Real-time Simulation of Hand Anatomy Using Medical Imaging

by

Mianlun Zheng

A Dissertation Presented to the
FACULTY OF THE USC GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(COMPUTER SCIENCE)

December 2024

I dedicate this thesis to my parents and brother for their love and encouragement;
to my husband Yuanzhe and our dog Arrow, for their steadfast love and support;
to my friends and intern managers for their friendship and guidance;
to my labmates for their assistance and thoughtful paper discussions;
to my committee members for reading and improving the thesis;
and to my supervisor, Prof. Jernej Barbič, for his invaluable inspiration, guidance,
and support throughout this journey.

Table of Contents

List of Tables

List of Figures

Abstract

Precision modeling of the hand's internal musculoskeletal anatomy has been largely limited to individual poses, and has not been connected to the continuous volumetric motion of the hand anatomy actuating across the hand's entire range of motion.

This thesis gives a method to simulate the volumetric shape of a hand's musculoskeletal organs to any pose in the hand's range of motion, producing external hand shapes and internal organ shapes that match ground truth optical scans and medical images (MRI) in multiple scanned poses. This part of the thesis was published at ACM SIGGRAPH Asia 2022. We achieve this by combining MRI images in multiple hand poses with FEM multibody nonlinear elastoplastic simulation. The system models bones, muscles, tendons, joint ligaments and fat as separate volumetric organs that mechanically interact through contact and attachments, and whose shape matches medical images (MRI) in the MRI-scanned hand poses. The match to MRI is achieved by incorporating pose-space deformation and plastic strains into the simulation. We show how to do this in a non-intrusive manner that still retains all the simulation benefits, namely the ability to prescribe realistic material properties, generalize to arbitrary poses, preserve volume, and obey contacts and attachments. We use our method to produce volumetric renders of the internal anatomy of the human hand in motion and to compute and render highly realistic hand surface shapes. We evaluate our method by comparing it to optical scans and demonstrate that we qualitatively and quantitatively substantially decrease the error compared to previous work. We test our method on five complex hand sequences, generated either using keyframe animation or performance animation using modern hand tracking techniques.

While the hand anatomy simulation system achieves high anatomical precision, there is also a growing need for techniques that balance this level of realism with the speed required for many real-time applications in computer graphics. The transition from anatomically accurate simulations to real-time mesh deformations introduces a complementary challenge.

Thus, the thesis also gives a hard-real-time multi-resolution mesh shape deformation technique for skeleton-driven soft-body characters, such as the human hand. This part of the thesis was published at ACM SIGGRAPH Asia 2024. Our work targets applications where the multi-resolution shapes must be generated at fast speeds ("hard-real-time", e.g., a few milliseconds at most and preferably under 1 millisecond), as commonly needed in computer games, virtual reality and Metaverse applications. We assume that the character mesh is driven by a skeleton, and that high-quality character shapes are available in a set of training poses originating from a high-quality (slow) rig such as volumetric FEM simulation. Our method combines multi-resolution analysis, mesh partition of unity, and neural networks, to learn the pre-skinning shape deformations in an arbitrary character pose. Combined with linear blend skinning, this makes it possible to reconstruct the training shapes, as well as interpolate and extrapolate them. Crucially, we simultaneously achieve this at hard real-time rates and at multiple mesh resolution levels. Our technique makes it possible to trade deformation quality for memory and computation speed, to accommodate the strict requirements of modern real-time systems. Furthermore, we propose memory layout and code improvements to boost computation speeds. Previous methods for real-time approximations of quality shape deformations did not focus on hard real-time, or did not investigate the multi-resolution aspect of the problem. Compared to a "naive" approach of separately processing each hierarchical level of detail, our method offers a substantial memory reduction as well as computational speedups. It also makes it possible to construct the shape progressively level by level and interrupt the computation at any time, enabling graceful degradation of the deformation detail. We demonstrate our technique on several examples, including a stylized human character, human hands, and an inverse-kinematics-driven quadruped animal.

Chapter 1

Introduction

1.1 Simulation of Hand Anatomy Using Medical Imaging

Modeling and animation of hands has numerous applications in film, computer games, virtual reality, CAD/CAM, medicine, and related fields. Precise modeling of the motion of the internal anatomy of the human hand can help with designing hand prosthetics, better surgical tools, or improve the ergonomics of tools and apparel. Hand anatomical models can also be used to generate high-quality datasets of hand shapes across the range of motion, which deep learning can use to track hand poses and recognize hand activity.

The importance of hand modeling and animation has long been recognized, and there is a large body of work on the topic. Most existing methods, however, focus on the external hand shape and subsume internal anatomy with simplified non-anatomical representations. We give a method to accurately model the motion of the internal musculoskeletal anatomy of the human hand across the entire range of motion of the human hand, and do so in a manner that demonstratedly matches medical imaging (MRI) across multiple hand poses. We model hand bones, tendons, ligaments, muscles, and fat, all as separate volumetric three-dimensional tissues that move and deform as observed in the MRI scans of a real person's hand.

Our method is a hybrid data-driven simulation method that uses data (MRI scans) to steer a Finite Element Method volumetric multibody simulation to be able to "extrapolate" from the MRI scans to the entire range of motion of the hand. The steering is achieved in different novel

ways for the different tissues; for example, tendons are steered through novel sliding constraints defined using pose-space deformation [1] against the MRI data (Section 3.3), and muscles and fat are steered with novel per-organ and per-pose plastic strains, optimized so that the FEM simulation matches the MRI data in the scanned MRI poses (Sections 3.2, 3.5). We limit our muscle modeling to concentric isotonic contraction, i.e., relaxed poses free of (substantial) muscle firings such as those encountered during free hand motion or gentle grasping. This is because the poses in our MRI scans are also relaxed. We are not aware of any method that has simultaneously captured hand MRI data and muscle activation patterns in multiple poses, nor do we attempt to do so in this work. This is still sufficient for free-space hand motion as commonly encountered in many applications, and analysis and simulation of its internal anatomical motion.

Previous volumetric methods have attempted to model the entire hand as a single soft tissue [2, 3], but they ignored the fact that different hand organs have very different mechanical properties. For example, muscles are much stiffer than fat; and are active tissues. A hand tendon behaves like an inextensible rod. In addition, adjacent tissues usually move relative to each other, such as muscles sliding against each other. These effects cannot be captured by merely modeling the entire hand as a single soft tissue. For accurate hand modeling, it is important to mesh and simulate different organs separately. While different hand organs have been widely studied in many research fields independently, to the best of our knowledge, nobody has attempted to model the entire hand as a complete biomechanical volumetric system. Existing volumetric simulation models are not designed for matching medical images such as MRI, whereas our method matches them both qualitatively and quantitatively.

As commonly done in VFX industry, our simulation proceeds in layers [4]. The input to our work is a hand joint hierarchy animation generated using any suitable means, e.g., hand motion capture, or vision-based hand tracking systems. The bones (rigid objects) are then driven kinematically using the joint hierarchy, followed by bone fascia (cloth), and then soft-tissue simulation layers: tendons (rods), ligaments, muscles (elastic solids) and muscle fascia (cloth). Our last layer is the fat tissue (elastic solid), which gives us the external shape of the hand. We use the "cloth"

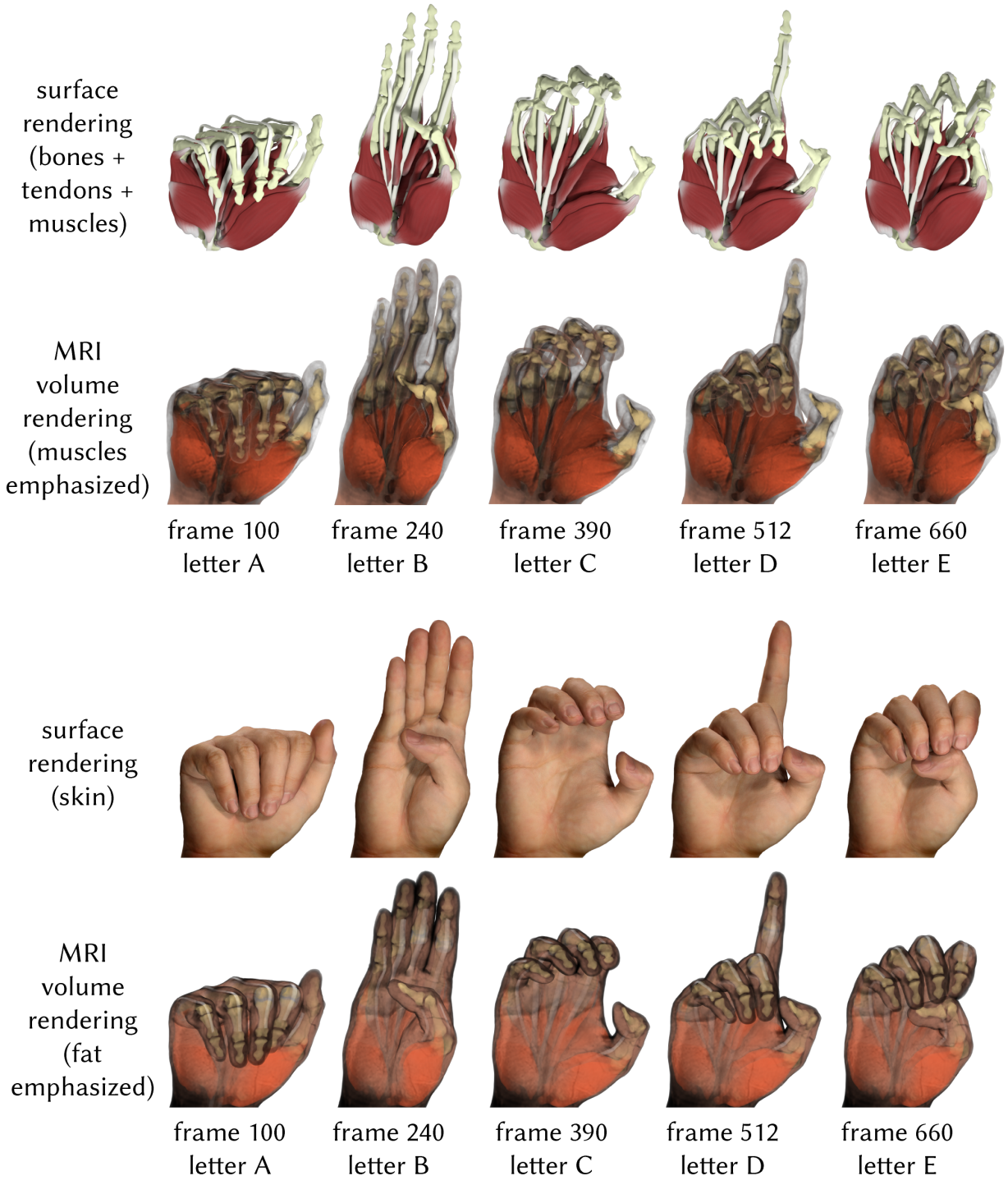


Figure 1.1: **Human hand anatomy animated across the hand's range of motion:** These surface and volumetric renders display complex hand organ volumetric motion computed using our method. The hand was posed into letters A-E of the American Sign Language; these poses were not scanned by MRI, but are computer-simulated.

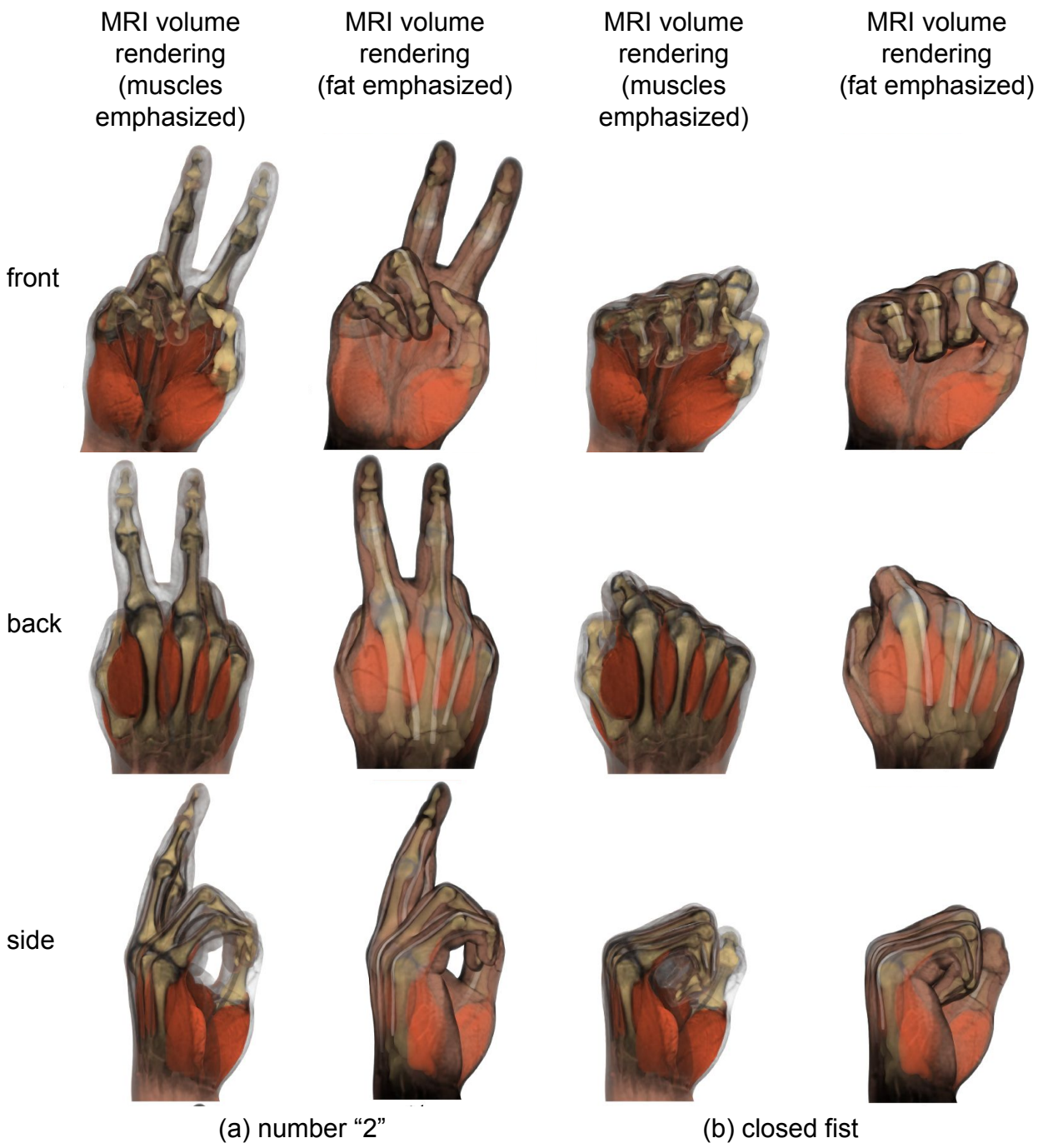


Figure 1.2: **Representative frames for motion “Numbers 1-5” and “Close the fist”.** We display the front, back and the side of the hand. Our method can animate the hand internal musculoskeletal organs to arbitrary poses in the hand’s range of motion, in a manner that matches the medical images in the example poses (Figure 3.30) and even non-example poses (Figure 3.31). In addition to musculoskeletal organs, arteries and veins are also clearly visible in the “fat emphasized” MRI sequence.

terminology because it is common in computer animation; "cloth" here simply means an elastic thin-shell. We employ a layered simulation, as opposed to a coupled simulation of all hand organs, for two reasons: (1) even layered simulation is at the limit of what we can achieve computationally today with state of the art algorithms; coupled simulation is computationally infeasible, and (2) layered simulation is better suited for our goal of the organs matching the MRI scans in example poses. Namely, a fully coupled hand has too many diverse tissues and too complex interdependencies, which makes it intractable to simultaneously match the MRI scans of the different hand organs in multiple poses.

We demonstrate that we can closely match the ground truth medical images in each example pose (Section 3.6, Figure 3.30, Table 3.5): all the organs are volumetrically matching their shapes in the MRI scans, with average errors of less than 1mm in all example poses for skin, as compared to an optical scan. Furthermore, we produce a good match to optical scans even in non-scanned poses (Section 3.6, Figure 3.31). Our volumetric simulation produces realistic organ shapes across the entire range of the hand's motion (ROM), by nonlinearly "interpolating" the shapes of organs seen in the MRI scans. We are not aware of any prior work that has demonstrated volumetric simulation of the entire hand's musculoskeletal system that matches medical image ground truth data, and that stably interpolates to the entire hand's ROM. We use our method to produce volumetric renders of the hand organs under complex nonlinear motion as the hand actuates (Figures 1.1, 1.2). Furthermore, our model qualitatively reproduces important features seen in the photographs of the subject's hand, such as a similar overall organic shape and formation of bulges due to the activated muscles.

1.2 Multi-Resolution Real-Time Deep Pose-Space Deformation

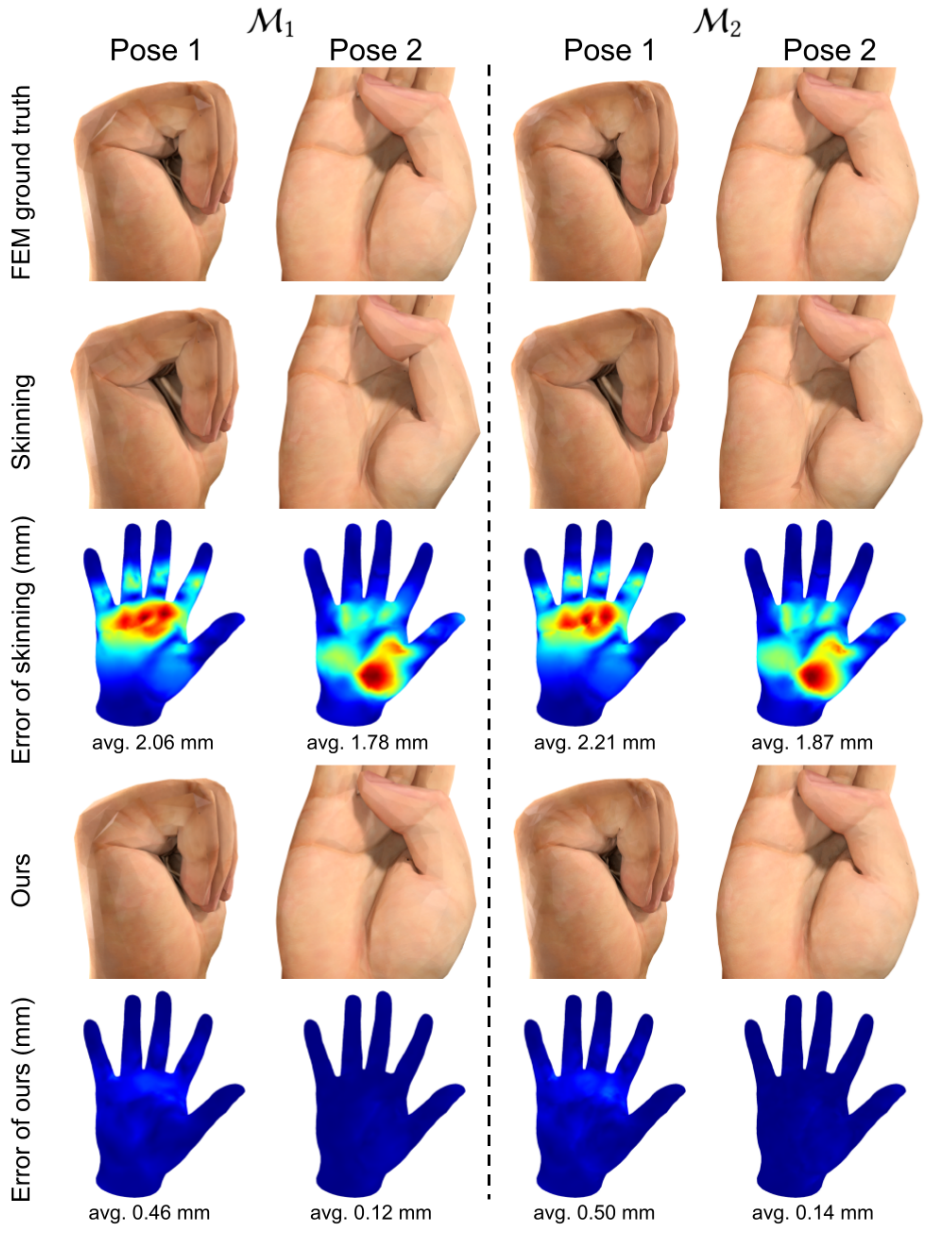
The first study focuses on the intricate simulation of the hand's internal musculoskeletal system, achieving high anatomical precision through FEM-based modeling. However, there is also an increasing demand for techniques that not only ensure this level of realism but also meet the speed

requirements for real-time applications. Transitioning from pose-driven, anatomically accurate simulations to real-time mesh deformations presents a new challenge: generating high-quality deformations across multiple levels of detail while optimizing computational efficiency.

There are many techniques to produce high-quality animated shapes of digital characters, such as production character rigs, FEM soft-tissue simulation, 4D scanning and manual shot-sculpting. However, these techniques are time-consuming and often cannot be performed in real time. In real-time systems, one often animates a 3D mesh based on the motion of the underlying joint hierarchy. This can be done to animate humans, animals, plants and even inanimate objects, and is pervasive in real-time systems such as computer games, virtual reality, virtual production and the Metaverse. The standard algorithm for this task (linear blend skinning (LBS) [5]) is fast and easy to use, but it also produces well-known artifacts. Those can be corrected with several real-time techniques such as dual quaternions [6], "delta mush" [7], helper joints [8] or pose-space deformation [1].

These techniques, however, do not address a critically important component of real-time systems, namely the need for multiple levels of detail. In a typical interactive application, important assets are represented with multiple meshes, with a progressive number of vertices and triangles ("Levels of Detail", LOD). LODs are very common, for example, in popular game engines such as Unreal and Unity. They are necessary to keep the real-time performance sufficiently high, so that a character that is far from the camera can be displayed at a lower resolution. Similarly, if multiple characters are in the foreground, it is useful to display them at a decreased resolution to maintain the required update frame rates. In our work, we give a multi-resolution shape deformation technique that, given any character pose, produces the output shape at any (or several) desired level(s) of detail (Figure 1.3), based on any suitable shape deformation training data; and does so at hard real-time rates.

Our technique is grounded in multi-resolution methods extensively investigated in computer graphics [9]. It is similar in spirit to Pose-Space Deformation [1] and Fast Deep Deformations [10] in that we also generate pose-dependent pre-skinning correctives to the neutral shape using neural networks, to which linear blend skinning is applied to produce the output shape. However, our key


$\mathcal{M}_1$
$\mathcal{M}_2$
Pose 1
Pose 2
Pose 1
Pose 2
FEM ground truth
Skinning
Error of skinning (mm)
avg. 2.06 mm
avg. 1.78 mm
avg. 2.21 mm
avg. 1.87 mm
Ours
Error of ours (mm)
avg. 0.46 mm
avg. 0.12 mm
avg. 0.50 mm
avg. 0.14 mm

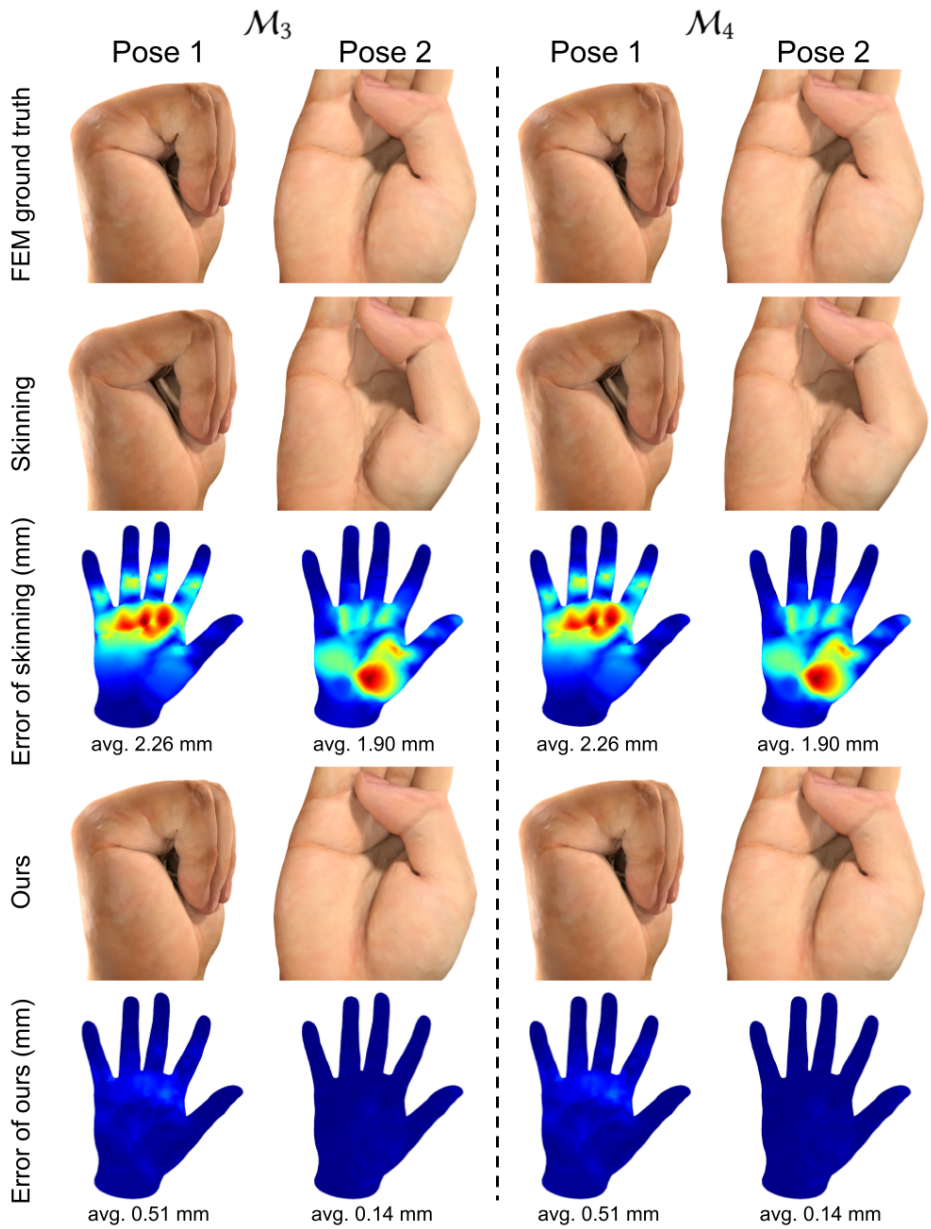


Figure 1.3: Our method permits one to place the hand into any pose in the hand's ROM and rapidly compute its quality shape, and do so at any (or several) desirable discrete mesh resolution levels. The training set consists of 3,607 frames of FEM musculoskeletal simulation exercising the ROM of the hand. We show the ground truth (computed using FEM musculoskeletal simulation; 48 seconds per frame), the output of linear blend skinning (visible artefacts), and our result. We show two representative hand poses that are **not** a part of the training set. The four mesh Levels of Detail $\mathscr{M}_0, \mathscr{M}_1, \mathscr{M}_2, \mathscr{M}_3$ have 1,133, 4,528, 18,105, 72,414 vertices, respectively. Skinning running times for the four levels are 162, 210, 446, 1,156 microseconds per frame, respectively. Our method computes the skinning correctives at the four levels (61, 144, 299, 548 microseconds per frame, respectively), plus performs the same skinning. For a small additional computational cost on top of skinning, we greatly improve the output quality compared to skinning (see images and the error plots), and speedup the computation by 100,000$\times$ compared to FEM simulation. Note that the error does not decrease down to zero on progressive LODs because at each LOD, the FEM training dataset contains new deformation detail only introduced and resolved by the mesh at this LOD.

distinction is that we give an algorithm to do so at multiple LODs, in an interruptible manner, and in a manner whereby output accuracy can be controled. Our first contribution is to define neural networks so that they predict the shape deformation at some LOD and in some spatially localized region, *relative to the shape deformation already determined on coarser levels*. We observe that if this is combined with the mesh prolongation operator, the neural networks at some LOD then only need to resolve the shape deformation detail arising at this resolution level, as opposed to re-creating the entire shape deformation from scratch; this leads to large memory savings. Next, we contribute the observation that the spatially localized regions can be defined automatically using the prolongation operator [11], in a manner intertwined with the mesh simplification algorithm, avoiding tedious manual selection of regions. The different mesh LODs do not need to be subdivisions of each other. We use quadric error mesh simplification [12] to create our mesh LODs, but our technique is suitable for other mesh hierarchies, as long as they permit a definition of prolongation and restriction operators [11], e.g., progressive meshes [13], or mesh subdivision [14].

The LOD hierarchy and all other necessary datastructures are created during pre-processing. Analogous to how the prolongation operator defines the influence of a coarse vertex to the finer level, we use the prolongation operator to define the spatial influence weights for each neural network. Then at runtime, one can select any particular LOD (or several, or even all), and our technique produces quality pre-skinning deformations at those LODs, at hard real-time rates. An example of such an application are computer games, where different characters may be rendered at different LODs only known at runtime, based on character importance, distance to camera and other metrics [15]. They can also be rendered at several LODs at the same frame in case of a multi-player game where multiple cameras observe the same character from different distances; or at two LODs when blending two LODs to avoid popping. Other potential applications include progressive transmission of character deformation over a network, or multi-resolution contact handling with graceful degradation [16].

Our technique is fast; producing the complete LOD hierarchy of correctives for meshes with over 70,000 vertices (Figures 1.3, 1.4) in approximately 1 millisecond when using multiple cores

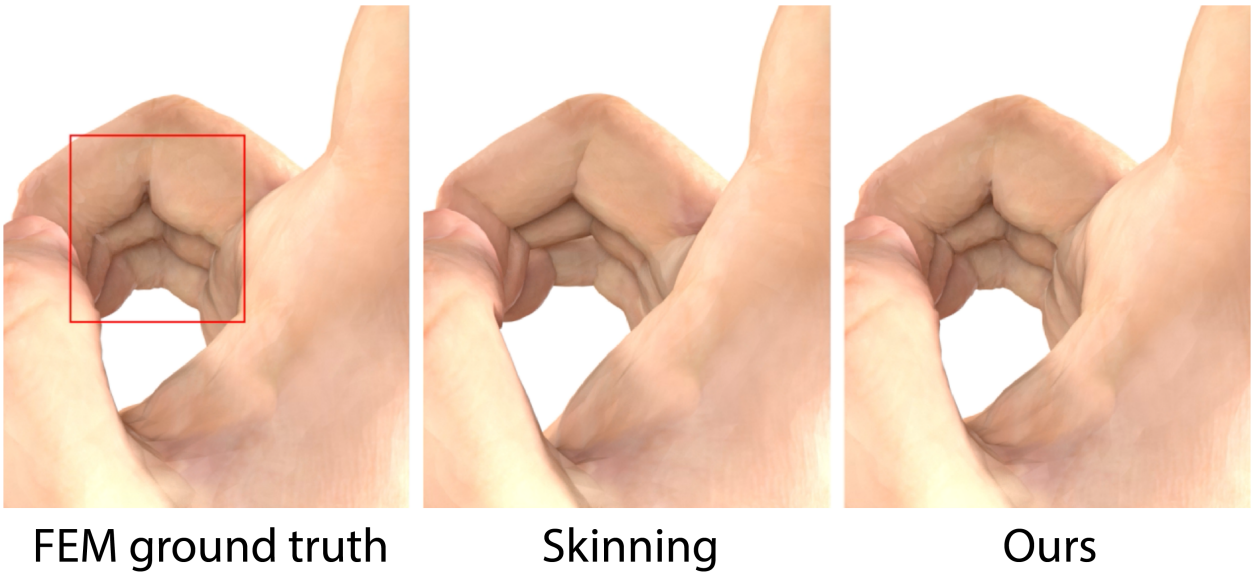


Figure 1.4: **Mesh detail on the hand.** Our method better captures the ground truth than skinning.

(with cold caches), and in a few milliseconds when using a single core. We are not aware of any other work that has investigated how to approximate given character shape training data in hard real time at multiple levels of detail. A naive solution would be to simply treat each LOD as a separate problem and train pre-skinning correctives separately for each mesh LOD, e.g., using Bailey's method [10]. We compare to such a method and demonstrate that our method greatly reduces the memory requirement to store the neural networks and other datastructures needed for runtime evaluation. Note that a small memory footprint is critically important for interactive applications such as games, whereby memory needs to be shared with the other game assets. Our training times are also substantially reduced, thanks to a lesser number of required neural networks. Our method maintains comparable or faster runtime speeds to non-hierarchical methods, depending on how many LODs are required at a particular frame at runtime. These results are possible thanks to the incremental nature of our multi-resolution neural networks that only need to add incremental deformation detail at each LOD. The "world" of real-time computing in the microsecond regime is very different to the more typical real-time computer animation applications with running times in tens of milliseconds or more. For example, for a mesh with 70,000 vertices, it takes 50 microseconds just to write a single already computed mesh shape to memory, let alone do any deformation

computation. Therefore, although our core contribution is algorithmic, we also discuss various code and memory optimizations suitable for such a fast computational regime.

Chapter 2

Related Work

2.1 Data-driven human hand appearance modeling

Human hand is one of the most flexible organs of the human body and plays a critical role in human interaction with other objects. Therefore, it has been intensively studied, not only for medical purposes but also in robotics, virtual reality, games and VFX.

The human hand is biomechanically complex. It consists of 27 bones, 34 muscles and over 100 ligaments and tendons, which are all confined to a small volumetric region. Due to this, most of the existing methods skip the complexity of the anatomy, and focus on the external hand appearance. Therefore, the widely adopted methods are data-driven methods.

The human hand can simply be modeled using skinning [6, 17] or implicit methods [18]. Modeling human hand using skinning often produces artifacts, because the underlying skeleton is imprecise and the skin of the hand does not usually follow such a simple model. As shown in [3], a simple skinning method produces suboptimal results due to the incorrect positions of joint centers.

To fix these issues, Pose-Space Deformation (PSD) has been proposed to incorporate artist-corrected pose shapes [1]. Artists can incrementally add "fixes" to the existing skinning model to avoid artifacts and produce deformation closer to real human hands. Kurihara and Miyata [19] presented a variant of PSD suitable for hand animation, and Rhee et al. [20] demonstrated how to efficiently implement it on a GPU. PSD has been widely used in industry due to its fast performance, simplicity and the ability to incorporate real-world scans and arbitrary artist corrections.

Human hands have also been successfully modeled using a SMPL model [21] or a MANO model [21–23]. To fit such a model, hundreds of 3D hands were captured using a complex optical capturing system [24]. The database is then used for reconstructing personalized human hands from a few images [25]. Compared to these methods that focused on the external appearance of the hand, we tackle the problem of modeling and animating complex internal anatomy, and simultaneously produce high-quality output skin shapes through physically based simulation.

2.2 Data-driven human hand anatomy modeling

Recently, a data-driven method ("NIMBLE") was presented to animate internal hand anatomy [23]; this method is completely data-driven (no simulation). Unlike our work, it is not focused on animation: their video only shows a few seconds of animation, and their hand motion is considerably simpler than ours. Our anatomical model is also significantly richer: NIMBLE has 7 muscles (vs ours 15), and has no ligaments, tendons, fascias, fingernails or fat. Please note that NIMBLE requires labeling segmentation masks on each MRI slice, which is a substantial manual effort. Quantitatively, NIMBLE average skin error is 1.5mm [Figure 9 in their paper] (ours: 0.5mm; Table 3.5). An advantage of our simulation method is that it produces good shapes even in poses away from the data, requiring fewer MRI scans.

We note that physically based simulation has the benefit that, once setup, it can generate the human hand in an *arbitrary* pose; and as such the range of motion of the human hand can be "mined" fully automatically merely with simulation, which has great benefits for machine learning of hand shapes. It is our belief that future Metaverse hand applications will likely employ a combination of physically based simulation and extensive optical scanning. Finally, data-driven methods are complementary to simulation: simulation output can serve as input to SMPL, MANO, NIMBLE, etc.

2.3 Physics-based human hand anatomy modeling

We simulate hands using physically-based modeling, which requires one to first acquire accurate internal anatomy. Anatomical models can be created manually by artists based on the medical literature or medical images, but the process is very time consuming [26], or does not produce anatomy closing matching MRI images [27]. Anatomy transfer has been successfully used for generating new human body anatomy given existing templates [28–30].

However, compared to a human body, a human hand is small yet complicated. Errors of a few millimeters can already cause large artifacts, because some tissues, such as distal phalanx bones or tendons, are merely 1-2mm in diameter. Moreover, existing high-quality human hand anatomy models are rare and used mainly for educational purposes. We examined well-known hand templates [31, 32]; despite their tissues being complete and generally correctly positioned, they are primarily used for anatomy demonstrations. Their organ shapes tend to be small with empty space in between, whereas in MRI images, organs are tightly packed. Therefore, we acquire internal anatomy directly from medical imaging and optical scanning [3]. We choose MRI because it enables extraction of many different soft tissues. We then segment the tissues using existing methods [3, 33].

