UCLA

UCLA Electronic Theses and Dissertations

Title

Novel Methods for Efficient Musculoskeletal-Driven Skin Deformation of Animated Characters

Permalink

https://escholarship.org/uc/item/4fk0x7n8

Author

Han, Yushan

Publication Date

2024

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

UNIVERSITY OF CALIFORNIA

Los Angeles

Novel Methods for Efficient Musculoskeletal-Driven Skin Deformation of Animated Characters

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy in Mathematics

by

Yushan Han

2024

© Copyright by

Yushan Han

2024

ABSTRACT OF THE DISSERTATION

Novel Methods for Efficient Musculoskeletal-Driven
Skin Deformation of Animated Characters

by

Yushan Han
Doctor of Philosophy in Mathematics
University of California, Los Angeles, 2024
Professor Joseph M. Teran, Co-Chair
Professor Chenfanfu Jiang, Co-Chair

We present a comprehensive neural network to model the deformation of human soft tissues including muscle, tendon, fat and skin. Our approach provides kinematic and active correctives to linear blend skinning [MLT89] that enhance the realism of soft tissue deformation at modest computational cost, aiming to revolutionize character animation in the context of metaverse and game development. Our network accounts for deformations induced by changes in the underlying skeletal joint state as well as the active contractile state of relevant muscles. Training is done to approximate quasistatic equilibria produced from physics-based simulation of hyperelastic soft tissues in close contact. We use a layered approach to equilibrium data generation where deformation of muscle is computed first, followed by an inner skin/fascia layer, and lastly a fat layer between the fascia and outer skin. We show that a simple network model which decouples the dependence on skeletal kinematics and muscle activation state can produce compelling behaviors with modest training data burden. Active contraction of muscles is estimated using inverse dynamics where muscle moment arms are accurately predicted using the neural network to model kinematic musculotendon geometry. Results demonstrate the ability to accurately replicate compelling musculoskeletal and skin deformation behaviors over a representative range of motions, including the effects of added weights.

Additionally, we present significant advancements in several related areas of the pipeline including a dynamic mode capture paradigm for augmenting secondary effects on top of our network, a robust meshing algorithm for generating volumetric hexahedron meshes from self-intersecting surfaces and a novel position-based nonlinear Gauss-Seidel approach for quasistatic simulation.

The dissertation of Yushan Han is approved.

Christopher R. Anderson

Andrea Bertozzi

Chenfanfu Jiang, Committee Co-Chair

Joseph M. Teran, Committee Co-Chair

University of California, Los Angeles

2024

To my grandmother, Gao Wei, who instilled in me the value of education

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to thank my advisor, Professor Joseph Teran, for his support, instruction, knowledge and time throughout the years of research. Additionally, I extend my thanks to the members of my doctoral committee: Professors Chenfanfu Jiang, Andrea Bertozzi, and Chris Anderson.

I am also grateful to all my fellow lab members who shared this journey with me, providing invaluable collaboration and emotional support: Yizhou Chen, Osman Akar, Jingyu Chen, Steven Gagniere, Alan Marquez-Razon, Victoria Kala, Ayano Kaneda, and Elias Gueidon. Special thanks go to my close friends Siyao Wu, Yongkang Du, and Lei Jin, whose Mamba mentality has been a constant source of motivation.

I want to express my gratitude to my parents and grandparents for their constant love, support, and encouragement. They demonstrated the values of education, work ethic and happiness with examples.

Lastly, I owe a profound debt of love and gratitude to my wife Jiaqi Cheng. We have been through countless happy times as well as some rough times together, and I would not be able to make through the latter without her unwavering belief in me. Our three cats, Frolic, Taizi and Feiwu, are credited for the help with code debugging.

VITA

2019	B.S. (Mathematics), UC Irvine
2020	Graduate Research Intern, Lawrence Berkeley National Lab
2021-2024	Research Intern, Epic Games
2020-2024	Teaching Assistant, Department of Mathematics, UCLA

PUBLICATIONS

Yongxu Jin, Yushan Han, Zhenglin Geng, Joseph Teran, and Ronald Fedkiw. 2022. Analytically Integratable Zero-restlength Springs for Capturing Dynamic Modes unrepresented by Quasistatic Neural Networks. In ACM SIGGRAPH 2022 Conference Proceedings (SIGGRAPH '22). Association for Computing Machinery, New York, NY, USA, Article 37, 1–9.

Yizhou Chen, Yushan Han, Jingyu Chen, Shiqian Ma, Ronald Fedkiw, and Joseph Teran. 2023. Primal Extended Position Based Dynamics for Hyperelasticity. In Proceedings of the 16th ACM SIGGRAPH Conference on Motion, Interaction and Games (MIG '23). Association for Computing Machinery, New York, NY, USA, Article 21, 1–10.

Alan Marquez Razon, Yizhou Chen, Yushan Han, Steven Gagniere, Michael Tupek, and Joseph Teran. 2023. A Linear and Angular Momentum Conserving Hybrid Particle/Grid Iteration for Volumetric Elastic Contact. Proc. ACM Comput. Graph. Interact. Tech. 6, 3, Article 44 (August 2023), 25 pages.

Steven Gagniere, Yushan Han, Yizhou Chen, David Hyde, Alan Marquez-Razon, Joseph Teran, and Ronald Fedkiw. 2024. A Robust Grid-Based Meshing Algorithm for Embedding Self-Intersecting Surfaces. Computer Graphics Forum, 43: e14986.

CHAPTER 1

Introduction

1.1 Musculoskeletal-Driven Skin Deformation Neural Network

Animation of human body motion is one of the most important aspects of computer graphics, and when animations accurately represent the underlying physics and physiology of movement, they can even have broad applications in biological and clinical research. Motion is typically created at the skeleton level, with the outer skin kinematically driven by the motion of underlying bones. The highest level of realism is achieved with biomechanical modeling and physical simulation of soft tissues like the muscle, tendon and fat that lie between the bones and the visible skin [Ng 98, LSN13, FLP14, PLF14, MZS11, LST09, SGK18, ARM19]. However, this requires expensive modeling and simulation which is not feasible in real-time and interactive applications. Recent approaches have shown that neural networks can be used to create efficient character rigs trained to approximate expensive simulation-based techniques [BOD18, LMR15, SGO20, JHG22, CMM20]. In these approaches, a neural network typically provides a trained delta corrective to an efficient technique like linear blend skinning (LBS) [MLT89]. While promising, past approaches have failed to incorporate how muscles activate and deform, a key driver of the skin and body deformation we observe in the real world. We show that a neural network model is indeed able to capture these effects.

Skeleton-driven character animation techniques that utilize simulation of soft tissues come in two basic levels of detail: those that include anisotropic musculotendon modeling [LSN13, FLP14, PLF14, MZS11, LST09] and those that utilize a simple isotropic flesh layer between bones and the outer skin [TSB05, MZS11, SGK18, BOD18, JHG22]. Inclusion of anisotropic muscle and tendon allows for greater realism in practical skin deformation, but is comparatively expensive. To date, a few neural network rig models have included the effects of muscle and tendon [LZJ22, CMM20], but most only consider a simple isotropic layer of tissue between skin and bone [BOD18, LMR15, SGO20, JHG22]. Since muscles generate force not only based on passive stretching, but also through active contraction of the fibers, musculotendon dynamics affect how muscles deform. While a few learning-based approaches have included musculotendon anatomy, our neural network model is the first to incorporate musculotendon dynamics to estimate active contraction of muscle, adding a new dimension of realism since skin deformation appears more alive and less cadaveric. This is particularly evident for lean, low body fat percentage characters and when the character is lifting heavy objects.

Muscle activation reflects the degree to which muscles actively contract due to neurological stimulation. Estimation of these activations from observed skeletal kinematics has a long history in biomechanics research [DAA07]. A common approach is to first solve the inverse dynamics problem to calculate joint torques based on skeletal kinematics and external forces. Then, using these joint torques and a model's muscle geometry, activations are estimated through optimization methods that distribute the force needed to generate the joint torques across many muscles [CB81]. Typically, models use piece-wise linear approximations to approximate musculotendon geometry, but these are not accurate enough to capture the complex, non-linear interactions between muscle fibers needed for realistic deformation [DLH90, BPD05]. Ideally, the finite element method (FEM) can be used to resolve these interactions for optimal accuracy; however, the cost is considerable [BPD05].

We show that a machine learning model can be used to compress the muscle deformation data generated over a wide range of FEM simulations with the efficiency of piece-wise linear models and the accuracy of FEM. We first capture inactive, cadeveric muscle deformation with a passive neural network (PNN) model like many used in the literature

[BOD18, LMR15, SGO20, JHG22, CMM20]. Specifically, our network learns a corrective delta applied to standard LBS deformation of musculotendon geometry designed to better capture deformations observed with FEM simulation. To capture the effects of active contraction, we train a second active neural network (ANN) to resolve the deformation of each muscle as it is activated over a representative range of values. The PNN and ANN decouple the dependence on skeletal kinematic and muscle activation states to reduce the volume of training data required in practice. We couple them together by linear blend skinning the active deformation generated by the ANN forward to the given skeletal state. We demonstrate the efficacy of our approach in a number of representative character animations, including body building with varying body composition and amount of lifted weight. Our results demonstrate considerable gains in realism over standard LBS techniques with modest additional costs. Our primary contributions are as follows:

- A passive neural network for estimating muscle fiber lines of action in inverse dynamics and activation calculations.
- Decoupled passive and active networks for skeleton driven soft tissue deformation with tractable training data burden.
- Biomechanics-based estimation of muscle activation that reproduces observed muscle activity for several common movements.
- A decoupled muscle/fascia/fat model to generate simulated training data.
- Control of body fat percentage during skinning.

1.2 Analytically Integratable Zero-restlength Springs for Dynamic Mode Capture

Recently, there has been a lot of interest in using neural networks to approximate dynamic simulation (see e.g. [HDD19, PFS20, SGP20, SGO20]), especially because neural network inference has the potential to run in real-time (on high-end GPUs). Unfortunately, one requires an exorbitant amount of training data in order to represent all the possible temporal transitions between states that these networks aim to model. These networks do not typically generalize well when not enough training data is used. Even if one had access to the exorbitant amount of training data required, an unwieldy amount of network parameters would be required to prevent underfitting.

Some aspects of a dynamic simulation depend mostly on the configuration, whereas others more strongly depend on the time history of configuration to configuration transitions; thus, we propose the following paradigm. Firstly, we construct a neural network that depends only on the configurations (and as such cannot capture dynamic modes). Secondly, we subtract this configuration-only model from the full dynamics in order to obtain a dynamics layer. Thirdly, we propose a dynamic simulation model that can approximate the dynamics layer. Theoretically, a well-approximated dynamics layer has the potential to augment the configuration-only neural network in a way that exactly matches the original simulations. Moreover, if the configuration-only neural network can capture enough of the non-linearities, then the dynamics layer has the potential to be quite simple (and thus real-time). In this paper, we use the PNN (see Section 1.1) as the configuration-only neural network.

Although we expect that an entire cottage industry could be developed around the modelling and real-time simulation of dynamics layers, we propose only a very simple demonstrational model here (but note that it works surprisingly well). Importantly, the zero-restlength spring approximation to the dynamics layer can be integrated analytically and thus has zero truncation error and no time step stability restrictions, making it quite fast and accurate.

Furthermore, one can (automatically) robustly learn spring constitutive parameters from a very small amount of dynamic simulation data.

1.3 Volumetric Meshing Algorithm for Self-intersecting Surfaces

In many computer graphics and computational mechanics applications, it is necessary to create a volumetric mesh associated with the interior of an input polygonal surface mesh. Most commonly a volumetric tetrahedron mesh is created whose boundary coincides topologically and/or geometrically with an input triangle mesh [MBT03, LS07, HZG18, Si15]. A volumetric embedding mesh that contains the input surface but whose boundary is different than an input triangle mesh is also commonly used [SDF07, TBF19, KBT17, TSB05]. It is generally required that the surface mesh be closed and orientable. It is also generally required that the surface mesh is free of self-intersection or overlap. While the closed and orientable requirements are relatively easy to satisfy in practice, the self-intersection constraint is more challenging, particularly near regions of high-curvature. In many computer graphics applications, this constraint can be violated without any artifacts since the overlap regions are not visible, however most volumetric mesh creation techniques either break down or give numerically "glued" meshes if the constraint is violated. Even intersection free, but nearly intersecting meshes can cause problems for many volumetric mesh creation techniques.

While many surface geometry creation techniques address the importance of its prevention [HPS11, FTS06, Att10, ACW06, GD01], as noted in e.g. [SJP13, LB18], self-intersecting surface meshes are common in practice. Often those involved in the surface geometry creation process are not involved in volumetric simulation or similar down-stream portions of the production pipeline and introduction of self-intersecting regions arises from a lack of communication. Furthermore, completely removing all regions of self-intersection is often deemed not worthy of the effort since it can significantly increase modeling time. In some cases it is even desirable to have an overlapping input surface. E.g. it is desirable to have overlapping lips in the neutral pose of a deformable volumetric face mesh since lips resting in non-overlapping contact are not in a stress free state [CBE15, CBF16]. It should be noted that although in practice a non-negligible number of slightly overlapping or nearly overlapping regions are common, generally the intersection-free constraint is not violated to an extreme degree with overlap regions typically having minimal volume.

Various approaches have developed volumetric mesh creation techniques specifically designed to be robust to self-intersecting [SJP13, LB18] or nearly self-intersecting [TSB05, LB18] input surfaces. Sacht et al. [SJP13] use conformalized mean-curvature flow (cMCF) to first evolve the surface to a self-intersection-free state from which the flow is reversed, attracting the surface to its original, self-intersecting state but with a collision prevention safeguard. This defines an intersection free counterpart to the original input surface which can be meshed with standard techniques. Li and Barbič [LB18] create embedding tetrahedron meshes from unmodified surface meshes with self-intersection by computing locally-injective immersions that can be used to unambiguously duplicate embedded mesh regions near overlaps. They sew these duplicated regions together using a technique inspired by the Constructive Solid Geometry (CSG) approaches in [TSB05, SDF07] but with reduced use of expensive exact precision arithmetic. Teran et al. [TSB05] use an element duplication/sewing technique to create embedding tetrahedron meshes for nearly intersecting input surfaces meshes.

We design an approach for the construction of a uniform-grid-based embedding hexahedron mesh counterpart $\mathcal{V}$ to an input triangulated surface mesh $\mathcal{S}$ that is well-defined (i.e. free from numerical mesh "gluing" artifacts) when the surface is self-intersecting. As in [SJP13], we assume there exists a nearby non-self-intersecting mesh $\tilde{\mathcal{S}}$ and a mapping $\boldsymbol{\phi}^S_{\tilde{S}} : \tilde{\mathcal{S}}^V \to \mathbb{R}^3$ with non-singular Jacobian determinant (see Figure 1.1). Here $\tilde{\mathcal{S}}^V$ is the unambiguously defined interior of the non-self-intersecting $\tilde{\mathcal{S}}$. Intuitively, if we can find a mapping $\boldsymbol{\phi}^S_{\tilde{S}}$ then we can define a volumetric embedding mesh for $\tilde{\mathcal{S}}$ unambiguously with standard techniques and then push it forward under the mapping. However, unlike Sacht et

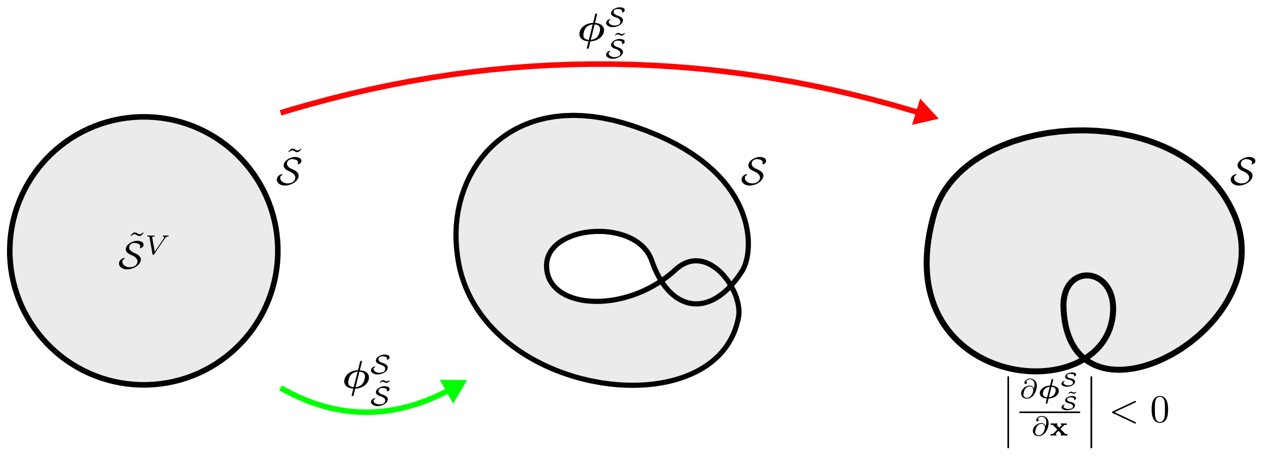

$\phi_{\tilde{\mathcal{S}}}^{\mathcal{S}}$
$\tilde{\mathcal{S}}$
$\tilde{\mathcal{S}}^V$
$\mathcal{S}$
$\mathcal{S}$
$\phi_{\tilde{\mathcal{S}}}^{\mathcal{S}}$
$\left|\frac{\partial \phi_{\tilde{\mathcal{S}}}^{\mathcal{S}}}{\partial \mathbf{x}}\right| < 0$

Figure 1.1: **Intersection-free mapping.** Two mappings from a non-self-intersecting region $\tilde{\mathcal{S}}^V$ to self-intersecting boundary $\mathcal{S}$ are shown. The second mapping (right) requires the existence of a negative Jacobian determinant.

al. [SJP13], we do not explicitly create $\boldsymbol{\phi}_{\tilde{\mathcal{S}}}^{\mathcal{S}}$ or $\tilde{\mathcal{S}}$ but rather use their existence to guide our mesh creation strategy.

We build our embedding hexahedron mesh $\mathcal{V}$ from the intersection of the input surface $\mathcal{S}$ with a uniform background grid where cells in contiguous regions are copied to form sub-meshes that are sewn together using techniques inspired by Teran et al. [TSB05] and Sifakis et al. [SDF07] but in a manner designed to mimic the image of $\boldsymbol{\phi}_{\tilde{\mathcal{S}}}^{\mathcal{S}}$. Our approach is ultimately similar to that of Li and Barbič [LB18] in that we create the volumetric embedding mesh without modifying the self-intersecting surface and our region duplication/sewing is equivalent to discovering immersions. Unlike [LB18], our approach uses nearly no exact and/or adaptive precision arithmetic as we do not resolve the geometry of intersection from triangles in $\mathcal{S}$ with themselves or with cells in the background grid and we do not use CSG operations as in [SDF07]. We simply require accurate determination of which triangles intersect which grid cells. This limits the accuracy of our method for large grid spacing (low-resolution) and we run with smaller grid spacing (high-resolution) when necessary. To prevent this from causing excessive element counts, we provide a topology-preserving mesh coarsening strategy similar to that of Wang et al. [WJS14]. Lastly, we provide a technique for efficiently converting the uniform-grid-based embedding hexahedron mesh to a tetrahedron mesh that robustly handles duplicated regions of the hexahedron mesh near self-intersecting features. As in [LB18], we use a body-centered cubic (BCC) structure [MBT03] for this conversion.

1.4 Position-Based Nonlinear Gauss-Seidel for Quasistatic Hyperelasticity

We consider large strain hyperelastic solids [BW08] whose governing equations are discretized in space with the finite element method (FEM) [SB12]. Our primary focus is quasistatic problems with negligible inertial effects. Quasistatic solvers are becoming increasingly important due to their use in generating training data for PNNs (See Section 1.1). For example, various authors have shown that PNNs can be effectively trained for elastic materials in cloth and skinning applications [JHG22, BME21, GJF20, JZG20, LSW20, BOD18, CMM20]. These networks engender real-time performance at resolutions orders of magnitude above what is achievable with any existing simulation techniques on modern hardware. However, PNNs require tens of thousands of high-resolution equilibria for training data sets. While the creation of these data sets is an "off-line" process, it is desirable to create them with minimal user interaction and computation time. Furthermore, extremely accurate solutions to the governing equations are not necessary since the network need only approximate visually convincing behaviors. Therefore, simulation techniques that generate visually plausible behavior in a minimal amount of computation with minimal user interaction/parameter tuning are ideal. While many methods exist for solving the FEM-discretized implicit equations of motion for hyperelastic solids (see Zhu et al. [ZBK18] and Li et al. [LGL19] for recent summaries), the Position Based Dynamics (PBD) approach of Müller et al. [MHH07] is a natural candidate for generating training data for PNNs. It has remarkable robustness and

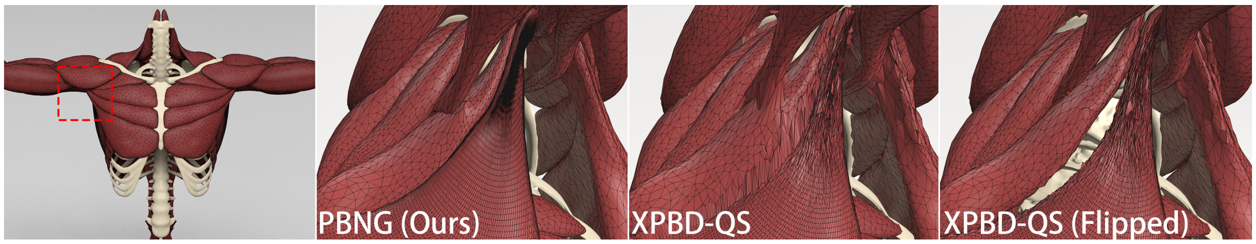


Figure 1.2: **PBNG vs XPBD**. Muscle simulation demonstrates iteration-order-dependent behavior with XPBD and quasistatics. A zoom-in view under the right armpit region is provided. Each method is run 130 iterations. PBNG converges to the desired solution, binding the muscles closely together. XPBD-QS and XPBD-QS (Flipped) fail to converge, leaving either artifacts or gaps between the muscles.

stability properties and can produce visually plausible results with minimal computational budgets. However, constitutive control over PBD behavior is challenging as effective material stiffnesses etc. vary with iteration count and time step size. The Extended Position Based Dynamics (XPBD) approach of Macklin et al. [MMC16] addressed these issues by reformulating the original PBD approach in terms of a Gauss-Seidel technique for discretizing a total Lagrange multiplier formulation of the backward Euler system for implicit time stepping. This formulation has similarities to PBD, but with the elastic terms handled properly where PBD can be seen as the extreme case of infinite elastic modulus.

Despite its many strengths, PBD/XPBD has a few limitations that hinder its use in quasistatic applications. First, XPBD is designed for backward Euler and omitting the inertial terms for quasistatics is not possible (it would require dividing by zero). Indeed Chentanez et al. [CMM20] generate quasistatic training data with XPBD by running backward Euler simulations to steady state. We show that PBD when viewed as the limit of infinite stiffness in XPBD (as detailed in Macklin et al. [MMC16]) is an approximation to the quasistatic equations. Unfortunately, this limit incorrectly and irrevocably removes the external forcing terms. Second, PBD/XPBD can only discretize hyperelastic models that are quadratic in some notion of strain constraint [MMC16, MM21]. This prevents the adoption of many models from the computational mechanics literature. Lastly, as noted in Chen et al. [CHC23] the constraint-centric Gauss-Seidel iteration in PBD/XPBD does not reliably reduce time stepping system residuals. We show that in quasistatic problems this causes artifacts near vertices that appear in different types of constraints (see Figure 1.2).

We present a position-based (rather than constraint-based) nonlinear Gauss-Seidel method that resolves the key issues with PBD/XPBD and hyperelastic quasistatic time stepping. In our approach, we iteratively adjust the position of each simulation node to minimize the potential energy (with all other coupled nodes fixed) in a Gauss-Seidel fashion. This makes each position update aware of all constraints that a node participates in and removes the artifacts of PBD/XPBD that arise from processing constraints separately. Our approach maintains the essential efficiency and robustness features of PBD and has an accuracy that rivals Newton's method for the first few orders of magnitude in residual reduction. Furthermore, unlike Newton's method, our approach is stable when the computational budget is extremely limited. Lastly, since our approach is based on Gauss-Seidel, we show that its convergence is naturally accelerated with successive over relaxation (SOR), Chebyshev and novel multiresolution-based techniques.

The minimization involved in the position update of each node amounts to a nonlinear system of equations (3 equations in 3D and 2 in 2D). We approximate the solution with Newton's method. The linearization of hyperelastic terms can have symmetric indefinite matrices. We develop an inexpensive yet effective technique for projecting any isotropic potential energy density Hessian to a symmetric positive definite counterpart as in [TSI05]. However, unlike the definiteness projections in [TSI05] and [SGK19], it does not require the singular value decomposition of the deformation gradient. Furthermore, unlike the definiteness projection in [TSI05], it does not require the solution of 3×3 or 2×2 symmetric eigensystems. As with PBD and other Gauss-Seidel approaches, a degree of freedom coloring technique is needed for efficient parallel performance. We provide a simple approach for this coloring and show that the position-based view tends to have far fewer colors than the constraint-based view in PBD and that this improves scalability and performance. Lastly, although our technique is designed for quasistatics, it is easily applicable to backward Euler discretizations of problems with inertia if we minimize the incremental potential [MTG11, GSS15, LBO13, SD06, BML14, NOB16] rather than the potential energy.

1.5 Dissertation Overview

Chapter 2 provides a brief overview of continuum mechanics theories and governing equations. Constitutive models, finite element discretization and weak constraints are discussed within the context of this thesis.

Chapter 3 presents a comprehensive neural network to model the deformation of human soft tissues including muscle, tendon, fat and skin. Our approach provides kinematic and active correctives to linear blend skinning [MLT89] that enhance the realism of soft tissue deformation at modest computational cost, aiming to revolutionize character animation in the context of metaverse and game development. This chapter is based on [HCO24].

Chapter 4 presents a simple decoupled analytically integratable zero-restlength spring model for modeling certain types of dynamic simulation, augmenting secondary effects such as inertia on top of our network. This chapter is based on [JHG22].

Chapter 5 discusses a meshing algorithm for creating volumetric embedding hexahedron mesh from a self-intersecting input triangle mesh as the preparation for hyperelastic simulation. This chapter is based on [GHC24].

Chapter 6 discusses a position-based nonlinear Gauss-Seidel approach that resolves a number of issues with the PBD method [MHH07], particularly in the quasistatic simulation setting when generating training data for the network. This chapter is based on [CHC24].

CHAPTER 2

Continuum Mechanics

2.1 Governing Equations

We consider continuum mechanics conceptions of the governing physics where a flow map $\boldsymbol{\phi} : \Omega^0 \times [0, T] \rightarrow \mathbb{R}^d$, $d = 2$ or $d = 3$, describes the motion of the material. Here the time $t \in [0, T]$ location of the particle $\mathbf{X} \in \Omega^0 \subset \mathbb{R}^d$ is given by $\boldsymbol{\phi}(\mathbf{X}, t) \in \Omega^t \subset \mathbb{R}^d$ where Ω^0 and Ω^t are the initial and time t configurations of material respectively. The flow map $\boldsymbol{\phi}$ obeys the partial differential equation associated with momentum balance

$$R^0 \frac{\partial^2 \boldsymbol{\phi}}{\partial t^2} = \nabla^{\mathbf{X}} \cdot \mathbf{P} + \mathbf{f}^{\text{ext}} \tag{2.1}$$

where R^0 is the initial mass density of the material, $\mathbf{P}$ is the first Piola-Kirchhoff stress and $\mathbf{f}^{\text{ext}}$ is external force density. This is also subject to boundary conditions

$$\boldsymbol{\phi}(\mathbf{X}, t) = \mathbf{x}_D(\mathbf{X}, t),\ \mathbf{X} \in \partial\Omega^0_D \tag{2.2}$$

$$\mathbf{P}(\mathbf{X}, t)\mathbf{N}(\mathbf{X}, t) = \mathbf{T}_N(\mathbf{X}, t),\ \mathbf{X} \in \partial\Omega^0_N \tag{2.3}$$

where $\partial\Omega^0$ is split into Dirichlet ($\partial\Omega^0_D$) and Neumann ($\partial\Omega^0_N$) regions where the deformation and applied traction respectively are specified. Here $\mathbf{T}_N$ denotes externally applied traction boundary conditions. For hyperelastic materials, the first Piola-Kirchhoff stress is related to a notion of potential energy density $\Psi : \mathbb{R}^{d \times d} \rightarrow \mathbb{R}$ as

$$\mathbf{P}(\mathbf{X}, t) = \frac{\partial \Psi}{\partial \mathbf{F}}(\frac{\partial \boldsymbol{\phi}}{\partial \mathbf{X}}(\mathbf{X}, t)),\ \text{PE}(\boldsymbol{\phi}(\cdot, t)) = \int_{\Omega^0} \Psi(\frac{\partial \boldsymbol{\phi}}{\partial \mathbf{X}}) d\mathbf{X} \tag{2.4}$$

where $\text{PE}(\boldsymbol{\phi}(\cdot, t))$ is the potential energy of the material when it is in the configuration defined by the flow map at time t. Note that we will typically use $\mathbf{F} = \frac{\partial \boldsymbol{\phi}}{\partial \mathbf{X}}$ to denote the spatial derivative of the flow map (or deformation gradient). We refer the reader to [GS08, BW08] for more continuum mechanics detail.

In quasistatic problems, the inertial terms in the momentum balance (Equation (2.1)) can be neglected and the material motion is defined by a sequence of equilibrium problems

$$\mathbf{0} = \nabla^{\mathbf{X}} \cdot \mathbf{P} + \mathbf{f}^{\text{ext}} \tag{2.5}$$

subject to the boundary conditions in Equations (2.2)-(2.3). This is equivalent to the minimization problems

$$\boldsymbol{\phi}(\cdot, t) = \underset{\boldsymbol{\Upsilon} \in \mathcal{W}^t}{\text{argmin}} \; \text{PE}(\boldsymbol{\Upsilon}) - \int_{\Omega_0} \mathbf{f}^{\text{ext}} \cdot \boldsymbol{\Upsilon} d\mathbf{X} - \int_{\partial\Omega_N^0} \mathbf{T}_N \cdot \boldsymbol{\Upsilon} ds(\mathbf{X}) \tag{2.6}$$

where $\mathcal{W}^t = \left\{ \boldsymbol{\Upsilon} : \Omega_0 \to \mathbb{R}^d \,|\, \boldsymbol{\Upsilon}(\mathbf{X}) = \mathbf{x}_D(\mathbf{X}, t), \; \mathbf{X} \in \partial\Omega_D^0 \right\}$.

