{\partial \boldsymbol{d}_a}{\partial \mathsf{q}} = \frac{1}{l_a}(I - \boldsymbol{a}\boldsymbol{a}^T)\begin{pmatrix} -\mathsf{J}_a & 0 & \mathsf{J}_c \end{pmatrix}. \tag{B.8}$$

Similarly,

$$\frac{\partial \boldsymbol{b}}{\partial \mathsf{q}} = \frac{1}{l_b}(I - \boldsymbol{b}\boldsymbol{b}^T)\begin{pmatrix} 0 & \mathsf{J}_b & -\mathsf{J}_c \end{pmatrix}. \tag{B.9}$$

To get $\partial\alpha/\partial\mathsf{q}$, we first need $\partial l_a/\partial\mathsf{q}$ and $\partial l_b/\partial\mathsf{q}$.

$$\begin{aligned}
\frac{\partial l_a}{\partial \mathsf{q}} &= \frac{\partial \|\boldsymbol{d}_a\|}{\partial \boldsymbol{d}_a}\frac{\partial \boldsymbol{d}_a}{\partial \mathsf{q}} = \boldsymbol{a}^T \begin{pmatrix} -\mathsf{J}_a & 0 & \mathsf{J}_c \end{pmatrix} \\
\frac{\partial l_b}{\partial \mathsf{q}} &= \frac{\partial \|\boldsymbol{d}_b\|}{\partial \boldsymbol{d}_b}\frac{\partial \boldsymbol{d}_b}{\partial \mathsf{q}} = \boldsymbol{b}^T \begin{pmatrix} 0 & \mathsf{J}_b & -\mathsf{J}_c \end{pmatrix} \\
\frac{\partial \alpha}{\partial \mathsf{q}} &= \frac{1}{l}\begin{pmatrix} -\beta\boldsymbol{a}\mathsf{J}_a & -\alpha\boldsymbol{b}\mathsf{J}_b & (\beta\boldsymbol{a}+\alpha\boldsymbol{b})\mathsf{J}_c \end{pmatrix}
\end{aligned} \tag{B.10}$$

Finally, $\partial\dot{\boldsymbol{x}}/\partial\mathsf{q}$ is obtained from `getJacQ()`.