Many methods have been developed for physically-based modeling of human hands. For modeling the entire hand, the existing approaches entail simulating a single soft tissue mesh constrained to the underlying skeleton [2, 3, 34–39]. Due to modeling all soft tissues as a single volumetric layer, these methods fail to capture many key features of the human hand. Lee et al. [40] modeled the upper human body using anatomically based simulation comprehensively. However, they only modeled human hands kinematically, without simulating the anatomical structure of the hands. In contrast, we model bones, muscles, tendons, ligaments and fat, all resolved as separate volumetric objects.

To the best of our knowledge, there are no good models for simulating hand internal anatomy that demonstratedly match medical imaging across multiple poses. We are also not aware of any methods to model the volumetric three-dimensional motion of internal hand organs in the medical

literature; instead, the focus in medicine is to understand static hand shapes [41], at best with minimal animation as needed to understand how to treat hand disease and injury.

2.3.1 Bones

We treat each tissue of a human hand separately. For bones, Kurihara and Miyata [19] gave a bone rig extracted from multiple CT scans. Prior work also analyzed hand bone motion using MRI [3, 42–45]. Keller et al. [46] inferred the anatomic skeleton of a person from the 3D body surface. We follow the approach from [3], because it gives an artist-friendly rigging system built from MRI while simultaneously closely matching the MRI data.

2.3.2 Muscles

Abdrashitov et al. [47] proposed a new shape representation of musculoskeletal tissues, and an intuitive user interface to help ease the complexity of modeling volumetric anatomy. Angles et al. [48] modeled muscles as a collection of generalized rods with volume conservation. Modi et al. [49] proposed an efficient finite element scheme to simulate bulky muscles with heterogeneous materials. Such models have many parameters to tune and are not designed to precisely match markers delineated in MRI images.

Muscle activation is often modeled by a widely used Hill model [26, 40, 50, 51]. However, extracting muscle fiber directions from MRI is not easily feasible or reliable. It requires imaging techniques such as diffuse tensor imaging, which is prohibitively expensive and rarely available. If not available, it is only possible to approximate fiber fields based on the shape of the muscle [29]. When attempting to match muscle shapes to medical images in multiple poses, these facts make it difficult to apply Hill's model.

Muscle activation can also be modeled using plastic deformation [29, 52], or implicit skinning [53]. For hand muscle activation, we adopt the 6-DOF plastic model of [52] previously proposed for facial muscles. Compared to their method, we modeled each muscle as an individual object. This is because muscles are heavily sliding against each other inside the human hand, and

therefore embedding them into a single mesh causes artifacts. To the best of our knowledge, no existing methods have previously been given to build a muscle system that matches ground truth internal anatomy seen in medical images in multiple example poses.

2.3.3 Tendons

Existing methods for modeling tendons are primarily used for controlling hand articulation, or for medical applications [27, 54–56]. While these methods are sophisticated, the input anatomy must be created manually, and often does not precisely match any real data. Prior work has attempted to extract tendons from medical images such as MRI [57–59] or ultrasound [60], but they only focused on one or a few local regions of the hand in a single pose.

We model tendons as discrete elastic rods, similar to [61] and [62]. However, these previous methods were designed for forward simulation and not solving inverse problems on rods; Bergou et al. [61] used positions as primary variables and inferred orientation (normals) indirectly, whereas Kugelstadt and Schömer [62] used quaternions to represent rotations.

In contrast, we directly use vertex positions *and* segment normals as our primary degrees of freedom, as this better fits into the iterative closest point (ICP) algorithm. We can thus more easily incorporate real MRI tendon data to guide our tendon simulations. For example, our sliding constraints are naturally expressed as positions, skinning can directly use the position and normals to form the local coordinate system at each line segment, and non-rigid registration naturally operates with positions and normals also.

2.4 Real-time shape approximation

Starting from a set of rig poses and matching shapes as input, there are several learning-based methods to reproduce and interpolate high-quality static shape deformations in *real time*. One can learn pre-skinning residuals using Radial Basis Functions (RBFs) [1], or further process the residual using PCA [2]. The latter was applied to hand deformation with training obtained from FEM,

similar to our hand example but using a soft-tissue FEM pipeline [37] as opposed to musculoskeletal anatomy-based simulation. Bailey et al. [10, 63] used both PCA and deep neural networks to learn real-time film-quality character mesh deformations originating from production rigs. Similarly for facial animation, Song et al. [64] used joint transforms as input and learned localized character shapes in differential coordinates. These methods did not pursue multi-resolution shape deformation. In follow-up work, Bailey et al. [65] presented a method that uses convolutional neural networks for approximating facial mesh deformations. While this method employed a 2-level coarse/fine deformation structure, their construction uses texture mapping, and is specific to facial deformation (e.g., to accommodate wrinkles).

Our method is designed for general meshes and generic multi-level mesh simplifications schemes; we use 4 LODs in our examples. Multi-resolution hierarchies are commonly used in computer graphics, see, e.g., the survey of Boier-Martin et al. [9]. They can be combined with FEM simulation [66–69], and form the core of the multigrid method [70, 71]. These methods are typically not designed for hard real-time systems, however. For fast evaluation in games, progressive upscaling has been previously explored for cloth simulation [72]; in our work, we drive character skins using a skeleton hierarchy.

2.5 Learning physics

Numerous studies have demonstrated that learning-based approaches are effective in addressing the computational challenges of physics-based simulation methods, particularly in subspace learning and modeling. Fulton et al. [73] utilized an autoencoder neural network to generate nonlinear reduced spaces for deformation dynamics, with the reduced space being drastically smaller than conventional linear model reduction, improving performance and robustness. and Holder et al. [74] used a neural network to learn the motion and interaction of deformable objects in a subspace. Badias et al. [75] applied model reduction to contact solutions in a reduced space, enabling real-time

interaction between virtual and physical objects. To achieve fast and detailed contact-driven deformation, Casas et al. [76] integrated nonlinear learning-based corrections with existing linear handle-based subspace formulations, whereas Romero et al. [77] machine-learned contact deformations in a contact-centric manner and integrated them with subspace dynamics. , demonstrating real-time dynamics with fine contact deformation detail.

Unlike the above methods that benefit from the subspace, the light-weight NNWarp [78] corrects a linear displacement by predicting a per-vertex nonlinear incremental displacement, allowing real-time simulation of large models. Statistical and learning approaches have been used for a long time to build approximations of real-world scanned deformations, for example [79] and its many follow-up works. These methods have however generally not been designed with real-time or multi-resolution performance in mind. There are also researchers working on producing character soft-tissue dynamics using neural networks [80–83].

Simulating clothing using deep learning is another well-studied field. Given a garment and human body rig pose, Santesteban et al. [84] trained a recurrent neural network of cloth drapes and wrinkles, as a function of body shape and dynamics, on the database of physically-based dressed character simulations. Chentanez et al. [85] introduced a triangle mesh-based convolutional neural network that trains on the data from physics based simulations (XPBD and FEM). Unlike the previous two methods that are supervised, the method of [86] formulates deep learning as an implicit Physically Based Simulator to learn the realistic cloth Pose Space Deformations in an unsupervised way.

As discussed above, the application of machine learning to predict shape deformation has been extensively researched. Few methods have discussed real-time performance, however. Moreover, previous methods did not address the need to simultaneously support multiple LODs and hard real-time deformation. In our work, we give a method that computes the shape at any level(s) of detail at hard real-time rates, and does so intertwined with standard widely used mesh level-of-detail algorithms, e.g., quadric error surfaces [12].

2.6 Learning the rig

Researchers have also explored learning-based methods to produce a LBS rig from mesh animations [87], or avoid the standard LBS shortcomings. Liu et al. [88] presented an end-to-end deep graph convolution network to automatically compute skin weights for skeleton-based deformation of production characters. Deng et al. [89] compute neural indicator functions to efficiently represent articulated deformable objects, and RigNet [90] utilized a deep architecture to predict skeletons and surface skin weights from input 3D character meshes. Similarly, the work [91] developed a neural network capable of rigging an input character mesh, along with neural blend shapes to improve the deformation quality. The above methods specifically focused on rigging and skinning weights, whereas we investigate LOD mesh deformation at hard real-time rates. Our training data comes from physically based simulation and incorporates volume preservation and contact resolution.

Chapter 3

Simulation of Hand Anatomy Using Medical Imaging

3.1 Overview

A human hand contains multiple organs (Figure 3.1) with diverse function and mechanical properties. For example, a fat tissue is passive and softer than muscle (even when unfired); but muscles are of course active. In order to capture this variability, we model tendons, fat, muscles, bones, and ligaments as separate simulation objects (Figure 3.2). The input to our system is an animation of the hand's joint hierarchy, obtained using any suitable method, such as motion capture, keyframe interpolation or monocular optical tracking. Our simulation proceeds in layers (Figure 3.3). The output of our system is the animation of the skin (outer surface of the fat), as well as animation of the other musculoskeletal organs.

In order to animate the geometry of the bones (i.e., the bone triangle meshes), we adopt the method and data made publicly available by [3]: the translation and rotation of each bone relative to the parent bone is calculated using a data-driven method, trained by observing bone motion in multiple hand poses using MRI. This gives a deterministic method that correctly positions each bone mesh for any plausible joint angles (pose), across the range of motion of the hand. Because we also use the MRI scans made publicly available by [3] to model the rest of our hand anatomy, the subject used in our work is the same as that used by [3] (male, late 20s), i.e., our non-bone simulation layers use the same MRI data that was used to create the bone rig, giving us internal consistency of our layers.

We then perform a bone fascia simulation to fill out the valleys between the bones, using a cloth solver (Section 3.4). Thereafter, we perform tendon simulation, using the computed bones and bone fascia mesh animations; our tendon hybrid data-driven + simulation method is novel, and is described in Section 3.3. Next, we perform our novel muscle simulation (Section 3.2), again treating the results of the previous layers as known fixed mesh animations; followed by a muscle fascia simulation. The next step is to perform joint ligament simulation. Ligaments are treated similarly to muscles, namely controlled by plastic strains. Finally, we perform our novel fat simulation, by constraining it to tendons, the muscle fascia, the bone fascia, and joint ligaments, again treating all previously simulated objects as fixed mesh animations (Section 3.5). Finally, we render the results using volume rendering, and standard surface-based rendering techniques (Maya Arnold and Pixar Renderman).

We extensively use the ICP algorithm in several parts of our system. Except where we explicitly state that we used Wrap3 [92], we perform ICP using our implementation of [93].

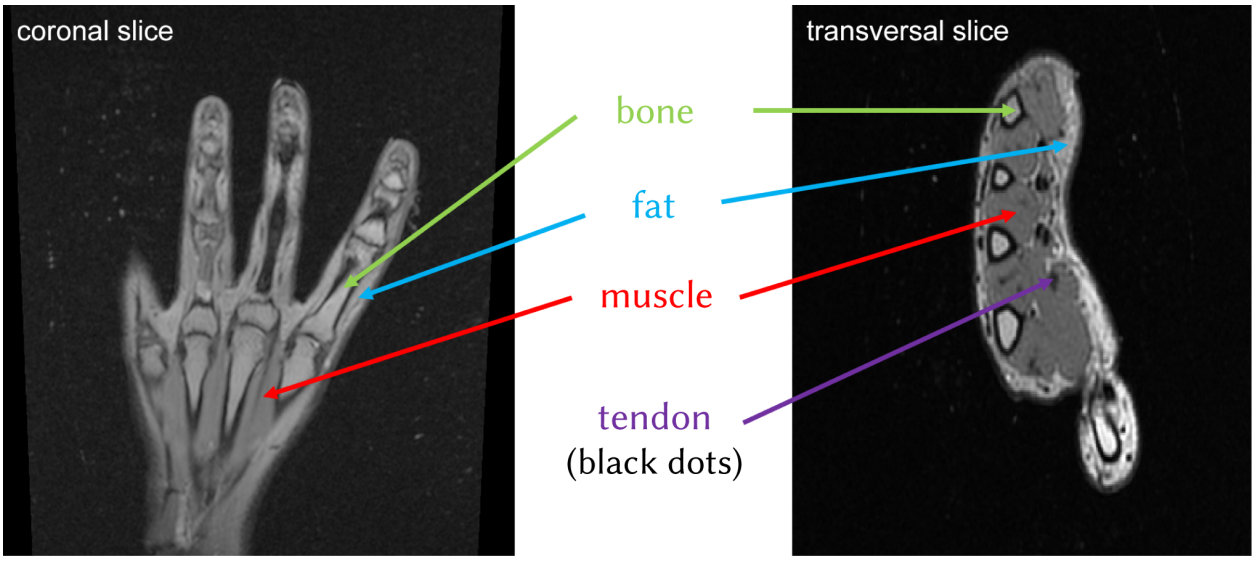


Figure 3.1: **Human hand MRI slices (coronal and transversal plane).** Hand has several tissues represented by different MRI intensities.

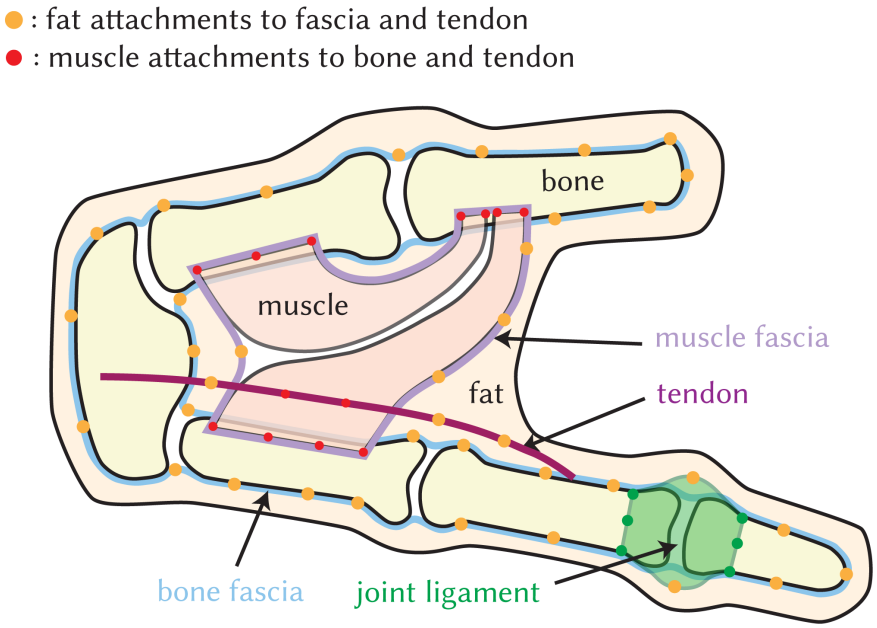


Figure 3.2: **Overview of our human hand rig.** We model tendons (dark red), fat (light orange), muscles (light red), bones (yellow), and ligaments (green) around joint regions. To improve simulation quality, we also model two additional fascia layers: bone fascia (blue) and muscle fascia (purple). The fat tissue is directly attached to fascia layers and ligaments.

3.2 Muscles

We start the description of our layered simulation by describing our muscle preprocessing pipeline and simulation model. Although muscles are not the first simulation layer, they are the technically most complex and most important part of the thesis, hence we describe them first. As always, before simulating muscles, the assumption is that the previous simulation layers have already been completed (Figure 3.3).

3.2.1 Muscle Simulation

The shapes of our muscles evolve under constraints to the bones and other muscles, as well as due to muscle firing. We model muscle firing using pose-varying muscle plasticity, controlled by joint transformations. We model our muscles as a coupled flexible multibody dynamic system, simulated using the Finite Element Method. Each muscle is a separate deformable object. In our work,

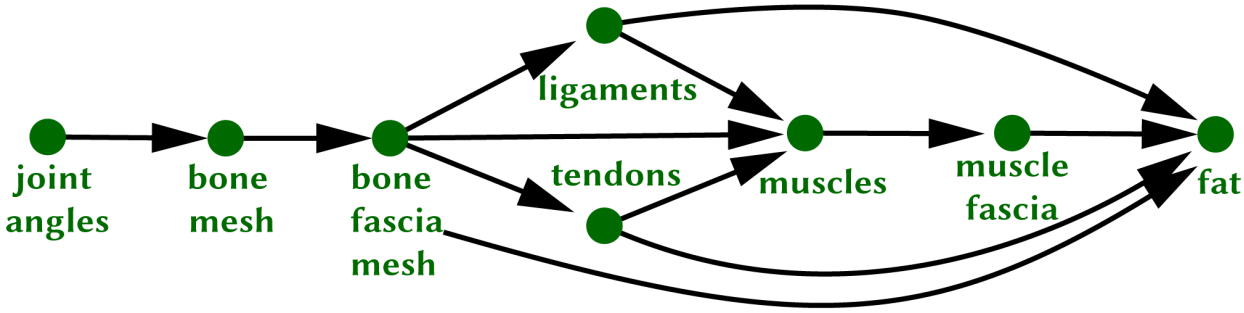


Figure 3.3: **Our simulation layers.** The input to our system is a hand joint angle animation. The output is the animated external surface of the fat ("skin"), as well as the animated shapes of the internal musculoskeletal organs. Arrows denote dependencies: before a layer (green node) can be simulated, all inputs must already be simulated.

we are interested in static shapes under the given plastic strains. We found that results are more stable when performing dynamic simulations, using a simple and stable integrator, namely implicit backward Euler, as opposed to employing quasi-static solving. This is because the presence of mass and damping in the dynamic simulation acts as regularization. The dynamic simulator does produce some secondary motion, but it is extremely small and we neglect it.

Constraints: We apply four types of constraints to mimic muscle biomechanical behavior. They are (1) muscle attachments to bones and tendons; (2) muscle contacts to bones; (3) muscle inter-contacts and self-contacts; and (4) muscle inter-sliding. Namely, muscles are attached to the bones and tendons (1), they cannot penetrate the bones (2), other muscles (3), and themselves (3). Muscles are also sliding against neighboring muscles (4); this is modeled by sliding constraints. If there are no sliding constraints, large empty space appears between neighboring muscles, which is unrealistic. To specify the sliding constraints between a pair of adjacent muscles, we first find the "proximity" vertices between them, in each example pose. "Proximity" vertices are those either in contact with, or close to the neighboring muscle. The sliding vertices for this pair of adjacent muscles are the intersection of proximity vertices across all example poses. As shown in Figure 3.6, a muscle is attached to several rigid bones. The bones undergo rigid motions around their parent bones [3]. The attachment vertices (in red) move rigidly with the attached bones.

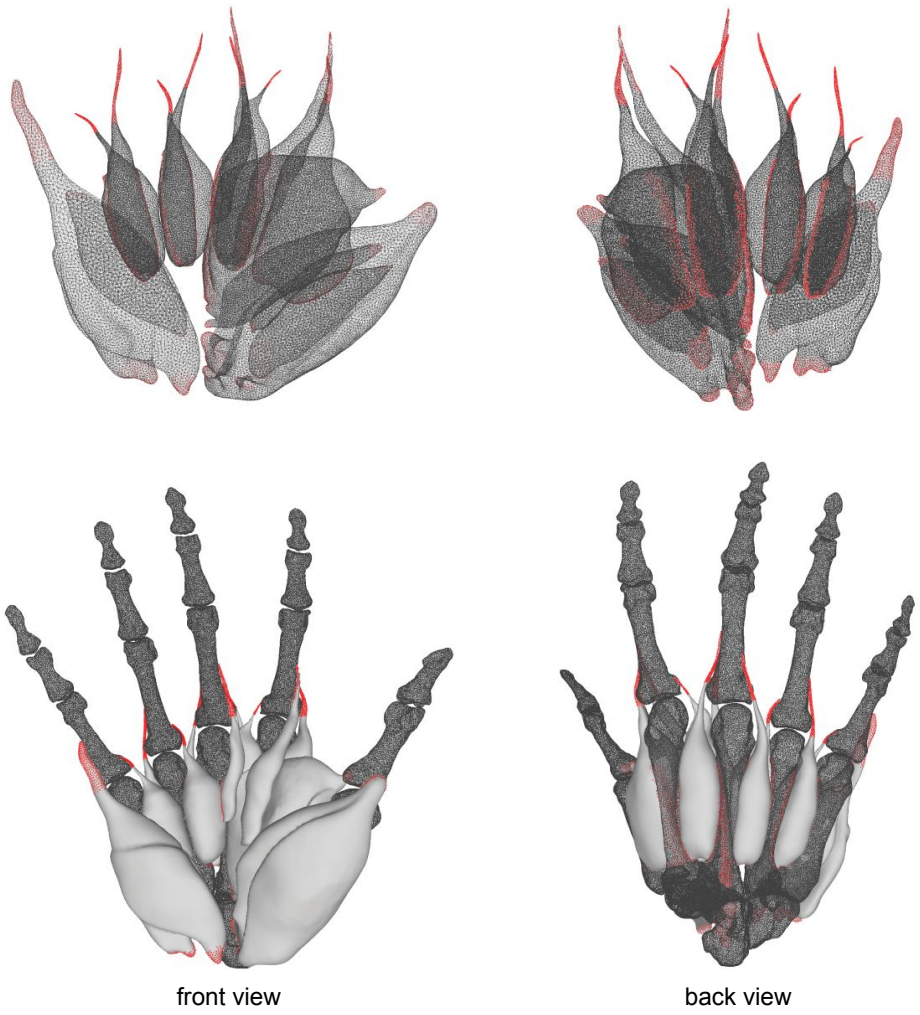


Figure 3.4: **Muscle attachments.** Muscle attachment vertices are depicted as red dots. To improve the viewing of attachments, the muscles are visualized in dark wireframe in the left two figures. In the right two figures, we show bones in dark wireframe to display the relative bone locations.

Pose-varying muscle activation: As stated in the introduction, we limit muscle activation to concentric isotonic contractions. Under this assumption, muscle activation becomes a unique function of the hand's pose. This means that we can model activation by calculating a pose-varying plasticity field across the hand: in each pose, the shape of the muscle is obtained by calculating the static equilibrium under the given "plastic" deformation in this pose, the nonlinear muscle elasticity, and the constraints. We note that this is related to the 6-DOF plastic model from [52]; however, we do not embed all muscles into a single global tetrahedral mesh, but rather model each muscle as a separate object with its own tetrahedral mesh. The first reason for this choice is that muscles

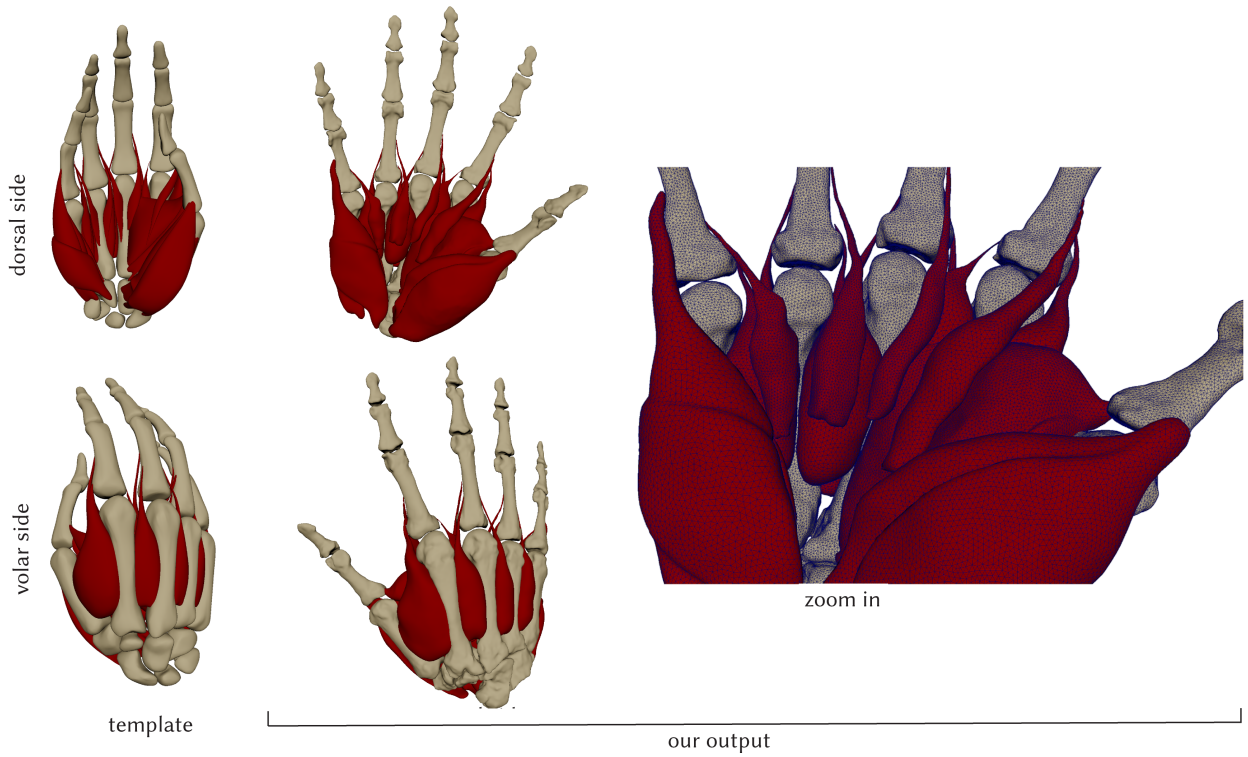


Figure 3.5: **Seventeen muscles of the human hand extracted from MRI.** Observe that the template hand is larger than the scanned hand. The pose is also different. Our method addresses this using bone attachments.

are very often undergoing sliding contact against each other, in addition to attaching to certain locations, which cannot be modeled using a single tetrahedral mesh. The second reason is that it is much more feasible and robust to define the pose-space for each muscle individually, as opposed to employ a global pose-space for all the muscles.

Unlike the human face, human hand has a highly concave shape and has many flexible joints. If a single tetrahedral mesh was used, each joint would contribute several DOFs into the global pose-space, thereby causing a very large final dimension of the pose-space. This would be problematic given that our example poses are sparsely distributed in the pose-space, resulting in lower interpolation accuracy. Admittedly, we could define a spatially-varying pose space, meaning that different parts of the hand have different pose spaces, but an accurate and smooth separation of the human hand is not easy.

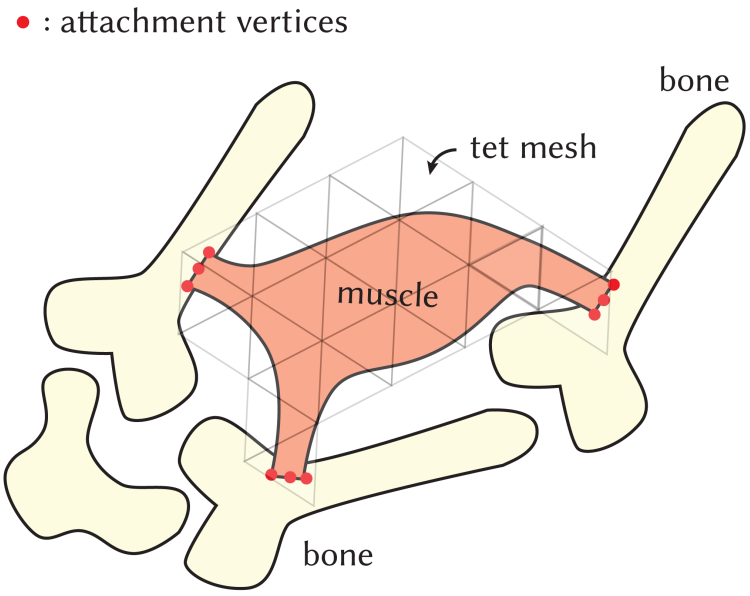


Figure 3.6: **Overview of muscle simulation.** A muscle is in general attached to several bones (and/or tendons). The muscle surface mesh is embedded into a tetrahedral mesh. The pose-varying muscle contraction is modeled via the plastic strain of each tetrahedron.

What remains to be discussed is how one calculates the plastic strains at each tetrahedron in the given (arbitrary) pose. We do this by extracting plastic strains in each *example* pose, from the MRI scan (Section 3.2.4). Next, we interpolate those strains to the given (arbitrary non-example) pose. The weights for this interpolation are different for each muscle, and are determined by defining a low-dimensional pose-space for each muscle, described in Section 3.2.3. During each timestep, we convert the bone rigid transformations to the pose-space vector a for each muscle (Section 3.2.3). The interpolation weight w_i for example pose i is then defined as

$$w_i = \frac{\phi(a,a_i)}{\sum_j^N \phi(a,a_j)}, \qquad \text{for} \qquad \phi(a,b) = \frac{1}{\|a-b\|}, \tag{3.1}$$

where N is the number of example poses. Plastic strain $P \in \mathbb{R}^{6m}$ of the muscle is computed as

$$P = \sum_i^N w_i P_i, \tag{3.2}$$

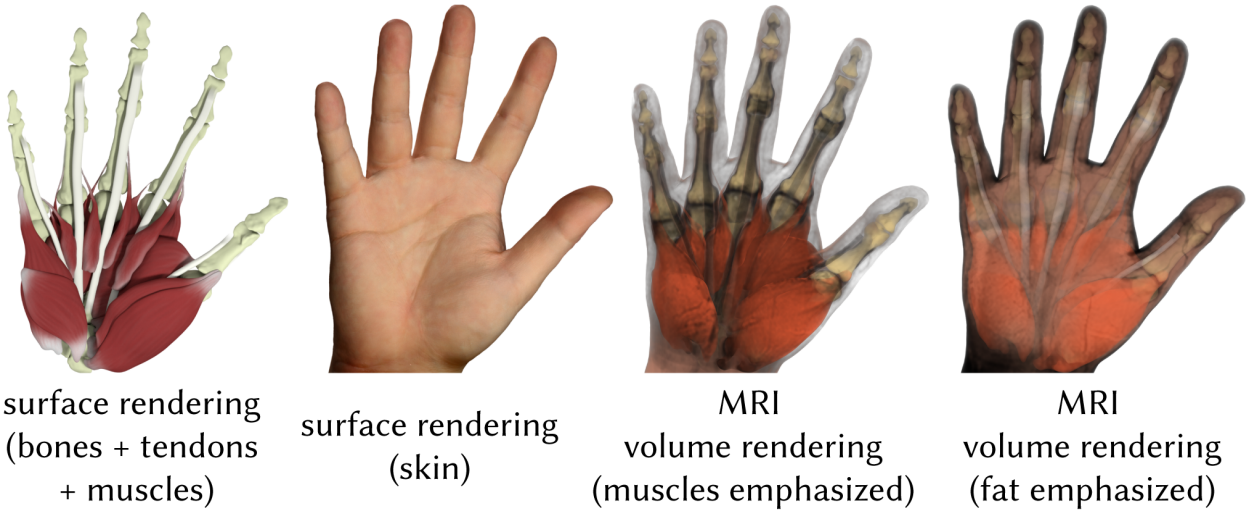


Figure 3.7: **Neutral pose of the hand and its interior musculoskeletal structure as modeled in our method.** Note that muscles are not completely relaxed in this pose. However, the pose is close to the "true neutral pose" (relaxed muscles), and is easy to capture and process (no occlusions).

where a_i and $P_i \in \mathbb{R}^{6m}$ are the pose vector and example plastic strain at example i, respectively, and m is the number of tets of this muscle. The plastic strain vector P_i has 6 entries per tet, i.e., symmetric part of a 3×3 matrix. We use isotropic elastic materials (stable neo-Hookean material [39]) for our muscles (and also fat). Therefore, for each tet, only the the symmetric part of the 3×3 plastic strain matrix matters, as the rotation is absorbed by the elastic material; therefore, P_i has 6 entries per tet.

We do not use radial basis functions to interpolate the plastic strains, because this introduces negative weights, which in turn causes the determinant of the plastic strain to be negative or close to zero, thereby leading to simulation instabilities.

We simulate joint ligaments using the same method as muscles (Figure 3.8); except that we do not model contacts and sliding between ligaments, as they are not in close geometric proximity. Therefore, we can simulate each joint ligament separately (Figure 3.8).

3.2.2 Creating Neutral Muscle Shapes From MRI

During preprocessing, we extracted the shapes of all muscles in the *neutral pose* (Figure 3.7); we use the method described in [33]. This method starts from a generic surface and tetrahedral mesh

muscle template, and reshapes both the surface and tetrahedral template to match the MRI image. Matching the medical image involves "landmarks" (locations on the template muscle and MRI image that are known to correspond to the same anatomical location), "ICP markers" (location in the MRI image that are, with a high degree of confidence, located on the boundary of the muscles; without there being a known correspondence on the template), and "attachments" (locations where a muscle attaches to a bone).

We manually specified landmarks, attachments and ICP markers in the MRI images, with the aid of medical literature and a medical doctor radiologist. We first specify the attached vertices on the template mesh. Then, we extrapolate the vertex positions based on the deformation from the template bone meshes to our bone meshes, followed by any (typically minor) manual adjustments to ensure the final positions match the MRI. Specifically, for each attachment vertex, we find the closest patch on the template bone meshes. The patch is defined as a set of connected triangles. Then, we extract the transformation based on the deformation of the same patch from the template bone meshes to our bone meshes. Finally, the attachment vertex is transformed based on the extracted transformation. The resulting attachments are shown in Figure 3.4. The entire process to create the neutral muscle surface mesh and tet mesh took approximately 3 days total for all the 17 muscles (Figure 3.5) of the human hand, including manual work and computer time.

3.2.3 Muscle Pose Space

Without loss of generality, consider a single hand muscle. A single muscle usually attaches (originates/inserts) to several bones and tendons. The tendons, as described subsequently (Section 3.3), are also controlled by the motion of the bones. If a muscle is attached to tendon(s), then we treat the bones that control the tendon as the bones that also control the muscle, in addition to the directly attached bones. Thus, we can consider that the muscle is solely controlled by bones. We define the muscle pose space using the skinning rotation of the controlling bones. Skinning rotation of a bone is the rotation of the bone relative to its neutral pose. This rotation is commonly used, for example, in linear blend skinning. Because applying the same global rotation to all controlling bones should

not affect the pose space, we eliminate one controlling bone from the pose-space. We express the skinning rotations of all other controlling bones relative to the skinning rotation of that bone, and then remove it. The selected/removed bone is the controlling bone that has the largest number of descendant controlling bones.

To represent a rotation, there are several choices: (1) a quaternion; (2) a 3D rotation matrix; (3) a 3D axis-angle vector ("exponential map" representation). Using a 3D rotation matrix leads to a high dimension of the pose space. In addition, when using a 3D rotation matrix or a quaternion, it is more difficult to measure the distance to the example pose; this is needed to compute interpolation weights. In contrast, a 3D axis-angle vector only has 3 DOFs, resulting in a compact pose space. As it is defined in the Euclidean space, one can easily compute the distance to each example pose. As is well-known, adding any integer multiple of 2π to the angle produces the same rotation. We first convert the rotation matrix to a unit quaternion $q = cos(\phi/2) + sin(\phi/2)a$, and finally to the 3-vector ϕa, where we enforce $\phi \in [-\pi, \pi)$. Note that using q vs $-q$ increases ϕ by 2π, but this disappears when standardizing to $[-\pi, \pi)$, i.e., identical ϕa is produced. Limiting the angle to $[-\pi, \pi)$ does not produce discontinuities in our method because no pair of controlling bones for the same muscle undergo a rotation greater than 180 degrees across the range of motion of a hand. There are pairs of joints in the human hand that can undergo a relative rotation greater than 180 degrees (but always less than 360 degrees), e.g., fingertip joint relative to palmar bones; but those do not appear as controlling bones of the same muscle in our hand model. Because the rotation is always less than 360 degrees, if such bone pairs needed to be accommodated, we could shift the $[-\pi, \pi)$ interval to a more suitable per-muscle interval, e.g., $[-\pi/4, 2\pi - \pi/4]$ or similar.

We thus convert all bone transformations into 3D vectors, and combine them into a per-muscle pose-space vector. If a muscle is controlled by N_e bones, the dimension of the pose-vector for this muscle is $3(N_e - 1)$. When applied to example pose i, this gives us the pose vector a_i for this muscle. Note that different muscles have different pose-spaces and different controlling bones and a different N_e. The maximum value of N_e for a muscle in our hand model is 3. Note that we only model muscles within the hand; we do not model lower and upper arm muscles.

3.2.4 Muscle Plastic Strain Extraction in Example Poses

We now describe how we find the plastic strains P_i in all example poses i, so that the plastic strains change smoothly with pose, and so that, in each example pose, the muscle in static equilibrium under the plastic strain P_i matches the MRI landmarks and ICP markers of that example pose, and the muscles are not colliding with each other. Namely, we need to achieve the property that, for each muscle, for similar poses, the corresponding plastic strains are similar. We observed that without such pose-space smoothness, the muscle simulation outputs are visually temporally non-smooth. We now describe the steps of this process. We define plastic strains on a single per-muscle tetrahedral mesh for all example poses, i.e., per-muscle, there is the same surface and tet mesh topology for all poses, but different vertex positions. Before settling on our solution method below, we tried other methods, which did not work well and we abandoned them; we describe them in Section 3.2.5.