2.2 Constitutive Models

We demonstrate our approach with a number of different hyperelastic potentials commonly used in computer graphics applications. The "corotated" or "warped stiffness" model [MDM02, EGS03, MG04, ST08, CPS10] has been used for many years with a few variations. We use the version with the fix to the volume term developed by Stomakhin et al. [SHS12]

$$\Psi^{\text{cor}}(\mathbf{F}) = \mu |\mathbf{F} - \mathbf{R}(\mathbf{F})|_F^2 + \frac{\lambda}{2}(\det(\mathbf{F}) - 1)^2. \tag{2.7}$$

Here $\mathbf{F} = \mathbf{R}(\mathbf{F})\mathbf{S}(\mathbf{F})$ is the polar decomposition of $\mathbf{F}$. Neo-Hookean models [BW08] have also been used since they do not require polar decomposition and since recently some have been shown to have favorable behavior with nearly incompressible materials [SGK18]. In Chapter 6, we use the Macklin and Müeller [MM21] formulation due to its simplicity and natural use with XPBD

$$\Psi^{\text{nh}}(\mathbf{F}) = \frac{1}{2}\mu |\mathbf{F}|_F^2 + \frac{\hat{\lambda}}{2}(\det(\mathbf{F}) - 1 - \frac{\mu}{\hat{\lambda}})^2. \tag{2.8}$$

Here $\hat{\lambda} = \mu + \lambda$. λ and μ are the Lamé parameters and are related to the Young's modulus (E) and Poisson's ratio (ν) as

$$\mu = \frac{E}{2(1+\nu)}, \ \lambda = \frac{E\nu}{(1+\nu)(1-2\nu)}. \tag{2.9}$$

We also support the stable Neo-Hookean model proposed in [SGK18]

$$\Psi^{\text{snh}}(\mathbf{F}) = \frac{1}{2}\mu(|\mathbf{F}|_F^2 - d) + \frac{1}{2}(\det(\mathbf{F}) - 1 - \frac{3\mu}{4\lambda})^2 - \frac{1}{2}\mu\log(1 + |\mathbf{F}|_F^2). \tag{2.10}$$

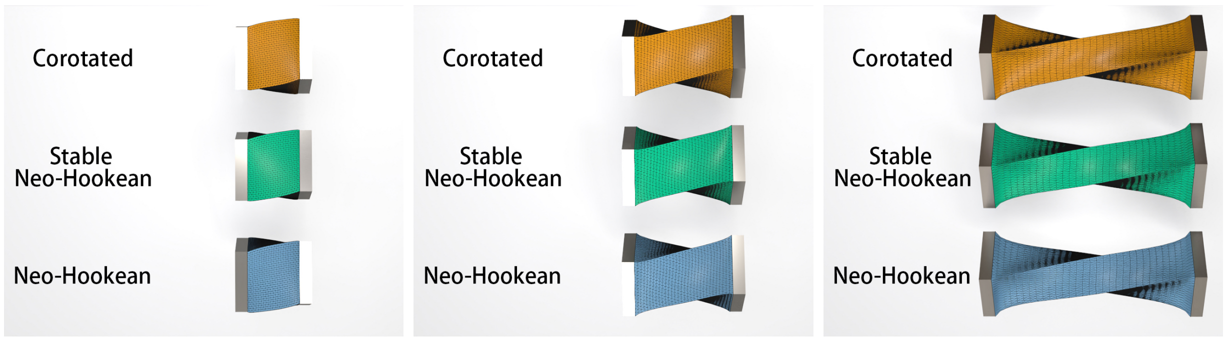


Figure 2.1: **Different Constitutive Models.** PBNG (Chapter 6) works with various constitutive models. We showcase the corotated, Neo-Hookean, and stable Neo-Hoookean models through a block twisting and stretching example.

2.3 Discretization

We use the FEM discretization of the quasistatic problem in Equation (2.5)

$$\mathbf{f}_i(\mathbf{x}^{n+1}) + \hat{\mathbf{f}}_i^{\text{ext}} = \mathbf{0}, \ \mathbf{X}_i \notin \Omega_D^0 \tag{2.11}$$

$$\mathbf{x}_i^{n+1} = \mathbf{x}_D(\mathbf{X}_i, t^{n+1}), \ \mathbf{X}_i \in \Omega_D^0. \tag{2.12}$$

Here the flow map is discretized as $\boldsymbol{\phi}(\mathbf{X}, t^{n+1}) = \sum_{j=0}^{NN-1} \mathbf{x}_j^{n+1} N_j(\mathbf{X})$ where the $N_j(\mathbf{X})$ are piecewise linear interpolating functions defined over a tetrahedron mesh ($d = 3$) or triangle mesh ($d = 2$) and the $\mathbf{x}_j^{n+1} \in \mathbb{R}^d$, $0 \leq j < N^N$ are the locations of the vertices of the mesh at time t^{n+1}. Note that we use $\mathbf{x}^{n+1} \in \mathbb{R}^{dN^N}$ to denote the vector of all vertex locations and $x_{i\beta}^{n+1}$ to denote the $0 \leq \beta < d$ components of the position of vertex i in the mesh. The forces are given as

$$\mathbf{f}_i(\mathbf{y}) = -\frac{\partial \hat{\text{PE}}}{\partial \mathbf{y}_i}(\mathbf{y}) \tag{2.13}$$

$$\hat{\text{PE}}(\mathbf{y}) = \hat{\text{PE}}^{\Psi}(\mathbf{y}) + \hat{\text{PE}}^{\text{wc}}(\mathbf{y}) \tag{2.14}$$

$$\hat{\text{PE}}^{\Psi}(\mathbf{y}) = \sum_{e=0}^{N^E-1} \Psi\Big(\sum_{j=0}^{N^N-1} \mathbf{y}_j \frac{\partial N_j^e}{\partial \mathbf{X}}\Big) V_e^0 \tag{2.15}$$

$$\hat{\mathbf{f}}_i^{\text{ext}} = \int_{\Omega^0} \mathbf{f}^{\text{ext}} N_i d\mathbf{X} + \int_{\partial\Omega_N^0} \mathbf{T}_N N_i ds(\mathbf{X}) \tag{2.16}$$

where $\hat{\text{PE}}^{\Psi} : \mathbb{R}^{dN^N} \rightarrow \mathbb{R}$ is the discretization of the potential energy, $\sum_{j=0}^{N^N-1} \mathbf{y}_j \frac{\partial N_j^e}{\partial \mathbf{X}}$ is the deformation gradient induced by nodal positions $\mathbf{y} \in \mathbb{R}^{dN^N}$ in tetrahedron ($d = 3$) or triangle ($d = 2$) element e with $0 \leq e < N^E$, $\frac{\partial N_i^e}{\partial \mathbf{X}}$ is the derivative of the interpolating function in element e (which is constant since we use piecewise linear interpolation) and V_e^0 is the measure of the element. We refer the reader to [BW08, SB12] for more detail on the FEM derivation of potential energy terms in a hyperelastic formulation. Also, note that we add another term to the discrete potential energy $\hat{\text{PE}}^{\text{wc}} : \mathbb{R}^{dN^N} \rightarrow \mathbb{R}$ in Equation (2.14) to account for self-collisions and similar weak constraints. Similar to the non-discrete case, the constrained minimization problem

$$\mathbf{x}^{n+1} = \underset{\mathbf{y} \in \mathcal{W}_{\Delta x}^{n+1}}{\text{argmin}} \ \hat{\text{PE}}(\mathbf{y}) - \mathbf{y} \cdot \hat{\mathbf{f}}^{\text{ext}} \tag{2.17}$$

where $\mathcal{W}_{\Delta x}^{n+1} = \left\{ \mathbf{y} \in \mathbb{R}^{dN^N} | \mathbf{y}_i = \mathbf{x}_D(\mathbf{X}_i, t^{n+1}), \ \mathbf{X}_i \in \partial\Omega_D^0 \right\}$ is equivalent to Equations (2.11)-(2.12).

2.4 Weak Constraints

We support weak constraints for self-collision and other similar purposes (as in [MZS11]). These are terms added to the potential energy in the form

$$\hat{\mathrm{PE}}^{\mathrm{wc}}(\mathbf{y}) = \frac{1}{2}\sum_{c=0}^{N^{\mathrm{wc}}-1} \mathbf{C}_c(\mathbf{y})^T \mathbf{K}_c \mathbf{C}_c(\mathbf{y}) \tag{2.18}$$

$$\mathbf{C}_c(\mathbf{y}) = \sum_{j=0}^{N^N-1} w^c_{0j}\mathbf{y}_j - w^c_{1j}\mathbf{y}_j. \tag{2.19}$$

Here the w^c_{0j}, w^c_{1j} are interpolation weights that sum to one and are non-negative. This creates constraints between the interpolated points $\sum_{j=0}^{N^N-1} w^c_{0j}\mathbf{y}_j$ and $\sum_{j=0}^{N^N-1} w^c_{1j}\mathbf{y}_j$. The stiffness of the constraint is represented in the matrix $\mathbf{K}_c$. This can allow for anisotropic responses where $\mathbf{K}_c = k_n\mathbf{n}\mathbf{n}^T + k_\tau\left(\boldsymbol{\tau}_0\boldsymbol{\tau}_0^T + \boldsymbol{\tau}_1\boldsymbol{\tau}_1^T\right)$. Here $\mathbf{n}^T\boldsymbol{\tau}_i = 0$, $i = 0, 1$ and k_n is the stiffness in the $\mathbf{n}$ direction while k_τ is the stiffness in response to the motion in the plane normal to $\mathbf{n}$. In the case of an isotropic constraint ($k_c = k_n = k_\tau$), we use the scalar k_c in place of $\mathbf{K}_c$ since $\mathbf{K}_c = k_c\mathbf{I}$ is diagonal. In most of our examples, the isotropic model is used for fascia constraints (See Section 3.3.1.2) and the anisotropic model is used for collision constraints where $\mathbf{n}$ is the collision constrain direction.

CHAPTER 3

Musculoskeletal-Driven Skin Deformation Neural Network

3.1 Related Work

Character animation in visual effects and gaming applications typically makes use of skeleton-driven motion where rig-based bone transforms define skin deformation [MCC11]. The most widespread technique of this type is the linear blend skinning (LBS) of Magnenant-Thalmann et al. [MLT89]. In LBS, vertices move as a linear combination of rigid trajectories based on their proximity to bones. LBS is very effective and is widely adopted; however, it has many known artifacts. Many approaches, including ours, are designed to improve and/or repair artifacts in LBS. We first discuss techniques of this type, then we discuss simulation techniques used to create realistic tissue deformations since we use a similar approach to generate training data. Lastly, we also discuss work related to muscle activation estimation needed for simulation.

3.1.1 Skinning

The addition of extra joints in the rig to help reduce artifacts [MG03, KCO09] and extra scalar weight functions per bone [KS12] have been used to improve LBS. Mancewicz et al. [MDR14] use Laplace smoothing near joints to reduce artifacts. Kavan et al. [KZ05, KCv07] show that loss of rigidity in the per-vertex transforms causes most of the artifacts. Many approaches modify LBS with example-based data generated from desired poses

[WP02, LCF00, MMG06]. Mohr and Gleicher [MG03] add extra joints to better approximate poses/deformed skins from higher-end rigs. Unfortunately, as noted by Bailey et al. [BOD18], subtle biomechanical details like skin sliding and muscle bulging are very hard to capture with LBS alone. Following the SCAPE formulation of Anguelov et al. [ASK05], many techniques have been developed based on motion capture data [CLZ13, PRM15]. Loper et al. [LMR15] use machine learning to enhance LBS realism. The use of such machine learning techniques is increasingly common. These approaches typically separate soft tissue deformation into a linear (LBS) and a nonlinear (neural network) component [SSR20, BOD18, SOC19, SGO20, JHG22, JZG20]. Yuwei et al. [LZJ22] develop a parametric model of the hand with a variation of the approach of Loper et al [LMR15]. This has proven very effective in the creation of virtual hands [CMM20, LAH21] and faces [SWR21].

3.1.2 Simulation

Chen and Zeltzer [CZ92] were the first graphics researchers to use FEM simulation of muscle. Dao and Tho [DT18] give a thorough review of the state-of-the-art in hyperelastic constitutive models used for FEM simulation of skeletal muscle for biomechanics applications. Early efforts used procedural simplifications to model soft tissues underlying an elastic skin [SPC97, WG97]. Ng-Thow-Hing [Ng 98], Teran et al. [TSB05], Sifakis et al. [SNF05] and Lee et al. [LST09] expanded the scale of graphical muscle simulation. Sueda et al. [SKP08] show that musculotendon simulation in the hand greatly enhances visual realism over simple skinning. Modi et al. [MFJ21] design a novel deformation gradient/displacement formulation for more efficiently solving the nonlinear systems of equations associated with quaistatic time stepping for muscluloskeletal simulation. Saito et al. [SZK15] use simulation of fat and muscle layers to generate a range of body types. Recent simulation efforts have focused on the creation of realistic hands [LZJ22, ZWH22, WMB19]. [FLP14, PLF14] use an Eulerian-on-Lagrangian approach to simulating soft tissues in close contact. Li et al. [LSN13] use a similar approach for simulating skin sliding over underlying tissues. Wang et al. [WMB21] develop subject specific FEM simulations based on an anatomically accurate template mesh and plastic deformation. Lee et al. [LJL23] simulate connective tissues that articulate the bones in the upper torso. Smith et al. [SGK18] develop a stable Neo-Hookean model for isotropic skeleton driven flesh simulation. Recent works utilize machine learning approaches to approximate FEM simulation. Kneifl et al. [KRA23] use a low-rank regression model to approximate activation-driven FEM muscle simulation. However, they only consider a static pose (isometric contraction). Romero et al. [RCC22] note that ML can resolve collision in elasticity simulation and demonstrate its efficacy on fingers in close contact. Meister et al. [MPM20] develop data-driven acceleration of simulation with Total Lagrangian Explicit Dynamics (TLED).

Muscle and tendon are anisotropic and require the modeling of a fiber direction field. Ng-Thow-Hing [Ng 98] creates these from a B-spline model. Modi et al. [MFJ21] create fiber directions as the gradients of a potential defined from a heat-equation with Dirichlet boundary conditions applied where muscles attach to bones. These are referred to as origins (attachments that do not move during contraction) and insertions (attachments that move during contraction). Inouye et al. [IHB15] use a similar technique and solve the Laplace equation for a fiber potential with zero Dirichlet boundary conditions on origins and one Dirichlet boundary conditions on insertions. Zero Neumann boundary conditions are used on the remaining portion of the musculotendon boundary. Also, a muscle contracts actively to produce skeletal motion, and determination of activation state from a given motion is an inverse problem. This is done extensively in biomechanics applications [DLH90, DAA07]. Ryu et al. [RKL21] and Wang et al. [WVY22] use curved lines of action associated with the fiber fields when determining muscle activations. In graphics applications, determination of these activations is typically done to create more realistic movement [LT06, JWG19, LPK14, LPL19, WHD12, SKP08]. Very few works have utilized active contraction to improve the realism of skin deformation. However, Sifakis et al. [SNF05] and Ichim et al. [IKK17] did this for facial animation, and Lee et al. [LST09] were the first to do this for the upper torso.

Almost all of these approaches simplify musculotendon geometry in terms of representative curved paths; however, recent approaches have shown the FEM mesh itself can be used [VSP18, LYP18, SNF05]. This is considerably more expensive, but e.g. Blemker et al. [BPD05] have shown that it is more accurate. We note that there are still key gaps that need to be addressed such as speed v.s. accuracy.

3.2 Overview

We define our approach in four layers: (i) the bones in the kinematic rig, (ii) the muscles and tendons that attach to the bones, (iii) a thin connective tissue membrane that wraps the muscles (fascia) and (iv) the fat layer between the outer skin and the inner fascia. The fat layer is the most important visually since its outer surface represents the visible skin. We define the skinning kinematics of the fat layer as

$$\boldsymbol{\phi}^s(\mathbf{X};\boldsymbol{\Theta},\mathbf{a}) = \mathbf{LBS}(\mathbf{X} + \mathbf{PNN}^s(\mathbf{X};\boldsymbol{\Theta}) + \mathbf{ANN}^s(\mathbf{X};\mathbf{a});\boldsymbol{\Theta})$$

where $\mathbf{X} \in \mathbb{R}^3$ is a point in the A-pose (see Figure 3.1 (b3)) of the skin/fat layer, $\boldsymbol{\Theta} \in \mathbb{R}^{N_J \times 9}$ defines the rotation matrices representing joint states of the rig (where N_J is the number of joints), $\mathbf{a} \in \mathbb{R}^{N_M}$ is a vector of muscle activation values (where N_M is the number of muscles). Here $\mathbf{LBS}(\mathbf{Y};\boldsymbol{\Theta}) = \sum_{i=0}^{N_B-1} w_i^{LBS}(\mathbf{Y})\left(\mathbf{R}_i(\boldsymbol{\Theta})\mathbf{Y} + \mathbf{t}_i(\boldsymbol{\Theta})\right)$ is the standard LBS operator where N_B is the number of bones in the skeleton and $(\mathbf{R}_i(\boldsymbol{\Theta}), \mathbf{t}_i(\boldsymbol{\Theta}))$ are the rotation and translation of the i^{th} bone and $w_i^{LBS}(\mathbf{Y})$ is the LBS weight of skin/fat point $\mathbf{Y}$. $\mathbf{PNN}^s$ and $\mathbf{ANN}^s$ are neural networks trained to allow tissue deformation to match data generated from FEM simulation over a range of representative joint states $\boldsymbol{\Theta}$ and activation states $\mathbf{a}$. We note that the effect of the passive neural network $\mathbf{PNN}^s$ on the kinematics is similar to those in [BOD18, JHG22]; however, our key novelty is in the addition of the active network $\mathbf{ANN}^s$. We define the kinematics of the muscle layer $\boldsymbol{\phi}^m(\mathbf{X};\boldsymbol{\Theta},\mathbf{a})$ similarly in terms of passive and active neural networks $\mathbf{PNN}^m$ and $\mathbf{ANN}^m$ respectively.

We provide the details for the creation of passive simulation data of the muscles, tendons, fascia and fat in Section 3.3 as well as the training process and neural network architecture in Section 3.4. Once the muscle kinematics have been defined via trained $\mathbf{PNN}^m$ and $\mathbf{ANN}^m$, we use $\mathbf{PNN}^m$ to define muscle fiber lines of action needed to solve for the activation state as a function of the joint state and external forcing $\mathbf{f}^{\text{ext}}$ (from gravity and any added weight (see Figure 3.1 (c1-c4)). Specifically, activations are chosen in a manner that gives rise to muscle forces capable of balancing torques induced by $\mathbf{f}^{\text{ext}}$ at a given skeletal joint state $\mathbf{\Theta}$. In this case we write $\mathbf{a} = \mathbf{a}(\mathbf{\Theta}, \mathbf{f}^{\text{ext}})$ and the skin/fat kinematics take on the form

$$\boldsymbol{\phi}^s(\mathbf{X}; \mathbf{\Theta}, \mathbf{f}^{\text{ext}}) = \mathbf{LBS}(\mathbf{X} + \mathbf{PNN}^s(\mathbf{X}; \mathbf{\Theta}) + \mathbf{ANN}^s(\mathbf{X}; \mathbf{a}(\mathbf{\Theta}, \mathbf{f}^{\text{ext}})); \mathbf{\Theta}).$$

We detail the activation solution process in Section 3.5.

3.3 Training data: soft tissue simulation

We decouple soft tissue simulation layer-by-layer outwards from the bones. This decoupling accelerates the solution process and allows for more control over the manner in which fat and skin interact with underlying muscles. First, the equilibrium state $\hat{\mathbf{x}}^m(\mathbf{\Theta}, \mathbf{a}) \in \mathbb{R}^{3N_{M^v}}$ of vertices in the musculotendon tetrahedron meshes (where N_{M^v} is the number of musculotendon vertices) are solved for given Dirichlet boundary conditions defined over muscle origin and insertion vertices that move rigidly with the bone transforms as determined by the joint state $\mathbf{\Theta}$. Then, we solve for the fascia layer where vertices sufficiently close to muscle are Dirichlet constrained based on their barycentric locations in the closest triangle in the muscle mesh boundary. Fascia vertices not sufficiently close are put under tension and collide against static muscles and bones to produce a smooth inner boundary of the skin layer. We denote the fascia equilibrium as $\hat{\mathbf{x}}^e(\hat{\mathbf{x}}^m(\mathbf{\Theta}, \mathbf{a}))$ to emphasize its dependence on the muscle equilibrium state. Lastly, we simulate the fat/skin layer with its inner boundary fixed to the fascia equilibrium. We simplify this by defining the topology of the outer skin to match that of the fascia layer. We refer to this equilibrium state as $\hat{\mathbf{x}}^s(\hat{\mathbf{x}}^e(\hat{\mathbf{x}}^m(\mathbf{\Theta}, \mathbf{a})))$. We train the muscle ($\mathbf{PNN}^m$) and fat/skin ($\mathbf{PNN}^s$) passive neural networks on $\mathbf{\Theta}$ using equilibrium

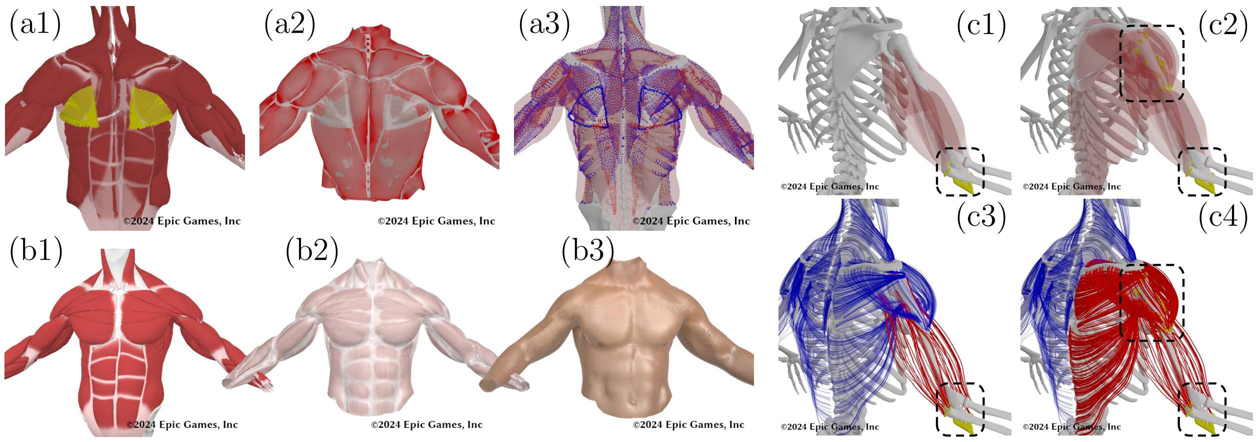


Figure 3.1: **Simulation Setup**. **(a1)**: Connective tissue membrane (in yellow). **(a2)**: Fascia with constrained vertices in red and simulated membrane in grey. **(a3)**: Muscle weak constraint visualization: fascia constraints (in blue) and contact constraints (in red). **(b1)-(b3)**: Reference A-pose for undeformed state definition for muscles, fascia and skin/fat. **(c1)-(c4)**: Streamline forces applied to the bones with respect to elbow joint (left) and shoulder joint (right). Forces are applied on muscle origins and insertions (in yellow).

states $\hat{\mathbf{x}}^m(\boldsymbol{\Theta}, \mathbf{0})$ and $\hat{\mathbf{x}}^s(\hat{\mathbf{x}}^e(\hat{\mathbf{x}}^m(\boldsymbol{\Theta}, \mathbf{0})))$ where the $\mathbf{a} = \mathbf{0}$ muscle activation state indicates completely passive deformation. We provide more detail about the solution process for each layer equilibrium in the subsections that follow.

3.3.1 Musculotendon Equilibrium

Each muscle (together with the tendon) is represented as a volumetric tetrahedron mesh. The equilibrium state $\hat{\mathbf{x}}^m(\boldsymbol{\Theta}, \mathbf{a})$ is determined by minimizing the hyperelastic potential energy

$$\mathrm{PE}^m(\mathbf{x}^m, \mathbf{x}^c; \boldsymbol{\Theta}, \mathbf{a}) = \sum_t \Psi^{\mathrm{cor}}(\mathbf{F}_t(\mathbf{x}^m; \mathbf{a}, \mathbf{D}_t), t)V_t + \sum_{\hat{t}} \frac{\hat{k}}{2}|\mathbf{F}^{\mathrm{cod}}_{\hat{t}}(\mathbf{x}^m)|^2_F A_{\hat{t}} \qquad (3.1)$$
$$+ \sum_j \frac{1}{2}\mathbf{r}_j(\mathbf{x}^m, \mathbf{x}^c)^T \mathbf{K}_j(\mathbf{x}^m, \mathbf{x}^c)\mathbf{r}_j(\mathbf{x}^m, \mathbf{x}^c)$$

subject to Dirichlet constraints

$$\mathbf{x}_i^m(\mathbf{\Theta}) = \mathbf{R}_{i_j}(\mathbf{\Theta})\mathbf{X}_i^m + \mathbf{t}_{i_j}(\mathbf{\Theta}),\ i \in \Omega^D.$$

Here Ω^D refers to the collection of musculotendon indices associated with vertices that are inside bones in the A-pose of the character and $\mathbf{R}_{i_j}(\mathbf{\Theta})\mathbf{X}_i^m + \mathbf{t}_{i_j}(\mathbf{\Theta})$ is the transformation of the constrained vertices $\mathbf{x}_i^m(\mathbf{\Theta})$ under bone transform i_j based on joint state $\mathbf{\Theta}$. Furthermore, $\mathbf{X}_i^m$ is the location of the musculotendon vertex in the A-pose (see Figure 3.1 (b1)). Ψ^{cor} is the hyperelastic potential energy density from Equation 2.7. We also couple in extra connective tissue with vertices we denote as $\mathbf{x}^c \in \mathbb{R}^{3N_c}$ (see Section 3.3.1.3). We model collision and contact between muscle/muscle, muscle/bone and muscle/connective tissue with the constraint direction $\mathbf{r}_j \in \mathbb{R}^3$ and the stiffness matrix $\mathbf{K}_j \in \mathbb{R}^{3\times3}$ (see Section 2.4). Note that the minimization of Equation (3.1) is done simultaneously over both musculotendon $\mathbf{x}^m$ and connective tissue vertices $\mathbf{x}^c$. Furthermore, index t refers to tetrahedra in the musculotendon meshes, $\mathbf{F}_t \in \mathbb{R}^{3\times3}$ is the deformation gradient in the tetrahedron, $\hat{t}$ refers to triangles in connective tissue meshes and $\mathbf{F}_{\hat{t}}^{\text{cod}} \in \mathbb{R}^{3\times2}$ is the deformation gradient in the triangle (see Section 4.1.3). We refer the reader to [TSI05] for more detail about the general approach for solving this nonlinear minimization problem as well as [SB12] for FEM discretization of hyperelastic solids.

3.3.1.1 Musculotendon Fiber Fields

Although muscles are anisotropic and deform under activation state $\mathbf{a}$, we use the isotropic fixed corotated potential Ψ^{cor} from Equation 2.7. Similar to Li et al. [LSN13], we modify the rest state of muscle tetrahedra in a procedural way to introduce active anisotropic deformation in muscle fiber direction $\mathbf{D}_t \in \mathbb{R}^3$ (see Figure 3.3). This is advantageous since convergence properties of the nonlinear solver are generally better for isotropic models. Our active anisotropic modification to the deformation gradient in the tetrahedron t is defined as $\mathbf{F}_t(\mathbf{x}^m; \mathbf{a}, \mathbf{D}_t) = \mathbf{F}_t(\mathbf{x}^m)\mathbf{U}_t(\mathbf{D}_t)\mathbf{\Sigma}(\mathbf{a}_{i(t)})\mathbf{U}_t(\mathbf{D}_t)^T$ where $\mathbf{F}_t(\mathbf{x}^m)$ is the standard deformation gradient in the tetrahedron based on deformation from the A-pose (see Figure 3.1 (b1)), $\mathbf{a}_{i(t)}$ is the activation of the muscle i associated with tetrahedron t and $\mathbf{\Sigma}(\mathbf{a}_{i(t)})$ is diagonal with $\mathbf{\Sigma}_{11}(\mathbf{a}_{i(t)}) = \frac{1+\mathbf{a}_{i(t)}}{1+\gamma\mathbf{a}_{i(t)}}, \mathbf{\Sigma}_{22}(\mathbf{a}_{i(t)}) = \mathbf{\Sigma}_{33}(\mathbf{a}_{i(t)}) = \left(\frac{1+\gamma\mathbf{a}_{i(t)}}{1+\mathbf{a}_{i(t)}}\right)^{\alpha}$. $\mathbf{U}_t(\mathbf{D}_t) = [\mathbf{D}_t, \mathbf{D}_t^1, \mathbf{D}_t^2]$ is a rotation matrix with columns $\mathbf{D}_t^1, \mathbf{D}_t^2$ orthogonal to the fiber direction $\mathbf{D}_t$. γ controls the level of fiber compression and α controls the level of volume preservation during active contraction. We found that $\gamma = 0.4$ and $\alpha = 1$ gave desirable active contractile behavior and visual bulging in practice. In Figure 3.2, we show effects of different choices of α and γ on a fully contracting bicep.

Tendon attaches to bones in the skeleton at origin (proximal) and insertion (distal) locations as shown in Figure 3.3. We manually select origin and insertion vertices in the musculotendon meshes as in [TSB05]. These are used to define fixed vertex boundary conditions when minimizing Equation (3.1). However, we also use origin/insertion points to define fiber directions in each tetrahedron in each musculotendon mesh as in [IHB15]. We discretize the Laplace equation in their model with FEM and piecewise linear interpolation over the musculotendon tetrahedron mesh [Hug00]. Fiber directions are then defined as the gradient of this potential evaluated in each tetrahedron. For inverse dynamics and activation calculations, we need to know the paths of fibers from origin to insertion to estimate length-based force capacity at a given joint and activation state. We use the fiber direction field defined by the per-tetrahedra $\mathbf{U}_t$ to define a flow field and create fiber streamlines that pass from origin to insertion. We randomly sample one fiber streamline starting point inside each origin tetrahedron and create 2624 streamlines on 46 muscles (see Figure 3.1 (c3-c4)). We illustrate this process in Figure 3.3 and note that fiber direction and streamline creation is a pre-computation done once in the A-pose (and then deformed barycentrically in the musculotendon tetrahedron mesh).