Our method employs three stages. Stage 1 entails executing [33] in each pose, as described for the neutral shape above in Section 3.2.2, with some additional "helper" algorithms to provide a better initial guess than starting from the neutral tet mesh. Our initial guess is obtained simply by simulating the hand to each example pose, using the bones, bone fascia, tendons, and muscle tet meshes generated in the neutral pose. When this proved too challenging due to example poses being too different from the neutral pose, we relied on a "pre-initial" guess obtained using optimization in the "computational bodybuilding" [29] space. The output of the method of Section 3.2.2 applied to this non-neutral pose is a surface mesh (and a tet mesh) matching the MRI scan in this pose.

We use the following strategy to stably deform the tet mesh into a pose that is sufficiently similar to the MRI pose. We first create a muscle fiber field on the muscle neutral tet mesh by solving a Laplace equation [29]. Then, we define

$$R\,\mathrm{diag}(a,1,1)\,R^T$$

as the plastic strain of each muscle tetrahedron, where R is the local frame defined by the muscle fiber direction, and a is the muscle contraction parameter [29]. We consider a simple optimization problem

$$arg\,min_{\mathbf{a},\mathbf{x}} \|\mathbf{L}\mathbf{a}\|^2 + c_1 E_{\text{elastic}}(\mathbf{a},\mathbf{x}) + c_2 E_{\text{att}}(\mathbf{x}), \tag{3.3}$$

where $\mathbf{a}$ is the muscle activation at each tet, $\mathbf{x}$ contains the vertex positions of the muscle's tet mesh (our output, i.e., the initial guess), E_{elastic} is the elastic energy under the neutral rest shape, and E_{att} is the attachment energy of the muscle to the bones, modeled by zero rest-length springs at the attachment sites.

Intuitively, this objective function attempts to discover the muscle activation so that, in the static equilibrium under the contraction, the attachment energy to the bones is small. This problem is a simplified version of the optimization problem defined in [33], because (1) it uses a much smaller space for the plastic strains, (2) it does not attempt to match the sparse markers, and (3) we treat the static equilibrium constraint as a penalty term in the objective function. The unconstrained optimization problem can be solved by using Newton's method. The DOFs provided by the contraction $\mathbf{a}$ are necessary, because the shape of a straight muscle would bend, as opposed to contracting, when two insertions come closer to each other. This is due to volume preservation terms in the elastic energy.

To simplify the problem, we performed model reduction over the vector $\mathbf{a}$. We manually select a few tetrahedra (usually 3-4) on the tetrahedral mesh and treat them as "handles," meaning that the contractions of these tetrahedra will control the entire $\mathbf{a}$. Then, we generate the subspace basis $\mathbf{U}$ by calculating bounded bi-harmonic weights [94]. Then, the contraction $\mathbf{a} = \mathbf{U}\hat{\mathbf{a}}$, where $\hat{\mathbf{a}}$ is the subspace quantity, and our optimization problem becomes

$$arg\,min_{\hat{\mathbf{a}},\mathbf{x}} \|\mathbf{L}\mathbf{U}\hat{\mathbf{a}}\|^2 + c_1 E_{\text{elastic}}(\mathbf{U}\hat{\mathbf{a}},\mathbf{x}) + c_2 E_{\text{att}}(\mathbf{x}). \tag{3.4}$$

It takes (on average) 2 minutes per muscle per pose to optimize Equation 3.4. Optimizing Equation 3.4 as opposed to Equation 3.3 gives us a 5$\times$ speedup and most importantly, easy optimization convergence when the dimension of $\hat{\mathbf{a}}$ is small.

Two problems now become readily apparent: as we extracted muscles separately, the different muscle meshes collide in this pose, and; for each muscle, the plastic strains induced by the optimized tet mesh are (in general) *not* a smooth function of the pose. For this reason, we discard the tet meshes, and proceed to Stage 2 whereby we correct the Stage 1 muscle surface meshes to not be colliding.

Our Stage 2 entails using the ShapeOp bending energy [95] combined with a contact penalty energy term that pushes each volumetrically colliding vertex to the closest point on the surface; this dramatically improved performance. In our experiments, it takes less than 10 minutes to resolve the inter-contacts between muscles for each example pose. A volumetric method to remove muscle interpenetrations was reported in [3]. However, we found that it takes hours to resolve the contact in a single example pose, which is too slow to process all muscles in all example poses. This is because the muscle volumetric meshes have many DOFs, and the effect of contact propagates volumetrically, thereby imposing a large computational burden. In contrast, the surface of the muscle is easier to deform under contact if a surface-based method is used. Therefore, we prefer a surface-based deformation method.

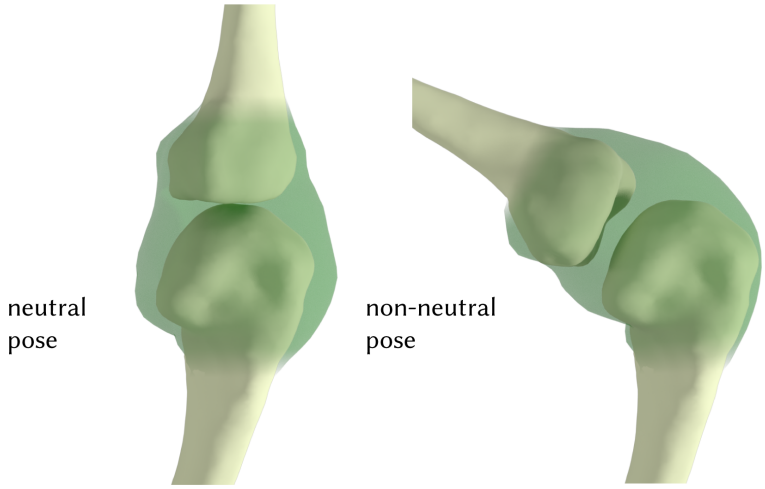


Figure 3.8: **Our simulated ligament around a hand joint, in two poses.**

Finally, in Stage 3, we ensure pose-space smoothness of the plastic strains. The output of Stage 2 are non-colliding muscle surface meshes (in each pose) that are close to MRI landmarks and ICP markers. We define the following energy which (separately for each muscle), jointly optimizes the plastic strains at all example poses, in a manner that causes the embedded muscle meshes to closely match the outputs of Stage 2, while ensuring pose-space smoothness. We note that, in order to avoid "spikes" at the sparse MRI landmarks and ICP markers, we intentionally do *not* use these landmarks and markers in this process here, but instead use a "dense correspondence" against the surface mesh output of Stage 2. This also safeguards against re-introducing muscle contacts. We optimize

$$\begin{aligned} arg\,min_{\mathbf{x}_1,\ldots,x_N} \quad & \sum_{i=1}^{N} \left(||\mathbf{L}\,\mathbf{S}(\mathbf{x_i})||^2 + \alpha \mathscr{E}_{\text{Dense}}(\mathbf{x}_i) \right) + \\ & + \gamma \sum_{j=1}^{m} ||\mathbf{L}_{\text{pose}} \left[\mathbf{S^j}(\mathbf{x_1}), \mathbf{S^j}(\mathbf{x_2}), \ldots, \mathbf{S^j}(\mathbf{x_N}) \right]^{\mathbf{T}} ||^2, \end{aligned} \tag{3.5}$$

where $x_i \in \mathbb{R}^{3n}$ are tet mesh vertex positions in example pose i (number of tet mesh vertices is n), N is the number of example poses and $\mathbf{L} \in \mathbb{R}^{6m \times 6m}$ is the volumetric mesh Laplacian (extended trivially to operate on 6-vectors; m is the number of tets). The function $\mathbf{S}(\mathbf{x}) \in \mathbb{R}^{6m}$ first calculates the deformation gradient at each tet, relative to the neutral shape, and then performs polar decomposition to extract the symmetric part and stores it (6 entries per tet, denoted by $\mathbf{S^j}(\mathbf{x})$); the gradient and Hessian of $\mathbf{S}(\mathbf{x})$ (needed for optimization) can be computed as in [33]. Under this definition, the objective function does not penalize the growth of the object and is unaffected by rotation. The quadratic term $\mathscr{E}_{\text{Dense}}(\mathbf{x_k})$ sums the squared surface mesh vertex Euclidean distances to the muscle surface output of Stage 2. The matrix $\mathbf{L}_{\text{pose}} \in \mathbb{R}^{N \times N}$ measures pose-space smoothness of plastic strains. It is the graph Laplacian matrix of the *pose graph* defined as follows: nodes are example poses, and edges connect each pose to all other poses, with weights inversely proportional to pose-space distance.

For muscle optimization in Equation 3.5, the quantity $\mathscr{E}_{\text{Dense}}(\mathbf{x_i})$ is simply the squared distance energy. We use block-coordinate descent. The gradient and Hessian of the pose-space smoothness energy $\mathscr{E}_{ps}(\mathbf{x_i}) = \frac{1}{2}||\mathbf{L}\mathbf{S}(\mathbf{x_i})||^2$ are

$$\nabla_{\mathbf{x_i}} \mathscr{E}_{ps}(\mathbf{x_i}) = (\mathbf{L}^T\mathbf{L}\mathbf{S})^T \frac{\partial \mathbf{S}}{\partial \mathbf{F}}\mathbf{G}, \tag{3.6}$$

$$\nabla^2_{\mathbf{x_i}} E_{ps}(\mathbf{x_i}) = \mathbf{G}^T(\frac{\partial \mathbf{S}}{\partial \mathbf{F}}^T \mathbf{L}^T\mathbf{L}\frac{\partial \mathbf{S}}{\partial \mathbf{F}} + \mathbf{S}^T\mathbf{L}^T\mathbf{L}\frac{\partial^2 \mathbf{S}}{\partial \mathbf{F}^2})\mathbf{G}, \tag{3.7}$$

where $\mathbf{F} = \mathbf{G}(\mathbf{x}_i - \mathbf{X}_i)$ is a vector concatenating deformation gradients of all elements, and $\mathbf{G}$ is the linear gradient operator matrix. The gradient and Hessian of $\mathbf{S}$ with respect to $\mathbf{F}$ can be computed as specified in [33]. We evaluate all terms using the sparse matrix form, except the term $\mathbf{S}^T\mathbf{L}^T\mathbf{L}\,\partial^2\mathbf{S}/\partial\mathbf{F}^2$. This term is efficiently evaluated by first calculating $\mathbf{S}^T\mathbf{L}^T\mathbf{L}$, which results in a vector. Then, the product of this vector and the 3D tensor $\partial^2\mathbf{S}/\partial\mathbf{F}^2$ is evaluated at each element in parallel (due to independence).

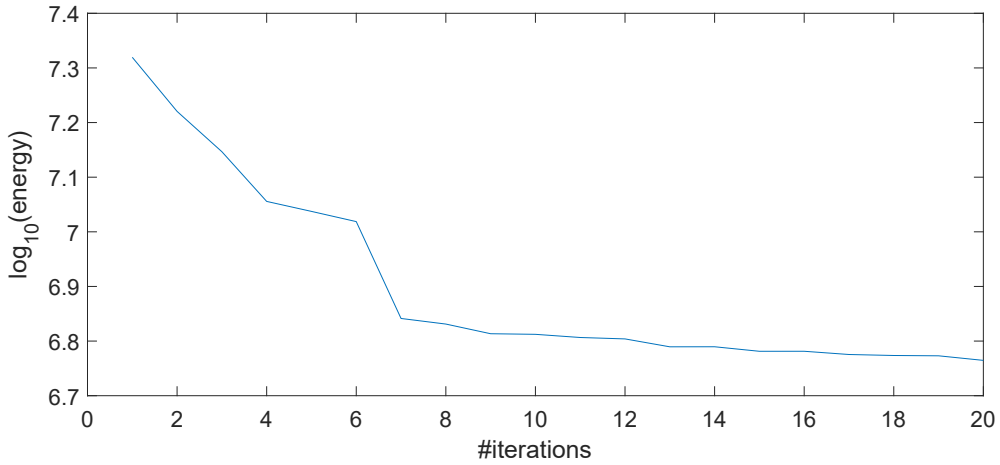


Figure 3.9: **The energy of our objective function (Equation 3.5) in each iteration**. The curve shows the energy in each iteration when we optimize the plastic strain for the thumb muscle group.

To optimize Equation 3.5 efficiently, we first solve N separate optimization problems without the pose-space smoothness term, starting from the neutral tet mesh. The solution for each example pose is used as the initial guess $\mathbf{x}$ to our optimization problem (Equation 3.5), which we solve using block-coordinate descent: in each iteration, we optimize for one $\mathbf{x_i}$, while freezing all $\mathbf{x_j}$ for

$j \neq i$. We repeat the entire block-coordinate iteration (all N poses) 20 times, for all our muscles. The output vertex positions $\mathbf{x_i}$ are then converted to plastic strain by calculating the deformation gradients relative to the neutral shape. The weight γ is tuned based on the maximum distance between the output surface and the input surface. We use 0.5mm as the maximum allowed distance. By using the strategy of bisection, we can automatically determine the value of γ for each muscle. Note that we did not add an explicit constraint that guarantees that the deformation gradients corresponding to $\mathbf{x}$ are positive-definite. This is because even without the constraint the eigenvalues of our resulting plastic strains for all muscles in all example poses were between 0.08 and 2.3 in our experiments. Moreover, it makes the optimization much easier. Our optimization converges for all muscles. The energy of the objective function in each iteration for one of the muscles are shown in Figure 3.9. The entire optimization typically takes 2.5 hours for all poses and all muscles.

3.2.5 Other solutions for volumetric plastic strains smoothness

The first abandoned idea was to augment the method described in [33] such that it supports pose-space smoothness across all example poses. The optimization is defined as follows,

$$arg\,min_{\mathbf{s_i},\mathbf{x_i}} \quad \sum_{i}^{N} \left(||\mathbf{L}\,\mathbf{s_i}||^2 + \alpha \mathscr{E}_{\mathrm{MI}}(\mathbf{x_i}) + \beta \mathscr{E}_{\mathrm{a}}(\mathbf{x_i}) \right) + \gamma ||\mathbf{L_{ps}}\,\mathbf{s}||^2, \tag{3.8}$$

$$\text{st. } \mathbf{f}_{\mathrm{e}}\left(\mathbf{F_p}(\mathbf{s_i}), \mathbf{x_i}\right) + \mathbf{f}_{\mathrm{a}}(\mathbf{x_i}) = 0, \text{for each } \; i = 1, 2, \ldots, N \tag{3.9}$$

where $\mathbf{L_{ps}}$ is a pose-space Laplacian matrix. Directly solving this is not feasible because the number of unknowns is enormous. Thus, we also use the block-gradient descent method, whereby in each iteration, we randomly select pose $k \in [1, N]$, and freeze unknowns related to all other poses. Then, the problem in each iteration becomes

$$arg\,min_{\mathbf{s_k},\mathbf{x_k}} \quad ||\mathbf{L}\,\mathbf{s_k}||^2 + \alpha \mathscr{E}_{\mathrm{MI}}(\mathbf{x_k}) + \beta \mathscr{E}_{\mathrm{a}}(\mathbf{x_k}) + \gamma ||\mathbf{s_k} - \bar{\mathbf{s}}_\mathbf{k}||^2, \tag{3.10}$$

$$\text{subject to:} \quad \mathbf{f}_{\mathrm{e}}\left(\mathbf{F_p}(\mathbf{s_k}), \mathbf{x_k}\right) + \mathbf{f}_{\mathrm{a}}(\mathbf{x_k}) = 0, \tag{3.11}$$

where $\bar{\mathbf{s}}_\mathbf{k} = \sum_{j \in N(k)} w_{kj} \mathbf{s}_\mathbf{j}$ is constant during each iteration.

By closely examining the objective function, we observed that this formulation has flaws. Remember that $\mathscr{E}_{\mathrm{MI}}(\mathbf{x_k})$ and $\mathscr{E}_{\mathrm{a}}(\mathbf{x_k})$ are terms to match the sparse observations. They penalize the distances of a sparse set of surface points to the target positions. On the other hand, the term $||\mathbf{s_k} - \bar{\mathbf{s}}_\mathbf{k}||^2$ is defined densely for each tetrahedron, and causes the shape of the volumetric mesh to match $\bar{\mathbf{s}}_\mathbf{k}$. To minimize the energy, in the region where there are no markers, the shape of the volumetric mesh follows $\bar{\mathbf{s}}_\mathbf{k}$. In the region where there are markers, however, these terms will combat each other. This is because in general the shape corresponding to $\bar{\mathbf{s}}_\mathbf{k}$ does not meet the markers. As a result, bumps appear when $\alpha, \beta > \gamma$. Conversely, the surface simply cannot meet the markers if $\alpha, \beta < \gamma$. We have implemented this approach and verified experimentally that the method suffers from the listed negative outcomes. To resolve this problem, we have to use a dense observation of the markers, which in turn causes a prohibitive time complexity. Therefore, this approach is not feasible and we abandoned it.

We next try to solve using a geometric shape modeling method. To use a dense correspondence, namely a complete target surface, we could simplify our previous objective function and treat it as a pure geometric shape modeling problem. We define smoothness as the Laplacian of the deformation gradient [29, 96], resulting in

$$arg\,min_{\mathbf{x_i}} \quad \sum_{i}^{N} \left(||\mathbf{L}\,\mathbf{F}(\mathbf{x_i})||^2 + \alpha \mathscr{E}_{\mathrm{MI}}(\mathbf{x_i}) \right) + \gamma ||\mathbf{L_{ps}}\,\mathbf{F}(\mathbf{x}))||^2, \tag{3.12}$$

where $\mathbf{F}(\mathbf{x})$ is the deformation gradients of $\mathbf{x}$, and $\mathscr{E}_{\mathrm{MI}}(\mathbf{x_i})$ are now dense correspondences. Here, $\mathbf{F}(\mathbf{x}) = \mathbf{G}(\mathbf{x} - \bar{\mathbf{x}})$, where $\mathbf{G}$ is the gradient operator matrix and $\bar{\mathbf{x}}$ is the rest position of the volumetric mesh. As we can see, the deformation gradient $\mathbf{F}$ is a linear function of the position $\mathbf{x}$. The resulting $\mathbf{x}$ can then be converted to plastic strains. Because we are using isotropic hyperelastic materials, the plastic strain is essentially the symmetric matrix of the polar decomposition of the deformation gradient F for each tetrahedron. Although it is a quadratic energy and easy to optimize, this method cannot handle rotations well, leading to a large amount of inverted tetrahedra.

To resolve this problem, we attempted to add a non-linear inequality constraint that guarantees that the determinant F of each tetrahedron is positive, that is, $\det(F) > \varepsilon$, where ε is a very small positive number such as 0.02. Nonetheless, this method creates extreme plastic strains whose determinants constantly hit the ε boundary, and whose eigenvalues span a wide range, such as from ε to 15.2. This causes extreme plastic strains when interpolating the plastic strains during simulation. Moreover, solving such a constrained nonlinear optimization is difficult. We have encountered failure muscles.

To address the problem of bad rotations, we also attempted to replace the Laplacian smoothing energy with a standard elastic energy [39]. By adjusting the resolution of the tetrahedral mesh, we were able to solve this optimization for every muscle. Nonetheless, this method still created extreme plastic strains. This is because an elastic energy penalizes the growth of the object, while the muscles are growing/shrinking all the time in different example poses. Therefore, we abandoned this method as well.

3.3 Tendons

Tendons are important not only because they control the motion of the hand, but also because they affect the appearance of the hand. Tendons are often beam-shaped and are always black in the MRI image, thereby giving them good contrast to the neighboring tissues. However, tendon extraction is difficult in practice because of the limited resolution of the MRI; thin tendons are often impossible to extract. We model tendons extruded from the flexor digitorum superficialis/profundus muscles, flexor pollicis longus muscle, extensor pollicis longus muscle, extensor digitorum muscle, extensor indicis muscle and extensor digiti minimi muscle. All tendons are very dark (black) in the MRI. Thus, if tendons travel through the same pulley, it is difficult to distinguish them in the MRI. To address this issue, we group tendons that go through the same pulley into a tendon group, shown here in Table 3.1.

There are 10 tendon groups in our system. Groups contain either a single tendon or two tendons, as indicated. There is a separate group for the palmar and dorsal side of each finger (10 groups total). On real hands, there are three tendons on the dorsal side of the thumb finger: extensor pollicis longus, extensor pollicis brevis, and abductor pollicis longus. The latter two are not visible in the MRI scan, and therefore we only simulate the extensor pollicis longus. The extensor digitorum tendon spans three fingers (index, middle, ring), and there is a tendon "bridge" (in the upper dorsal palmar area) connecting the three fingers; these bridges are not simulated. Similarly, we do not simulate the tendon "bridge" between the extensor digitorum and extensor digiti minimi. As a side effect, however, the tendons from the same group cannot slide relative to each other, which is a limitation of our work.

Table 3.1: **Tendon Groups.** We model ten tendon groups in our system. Groups contain either a single tendon or two tendons, as indicated.

finger	palmar group	dorsal group
thumb	flexor pollicis longus	extensor pollicis longus
index	flexor digitorum superficialis, flexor digitorum profundus	extensor indicis, extensor digitorum
middle	flexor digitorum superficialis, flexor digitorum profundus	extensor digitorum
ring	flexor digitorum superficialis, flexor digitorum profundus	extensor digitorum
pinky	flexor digitorum superficialis, flexor digitorum profundus	extensor digiti minimi

3.3.1 Tendon Simulation

Tendons in the real world and in our work serve as attachments for muscles, muscle fascia and skin, and therefore they substantially affect the simulations of those layers. In Section 3.6, we give an ablation study where we demonstrate that hand knuckles are not properly resolved without tendons (Figure 3.29, (a)). Our tendon simulation proceeds as follows. As illustrated in Figure 3.10

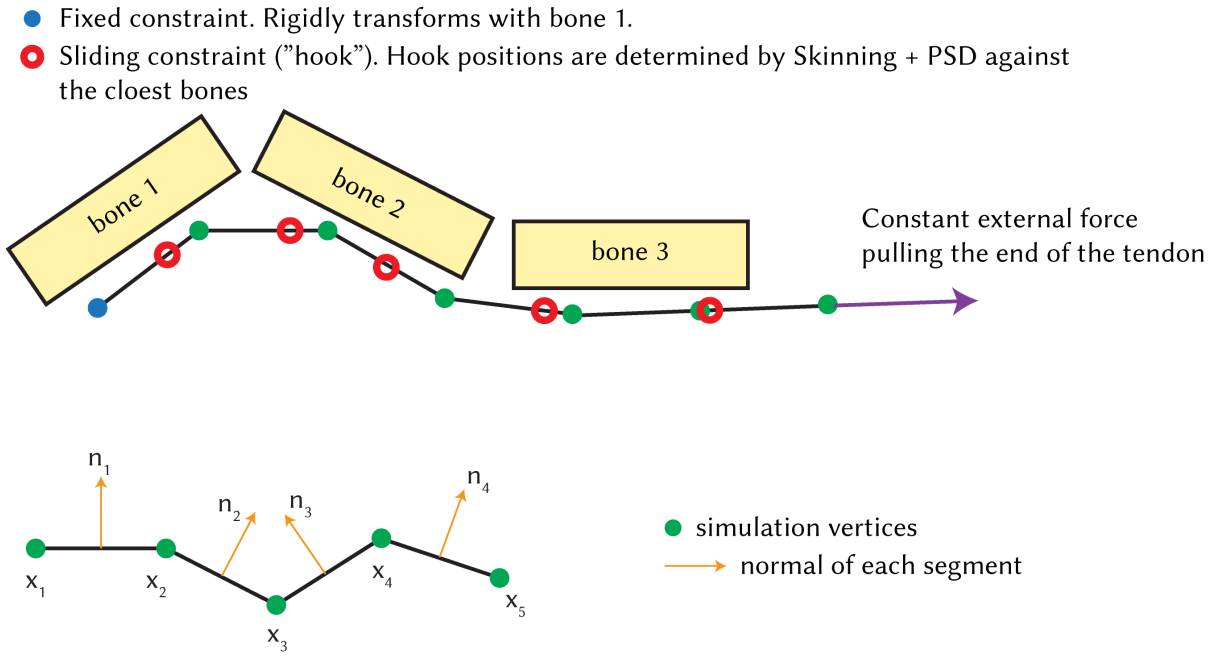


Figure 3.10: **Tendon simulation model.** Left: our tendons are simulated as rods consisting of multiple segments. Each segment is represented by the positions of two end vertices and a normal defining the orientation. The normal (orange) is perpendicular to the segment. The rod is constrained to slide through a few locations ("hooks"; red circles) that are skinned (with PSD correction) to the closest bone(s). One end of the tendon is attached to a bone with a fixed constraint. We apply a constant force (purple) at the other end of the tendon to stretch it without invalidating the constraints.

(left), we first discretize the tendon into Q small segments (Q is approximately 300 on average in our examples). Each segment i is controlled by the two endpoint vertex positions x_i, x_{i+1} and the normal of the segment n_i. Normal n_i is perpendicular to segment i. The simulation DOFs are x_i and n_i. As shown in Figure 3.10 (right), one end of the entire tendon (blue) is attached to the closest bone ("bone 1") and is rigidly transformed with it. The other end of the entire tendon is pulled by a constant external force (purple) in the longitudinal direction to mimic the fact that the muscles of the forearm are pulling the tendon. The tendon is constrained by a series of points (shown in red circles) located near the bones that we call "hooks". The tendon centerline is constrained to slide through the hooks.

Our tendon model is designed to keep the length of the tendon constant, and prevent tendon twisting. Note that our hooks are not biological (they do not exist in the real hand). We use them

to cause the tendon simulation to conform to the MRI data; i.e., our hooks enable us to fuse tendon physically based simulation with MRI data. We note that Sachdeva et al. [27] also modeled hand tendons biomechanically using rods. In their model, tendons slide through real anatomical pulleys, which were obtained manually by referencing the medical literature. Their goal was to create a robust control system for the hand, and not match MRI data for a specific individual. The real anatomical pulleys are unfortunately not visible in our MRI images; and hence we did not adopt their method and instead use data-driven hooks.

Given the current hook locations $\bar{x}_1, \ldots, \bar{x}_k$, constant external forces and fixed attachments, we deform our tendon by minimizing the following optimization problem:

$$arg\,min_{x,n,t}\; E_{\text{sliding}}(x,t) + c_1 E_{\text{pulling}}(x) + c_2 E_{\text{twist-bend}}(n) \tag{3.13}$$

$$\text{subject to } (x_{i+1} - x_i)^2 - \ell^2 = 0, \quad \forall i \in [1, Q], \tag{3.14}$$

$$n_i^T (x_{i+1} - x_i) = 0, \quad \forall i \in [1, Q], \tag{3.15}$$

$$n_i^T n_i - 1 = 0, \quad \forall i \in [1, Q], \tag{3.16}$$

$$0 \le t_j \le 1, \quad \forall j \in [1, k], \tag{3.17}$$

$$x_1 = \hat{x}_1. \tag{3.18}$$

The energies are defined as

$$E_{\text{sliding}}(x,t) = \sum_{j=1}^{k} \Big(\big(1 - t_j\big) x_{s_j} + t_j x_{s_j+1} - \bar{x}_j \Big)^2, \tag{3.19}$$

$$E_{\text{pulling}}(x) = -x_{Q+1}^T f, \tag{3.20}$$

$$E_{\text{twist-bend}}(n) = -\sum_{i=1}^{Q-1} n_i^T n_{i+1} - c_3 n_1^T \bar{n}_1. \tag{3.21}$$

Here, k is the number of hooks, and $t_j \in [0,1]$ (for $1 \le j \le k$) is the line segment barycentric coordinate giving the closest position to the given hook location $\bar{x}_j$; and s_j is the index of the closest segment to hook j. Within each optimization iteration, the hook is therefore constrained to

stay on the same rod segment. However, we update the closest segments s_j after every optimization iteration; and this permits multiple rod segments to travel through a hook, i.e., a hook is not "stuck" on one rod segment. Parameters c_1, c_2, c_3 control the optimization tradeoffs (we use $c_1 = 0.5$, $c_2 = 1.0$, $c_3 = 1.0$), ℓ is the length of one rigid segment (computed as the total length divided by the number of segments); $\hat{x}_1$ is the fixed attachment location on bone 1; and f is the constant upper arm muscle pulling force. We define the sliding constraint as the sum of the squared distance from each hook to the closest point on the closest segment. We define the twist-bend energy as the negative dot product between two neighboring normals; the larger the dot product, the less twisting and bending. As we define our normals to point in a direction close to the bending axis, the energy $E_{\text{twist-bend}}$ is mainly penalizing the twisting (as opposed to bending). Note that it is not sufficient to determine all n_i because the tendon could be globally rotated by the same rotation. We determine the global orientation using the normal $\bar{n}_1$, calculated by skinning the neutral pose first line segment normal with the transformation of "bone 1". Our constraints also enforce that n_i is perpendicular to the segment i, and is a unit vector.

The optimization problem is solved using the interior-point method [97]. In practice, the optimizer was able to easily find an optimal solution. A single optimization iteration (solving Equations 3.13-3.21 under fixed ICP closest positions) takes approximately 0.1 seconds on average per tendon in our examples. Total optimization time for all tendons is approximately 6 seconds.

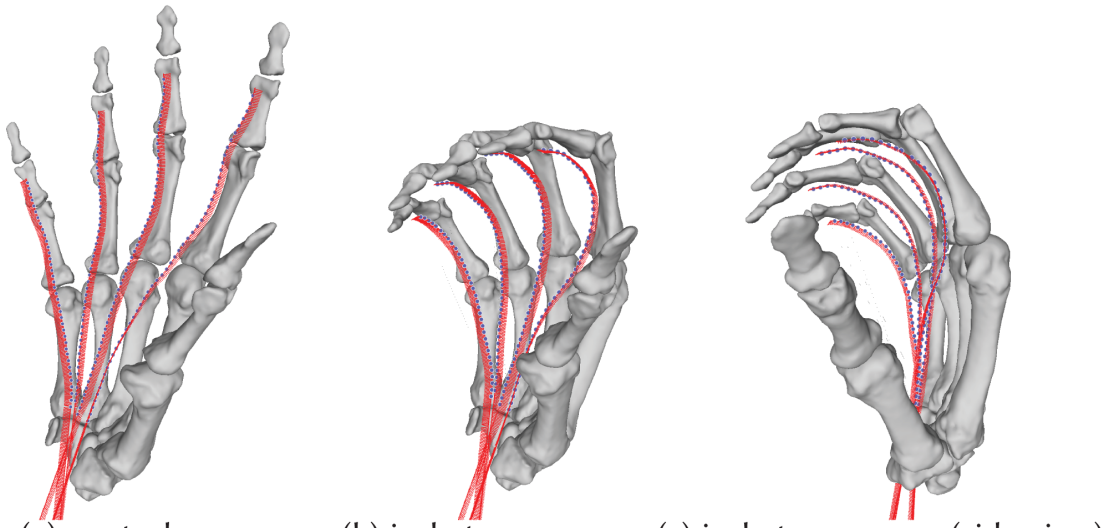
For tendon optimization, we use "autodiff" [98] to calculate the gradient and Hessian of $E_{\text{sliding}}(x,t)$ (Equation 3.19) and $E_{\text{ICP}}(x,n)$ (Equation 3.30). Autodiff is a C++17 library that enables automatic efficient computation of derivatives. The gradients and Hessians of $E_{\text{pulling}}(x)$ and $E_{\text{twist-bend}}(n)$ are

$$\nabla_x E_{\text{pulling}}(x) = -f, \qquad \nabla_x^2 E_{\text{pulling}}(x) = 0, \tag{3.22}$$

$$\nabla_{n_i,n_{i+1}} E_{\text{twist-bend}}(n) = (-n_{i+1}^T, -n_i^T)^T, \tag{3.23}$$

$$\nabla^2_{n_i,n_{i+1}} E_{\text{twist-bend}}(n) = \begin{bmatrix} & -\mathbb{I}_3 \\ -\mathbb{I}_3 & \end{bmatrix}. \tag{3.24}$$

For the fat optimization (Equation 3.36, after linearization of Equation 3.34), quantity $\mathscr{E}_{\text{skin}}(\mathbf{x}^i + \Delta\mathbf{x})$ is a squared distance energy and $||\mathbf{L}(\mathbf{s}^\mathbf{i} + G\Delta\mathbf{x})||^2$ is a quadratic energy.



(a) neutral pose (b) in-between pose (c) in-between pose (side view)

Figure 3.11: **Tendon simulation.** Here, we show the rods that represent tendons. The red lines are the rigid segments and their normals. The blue dots are the hooks.

With the runtime simulation formulation now specified, the only remaining task is to determine the hook locations, the fixed attachment location, and the constant force. Fixed attachment location $\hat{x}_1$ is computed by rigidly transforming its initial position using the transformation of the closest bone ("bone 1"). The direction of the pulling force is defined as the opposite of the average direction of the last few segments in the neutral configuration; this is because these segments are usually located in the wrist region, and slide along their longitudinal (i.e., tangential) direction in the real world. At each simulation frame, we compute the hook locations using skinning, augmented via pose-space deformation ("PSD") corrections [1], driven by bone rotations. The process to obtain skinning weights and the pose-space corrections is explained in Section 3.3.3. This makes the hook locations a unique function of the bone transformations. These hooks, however, may not be evenly distributed. We observed that some points may be very close to each other. To address it, we filtered out points that are very close to each other, because these hooks would introduce noise. Figure 3.11 shows the tendon hooks (blue), and the tendon simulation results.

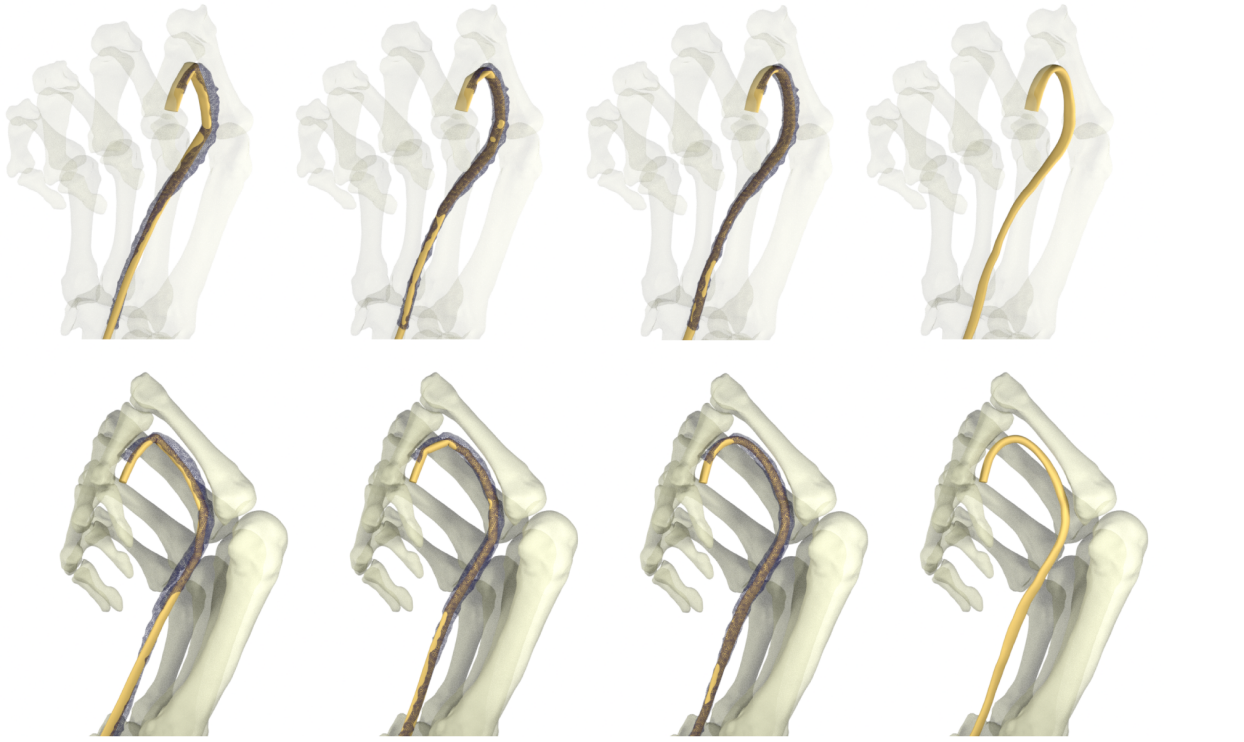
3.3.2 Tendon Extraction From MRI

To perform the simulation, we need to know the tendon hook locations. They are determined from MRI in multiple example poses. Therefore, we first extract the tendons from MRI. Although we model tendons as rods, real tendons are volumetric objects. To extract tendons, we first treat them as cylinder tubes whose centerlines are our simulation rods. We assume that the radius of the tendon does not change during the simulation and, thus, the surface shape of the cylinder is skinned by the centerline.

We create a template cylinder manually. We then extract the tendon mesh from the neutral pose MRI using classic computer vision techniques (3D Slicer [99]). Next, we non-rigidly deform the template cylinder tube to match the MRI mesh using Wrap3 [92]. As real tendons and the target MRI-segmented mesh have a non-circular cross-section, so does the resulting ICP-ed mesh. The center rod is deformed by following the surface mesh, as we can consider it to be embedded into the tube volume. In this way, we create our neutral tendon simulation rod and its surface tube mesh. For non-neutral poses, we only extract the surface of the tendon from MRI, again using 3D Slicer. These segmented meshes will be used for tendon non-rigid registration in Section 3.3.3.

3.3.3 Tendon Registration

Now, we have the tendon rod and the surface mesh in the neutral pose, and the extracted surface mesh in non-neutral poses. We need the tendon rod in these non-neutral poses. We want our tendon skinning surface mesh driven by the tendon rod to match the MRI mesh as closely as possible in each example pose. This is a standard non-rigid registration problem, but occurring on a skinned surface controlled by the rod. As is well-known, non-rigid registration requires a good initial guess to produce high-quality results. Therefore, the tendon registration at example poses is performed in three stages (shown in Figure 3.12). The first two stages create a good initial guess and the last stage performs the actual non-rigid registration.



(a) stage 1 (b) stage 2 (c) stage 3 (d) stage 3 (w/o MRI mesh)

Figure 3.12: **Stages to register tendons in non-neutral poses.** In stage 1, we simulate the tendons based on the sliding constraints that are only rigidly transforming with the closest bones. This gives a relatively correct position of the tendon. However, it does not match the MRI shape (blue wireframe). To improve the match, we first create an initial guess for our non-rigid registration (stage 2). As shown in (2), the resulting mesh more closely matches the MRI shape. In stage 3, we perform non-rigid registration, which enables us to match the MRI shape very closely. Top and bottom give different camera angles and change bone transparency for easier viewing.

Stage 1: Roughly deform the tendon to match the target example pose. We manually select a few points on the tendon rod, which we think are rigidly transforming with the closest bones. These points are treated as hooks (used only in this stage; discarded otherwise). We then slowly deform our bone rig from the neutral pose to the target example pose. In each iteration, we compute the positions of the selected points. Since they are rigidly transforming with the closest bones, the positions are easily obtainable. Then, we perform our tendon simulation to obtain the shape of the tendon using Equations 3.13-3.21. The result is depicted in Figure 3.12(a). Stage 1 takes approximately 1 minute per tendon. We note that stage 1 is needed because otherwise the initial guess for later stages can be quite bad, due to poses being very different from the neutral pose.

Stage 2: Improve the tendon by matching the centerline of the extracted MRI mesh. After the first stage is completed, the position of the tendon in this non-neutral pose is close to the tube mesh extracted from MRI. Because the tendon is thin, we need to further match the MRI mesh to guarantee the quality of the final non-rigid registration. To do so, we first select a few points (~ 10) on the approximate tendon centerline manually, based on the MRI mesh in this pose. We use these points as hooks (again only in this stage; discarded otherwise) and run our simulation, starting from the output of stage 1. The result of stage 2 is depicted in Figure 3.12(b). Stage 2 takes approximately 1 minute per tendon. Stage 2 is needed because after running stage 1, sometimes the tendon surface is completely outside of the MRI surface. This causes difficulties in ICP (incorrect determination of closest point) if one goes to stage 3 directly after stage 1.

Stage 3: Deform tendon to match the extracted MRI mesh. In the last stage, we perform actual non-rigid registration by solving the following optimization problem:

$$arg\,min_{x,n} \quad c_4 E_{\text{ICP}}(x,n) + c_1 E_{\text{pulling}}(x) + c_2 E_{\text{twist-bend}}(n) \tag{3.25}$$

$$\text{subject to} \quad (x_{i+1} - x_i)^2 - \ell^2 = 0, \quad \forall i \in [1,Q], \tag{3.26}$$

$$n_i^T (x_{i+1} - x_i) = 0, \quad \forall i \in [1,Q], \tag{3.27}$$

$$n_i^T n_i - 1 = 0, \quad \forall i \in [1,Q], \tag{3.28}$$

$$x_1 = \hat{x}_1, \qquad \text{where} \tag{3.29}$$

$$E_{\text{ICP}}(x,n) = \sum_i \left(\hat{n}_i^T \left(\left(\sum_{j \in M_i} w_{ij} T_j(x_j, x_{j+1}, n_j) \bar{p}_{ij} \right) - \hat{p}_i \right) \right)^2, \tag{3.30}$$

and $E_{\text{pulling}}(x)$ and $E_{\text{twist-bend}}(n)$ are as defined earlier. Here, $\bar{p}_{ij}$ is the unskinned rest position of tendon surface mesh vertex i in the frame of reference of tendon segment j in the neutral pose; the frame of reference of a segment is defined by the two endpoints and the normal. The mapping $T_j(x_j, x_{j+1}, n_j)$ performs the skinning transformation for segment j obtained using x_j, x_{j+1}, n_j; here, w_{ij} is the skinning weight of surface vertex i against the tendon segment j. The skinning weights were determined using inverse closest distances; we use 3 segments per vertex, and those

segments are stored into the set M_i. The position $\hat{p}_i$ is the ICP target location on the MRI mesh. It is updated after each iteration in the usual ICP fashion by finding the closest position on the MRI mesh to the current vertex position). The ICP target normal is $\hat{n}_i$, determined as the pseudonormal at the closest location on the MRI mesh. Again, we solve this problem using the interior-point method [97]. The ICP registration converged well for all our tendon groups. Figure 3.13 shows the energy of tendon non-rigid registration at each iteration, for one of our tendon groups. The output of stage 3 is our final registered tendon centerline and surface mesh, and is shown in Figure 3.12(c). As can be seen, it is smooth and closely matches the MRI mesh compared to the previous stages. We also observe that our optimization quickly converges in a few iterations, as illustrated in Figure 3.13. Stage 3 takes approximately 10 minutes per tendon.

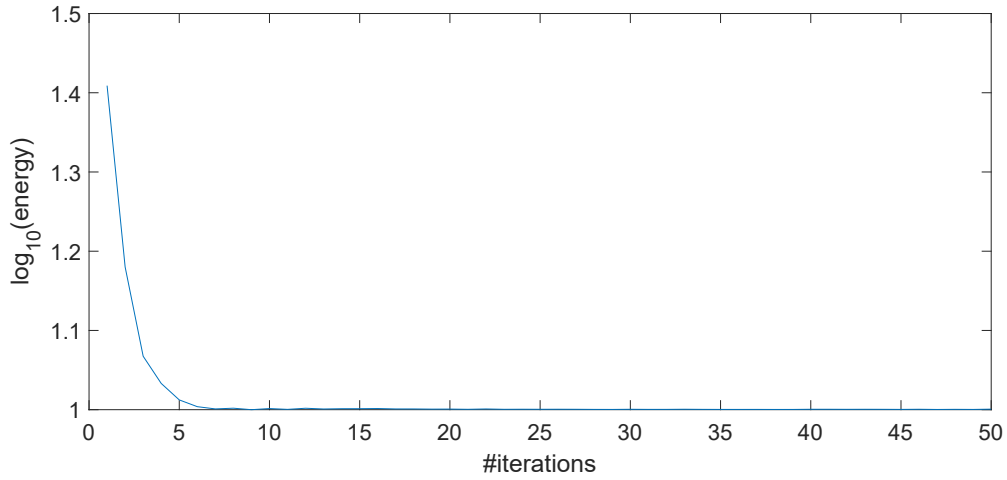


Figure 3.13: **The energy of tendon non-rigid registration at each iteration.** The curve shows the energy of Equation 3.30 at each Newton iteration when performing the non-rigid ICP on the extensor tendon of the middle figure. The optimized energy plateau is not zero because the forearm pulling force introduces an (unimportant) constant energy offset; we added a properly chosen such an offset to improve figure readability.

3.3.4 Tendon Hooks

Using the method described above, we obtain all example tendon rods. Then, we sparsely sample the tendon rod in the neutral pose; these sample points are our neutral hooks. For each hook, we assume that it is driven by N_s closest bones (we use $N_s = 2$). We then find the skinning weights w

by minimizing the distance of the skinned hook position to the example tendon rod mesh across all example poses,

$$arg\,min_w \quad \sum_{j=1}^{N} \left(\sum_{k=1}^{N_s} w_k T_{jk} \bar{p}_{ik} - \hat{p}_j \right)^2, \tag{3.31}$$

$$\text{s.t.} \quad \sum_{k=1}^{N_s} w_k = 1, \tag{3.32}$$

$$0 \leq w_k \leq 1, \forall k \in [1, N_s], \tag{3.33}$$

where T_{jk} is the example transformation for bone k in pose j, $\bar{p}_{ik}$ is the unskinned rest position of hook i to bone k, and $\hat{p}_j$ is the closest position on the rod to the hook location, in example pose j. The example rod meshes are obtained from our non-rigid registration. To solve this optimization problem, we first initialize the weight of the closest bone to 1, and the weights of the other bones to 0. Then, we compute the skinned positions in the example poses. Thereafter, we find the closest position $\hat{p}_j$ in each example pose. Next, we run our optimization to determine the skinning weights w. We repeat these steps until the change of w between two consecutive iterations is very small (relative change of 10^{-4}). After the optimization problem is solved, we store the vector between the skinned hook position and the closest point on the tendon rod as our pose-space correction in this example pose. Similarly to muscles, the pose-space vector consists of rotations of bones that drive the tendon. The representation of rotations as 3-vectors is the same as for muscles (Section 3.2.3).

3.4 Bone and Muscle Fascia

The gaps and valleys around bones and muscles cause problems (squeezing, mesh gets stuck, etc.) if one simulates the fat tetrahedral mesh directly on top of the bones and muscles. Therefore, we create two fascia layers, one wrapping muscles and bones, respectively (Figure 3.14). Each fascia is a triangle mesh. For bone fascia, we merge our bone meshes with the convex hulls of the joint space in between the bones [3]. This gives us the triangle mesh of the bone fascia. Then, we shrink the cloth material coordinates to 35% of their original size, so that the fascia tightly

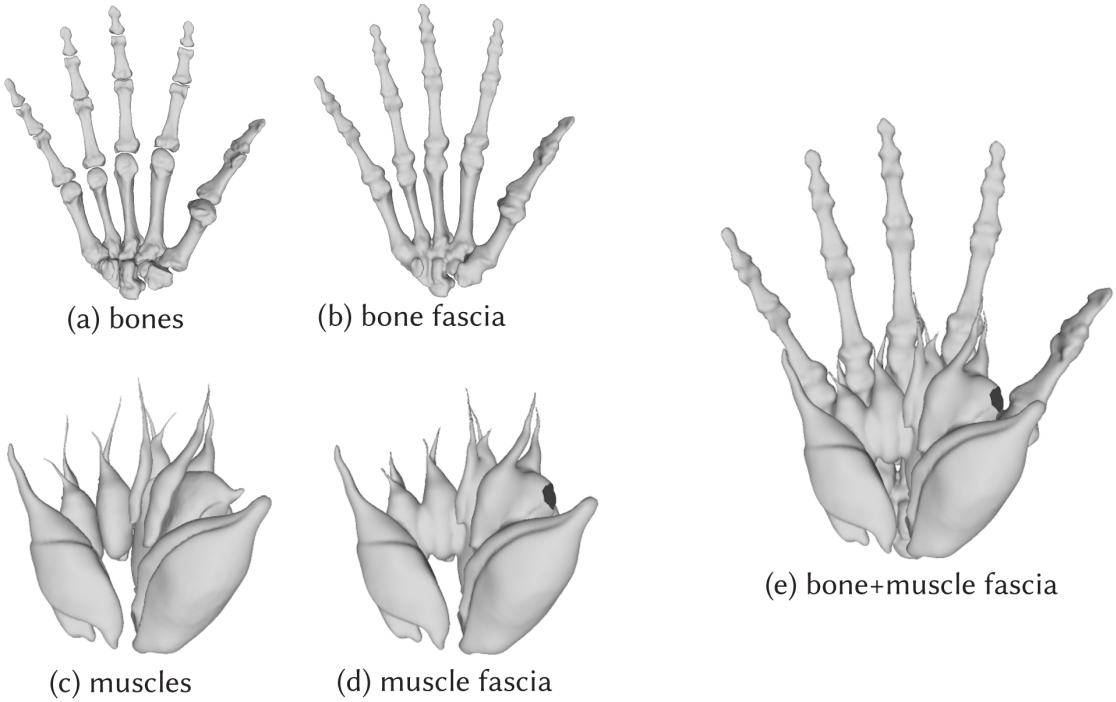


Figure 3.14: **Fascia meshes.** (a) Bone meshes, (b) corresponding bone fascia mesh, (c) muscle meshes, (d) muscle fascia mesh, and (e) both fascias.

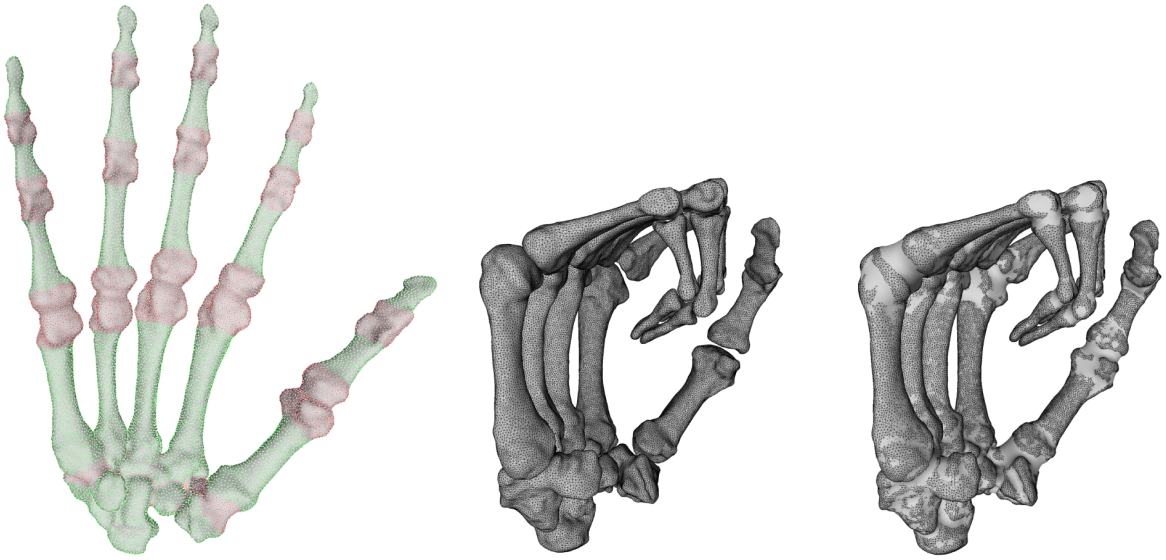


Figure 3.15: **Bone fascia.** (a) The constraints applied to bone fascia simulation. Vertices in green are attached to the closest bone using fixed constraints. Vertices in red are in contact with the bone meshes. (b, c) Bones and bone fascia in the fist pose.

wraps the bone geometry. Similar to bones, we merge all muscle surface meshes together and retain only the external envelope; this produces the muscle fascia mesh. For muscles, we shrink the cloth material coordinates to 40% of their original size. Our system is not particularly sensitive to these shrinking percentages (i.e., we could easily make the two percentages equal), as long as they are small enough. The fascia meshes for bones and muscles are shown in Figure 3.14. In each simulation timestep, we first rigidly transform the bones and then simulate the bone fascia using a cloth solver [100]. After that, we perform muscle simulation, and then simulate the muscle fascia, also using the same cloth solver.

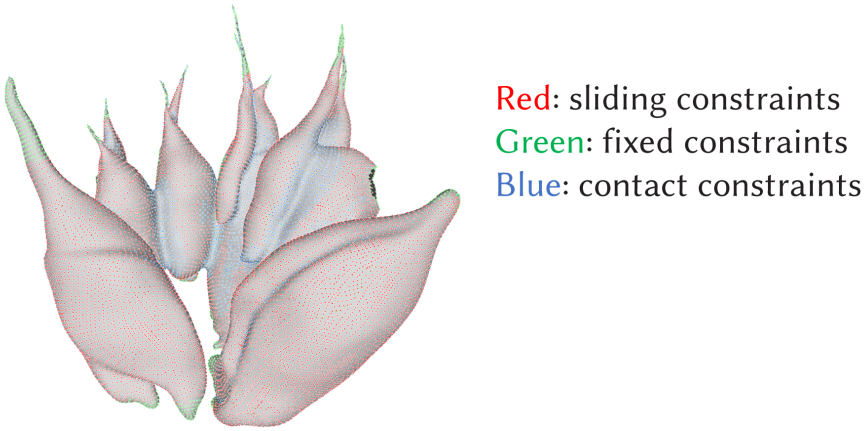


Figure 3.16: **Muscle fascia constraints.** Here, we show the constraints applied during our muscle fascia simulation. Vertices in green are attached to the muscles using fixed constraints. Vertices in red are sliding against the muscle surface meshes.

We now need to specify how the two fascias are constrained in the cloth simulation. There are three types of constraints in our model: fixed constraints, sliding constraints, and contact constraints, as shown in Figure 3.21. A fixed constraint anchors a point to a target location. A sliding constraint limits a point to travel in a (time-varying) target plane. A contact constraint penalizes penetration of a point beneath a (time-varying) contact plane. For bone fascia, the fascia vertices located away from bone heads are attached to the bone mesh with a fixed constraint, and move with the bone rigidly. The other bone fascia vertices (i.e., those at bone heads and at joints) are not attached, but are instead simulated to undergo contact with the bones' surface. The two types of vertices are selected manually, by painting on the bone fascia mesh in the neutral pose. Due to

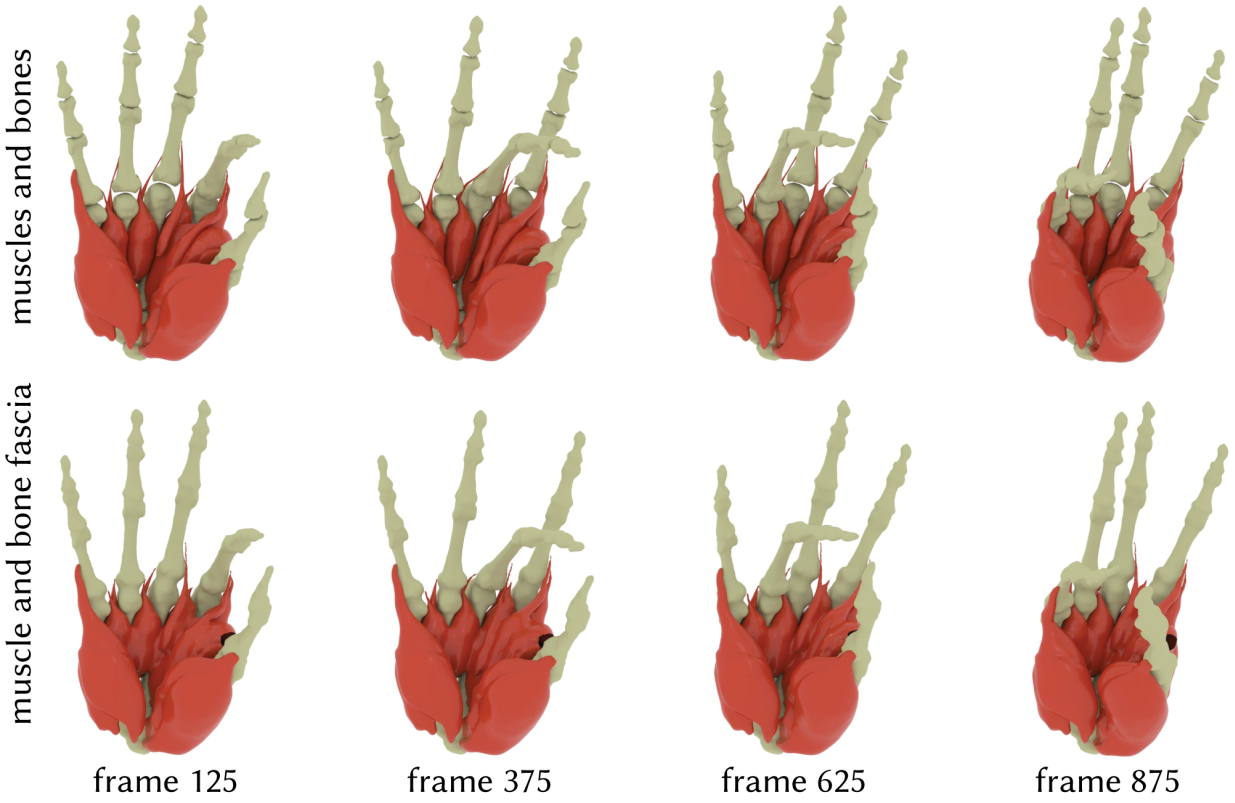


Figure 3.17: **Fascia animation.** Top: a few frames of an animation sequence of bones and muscles. Bottom: muscle and bone fascia simulation results.

the attachments and contact and shrinking of the rest material coordinates, the bone fascia mesh remains tense and does not fold as the bones move (Figure 3.15).

For muscle fascia, we use all three constraint types: fixed, sliding and unilateral contact. We use fixed constraints near muscle insertions, sliding in regions where the muscle fascia is on top of a single muscle, and contact where two muscles meet (Figure 3.16). These regions were painted manually in Maya in the neutral mesh, with smooth transitions between them. Even though such a process is manual, it is a simple matter of painting regions; we use Maya as a painting tool and export the maps to our simulator. The painting usually takes less than 10 minutes; it only needs to be done once. Representative bone and fascia animations are shown in Figure 3.17.

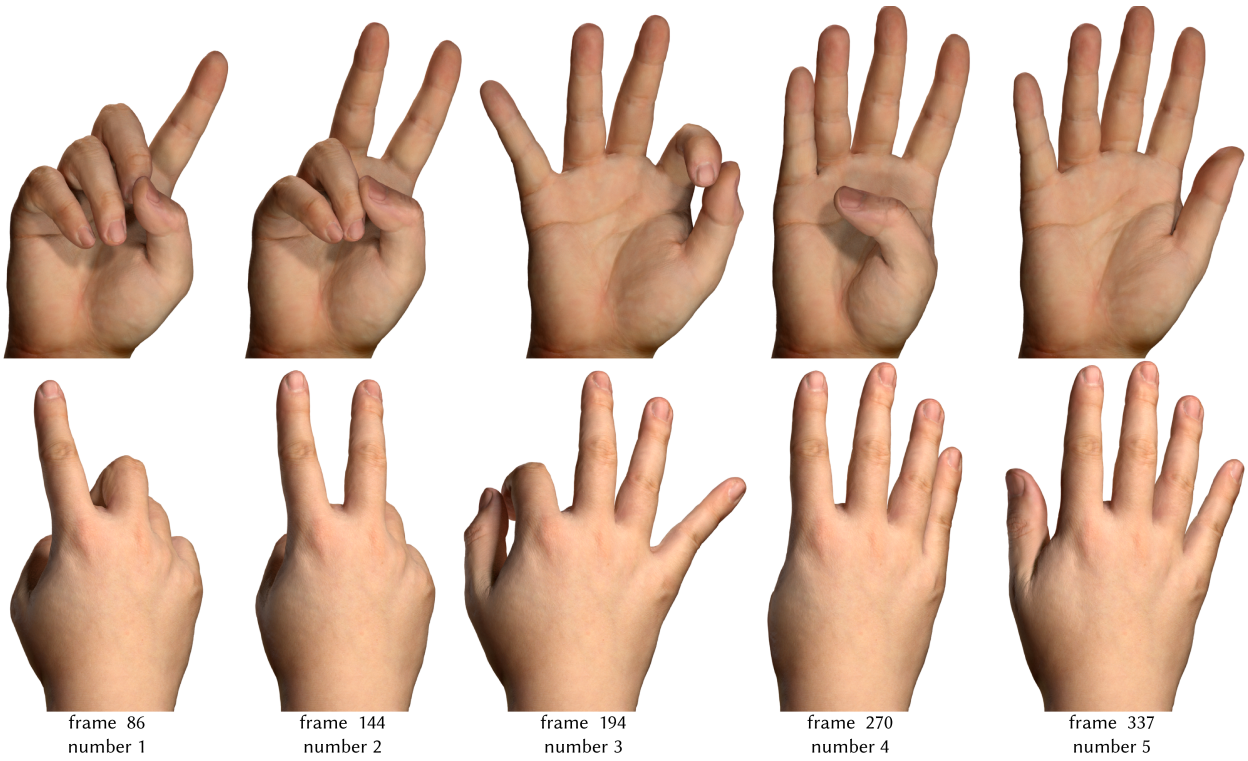


Figure 3.18: **Representative frames for motion "Numbers 1-5".** Displayed are the front (first row) and back (second row) side of the skin. Our model produces realistic skin shapes.

3.5 Fat

Because fat is a passive tissue, our fat simulation is similar to the method proposed in [3], but with a few changes. Our fat tissue is a volume bounded by two surfaces: the exterior surface (the "skin"; Figure 3.19(a)), and the interior surface (Figure 3.19(b)) that is in contact with the bones, muscles, and joint ligaments. Each of these two surfaces is topologically a sphere. The skin surface mesh is from the same subject as the MRI scans, and was obtained from the project [3]. We mesh the volume between the skin and the inferior surface using a tetrahedral mesh, and embed the two surfaces into it. We can significantly reduce the number of tetrahedra because the volumetric mesh does not have to conform to the surfaces. The tet mesh is created by first meshing the volume enclosed by the skin, and then removing all tets that are completely inside the volume enclosed by the interior surface. We assign a single material property to the fat, based on values in medical literature (Young's modulus=1 kPa) [101]. After creating the tetrahedral mesh, we nevertheless

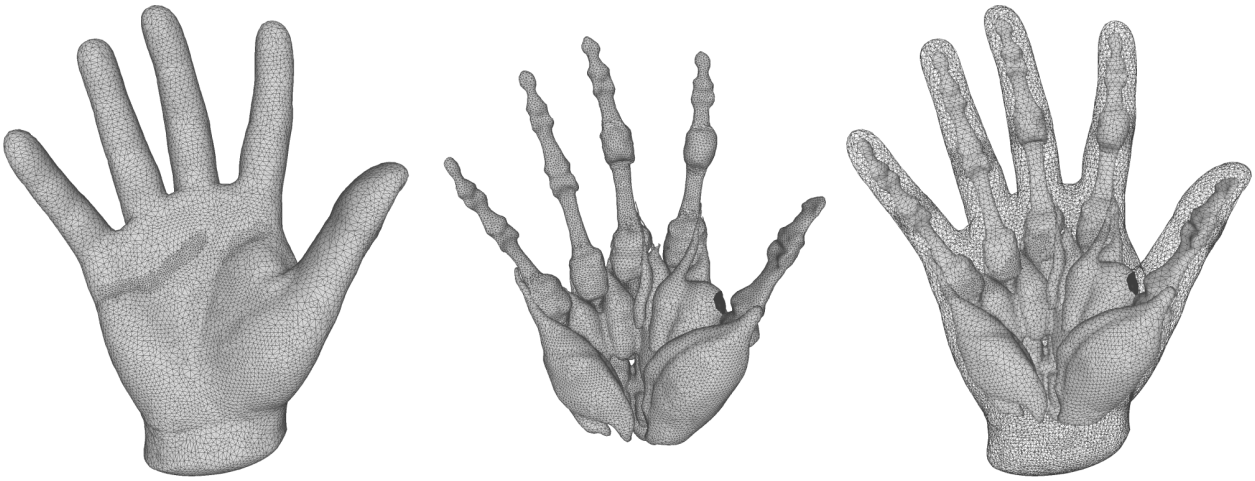


Figure 3.19: **The exterior and interior surface of the fat tissue.** (a) The exterior surface of the fat tissue is the skin. (b) The interior surface of the fat tissue is composed of the bone fascia, muscle fascia, and joint ligaments.

need to assign spatially varying materials, because some tetrahedra are only partially occupied by the fat tissue. Therefore, we weigh the Young's modulus and mass density by the volume occupation.

We then define constraints of the fat against previous layers (Section 3.5.1). For all of our six MRI example poses, and six additional non-example poses, we obtained a matching high-precision (0.1 mm) optical scan from the project [3]. The six optical scans in MRI example poses serve to optimize per-pose spatially varying tet plastic strains, to cause the FEM fat simulation to match ground truth optical scans (Section 3.5.3). The six non-example optical scans serve to test our method in non-example poses (Section 3.6, Figure 3.31).

3.5.1 Constraints for Fat Simulation

After we obtain the simulation mesh, we build the constraints between the interior surface of the fat and the muscle fascia, bone fascia, and joint ligament layers. Inspired by biomechanics, for constraining the fat to the bone fascia, we create fixed constraints at vertices near the bodies of the bones, and use contact around bone heads. For constraining fat to muscle fascia, we found that

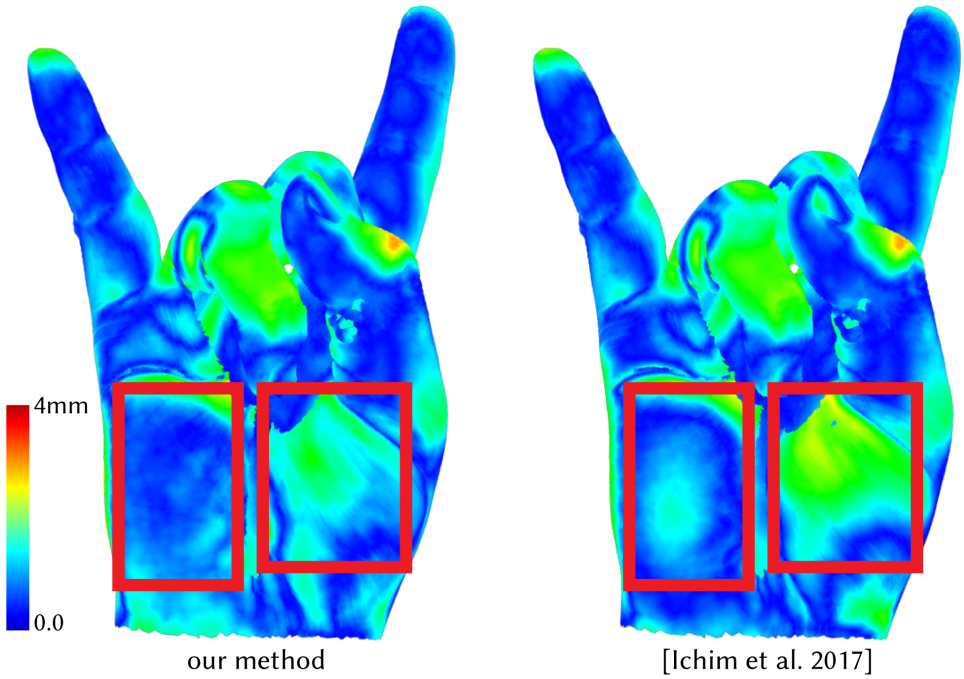


Figure 3.20: **Comparison to [52].** We compare the skin of our system and of [52] to the ground truth optical scan at the same pose. This pose was *not* used as an example pose in either system. The color maps show the distance from the ground truth mesh to the simulated skin mesh, for each method. Due to better muscle anatomical modeling, our method has a lower error in the palm area. Namely, Ichim et al. [52] treats the entire hand as a single soft tissue with spatially varying plastic strains, whereas we model each tissue separately, model sliding, and specialize the pose-space and spatially varying strains to each tissue.

the sliding constraints + contact constraints yield the best results for all vertices, except in places where the muscle is attached to bones. In such places, we attach fat to muscle fascia using fixed constraints. We place sliding constraints near the muscle fascia surface, and use contact near the valleys between muscles and/or bones. Note that if all interior fat vertices were attached to fascias using fixed constraints, this would produce visible bumps on the skin surface. We manually determined the weights for sliding constraints and contact constraints. Much like with the fascias (Section 3.4), this manual process is not very difficult. We use Maya's painting ability and then exported the maps into our simulator; the entire painting process takes under 15 minutes. After a simulation run, we optionally adjust the painted maps to improve the constraints, but these adjustments are also fast and not very frequent. Moreover, the weights are the same for all example poses. The procedure for the joint ligaments is similar to the muscles. We apply contact constraints to all

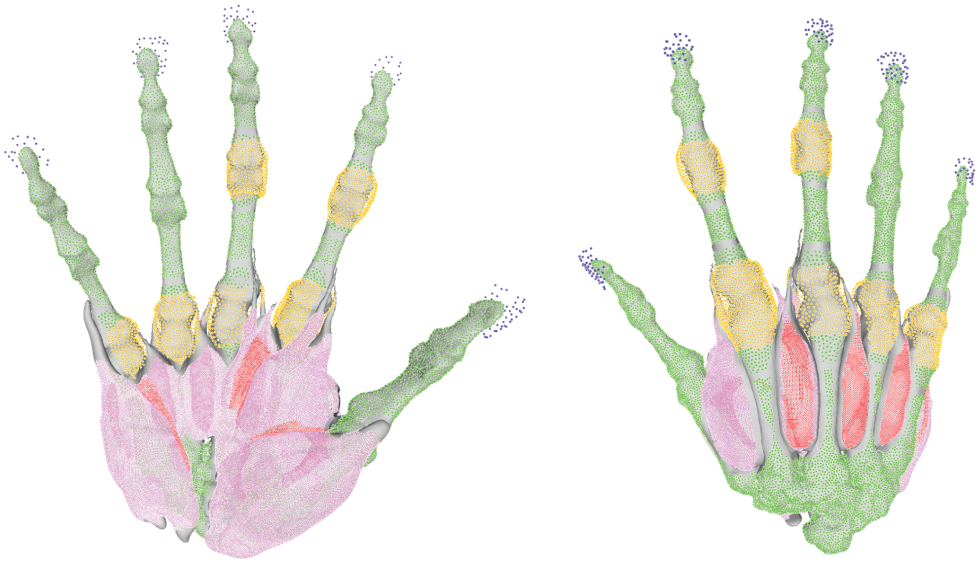


Figure 3.21: **The constraints for simulating the fat.** The fat is attached to bone and muscle fascia, ligaments, and tendons. Green dots are fixed constraints to the bone fascia. Yellow dots are contact constraints to the ligaments. Pink dots are sliding constraints against the muscle fascia, and red dots are contacts against the muscle fascia. Purple dots are points for nail rigidity.

vertices except the attachment vertices to the bones. Finally, as mentioned in [3], we also apply fixed constraints to the nails of the fingers, to keep them rigid. In addition to the described constraints on the interior and exterior surface of the fat volume, we also apply self-collision handling on the exterior surface (skin).

3.5.2 Modeling and Animating Finger Nails

Although finger nails can in principle deform too, they are much stiffer than fat and muscles, and we model them as rigid objects; their animation therefore entails finding their rigid body motion. We first attempted to models nails as embedded into the volumetric fat simulation, which produced visibly incorrect non-rigid nails. We then tried to rigidly transform the nails with the closest bone; however, this method also failed, producing a poor match to the nails seen in the hand skin optical scans in the example poses. To resolve this mismatch, we add a pose-space correction [1] to the rigid transformation of each nail.

To obtain the example rigid transformation for each nail at each example pose, we first perform hand simulation from the rest pose to each example pose, whereby each nail is rigidly transformed by its closest bone. Next, we perform non-rigid ICP registration to the ground truth skin mesh obtained using optical scanning. This is done by manually placing corresponding landmarks near and on the nails on the simulation output mesh and on the optically scanned skin mesh. By doing so, we obtain the ground truth nail mesh in the same mesh topology at each example pose. Next, we use shape matching [102] between the neutral pose nail and each example pose nail, to extract the example rigid transformation of the nail. Finally, the nail pose-space correction transformation is obtained by computing the difference between the transformation of its closest bone at the example pose and the example transformation. To ex/interpolate the pose-space correction to an arbitrary pose, we use the same interpolation method that we used for muscle plastic strains, controlled by the closest joint for each nail.

3.5.3 Fat Plastic Strains

The above setup produces simulated skin shapes that reasonably match the optically scanned ground truth mesh at each example pose, but some mismatch remains. To further eliminate this discrepancy, we apply pose-varying plastic strains to the fat layer. We stress that fat, unlike muscles, is a passive tissue and has no activation, but we re-use the plastic strains idea nonetheless, this time as a tool to steer the simulation toward optically scanned skin shapes in the example poses. We obtained optical scans using the plastic casting method of [3]. We align the optical scans with simulation as follows. Wang et al. [3] aligned MRIs and optical scans (their Section 5.2), and we use their method. Our simulations operate in the same frame as the neutral MRI scan, and are therefore also aligned. To obtain the plastic strains for each example pose, we formulate the following optimization problem:

$$arg\,min_{\mathbf{s},\mathbf{x}} \quad ||\mathbf{L}\,\mathbf{s}||^2 + \alpha \mathscr{E}_{\text{skin}}(\mathbf{x}), \tag{3.34}$$

$$\text{subject to:} \quad \mathbf{f}_{\text{e}}\big(\mathbf{F}_{\mathbf{p}}(\mathbf{s}),\mathbf{x}\big) + \mathbf{f}_{\text{c}}(\mathbf{x}) = 0, \tag{3.35}$$

where x are tet mesh vertex position, and $\mathbf{s}$ contains the plastic strains at all tets. As explained in Section 3.2.1, $\mathbf{s}$ has 6 entries per tet. Like with muscles, we use isotropic elastic materials (stable neo-Hookean material [39]). Observe that the second equation (Eq. 3.35) enforces that $\mathbf{x}$ contains the static equilibrium tet mesh vertex positions under the given plastic strains $\mathbf{s}$ and fat constraints. The first term in Equation 3.34 represents the spatial smoothness of the plastic strains, $\mathscr{E}_{\text{skin}}(\mathbf{x})$ is the ICP energy between the simulated skin and the target ground truth skin, $\mathbf{f}_{\text{e}}\big(\mathbf{F}_{\mathbf{p}}(\mathbf{s}),\mathbf{x}\big)$ are the fat elastic forces, and $\mathbf{f}_{\text{c}}(\mathbf{x})$ are the forces from all types of constraints applied to the fat. This energy essentially finds per-tet plastic strains that are spatially smooth and so that in the static equilibrium under those strains and the fat constraints, the embedded skin surface mesh matches the optical scan.