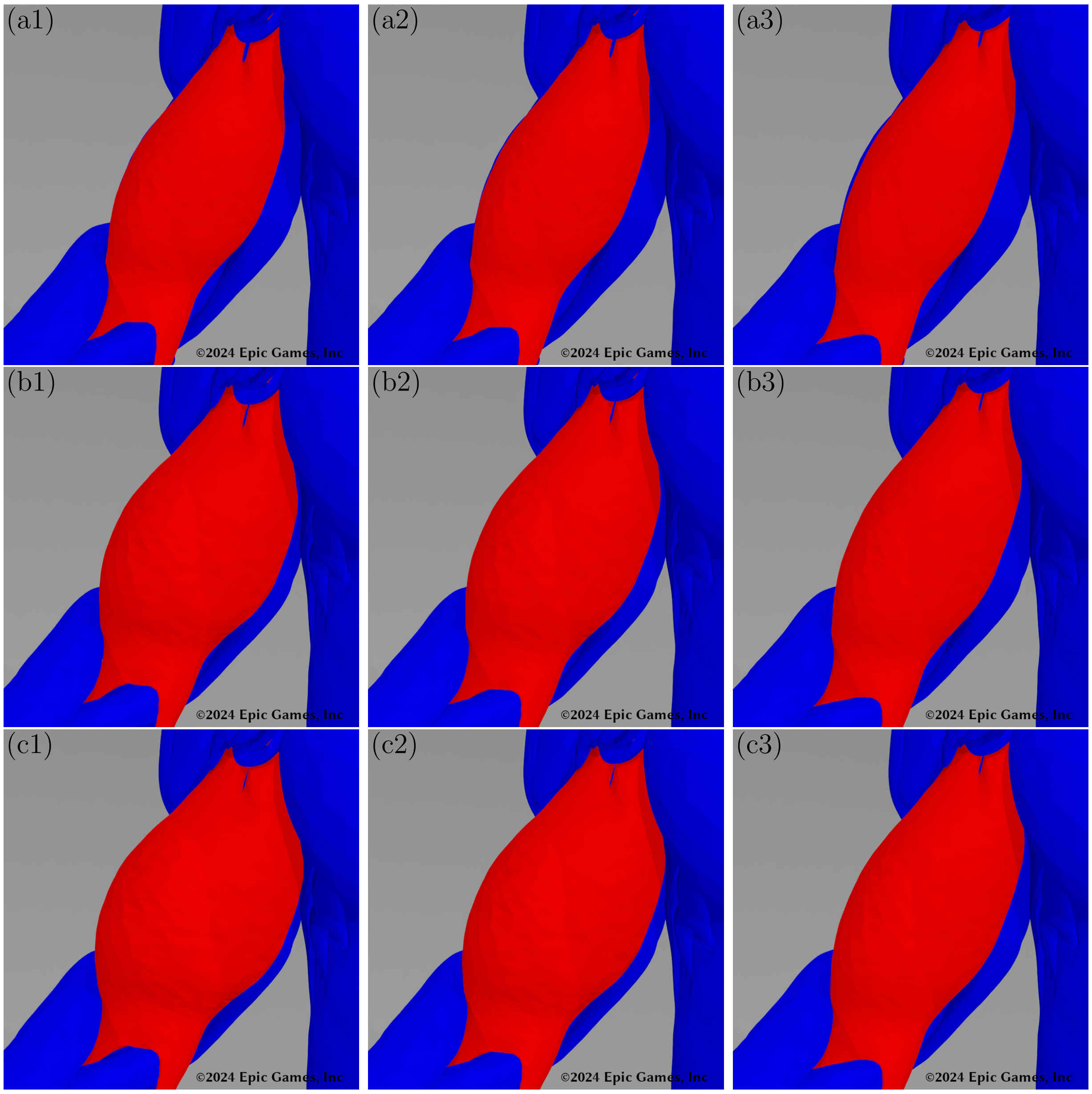


Figure 3.2: **Fiber Compression and Volume Preservation Parameters**. **Row (a-c)**: Volume preservation $\alpha = 0.5, 1, 1.2$. **Column (1-3)**: Fiber compression $\gamma = 0.3, 0.4, 0.6$.

3.3.1.2 Contact Model

We adopt the model discussed in Section 2.4 for the weak constraints used in Equation (3.1) to model the effects of contact between muscles and connective tissue like fascia. Fascia

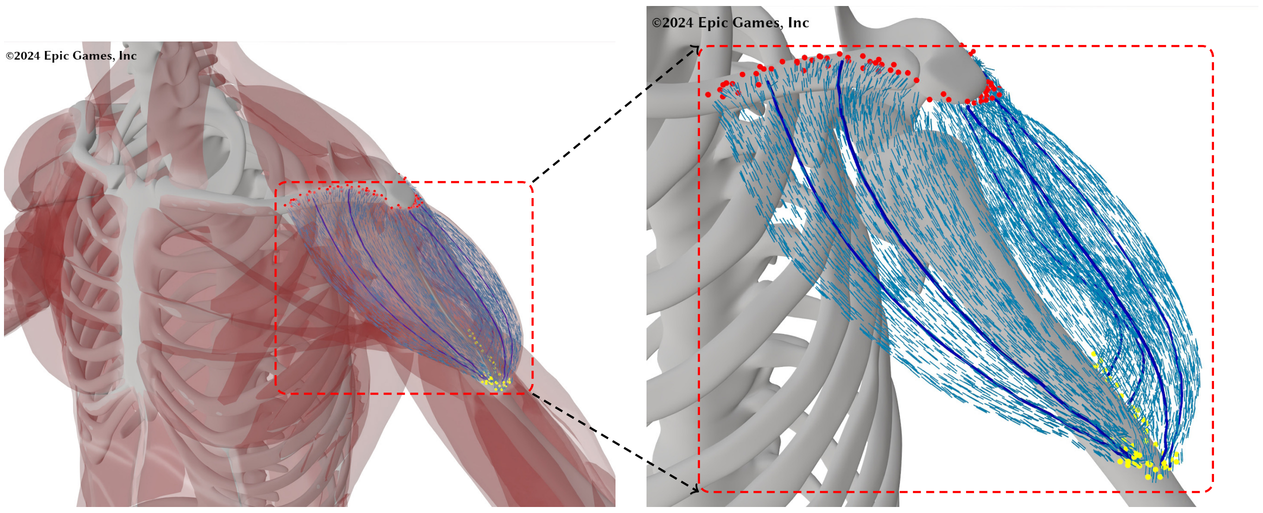


Figure 3.3: **Muscle Fibers and Streamlines**. Muscle fiber directions $\mathbf{D}_t$ are shown in blue. Origin points are shown in red and insertion points are shown in yellow. A few representative streamlines are shown as solid blue curves.

constraints between points ($\mathbf{x}^m_{j_i}$) on the boundary of the musculotendon meshes and their barycentric location (with barycentric weights $w^m_{j_i}$) in the nearest triangle (in a separate muscle) are defined to model fascia like connective tissues and allow us to ignore the effect of the fascia and fat/skin in our layer-by-layer decoupled strategy. Contact constraints are defined analogously, but are dynamically turned on at off at each time step. A contact constraint is only defined if a point is determined to have penetrated a different muscle (determined from a dot product with the surface normal at the closest boundary point). This is a rather simplistic contact model but we found that it strikes the right balance of accuracy and efficiency. In either case, the contact or fascia constraints are of the form

$$\mathbf{r}_j(\mathbf{x}^m, \mathbf{x}^c) = \mathbf{x}^m_{j_0} - w^m_{j_1}\mathbf{x}^m_{j_1} - w^m_{j_2}\mathbf{x}^m_{j_2} - w^m_{j_3}\mathbf{x}^m_{j_3}$$

where $\{w^m_{j_1}, w^m_{j_2}, w^m_{j_3}\}$ are the barycentric coordinates of the closest triangle $\{\mathbf{x}^m_{j_1}, \mathbf{x}^m_{j_2}, \mathbf{x}^m_{j_3}\}$ to $\mathbf{x}^m_{j_0}$ in the boundary of the musculotendon mesh. We visualize the two cases in a practical example in Figure 3.1 (a3). In our examples, the anisotropic model is used for contact constraints and the isotropic model is used for fascia constraints. We provide parameters used to generate our simulation examples in Section 3.6.3.

3.3.1.3 Connective Tissue Membrane

Although our bone, muscle, fascia, and fat/skin layer-by-layer decoupling strategy works well most of the time, we found that near the scapula extra care was needed. In this region, the scapula motion caused excessive distance between the latismus dorsi and the trapezius muscles. This is due to inaccuracy in the scapula joint motion as well as a lack of data for some of the muscles under the scapula. We found that explicitly coupling in a connective membrane between these two muscles was a simple fix for the issue. We added a scapula membrane triangle mesh with additional vertices $\mathbf{x}^c$ coupled to the original muscle vertices $\mathbf{x}^m$ through the energy in Equation (3.1). The membrane is put under tension using the quadratic potential $\sum_{\hat{t}} \frac{\hat{k}}{2} |\mathbf{F}^{\text{cod}}_{\hat{t}}(\mathbf{x}^c)|^2_F A_{\hat{t}}$. This potential gives rise to a linear term in the energy gradient and a constant, positive definite term in the Hessian. Here $\mathbf{F}^{\text{cod}}_{\hat{t}}(\mathbf{x}^c) \in \mathbb{R}^{3\times 2}$ is the surface deformation gradient defined as

$$\mathbf{F}^{\text{cod}}_{\hat{t}} = \left[\begin{array}{c|c} \mathbf{v}^c_{\hat{t}_1} & \mathbf{v}^c_{\hat{t}_2} \end{array} \right] \left[\begin{array}{c|c} |\mathbf{V}^c_{\hat{t}_1}| & 0 \\ \hline \frac{\mathbf{V}^c_{\hat{t}_1}}{|\mathbf{V}^c_{\hat{t}_1}|} \cdot \mathbf{V}^c_{\hat{t}_2} & |\frac{\mathbf{V}^c_{\hat{t}_1}}{|\mathbf{V}^c_{\hat{t}_1}|} \times \mathbf{V}^c_{\hat{t}_2}| \end{array} \right]^{-1}$$

where $\mathbf{v}^c_{\hat{t}_i} = \mathbf{x}^c_{\hat{t}_i} - \mathbf{x}^c_{\hat{t}_0} \in \mathbb{R}^3, \mathbf{V}^c_{\hat{t}_i} = \mathbf{X}^c_{\hat{t}_i} - \mathbf{X}^c_{\hat{t}_0} \in \mathbb{R}^3$ are in the deformed state and A-pose respectively. The potential has a global minimum when all mesh triangles are collapsed to a single point. However, we balance this by adding constraints between points on the boundary of the connective membrane to their closest points in the boundary of the muscle tetrahedron meshes as

$$\mathbf{r}_j(\mathbf{x}^m, \mathbf{x}^c) = \mathbf{x}^c_{j_0} - w^m_{j_1}\mathbf{x}^m_{j_1} - w^m_{j_2}\mathbf{x}^m_{j_2} - w^m_{j_3}\mathbf{x}^m_{j_3}$$

where $\{w^m_{j_1}, w^m_{j_2}, w^m_{j_3}\}$ are the barycentric coordinates of the closest triangle $\{\mathbf{x}^m_{j_1}, \mathbf{x}^m_{j_2}, \mathbf{x}^m_{j_3}\}$ to $\mathbf{x}^c_{j_0}$ in the boundary of the musculotendon mesh. We illustrate this connective tissue in yellow in Figure 3.1 (a1).

3.3.2 Fascia Layer

Given the equilibrium configuration of the musculotendon geometry $\hat{\mathbf{x}}^m(\mathbf{\Theta})$ for a given joint state $\mathbf{\Theta}$, we solve for the fascia layer $\hat{\mathbf{x}}^f(\hat{\mathbf{x}}^m(\mathbf{\Theta}))$. The fascia represents connective tissues that tightly wrap around the muscles. We model this as a triangle mesh and constrain its vertices barycentrically to their closest points in the boundary triangles of the musculotendon tetrahedron meshes if they are within a threshold distance in the A-pose. This accounts for 90% of vertices in the fascia mesh (see Figure 3.1 (a2)); we simulate the remaining 10% by putting the membrane under tension with the same model as the connective tissue membrane in Section 3.3.1.3. We also collide these against the boundary of the musculotendon meshes. This serves as the inner layer of the fat/skin mesh.

3.3.3 Fat and Skin Layer

In the last layer of our approach, we solve for the equilibrium configuration $\hat{\mathbf{x}}^s(\hat{\mathbf{x}}^f(\hat{\mathbf{x}}^m(\mathbf{\Theta})))$ of a fat/skin tetrahedron mesh. We model the layer of fat between the fascia and outer skin as a volumetric elastic solid with a tetrahedron mesh. We create this mesh so that the inner and outer triangle mesh boundaries have the same topology. That is, the outer skin and the inner fascia have the same triangle mesh topology. We solve for the equilibrium of this layer by minimizing the potential

$$\mathrm{PE}^m(\mathbf{x}^s; \mathbf{x}^e(\mathbf{x}^m(\mathbf{\Theta}))) = \sum_t \Psi^{\mathrm{cor}}(\mathbf{F}_t(\mathbf{x}^s))V_t$$

with respect to non-fascia vertices of the fat layer mesh. Here $\mathbf{F}_t(\mathbf{x}^s)$ refers to the deformation gradient in the t^{th} fat mesh tetrahedron. Furthermore, Ψ^{cor} is the again isotropic potential from Equation (2.7). The inner fascia vertices are fixed based on the joint state through $\mathbf{x}^e(\mathbf{x}^m(\mathbf{\Theta}))$. We use the A-pose geometry of this tetrahedron mesh to define the undeformed configuration for deformation gradient computations. We note that the thin volumetric fat mesh with matching inner and outer topology allows us to easily adjust body fat percentage by scaling the outer skin vertices towards their inner fascia counterparts as shown in Figure 3.7.

3.4 Neural Network

Our passive ($\mathbf{PNN}(\mathbf{\Theta}; \mathbf{X}) \in \mathbb{R}^3$) and active ($\mathbf{ANN}(\mathbf{a}; \mathbf{X}) \in \mathbb{R}^3$) neural network models learn per-vertex displacements which are added to A-pose coordinates $\mathbf{X}$. We adopt the network structure from [JZG20] and utilize a PCA final layer so that the network output is simply tens of PCA coefficients. The vertex displacements can be then recovered with a precomputed PCA basis. This approach has been proven capable [JZG20, BOD18] of capturing major deformation modes and preserving spatial smoothness. The full network structure is illustrated in Figure 3.4. Our PNN input consists of $N_J = 23$ joint local rotations and the ANN input consists of $N_M = 46$ muscle activations. The PNN dataset includes around 6000 frames of passive simulation data for the muscles, fascia and skin layer over various ranges of motions as shown in Figure 3.5. We apply inverse linear blend skinning to obtain vertex displacements on the A-pose and we use a loss function equal to the L^2 distance on mesh vertices. The musculotendon neural network $\mathbf{PNN}^m$ is trained on volumetric tetrahedron meshes and infers fiber streamline positions barycentrically. The fat/skin network $\mathbf{PNN}^s$ is trained on the boundary triangle mesh of the fat/skin tetrahedron mesh simulation data. The active network counterparts $\mathbf{ANN}^m$ and $\mathbf{ANN}^s$ are also defined over the A-pose. We generate 920 frames of active simulation data of the muscles, fascia and skin. For each of the $N_M = 46$ muscles, we sequentially sample 20 activations one muscle at a time with values ranging from 0 to 3. Scaling of muscle physiological cross sectional area by factors of 2-3 is commonly adopted in biomechanics applications to account for uncertainties [HUS15]. While activation values are typically constrained between 0 and 1, we allowed activations up to 3 as a similar mechanism. The final PCA layer for the PNN networks uses 20 components and the ANN networks use 92 components. We choose to split each dataset into an 80% training dataset and a 20% evaluation dataset. We trained each network with 1000 epochs

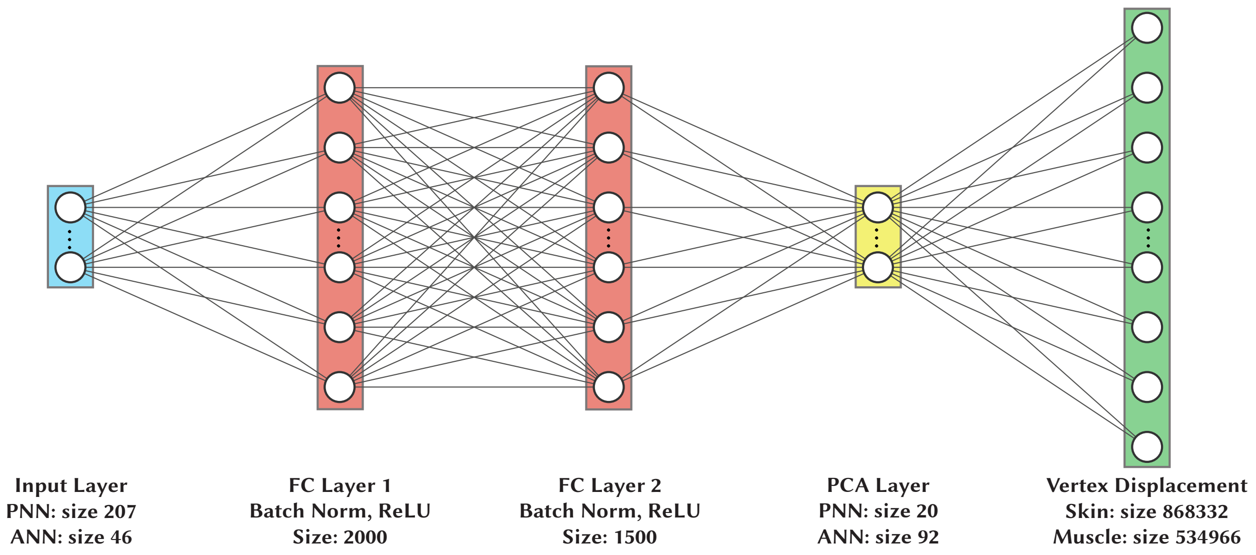


Figure 3.4: **Architecture for PNN and ANN**. We use two fully connected (FC) hidden layers with batch normalization and ReLU activation function.

and stochastic gradient descent. The training and evaluation loss as well as training time are provided in Table 3.1.

3.5 Inverse Activation

We solve for the muscle activations $\mathbf{a} \in \mathbb{R}^{N_M}$ given the rig joint state $\mathbf{\Theta} \in \mathbb{R}^{N_J \times 9}$ using the quasistatic assumption that muscles produce forces that when transmitted to the bones perfectly cancel out the effects of gravity and other external forces and maintain the static pose of the skeleton associated with the joint state $\mathbf{\Theta}$. Muscles span multiple joints in redundant ways that make this perfect balance of forces and torques achievable with non-unique activations. Furthermore, we model the force transmission from multiple muscle fibers per muscle. We denote the vector of all these activations as $\hat{\mathbf{a}} \in \mathbb{R}^{N_F}$. We adopt standard practices from biomechanics and choose a unique activation state $\hat{\mathbf{a}}(\mathbf{\Theta})$ by assuming that the minimal amount of activation is used to maintain the pose [DLH90]. Note that $N_F > N_M$ as discussed in Section 3.3.1.1 and to define the per-muscle activation state $\mathbf{a} \in \mathbb{R}^{N_M}$ we use the average of all of the fibers that a given muscle contains. For clarity (and brevity) of exposition, we define $\mathbf{q} \in \mathbb{R}^{N_C}$ from $\boldsymbol{\Theta}$ as the torque-relevant joint rotation angles. $N_C < N_J$ is the number of joint angles that can articulate in response to muscle and gravitational forces. Each joint has at most 3 degrees of freedom and pin joints such as elbows only have 1. In our example $N_C = (3*3+1)*2 = 20$ as we consider articulations of the sternoclavicular (clavicle), shoulder, scapula and elbow joint on both sides of the upper body.

3.5.1 Torque Equilibrium Derivation

Here we derive the torque equilibrium equations associated with joint articulations. Intuitively, in order to maintain quasistatic poses, the total torque contribution from all forces applied to all points that are articulated by the joint rotation should be zero. We denote the spatial domain of each bone as $\Omega_b \subset \mathbb{R}^3$ and each individual bone mesh consists of locations $\mathbf{x} \in \Omega_b$ sampled in the bone domains. We denote the spatial domain consisting of all the bones in the skeleton as union $\Omega = \cup_b \Omega_b$. For any $\mathbf{x} \in \Omega_b$, a top-down joint rotation hierarchy $\{j_0, \ldots, j_i, \ldots, j_b\}$ originating from a root bone Ω_0 (the sternum/rib cage in our examples) defining the articulation kinematics on bone Ω_b is known. The articulated position $\boldsymbol{\phi}^b(\mathbf{x}; \mathbf{q}) \in \mathbb{R}^3$ of $\mathbf{x} \in \Omega_b$ is composed of a combination of joint transforms $\mathbf{L}^{j_i}(\cdot; q_{j_i})$ defined on each joint rotation j_i as

$$\begin{aligned}
\boldsymbol{\phi}^b(\mathbf{x}; \mathbf{q}) &= \mathbf{L}^{j_0}(\mathbf{L}^{j_1}(\mathbf{L}^{j_2}(\ldots); q_{j_1}); q_{j_0}) \\
\mathbf{L}^{j_i}(\mathbf{x}; q_{j_i}) &= \mathbf{R}^{j_i}(q_{j_i})(\mathbf{x} - \mathbf{x}^{j_i}) + \mathbf{x}^{j_i} \\
\mathbf{R}^j(q_j) &= \mathbf{U}^j \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(q_j) & -\sin(q_j) \\ 0 & \sin(q_j) & \cos(q_j) \end{pmatrix} \mathbf{U}^{jT} \\
\mathbf{U}^j &= \begin{pmatrix} u_0^j, v_0^j, w_0^j \\ u_1^j, v_1^j, w_1^j \\ u_2^j, v_2^j, w_2^j \end{pmatrix} \in \mathbb{R}^{3\times 3}.
\end{aligned}$$

Here $\mathbf{R}^j$ is a rotation matrix of local joint rotation q_j radians around the associated pivot $\mathbf{x}^j$. $\mathbf{U}^j$ is an orthogonal matrix representing the rotation axis $\mathbf{u}^j \in \mathbb{R}^3$. Note that the vector $\mathbf{q} \in \mathbb{R}^{N_C}$ is made up of the components q_j. The Jacobian of the skeletal kinematics $\mathbf{J}^b(\mathbf{x};\mathbf{q}) \in \mathbb{R}^{3\times N_C}$ with respect to the joint state $\mathbf{q}$ is defined as $\mathbf{J}^b(\mathbf{x};\mathbf{q}) = \frac{\partial \boldsymbol{\phi}^b}{\partial \mathbf{q}}(\mathbf{x};\mathbf{q}) = \mathbf{R}^{j_0}(q_{j_0})\ldots\mathbf{R}^{j_{i-1}}(q_{j_{i-1}})\mathbf{R}^{j_i\prime}(q_{j_i})(\mathbf{L}^{j_{i+1}} - \mathbf{x}^{j_i})$. See Appendix A for more detail.

The principle of virtual work reveals the joint torques required to maintain the pose $\mathbf{q}$ in the presence of external forcing

$$\begin{aligned}\delta W = \int_\Omega \mathbf{f}(\mathbf{x}) \cdot \delta\boldsymbol{\phi}^b(\mathbf{x};\mathbf{q})d\mathbf{x} &= \int_\Omega \mathbf{f}(\mathbf{x}) \cdot (\mathbf{J}^b(\mathbf{x};\mathbf{q})\delta\mathbf{q})d\mathbf{x} \\ &= \delta\mathbf{q}^T \int_\Omega \mathbf{J}^{b^T}(\mathbf{x};\mathbf{q})\mathbf{f}(\mathbf{x})d\mathbf{x} = 0.\end{aligned}$$

Here $\delta\mathbf{q}$ is an arbitrary perturbation in the joint state. Intuitively, this states that the residual of the external and muscle forcing $\mathbf{f}(\mathbf{x}) = \rho(\mathbf{x})\mathbf{g} + \mathbf{f}^{\mathrm{m}}(\mathbf{x})$ can only be non-zero in components orthogonal to the articulation. This yields the torque constraints

$$\int_\Omega \sum_{\alpha=0}^{d-1} J^b_{\alpha j}(\mathbf{x};\mathbf{q}) f_\alpha(\mathbf{x}) d\mathbf{x} = \int_{\hat{\Omega}_j} ((\mathbf{x} - \mathbf{x}^j) \times \mathbf{f}(\mathbf{x})) \cdot \mathbf{u}^j d\mathbf{x} = 0 \tag{3.2}$$

where $J^b_{\alpha j}$ and f_α are the components of the Jacobian and force respectively. Also, $\hat{\Omega}_j \subset \Omega$ is defined to be all bodies affected by articulation of joint j. See Appendix A for more detail.

3.5.2 Active Muscle Force Model

Each fiber streamline in each muscle originates on a bone and inserts on another bone (see Figure 3.1 (c1-c4)). We model these curves as lines of action under tension that transmit force to the bones they attach to based on their degree of active contraction, in addition to passive tension arising from extension. We use a standard force/activation model [DLH90]. The force transmitted to origin/insertion by a fiber streamline j is defined as

$$\mathbf{f}^j(l_j;\hat{a}_j) = \sigma_j^{\max}(f_p(l_j) + \hat{a}_j f_a(l_j))\mathbf{n}_j \tag{3.3}$$

where l_j is the normalized fiber streamline length, i.e., the ratio of the current fiber streamline length to its length in the A-pose., $\sigma_j^{\max}$ is the peak isometric streamline tension and $\mathbf{n}_j$ is tangent to the streamline curve at the origin/insertion. Note that the muscle deformation map created from the $\mathbf{PNN}^m$ correction to LBS defines the normalized streamline length l_j of each fiber streamline as it deforms under the current joint state $\mathbf{q}$. Also, note that we adopt the same peak isometric muscle tension from [SHU18] for corresponding muscles but divided by the number of fiber streamlines used in each of the respective muscles. Both passive and active components of the fiber tension have dependence on the normalized streamline length as

$$f_p(l) = \begin{cases} e^{l-1} - 1 & \text{if } l > 1, \\ 0 & \text{otherwise,} \end{cases} \qquad f_a(l) = \begin{cases} l(2-l) & \text{if } 0 \le l \le 2, \\ 0 & \text{otherwise.} \end{cases}$$

Note that we widen the active force-length curve beyond the biomechanics standards [DLH90] to account for extreme fiber compression and stretch observed in practical character animation. Compared with a typical Hill-type muscle model, we introduce simplifications that minimally affect results for slower motions, such as the ones in our study. For example, our model does not include a force-velocity relationship, the tendon is inextensible, and the force-length curves are relatively simple compared to other parameterizations [HUS15].