We first attempted to optimize Equations 3.34, 3.35 directly, similarly to [52], but the method failed to converge and generally did not scale when applied to our problem. To achieve a high accuracy of skin deformation (at or under a millimeter), we use a tetrahedral mesh with $\sim$ 50K vertices and $\sim$ 230K tetrahedra. On the other hand, Ichim et al. used a volumetric much with only $\sim$ 8K vertices and $\sim$ 35K tetrahedra, which produces matching errors on the order of a centimeter. We also tried using [33], but their method failed to handle the volumetric mesh at this scale, the dense correspondences against the ground truth skin, and complex constraints such as contacts and sliding.

The main difficulty in optimizing Equations 3.34, 3.35 arises due to the highly nonlinear and highly dimensional constraints. Our approach to tackle this problem is to generate a good line search direction by approximating the complex relationship between $\mathbf{s}$ and $\mathbf{x}$ (defined using static equilibrium; Eq. 3.34). Namely, observe that, starting from the current shape as a "rest shape", if one slightly perturbes vertex positions by $\Delta\mathbf{x}$, this causes a small change to the deformation gradient $\mathbf{I}+G\Delta\mathbf{x}$, where G is the gradient operator, and $\mathbf{I}$ is a vector of identity matrices at all tets. Therefore, by our definition of $\mathbf{s}$, we have $\Delta\mathbf{s} = \text{Polar}(\mathbf{I}+G\Delta\mathbf{x})$, where G is the gradient operator, and $\text{Polar}(M)$ computes the symmetric matrix in the polar decomposition of M (done at all the tets separately). Furthermore, in each iteration, $\Delta\mathbf{x}$ is usually small and $\mathbf{s}^{\mathbf{i}}$ is a constant,

i.e., we are performing polar decomposition on a nearly identity matrix, and we can approximate $\text{Polar}(\mathbf{I} + G\Delta\mathbf{x}) = \mathbf{I} + G\Delta\mathbf{x}$. This enables us to linearize the objective function of Equation 3.34 to (note that $\mathbf{L}\,\mathbf{I} = 0$)

$$arg\,min_{\Delta\mathbf{x}} \quad ||\mathbf{L}(\mathbf{s^i} + G\Delta\mathbf{x})||^2 + \alpha\mathscr{E}_{\text{skin}}(\mathbf{x}^i + \Delta\mathbf{x}). \tag{3.36}$$

This is now a quadratic energy in $\Delta\mathbf{x}$, and we can solve for $\Delta\mathbf{x}$ directly by solving a linear system. Once we obtain $\Delta\mathbf{x}$, we compute the deformation gradient of each tet as $\mathbf{I} + G\Delta\mathbf{x}$, and extract the symmetric matrices using polar decomposition. These symmetric matrices form a search direction $\Delta\mathbf{s}$ for updating the plastic strain $\mathbf{s}$ at the i-th iteration. Next, we perform a line search along $\Delta\mathbf{s}$ using the original objective function from Equation 3.34. During each line search iteration, we first compute plastic strains as $\mathbf{s} = \mathbf{s}^i + \eta\Delta\mathbf{s}$. We then perform forward hand simulation to obtain $\mathbf{x}$, and evaluate the exact energy in each line search iteration. We find η that gives the minimal energy and update $\mathbf{s}^{i+1}, \mathbf{x}^{x+1}$ using this η. Our method converges efficiently, as shown in Figure 3.22. Using our method, we successfully obtained all plastic strains for all example poses.

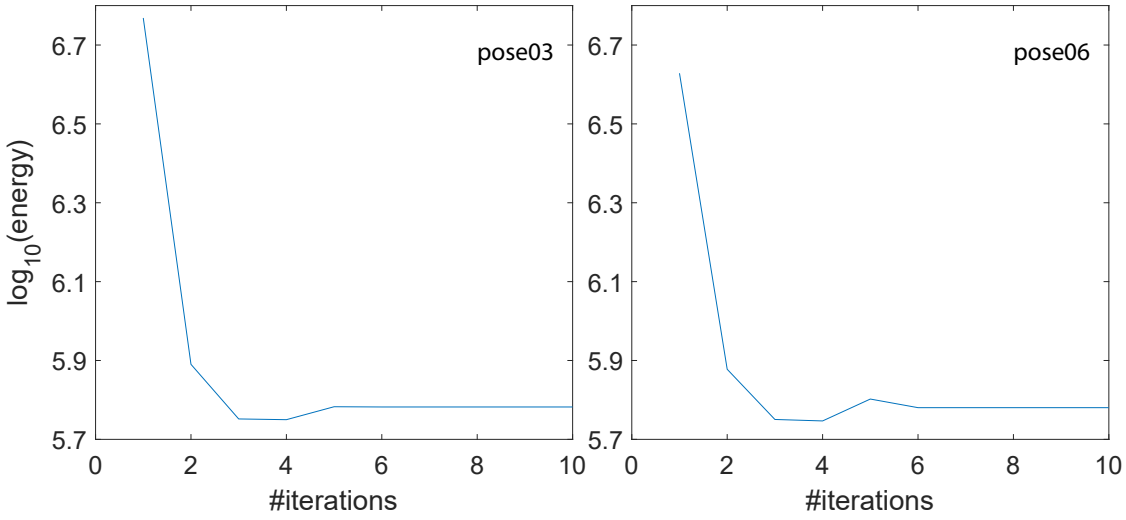


Figure 3.22: **The convergence of fat plastic strain optimization.** The plot shows the energy at each iteration during optimization, for poses 03 and 06. We can see that our algorithm quickly decreases the energy.

3.6 Results

We used an Intel Xeon(R) W-3275 CPU (56 cores @ 2.5 GHz) with 196 GB RAM; max RAM consumption was 30 GB during the simulation. We obtained the MRI data and matching optical scans from [3]. The MRI dataset (public at [103]) was created on a 3T GE MRI scanner using a PD CUBE (3D fast spin echo) sequence, with a slice thickness of 1 mm and slice spacing of 0.5 mm. Among 12 poses in the dataset, we selected six example poses ("training poses") for use in our work (Figure 3.23); the selection criterion was to pick example poses that maximize the exertion of the musculoskeletal tissues as much as possible. The remaining six dataset poses are treated as non-example poses ("testing poses"). Note that poses 01, 02, 03, 04, 05, 06 in the thesis correspond to poses 01, 02, 03, 04, 05, 12 in the dataset [103], respectively.

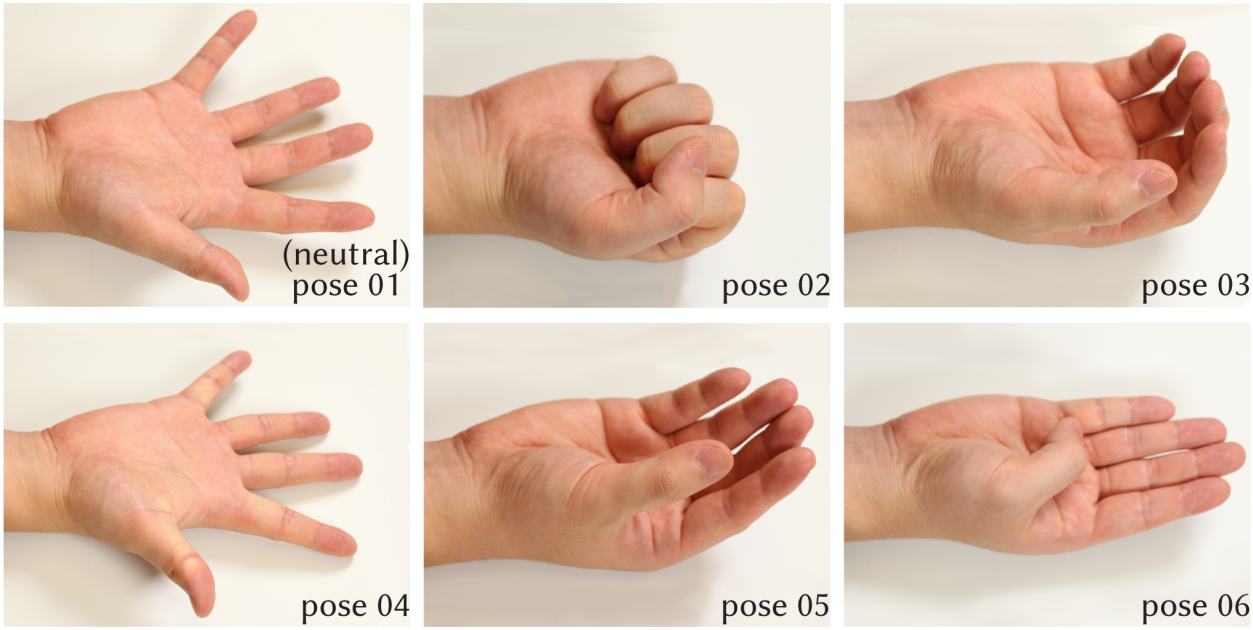


Figure 3.23: **Example hand poses used in our work.**

The specification of the tissue meshes and simulation types for each layer are shown in Table 3.2. The parameters of our simulation and optimization are shown in Table 3.3. On average, it takes 47.7 seconds to complete one step on the entire hand simulation (all layers), where the bone rig takes a negligible amount of time, the bone fascia takes 4.5 seconds (9.50%), tendons takes 6.1 seconds (1.27%), muscles takes 11.9 seconds (24.98%), muscle fascia takes 6.6 seconds (13.94%),

Table 3.2: **Specifications of meshes of all hand tissues.** We model 23 bones, 17 muscle groups, 10 tendon groups, one bone fascia, one muscle fascia and fat. For tissue types that contain more than one tissue (e.g., 17 muscles), we also list the minimal/maximal/average/sum of the number of vertices and triangles/tetrahedra, respectively. In addition, we show the method used for animating each tissue in the "sim type" column.

tissue type	sim type	# vertices	# triangles/tetrahedra
bones	skinning	788/8,921/3,381/77,771	1.572/17,838/6,759/155,450
muscles	volumetric	457/5,728/1,580/26,857	1,408/27,194/6,566/111,614
tendons	rod	233/385/321/3,210	N/A
bone fascia	cloth	23,829	47,654
muscle fascia	cloth	26,740	53,444
fat	volumetric	48,672	227,314

and the fat takes 24.0 seconds (50.30%). We successfully executed our method on five challenging hand motion sequences; three of which were keyframe-animated, and two were acquired using hand tracking. All sequences use the same simulation settings, attachments, contact parameters, PSD corrections settings, etc. In other words, all simulation layers are completely identical for all the 5 motions; the only difference is different input joint animations. This fact demonstrates that our simulation technology is robust. Performance statistics are shown in Table 3.4.

Our model generates realistic skin deformation, as shown in Figure 3.18. To evaluate the accuracy of our model, we extensively compared our simulation results with the ground truth quantitatively and qualitatively, as shown in the remainder of this Section. The extracted meshes of the muscles (Section 3.2.4) in non-neutral example poses are presented in Figure 3.24. We compare the simulated muscles in example poses with the ground truth muscle meshes extracted from MRI in those same poses (Figure 3.25). We do so in three example poses that are the most extreme among all six example poses. It is evident that the simulation results closely overlap with the ground truth meshes. Still, small errors remain. These errors are expected as our model contains inter-muscle sliding forces and contact forces, for which no precise ground truth exists. Nonetheless, our method ensures that these forces are minimized at example poses, because contact resolving cleans up the example surface meshes.

Table 3.3: **Parameters used in our simulation and optimization.** There are some additional parameters not listed here, e.g., muscle fitting γ, and the sliding and fixed constraints stiffnesses. These parameters are either determined interactively, or painted in a spatially-varying manner.

parameter	value
bone/muscle fascia E	100Pa
bone/muscle fascia ν	0.48
bone fascia UV scale	35%
muscle fascia UV scale	40%
muscle E	6,000Pa
muscle ν	0.48
fat E	1,000Pa
fat ν	0.48
tendon c_1, c_2, c_3, c_4	0.5, 1, 1, 50
muscle mass density	1,000 kg/m^3
fat mass density	500 kg/m^3
timestep	0.01s
tendon #sub-timesteps	10
muscle #sub-timesteps	1
fat #sub-timesteps	1-10
fat fitting α	1e10
muscle fitting α	1e9

We also compare our simulated tendons to the ground truth. As shown in Figure 3.26, the simulation results closely match the ground truth tendon meshes. It is expected that there will be some small error, as we only control the hooks. We did not attempt to match the orientations of the tendon rod vertices (i.e., normals) in the example poses. We also evaluated the benefits of including tendons into the overall hand simulation. Tendons are not serving only as tissues to which to attach the muscles and fat, but they also produce a more correct skin output shape, e.g., in the knuckles area (Figure 3.29, (a)).

Our hand model is superior to previous work and better matches the ground truth. We measured the accuracy improvement between a method that simulates all soft tissues using a single tet mesh [3] vs our method. For each vertex on the ground truth optically scanned surface mesh in

Table 3.4: **The time cost of each sequence.** We successfully simulated five sequences. In the "type" column, we show how the input animation was created, whereby "keyframe" refers to keyframe animation, "LeapMotion" means that the animation was created by tracking the hand of a live subject using LeapMotion [104], and "MediaPipe" means that the animation was created by tracking the hand of a live subject using Google MediaPipe [105]. Columns n_f and n_s give the number of frames and the number of simulation timesteps, respectively. Column t shows the total time cost for each sequence.

sequence name	type	n_f	n_s	t[hr]
"Close the fist"	keyframe	132	1067	14.1
"Opposition of the thumb"	keyframe	996	3381	44.8
"Performance animation"	LeapMotion	653	3525	46.7
"Numbers 1-5"	MediaPipe	360	3233	42.8
"American Sign Language"	keyframe	732	4822	63.8

each example pose, we computed the closest distance to the simulation mesh, for either method. Table 3.5 presents average distance, median distance, and max distance for each example pose. We can see that our separate-mesh method reduces the error to 40.2% for average distance, 48.9% for median distance, and 63.4% for maximal distance. For all 12 poses, our results are better in average, median and max distance. In Table 3.5, we also demonstrate that applying the fat plastic strains improves the accuracy of our model, i.e., our method without fat plastic strains is clearly worse than with plastic strains (ablation study).

We also compared our simulation with [3] visually (qualitatively), on the motion sequence used in their work. We observed a clear improvement in the palm area (Figures 3.27, 3.28). Compared to [3], our muscles are modeled using plastic strains, controlled by bone transformations, which means that we "activate" the muscles based on the hand pose; doing so improves the simulation results. Because we model fat plastic strains, tendons and ligaments, whereas previously these tissues were not modeled, we are able to match the ground truth well (Figure 3.28), including better muscle bulges and more correct hand silhouettes.

We also compared our skin deformations with the method of [52]. Note that Ichim et al. [52] was designed for facial simulation; they did not attempt hands; and their method was not designed

Table 3.5: **Comparisons between simulated and ground truth skins.** Model m_0 refers to our proposed complete model, model m_1 denotes our model without using plastic strains in the fat layer (ablation study), and model m_2 simulates all soft tissues using a single mesh (proposed in [3]). Column "%" denotes the ratio between the value in m_0 and the value in m_2. The first five rows (plus the rest shape) are example poses used for our model, whereas the last six rows are non-example (unseen) poses. All distances are reported in **millimeters**.

pose	average				median				max			
	m_0	m_1	m_2	%	m_0	m_1	m_2	%	m_0	m_1	m_2	%
2	0.23	0.94	1.35	17.0%	0.13	0.76	1.22	10.7%	2.42	4.81	4.70	51.4%
3	0.14	0.72	1.19	11.8%	0.08	0.58	0.88	9.1%	2.72	3.89	6.34	42.9%
4	0.11	0.93	1.31	8.4 %	0.07	0.77	0.86	8.1%	2.59	3.34	7.09	36.5%
5	0.22	1.02	1.21	18.2%	0.13	0.83	1.00	13.0%	3.66	4.60	5.61	65.2%
6	0.15	0.72	1.15	13.0%	0.09	0.59	0.90	10.0%	2.60	4.89	7.86	33.1%
7	0.54	0.63	0.84	64.3%	0.43	0.50	0.66	65.2%	2.70	2.85	2.70	100.0%
8	0.59	0.89	1.02	57.8%	0.44	0.69	0.80	55.0%	4.10	5.20	4.67	87.8%
9	0.79	0.97	1.08	73.2%	0.66	0.85	0.92	71.7%	3.79	4.15	4.80	79.0%
10	0.81	1.09	1.43	56.6%	0.65	0.80	1.08	60.2%	3.42	4.89	6.30	54.3%
11	0.70	0.88	1.24	56.5%	0.46	0.62	0.87	52.9%	4.54	5.63	7.69	59.0%
12	0.89	1.01	1.05	84.8%	0.72	0.86	0.78	92.3%	4.98	4.51	5.46	91.2%
All	0.47	0.89	1.17	40.2%	0.43	0.76	0.88	48.9%	4.98	5.63	7.86	63.4%

for matching internal anatomy to medical images. They modeled the entire face using a single tetrahedral mesh. When applying their method to a hand, we model the entire hand as a single soft tissue with spatially varying material properties and plastic strains in each example pose. One advantage of our method over [52] is that we produce an animation of the internal anatomy. In addition, our method produces less skin position error in the palm region of an unseen pose, compared to the ground truth (Figure 3.20). Moreover, our model generates a sharper (more correct) silhouette around the knuckles (Figure 3.29(b)).

We also evaluated how well the muscle and skin shapes are reproduced in both example poses and non-example poses, against the MRI scan. Figure 3.30 demonstrates that our simulation result closely matches the MRI scan in example poses. Our model also produces reasonable results in non-example (unseen) poses (Figure 3.31).

Finally, we visualized the MRI data using volume rendering [106]. As shown in Figure 1.2, two types of transfer functions were used to emphasize the muscles and the fat tissue, respectively. In the "muscle-emphasized" transfer function, the fat tissue was assigned low opacity and is therefore partially hidden. This visualization mode shows the internal organs very well, such as muscles and bones. In the "fat-emphasized" transfer function, every tissue except the fat was given a constant color, whereas the fat color was computed using the MRI value. As a result, we can see the nerves and veins clearly (see, e.g., the back side of the hand), and the nerves and veins nicely animate in our animation sequences.

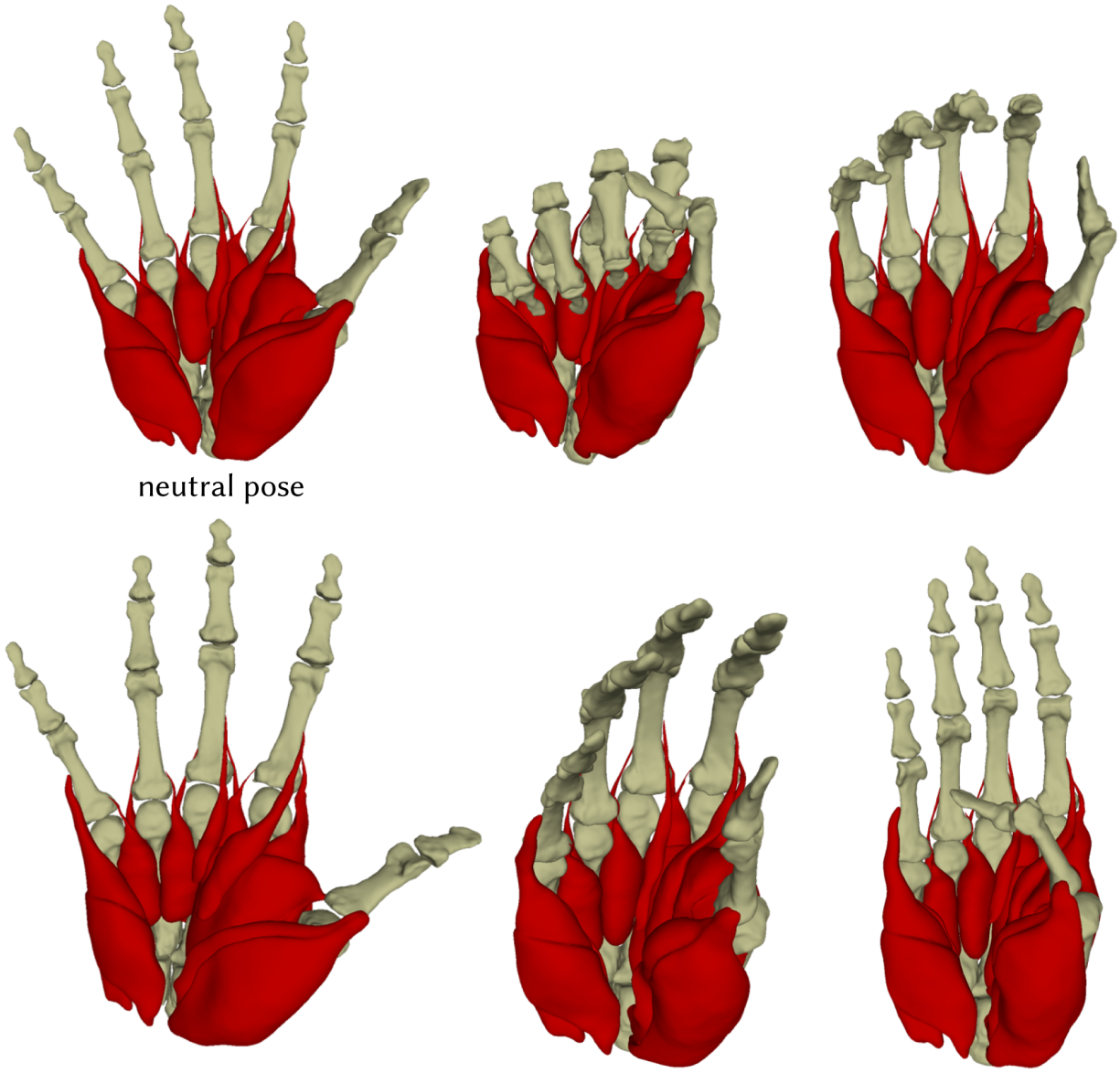


Figure 3.24: **Extracted muscle meshes in example poses,**
using the process described in Section 3.2.4.

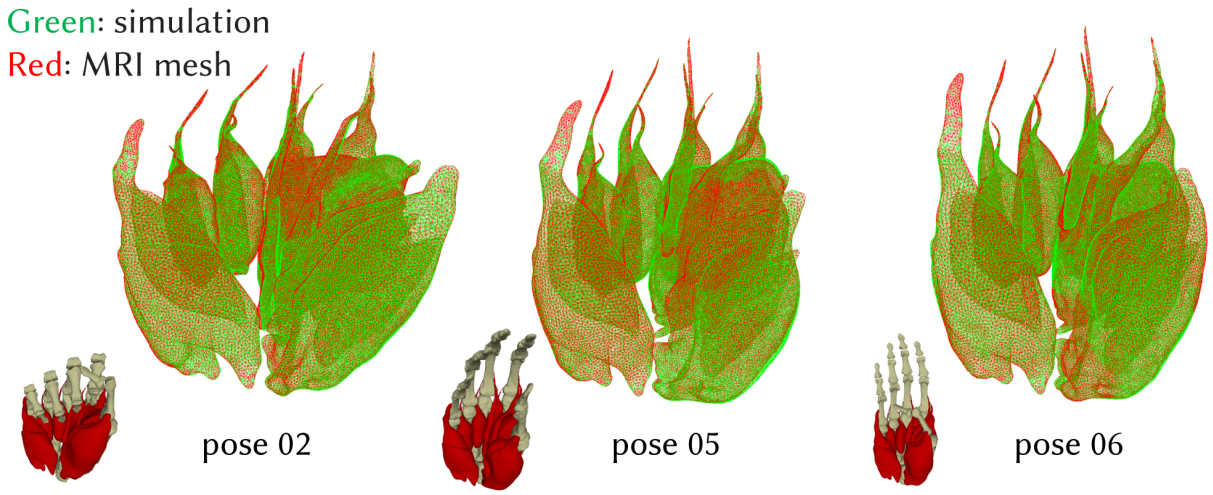


Figure 3.25: **Comparisons between muscle simulation results and the ground truth meshes in example poses.** We simulated hand muscles using our model to reach a few example poses that we believe are the most extreme poses among all six example poses. Simulation results (green wireframe) closely overlap with the ground truth meshes (red wireframe).

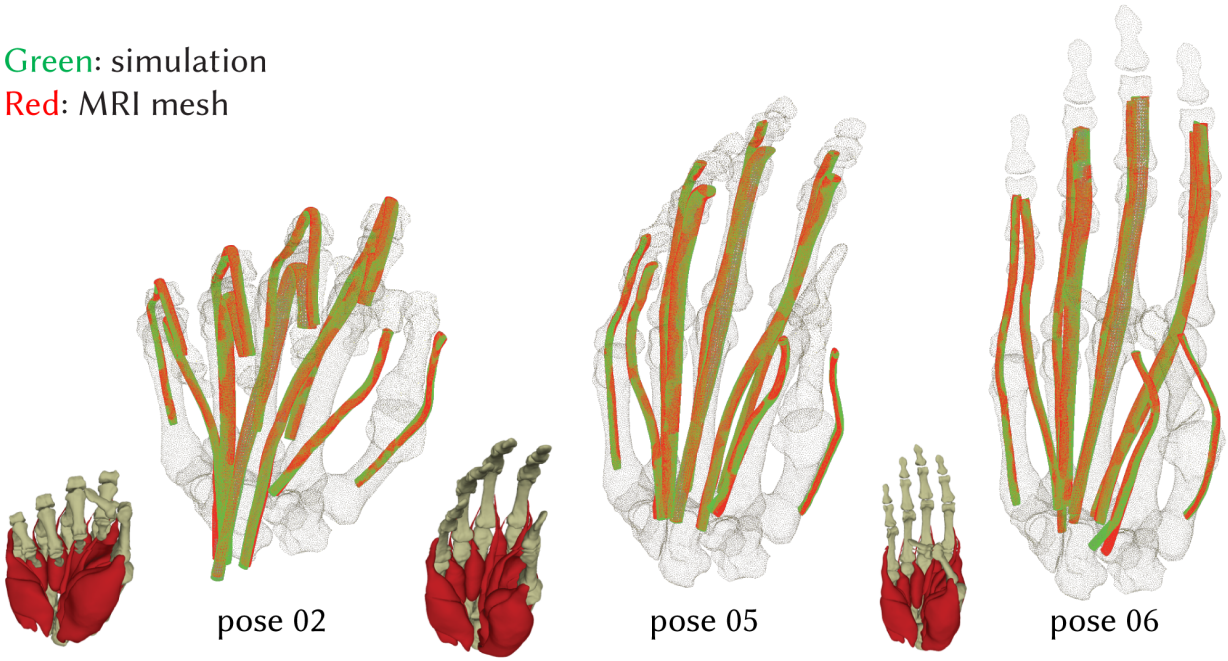


Figure 3.26: **Comparisons between tendon simulation results and the ground truth meshes in example poses.** We simulated hand tendons using our model to reach a few example poses (same as in the muscle experiment). Simulation results (green wireframe) closely overlap with the ground truth meshes (red wireframe). Note that the minor mismatch at the bottom of some tendons is because we do not have MRI data in that region.

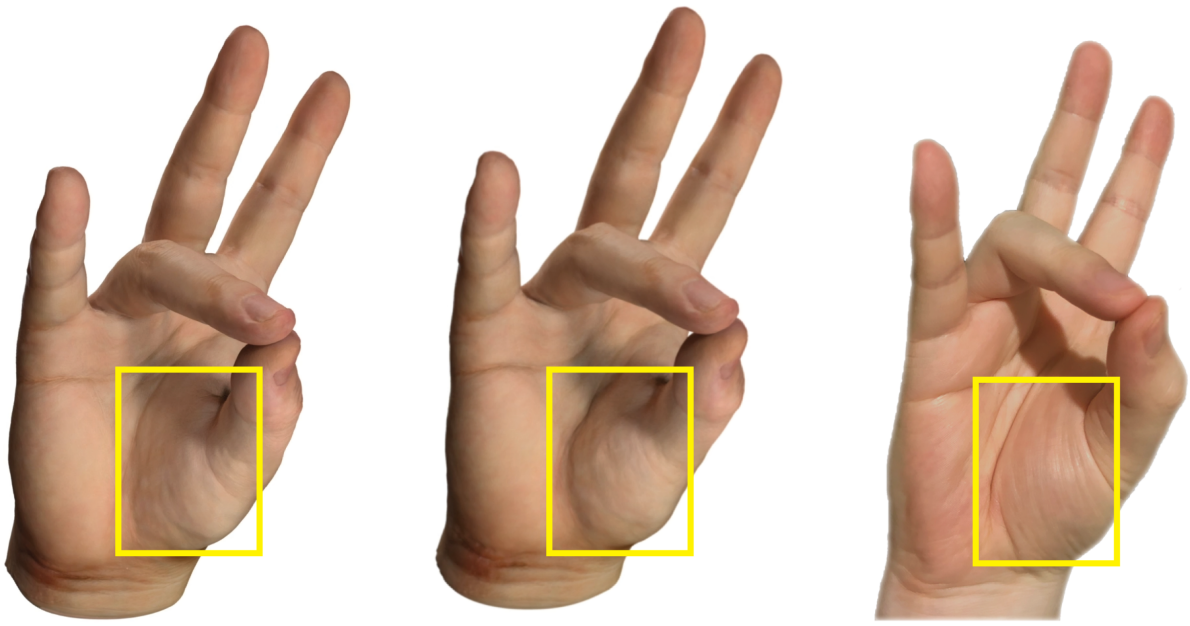


Figure 3.27: **Visual comparisons to [3].** Left: [Wang et al. 2019]. Middle: our method. Right: photograph of the same subject. The muscle at the base of the thumb is too flat in [Wang et al. 2019]. Due to modeling of pose-varying muscle activations via plastic strains, the muscle bulges much more in our method, which is closer to real-world behavior. Note that this is a **non-example (unseen)** pose for our method.

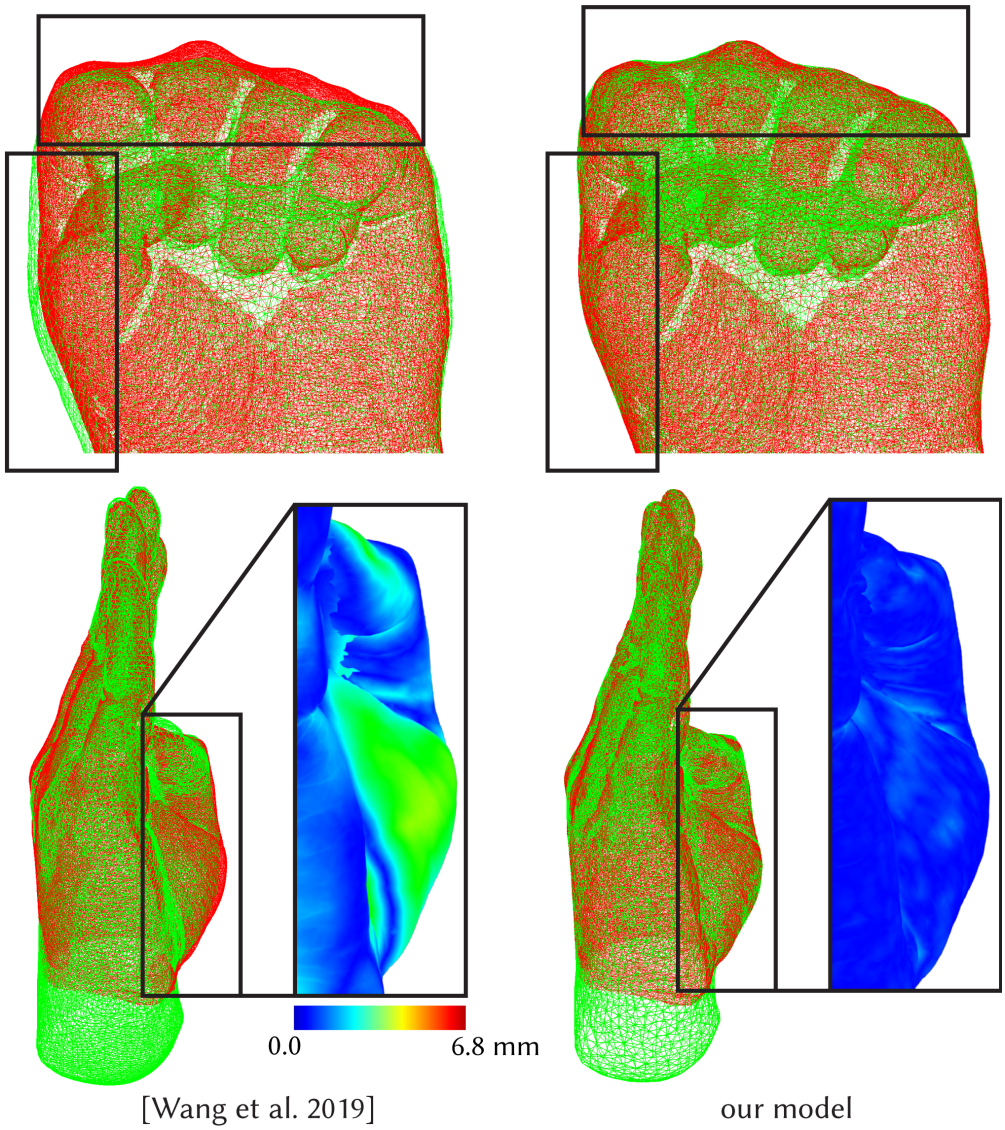
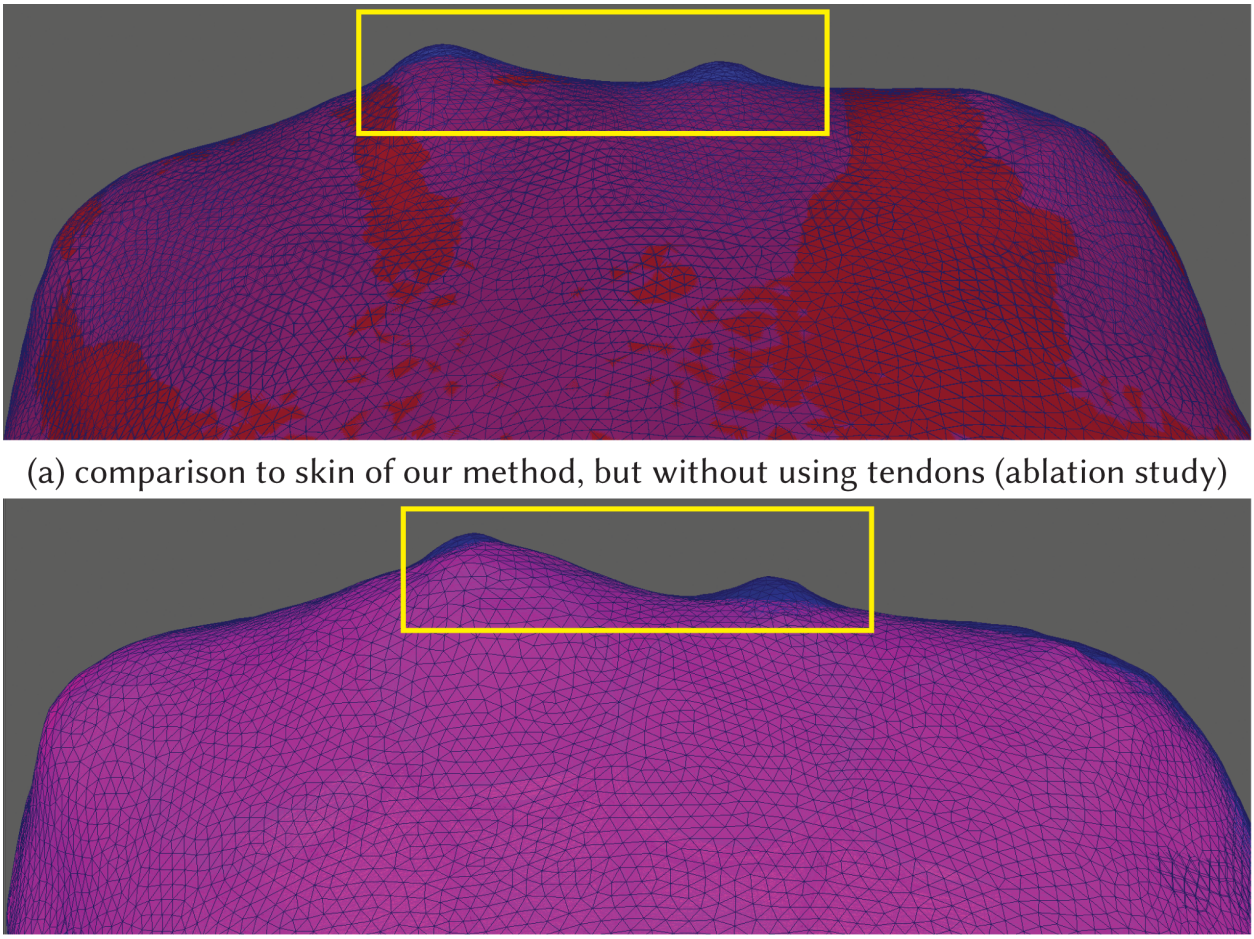


Figure 3.28: **Visual comparisons to Wang et al. [3] in example poses.** The ground truth mesh captured using a optical scanner is depicted in red wireframe, whereas the simulation results are shown in green wireframe. The palm and knuckle joint areas (highlighted by rectangles) bulge more correctly in our simulation model compared to the single-layer soft tissue model. The color-mapped inset shows the error against the optical scan quantitatively: our model produces significantly lower error in areas of significant muscle activity (e.g., muscle below the thumb).



(a) comparison to skin of our method, but without using tendons (ablation study)

(b) comparison to [Ichim et al. 2017]

Figure 3.29: **Tendon modeling is necessary to produce good knuckles.** In (a), we evaluate how the presence of tendons affects the skin output shape. In the real hand, knuckles are pronounced visual features, due to the underlying tendon which lifts the skin in the knuckle area. As highlighted in the yellow rectangle in (a), when we run our method with tendons (blue wireframe), knuckles correctly appear. When we omit the tendons from our method, knuckles are not properly resolved and the skin silhouette is flat (purple solid). (b) We also compare our method with [52]. Our model (blue wireframe) successfully captures the sharp silhouette of the knuckles, whereas [52] (purple solid) produces a visibly flatter surface, due to the missing tendons.

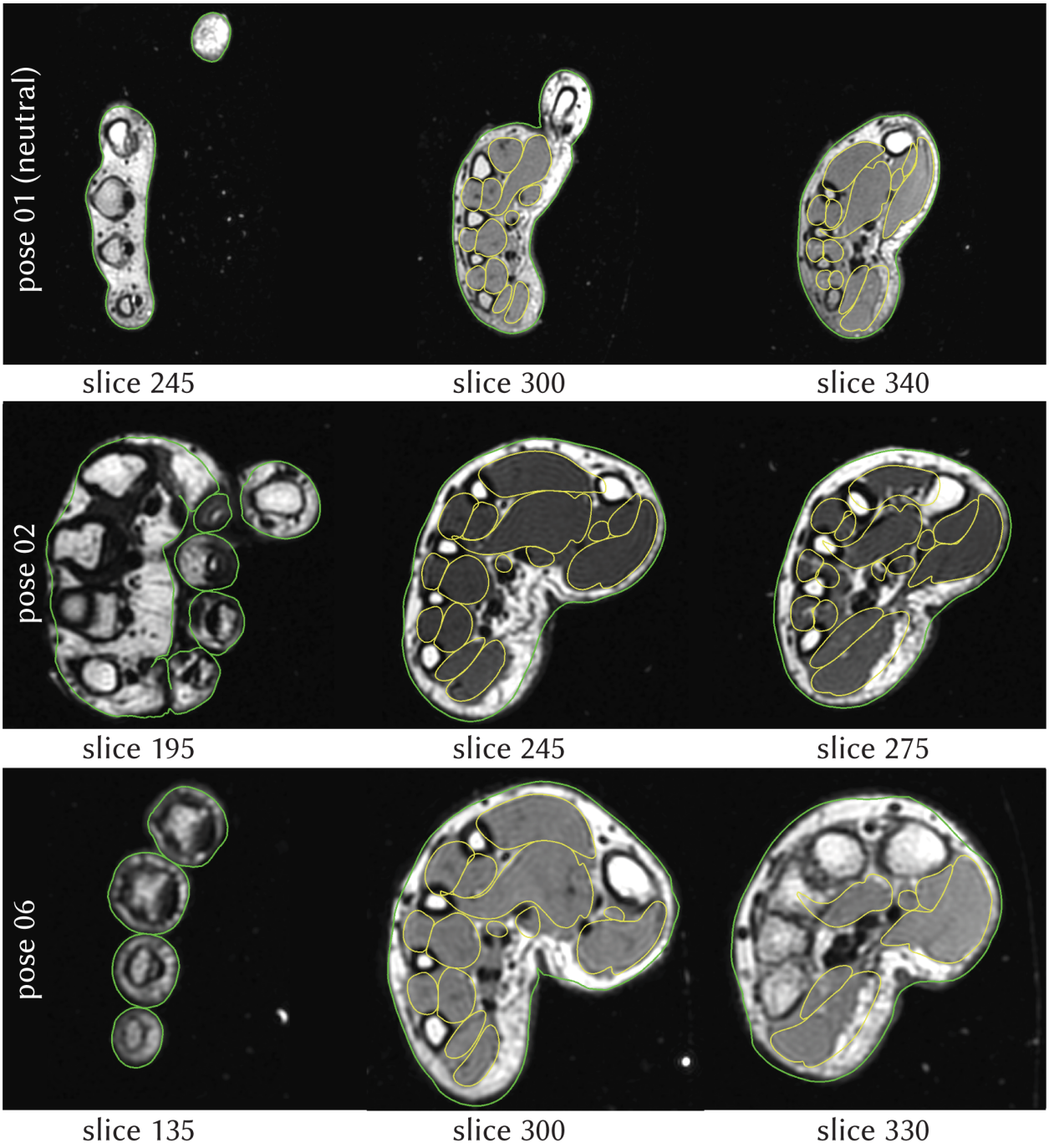


Figure 3.30: **Visual comparisons to MRI slices in example poses**. We compare our simulated skin and muscle surfaces with the MRI images in example poses 1, 2 and 6. Pose 1 is the neutral pose. Pose 2 (fist) and 6 (thumb moving to the extreme opposite palm location) are extreme example poses. We intersect our surface meshes with the MRI slices. The intersections between skin and MRI slices are shown in green, and between muscles and MRI slices in yellow. Observe that the contour lines very closely match the anatomical structures seen in MRI, both for the skin and muscles.

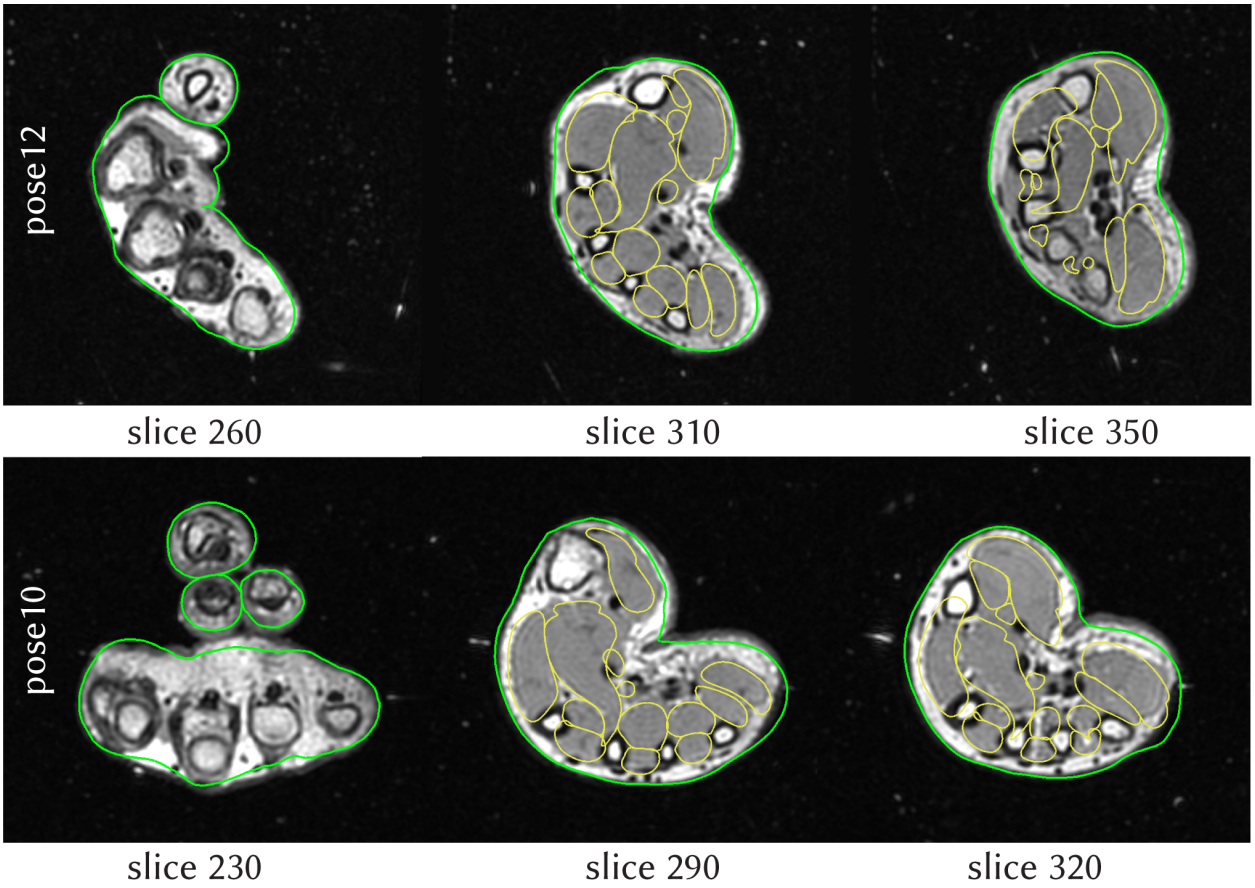


Figure 3.31: **Visual comparisons to MRI slices in non-example poses**. We compare our simulated skin and muscles with the MRI images in two non-example poses. The intersections between skin and the slices are shown in green, and between muscles and the slices in yellow. We can observe a reasonably close match to the MRI data for both the skin and the muscles.

Chapter 4

Multi-Resolution Real-Time Deep Pose-Space Deformation

4.1 Overview

We assume that we are given a neutral-pose mesh at multiple LODs $\mathscr{M}_1, \mathscr{M}_2, \ldots, \mathscr{M}_r$, where each $\mathscr{M}_i$ is a triangle mesh, and level 1 is the coarsest and r is the finest. Let n_i be the number of vertices in $\mathscr{M}_i$. We assume that the LODs are generated using a mesh simplification algorithm that can define the prolongation and restriction operators between adjacent levels, for example, as described in [11]. Thus, we first create our restriction (coarsening) $\mathscr{C}_i \in \mathbb{R}^{3n_{i-1} \times 3n_i}$ and prolongation (upsampling) $\mathscr{U}_i \in \mathbb{R}^{3n_{i+1} \times 3n_i}$ operators (Figure 4.1), following [11]. Here, $\mathscr{C}_i$ coarsens any scalar field from $\mathscr{M}_i$ to $\mathscr{M}_{i-1}$, and $\mathscr{U}_i$ upsamples any scalar field from $\mathscr{M}_i$ to $\mathscr{M}_{i+1}$; we have $\mathscr{C}_{i+1}\mathscr{U}_i = 1_{\mathscr{M}_i}$.

We also assume that we are provided with a joint hierarchy and skinning weights on the finest level $\mathscr{M}_r$. We use the coarsening operators to coarsen the skinning weights from $\mathscr{M}_r$ to all coarser LODs.

We assume that we are given training data (p^j, u_r^j) consisting of N frames, $j = 1, \ldots, N$, whereby $p^j \in \mathbb{R}^P$ is a pose, and $u_r^j \in \mathbb{R}^{3n_r}$ gives the displacements of all vertices away from the neutral-pose mesh $\mathscr{M}_r$. In our examples, the training shapes come from two types of volumetric FEM rigs, namely soft-tissues surrounding volumetric bones [37] and musculoskeletal FEM rigs [107]. However, our method makes no explicit assumption on the source of quality mesh deformations, and could in principle be applied, e.g., even to 4D optical scans followed by mesh

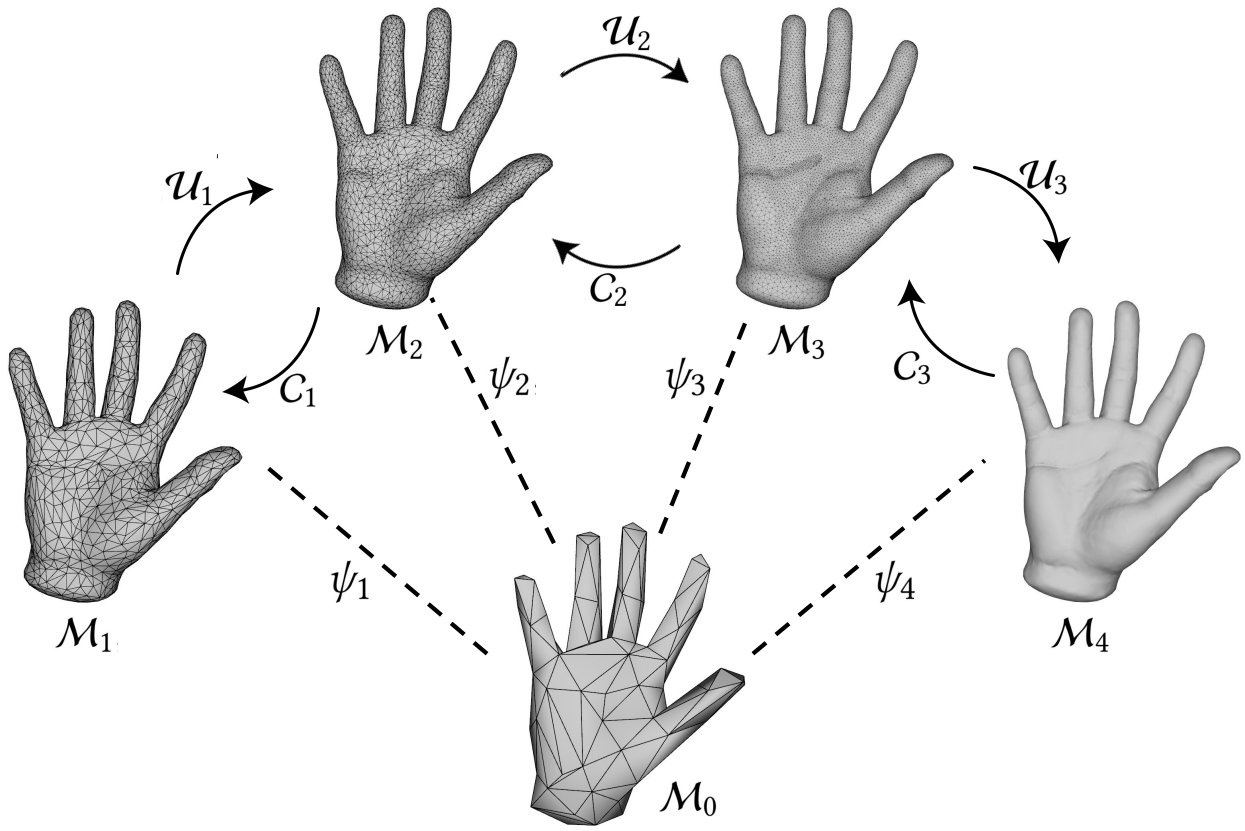


Figure 4.1: **Multi-resolution mesh hierarchy and the upsampling and coarsening operators.** Level 0 is only used for defining our "basis functions" for shape deformation (Section 4.3).

registration. The pose vector p^j contains the joint angles of the joint hierarchy, represented using Euler angles.

We transform the training shapes into the pre-skinning space using inverse skinning, and then use these shapes to train level-specific and spatially localized neural networks (Section 4.2). This is done incrementally so that the neural networks at each level only add the deformation detail introduced at that level (Figure 4.2). At runtime, given a pose p, the networks then produce progressive pre-skinning displacements on meshes $\mathscr{M}_1, \mathscr{M}_2, \ldots, \mathscr{M}_r$ until either the computation is terminated, or the last level has been computed. Finally, by applying skinning, we compute the output shape for pose p. Our output shapes reasonably re-create the training data in the training poses, and interpolate and extrapolate it to unseen poses. Of course, outside of the training ROM, our method under-performs (Figure 4.14).

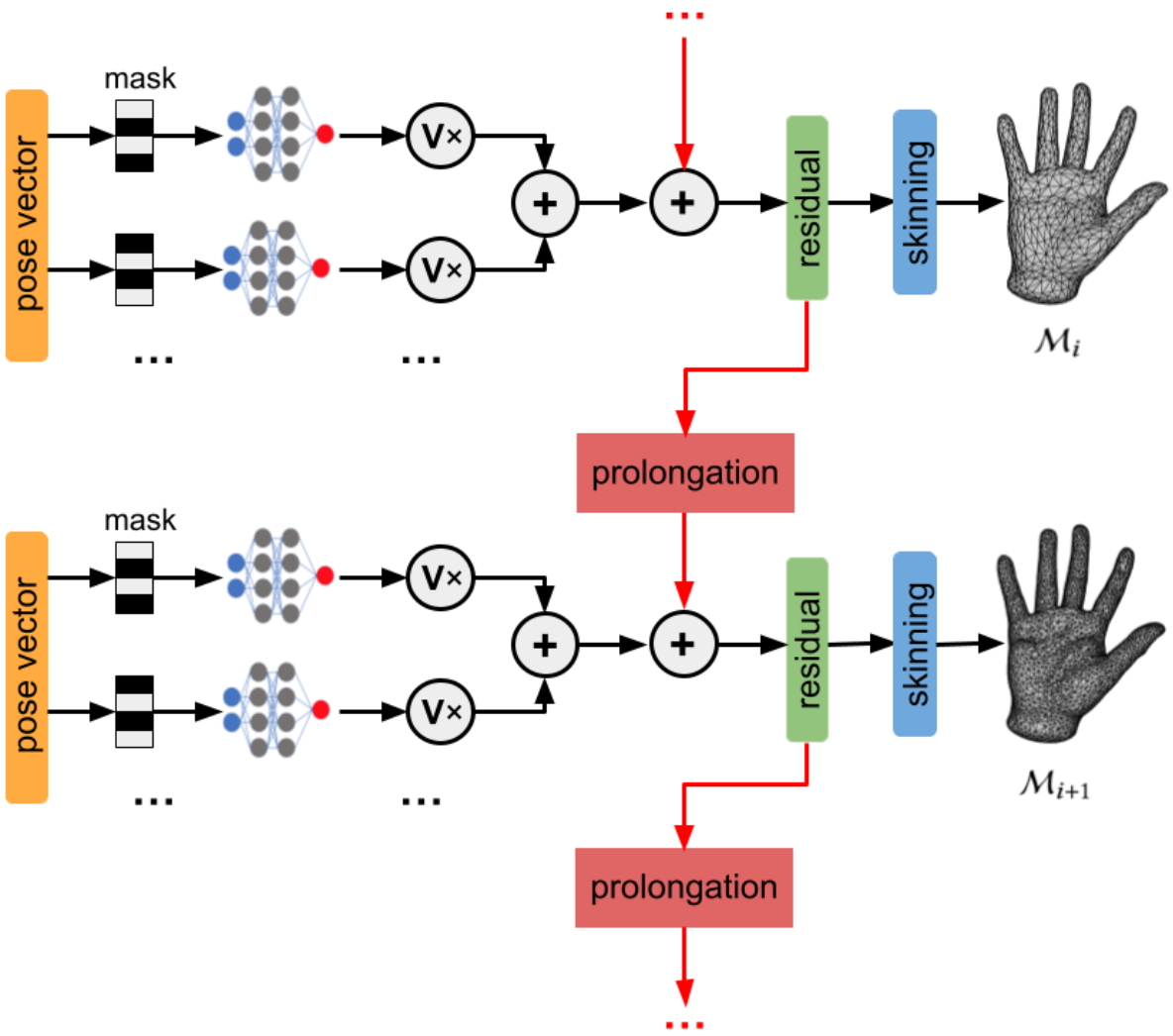


Figure 4.2: **Multi-resolution pose-space deformation.** Pre-skinning displacements are constructed progressively, level by level, using a set of spatially localized neural networks at each LOD. The mask exists because only a part of the pose enters the neural network, determined using perturbation analysis on each joint as explained in [10].

4.2 Multi-resolution pose-space deformation

As is commonly done [1, 2, 10], our method operates completely on pre-skinning displacements. At runtime, it constructs pre-skinning displacements in a given pose; and skinning then only transforms those displacements into the world-space. Given training poses p^j and vertex deformations u_r^j at the finest mesh level of detail $\mathscr{M}_r$, for $j = 1, \ldots, N$, we first invert skinning to convert them to the pre-skinning space, producing pre-skinning displacements X_r^j. This is done by computing the skinning transformation (consisting of a 3x3 linear matrix and a translation) at each vertex, and X_r^j

is then obtained by applying its inverse to the world-coordinate training vertex position, followed by subtraction of the undeformed vertex position. We then coarsen each X_r^j to each level of our hierarchy, using the coarsening operators $\mathscr{C}_i$, producing pre-skinning displacements X_i^j, for levels $i = r, \ldots, 1$.

At each level i, our goal is to generate a deformation function F_i that maps pose p to a pre-skinning displacement X on $\mathscr{M}_i$. This is *not* done directly by using X_i^j as the training data, because this would not leverage the work already done at coarser LODs. Instead, we train our F_i by leveraging the already trained F_{i-1}, i.e., we train deformation functions in the order $F_1, F_2, \ldots, F_r$, using "induction". Suppose F_{i-1} is already trained and we now need to train F_i. Given a training pose p^j, we first evaluate $F_{i-1}(p^j)$, upsample it to level i using the upsampling operator, and subtract it from X_i^j,

$$Y_i^j = X_i^j - \mathscr{U}_i F_{i-1}(p^j). \tag{4.1}$$

Note that $\mathscr{U}_i F_{i-1}(p^j)$ is the predicted pre-skinning displacement, on the mesh of level i, as predicted by the coarser levels $1, \ldots, i-1$, and Y_i^j is the "residual" deformation detail added on level i. It is this residual deformation detail that is approximated by the neural networks at level i. Specifically, we use neural networks to construct a deformation function N_i that maps an arbitrary pose p to a residual Y_i. We then compute our pre-skinning displacements at level i as

$$F_i(p) = \mathscr{U}_i F_{i-1}(p) + N_i(p). \tag{4.2}$$

The "induction" starts ($i = 1$) without any previous level; i.e., $N_1(p)$ is trained directly using the training shapes X_1^j.

At runtime, given a new pose p, we then first evaluate $F_1(p) = N_1(p)$, and then use Equation 4.2 to compute $F_2(p), F_3(p), \ldots$. This progressively reconstructs the pre-skinning shape at finer and finer LODs. This process is interruptible, permitting us to trade the output mesh quality for speed, and continues until the desired LOD is reached. Finally, we apply skinning (at any required level)

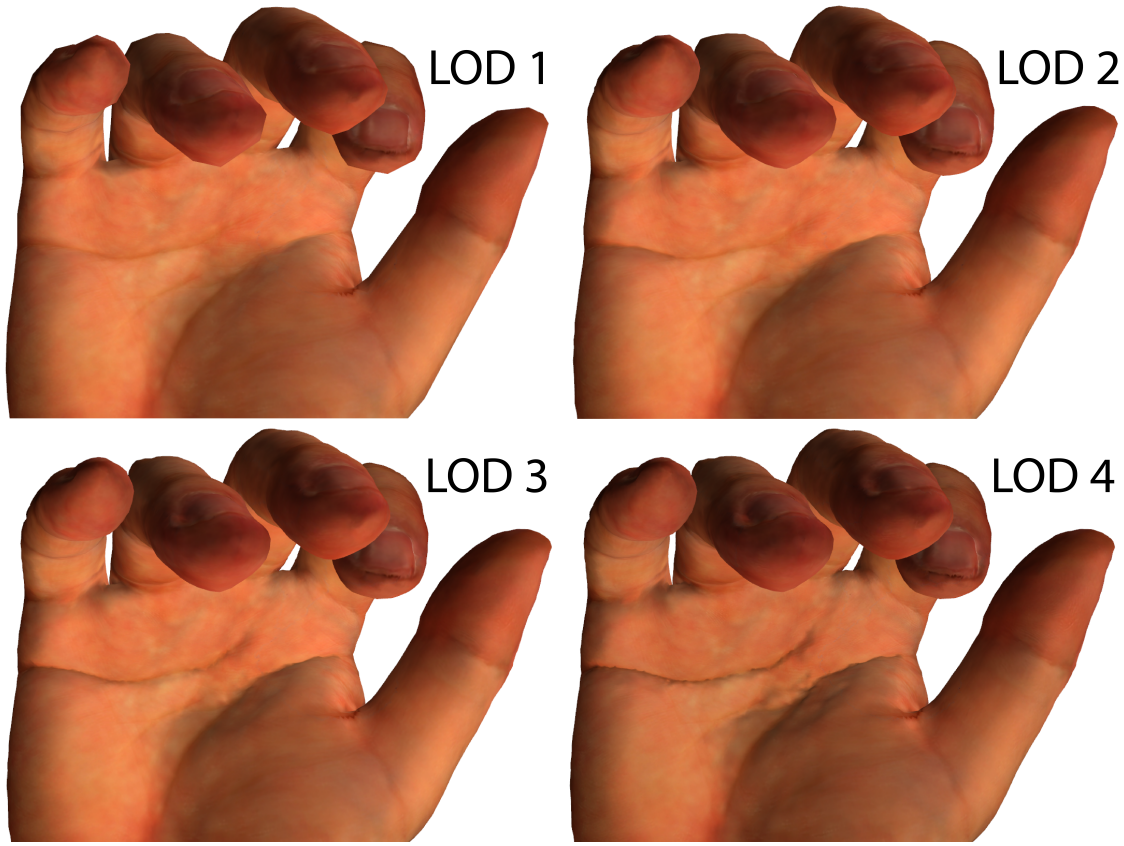


Figure 4.3: **Hand shape computation and real-time rendering at multiple LODs.** Phong shading + texture mapping, dynamic normals. Observe the increasing geometric detail seen in shading as LOD is increased. At lower LODs, geometric detail is lost and only texture mapping remains.

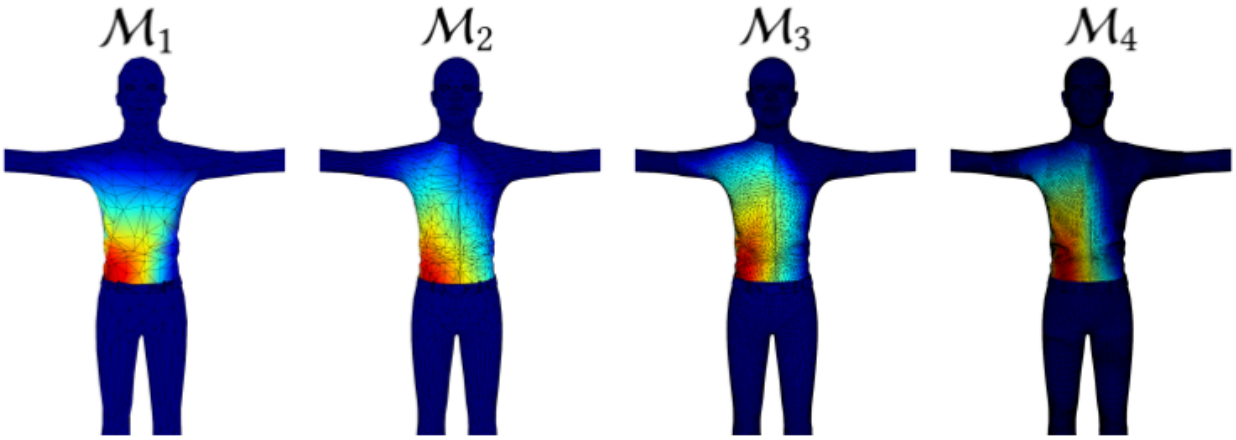


Figure 4.4: **Multi-resolution shape function.** We show a representative shape function at several LOD levels.

to compute the output character shape. Figure 4.3 demonstrates this runtime process, combined with real-time rendering.

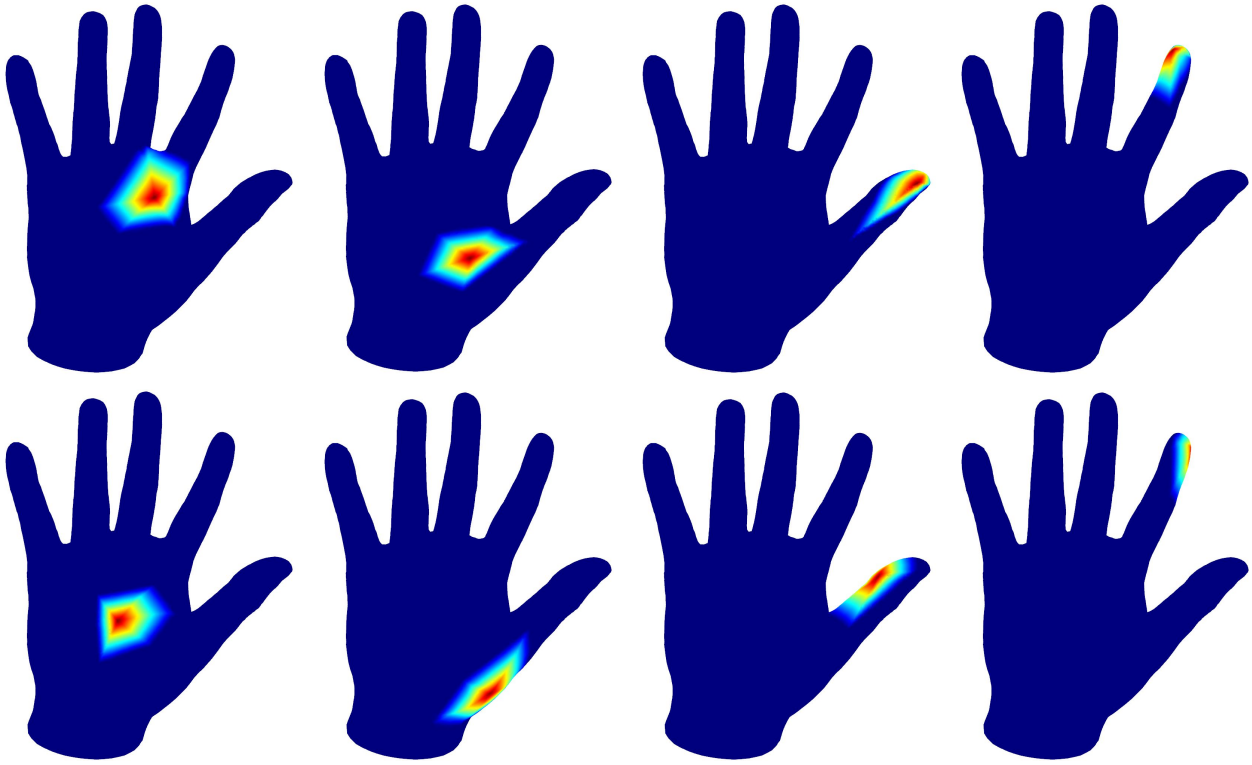


Figure 4.5: **A few basis functions** on $\mathscr{M}_4$ (not the complete set).

4.3 Neural deformation functions

We now describe how we create our deformation functions N_i at each LOD level i. We do this by training a set of spatially-localized neural networks, using the concept of a mesh partition of unity. A partition of unity on $\mathscr{M}_i$ is a set of smooth non-negative scalar functions $\psi_i^k : \mathscr{M}_i \to \mathbb{R}$ that add to 1 at each vertex, $\sum_k \psi_i^k = 1_{\mathscr{M}_i} \equiv [1,\ldots,1]^T$ (1 repeated n_i times). Partitions of unity are of course not new, see, e.g. [108, 109]; our contribution is to combine them with neural networks and a multi-resolution scheme.

We first create a partition of unity on the coarsest resolution level, as follows. First, observe that column k of the upsampling operator $\mathscr{U}_i$ gives the "influence" of vertex k of $\mathscr{M}_i$ to vertices on $\mathscr{M}_{i+1}$. This column can be seen as a non-negative scalar field defined on vertices of $\mathscr{M}_{i+1}$; we will denote it by $\mathscr{U}_i^{(k)}$. Observe that the field is localized because the influence of each vertex of $\mathscr{M}_i$ onto mesh $\mathscr{M}_{i+1}$ is spatially limited. To partition the unity on $\mathscr{M}_1$, we execute the mesh simplification algorithm on $\mathscr{M}_1$, producing an even coarser LOD $\mathscr{M}_0$. This LOD will not actually

be constructed at runtime, and will not have any neural networks; we will only use it to define our partition of unity. Namely, our partition of unity on $\mathscr{M}_1$ consists of n_0 scalar fields $\psi_1^k = \mathscr{U}_0^{(k)} \in \mathbb{R}^{3n_1}$, where $k = 1, \dots n_0$. We then prolong each scalar function ψ_1^k to all the resolution levels $\mathscr{M}_2, \dots, \mathscr{M}_r$, using upsampling operators $\mathscr{U}_1, \dots, \mathscr{U}_{r-1}$ (Figure 4.4)

$$\psi_i^k = \mathscr{U}_{i-1} \psi_{i-1}^k \in \mathbb{R}^{3n_i}, \quad \text{for } k = 1, \dots, n_0, \text{ and } i = 2, \dots, r. \tag{4.3}$$

Partition of unity proof : We now prove that the resulting functions ψ_i^k are still a partition of unity at each level; we call them "basis functions" (of each level) because they define a linearly independent basis of a subspace of scalar function on that level (Figure 4.5). We proceed by induction. Suppose ψ_i^k are a partition of unity of $\mathscr{M}_i$, i.e., $\sum_{k=1}^{n_0} \psi_i^k = [1 \dots 1]^T$ (n_i 1s). Then,

$$\sum_{k=1}^{n_0} \psi_{i+1}^k = \sum_{k=1}^{n_0} \mathscr{U}_i \psi_i^k = \mathscr{U}_i \sum_{k=1}^{n_0} \psi_i^k = \tag{4.4}$$

$$= \mathscr{U}_i [1 \dots 1]^T \ (n_i \text{ 1s}) = [1 \dots 1]^T \ (n_{i+1} \text{ 1s}). \tag{4.5}$$

Last equality holds because each row of $\mathscr{U}_i$ sums to 1, for all i; this is because the sum of influences of vertices of $\mathscr{M}_i$ to any particular vertex of $\mathscr{M}_{i+1}$ must be 1. By the same argument, ψ_1^k (columns of $\mathscr{U}_0$) are also a partition of unity (base induction case). ■

Spatially-localized neural networks At each LOD level $i = 1, \dots r$, we use the basis functions ψ_i^k, $k = 1, \dots, n_0$, to define n_0 spatially-localized neural networks of level i; each neural network predicts the residual in the spatially localized region of the support of ψ_i^k. The construction below is repeated separately for each $k = 1, \dots, n_0$. We start with the residuals $Y_i^j \in \mathbb{R}^{3n_i}$ (Equation 4.1), for $j = 1, \dots, N$. We then form the training shapes $\hat{Y}_i^{j,k} = \psi_i^k Y_i^j \in \mathbb{R}^{3n_i}$, where we have multiplied the residual of every vertex with the value of ψ_i^k at that vertex; this localizes the residuals to the

support of ψ_i^k. We then perform dimensional reduction on $\hat{Y}_i^{j,k}$ using PCA along the index j [2, 10], expressing

$$\begin{bmatrix} \hat{Y}_i^{1,k} & \hat{Y}_i^{2,k} & \dots & \hat{Y}_i^{N,k} \end{bmatrix} \approx V_i^k \begin{bmatrix} z_i^{1,k} & z_i^{2,k} & \dots & z_i^{N,k} \end{bmatrix}, \tag{4.6}$$

where $V_i^k \in \mathbb{R}^{3n_i \times d_i}$ is the PCA basis matrix, $z_i^{j,k} \in \mathbb{R}^{d_i^k}$, and $d_i^k \geq 0$ is the retained dimension of region k on level i (determined as explained in the following paragraphs).

Here, in notation, we indicated that the number of rows of V_i^k is $3n_i$, i.e., three DOFs for each vertex of $\mathscr{M}_i$, but of course due to the finite support of ψ_i^k, many (most) rows of V_i^k are zero, and this is exploited in our work to reduce memory storage. Next, we learn a neural network $\mathscr{N}_i^k$ that approximates the function $p \mapsto z_i^k$. We use a fully connected neural network with two hidden layers, with the tanh activation function. We experimented with various sizes of the two hidden layers and settled on $5 \cdot \max(P, d_i^k)$, where P is the pose dimension.