3.5.3 Optimization Problem

In order to find a unique activation state $\hat{\mathbf{a}} \in \mathbb{R}^{N_F}$ we adopt the standard biomechanical regularizer [DLH90] and minimize the squared L^2 norm of the activation vector subject to the constraint that muscle forces maintain the static pose (Equation (A.1)). The inverse activation problem can be formulated as minimizing the quadratic total energy spent subject to linear torque equilibrium equations. Specifically,

$$\hat{\mathbf{a}} = \arg\min_{\mathbf{a}} \frac{1}{2}\mathbf{a}^T \boldsymbol{\sigma}^{\max}\mathbf{a} + \frac{1}{2}(\mathbf{a} - \hat{\mathbf{a}}_{\text{lb}})^T \mathbf{M}^p (\mathbf{a} - \hat{\mathbf{a}}_{\text{lb}})$$

subject to

$$\Big(\sum_{\text{streamline } j_i} \tilde{\mathbf{x}}^{SL}_{j_i} \times \mathbf{f}^{j_i}(\mathbf{a}_{j_i}) + \Big(\sum_{\mathbf{x}_{j_k} \in \hat{\Omega}_j} m_{j_k} \tilde{\mathbf{x}}_{j_k} \Big) \times \mathbf{g} \Big) \cdot \mathbf{u}_j = \mathbf{0}$$

where the Heaviside penalty function $\mathbf{M}^p_{jj}(\mathbf{x}) = \begin{cases} 1e10 & \text{if } \mathbf{x}_j < 0 \\ 0 & \text{otherwise} \end{cases}$ is a diagonal matrix that enforces activations to be above the specified activation lower bound $\hat{\mathbf{a}}_{\text{lb}}$. We take $\hat{\mathbf{a}}_{\text{lb}} = \mathbf{0}$ to enforce the activations to be non-negative. $\tilde{\mathbf{x}}^{SL}_{j_i} = \mathbf{x}^{SL}_{j_i} - \mathbf{x}^j$ is the local position of endpoints $\mathbf{x}^{SL}_{j_i}$ on streamlines associated with joint state j. $\tilde{\mathbf{x}}_{j_k} = \mathbf{x}_{j_k} - \mathbf{x}^j$ is the local position of vertices in $\hat{\Omega}_j$. To include the effect of the weight of soft tissues in the torque calculations, we assign the vertices of each muscle/tendon to a unique bone. These vertices are included in the $\mathbf{x}_{j_k}$ used in each $\hat{\Omega}_j$. Since the fiber force $\mathbf{f}^j(l_j; \hat{a}_j)$ is linear in the activation $\hat{a}_j$ from Equation (3.3), we can represent the Equation (A.1) constraints as $\mathbf{W}\hat{\mathbf{a}} + \mathbf{b} = \mathbf{0}$ where $\mathbf{W} \in \mathbb{R}^{N_C \times N_F}$. We solve this minimization iteratively (3-5 iterations in practice) to allow the barrier potential to preserve non-negative activations. At each iteration, we obtain the following KKT system

$$(\boldsymbol{\sigma}^{\max} + \mathbf{M}^{p,k-1})\hat{\mathbf{a}}^k + \mathbf{W}^T \hat{\boldsymbol{\lambda}}^k = \mathbf{M}^{p,k-1}\hat{\mathbf{a}}_{\text{lb}}$$
$$\mathbf{W}\hat{\mathbf{a}}^k + \mathbf{b} = \mathbf{0}.$$

Here $\boldsymbol{\sigma}^{\max} \in \mathbb{R}^{N_F \times N_F}$ is a diagonal matrix whose entries are equal to the peak isometric stresses of each fiber. Intuitively, this adds extra cost to the activation of stronger fibers, which would increase energy consumption which humans tend to reduce [SOW15]. The Heaviside penalty against negative activation is expressed through the diagonal matrix $\mathbf{M}^{p,k-1} \in \mathbb{R}^{N_F \times N_F}$ whose entries are set to $1e^{10}$ if the corresponding activation is negative in the previous iteration $(k-1)$. Using $\mathbf{D}^{k-1} = \boldsymbol{\sigma}^{\max} + \mathbf{M}^{p,k-1}$, the update for the kth iteration is

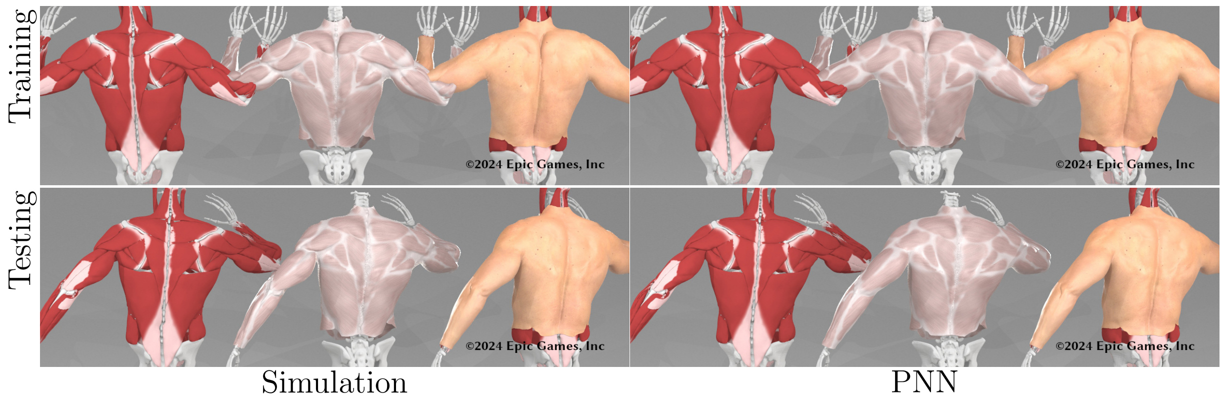


Figure 3.5: **PNN Fitting and Generalization**. **Top Row**: Muscle and fat/skin PNNs fit training data effectively. Simulation (left) is compared to the PNNs (right). **Bottom Row**: Muscle and fat/skin PNNs generalize effectively to unseen testing data. Simulation (left) is compared to PNNs (right).

$$\begin{aligned}\hat{\boldsymbol{\lambda}}^k &= (\mathbf{W}(\mathbf{D}^{k-1})^{-1}\mathbf{W}^T)^{-1}(\mathbf{W}(\mathbf{D}^{k-1})^{-1}\mathbf{M}_n\hat{\mathbf{a}}_{\text{lb}} + \mathbf{b}) \\ \hat{\mathbf{a}}^k &= -\mathbf{D}^{k-1^{-1}}(\mathbf{M}^{p,k-1}\hat{\mathbf{a}}_{\text{lb}} - \mathbf{W}^T\hat{\boldsymbol{\lambda}}^k).\end{aligned} \tag{3.4}$$

Note $\mathbf{D}^{k-1}$ is positive diagonal and $\mathbf{W}(\mathbf{D}^{k-1})^{-1}\mathbf{W}^T$ is symmetric positive definite. Furthermore, $\mathbf{W}(\mathbf{D}^{k-1})^{-1}\mathbf{W}^T$ is of the modest size $N_C \times N_C$. That is, the size depends on the number of articulation degrees of freedom not on the number of fiber streamlines. We use a direct QR decomposition to solve Equation (3.4) since the size is negligible.

3.6 Results

We demonstrate the efficacy of our approach with a number of character animation examples. We note that the anatomy geometries (surface meshes of muscles, bones and skin) were created from MRI scans. These examples emphasize the added realism that activation provides compared to standard LBS. Furthermore, we demonstrate the accuracy of our muscle activation estimations with a comparison to experimental data.

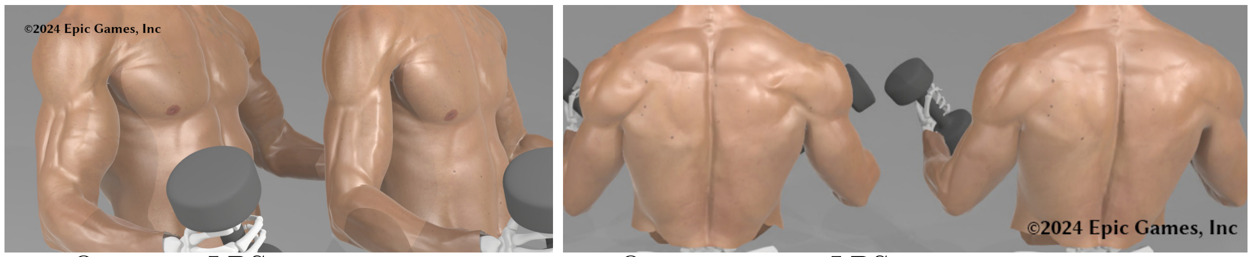


Figure 3.6: **Active neural network deformation**. In each image, the left body illustrates the benefit of our ANN and PNN enhancement of LBS by comparing it to standard LBS in the right body counterpart.

3.6.1 Network Deformation Demonstrations

We first show the per-vertex average MSE loss of PNN and ANN on the training data and evaluation data in Table 3.1. We demonstrate that our muscle and fat/skin PNNs are able to fit the training data and generalize effectively to unseen testing data in Figure 3.5. In the top row, we show that our networks are able to accurately match simulations in the training set. The bottom row shows that our network still matches on poses not in the training set. We further demonstrate this significance by comparing our combined PNN and ANN deformations with standard LBS on a bicep curl animation in Figure 3.6. Note that this motion is not in the training data set. Our model shows more realistic muscle contraction in the bicep, trapezius and deltoid muscles. We also show our ability to control body fat percentage by scaling the outer skin surface vertices towards their counterparts on the inner fascia boundary of the fat/skin mesh in Figure 3.7. Our inverse activation model naturally captures the effects of increasing the amount of weight lifted by the animated character. We demonstrate this effect by increasing the dumbbell weights held in the hands during the biceps curl motion in Figure 3.8. As expected, increased muscle contraction and bulging (e.g. in the biceps and deltoids) are computed to account for the increased weights. Finally, we visualize the fiber streamline activations arising from our inverse activation calculations during a few motions in Figure 3.9. We note that the activations are asymmetric due to

Table 3.1: **Training and Evaluation Loss**. We show that trained PNN and ANN are able to generalize to the evaluation data. The networks were trained using AMD Ryzen PRO 3995WX CPU (128 threads).

Network	Training Loss	Evaluation Loss	Training Time
$\mathbf{PNN}^s$	$1.017e^{-2}$	$9.821e^{-3}$	8 hours
$\mathbf{ANN}^s$	$2.034e^{-4}$	$1.013e^{-4}$	8.5 hours
$\mathbf{PNN}^m$	$6.241e^{-3}$	$6.484e^{-3}$	4.5 hours
$\mathbf{ANN}^m$	$6.668e^{-5}$	$5.821e^{-5}$	7.5 hours

asymmetric anatomical geometries from MRI scans and asymmetric fiber streamlines generated with randomized starting points. Symmetry can be achieved on mirrored geometries and streamlines and symmetric animations.

3.6.2 Comparison with Electromyography Data

To assess the accuracy of our estimations of muscle activation, we compare our estimations with electromyography (EMG) data from a state-of-the-art biomechanics shoulder study [SDM19]. EMG data are direct measurements of the electrical activity of muscles and serve as the best available ground truth for muscle activation patterns. Seth et al. collected the EMG data for shoulder flexion and abduction tasks, and normalized values using a maximum voluntary contraction task. They then used the OpenSim tool called Computed Muscle Control (CMC) to estimate muscle activations given an input motion [SHU18, TA06] and compared their estimated activations to the EMG data to validate their model. For the shoulder abduction and flexion tasks shown in Figure 3.10, our model captures many of the

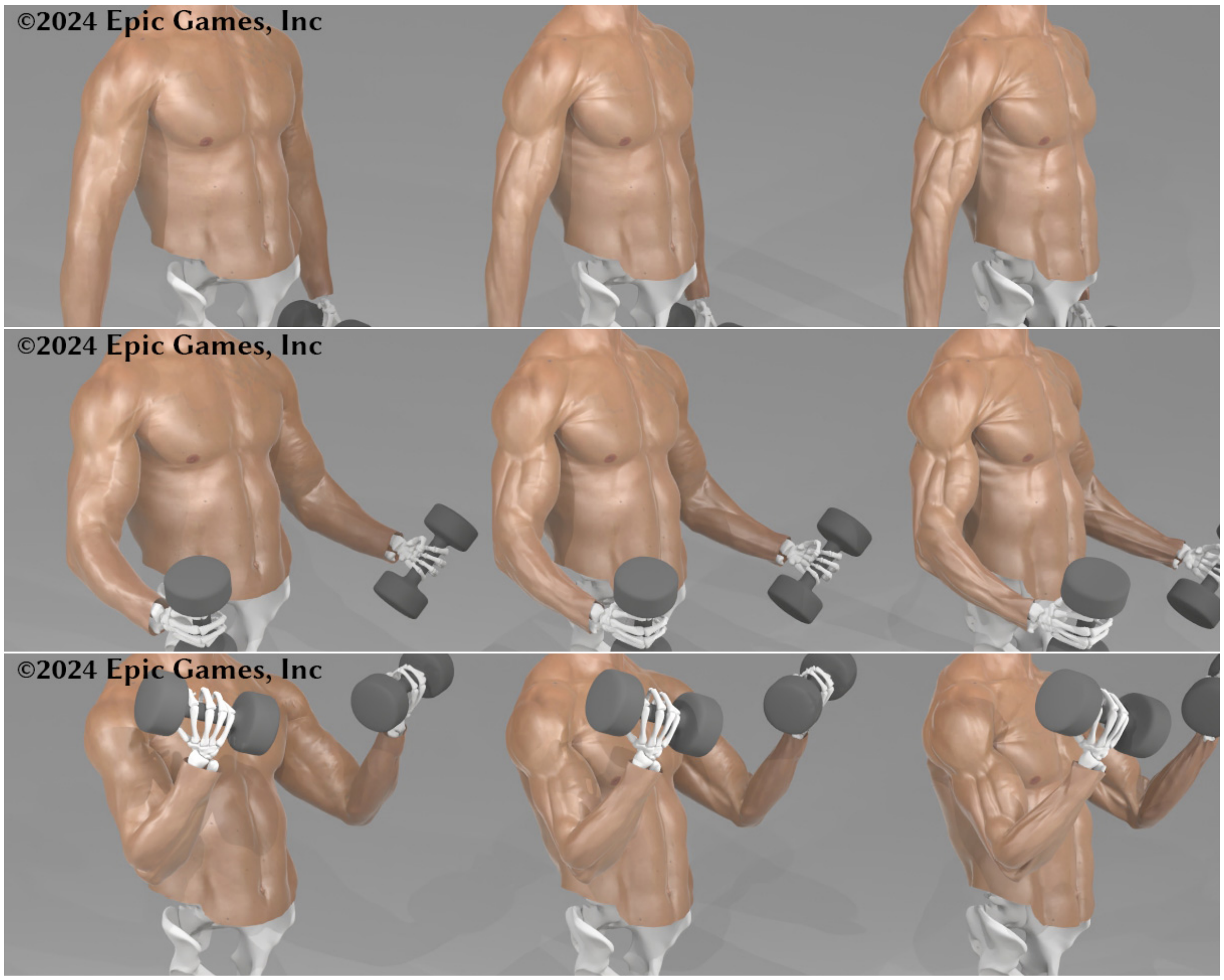


Figure 3.7: **Variation in Body Fat Percentage**. Left: unmodified outer skin surface, middle: halfway between skin and fascia, right: 0% body fat/fascia.

characteristics of the observed EMG patterns, such as the ramp in superior trapezius activity for both movements, late-phase activity of the posterior deltoid in the flexion task, and late phase activity of the bicep in the abduction task. The comparisons to EMG data are similar, and in some cases, improved when comparing to the estimations using CMC with a typical biomechanical model with fewer, piece-wise linear musclulotendon actuators [SDM19] (e.g. our estimations better capture biceps muscle activity in the latter half of the tasks).

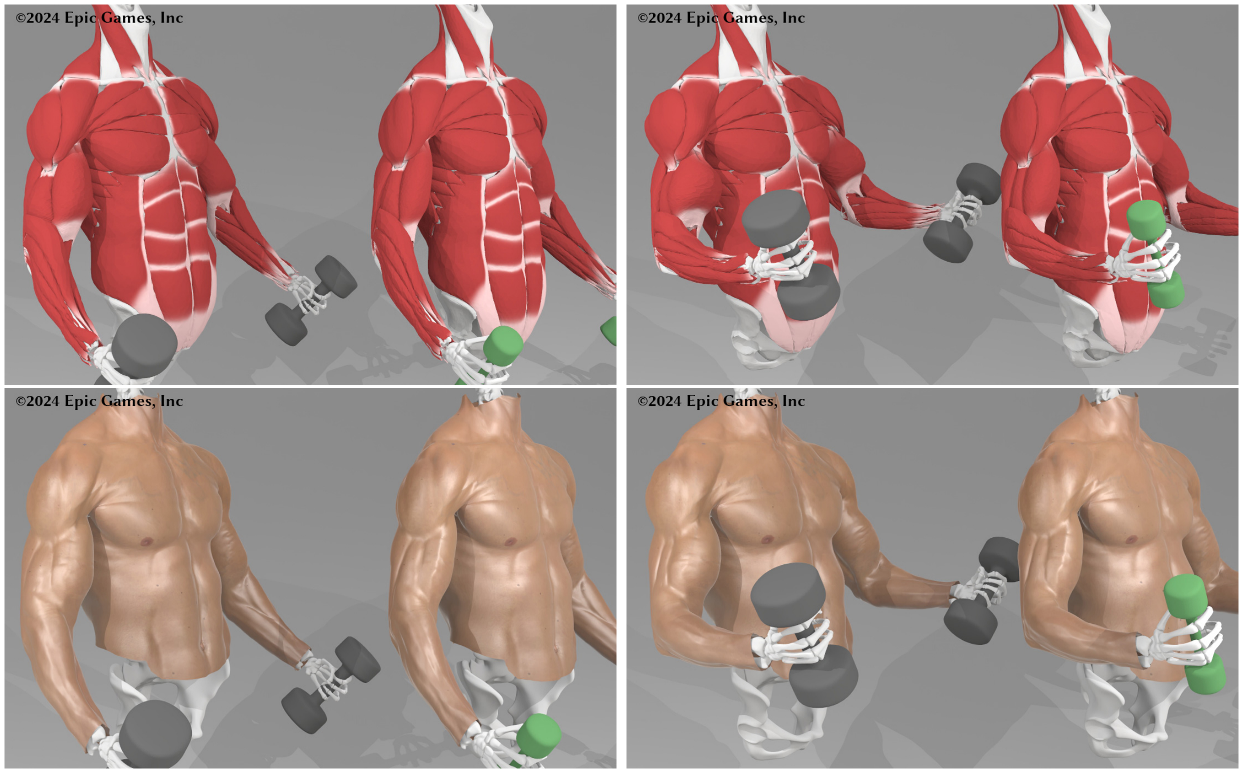


Figure 3.8: **Effect of Increased Weights on Muscles and Skin**. Heavy dumbbell (gray) v.s. light dumbbell (green).

3.6.3 Simulation Parameters and Runtime

We use relatively few physical parameters in our simulations. We set our Lamé parameters μ and λ from Young's moduli of $1e^5$ Pa for muscles and fat, $5e^5$ Pa for tendons and $1e^2$ Pa for membrane as well as Poisson's ratio of 0.3 in all cases. We found that a Poisson ratio closer to 0.5 preserves volume better but burdens the solver when creating training data, and 0.3 allows for satisfactory visual volume preservation in our examples. We use a fascia constraint stiffness equal to $2e^6$ times the weighted average of the mass of vertices in each constraint. Similarly, contact constraint stiffness is set to $1e^8$ times the same weighted average and zero for the tangential directions. We compare the runtime for PNN and ANN enhancement of LBS as well as the inverse activation solve in Table 3.2. Our method is more than one thousand times faster than the simulation and can be run in real time if slightly

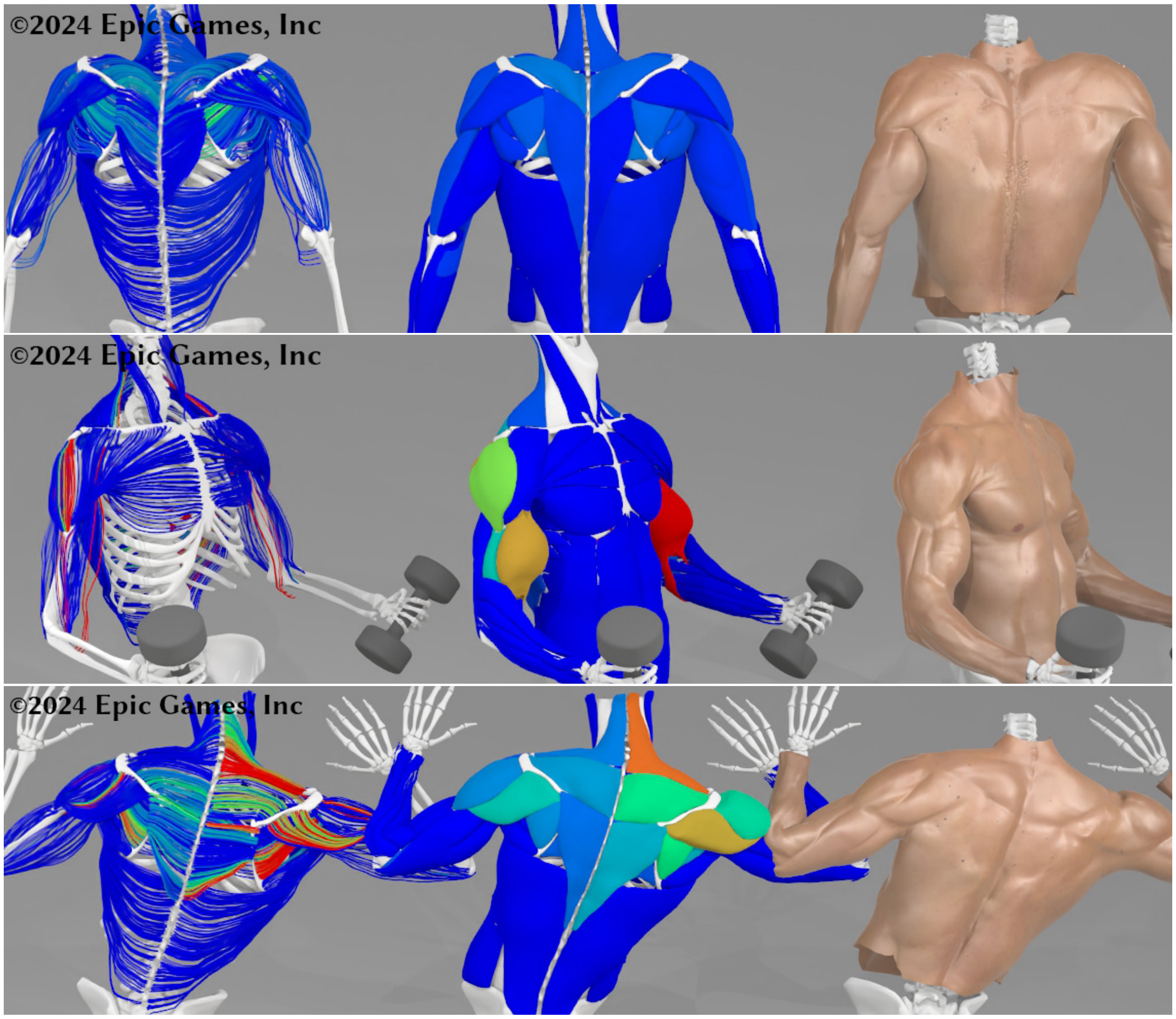


Figure 3.9: **Streamline Activations on Various Poses**. **Left to right**: streamline activations, muscle activations with active network muscle contraction and skin with active network deformation. **Top**: shoulder shrug at time $t = 0.4$s. **Middle**: bicep curl at time $t = 0.53$s. **Bottom**: motion capture at time $t = 48.1$s.

fewer curves and mesh vertices are used.

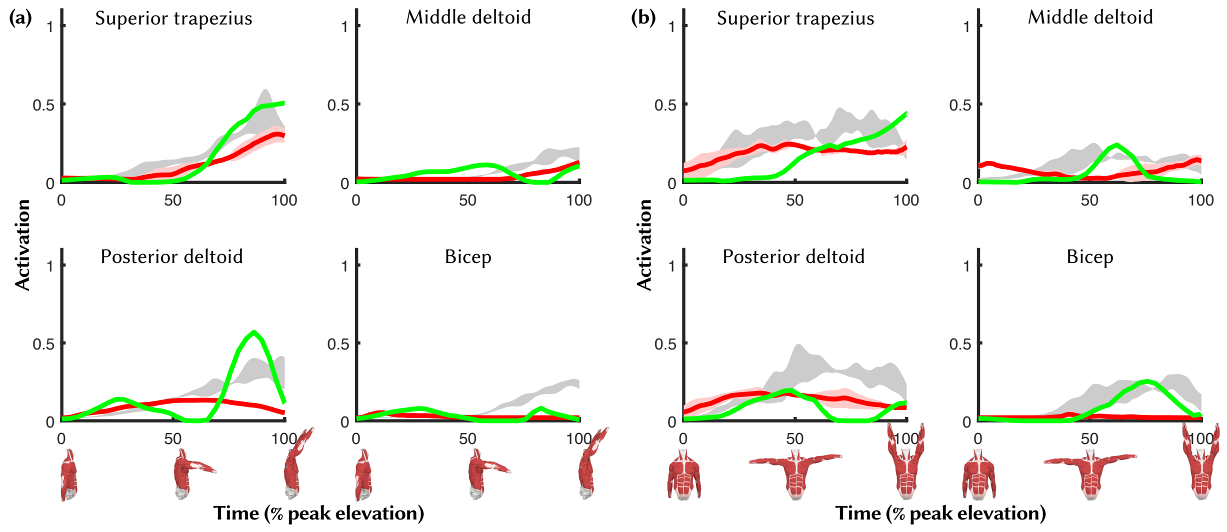


Figure 3.10: **Activation Comparison**. We compare our computed muscle activations with the state-of-the-art approach in Seth et al. [SHU18]. Ground-truth, experimentally observed EMG Data is provided. Our comparisons to EMG data are similar and in some cases improved over Seth et al. [SHU18]. **Green**: Ours. **Red**: Seth et al. [SHU18]. **Gray**: EMG data. **(a)**: Shoulder flexion, **(b)**: Shoulder abduction.

Table 3.2: **Runtime Comparison**. We compare the runtime of our approach against simulation. Times are averaged over the testing EMG animation. Examples were run using AMD Ryzen PRO 3995WX CPU (128 threads).

Task	# Vertices	Runtime (ms/frame)
PNN+ANN,Muscle	182K	36
PNN+ANN,Skin	289K	52
Inverse Activation	2624 curves	180
Simulation, Muscle+Fascia+Skin	182K+145K+145K	283K+25K+39K

3.7 Conclusions and Discussion

While promising, there are still many aspects of our approach that can be improved. We note that our adoption of the A-pose as the reference/undeformed configuration is clearly inaccurate. A better estimate of soft tissue reference states is an interesting direction for future work. In the simulation stage, we allowed the fascia to slide against muscles and bones. Allowing the fat to slide against the fascia will further improve realism. Also, while our rig is designed in a somewhat biomechanically accurate way (e.g. rigidity of bones is preserved), the joints we use could be made more accurate, particularly near the scapula. Lastly, while our comparison with EMG results were promising, some aspects could be improved. For example, in the flexion task, the two deltoid muscles we analyzed ramped up to a burst of activity late in the task but with the posterior deltoid taking up more of the load than observed in the EMG data. However, overall our work advances the state-of-the-art for the animation of human characters with realistic soft tissue deformation and modest run-times, and with additional validation, could have broad applications in biomechanics research.

CHAPTER 4

Analytically Integratable Zero-restlength Springs for Capturing Dynamic Modes

4.1 Related Work

4.1.1 Stated-based Methods

We first discuss prior works that generate elastic deformation directly from spatial state without considering temporal or configurational history. Many works aim to upsample a low-resolution simulation to higher resolution: [FYK10] trains a regressor to upsample, [KGB11] learns an upsampling operator, and [CYJ18] rasterizes the vertex positions into an image before upsampling it and interpolating new vertex positions. [WHR10, ZBO13, XUC14] use example-based methods to synthesize fine-scale wrinkles from a database. [PLP20] predicts a low-frequency mesh with a fully connected network and uses a mixture model to add wrinkles. [CMM20] upsamples with graph convolutional neural networks. [WJG21] recovers high-frequency geometric details with perturbations of texture. [LCT18] uses a generative adversarial network (GAN) to upsample a cloth normal map for improved rendering. [BOD18, BOD20] use neural networks to drive fine scale details from a coarse character rig. Many works aim to learn equilibrium configurations from boundary conditions: [LSW20] uses a neural network to add non-linearity to a linear elasticity model. [MMC20] learns the non-linear mapping from contact forces to displacements. Such approaches are particularly common in virtual surgery applications, e.g. [LHE20, DDS11, SG21, RKS18, PRW19]. [JZG20] trains a CNN to infer a displacement map which adds wrinkles to skinned cloth, and [WGZ20] improves the accuracy of this approach by embedding the cloth into a volumetric tetrahedral mesh. [BME21] adds physics to the loss function, a common approach in physics-inspired neural networks (PINNs), see e.g. [RPK19]. To avoid the soft constraints of PINNs that only coerce physically-inspired behaviour, [GJF20, SWR21] add quasistatic simulation as the final layer of a neural network in order to constrain output to physically attainable manifolds.

4.1.2 Transition-based Methods

Here we discuss prior works that use a temporal history of states, typically for resolving dynamic/inertia related behaviors. In one of the earliest works (before the deep learning era) [GTH98] uses a neural network to learn temporal transitions and leverage back propagation to optimize control parameters. [AST10] incorporates an approximation to the quasistatic equilibrium that serves as a control for a dynamics layer. [GRH12] predicts a cloth mesh from body poses and previous frames, solving a linear system to fix penetrations. [HTC14] uses dynamic subspace simulation on an adaptive selected basis generated from the current body pose. [HDD19] computes a linear subspace of configurations with principal component analysis (PCA) and learns subspace simulations from previous frames with a fully connected network. [FMD19, TPG20, TGL18] obtain nonlinear subspaces with autoencoder networks. Similar methods are commonly used to animate fluids using regression forests [LJS15] or recurrent neural networks (RNNs) [WBT19]. [PFS20] and [SGP20] use graph networks to learn simulations with both fixed and changing topology. [CMM20] proposes a transition-based model with position and linear/angular velocity of the body as network input (in addition to a state-based model). [MPM20] uses a fully connected network to predict node-wise acceleration for total Lagrangian explicit dynamics. [DMH20] proposes a convolutional long short-term memory (LSTM) layer to capture elastic force propagation. [ZWC21] uses an image based approach to enhance detail in low resolution dynamic simulations.

4.1.3 Secondary Dynamics for Characters

Numerical methods that resolve the dynamic effects of inertia-driven deformation have a long history in computer graphics skin and flesh animation. We refer interested readers to only a few papers and a plethora of references therein (e.g. [WZB20, ZBL20, SLP21, XB16, CGC02]). We note that any of these techniques could be used to generate training data for learning-based methods. Secondary dynamics for characters have also been added using data-driven methods: [PRM15] provides a motion capture dataset with dynamic surface meshes, and proposes a linear auto-regressive model to capture dynamic displacements compressed by PCA. [MPM20] extends this method to the SMPL human model. See also [GJF20, SGO20, SZC21]. [KPP17] proposes a two layer approach which skins a volumetric body model as an inner layer and simulates a tetrahedral mesh as an outer layer. The constitutive parameters of the outer layer are learned from 4D scan data. [ZZC21] trains a network to approximate per-vertex displacements from temporal one-ring state using backward Euler simulation data of primitive shapes. [DMH20] also uses a one-ring based approach and trains with forward Euler.

4.1.4 Proportional-derivative Control

Our analytic zero-restlength spring targeting method resembles proportional-derivative (PD) control algorithms used in both computer graphics and robotics. We refer interested readers to several papers leveraging PD control and control parameter optimization for various usages [ANF11, WGF07, WHD12, HWB95, DSZ05].

4.2 Passive Neural Network

We use the (freely available) MetaHuman [Epi21] which has 122 joints and 13575 vertices as our human model. Given joint angles $\boldsymbol{\theta}$, we use a skinning function $\mathbf{x}^{skin}(\boldsymbol{\theta})$ to get the skinned position for each surface vertex. Any reasonable skinning approach (e.g. linear blend skinning [MLT89, LCF00] and dual quaternion skinning [KCv07]) may be used.

Starting from $\boldsymbol{\theta}$ and $\mathbf{x}^{skin}(\boldsymbol{\theta})$, we aim to train a neural network that predicts a more realistic surface mesh $\mathbf{x}^{net}(\boldsymbol{\theta})$. Generally speaking, we could add our analytically intergratable zero-restlength springs directly on top of the skinning result (and there are many interesting skinning-related methods being proposed recently, e.g. [WCC21]), although our proposed dynamics layer (likely) works best when the shape of the surface skin mesh is approximated as accurately as possible. We obtain ground truth for $\mathbf{x}^{net}(\boldsymbol{\theta})$ via quasistatic simulation as discussed in Section 4.2.1.

4.2.1 Quasistatic Simulation

First, we use Tetgen [Si15] (alternatively, [HZG18],[She98] could be used) to create a volumetric tetrahedron mesh whose boundary corresponds to the Metahuman surface mesh in a reference A-pose. Next, we interpolate skinning weights from the Metahuman surface vertices to the tetrahedron mesh boundary vertices, and subsequently solve a Poisson equation on the tetrahedron mesh to propagate the skinning weights to interior vertices [CBE15]. Then, we use a geometric approximation to a skeleton in order to specify which interior vertices of the tetrahedron mesh should follow their skinned positions with either Dirichlet boundary conditions or zero-restlength spring penalty forces.

Our training dataset includes about 5000 poses genetared randomly, from motion capture data, and manually specified animations. Given any target pose, specified by a set of joint angles $\boldsymbol{\theta}$, we solve for the equilibrium configuration of the volumetric tetrahedron mesh using the method from [TSI05] in order to avoid issues with indefiniteness and the method from [MCH22] to enforce contact boundary conditions on the surface of the tetrahedron mesh. Although simulation can be time-consuming, quasistatic simulation is much faster than dynamic simulation. Furthermore, the amount of simulation required is significantly smaller than that which would be needed to obtain similar efficacy for a network aiming to

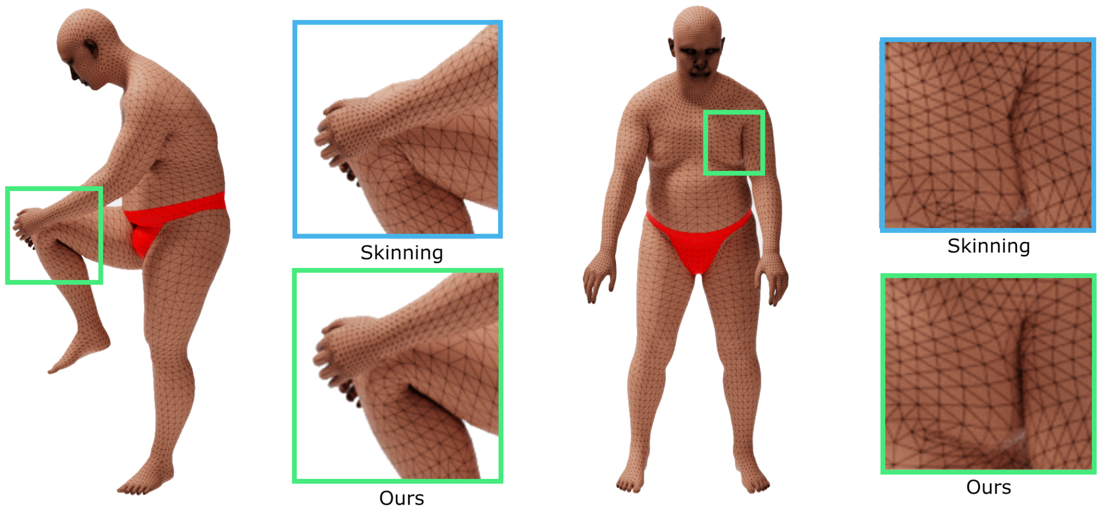


Figure 4.1: Our PNN resolves well-known skinning collision artifacts. We demonstrate this in extreme poses involving the back of the knee and the armpit.

capture temporal information, since such a network would require far more parameters to prevent underfitting.

4.2.2 PNN

Instead of inferring the positions of the surface vertices directly, we augment the skinning result $\mathbf{x}^{skin}(\boldsymbol{\theta})$ with per-vertex displacements $\mathbf{d}(\boldsymbol{\theta})$ so that the non-linearities from joint rotations $\boldsymbol{\theta}$ are mostly captured by the skinning. This reduces the demands on the neural network allowing for a smaller model and thus requiring less training data. Given ground truth displacements $\mathbf{d}(\boldsymbol{\theta})$, we train our passive neural network (PNN) to minimize the loss between $\mathbf{d}(\boldsymbol{\theta})$ and the network inferred result $\mathbf{PNN}(\boldsymbol{\theta})$ (See relationship with Section 3.2). We follow an approach similar to [JZG20] rasterizing the per-vertex displacements into a displacement map image so that a convolutional neural network (CNN) can be used. Of course, one could alternatively use PCA with a fully connected network; however, GPUs are more amenable to the image-based frameworks used by CNNs (see e.g. [Wan21], which discusses the benefit of using data structures that resemble images on GPUs). Our PNN can fix skinning artifacts like interpenetration and volume loss (see Figure 4.1), thus providing a simpler dynamics layer for analytic zero-restlength springs to capture (see Section 4.6 for discussions). Since the PNN is not our main contribution, we refer readers to the original paper [JZG20] for technical details (network architectures, optimizers, hyperparameters, etc.). The PNN used in this chapter can also be easily replaced with other state-based models.

4.3 Kinematics

The skeletal animation will be queried at a user-specified time scale (likely proportional to the frame rate). While these samples are inherently discrete, our approach utilizes the analytic solution of temporal ODEs; therefore, we extend these discrete samples to the continuous time domain. Specifically, given a sequence of skeletal joint angles $\{\boldsymbol{\theta}^1, \boldsymbol{\theta}^2, \ldots\}$, we construct a target function of surface vertex positions $\hat{\mathbf{x}}(t)$ defined for all $t \geq 0$. Options include e.g. Heaviside (discontinous), piecewise linear (C^0), or cubic (C^1) interpolation. We utilize cubic interpolation given its relative simplicity and favorable continuity. Between sample n at time t^n and sample $n+1$ at time $t^n + \Delta t$, we define

$$\hat{\mathbf{x}}(t^n + s\Delta t) = \hat{\mathbf{q}}^n (s\Delta t)^3 + \hat{\mathbf{a}}^n (s\Delta t)^2 + \hat{\mathbf{b}}^n s\Delta t + \hat{\mathbf{c}}^n \tag{4.1}$$

$$= \mathbf{q}^n s^3 + \mathbf{a}^n s^2 + \mathbf{b}^n s + \mathbf{c}^n, \tag{4.2}$$

where $s \in [0,1]$ and Equation 4.2 absorbs the powers of Δt into the non-hatted variables for simplicity of exposition. Enforcing C^1 continuity at times t^n and t^{n+1} requires the following position and derivative constraints

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 3 & 2 & 1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{q}^n \\ \mathbf{a}^n \\ \mathbf{b}^n \\ \mathbf{c}^n \end{bmatrix} = \begin{bmatrix} \mathbf{x}^{net}(\boldsymbol{\theta}^n) \\ \frac{1}{2}(\mathbf{x}^{net}(\boldsymbol{\theta}^{n+1}) - \mathbf{x}^{net}(\boldsymbol{\theta}^{n-1})) \\ \mathbf{x}^{net}(\boldsymbol{\theta}^{n+1}) \\ \frac{1}{2}(\mathbf{x}^{net}(\boldsymbol{\theta}^{n+2}) - \mathbf{x}^{net}(\boldsymbol{\theta}^{n})) \end{bmatrix}, \tag{4.3}$$

which can readily be solved to determine $\mathbf{q}^n, \mathbf{a}^n, \mathbf{b}^n, \mathbf{c}^n$. Here, $\mathbf{x}^{net}(\boldsymbol{\theta}^n)$ $= \mathbf{x}^{skin}(\boldsymbol{\theta}^n) + \mathbf{PNN}(\boldsymbol{\theta}^n)$ are PNN-inferred surface vertex positions at time t^n. Note, in the first interval, $\frac{1}{2}(\mathbf{x}^{net}(\boldsymbol{\theta}^{n+1}) - \mathbf{x}^{net}(\boldsymbol{\theta}^{n-1}))$ is replaced by the one-sided difference $\mathbf{x}^{net}(\boldsymbol{\theta}^{n+1}) - \mathbf{x}^{net}(\boldsymbol{\theta}^n)$.

4.4 Dynamics

We connect a particle (with mass m) to each kinematic vertex $\hat{\mathbf{x}}(t^n + s\Delta t)$ using a zero-restlength spring (although other analytically integratable dynamic models could be used). The position of each simulated particle obeys Hooke's law,

$$\ddot{\mathbf{x}}(t) = k_s(\hat{\mathbf{x}}(t) - \mathbf{x}(t)) + k_d(\dot{\hat{\mathbf{x}}}(t) - \dot{\mathbf{x}}(t)), \tag{4.4}$$

where k_s and k_d are the spring stiffness and damping (both divided by the mass m) respectively. This equation can be analytically integrated (separately for each particle) to determine a closed form solution, which varies per interval because $\mathbf{q}^n, \mathbf{a}^n, \mathbf{b}^n, \mathbf{c}^n$ vary. Consider one interval $[t^n, t^{n+1}]$ with initial conditions

$$\mathbf{x}^n = \mathbf{x}(t^n) \tag{4.5}$$

$$\dot{\mathbf{x}}^n = \dot{\mathbf{x}}(t^n) \tag{4.6}$$

determined from the previous interval; then, the closed form solution in this interval can be written as

$$\mathbf{x}(t^n + s\Delta t) = e^{-\frac{k_d}{2}\Delta t s}\mathbf{g}(t^n + s\Delta t) + \mathbf{p}(t^n + s\Delta t) \tag{4.7}$$

where $s \in [0, 1]$. Here $\mathbf{p}(t^n + s\Delta t)$ is the particular solution associated with the inhomogeneous terms arising from the targets $\hat{\mathbf{x}}(t^n + s\Delta t)$

$$\mathbf{p}(t^n + s\Delta t) = \hat{\mathbf{x}}(t^n + s\Delta t) - \frac{6\mathbf{q}^n s + 2\mathbf{a}^n}{k_s \Delta t^2} + \frac{6k_d\mathbf{q}^n}{k_s^2 \Delta t^3} \tag{4.8}$$

The spring is overdamped when $k_d^2 - 4k_s > 0$, underdamped when $k_d^2 - 4k_s < 0$, and critically damped when $k_d^2 - 4k_s = 0$. Defining a (unitless) ϵ for both the overdamped case, $\epsilon = \frac{\Delta ts}{2}\sqrt{k_d^2 - 4k_s}$, and the underdamped case, $\epsilon = \frac{\Delta ts}{2}\sqrt{4k_s - k_d^2}$, allows us to write

$$\mathbf{g_o}(t^n + s\Delta t) = \boldsymbol{\gamma}_\mathbf{1}^n \frac{e^\epsilon + e^{-\epsilon}}{2} + \boldsymbol{\gamma}_\mathbf{2}^n \Delta ts \frac{e^\epsilon - e^{-\epsilon}}{2\epsilon} \tag{4.9}$$

$$\mathbf{g_u}(t^n + s\Delta t) = \boldsymbol{\gamma}_\mathbf{1}^n \cos\epsilon + \boldsymbol{\gamma}_\mathbf{2}^n \Delta ts \frac{\sin\epsilon}{\epsilon} \tag{4.10}$$

$$\mathbf{g_c}(t^n + s\Delta t) = \boldsymbol{\gamma}_\mathbf{1}^n + \boldsymbol{\gamma}_\mathbf{2}^n \Delta ts \tag{4.11}$$

where $\mathbf{g_o}$ is the overdamped case, $\mathbf{g_u}$ is the underdamped case, and $\mathbf{g_c}$ is the critically damped case. As $\epsilon \to 0$, we obtain $\frac{e^\epsilon + e^{-\epsilon}}{2} \to 1$, $\frac{e^\epsilon - e^{-\epsilon}}{2\epsilon} \to 1$, $\cos\epsilon \to 1$, $\frac{\sin\epsilon}{\epsilon} \to 1$; thus, $\mathbf{g_o} \to \mathbf{g_c}$ and $\mathbf{g_u} \to \mathbf{g_c}$. In all cases,

$$\boldsymbol{\gamma}_\mathbf{1}^n = \mathbf{x}^n - \mathbf{p}(t^n) \tag{4.12}$$

$$\boldsymbol{\gamma}_\mathbf{2}^n = \dot{\mathbf{x}}^n + \frac{k_d}{2}\boldsymbol{\gamma}_\mathbf{1}^n - \dot{\mathbf{p}}(t^n). \tag{4.13}$$

4.5 Learning the constitutive parameters

Given one or more temporal sequences $\left\{\boldsymbol{\theta}^1, \boldsymbol{\theta}^2, \ldots, \boldsymbol{\theta}^N\right\}$ and corresponding dynamic simulation or motion capture results $\{\mathbf{x}_D^1, \mathbf{x}_D^2, \ldots$
$, \mathbf{x}_D^N\}$, we automatically learn constitutive parameters k_s and k_d for each spring. For each such temporal sequence, we create a loss function of the form

$$\mathcal{L} = \sum_{n=1}^{N} \|\mathbf{x}(t^n) - \mathbf{x}_D^n\|_2^2 \tag{4.14}$$

where $\mathbf{x}(t^n)$ is determined as described in Section 4.4. When there is more than one temporal sequence, the loss function can simply be added together. Notably, the loss can be minimized separately for each particle in a highly parallel and efficient manner. We use gradient descent, where initial guesses are obtained from a few iterations of a genetic algorithm [Hol92].

The gradient of $\mathcal{L}$ with respect to the parameters k_d and k_s requires the gradient of $\mathbf{x}(t^n)$ with respect to k_d and k_s, i.e. $\frac{\partial \mathbf{x}}{\partial k_d}$ and $\frac{\partial \mathbf{x}}{\partial k_s}$. From Equation 4.7, one can readily see that the chain rule takes the form

$$\frac{\partial \mathbf{x}}{\partial k_s} = e^{-\frac{k_d}{2}\Delta t s}\frac{\partial \mathbf{g}}{\partial k_s} + \frac{\partial \mathbf{p}}{\partial k_s} \tag{4.15}$$

$$\frac{\partial \mathbf{x}}{\partial k_d} = e^{-\frac{k_d}{2}\Delta t s}\frac{\partial \mathbf{g}}{\partial k_d} - \frac{\Delta t s}{2}e^{-\frac{k_d}{2}\Delta t s}\mathbf{g} + \frac{\partial \mathbf{p}}{\partial k_d} \tag{4.16}$$

where $\frac{\partial \mathbf{g}}{\partial k_s}$, $\frac{\partial \mathbf{g}}{\partial k_d}$, and $\mathbf{g}$ all vary based on ϵ, i.e. based on whether k_s and k_d admit overdamping, underdamping, or critically damping. As we have seen (see Equation 4.9, 4.10, 4.11 and the discussion thereafter), $\mathbf{g}$ is continuous in the 2-dimensional k_s-k_d phase space; however, one needs to carefully implement $\frac{\sin\epsilon}{\epsilon}$ and $\frac{e^{\epsilon}-e^{-\epsilon}}{2\epsilon}$ to replace potentially spurious floating point divisions by the asymptotic result when ϵ is small. One can similarly show that $\frac{\partial \mathbf{g}}{\partial k_s}$ and $\frac{\partial \mathbf{g}}{\partial k_d}$ are continuous, and thus $\frac{\partial \mathbf{x}}{\partial k_s}$ and $\frac{\partial \mathbf{x}}{\partial k_d}$ are continuous.

To see that $\frac{\partial \mathbf{g}}{\partial k_s}$ and $\frac{\partial \mathbf{g}}{\partial k_d}$ are continuous, we expand them via the chain rule

$$\frac{\partial \mathbf{g}}{\partial k_s} = \frac{\partial \mathbf{g}}{\partial \boldsymbol{\gamma}_1^n}\frac{\partial \boldsymbol{\gamma}_1^n}{\partial k_s} + \frac{\partial \mathbf{g}}{\partial \boldsymbol{\gamma}_2^n}\frac{\partial \boldsymbol{\gamma}_2^n}{\partial k_s} + \left(\frac{1}{\epsilon}\frac{\partial \mathbf{g}}{\partial \epsilon}\right)\left(\epsilon\frac{\partial \epsilon}{\partial k_s}\right) \tag{4.17}$$

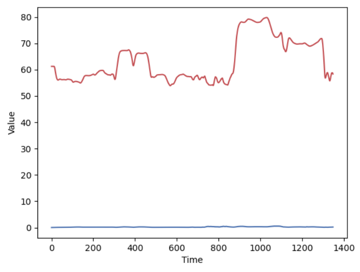
$$\frac{\partial \mathbf{g}}{\partial k_d} = \frac{\partial \mathbf{g}}{\partial \boldsymbol{\gamma}_1^n}\frac{\partial \boldsymbol{\gamma}_1^n}{\partial k_d} + \frac{\partial \mathbf{g}}{\partial \boldsymbol{\gamma}_2^n}\frac{\partial \boldsymbol{\gamma}_2^n}{\partial k_d} + \left(\frac{1}{\epsilon}\frac{\partial \mathbf{g}}{\partial \epsilon}\right)\left(\epsilon\frac{\partial \epsilon}{\partial k_d}\right) \tag{4.18}$$

and note that $\frac{\partial \mathbf{g}}{\partial \boldsymbol{\gamma}_1^n}$ and $\frac{\partial \mathbf{g}}{\partial \boldsymbol{\gamma}_2^n}$ are continuous for the same reasons that $\mathbf{g}$ is. As can be seen in Equations 4.12 and 4.13, $\frac{\partial \boldsymbol{\gamma}_1^n}{\partial k_s}$, $\frac{\partial \boldsymbol{\gamma}_1^n}{\partial k_d}$, $\frac{\partial \boldsymbol{\gamma}_2^n}{\partial k_s}$, and $\frac{\partial \boldsymbol{\gamma}_2^n}{\partial k_d}$ recursively depend on the prior interval via $\mathbf{x}^n$ and $\dot{\mathbf{x}}^n$ (and eventually the initial conditions) but add no new discontinuities of their own. We inserted $\frac{1}{\epsilon}$ and ϵ into the last term in both Equations 4.17 and 4.18 so that $\epsilon\frac{\partial \epsilon}{\partial k_s} = \mp\frac{1}{2}\Delta t^2 s^2$ and $\epsilon\frac{\partial \epsilon}{\partial k_d} = \pm\frac{1}{4}\Delta t^2 s^2 k_d$ are robust to compute (the $\mp$ and $\pm$ signs represent overdamping/underdamping respectively). Then, we write

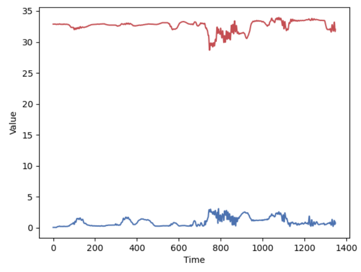
$$\frac{1}{\epsilon}\frac{\partial \mathbf{g_o}}{\partial \epsilon} = \boldsymbol{\gamma}_1^n\frac{e^{\epsilon}-e^{-\epsilon}}{2\epsilon} + \boldsymbol{\gamma}_2^n\Delta t s\frac{(\epsilon-1)e^{\epsilon}+(\epsilon+1)e^{-\epsilon}}{2\epsilon^3} \tag{4.19}$$

$$\frac{1}{\epsilon}\frac{\partial \mathbf{g_u}}{\partial \epsilon} = -\left(\boldsymbol{\gamma}_1^n\frac{\sin\epsilon}{\epsilon} + \boldsymbol{\gamma}_2^n\Delta t s\frac{\sin\epsilon-\epsilon\cos\epsilon}{\epsilon^3}\right) \tag{4.20}$$

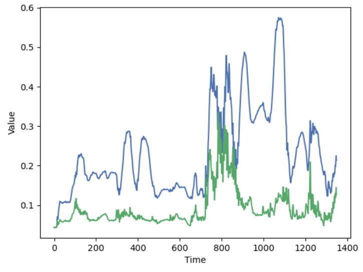
to identify two more functions that must be carefully implemented (as $\epsilon \to 0$, $\frac{(\epsilon-1)e^{\epsilon}+(\epsilon+1)e^{-\epsilon}}{2\epsilon^3} \to \frac{1}{3}$ and $\frac{\sin\epsilon-\epsilon\cos\epsilon}{\epsilon^3} \to \frac{1}{3}$). The sign difference between Equation 4.19 and 4.20 matches that in $\epsilon\frac{\partial \epsilon}{\partial k_s}$ and $\epsilon\frac{\partial \epsilon}{\partial k_d}$ showing that both $\frac{\partial \mathbf{g}}{\partial \epsilon}\frac{\partial \epsilon}{\partial k_s}$ and $\frac{\partial \mathbf{g}}{\partial \epsilon}\frac{\partial \epsilon}{\partial k_d}$ are continuous.



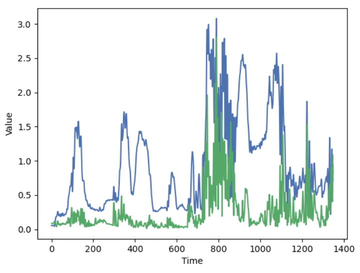
(a) all vertices averaged



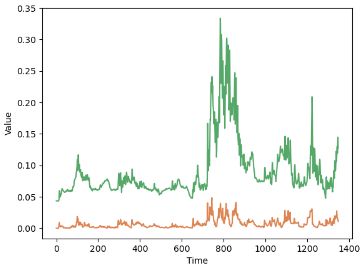
(b) one vertex



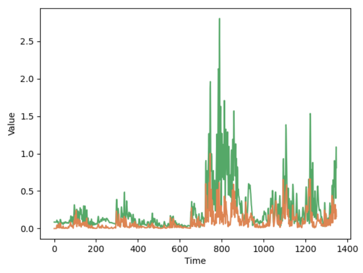
(c) all vertices averged



(d) one vertex



(e) all vertices averged



(f) one vertex

Figure 4.2: Red curve: ℓ_2 norm of vertex positions in the pelvis coordinate system. Blue curve: ℓ_2 norm of displacements from skinning to dynamics. Green curve: ℓ_2 norm of displacements from PNN to dynamics. Orange curve: ℓ_2 norm of displacements from PNN to zero-restlength springs.

Finally, it is worth noting that a 2-dimensional gradient cannot be computed on the codimension-1 curve associated with critically damping; however, taking the dot product of the continuous (between overdamping and underdamping) gradient with the tangent to the codimension-1 curve (and adjusting for either k_s or k_d parameterization) matches the derivative along the curve as expected.

4.6 Results and Discussion

Figure 4.2 quantitatively illustrates how our approach alleviates the demand on the neural network for a particular dynamic simulation example ("calisthenics"). Figure 4.2a shows the

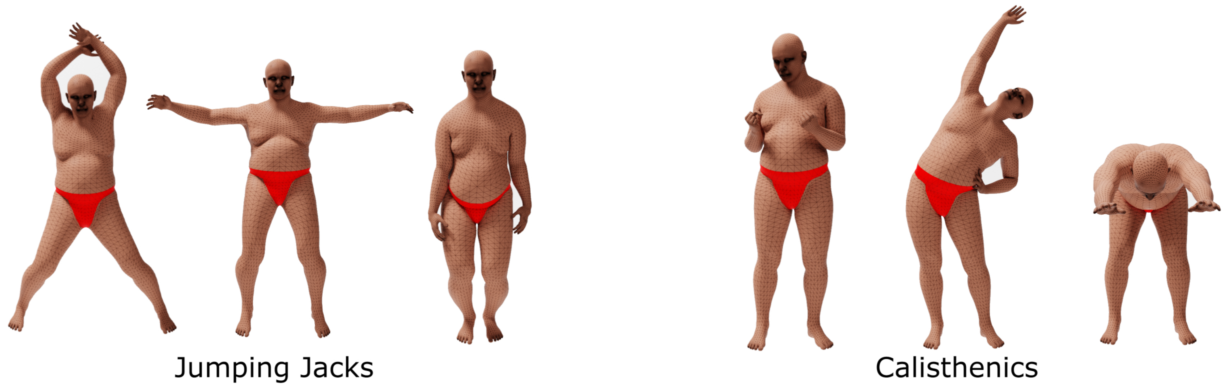


Figure 4.3: Dynamic simulation sequences used to learn zero-restlength spring constitutive parameters.

ℓ_2 norm of the vertex positions (red curve) measured relative to a coordinate system whose origin is placed on the pelvis joint. Figure 4.2b shows the same result for a single vertex on the belly. The ℓ_2 norm of the displacements from the skinned result (blue curve) is vastly smaller (as shown in Figures 4.2a and 4.2b), indicating that most of this function is readily captured via skinning (a number of authors have utilized this approach [PRM15, SGO20, SZC21, JZG20, MPM20]). In Figures 4.2c and 4.2d, we change the scale so that the blue curve can be more readily examined. In addition, we also plot the ℓ_2 norm of the displacements from our PNN result (green curve) as the dynamics layer we want to approximate. This dynamics layer has a relatively small magnitude and low variance (comparably), which is readily approximated/learned based on a few dynamic simulations of training data. Figures 4.2e and 4.2f show the dynamics layer approximated by our zero-restlength springs (orange curve). Our approach captures the approximated shape, but with smaller magnitude, due to regularization. However, even with regularization, our method still outputs quite compelling dynamics.

As mentioned in Section 4.5, we learn our spring constitutive parameters using a (surprisingly) small amount (less than 100 frames) of ground truth simulation data. We obtain the dynamic simulation results $\left\{\mathbf{x}_D^1, \mathbf{x}_D^2, \ldots, \mathbf{x}_D^N\right\}$ via backward Euler simulation. Figure 4.3

shows examples of two dynamic simulation sequences ("jumping jacks" and "calisthenics") we use to learn zero-restlength spring constitutive parameters. Note that any reasonable animation sequence with dynamics can be used, even motion capture data (see e.g. [PRM15]). Although we use a dataset with 5000 data samples in order to train a robust PNN (see Section 4.2), only a few dynamic simulation examples are required in order to learn zero-restlength spring constitutive parameters that generalize well to unseen animations. This also means that we only need to engineer the network architectures and hyperparameters for the configuration-only PNN, which is much easier than engineering a network that captures configuration transitions. Previous methods that add secondary motions to characters [PRM15, GJF20, SGO20, SZC21] usually require a large dataset with thousands of data samples to not overfit their network. In comparison, the minimal need of data from our method is a great ease for the data generation process. Our method is also unconditionally stable thanks to its analytic nature, and its optimized constitutive parameters are physically interpretable.

4.6.1 Examples

Our analytic zero-restlength spring model generalizes very well to unseen animations and does not face severe underfitting or overfitting, which is common in machine learning methods if the network architecture is not carefully designed and trained on a plethora of data. Figure 4.4 qualitatively shows two example frames comparing a skinning-only result with our analytic zero-restlength springs added on top of our PNN. The frame on the left ("jumping jacks") is taken from an animation sequence used in training while the frame on the right ("shadow boxing") is taken from an animation sequence not used in training. In both examples, our method successfully recovers ballistic motion (e.g. in the belly). Our method runs in real-time (30-90 fps, or even faster pending optimizations) and emulates the effects of accurate, but costly, dynamic backward Euler simulation remarkably well (the dynamic backward Euler simulation we use to generate training examples take about 16 minutes per

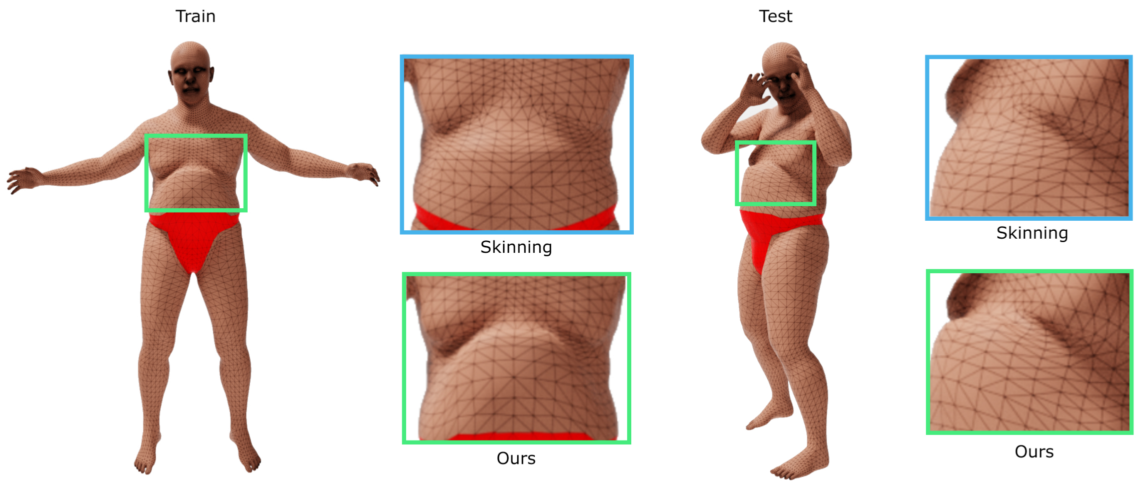


Figure 4.4: Comparison of our trained zero-restlength spring ballistic motion with the corresponding skinned result. Left: a motion sequence included in training. Right: a motion sequence not included in training. The ability to train on "jumping jacks" and generalize to "shadow boxing" would be impossible for a typical neural network approach.

frame with self-collision enabled). Our approach can be easily applied to different mesh topologies. Figure 4.5 shows the secondary dynamics added to a low-poly ankylosaurus. We refer readers to our supplementary video for a compelling demonstration, particularly of the secondary inertial motion.

Full dynamic simulation is costly and prone to instabilities. Often this results in a few simulated frames with visible errors. To avoid such artifacts, we modify our training procedure to avoid overfitting to poorly converged frames (that would lead to poor generalization). See Figure 4.6. We note that similar approaches are common in the computer vision community (see e.g. random sample consensus [FB81]).

Figure 4.7 shows a heatmap visualization of learned k_s, k_d and the overdamping/ underdamping indicator $k_d^2 - 4k_s$, respectively. Note how symmetric our optimization result is, even if we optimize each particle separately. In regions where rigid motion dominates (e.g. hands, feet, head, etc.), the optimization results in overdamped springs with large stiffness.