As explained in [10], we also perform per-neural network culling of input dimensions in p using perturbation analysis on each joint. Finally, we construct our deformation function N_i (see Equation 4.2) as

$$N_i(p) = \sum_{k=1}^{n_0} V_i^k \mathscr{N}_i^k(p). \tag{4.7}$$

Selecting the number of retained dimensions The value d_i^k is determined automatically by specifying a target average per-vertex per-training-frame PCA reconstruction error ε,

$$\frac{1}{n_i N} \left\| \begin{bmatrix} \hat{Y}_i^{1,k} & \dots & \hat{Y}_i^{N,k} \end{bmatrix} - V_i^k \begin{bmatrix} z_i^{1,k} & \dots & z_i^{N,k} \end{bmatrix} \right\|_F^2 = \tag{4.8}$$

$$= \frac{1}{n_i N} \sum_{\ell=d_i^k+1}^{\min(3n_i,N)} \sigma_\ell^2 < \varepsilon^2, \tag{4.9}$$

where σ_ℓ are the singular values of the LHS matrix in Equation 4.6. We use a uniform global threshold ε for all regions and all levels; this means that the number of retained dimensions will

vary across the model based on the deformation complexity; this ensures that all regions on the model resolve deformations to approximately the same quality level. By adjusting the parameter ε, one can trade the reconstruction accuracy for runtime reconstruction speed and required memory (Figure 4.6). Note that, depending on ε, some d_i^k may be zero; i.e., the L_2-norm of training residuals in a region is very small relative to the requested accuracy ε. In such a case, we do not train a neural network, but instead discard (cull) this region altogether. Culling enables us to localize computation to areas with the largest displacements.

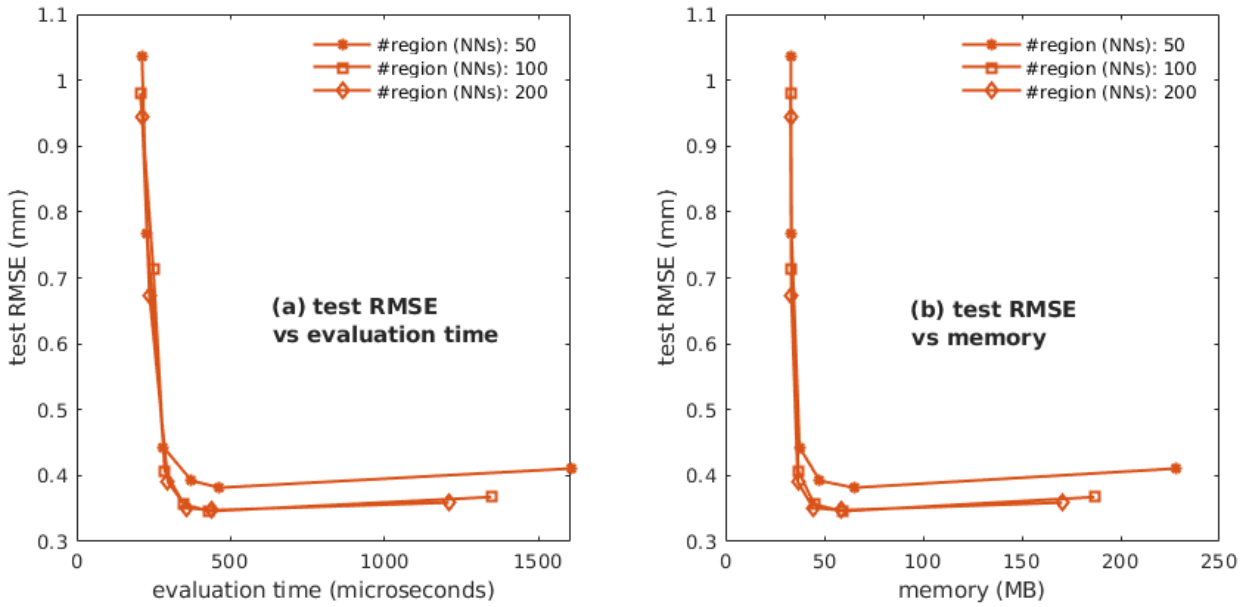


Figure 4.6: **Trading approximation accuracy for speed and memory,** by adjusting the ε threshold. Note that the y-axis shows the testing error, i.e., difference to the ground truth on motions *unseen* during training. All the datapoints on the same curve were obtained by keeping the number of basis functions (n_0) constant and varying ε. Hand example (Figure 1.3). Timings correspond to evaluating all four resolution levels.

4.4 Run-time code and memory considerations

Our work aims at hard-real-time animation suitable for computer games and VR systems, whereby the total computational budget to deform a character with 70,000 vertices is at most a few milliseconds. Under such constraints, it is necessary to employ certain code optimizations, as described

next. We are not aware of any prior work that has discussed such speed optimizations in the context of mesh LOD deformation computations for hard-real-time systems.

At runtime, we evaluate our neural networks using our own code; we do not use standard libraries such as PyTorch [110]. We first used the C/C++ distribution of libtorch at runtime, but found it approximately 5-10x slower than our code. This is not surprising as such code is designed for general neural network applications as opposed to real-time systems. In time-critical sections, our code prefers C and avoids C++, and in particular, we do not use the algorithms from the Standard Template Library (STL). We avoid all dynamic memory allocations (no malloc), as these would otherwise drastically slow down performance. We allocate all memory for our runtime datastructures as one contiguous block. All the runtime data for each spatial region at each hierarchical level (trained neural network weights, modal matrices) is stored contiguously in memory, to improve read cache coherence.

As is commonly the case in high-performance applications, our runtime code is memory-bound, i.e., it takes significantly more time to read neural network and upsampling weights from memory, than to actually use them for computation. The largest computational cost is incurred in upsampling, and therefore we optimize memory reads of upsampling weights by storing them in 16-bit precision. Specifically, each vertex (on any LOD level i) has three non-negative weights w_1, w_2, w_3 that sum to 1; they are weights of vertices of a level $i-1$ triangle "driving" this vertex. We only store and read w_1 and w_2 from memory; they are represented as 16-bit unsigned integers i_1 and i_2, packed into the lowest and highest 16 bits of an 32-bit unsigned integer, respectively. At runtime, after reading i_1 and i_2 from memory, we can access w_1, w_2, w_3 by "hacking" the IEEE 754 single-precision floating point format. Namely, we exploit the fact that a single-precision FP number in the format 1.binarydigits (1 and binary digits after the decimal point) conveniently stores the binarydigits into the 23-bit mantissa. Note that ALU and floating point operations are very cheap compared to memory reads.

```
// Read 32-bits from memory, containing two 16-bit unsigned ints.
unsigned int packedWeights = memoryArray[...]; // read 32 bits
// Create FP representations of the two 16-bit unsigned integers,
```

```
// by interpreting the low and high 16-bits as FP mantissas.
unsigned int w1i = ((packedWeights & 65535) << 7) | (127 << 23);
unsigned int w2i =
  (((packedWeights >> 16) & 65535) << 7) | (127 << 23);
// View as float. Only writing to local variables w1 and w2
// stored in registers. No memory write occurs here.
float w1 = *(float*) &w1i; float w2 = *(float*) &w2i;
// Now, use w1 - 1.0f and w2 - 1.0f, and (3.0f - w1 - w2) to
// access barycentric weights w1, w2, w3.
```

If a weight is 1.0, we re-order the weights so that $w_3 = 1$, and therefore $w_1 = w_2 = 0$. This is necessary because the above representation cannot store 1.0, but only values slightly smaller than 1: representable values range from 0 to $1 - 2^{16}$. Our approach accelerated upsampling by approximately 35% in our examples, at a negligible loss of runtime vertex position precision. Our upsampling speedup is applicable generally to any method that stores vertex quantities and needs to quickly upsample them using a triangle mesh LOD such as [11]. For fairness, we apply our code and memory optimizations to all the compared methods.

4.5 Results

We demonstrate our method on several examples from computer animation practice, including human hands (Figure 1.3), a full-body cartoon character (Figure 4.7), and a four-legged animal (an elephant) whose skeleton is driven using Inverse Kinematics (IK) (Figure 4.10).

4.5.1 Hands

Our training data was obtained from Section 3, which uses anatomically based FEM simulation with fat, muscles and bones. The linear blend skinning skeleton and weights were modeled and computed in Maya.

Because our goal is real-time shape deformation in interactive applications (games, VR, Metaverse), we incorporated our method into an OpenGL real-time application (GLUT window manager), together with real-time skinning, dynamic vertex normals and real-time rendering (Phong shading) (Figure 4.3). Real-time dynamic normals are required in such applications to properly visualize the deformed shape, both under skinning only, or skinning plus our correctives. Therefore we include the time to compute dynamic normals in our "skinning" timings. Furthermore, hard-real-time application are particularly challenging when running times drop below 1 millisecond because cache misses become extremely expensive. Namely, one crucial aspect is whether the "runtime datastructures" (skinning weights, PCA basis matrices, neural network coefficients, upsampling weights) are present in the memory caches ("hot cache"), or not/were evicted ("cold cache"). The typical situation in a real-time rendering system is a cold cache because the real-time renderer keeps evicting the runtime datastructures from the caches at each rendering frame; this effect can cause even a 2x slowdown of performance when in the hard-real-time regime. The performance reported in the thesis is evaluated with hot cache, while cold-cache performance drops to 2x slower.

Memory storage and consequently runtime performance (due to better caching) could be improved using techniques typically done in game industry such as by using lower-level APIs (e.g., Vulkan or Metal), or directly accessing the graphics drivers and window managers of each platform; we consider such optimizations beyond the scope of our academic work. In Figure 4.8, we demonstrate that our method brings benefits even when objects are far away from the camera, and not just when they are right in front of the camera. The coarsest LOD 1 is activated when the hand is far away from camera. Even in this case, there are clear silhouette differences between skinning and our method. This establishes that it is useful to compute the correctives even at coarse LODs, as opposed to merely use skinning.

4.5.2 Cartoon character

Our cartoon character (Megan) comes from the Mixamo project [111]. Mixamo provides an animated skeleton and skinning weights for the rendering mesh; we remeshed it in Maya to increase the resolution, and resampled the Mixamo skinning weights onto this new mesh, which then serves as our finest LOD4 mesh (the "skin").

Megan is simulated using FEM constrained dynamics similarly to [37]. Specifically, we first create a "shrunken skin", by computing the signed distance field D against the character polygon soup geometry [112]. We then use isosurface meshing [113] to create a quality surface triangle mesh of the skin ($D = 0$), and the shrunken skin ($D = -d$, where $d > 0$ is the "fat" thickness). Finally, we tet-mesh the volume between the shrunken skin and the skin [114], obtaining FEM mesh for the body "fat". The "shrunken skin" is then equipped with skinning weights using Maya, and animated with the Mixamo animations. Our "fat" is constrained to this animated "shrunken skin", but is otherwise free to deform, subject to collisions and self-collisions. This gives us good quality skin deformations and contact at the character joints. As shown in Figure 4.7, our real-time system is able to reproduce high-quality shapes, as well as interpolate them to new poses (unseen in the training data). The largest pre-skinning displacements occur near the joints, and are captured by our multi-resolution neural deformation functions F_i. Observe good-quality shape deformation near the characters' joints, where the contact is resolved well, and the elbow, where the volume collapsing is avoided.

4.5.3 IK example

In our third example, we equipped a quadruped's (an elephant) skeleton with inverse kinematics (Figure 4.10). We drag (in real-time) IK handles to generate representative training skeleton poses; we record every mouse move and this resulted in 3646 training poses. In this stage, the elephant's shape is deformed using LBS, producing the expected visual artefacts, i.e., severe self-collisions at the hip joints. We created the skinning weights in Maya using Maya's "HeatMap" method, followed by manual improvements and smoothing using the popular "brSmoothWeights" Maya

plugin. Next, we use FEM simulation (as described above with the "Megan" example [37]) to compute correct FEM training shapes for the 3646 training poses, incorporating volume preservation and skin self-collision handling. Finally, we train our system using these training poses and FEM shapes.

The end result is that we can then apply our trained method onto new motions (unseen during training) in hard real-time, such as a complete elephant walk cycle (Figure 4.10), all the while displaying high-quality approximations (outputs of our method) to the FEM shapes, as opposed to the erroneous LBS shapes. At the finest level LOD4 mesh (33,346 vertices), it takes 625 microsec to evaluate the skinned shape, 29.7 seconds to evaluate the FEM ground truth shape, and 1,142 microseconds for our shape (including skinning; 26,000× faster than FEM). Observe that, unlike skinning, our method resolves the self-collisions near the elephant's hips.

4.5.4 Experiments

In Figure 4.9, we compare our method to using a single-level method of [10] separately at each LOD level. We train our LOD method and the single-level method [10], under the same training dataset, the Hand example (Figure 1.3). In these plots, we only vary the number of regions and the PCA approximation threshold. The purpose of varying the PCA approximation threshold is to control the speed VS accuracy (and memory VS accuracy) tradeoff. We repeat the experiment for several #regions (50, 100, 200), essentially performing a parameter sweep to discover the number of regions producing best results. The purpose of varying #regions is to ensure comparison fairness, as different methods may be optimal under different #regions. For plot clarity, we do not show datapoints where there is another datapoint that is better *both* in test RMSE *and* the running time (for the computational speed subplot) and in test RMSE *and* the memory (for the memory subplot). Note that in Figure 4.9, we compare the evaluation time and the memory to compute a single level (the finest level), but it not the goal of the thesis, which is to construct the entire LOD hierarchy.

As shown, our method offers a large memory reduction. The goal of our work is to rapidly construct the shapes at any chosen hierarchical level (or levels) at runtime. For prior work to

offer such a capability, it needs to separately train and store neural networks at each hierarchical level, essentially reconstructing the shape from scratch at each level; this significantly increases the memory footprint. Note that prior work cannot "naively" achieve such a capability by only training on the finest level and then downsampling the result to any chosen level, because this would always incur the computational cost of the finest level, even if only, say, the coarsest level is needed.

The average PCA basis dimensions (for equal PCA reconstruction error) are also smaller in our method, enabling smaller neural networks. In the hand example (Figure 1.3) for our method, they are 13.1, 9.0, 5.6, 1.1, for LOD 1,2,3,4, respectively. They are 13.1, 13.7, 13.1, 13.1 when using the single-level method separately on each level; the difference is particularly salient at the finest LODs, where our method benefits from the work already done on coarser LODs. The same is observed in other examples. In terms of speed, our method is clearly better when all hierarchical levels need to be computed, and offers comparable speed when only a single level needs to be computed.

We use PyTorch [110] to perform the training, using Adam's optimizer [115]. We use a learning rate of 0.01 at the start of training, and decay it by a factor of 0.999999 after each epoch, for 5000 epochs. The different regions are independent and can be trained in parallel. In Figure 4.11, we measure the training time of our method. As expected, the time depends on the accuracy parameter ε and the number of basis functions n_0. Figure 4.11 also compares to prior work [10]: it can be seen that our training times are substantially faster, thanks to the lower dimensions of the per-level spatially localized regions made possible by the incremental nature of our shape construction.

In Figure 4.12, we compare our technique to Dual Quaternion Skinning [6], using the Megan exmaple. Because we learn from a quality rig, our method produces visually better shapes that resolve self-contact and preserve volume. Tables 4.1 and 4.2 present our performance statistics. Table 4.1 shows the computation time required to reconstruct a single shape at runtime for the Megan example (Figure 4.7). All times are reported in microseconds and averaged across the entire test animation (521 frames), using an Intel Xeon(R) W-3275 CPU with 56 cores @ 2.5 GHz (28 cores effectively utilized) and 196 GB of RAM. In both tables, each cell displays two times in the

Table 4.1: **Computation time** to reconstruct one shape at runtime. Megan example (Figure 4.7). All times are in **micro**seconds, and averaged over the entire testing animation (521 frames). Intel Xeon(R) W-3275 CPU, 56 cores @ 2.5 GHz (only 28 effectively utilized), with 196 GB RAM. Each cell reports two times in the format A/B, to compare our work (time "A") to simply using prior work [10] to separately process each level (time "B"). Our times correspond to the incremental time at each level; the total time to construct all levels is 697 microseconds. "NN time" is the time to evaluate neural networks. #NNs is the number of non-culled neural networks at each level, out of the $n_0 = 100$ available networks. Observe that, due to the incremental construction of deformation detail, our method has significantly fewer NNs at deeper levels than if separately processing each level.

LOD	#vtx	#NNs	NN **time**	$u = Uq$	upsampling	total
1	1,296	70 / 70	50 / 50	16 / 16	18 / 18	84 / 84
2	4,815	65 / 75	48 / 101	35 / 44	34 / 31	117 / 176
3	18,840	45 / 73	34 / 120	60 / 126	64 / 65	158 / 311
4	74,742	49 / 78	40 / 114	142 / 824	156 / 125	338 / 1063

format A/B: "A" represents our method, while "B" shows the time or memory taken by the previous approach [10] to process each level separately. In Table 4.1, our times reflect the incremental computation time at each level, with the total time for all levels being 697 microseconds. "NN time" refers to the time needed to evaluate the neural networks, and "#NNs" indicates the number of non-culled neural networks per level out of the initial $n_0 = 100$ networks. Notably, due to the incremental deformation construction, our method requires significantly fewer neural networks at deeper levels compared to processing each level independently. Table 4.2 summarizes the memory usage for reconstructing one shape at runtime for the same Megan example. The experimental setup and A/B comparison are the same as in Table 4.1. "NN mem" denotes the memory required to store the trained neural network coefficients. All values are reported in megabytes. For our method, we provide the incremental memory usage at each level, with a total of 79.7 MB across all levels. Our approach consumes significantly less memory than the method that processes each level independently.

In Figure 4.13, we further compared our method to the auto-rigging method of [91]. The auto-rigging method is capable of producing a fully rigged character solely from the character's neutral mesh, without any training data necessary for this specific character. This gives the method great

Table 4.2: **Memory** to reconstruct one shape at runtime. Megan example (Figure 4.7). Same experiment and same A/B comparison as in Table 4.1. "NN mem" is the memory to store all the trained neural network coefficients. All values are in MegaBytes. For our method, we give the incremental memory needed at each level; our total memory for all levels is 79.7 MB. Our memory consumption is much smaller than for the compared method that separately processes each level.

LOD	#vtx	#NNs	NN **mem**	$u = Uq$	upsampling	total
1	1,296	70 / 70	3.4 / 3.4	1.1 / 1.1	0 / 0	4.5 / 4.5
2	4,815	65 / 75	3.5 / 6.1	6.2 / 8.8	0.3 / 0	10.0 / 14.9
3	18,840	45 / 73	1.7 / 6.8	13.8 / 35.0	1.1 / 0	16.6 / 41.8
4	74,742	49 / 78	0.6 / 7.0	43.5 / 201	4.5 / 0	48.6 / 208

flexibility. We executed their method using their publicly available code. We found that visually, our method outperforms auto-rigging. It is also faster in terms of runtime speed.

Figure 4.14 and Figure 4.15 perform experiments on the elephant example. Figure 4.14 illustrates a failure case. Since our training data lacks examples where the elephant stands on its hind legs or where the front ankle is severely bent, our method fails to reproduce the FEM ground truth in these poses. This is a limitation of our approach, as its performance declines when applied to poses outside the trained range of motion. Moreover, in this experiment (Figure 4.15), we trained our elephant with incomplete training data, to test how this affects the output quality. In particular, we removed a part of the training data for the front-left leg. This resulted in visible penetration artefacts.

Figure 4.16 presents an analysis of multicore performance, showing CPU performance in relation to the number of cores used for a complex example with 72,414 vertices. The test was conducted on the hand example at the finest level of detail (LOD 4).

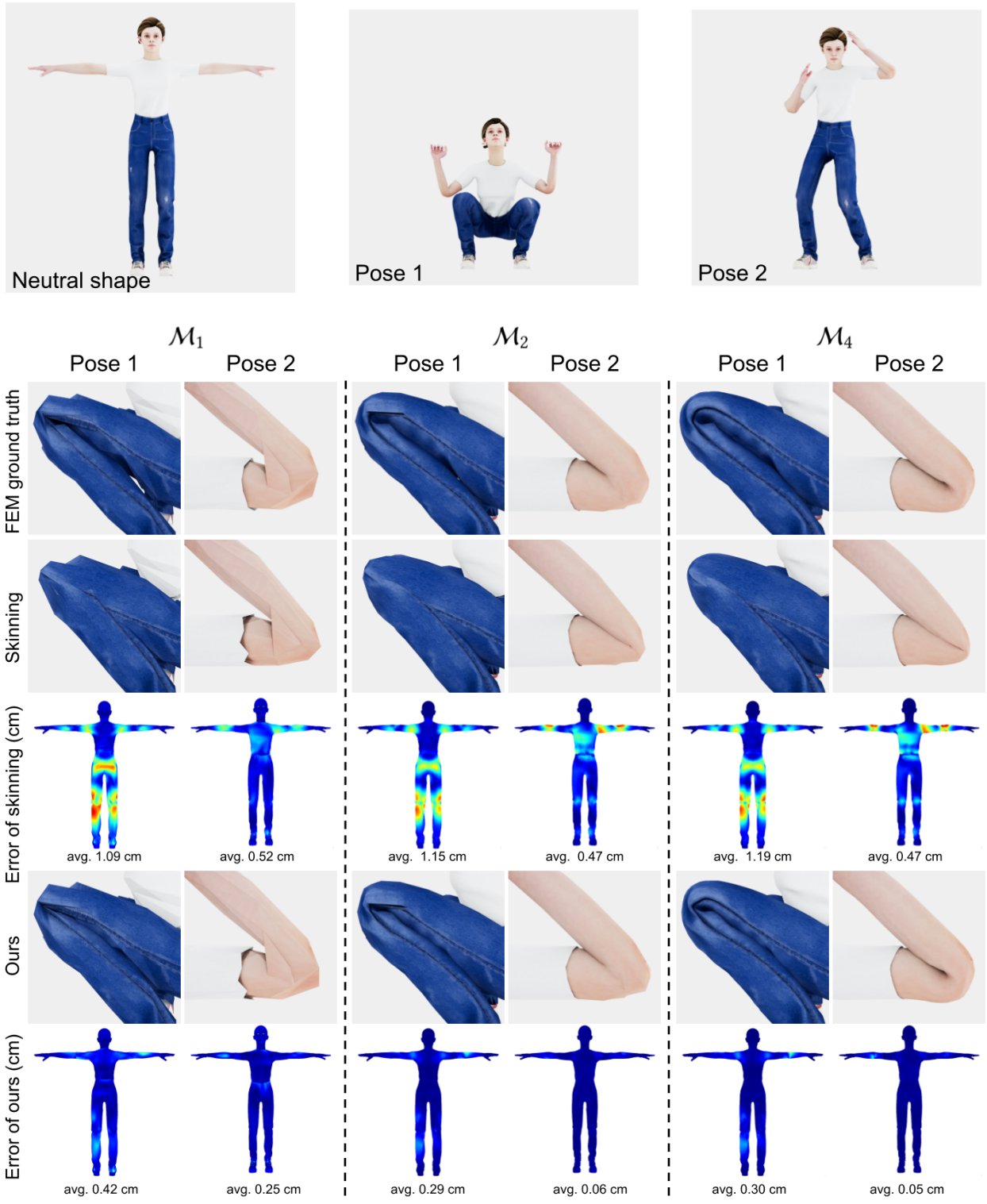


Figure 4.7: **Hard real-time LOD character deformation.** The ground truth of this character was computed using FEM deformable simulation, including self-collision handling. Our real-time system is able to regenerate high-quality shapes, as well as interpolate them to new poses. Observe good-quality shape deformation near the characters' joints.

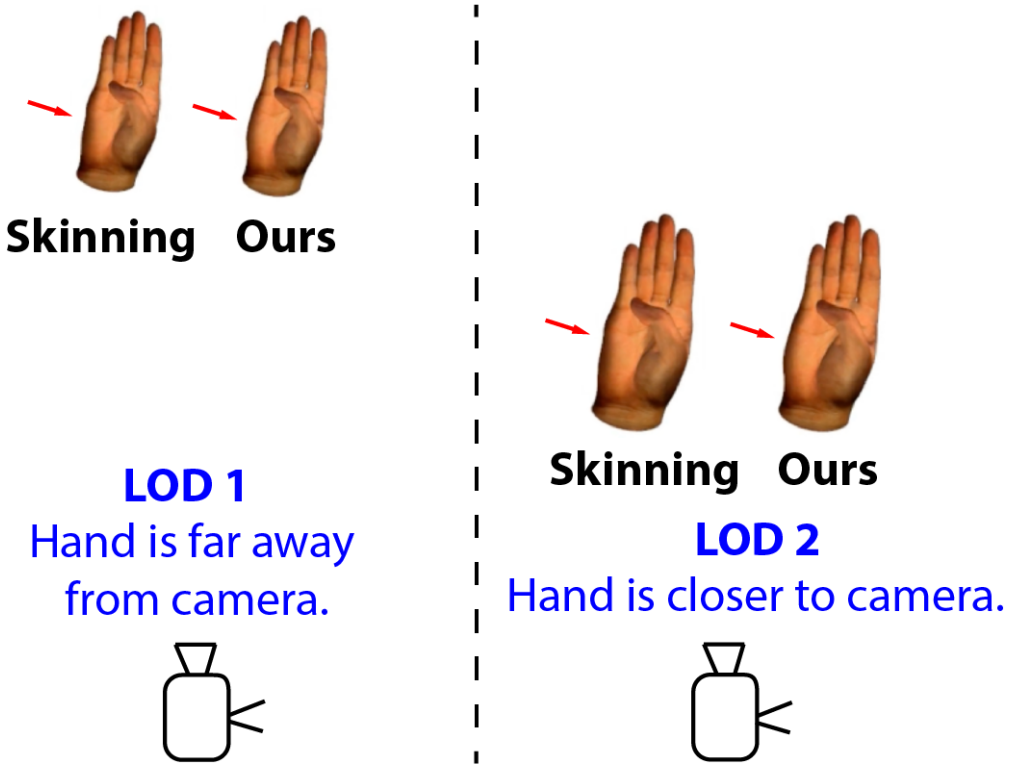
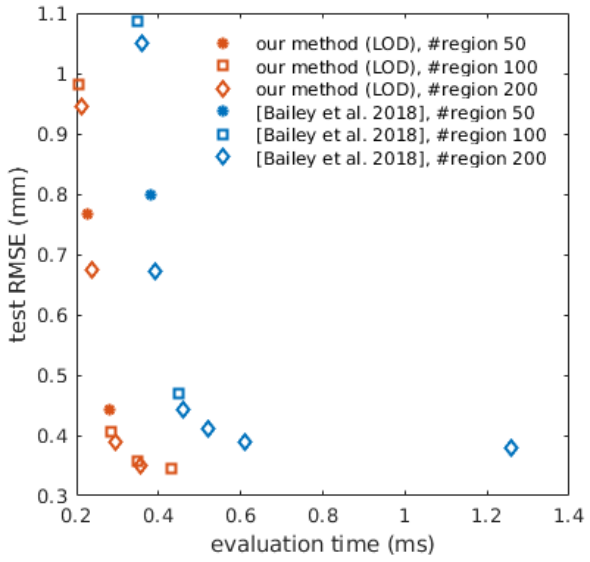
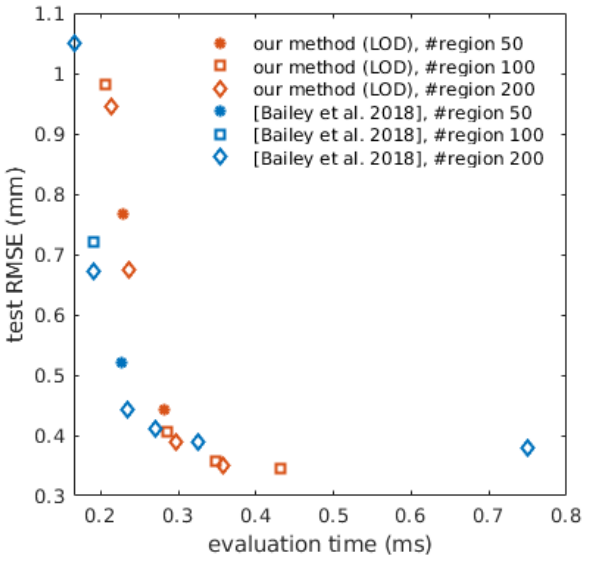


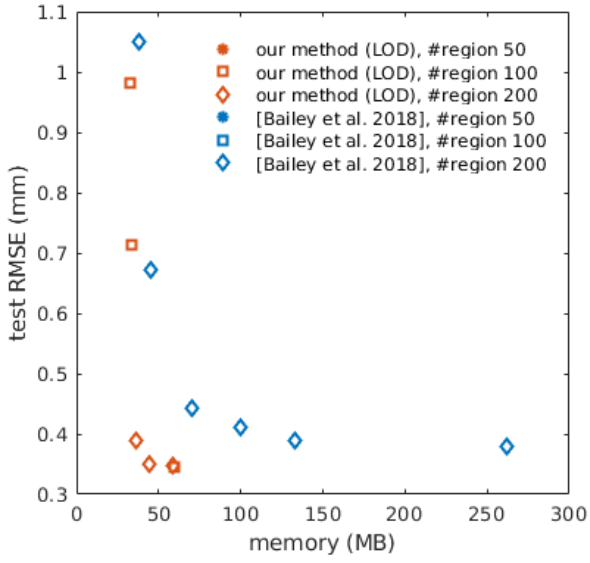
Figure 4.8: **Shape changes are visible even under coarse LODs.** The shown sizes of hands at LOD 1 (coarsest LOD) and LOD 2 (second coarsest LOD) are correctly proportional to their differences to the camera. The coarsest LOD 1 is activated when the hand is far away from camera. Even in this case, there are silhouette differences between skinning and our method.



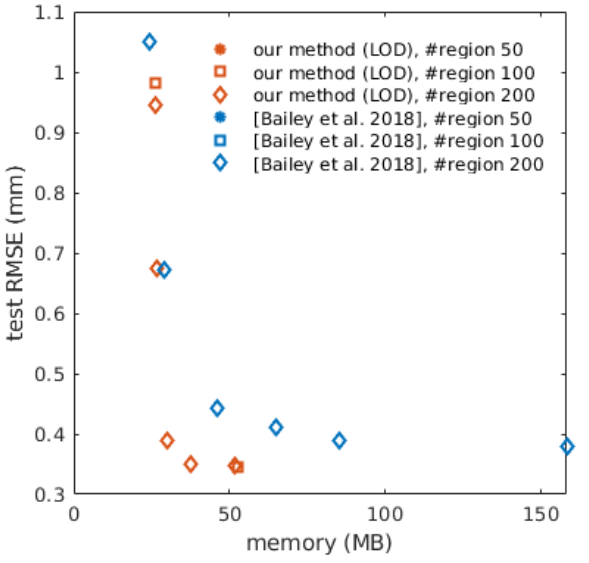
(a) test RMS vs evaluation time, all levels



(b) test RMS vs evaluation time, finest level



(c) test RMS vs memory, all levels



(d) test RMS vs memory, finest level

Figure 4.9: **Comparison to a single-level method.** We train our LOD method and the single-level method [10], under the same training dataset, the Hand example (Figure 1.3). We compare the evaluation time and the memory to construct all LOD levels and the finest level. We vary the number of regions and the PCA approximation threshold, essentially performing a parameter sweep to discover the number of regions producing best results. For plot clarity, we do not show datapoints where there is another datapoint that is better *both* in test RMSE *and* the running time (for the computational speed subplot) and in test RMSE *and* the memory (for the memory subplot).

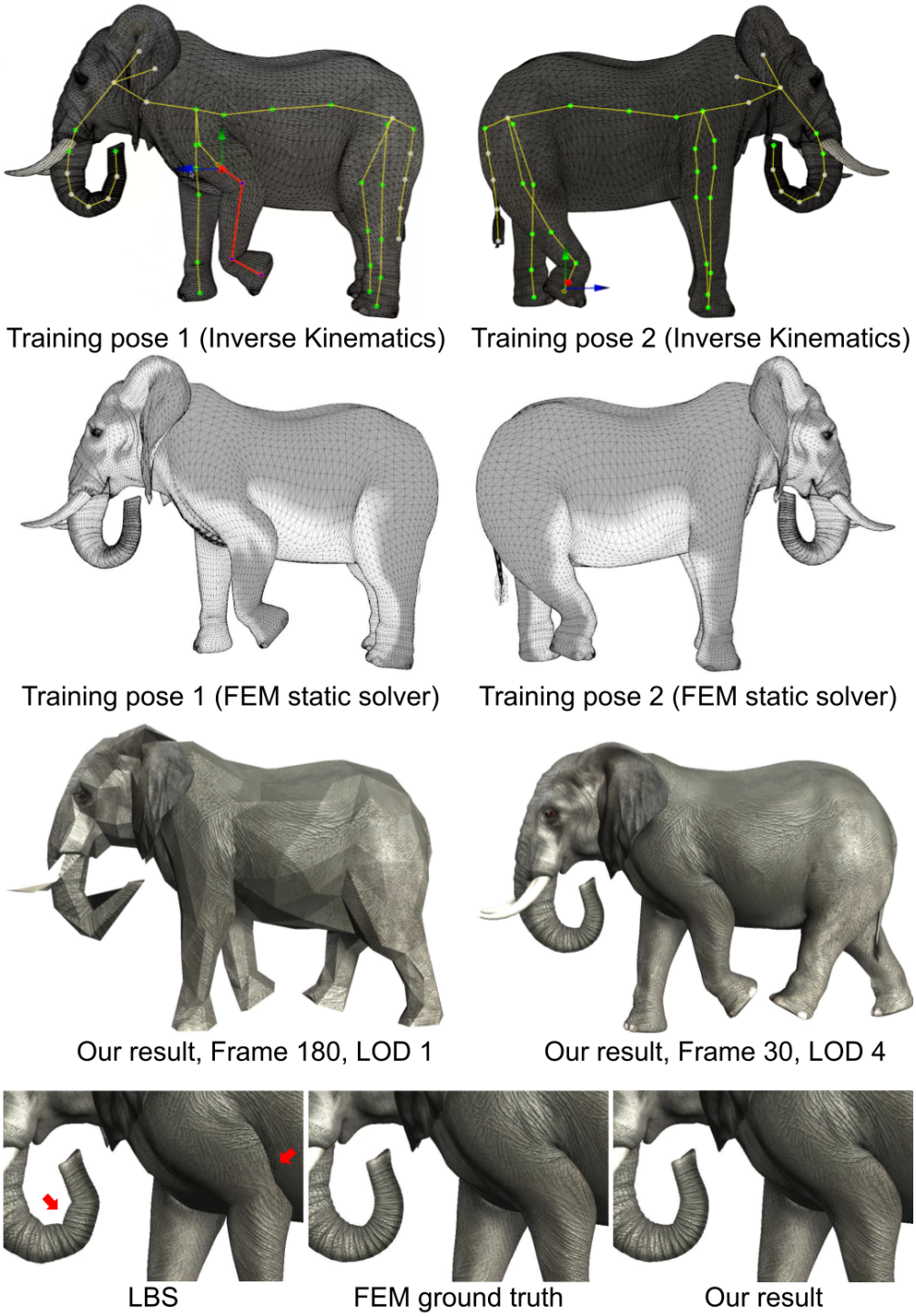


Figure 4.10: **Hard-real-time LOD quadruped.** Top row: Two representative training poses generated using IK, with the skinned shape. Next row: the training shapes for the above two training poses (FEM deformable simulation with self-collision handling). Next row: Representative frame of our output, shown at LODs 1 and 4. Bottom row: zoom of the skinned shape (625 microsec per frame), FEM ground truth (29.7 seconds per frame), and our shape (1,142 microseconds per frame including skinning; **26,000**× faster than FEM). All performance numbers are at LOD 4 (33,346 vertices). Observe that, unlike skinning, our method resolves self-collisions near the elephant's hips.

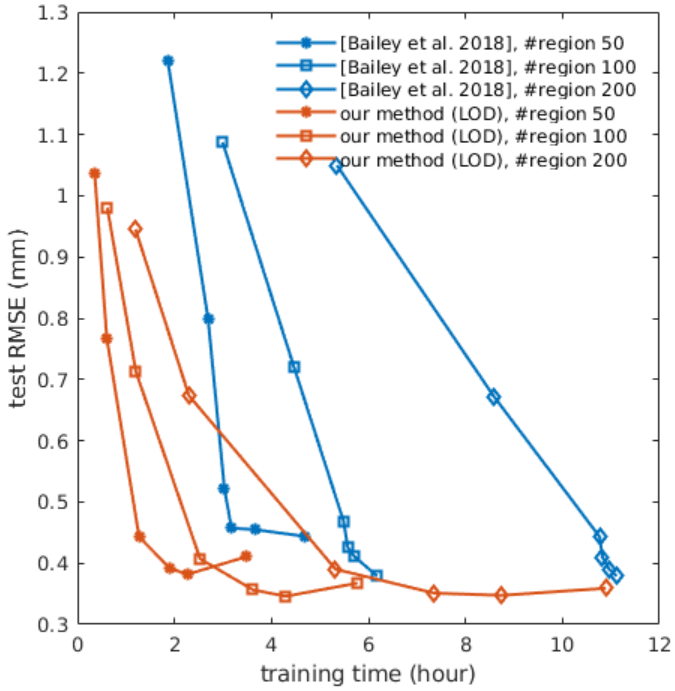


Figure 4.11: **Training time of our multi-resolution method,** corresponding to Figure 4.9. Thanks to the incremental nature of our neural networks, the reduced dimensions of our regions are significantly lower than in prior work [10], resulting in faster training times of our method.