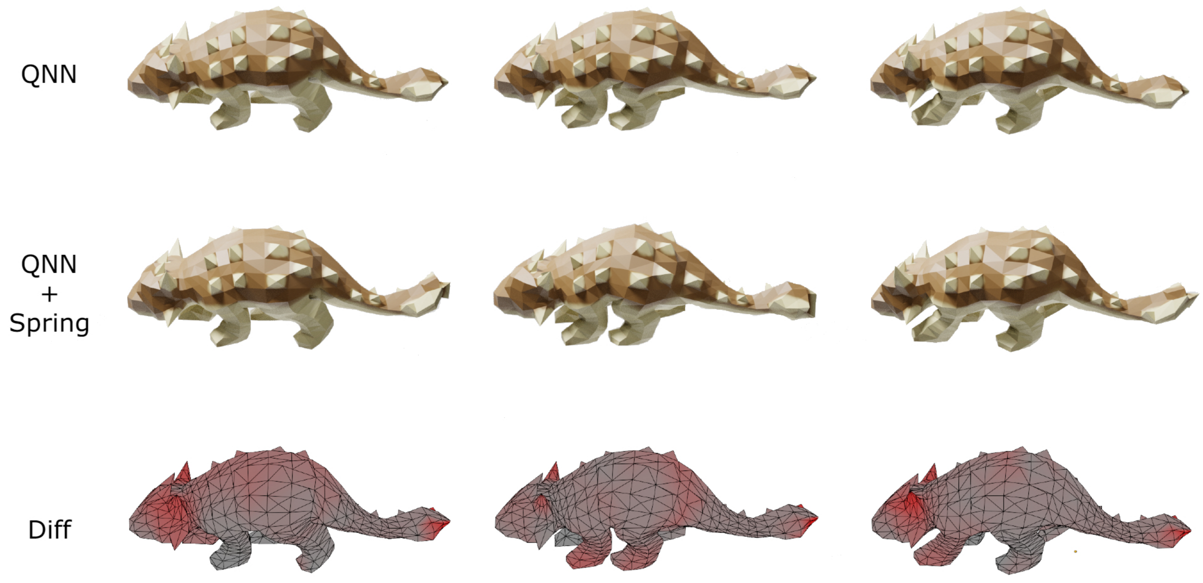
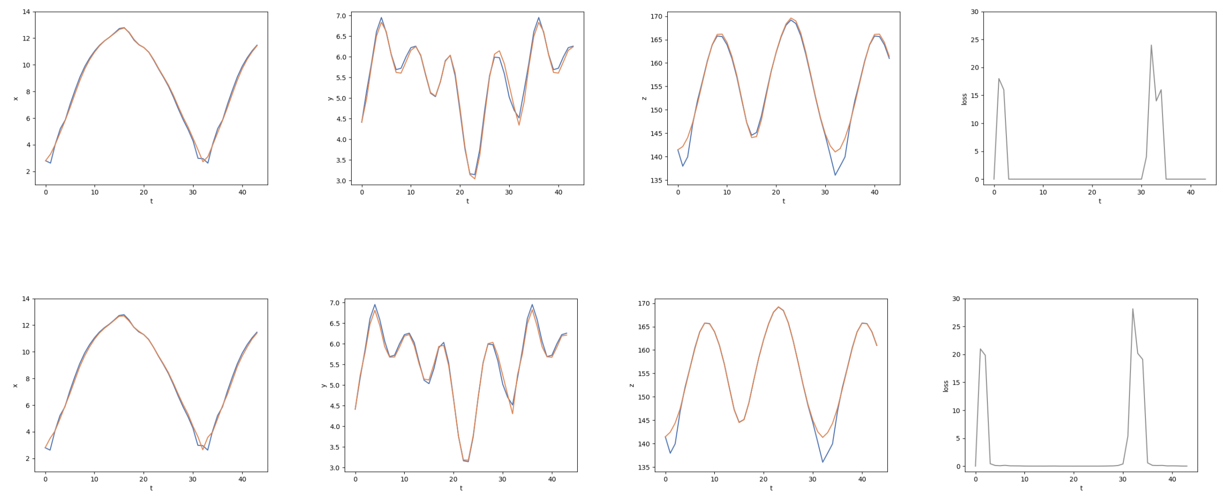


Figure 4.5: Secondary dynamics are added to a low-poly ankylosaurus. Notice how the zero-restlength springs (second row) manage to add dynamic motions on top of quasistatic result (first row), especially around the ears, tail, and back region.

The code can be accelerated by replacing the constitutive parameters of all such springs with a single set of consitutive parameters. In regions where soft-tissue dynamics dominates (e.g. belly, thigh, etc.), the optimization results in underdamped springs with small stiffness. Since our optimization is per particle decoupled, it is easy to troubleshoot (if necessary).

Some artifacts of our methods appear when the PNN is not well trained, resulting in physically incorrect quasistatic meshes during inference (interpenetrations, not preserving volume, etc.). This can be constantly improved by better PNN architecture and more extensive experiments on hyperparameter tuning, and is not the main focus of this chapter. Collision artifacts might also appear in the dynamic step (although not noticeable in our experiments), since our zero-restlength springs method does not handle collisions for efficiency.

As a final note, one could obviously add our zero-restlength springs on top of the skinned



(a) X axis (b) Y axis (c) Z axis (d) Loss

Figure 4.6: Robust training in the presence of simulation errors. Subfigures in columns (a)-(c) are per-axis trajectories of an example vertex in the jumping jack sequence. The backward Euler trajectory is shown in blue and our analytic zero-restlength spring trajectory is shown in orange. The high-frequencies in Frames 31-34 are caused by poorly converged dynamics in the presence of collisions. Subfigures in column (d) show the ℓ_2 loss between the zero-restlength springs and backward Euler. The first row is the initial training result and the second row is the re-trained result with the 10% highest-loss frames ignored. The second row more closely follows the backward Euler trajectory for the frames that don't have simulation errors.

result directly; however, we obtained better results using our PNN to fix skinning artifacts due to volume loss and collision.

4.7 Conclusion and Future Work

We present an analytically integratable physics model that can recover dynamic modes in real-time. The main takeaway is that the problem can be separated into a configuration-only

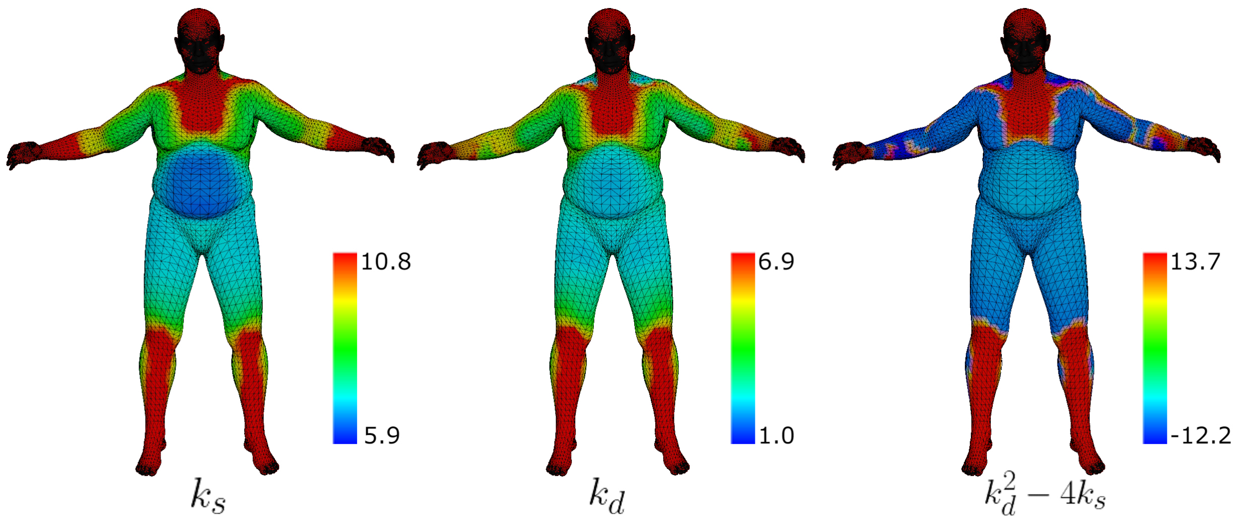


Figure 4.7: Heatmap visualization (logarithm scale) of stiffness k_s, damping k_d, and $k_d^2 - 4k_s$ which determines overdamping/underdamping, respectively. In heavily constrained regions the springs are stiffer and more overdamped, while in fleshy regions the springs are softer and more underdamped. Note that more constrained regions occur based on proximity to the bones used in the dynamic simulation training data (e.g. chest, forearms, shins, etc.).

quasistatic layer and a transition-dependent dynamics layer, where the dynamics layer can be well approximated by a simple physics model. The constitutive parameters of the physics model can be robustly learned from only a few backward Euler simulation examples. In particular, determining k_s and k_d requires a gradient that can erroneously overflow/underflow near the critical damping manifold in k_s-k_d phase space. We quite robustly addressed this by isolating non-dimensionalized functions that were trivially carefully implemented to obtain the correct asymptotic result in *all* cases. For more discussions on both numerical and analytical issues with gradients, we refer the interested readers to [JF22, MFS21].

CHAPTER 5

Volumetric Meshing Algorithm for Self-intersecting Surfaces

5.1 Algorithm Overview

The input to our algorithm is a triangulated surface mesh $\mathcal{S}$. The output is a uniform-grid-based embedding hexahedron mesh counterpart $\mathcal{V}$ to $\mathcal{S}$ that is well-defined (i.e., free from numerical mesh "glueing" artifacts) even when $\mathcal{S}$ is self-intersecting (see Section 5.8 for examples).

We briefly summarize the three main stages of our algorithm, as detailed in Figure 5.1. In the first stage, volumetric extension (Section 5.3), we create a hexahedron mesh $\mathcal{V}^S$ from the background grid that only covers the input surface $\mathcal{S}$ with connectivity designed to mimic it. We sign its vertices depending on inside/outside information derived from the hypothetical self-intersection-free counterpart $\tilde{\mathcal{S}}$. We emphasize that this volumetric extension mesh only surrounds $\mathcal{S}$. Accordingly, the second stage of the algorithm is interior extension region creation (Section 5.4). Nodes of the background grid are partitioned using the edges cut by $\mathcal{S}$, and then we decide which regions are interior. Interior regions will be copied to approximate the number of times portions of the interior of the hypothetical self-intersection-free counterpart $\tilde{\mathcal{S}}^V$ will need to overlap after being pushed forward by the hypothetical mapping $\boldsymbol{\phi}^S_{\tilde{S}}$. For each interior region j^I with at least one copy, we create a hexahedron mesh $\mathcal{V}^{j^I,c}$ for each copy c. In the third stage of the algorithm (Section 5.5), interior extension regions meshes $\mathcal{V}^{j^I,c}$ are sewn together and into the volumetric extension $\mathcal{V}^S$ to produce the

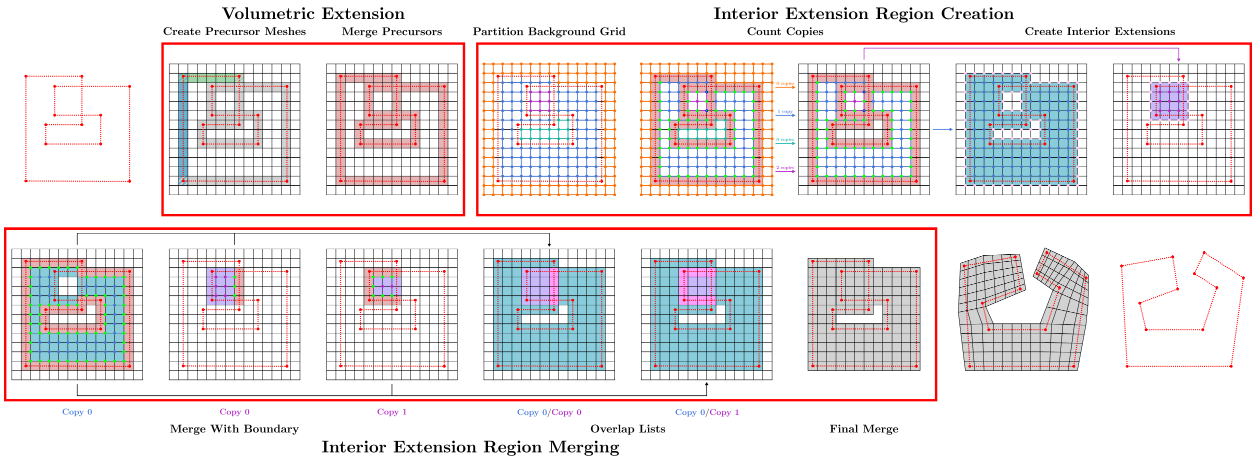


Figure 5.1: **Algorithm overview.** Given an initial input surface mesh $\mathcal{S}$, there are three major steps in the computation of the final volumetric extension mesh $\mathcal{V}$: Volumetric Extension, Interior Extension Region Creation, and Interior Extension Region Merging. *(Volumetric Extension)* In this step, we create a precursor mesh for each element in $\mathcal{S}$, and compute preliminary signing information for the vertices. We then merge the precursor meshes to create the volumetric extension $\mathcal{V}^S$ and correct the signing information where necessary. *(Interior Extension Region Creation)* In preparation for growing the volumetric extension into the interior, we first partition the nodes of the background grid using the edges cut by $\mathcal{S}$. We decide which regions are interior and count the copies of each region using the vertices of $\mathcal{V}^S$ which have negative sign. For each interior region j^I with at least one copy, we then create a hexahedron mesh $\mathcal{V}^{j^I,c}$ for each copy c. *(Interior Extension Region Merging)* The merging process begins with copying relevant hexahedra from $\mathcal{V}^S$ into $\mathcal{V}^{j^I,c}$. First, certain vertices of $\mathcal{V}^{j^I,c}$ are replaced by corresponding vertices from $\mathcal{V}^S$. Hexahedra to be replaced are then removed from $\mathcal{V}^{j^I,c}$ before the boundary hexahedra are copied in. We then merge the various meshes $\mathcal{V}^{j^I,c}$ by first determining where different meshes overlap, and then using these hexahedra overlap lists to perform the final merge.

final output mesh. We additionally provide a coarsening approach in Section 5.6 to provide user control over the embedding mesh resolution as well as a topologically-aware technique

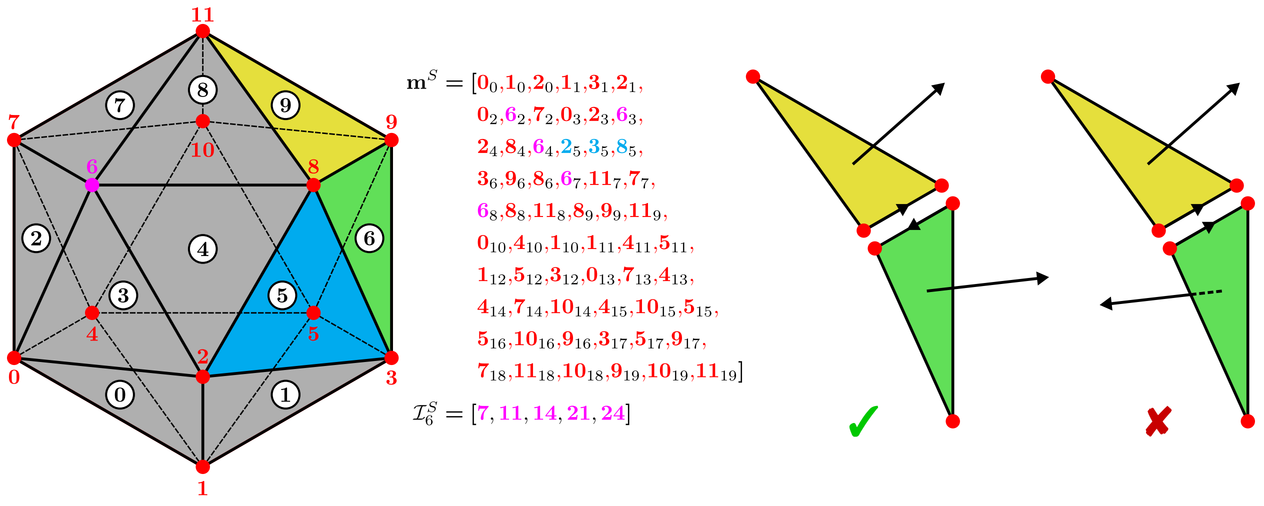
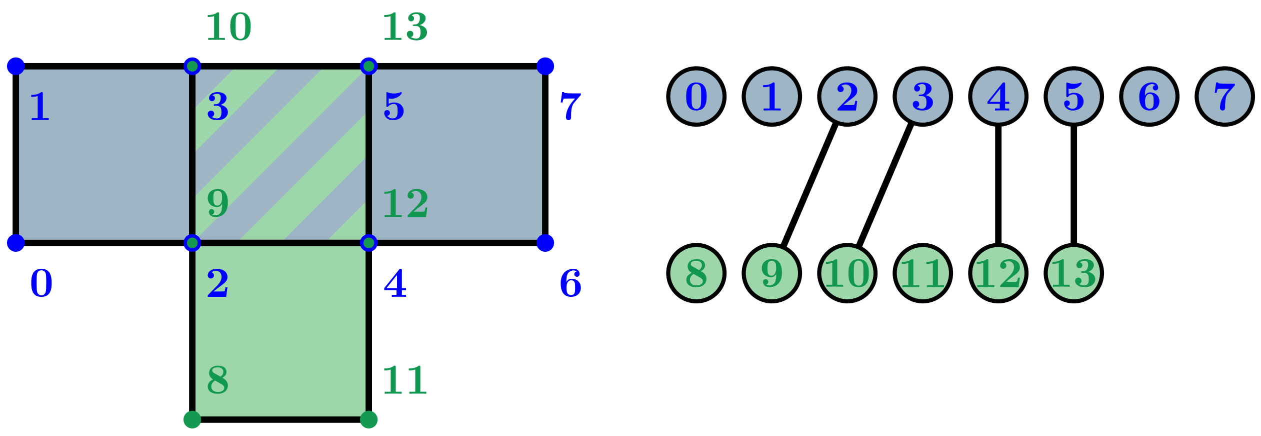

11
7
9
10
6
8
0
2
4
5
3
1
$\mathbf{m}^S = [\mathbf{0}_0,\mathbf{1}_0,\mathbf{2}_0,\mathbf{1}_1,\mathbf{3}_1,\mathbf{2}_1,$
$\mathbf{0}_2,\mathbf{6}_2,\mathbf{7}_2,\mathbf{0}_3,\mathbf{2}_3,\mathbf{6}_3,$
$\mathbf{2}_4,\mathbf{8}_4,\mathbf{6}_4,\mathbf{2}_5,\mathbf{3}_5,\mathbf{8}_5,$
$\mathbf{3}_6,\mathbf{9}_6,\mathbf{8}_6,\mathbf{6}_7,\mathbf{11}_7,\mathbf{7}_7,$
$\mathbf{6}_8,\mathbf{8}_8,\mathbf{11}_8,\mathbf{8}_9,\mathbf{9}_9,\mathbf{11}_9,$
$\mathbf{0}_{10},\mathbf{4}_{10},\mathbf{1}_{10},\mathbf{1}_{11},\mathbf{4}_{11},\mathbf{5}_{11},$
$\mathbf{1}_{12},\mathbf{5}_{12},\mathbf{3}_{12},\mathbf{0}_{13},\mathbf{7}_{13},\mathbf{4}_{13},$
$\mathbf{4}_{14},\mathbf{7}_{14},\mathbf{10}_{14},\mathbf{4}_{15},\mathbf{10}_{15},\mathbf{5}_{15},$
$\mathbf{5}_{16},\mathbf{10}_{16},\mathbf{9}_{16},\mathbf{3}_{17},\mathbf{5}_{17},\mathbf{9}_{17},$
$\mathbf{7}_{18},\mathbf{11}_{18},\mathbf{10}_{18},\mathbf{9}_{19},\mathbf{10}_{19},\mathbf{11}_{19}]$
$\mathcal{I}_6^S = [\mathbf{7}, \mathbf{11}, \mathbf{14}, \mathbf{21}, \mathbf{24}]$
✓
✘

Figure 5.2: **Mesh conventions.** *(Left)* A sample triangle mesh is shown, along with the vector $\mathbf{m}^S$. The incident elements $\mathcal{I}_6^S$ for vertex 6 are also shown. The first 10 faces, visible from the front, have been labeled on the mesh. *(Right)* The left pair of triangles are consistently oriented; the orientations of the edge induced by the normals point in opposite directions. For the right pair, the orientations on the common edge point in the same direction; this is not consistent.

for converting the hexahedron mesh $\mathcal{V}$ into a tetrahedron mesh $\mathcal{T}$.

5.2 Definitions and Notation

We take a triangle mesh $\mathcal{S} = (\mathbf{x}^S, \mathbf{m}^S)$ as input. We use $\mathbf{x}^S = [\mathbf{x}_0^S, \ldots, \mathbf{x}_{N_v^S-1}^S] \in \mathbb{R}^{3N_p^S}$ to denote the vector of triangle vertices $\mathbf{x}_i^S \in \mathbb{R}^3$ and $\mathbf{m}^S \in \mathbb{N}^{3N_e^S}$ to denote the vector of indices m_j^S for vertices in $\mathbf{x}^S$ corresponding triangles $t^S_{\lfloor \frac{j}{3} \rfloor}$, $0 \le \lfloor \frac{j}{3} \rfloor < N_e^S$. For example, for the mesh $\mathcal{S}$ in Figure 5.2, triangle t_5^S is made up of vertices $\mathbf{x}^S_{m_j^S}$ with $j = 2, 3, 8$. We assume that $\mathcal{S}$ is closed (every edge in the mesh has two incident triangles) and consistently oriented (each edge appears with opposite orientations in its two incident triangles). For each vertex $\mathbf{x}_i^S$ of $\mathcal{S}$, we use $\mathcal{I}_i^S$ to denote the set of incident mesh indices j such that $i = m_j^S$. Figure 5.2 demonstrates these conventions. We output a hexahedron mesh $\mathcal{V} = (\mathbf{x}^V, \mathbf{m}^V)$ with $\mathbf{x}^V \in \mathbb{R}^{3N_p^V}$ denoting



$$\mathbf{m}_0 = [0, 2, 3, 1, 2, 4, 5, 3, 4, 6, 7, 5]$$
$$\mathbf{m}_1 = [8, 11, 12, 9, 9, 12, 13, 10]$$
$$\mathbf{m}_2 = [0, 2, 3, 1, 2, 4, 5, 3, 4, 6, 7, 5, 8, 11, 4, 2, 2, 4, 5, 3]$$
$$\mathbf{m}_2 = [0, 2, 3, 1, 2, 4, 5, 3, 4, 6, 7, 5, 8, 11, 4, 2]$$

Figure 5.3: **Mesh merge.** An example of two meshes merging together. Vertices 2, 3, 4 and 5 merge with vertices 9, 10, 12 and 13, respectively. A new vector $\mathbf{m}_2$ is created to hold all of the hexahedron vertices post-merge, and the extra hexahedron (in red) is then removed.

the vector of hexahedron vertices and $\mathbf{m}^V \in \mathbb{N}^{8N_e^V}$ denoting the vector of indices in $\mathbf{x}^V$ corresponding to vertices in hexahedron h_e^V, $0 \leq e < N_e^V$. Each hexahedron in the mesh is geometrically coincident with one grid cell in a background uniform grid $\mathcal{G}_{\Delta x}$. We denote the spacing of this grid as Δx (uniformly in each direction). For ease of visualization, we use 2D counterparts to $\mathcal{S}$ and $\mathcal{V}$ in illustrative figures. In this case, $\mathcal{S}$ is a segment mesh and $\mathcal{V}$ is a quadrilateral mesh.

5.2.1 Merging

We construct the final hexahedron mesh $\mathcal{V}$ by merging portions of various precursor hexahedron meshes in a manner similar to techniques used in [TSB05, WDG19, WJS14, LB18]. As with $\mathcal{V}$, each hexahedron in a precursor mesh is geometrically coincident with background grid cells. All precursor meshes share the same vertex array $\mathbf{x}^V$, although its size will change as we converge to the final $\mathcal{V}$. At various stages of the algorithm, we will merge certain geometrically coincident precursor hexahedra. To perform a merge, we view the set of all vertices in $\mathbf{x}^V$ as nodes in a single undirected graph and introduce graph edges between nodes corresponding to geometrically coincident vertices. In subsequent sections, we refer to such edges in the undirected graph as adjacencies to distinguish them from edges in the various meshes. Once all adjacencies are defined, we compute the connected components of the graph using depth-first search. All vertices in a connected component are considered to be the same and we choose one representative for all mesh entries. We note that this operation may be carried out on more than two meshes at once and that it can lead to duplicate hexahedra and in this case we remove all but one. Furthermore, replacing all vertices in a connected component with one representative results in unused vertices in $\mathbf{x}^V$. We remove all unused vertices in a final pass, changing indexing in $\mathbf{m}^V$ accordingly. We illustrate the connected component calculation, vertex replacement and unused vertex removal in Figure 5.3.

5.3 Volumetric Extension

We first create a volumetric extension $\mathcal{V}^S$ of the surface $\mathcal{S}$. It is a hexahedron mesh that contains the input surface $\mathcal{S}$ and is designed to have topological properties analogous to $\mathcal{S}$. Since it is an extension of $\mathcal{S}$, we can sign the vertices of $\mathcal{V}^S$ depending on which side of the surface they lie on. Overlapping regions in $\mathcal{S}$ complicate this process, but it can be disambiguated by considering the pre-image of the surface to its overlap-free counterpart $\tilde{S}$ under the mapping $\boldsymbol{\phi}^S_{\tilde{S}}$. Signing points in $\mathbb{R}^3$ depending on whether or not they are inside $\tilde{S}$ is well-defined and our procedure for signing the vertices in the volumetric extension $\mathcal{V}^S$ is designed considering its pre-image under $\boldsymbol{\phi}^S_{\tilde{S}}$.

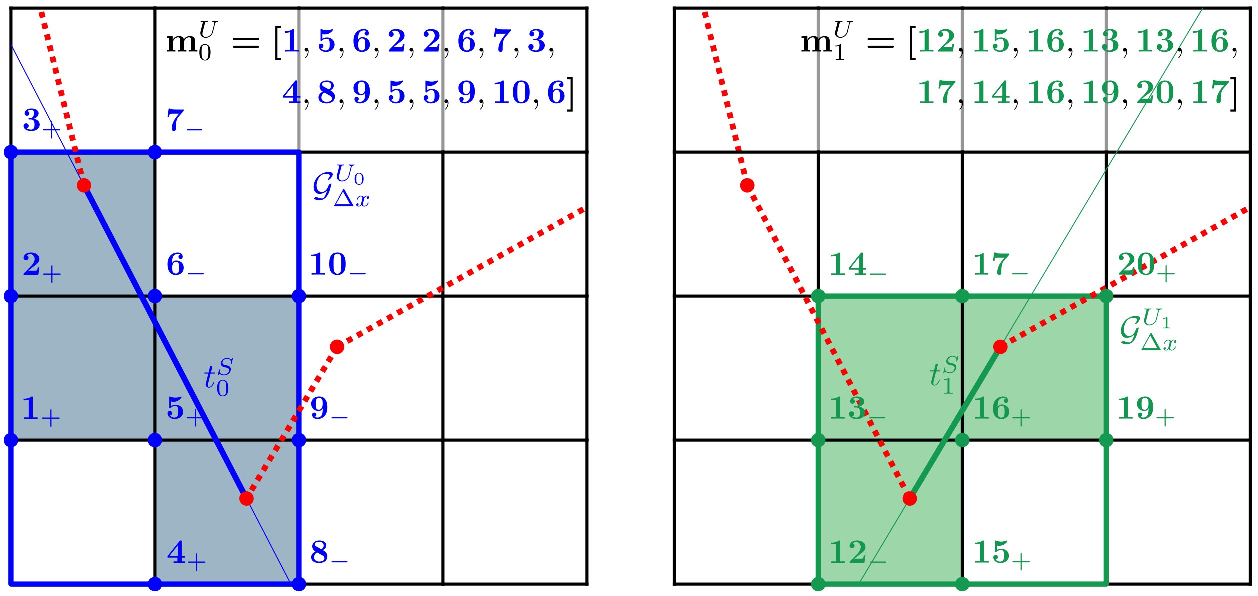

$\mathbf{m}_0^U = [1, 5, 6, 2, 2, 6, 7, 3, 4, 8, 9, 5, 5, 9, 10, 6]$
3_+
7_-
$\mathcal{G}_{\Delta x}^{U_0}$
2_+
6_-
10_-
t_0^S
1_+
5_+
9_-
4_+
8_-
$\mathbf{m}_1^U = [12, 15, 16, 13, 13, 16, 17, 14, 16, 19, 20, 17]$
14_-
17_-
20_+
$\mathcal{G}_{\Delta x}^{U_1}$
t_1^S
13_-
16_+
19_+
12_-
15_+

Figure 5.4: **Precursor meshes.** *(Left)* Surface element t_0^S creates quadrilateral mesh $\mathcal{V}_0^S$. *(Right)* Surface element t_1^S creates quadrilateral mesh $\mathcal{V}_1^S$. Each element creates copies of the grid cells it intersects by introducing new vertices which are geometrically coincident to grid nodes.

5.3.1 Surface Element Precursor Meshes

In order to mimic the topology of the $\mathcal{S}$, we create its volumetric extension $\mathcal{V}^S$ from precursor meshes $\mathcal{V}_e^S = (\mathbf{x}^V, \mathbf{m}_e^{V^S})$ associated with each triangle t_e^S in $\mathcal{S}$. Note that all precursor meshes share the common vertex array $\mathbf{x}^V$ and that this process begins its evolution to the final $\mathcal{V}$ vertex array. For each triangle t_e^S in $\mathcal{S}$, we define a hexahedron mesh from the subgrid $\mathcal{G}_{\Delta x}^{V_e^S}$ of $\mathcal{G}_{\Delta x}$ defined by the grid-cell-aligned bounding box of t_e^S. We add a new hexahedron to $\mathcal{V}_e^S$ corresponding to each background grid cell in $G_{\Delta x}^{V_e^S}$ intersected by t_e^S. We perform this operation using the intersection function from CGAL's 2D/3D Linear Geometry Kernel [The20, BFG20]. The hexahedron is geometrically coincident to the intersected grid cell in $\mathcal{G}_{\Delta x}$, however the vertices introduced into the vertex vector $\mathbf{x}^V$ are copies of the background grid nodes associated with the sub grid $\mathcal{G}_{\Delta x}^{V_e^S}$. Note that even though different triangles may intersect the same grid cells, their respective hexahedra correspond to distinct vertices in

$\mathbf{x}^V$. Further note that mesh elements in $\mathcal{V}_e^S$ inherit the connectivity of the sub grid $\mathcal{G}_{\Delta x}^{V_e^S}$, that is, hexahedra share common vertices if they are neighbors in $\mathcal{G}_{\Delta x}^{V_e^S}$. We sign the vertices in each $\mathcal{V}_e^S$ depending on which side of the plane containing the triangle t_e^S that they lie on. We illustrate this process in Figure 5.4. Lastly, we note that these signs are low-cost preliminary approximations to the signs in the final volumetric extension $\mathcal{V}^S$. In some cases the signs computed in this phase will not be accurate in the volumetric extension, and we provide a more accurate but costly signing when this occurs (discussed in Section 5.3.2; however, in many cases, they are equal to the final signs, and their comparably-low computational cost improves overall algorithm performance

5.3.2 Merge Surface Element Meshes

We merge portions of the precursor meshes $\mathcal{V}_e^S$ to form the volumetric extension hexahedron mesh $\mathcal{V}^S$ by defining adjacency between vertices in $\mathbf{x}^V$ as described in Section 5.2.1. We define this adjacency from the mesh connectivity of $\mathcal{S}$ using its incident elements $\mathcal{I}_i^S$ for each vertex $\mathbf{x}_i^S$. Geometrically coincident vertices in $\mathcal{V}^S_{\lfloor j_{i,0}^S/3 \rfloor}$ and $\mathcal{V}^S_{\lfloor j_{i,1}^S/3 \rfloor}$ for $j_{i,0}^S, j_{i,1}^S \in \mathcal{I}_i^S$ are defined to be adjacent if each are on hexahedrons in their respective meshes which are geometrically coincident. Note in particular that this is different from requiring that geometrically coincident vertices in $\mathcal{V}^S_{\lfloor j_i^S/3 \rfloor}$ for $j_i^S \in \mathcal{I}_i^S$ (see the geometry of Figure 5.12). In other words, all geometrically coincident hexahedra in element precursor meshes associated with triangles that share a common vertex are merged (see Figure 5.5). Merged vertices retain the sign they were given in $\mathcal{V}_e^S$ when possible. However, if merged vertices have differing signs, e.g. in regions with higher curvature (see Figure 5.6), then we must recompute the sign from their geometric relation to $\mathcal{S}$.

In regions of higher curvature where the preliminary signs of vertices in $\mathcal{V}_e^S$ cannot be adopted in $\mathcal{V}^S$, we use an eikonal strategy [OF03] to propagate positive signs from $\mathcal{S}$ in the direction of the surface normal and minus signs in the opposite direction. This is well defined in light of the assumed existence of the pre-image $\tilde{\mathcal{S}}$ of $\mathcal{S}$ under $\boldsymbol{\phi}_{\tilde{S}}^S$. Here, each vertex $\mathbf{x}_i^V$

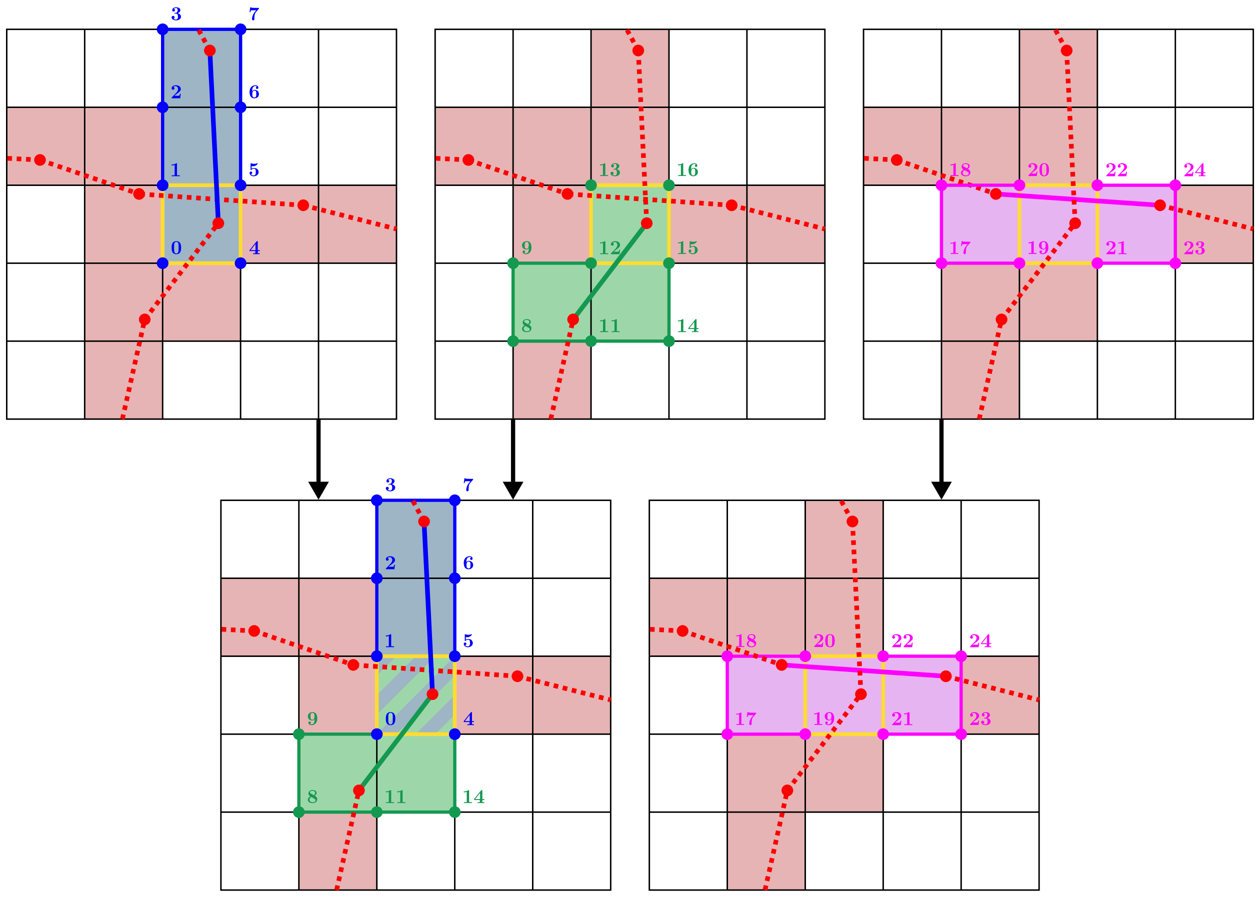


Figure 5.5: **Precursor merge.** The 12 vertices bordering the cell marked in yellow are merged into 8 resulting vertices. Blue vertices 0, 1, 4, 5 and green vertices 12, 13, 15, 16 are merged, respectively. However, magenta vertices 19, 20, 21, 22 do not merge with the blue or green vertices since their associated surface element is topologically distant.

in the volumetric extension $\mathcal{V}^S$ is associated with some collection of precursor meshes $\mathcal{V}^S_{e_i}$ where $\mathbf{x}^V_i$ was created in the merge of vertices in the $\mathcal{V}^S_{e_i}$. This defines a local patch S_{i^V} of surface triangles $t^S_{e_i}$ in $\mathcal{S}$ associated with $\mathbf{x}^V_i$. When propagating signs from $\mathcal{S}$ to $\mathbf{x}^V_i$, only these triangles are considered. It is important to only use this local surface patch since there may be triangles in $\mathcal{S}$ that are geometrically close to $\mathbf{x}^V_i$ but topologically distant. Note that this precludes the use of global point-in-polygon algorithms based on ray casting or winding numbers since those will not give correct results when $\mathcal{S}$ has self-intersection. Instead we

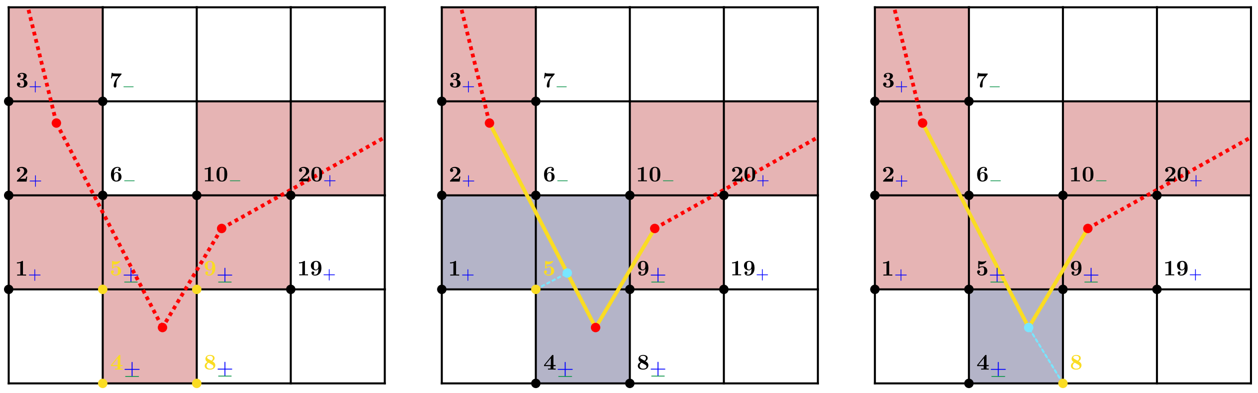


Figure 5.6: **Closest facet.** *(Left)* The four vertices in yellow all have ambiguous signs. *(Middle)* To sign vertex 5, we generate the local patch S_{5^V}, which are the segments shown in yellow. The closest facet (indicated in cyan) lies on a face. *(Right)* A similar process is illustrated for vertex 8, but here the closest facet is a vertex.

adopt the local point-in-polygon method of Horn and Taylor [HT89]. First, we compute the closest mesh facet (triangle, edge, or point) in S_{i^V} to $\mathbf{x}_i^V$. The closest facet calculation is performed by first storing S_{i^V} in a CGAL surface mesh and then using its class functions and the locate function from the Polygon Mesh Processing package [BSM20, LRT20]. If the closest facet is an edge or a point, we add triangles from $\mathcal{S}$ that are incident to the vertices in the edge or the point respectively to the patch S_{i^V} (if they are not already in it). If more triangles were added, we recompute the closest mesh facet. We illustrate this process in Figures 5.6 and 5.7. If the closest facet is a triangle, we compute the sign depending on the side of the plane containing the triangle that the point lies on. If the closest faces is an edge or point we use the conditions from [HT89], which we summarize below:

- If the closest facet is an edge, then the sign is -1 if the edge is concave (as determined by the normals of the incident faces) and $+1$ if it is convex.
- If the closest facet is a vertex, then there exists a discrimination plane with an empty half-space. Choosing any such plane, the sign is -1 if the edges defining the plane are concave and $+1$ if they are convex.

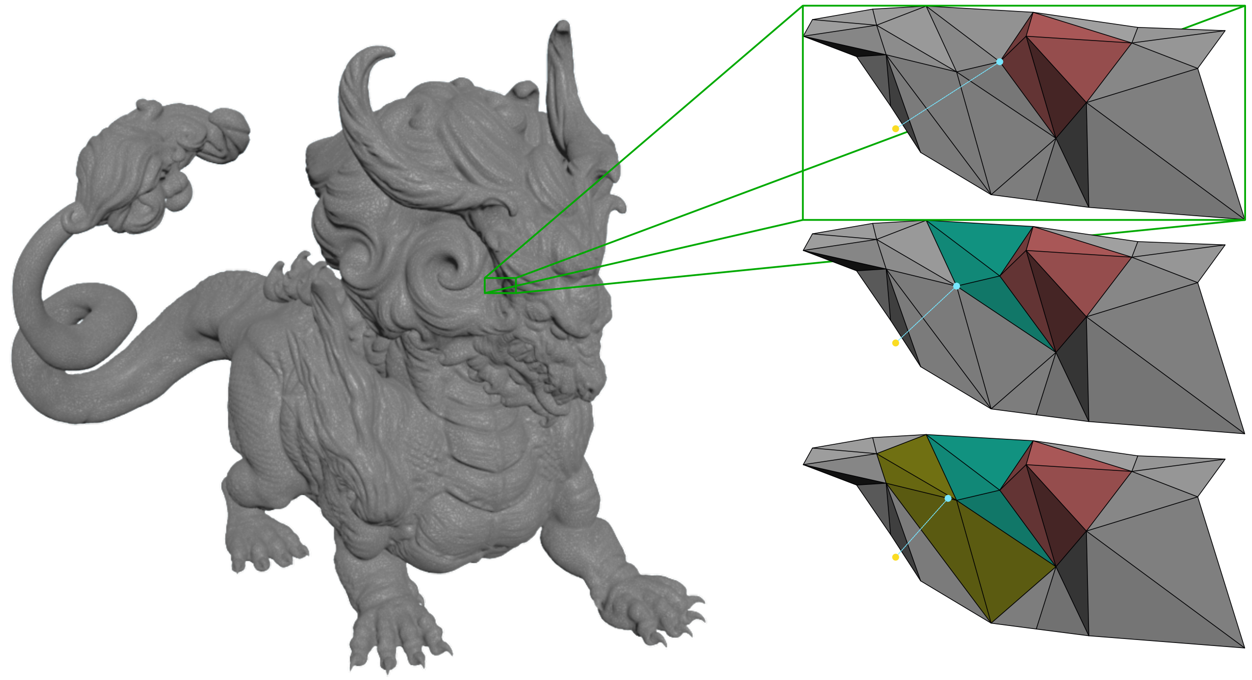


Figure 5.7: **Patch expansion.** The local patch $S_{i^{\mathcal{V}}}$ corresponding to the yellow vertex is shown. The initial patch is indicated in red, and the closest facet is a vertex of the red patch. We add the missing incident triangles (turquoise) and recompute the closest facet. This is again a vertex with incident triangles not in the patch, so we repeat the process (with new triangles in dark yellow). The closest feature is now on an edge, and we proceed to the edge criteria for signing.

A discrimination plane is defined by two non-collinear incident edges and it has an empty half-space if all incident faces and edges lie on one side of the plane or on the plane itself.

5.4 Interior Extension Region Creation

We grow the volumetric extension $\mathcal{V}^S$ on its interior boundary (defined by vertices with negative sign) to create the remainder of the volumetric mesh $\mathcal{V}$. We determine where to grow the extension by examining connected components of the background grid defined by its intersections with $\mathcal{S}$. We compute these components using depth-first search (as discussed

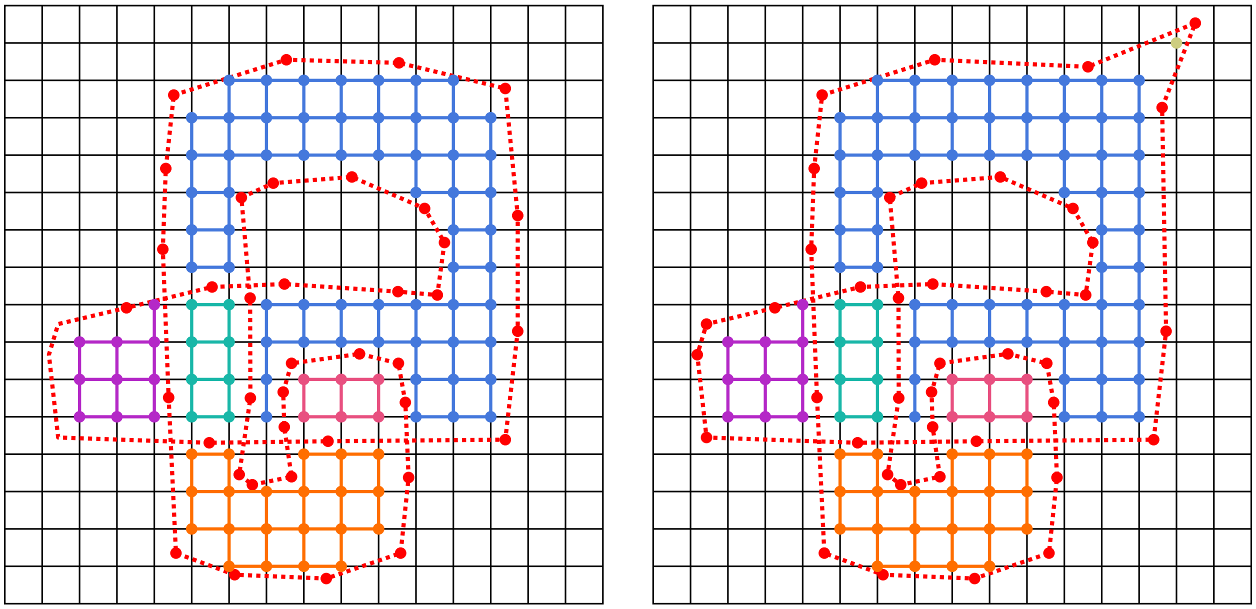


Figure 5.8: **Region over-count.** As the process of partitioning the grid only uses connectivity based on grid edges, it is possible for a contiguous region to be split into multiple regions. Shifting some of the vertices of $\mathcal{S}$ on the left results in the geometry on the right, which contains an additional region in the upper right corner since no edge connects this grid node to the larger blue region.

in Section 5.2.1), where adjacency between nodes in the background grid is defined between edge neighbors not divided by $\mathcal{S}$. We again use CGAL's intersection function from the 2D/3D Linear Kernel to determine whether or not an edge is divided. This is a simplistic criterion which can lead to an over-count in the number of interior regions, as demonstrated in Figure 5.8. A more accurate criteria would use material connectivity determined from the intersection of the surface $\mathcal{S}$ with the relevant background grid cells, similar to the CSG operations in [SDF07]. However, as noted in [LB18] these operations are extremely costly and our approach is robust to over-counting the number of interior regions since they are all merged together appropriately in the later stages of the algorithm.

Each connected component of background grid nodes constitutes a contiguous region. Regions that have a grid node with at least one geometrically coincident vertex in $\mathbf{x}^V$ with

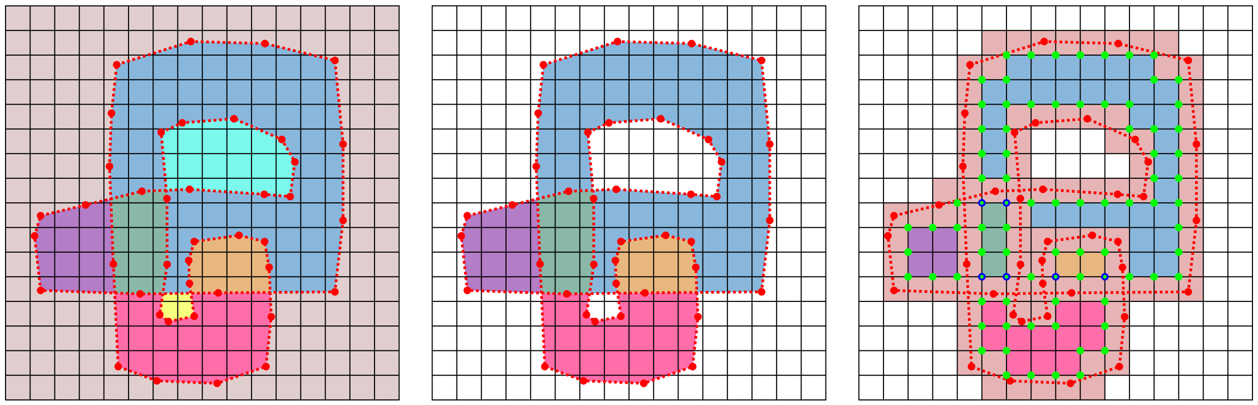


Figure 5.9: **Connected regions.** *(Left)* The surface partitions the background grid into contiguous regions. *(Middle)* The exterior regions are removed. *(Right)* The volumetric extension $\mathcal{V}^S$ is shown, along with the negatively signed vertices in green. Multiple geometrically coincident vertices are indicated using blue circles with green centers.

negative sign are defined to be interior. Exterior regions, those not containing a grid node with a geometrically coincident vertex in $\mathbf{x}^V$ with negative sign, are discarded. We create at least one hexahedron mesh $\mathcal{V}^{j^I,c}$ for each interior region j^I. Multiple copies of interior meshes are created near self-intersecting portions of $\mathcal{S}$ since here they represent multiple overlapping portions of the volumetric domain. We illustrate this process in Figure 5.9. We note that as before, each hexahedron mesh $\mathcal{V}^{j^I,c}$ uses the common vertex array $\mathbf{x}^V$.

We determine interior regions j^I that require multiple copies as those with grid nodes that have more than one geometrically coincident vertex in $\mathbf{x}^V$ with negative sign. For these regions, we create a copy $\mathcal{V}^{j^I,c}$ for each connected component c of vertices in $\mathbf{x}^V$ with negative sign that are geometrically coincident with a grid node in the region, as shown in Figure 5.10. Adjacency between these vertices is defined if they are in a common hexahedron in the volumetric extension $\mathcal{V}^S$. In general, this will be an over-count as multiple connected components may ultimately correspond to the same copy. We note that this process is analogous to the cell creation portion of the method of Li and Barbič [LB18]. They show that in the case of simple immersions, the correct number of copies is equal to the winding

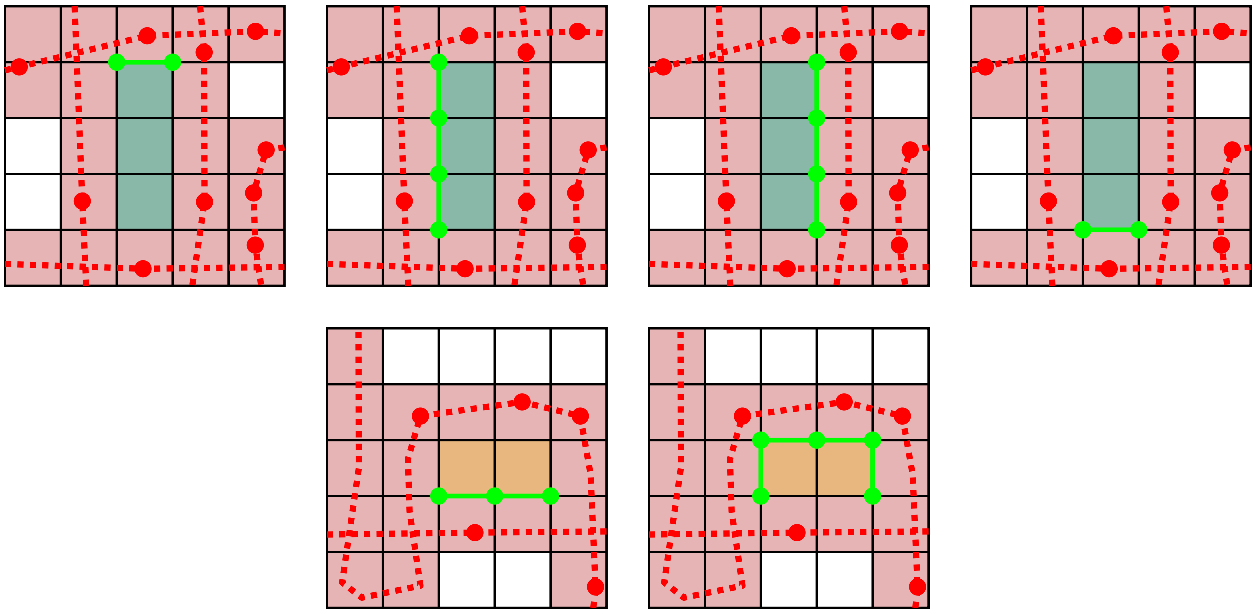


Figure 5.10: **Copy counting.** The two regions from Figure 5.9 having multiple copies are shown. Each copy is displayed with its corresponding connected component of vertices with negative sign.

number of the region. We do not compute the winding number since our over-count is typically resolved during the merging process described in Section 5.5. However, failure cases occur when the background uniform grid $\mathcal{G}_{\Delta x}$ cannot resolve thin features or high-curvature in $\mathcal{S}$. In these cases, an over-count that cannot be resolved in the later merging stages occurs. The background grid must be refined to resolve these cases, however using a strategy similar to that of Wang et al. [WJS14] we use a topology-preserving coarsening strategy (see Section 5.6) after the algorithm has run to prevent excessively small element sizes and associated high element counts. We also note again that unlike Li and Barbič [LB18], we cannot handle non-simple immersions.

As with $\mathcal{V}^S$, we construct the first copy of the hexahedron mesh for each interior region $\mathcal{V}^{j^I,0}$ from precursor hexahedron meshes $\mathcal{V}_{\mathbf{i}}^{j^I,0} = (\mathbf{x}^V, \mathbf{m}_{\mathbf{i}}^{V^{j^I,0}})$. Here $\mathbf{x}_{\mathbf{i}}$ are the grid nodes in region j^I. It should be noted that these are different than the vertices $\mathbf{x}_i^V \in \mathbf{x}^V$ and that $\mathbf{i} = (i_0, i_1, i_2)$ is used to denote the grid multi-index associated with the node. For each $\mathbf{x}_{\mathbf{i}}$,

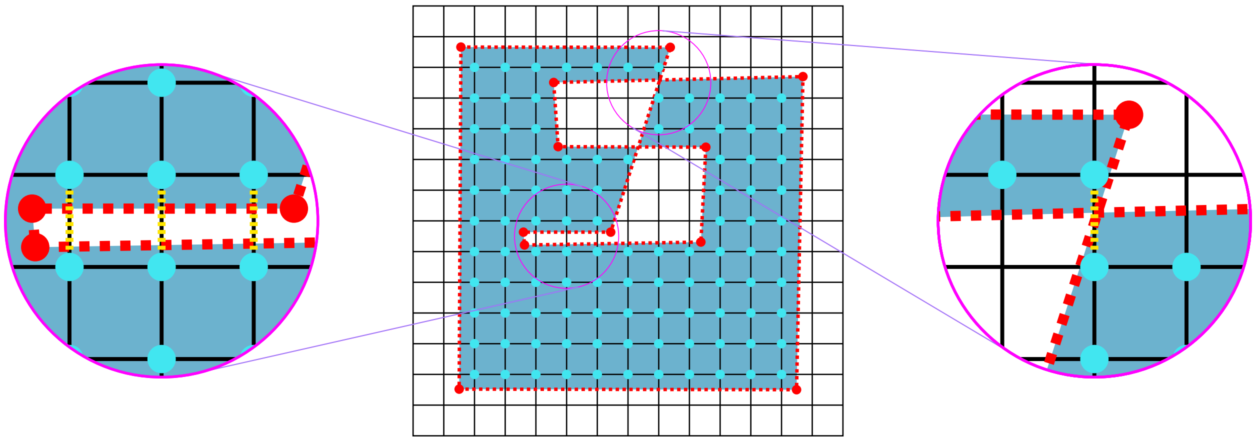


Figure 5.11: **Edge cut criterion.** Grid nodes $\mathbf{x_i}$ of a region are shown, along with two examples showing that adjacent grid nodes may have their common edge cut by a triangle (cut edges are indicated by the dashed yellow lines). In this case, adjacencies are not built between the corresponding vertices in $\mathcal{V}_{\mathbf{i}}^{j^I,0}$ to avoid unwanted sewing.

$\mathbf{m}_{\mathbf{i}}^{Vj^I,0}$ consists of 8 hexahedra which are geometrically coincident with the 8 local background grid cells incident to $\mathbf{x_i}$. Copies of $\mathbf{x_i}$ and the 26 background grid nodes surrounding $\mathbf{x_i}$ (whether or not they are in region j^I) are introduced into $\mathbf{x}^V$ to achieve this. We again merge these precursors as described in Section 5.2.1 where adjacencies between the vertices of $\mathbf{x}^V$ are defined as follows. For each pair of grid nodes $\mathbf{x_i}$ and $\mathbf{x_j}$ in region j^I, the geometrically coincident vertices in $\mathbf{x}^V$ corresponding to the hexahedra of $\mathcal{V}_{\mathbf{i}}^{j^I,0}$ and $\mathcal{V}_{\mathbf{j}}^{j^I,0}$ are adjacent if $\mathbf{x_i}$ and $\mathbf{x_j}$ are connected by an edge in $\mathcal{G}_{\Delta x}$ that is not cut by a triangle in $\mathcal{S}$. This edge cut criteria prevents connection between geometrically close but topologically distant features, as illustrated in Figure 5.11. We reemphasize that as described in Section 5.2.1 the final $\mathbf{m}^{Vj^I,0}$ is formed by concatenating all of the arrays $\mathbf{m}_{\mathbf{i}}^{Vj^I,0}$ (modified to account for merged vertex numbering) and removing any duplicated hexahedra. The remaining copies $\mathcal{V}^{j^I,c}$ are created by duplicating $\mathbf{m}^{Vj^I,0}$ with new vertices distinct from those corresponding to $\mathcal{V}^{j^I,0}$ and any other copy.

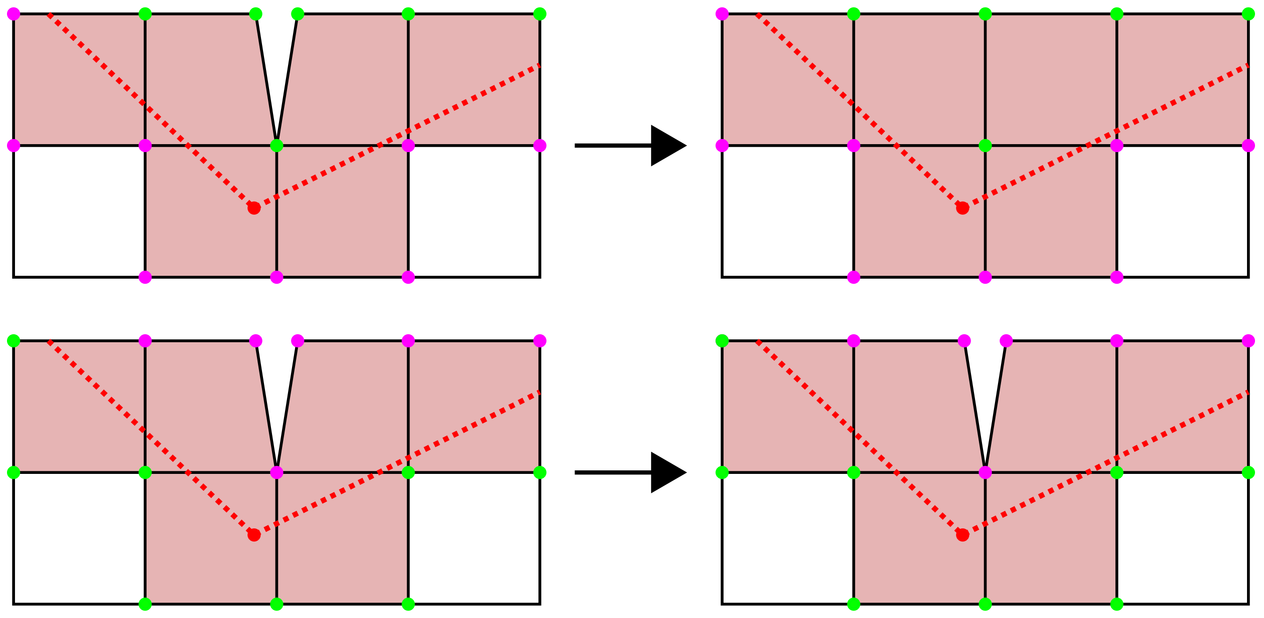


Figure 5.12: **Preliminary merge.** The construction of the volumetric extension $\mathcal{V}^S$ may result in geometrically coincident vertices which do not come from topologically distant parts of the mesh. Green vertices have negative signs, while purple vertices have positive sign. Above: The process in Section 5.5.1 merges these vertices into a single vertex. Below: We do not merge coincident positive vertices, to avoid unnecessarily sewing the exterior.