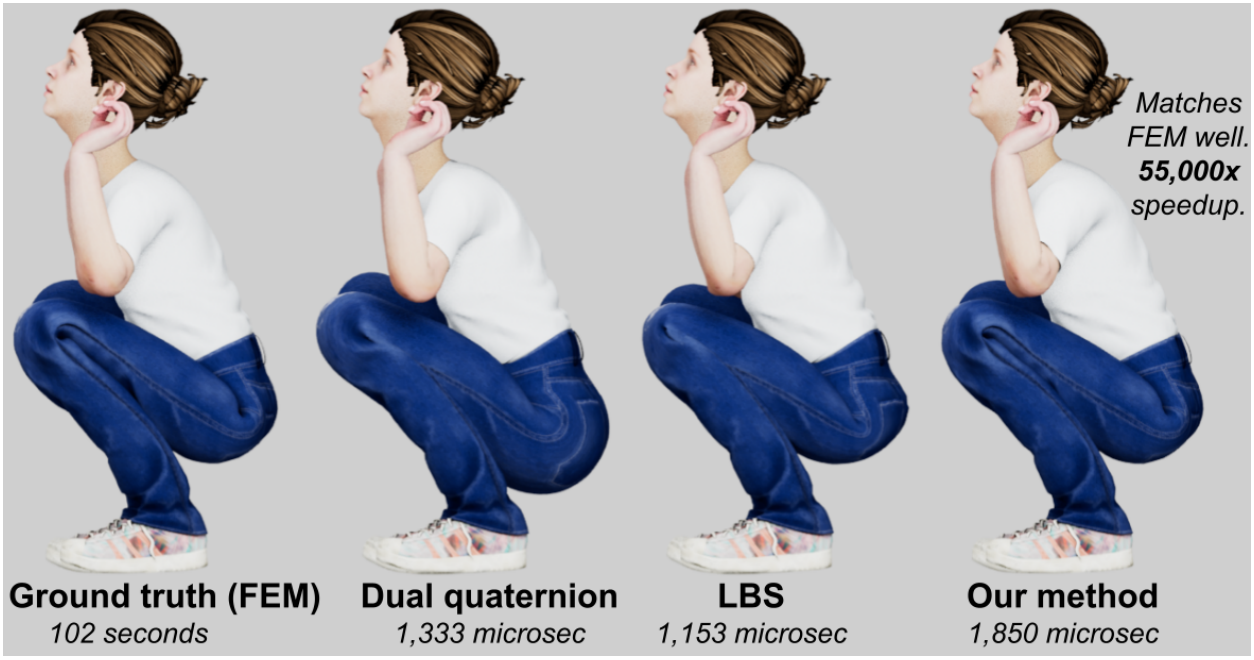


Figure 4.12: **Comparison to dual quaternions.** Our method produces visually better shapes that closely match the ground truth (FEM simulation with self-contact handling and volume preservation), at a modest increase of computing time. This pose was **unseen during training**.

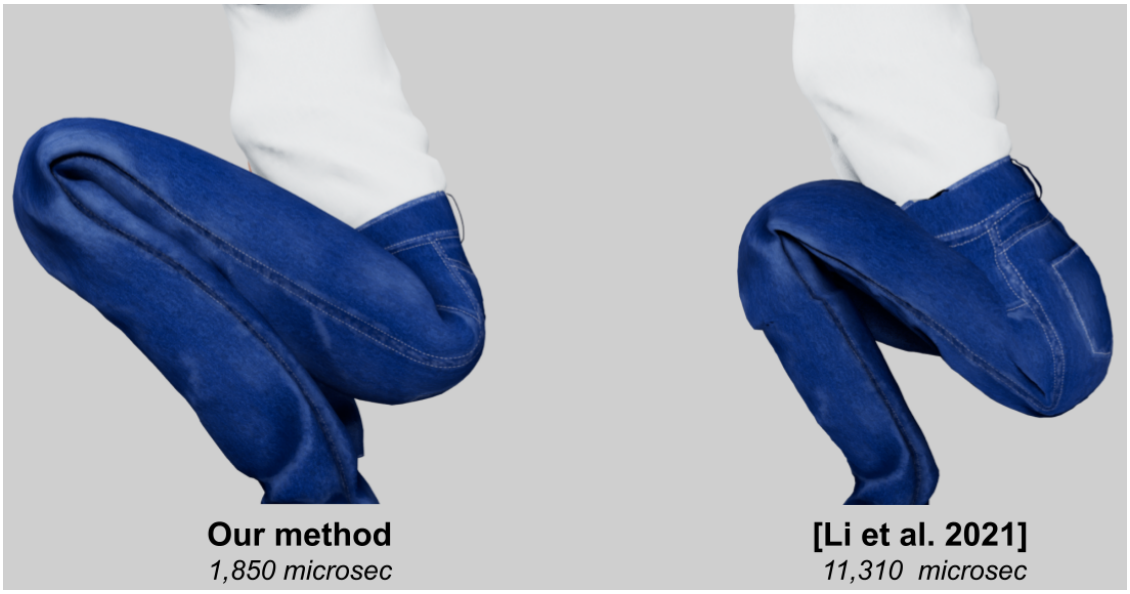


Figure 4.13: **Comparison to auto-rigging [91]**. Our method outperforms auto-rigging in terms of the shape quality and runtime speed.

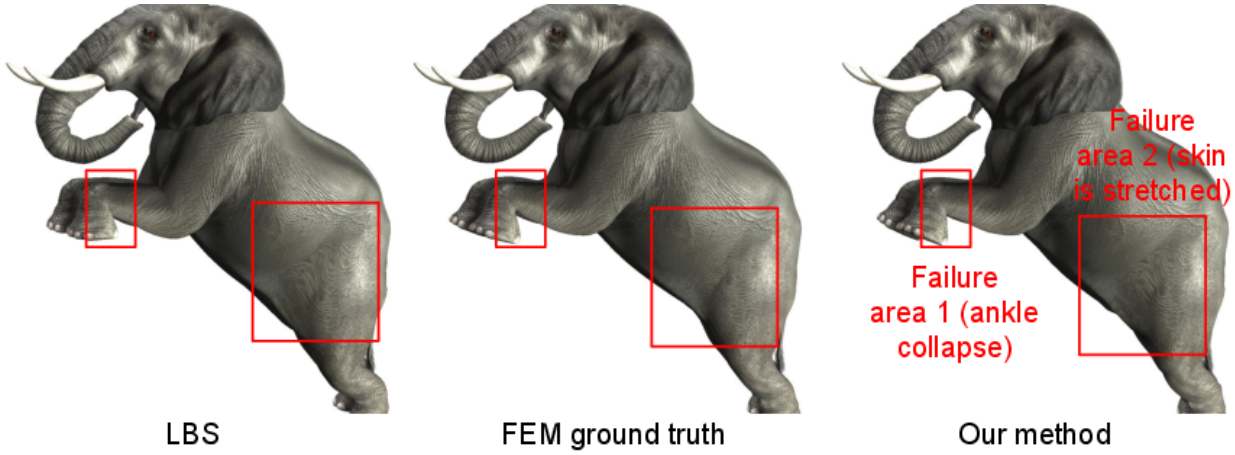


Figure 4.14: **Limitation of our method.** Our training data does not contain poses where the elephant stands on its last feet, or where the front ankle is bent severely. Our method fails in such poses.

Figure 4.15: **Using complete vs incomplete training data.** Observe that our method with incomplete training data produces visible penetration artefacts near the hip.

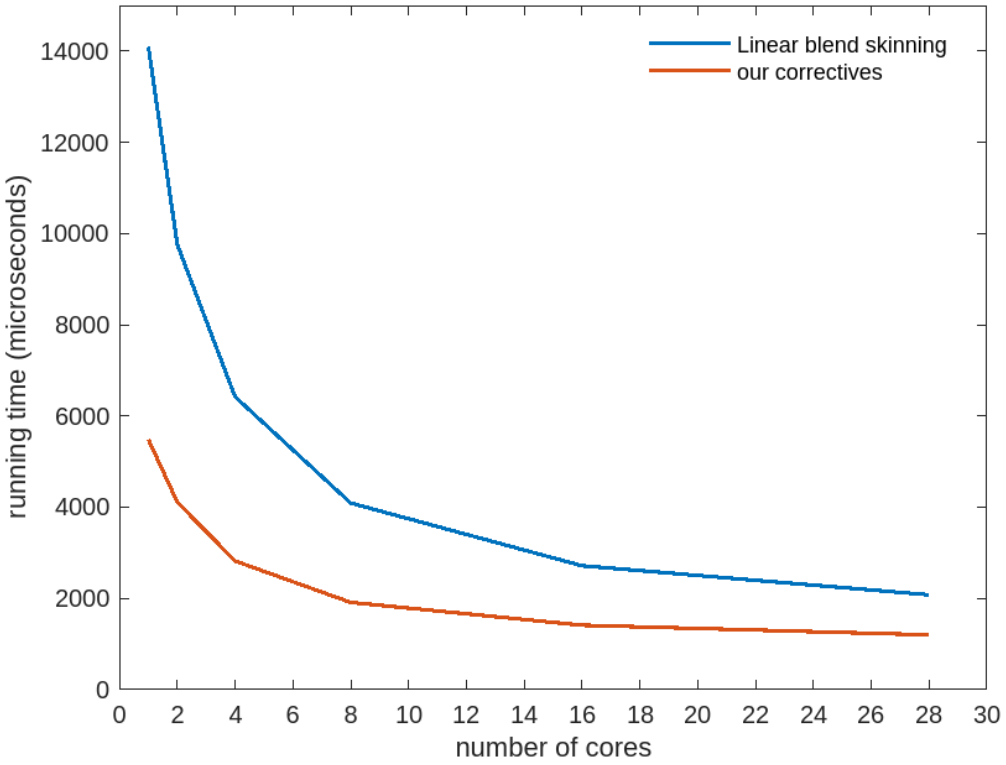


Figure 4.16: **Multicore performance.** CPU performance vs number of cores on a complex example (72,414 vertices). Hand example, finest LOD (LOD 4).

Chapter 5

Conclusion and future work

We gave a method to simulate complete musculoskeletal hand anatomy in a manner that matches medical images (MRI) in scanned example poses, and that generalizes smoothly to the entire range of motion of the hand, as demonstrated on five complex hand motion sequences. We are not aware of any work that has demonstrated such capabilities before.

Starting from MRI scans of the same subject's hand in multiple poses, we build a data-driven simulation system that can simulate the hand's anatomy to any pose in the hand's range of motion. This is achieved through layered FEM simulation whereby per-organ plastic strains are added to "guide" the simulation to the acquired medical images in the example poses. We present new methods to simulate hand muscles, tendons and fat, and demonstrate that this not only enables volumetric rendering of hand anatomy across the range of motion, but also improves the hand's surface shape.

Although simulation times are long, they are feasible on modern hardware, and especially so with public infrastructure such as AWS, Google Cloud, etc. In the future, our method could run on the cloud, systematically exploring the entire range of motion of the hand. In this way, one could produce high-quality training data for data-driven methods (e.g., SMPL [79], or deep neural networks), which could then compactify the data and greatly increase runtime speeds. Our work processed a single subject; for Metaverse applications, more subjects will be needed, and they will need to be properly selected to capture humankind's diversity. We only modeled concentric isotonic muscle contraction, and leave other muscle interactions, such as those involving heavy

grasping or lifting (isometric), or lengthening/overpowering the muscles (eccentric isotonic) for future work. In particular, (substantial) external contact will likely invalidate muscle activation. Our simulation involves one-way coupling between the layers because it is otherwise not feasible to solve the (coupled) optimization problems to make the simulation match the medical images. Two of the hand muscles (lumbricals III and IV) were too small to reliably resolve on the MRI in multiple poses, and we simulate them directly, i.e., based on their shape in the neutral shape and pose-varying bone attachments without using MRI. Our tendons are only modeled in the region where they are visible on the MRI scan. In general, MRI resolution is an important limiting factor, as many hand structures are at the limit of what the MRI scanner can resolve. However, by inferring the structure from multiple scans and when combined with physically based simulation, we were able to reasonably overcome these challenges and produce, to the best of our knowledge, the world's first simulation-ready volumetric hand musculoskeletal system that both matches and generalizes real medical images.

While our hand anatomy simulation system enables accurate simulation of the musculoskeletal anatomy of the hand, capturing its full range of motion based on MRI scans and FEM simulation, it is computationally demanding and time-intensive. This opens the door for data-driven approaches that can significantly reduce processing time without sacrificing accuracy. Building on this idea, we further present a technique to approximate and generalize arbitrary training mesh deformations using a set of hierarchically defined neural networks.

We do so by employing a partition of unity at multiple resolution levels, and defining the shape functions using the mesh prolongation (i.e., upsampling) operator of a standard and widely used mesh simplification algorithm. These functions then guide our neural networks support and construction. The resulting technique permits a progressive computation of mesh deformations, enabling one to trade computational accuracy for speed. Our technique greatly decreases the required memory in applications involving meshes at multiple resolution levels; such applications are common in computer games and other interactive systems. Our basis functions at all levels are prolongations of a partition of unity on $\mathscr{M}_0$. We considered an alternative scheme whereby

the basis functions on $\mathscr{M}_{i+1}$ are obtained directly from the upsampling operator $\mathscr{U}_i$, without using coarser levels. This produces basis functions that become progressively narrower on progressive levels; but did not improve performance in our examples because our basis functions are occupying relatively narrow regions as is (Figure 4.5). Our technique requires good-quality training shapes, which is a standard, but time-consuming process. As common with learning methods, quality outside of the training dataset diminishes. We only use the CPU to compute skinning and correctives, but both of those could be offloaded to the GPU, either using vertex or compute shaders, or CUDA. In the future, we would like to employ our multi-resolution construction for other applications in computer animation, e.g., to model or approximate other signals defined on 3D meshes.

Acknowledgements

This research was sponsored in part by NSF (IIS-1911224), USC Provost Fellowship to Mianlun Zheng, Adobe Research, and Bosch Research.

References

1. Lewis, J. P., Cordner, M. & Fong, N. *Pose Space Deformation: A Unified Approach to Shape Interpolation and Skeleton-Driven Deformation* in *Proc. of ACM SIGGRAPH 2000* (2000), 165–172.

2. Kry, P. G., James, D. L. & Pai, D. K. *EigenSkin: Real Time Large Deformation Character Skinning in Hardware* in *In Symp. on Computer Animation (SCA)* (2002).

3. Wang, B., Matcuk, G. & Barbič, J. Hand Modeling and Simulation Using Stabilized Magnetic Resonance Imaging. *ACM Trans. on Graphics (SIGGRAPH 2019)* **38** (2019).

4. Tissue. *Weta Digital: Tissue Muscle and Fat Simulation System* 2013.

5. Jacobson, A., Deng, Z., Kavan, L. & Lewis, J. P. *Skinning: Real-time Shape Deformation* in *ACM SIGGRAPH 2014 Courses* (2014).

6. Kavan, L., Collins, S., Zara, J. & O'Sullivan, C. Geometric Skinning with Approximate Dual Quaternion Blending. *ACM Trans. on Graphics* **27** (2008).

7. Le, B. H. & Lewis, J.-P. Direct delta mush skinning and variants. *ACM Trans. on Graphics (SIGGRAPH 2019)* **38,** 113–1 (2019).

8. Kavan, L., Collins, S. & O'Sullivan, C. *Automatic linearization of nonlinear skinning* in *Proc. of the Symp. on Interactive 3D Graphics and Games* (2009), 49–56.

9. Boier-Martin, I., Zorin, D. & Bernardini, F. A survey of subdivision-based tools for surface modeling. *DIMACS Series in Discrete Math and Theoretical CS* **67,** 1 (2005).

10. Bailey, S. W., Otte, D., Dilorenzo, P. & O'Brien, J. F. Fast and Deep Deformation Approximations. *ACM Transactions on Graphics (SIGGRAPH 2018)* **37** (2018).

11. Liu, H.-T. D., Zhang, J. E., Ben-Chen, M. & Jacobson, A. Surface multigrid via intrinsic prolongation. *ACM Transactions on Graphics (SIGGRAPH 2021)* **40** (2021).

12. Garland, M. & Heckbert, P. S. *Surface simplification using quadric error metrics* in *Proc. of ACM SIGGRAPH 97* (1997), 209–216.

13. Hoppe, H. *Progressive meshes* in *Proc. of ACM SIGGRAPH 96* (1996), 99–108.

14. James, D. L. & Pai, D. K. Multiresolution Green's Function Methods for Interactive Simulation of Large-scale Elastostatic Objects. *ACM Trans. on Graphics* **22,** 47–82 (2003).

15. Funkhouser, T. A. & Séquin, C. H. *Adaptive Display Algorithm for Interactive Frame Rates During Visualization of Complex Virtual Environments* in *Proc. of ACM SIGGRAPH 93* (1993), 247–254.

16. Otaduy, M. A. *6-DoF Haptic Rendering Using Contact Levels of Detail and Haptic Textures* PhD thesis (Department of Comp. Science, University of North Carolina at Chapel Hill, 2004).

17. Jacobson, A., Deng, Z., Kavan, L. & Lewis, J. P. *Skinning: Real-time Shape Deformation* in *ACM SIGGRAPH 2014 Courses* (2014).

18. Vaillant, R., Guennebaud, G., Barthe, L., Wyvill, B. & Cani, M. Robust Iso-surface Tracking for Interactive Character Skinning. *ACM Trans. on Graphics (SIGGRAPH Asia 2014)* **33,** 189:1–137:11 (2014).

19. Kurihara, T. & Miyata, N. *Modeling deformable human hands from medical images* in *Symp. on Computer Animation (SCA)* (2004), 355–363.

20. Rhee, T., Lewis, J. & Neumann, U. *Real-Time Weighted Pose-Space Deformation on the GPU* in *Proc. of Eurographics* **25** (2006).

21. Romero, J., Tzionas, D. & Black, M. J. Embodied Hands: Modeling and Capturing Hands and Bodies Together. *ACM Trans. on Graphics (SIGGRAPH Asia 2017)* **36,** 245:1–245:17 (2017).

22. Li, Y., Wu, M., Zhang, Y., Xu, L. & Yu, J. PIANO: A Parametric Hand Bone Model from Magnetic Resonance Imaging. *arXiv preprint arXiv:2106.10893* (2021).

23. Li, Y., Zhang, L., Qiu, Z., Jiang, Y., Li, N., Ma, Y., *et al.* NIMBLE: a non-rigid hand model with bones and muscles. *ACM Transactions on Graphics (SIGGRAPH 2022)* **41,** 1–16 (2022).

24. 3dMD. *3dMDHands* https://3dmd.com/. 2022.

25. Qian, N., Wang, J., Mueller, F., Bernard, F., Golyanik, V. & Theobalt, C. *HTML: A Parametric Hand Texture Model for 3D Hand Reconstruction and Personalization* in *Proc. of the European Conf. on Computer Vision (ECCV)* (Springer, 2020).

26. Sifakis, E., Neverov, I. & Fedkiw, R. Automatic determination of facial muscle activations from sparse motion capture marker data. *ACM Trans. on Graphics (SIGGRAPH 2005)* **24,** 417–425 (2005).

27. Sachdeva, P., Sueda, S., Bradley, S., Fain, M. & Pai, D. K. Biomechanical Simulation and Control of Hands and Tendinous Systems. *ACM Trans. Graph.* **34,** 42:1–42:10 (2015).

28. Dicko, A. H., Liu, T., Gilles, B., Kavan, L., Faure, F., Palombi, O., *et al.* Anatomy Transfer. *ACM Trans. on Graphics (SIGGRAPH)* **32,** 188:1–188:8 (2013).

29. Saito, S., Zhou, Z. & Kavan, L. Computational Bodybuilding: Anatomically-based Modeling of Human Bodies. *ACM Trans. on Graphics (SIGGRAPH 2015)* **34** (2015).

30. Kadlecek, P., Ichim, A.-E., Liu, T., Krivanek, J. & Kavan, L. Reconstructing Personalized Anatomical Models for Physics-based Body Animation. *ACM Trans. Graph.* **35** (2016).

31. Zygote. *Zygote body* http://www.zygotebody.com. 2016.

32. C. Erolin. *Hand Anatomy* University of Dundee, Centre for Anatomy and Human Identification. https://sketchfab.com/anatomy_dundee/collections/hand-anatomy. 2019.

33. Wang, B., Matcuk, G. & Barbič, J. Modeling of Personalized Anatomy using Plastic Strains. *ACM Trans. on Graphics (TOG)* **40** (2021).

34. Garre, C., Hernández, F., Gracia, A. & Otaduy, M. A. *Interactive simulation of a deformable hand for haptic rendering* in *IEEE World Haptics Conf.* (2011), 239–244.

35. Capell, S., Burkhart, M., Curless, B., Duchamp, T. & Popović, Z. *Physically Based Rigging for Deformable Characters* in *Symp. on Computer Animation (SCA)* (2005), 301–310.

36. Kim, J. & Pollard, N. S. Fast simulation of skeleton-driven deformable body characters. *ACM Trans. on Graphics (TOG)* **30,** 121 (2011).

37. McAdams, A., Zhu, Y., Selle, A., Empey, M., Tamstorf, R., Teran, J., *et al.* Efficient elasticity for character skinning with contact and collisions. *ACM Trans. on Graphics (SIGGRAPH 2011)* **30** (2011).

38. Liu, L., Yin, K., Wang, B. & Guo, B. Simulation and control of skeleton-driven soft body characters. *ACM Trans. on Graphics (SIGGRAPH Asia)* **32,** 215 (2013).

39. Smith, B., Goes, F. D. & Kim, T. Stable Neo-Hookean Flesh Simulation. *ACM Trans. Graph.* **37,** 12:1–12:15 (2018).

40. Lee, S. H., Sifakis, E. & Terzopoulos, D. Comprehensive Biomechanical Modeling and Simulation of the Upper Body. *ACM Trans. on Graphics* **28,** 99:1–99:17 (2009).

41. Kapandji, A. *The physiology of the joints, 6th Edition, Vol. 1: The Upper Limb* (Elsevier Exclusive, 2009).

42. Miyata, N., Kouch, M., Mochimaru, M. & Kurihara, T. *Finger joint kinematics from MR images* in *IEEE Int. Conf. on Intelligent Robots and Systems* (2005), 2750–2755.

43. Rusu, A. *Segmentation of bone structures in Magnetic Resonance Images (MRI) for human hand skeletal kinematics modelling* MA thesis (German Aerospace Center, 2011).

44. Van der Smagt, P. & Stillfried, G. *Using MRI data to compute a hand kinematic model* in *Conf. on Motion and Vibration Control (MOVIC)* (2008).

45. Stillfried, G. *Kinematic modelling of the human hand for robotics* PhD thesis (Technische Universität München, 2015).

46. Keller, M., Zuffi, S., Black, M. J. & Pujades, S. *OSSO: Obtaining Skeletal Shape from Outside* in *Conf. on Computer Vision and Pattern Recognition (CVPR)* (2022), 20492–20501.

47. Abdrashitov, R., Bang, S., Levin, D., Singh, K. & Jacobson, A. Interactive Modelling of Volumetric Musculoskeletal Anatomy. *ACM Transactions on Graphics* **40** (2021).

48. Angles, B., Rebain, D., Macklin, M., Wyvill, B., Barthe, L., Lewis, J., *et al.* Viper: Volume invariant position-based elastic rods. *Proc. of ACM on Computer Graphics and Interactive Techniques* **2,** 1–26 (2019).

49. Modi, V., Fulton, L., Jacobson, A., Sueda, S. & Levin, D. *Emu: Efficient muscle simulation in deformation space* in *Computer Graphics Forum* **40** (2021), 234–248.

50. Zajac, F. Muscle and tendon: properties, models, scaling, and application to biomechanics and motor control. *Critical reviews in biomedical engineering* **17,** 359–411 (1989).

51. Lee, S., Yu, R., Park, J., Aanjaneya, M., Sifakis, E. & Lee, J. Dexterous manipulation and control with volumetric muscles. *ACM Transactions on Graphics (SIGGRAPH 2018)* **37,** 57:1–57:13 (2018).

52. Ichim, A., Kadlecek, P., Kavan, L. & Pauly, M. Phace: Physics-based Face Modeling and Animation. *ACM Trans. on Graphics (SIGGRAPH 2017)* **36** (2017).

53. Roussellet, V., Rumman, N. A., Canezin, F., Mellado, N., Kavan, K. & Barthe, L. Dynamic implicit muscles for character skinning. *Computers & Graphics* **77,** 227–239 (2018).

54. Sueda, S., Kaufman, A. & Pai, D. K. Musculotendon Simulation for Hand Animation. *ACM Trans. Graph. (Proc. SIGGRAPH)* **27** (2008).

55. Fok, K. S. & Chou, S. M. Development of a finger biomechanical model and its considerations. *J. of Biomechanics* **43,** 701–713 (2010).

56. Dogadov, A., Alamir, M., Serviere, C. & Quaine, F. The biomechanical model of the long finger extensor mechanism and its parametric identification. *J. of Biomechanics* **58,** 232–236 (2017).

57. Wilson, D. L., Zhu, Q., Duerk, J. L., Mansour, J. M., Kilgore, K. & Crago, P. E. Estimation of tendon moment arms from three-dimensional magnetic resonance images. *Annals of Biomedical Engineering* **27,** 247–256 (1999).

58. Chen, H., Chen, C., Yang, T., Kuo, L., Jou, I., Su, F., *et al. Model-based segmentation of flexor tendons from magnetic resonance images of finger joints* in *Annual Int. Conf. of the IEEE Engineering in Medicine and Biology Society* (2011), 8009–8012.

59. Garland, A. K., Shah, D. S. & Kedgley, A. E. Wrist tendon moment arms: Quantification by imaging and experimental techniques. *J. of Biomechanics* **68,** 136–140 (2018).

60. Kuok, C., Yang, T., Tsai, B., Jou, I., Horng, M., Su, F., *et al.* Segmentation of finger tendon and synovial sheath in ultrasound image using deep convolutional neural network. *Biomedical engineering online* **19,** 1–25 (2020).

61. Bergou, M., Wardetzky, M., Robinson, S., Audoly, B. & Grinspun, E. Discrete Elastic Rods. *ACM Transactions on Graphics (SIGGRAPH)* **27,** 63:1–63:12 (2008).

62. Kugelstadt, T. & Schömer, E. *Position and Orientation Based Cosserat Rods* in *Symp. on Computer Animation (SCA)* (2016).

63. Bailey, S. W. *Applications of Machine Learning for Character Animation* (University of California, Berkeley, 2020).

64. Song, S. L., Shi, W. & Reed, M. Accurate face rig approximation with deep differential subspace reconstruction. *ACM Trans. on Graphics (SIGGRAPH 2020)* **39,** 34–1 (2020).

65. Bailey, S. W., Omens, D., Dilorenzo, P. & O'Brien, J. F. Fast and Deep Facial Deformations. *ACM Transactions on Graphics (SIGGRAPH 2020)* **39** (2020).

66. Debunne, G., Desbrun, M., Cani, M.-P. & Barr, A. H. *Dynamic Real-Time Deformations Using Space & Time Adaptive Sampling* in *Proc. of ACM SIGGRAPH 2001* (2001), 31–36.

67. Capell, S., Green, S., Curless, B., Duchamp, T. & Popović, Z. *A Multiresolution Framework for Dynamic Deformations* in *Proc. of the Symp. on Comp. Animation 2002* (2002), 41–48.

68. Grinspun, E., Krysl, P. & Schröder, P. CHARMS: A Simple Framework for Adaptive Simulation. *ACM Trans. on Graphics* **21,** 281–290 (2002).

69. Zhang, J. E., Dumas, J., Fei, Y., Jacobson, A., James, D. L. & Kaufman, D. M. Progressive simulation for cloth quasistatics. *ACM Transactions on Graphics (SIGGRAPH Asia 2022)* **41,** 1–16 (2022).

70. Zhu, Y., Sifakis, E., Teran, J. & Brandt, A. An efficient multigrid method for the simulation of high-resolution elastic solids. *ACM Trans. on Graphics (TOG)* **29,** 16 (2010).

71. Otaduy, M., Germann, D., Redon, S. & Gross, M. *Adaptive deformations with fast tight bounds* in *Symp. on Computer Animation (SCA)* (2007), 181–190.

72. Kavan, L., Gerszewski, D., Bargteil, A. W. & Sloan, P.-P. Physics-Inspired Upsampling for Cloth Simulation in Games. *ACM Trans. Graph.* **30** (2011).

73. Fulton, L., Modi, V., Duvenaud, D., Levin, D. I. & Jacobson, A. *Latent-space dynamics for reduced deformable simulation* in *Computer Graphics Forum* **38** (2019), 379–391.

74. Holden, D., Duong, B. C., Datta, S. & Nowrouzezahrai, D. *Subspace neural physics: Fast data-driven interactive simulation* in *Symp. on Computer Animation (SCA)* (2019), 1–12.

75. Badías, A., González, D., Alfaro, I., Chinesta, F. & Cueto, E. Real-time interaction of virtual and physical objects in mixed reality applications. *International Journal for Numerical Methods in Engineering* **121,** 3849–3868 (2020).

76. Casas, D., Pérez, J., Otaduy, M. A., Romero, C., *et al.* Learning Contact Corrections for Handle-Based Subspace Dynamics (2021).

77. Romero, C., Casas, D., Chiaramonte, M. M. & Otaduy, M. A. Contact-centric deformation learning. *ACM Trans. on Graphics (SIGGRAPH 2022)* **41,** 1–11 (2022).

78. Luo, R., Shao, T., Wang, H., Xu, W., Chen, X., Zhou, K., *et al.* NNWarp: Neural network-based nonlinear deformation. *IEEE Trans. on Visualization and Computer Graphics (TVCG)* **26,** 1745–1759 (2018).

79. Loper, M., Mahmood, N., Romero, J., Pons-Moll, G. & Black, M. J. SMPL: A Skinned Multi-Person Linear Model. *ACM Trans. on Graphics (SIGGRAPH Asia 2015)* **34,** 248:1–248:16 (2015).

80. Zheng, M., Zhou, Y., Ceylan, D. & Barbič, J. *A deep emulator for secondary motion of 3d characters* in *Proceedings of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition* (2021), 5932–5940.

81. Habermann, M., Liu, L., Xu, W., Zollhoefer, M., Pons-Moll, G. & Theobalt, C. Real-time deep dynamic characters. *ACM Trans. on Graphics (SIGGRAPH 2021)* **40,** 1–16 (2021).

82. Santesteban, I., Garces, E., Otaduy, M. A. & Casas, D. *SoftSMPL: Data-driven Modeling of Nonlinear Soft-tissue Dynamics for Parametric Humans* in *Computer Graphics Forum* **39** (2020), 65–75.

83. Romero, C., Otaduy, M. A., Casas, D. & Perez, J. *Modeling and estimation of nonlinear skin mechanics for animated avatars* in *Computer Graphics Forum* **39** (2020), 77–88.

84. Santesteban, I., Otaduy, M. A. & Casas, D. *Learning-based animation of clothing for virtual try-on* in *Computer Graphics Forum* **38** (2019), 355–366.

85. Chentanez, N., Macklin, M., Müller, M., Jeschke, S. & Kim, T.-Y. *Cloth and skin deformation with a triangle mesh based convolutional neural network* in *Computer Graphics Forum* **39** (2020), 123–134.

86. Bertiche, H., Madadi, M. & Escalera, S. PBNS: Physically based neural simulator for unsupervised garment pose-space deformation. *ACM Trans. on Graphics (SIGGRAPH Asia 2021)* **40** (2021).

87. Le, B. H. & Deng, Z. Robust and accurate skeletal rigging from mesh sequences. *ACM Trans. on Graphics (SIGGRAPH 2014)* **33,** 1–10 (2014).

88. Liu, L., Zheng, Y., Tang, D., Yuan, Y., Fan, C. & Zhou, K. Neuroskinning: Automatic skin binding for production characters with deep graph networks. *ACM Trans. on Graphics (SIGGRAPH 2019)* **38,** 1–12 (2019).

89. Deng, B., Lewis, J. P., Jeruzalski, T., Pons-Moll, G., Hinton, G., Norouzi, M., *et al. NASA: Neural articulated shape approximation* in *Proc. of Euro. Conf. on Comp. Vision (ECCV)* (2020), 612–628.

90. Xu, Z., Zhou, Y., Kalogerakis, E., Landreth, C. & Singh, K. Rignet: Neural rigging for articulated characters. *ACM Trans. on Graphics (SIGGRAPH 2020)* **39,** 14 (2020).

91. Li, P., Aberman, K., Hanocka, R., Liu, L., Sorkine-Hornung, O. & Chen, B. Learning skeletal articulations with neural blend shapes. *ACM Trans. on Graphics (SIGGRAPH 2021)* **40,** 1–15 (2021).

92. Wrap3. *Nonlinear Iterative Closest Point mesh registration software* https://www.russian3dscanner.com. 2018.

93. Amberg, B., Romdhani, S. & Vetter, T. *Optimal Step Nonrigid ICP Algorithms for Surface Registration* in *Conf. on Computer Vision and Pattern Recognition (CVPR)* (2007).

94. Jacobson, A., Baran, I., Popović, J. & Sorkine, O. Bounded biharmonic weights for real-time deformation. *ACM Trans. on Graphics (TOG)* **30,** 78 (2011).

95. Bouaziz, S., Martin, S., Liu, T., Kavan, L. & Pauly, M. Projective Dynamics: Fusing Constraint Projections for Fast Simulation. *ACM Trans. Graph.* **33,** 154:1–154:11 (2014).

96. Sumner, R. W. & Popović, J. Deformation transfer for triangle meshes. *ACM Trans. on Graphics (SIGGRAPH 2004)* **23,** 399–405 (2004).

97. Artelys. *Knitro* https://www.artelys.com/solvers/knitro/. 2019.

98. Leal, Allan. *autodiff* https://github.com/autodiff/autodiff/. 2018.

99. Fedorov, A., Beichel, R., Kalpathy-Cramer, J., Finet, J., Fillion-Robin, J., Pujol, S., *et al.* 3D Slicer as an image computing platform for the Quantitative Imaging Network. *Magnetic Resonance Imaging* **30,** 1323–1341 (2012).

100. Volino, P., Magnenat-Thalmann, N. & Faure, F. A Simple Approach to Nonlinear Tensile Stiffness for Accurate Cloth Simulation. *ACM Trans. Graph.* **28** (2009).

101. Comley, K. & Fleck, N. A. A micromechanical model for the Young's modulus of adipose tissue. *Int. J. of Solids and Structures* **47,** 2982–2990 (2010).

102. Müller, M., Heidelberger, B., Teschner, M. & Gross, M. *Meshless Deformations Based on Shape Matching* in *Proc. of ACM SIGGRAPH 2005* (2005), 471–478.

103. Wang, B., Matcuk, G. & Barbič, J. *Hand MRI dataset* http://www.jernejbarbic.com/hand-mri-dataset. 2020.

104. LeapMotion. https://www.leapmotion.com. 2017.

105. Lugaresi, C., Tang, J., Nash, H., McClanahan, C., Uboweja, E., Hays, M., *et al.* *MediaPipe: A Framework for Building Perception Pipelines* 2019.

106. Hadwiger, M., Kniss, J. M., Rezk-salama, C., Weiskopf, D. & Engel, K. *Real-Time Volume Graphics* (A. K. Peters, Ltd., 2006).

107. Zheng, M., Wang, B., Huang, J. & Barbič, J. Simulation of Hand Anatomy Using Medical Imaging. *ACM Trans. on Graphics (SIGGRAPH Asia 2022)* **41** (2022).

108. Babuška, I., Caloz, G. & Osborn, J. E. Special Finite Element Methods for a Class of Second Order Elliptic Problems with Rough Coefficients. *SIAM Journal on Numerical Analysis* **31,** 945–981 (1994).

109. Ohtake, Y., Belyaev, A., Alexa, M., Turk, G. & Seidel, H.-P. Multi-level partition of unity implicits. *ACM Trans. on Graphics (SIGGRAPH 2003)* **22,** 463–470 (2003).

110. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., *et al.* PyTorch: An Imperative Style, High-Performance Deep Learning Library. *arXiv preprint arXiv:1912.01703* (2019).

111. Mixamo. *Adobe Mixamo project, https://www.mixamo.com* 2024.

112. Xu, H. & Barbič, J. Signed Distance Fields for Polygon Soup Meshes. *Graphics Interface 2014* (2014).

113. Boissonnat, J.-D. & Oudot, S. Provably good sampling and meshing of surfaces. *Graphical Models* **67,** 405–451 (2005).

114. Hang Si. *TetGen: A Quality Tetrahedral Mesh Generator and a 3D Delaunay Triangulator* 2011.

115. Kingma, D. & Ba, J. *Adam: A Method for Stochastic Optimization* in *Int. Conf. on Learning Representations (ICLR)* (2015).