5.5 Interior Extension Region Merging

Having created the interior extensions $\mathcal{V}^{j^I,c}$, the merging of these meshes with the volumetric extension $\mathcal{V}^S$ and with each other (to account for possible over-counting in their creation) is carried out in multiple steps. We first merge hexahedra from $\mathcal{V}^S$ into $\mathcal{V}^{j^I,c}$ in a process described below. We then determine which of the interior extensions should merge to each other, using hexahedra from $\mathcal{V}^S$ which merge into multiple $\mathcal{V}^{j^I,c}$ to generate a list of overlapping hexahedra between meshes of different regions and copies. Next, we use these overlaps to determine which copies of the same region are duplicated and merge the duplicates together. Finally, these overlapping hexahedra are used to define the adjacencies in the final merging process.

5.5.1 Merge With Boundary

Recall from Section 5.4 that in regions with more than one copy, we create a copy $\mathcal{V}^{j^I,c}$ for each connected component c of vertices in $\mathbf{x}^V$ located in region j^I with negative sign. We use $\mathcal{C}_c^{j^I}$ to denote the collection of these nodes in the connected component c. For regions with only one copy, $\mathcal{C}_0^{j^I}$ instead denotes the collection of all vertices in $\mathbf{x}^V$ located in region j^I with negative sign, as we do not generate connected components in this case. Not that for these single copy regions, the vertices of $\mathcal{C}_0^{j^I}$ need not be connected (see the geometry of Figure 5.13, where the vertices $\mathcal{C}_0^{j^I}$ are composed of two connected components on the outer and inner boundaries). We merge vertices of $\mathcal{V}^{j^I,c}$ with vertices in $\mathcal{C}_c^{j^I}$ using the merge described in Section 5.2.1. Before this merge, we first perform a preliminary merge of vertices in $\mathcal{C}_c^{j^I}$ which are geometrically coincident. Here, two vertices of $\mathbf{x}^V$ are adjacent if they are geometrically coincident and both in $\mathcal{C}_c^{j^I}$. The effect of this preliminary merge is to close unwanted interior voids without 'sewing' the exterior and without merging topologically distant vertices of $\mathcal{V}^S$, as shown in Figure 5.12. The merge between the vertices of $\mathcal{V}^{j^I,c}$ and $\mathcal{C}_c^{j^I}$ is then defined by the following adjacency. Vertices of $\mathcal{V}^{j^I,c}$ and $\mathcal{C}_c^{j^I}$ are adjacent if they are geometrically coincident and the vertex of $\mathcal{V}^{j^I,c}$ was created from an interior connected component of vertices in the $\mathcal{V}_{\mathbf{i}}^{j^I,0}$ that gave rise to $\mathcal{V}^{j^I,c}$ via the merge described in Section 5.4. Here, an interior connected component is one that contains the center vertex (as opposed to one of the surrounding 26 vertices) introduced in the creation of $\mathcal{V}_{\mathbf{j}}^{j^I,0}$ for some grid node $\mathbf{x}_{\mathbf{j}}$ in the region j^I. This requirement effectively means that vertices of $\mathcal{C}_c^{j^I}$ should only merge to the those vertices of $\mathcal{V}^{j^I,c}$ which are actually interior to the region, and not the vertices which are overlapping from a topologically far part of $\mathcal{V}^{j^I,c}$. We illustrate this in Figure 5.13. Note that after this merge has been performed, we update the indices in $\mathcal{C}_c^{j^I}$ accordingly as this set will be used in latter steps of the merging procedure.

We next use a strategy different to that in Section 5.2.1 for merging hexahedral elements in $\mathcal{V}^S$ to their geometrically coincident counterparts in$\mathcal{V}^{j^I,c}$. This modified merging strategy is designed to prefer the structure of $\mathcal{V}^S$ over that in $\mathcal{V}^{j^I,c}$. For instance, if two hexahedra

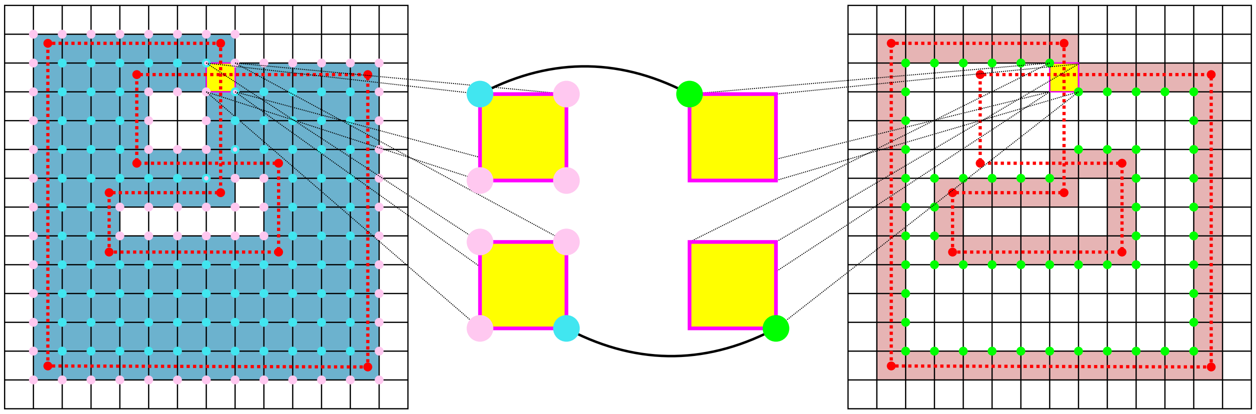


Figure 5.13: **Vertex adjacency.** The merge process between vertices of $\mathcal{V}^{j^I,c}$ and $\mathcal{C}_c^{j^I}$. For the cell highlighted in yellow, there are 2 hexahedra from $\mathcal{V}^{j^I,c}$ and therefore 4 pairs of geometrically coincident vertices. The two negatively signed vertices (in green) from $\mathcal{C}_c^{j^I}$ are matched to the vertices which came from an interior connected component (marked in cyan) and not the ones which did not (marked in pink).

of $\mathcal{V}^S$ are geometrically coincident but share only vertices on one face, then they will still have this connectivity after merging to $\mathcal{V}^{j^I,c}$. We merge the hexahedra in $\mathcal{V}^S$ incident to the vertices in $\mathcal{C}_c^{j^I}$ to their geometrically coincident counterparts in $\mathcal{V}^{j^I,c}$. Specifically, for each vertex $\mathbf{x}_i^V$ with $i \in \mathcal{C}_c^{j^I}$ and $k_i^{\mathcal{V}^S} \in \mathcal{I}_i^{\mathcal{V}^S}$, the hexahedron $\lfloor \frac{k_i^{\mathcal{V}^S}}{8} \rfloor$ is marked for merging. We denote the collection of hexahedra in $\mathcal{V}^S$ marked to be merged with their counterparts in copy c of region j^I as $\mathcal{I}_H^{j^I,c}$. Note that it is possible that some hexahedra of $\mathcal{V}^S$ are not included in any such collection. To perform this modified merging procedure, we first remove hexahedra from $\mathbf{m}^{V^{j^I,c}}$ that are geometrically coincident with a hexahedron from $\mathcal{I}_H^{j^I,c}$ and incident to a vertex in $\mathcal{C}_c^{j^I}$. Note that a hexahedron in $\mathbf{m}^{V^{j^I,c}}$ can only be incident to a node in $\mathcal{C}_c^{j^I}$ after the merge described in the previous paragraph has been completed. Next, copies of the hexahedra in $\mathcal{I}_H^{j^I,c}$ are added to $\mathbf{m}^{V^{j^I,c}}$. The process following the preliminary merge is outlined in Figure 5.14.

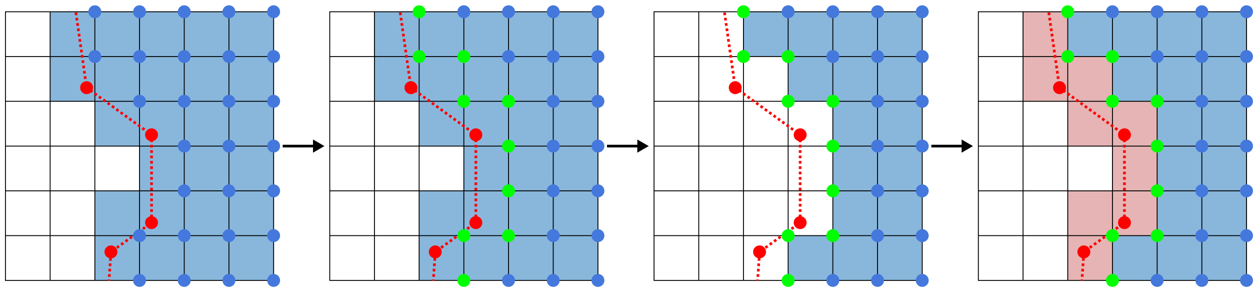


Figure 5.14: **Merge with boundary.** We illustrate the process of Section 5.5.1 following the preliminary merge of negatively signed vertices. First, specific vertices of $\mathcal{V}^{j^I,c}$ are merged with vertices of $\mathcal{C}_c^{j^I}$. Next, hexahedra to be replaced are removed from the $\mathcal{V}^{j^I,c}$. Finally, copies of hexahedra from $\mathcal{V}^S$ are added to this mesh.

5.5.2 Overlap Lists

We next merge differing regions $\mathcal{V}^{j_0^I,c}$ along their appropriately defined common boundaries. The boundary region between any two region copy meshes $\mathcal{V}^{j_0^I,c_0}$ and $\mathcal{V}^{j_1^I,c_1}$ is grown from seeds which we define by hexahedra in the respective meshes that are equal and in $\mathcal{V}^S$. For example, suppose that $\mathcal{V}^{j_0^I,c_0}$ and $\mathcal{V}^{j_1^I,c_1}$ contain such a hexahedron. In this case there are hexahedra with indices $h_{e_0}^{V^{j_0^I,c_0}}, h_{f_0}^{V^{j_1^I,c_1}} \in \mathbb{N}$ sharing the same vertices as a hexahedron in $\mathcal{V}^S$ with index $h_{g_0}^{V^S} \in \mathbb{N}$ such that

$$m^{V^{j_0^I,c_0}}_{8h_{e_0}^{V^{j_0^I,c_0}}+i^e} = m^{V^{j_1^I,c_1}}_{8h_{f_0}^{V^{j_1^I,c_1}}+i^e} = m^{V^S}_{8h_{g_0}^{V^S}+i^e}, \ \ i^e \in \{0, 1, \ldots, 7\}\,. \tag{5.1}$$

When these hexahedra exist in two region copies j_0^I, c_0 and j_1^I, c_1 we use the notation $\mathbf{q} = (j_0^I, c_0, j_1^I, c_1)$ to denote a pair of region copies with common boundary (that which will eventually merge). We define $ß_0^{\mathbf{q}} = (h_{e_0}^{V^{j_0^I,c_0}}, h_{f_0}^{V^{j_1^I,c_1}})$ as a seed between the pair of region copies. Furthermore, we use $\mathbf{p^q} = [ß_0^{\mathbf{q}}, \ldots, ß_{N_s^{\mathbf{q}}-1}^{\mathbf{q}}]$ to denote the collection of all such seeds between j_0^I, c_0 and j_1^I, c_1 with $N_s^{\mathbf{q}}$ being the number of seeds. This collection, which we call an overlap list, is grown into the complete overlapping common boundary between j_0^I, c_0 and j_1^I, c_1.

We expand the initial seed collections $\mathbf{p^q}$ by first marking background grid cells geomet-

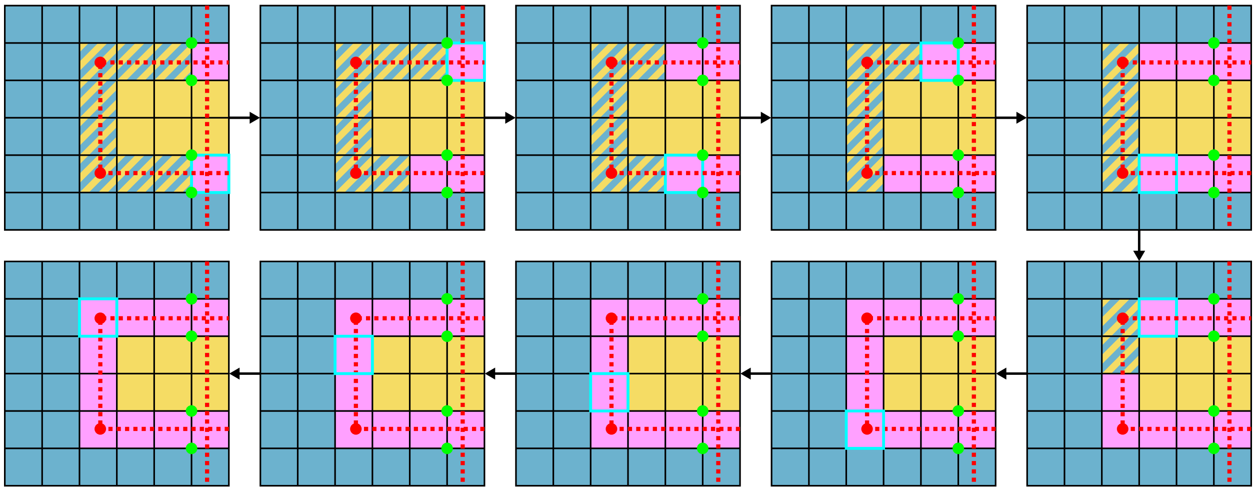


Figure 5.15: **Overlap lists.** A closeup of the overlap region from the geometry of Figure 5.13 is shown here. At the upper left, the seeds for the overlap between the two copies are shown in purple, as well as the incident negative vertices (green) to the seeds from each copy. At each step, the current seed is marked with a cyan border. New geometrically coincident neighbors of the seed hexahedra are then added in the next step. When all seeds have been traversed, the process stops.

rically coincident with hexahedra in the seeds as being visited. Then, starting with the seed $ß_0^{\mathbf{q}}$, we compute the neighbor hexahedra of each hexahedron in the seed (the neighbors of a hexahedron are those which share a common vertex). Geometrically coincident neighbors of the two hexahedra in the seed are added to $\mathbf{p^q}$ if the background grid cell to which they are geometrically coincident is unvisited. We then mark the cell as visited, and continue until every seed has been processed in this way. At the end of this expansion, $\mathbf{p^q}$ is a list of overlapping hexahedra that will be used to sew the regions together. We illustrated this process in Figure 5.15.

5.5.3 Deduplication

As mentioned in Section 5.4, the number of copies is generally an over count. We use the overlap lists $\mathbf{p}^{\mathbf{q}}$ to deduce which copies c of a region j^I are redundant. For each hexahedron h_e^S in $\mathcal{V}^S$, we create a list of hexahedra from geometrically coincident counterparts in interior region copies. This list is formed by considering each pair $\mathbf{q}$: if either hexahedron in a seed of $\mathbf{p}^{\mathbf{q}}$ is a copy of h_e^S (i.e. it uses the same vertices in $\mathbf{x}^S$ as in Equation (5.1)), both hexahedra in the seed are added to the list associated with h_e^S. Note that while the hexahedron pairs of the initial seeds in $\mathbf{p}^{\mathbf{q}}$ are both copies of hexahedra from $\mathcal{V}^S$ in accordance with Equation (5.1), subsequent seeds added during the overlap process may have both, one, or neither hexahedra equal to copies of hexahedra from $\mathcal{V}^S$. Should any list for any hexahedron h_e^S in $\mathcal{V}^S$ contain hexahedra from multiple copies c_0 and c_1 of the same region j^I, copies c_0 and c_1 are considered to be redundant duplicates of each other. Redundant copies are merged using the process of Section 5.5.1. This process is shown in Figure 5.16.

For each region, we compute connected components of its copies using duplication as the notion of adjacency. For each connected component of copies, we take the copy with the smallest index c_i as the representative copy. However, this copy's mesh only has the vertices of the component c_i. Likewise, only copies of the hexes in $\mathcal{I}_H^{c_i}$ are in $\mathcal{V}^{j^I,c_i}$. We remedy this by repeating the merge with boundary process of Section 5.5.1 on updated data. Specifically, we replace the connected component c_i of vertices with the union of all components c_j for copies in the connected component of copies. We then form an updated collection of incident hexahedra $\mathcal{I}_H^{c_i}$ before repeating the boundary merge process. Finally, we update the overlap lists. Any overlap list corresponding to a duplicated copy is recreated using the minimum representative in place of the original copy to account for updated hexahedron ordering. Redundant overlap lists resulting from this update are then discarded.

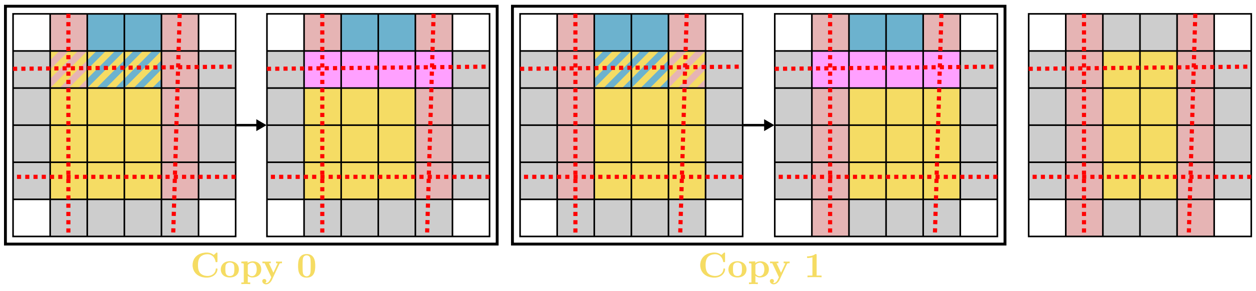


Figure 5.16: **Deduplication.** We show two of the four copies of the central region (yellow), corresponding to the right and left segments of $\mathcal{V}^S$. Each of copies 0 and 1 create an overlap list with the upper region (blue). The overlap list for copy 0 creates a pair between a non-boundary yellow hexahedron and a boundary hexahedron from the blue region. This boundary hexahedron is in a pair with a boundary hexahedron of copy 1, allowing us to deduce that copies 0 and 1 of the yellow region are duplicates. We then repeat the boundary merge process to create a deduplicated copy with complete boundary information.

5.5.4 Final Merge

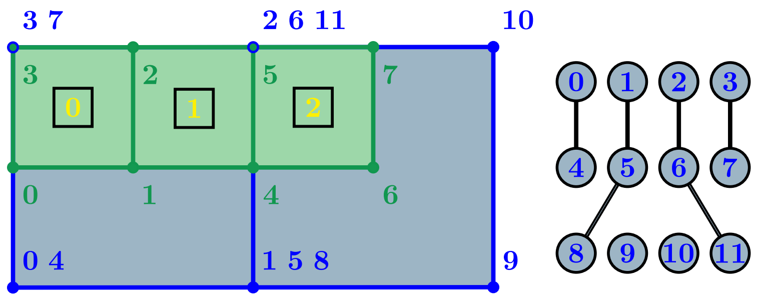
We now merge the vertices of $\mathbf{x}^V$ using the pattern of Section 5.2.1 with adjacencies defined by the overlap lists. For each seed ß in an overlap list, the geometrically coincident nodes of the two hexahedra in ß are considered adjacent. We then create the final mesh $\mathcal{V}$ by combining all of the arrays $\mathbf{m}^{V^{j^I,c}}$ from copies which are either the minimum representative, or not duplicated. Recall from Section 5.2.1 that some hexahedra of $\mathcal{V}^S$ are not copied into any copy's mesh. We add all such hexahedra to $\mathcal{V}$ to guarantee that $\mathcal{V}^S$ is contained in this final mesh, completing the interior extension region merging process.

5.6 Coarsening

Our method requires high-resolution (small Δx) background grids for high-curvature/detailed surfaces. We provide a topology-aware coarsening strategy to provide user control over the final volumetric mesh resolution/element counts. After the hexhedron mesh $\mathcal{V}$ is created, we coarsen the underlying grid by doubling Δx. We then create a maximal coarse mesh $\mathcal{M}$ based on the fine mesh $\mathcal{V}$. For each index m_j^V in $\mathcal{V}$, we define the initial connectivity for $\mathcal{M}$ as $m_j^M = j$. We then bin the center of each fine hexahedron $h^M \in \mathbb{N}^{N_e^M}$ into the coarsened grid and keep track of its multi-dimensional grid index $\mathbf{i}^{h^M}$. We initialize the position array $\mathbf{x}^M$ for $\mathcal{M}$ from the coarse grid cell corners of cell $\mathbf{i}^{h^M}$. Specifically, for each hexahedron in h^M in $\mathcal{M}$ we define $\mathbf{x}^M_{8h^M+i^e} = \mathbf{x}^{2\Delta x}_{\mathbf{i}^{h^M}} + \mathbf{o}_{i^e}$ where $\mathbf{o}_{i^e}$ is an offset from the coarse cell center $\mathbf{x}^{2\Delta x}_{\mathbf{i}^{h^M}}$ to the eight respective corners of the coarse grid cell $\mathbf{i}^{h^M}$. To build the final coarsened mesh, we merge portions of the maximal coarse mesh using Section 5.2.1 where adjacencies are defined from a hexahedron-wise notion of connectivity. Two maximal coarse hexahedra h_0^M and h_1^M are connected if their corresponding fine hexahedra $h_0^V = h_0^M$ and $h_1^V = h_1^M$ share a face $\mathbf{f}_i^V = \left[f_{i0}^V, f_{i1}^V, f_{i2}^V, f_{i3}^V\right] \in \mathbb{N}^4$ in $\mathcal{V}$. We define two types of connection: totally connected and partially connected. Maximal coarse hexahedra are totally connected if they have the same coarse grid index $\mathbf{i}^{h_0^M} = \mathbf{i}^{h_1^M}$ and their corresponding fine hexahedra h_0^V and h_1^V are not geometrically coincident. Maximal coarse hexahedra are partially connected if they are connected but are not totally connected. We define vertex adjacency from our notions of hexahedron connectivity. If two hexahedra h_0^M and h_1^M in the maximal coarse mesh are totally connected, then their eight respective geometrically coincident vertices are defined to be adjacent, i.e. vertex $m^M_{8h_0^M+i^e}$ is adjacent to vertex $m^M_{8h_1^M+i^e}$, $0 \leq i^e < 8$. If they are partially connected, then their corresponding fine hexahedra h_0^V, h_1^V share a face $\mathbf{f}_i^V = \left[f_{i0}^V, f_{i1}^V, f_{i2}^V, f_{i3}^V\right]$. We then identify an analogous face in each of h_0^V and h_1^V which we define in terms of the indices $k_{0\alpha}^V, k_{1\alpha}^V$, $\alpha \in \{0,1,2,3\}$. Only the vertices corresponding to the analogous face are defined to be adjacent

$$m^M_{8h_0^M+k_{0\alpha}^V} = m^M_{8h_1^M+k_{1\alpha}^V},\ \alpha \in \{0,1,2,3\}\,. \tag{5.2}$$

There are two cases that define the analogous face. First, if the fine hexahedron counterparts h_0^V, h_1^V are geometrically coincident, then the analogous face is the one on the analogous side of the coarse hexahedron. If they are not geometrically coincident, then the analogous face is the one geometrically coincident with the fine face defined from $\mathbf{f}_i^V$. The general coarsening



$$\mathbf{m}^V = [0,1,2,3,1,4,5,2,4,6,7,5]$$
$$\mathbf{m}^M = [0,1,2,3,4,5,6,7,8,9,10,11]$$
$$\mathbf{m}^M = [0,1,2,3,0,1,2,3,1,9,10,2]$$
$$\mathbf{m}^M = [0,1,2,3,1,9,10,2]$$

Figure 5.17: **Coarsening.** An example of fine mesh connections. Hexahedra 0 and 1 are totally connected, while hexahedra 1 and 2 are connected by a face. After merging the vertices of the coarse mesh (blue), the duplicated hexahedron (indicated in red) is removed.

procedure is illustrated in Figure 5.17.

5.7 Hexahedron Mesh To Tetrahedron Mesh Conversion

We design a topologically-aware BCC-based approach for the creation of a tetrahedron mesh $\mathcal{T}$ from the hexahedron mesh $\mathcal{V}$. We initialize the particle array for the tetrahedron mesh $\mathbf{x}^T$ to be the same as $\mathbf{x}^V$, but we add a new vertex in the center of each hexahedron and each boundary face. Tetrahedra are computed from the faces in the mesh $\mathcal{V}$. Normally a face in $\mathcal{V}$ would have one (boundary face) or two (interior face) incident hexahedra. However, since $\mathcal{V}$ is comprised of many geometrically coincident hexahedra there are more cases. We classify them as: standard boundary face (one incident hexahedraon), standard interior face (two non-geometrically coincident incident hexahedra), non-standard interior (more than two incident hexahedra, some geometrically coincident and some not geometrically coincident) and non-standard boundary (more than one incident hexahderon, all geometrically coincident). Each face contributes four tetrahedra to $\mathcal{T}$ in the case of standard boundary and standard interior faces. The tetrahedra consist of two vertices from the face and the cell centers on either side of the face in the case of standard interior faces. In the case of standard boundary faces, the face center is used in place of the second hexahedron center. For non-standard

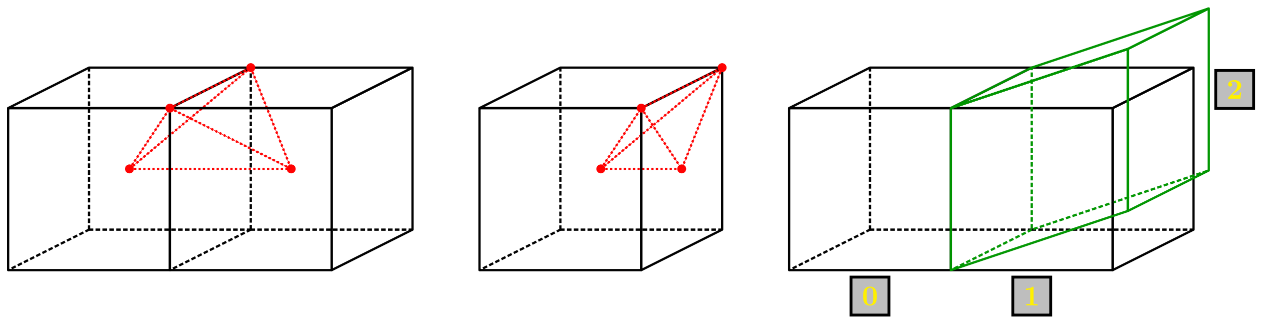
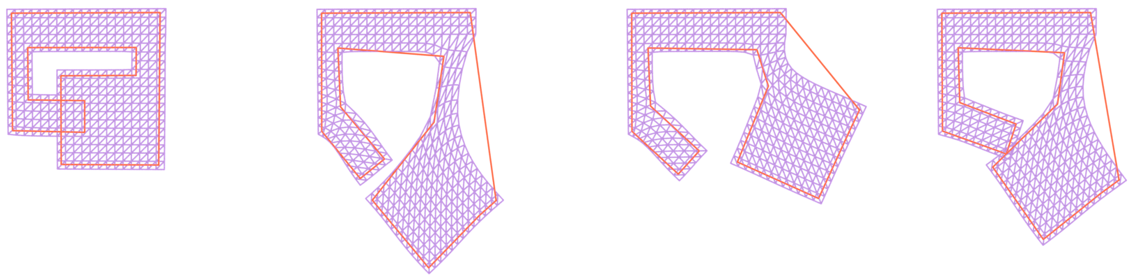


Figure 5.18: **Hexahedra tetrahedralization**. *(Left)* a standard interior face in $\mathcal{V}$. The centers of the two incident hexahedra are combined with two face vertices to form the tetrahedra (red). *(Middle)* a standard boundary face uses a face center instead of the missing incident hexahedron center. *(Right)* a non-standard interior face is shown. The right-most incident hexahedra are geometrically coincident. We form hexahedra pairs/faces (0,1), (0,2) and treat them respectively as standard interior, as in the left-most image.

interior faces, we take all pairs of non-geometrically coincident incident hexahedra and add tetrahedra as if their common face was a standard interior face. For non-standard boundary faces, tetrahedra are added for each incident hexahedron as if it were incident to a standard boundary face. We illustrate this procedure in Figure 5.18.

5.8 Examples

We consider a variety of examples in both two and three dimensions. To illustrate the capabilities of the final mesh connectivites, we treat the objects as deformable solids and run a finite element (FEM) simulation [SB12]. Performance statistics for the 3D examples are presented in Table 6.4. All experiments were run on a workstation with a single Intel® Core™ i9-10980XE CPU at 3.00GHz.



(a) Frame 0 (b) Frame 11 (c) Frame 27 (d) Frame 60

Figure 5.19: A self-intersecting shape is suspended from a ceiling. The geometry deforms under gravity, and both sides freely move regardless of the initial overlap.

5.8.1 2D Examples

5.8.1.1 Single Overlap

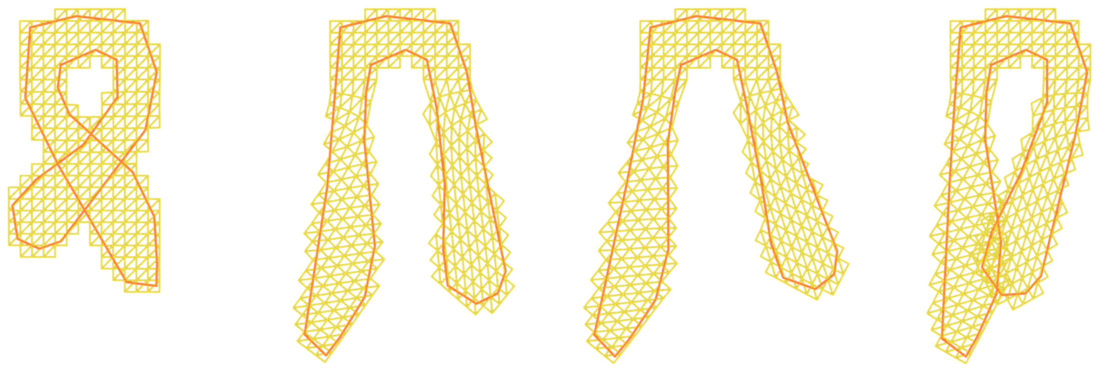
Figure 5.19 shows a deformable FEM simulation using a volumetric mesh produced by our algorithm. As evidenced by the geometry's ability to separate and freely move, our algorithm produces a mesh that properly resolves the single self-intersection present in the initial configuration.

5.8.1.2 Ribbon

Our algorithm can also handle more complex self-intersections. In Figure 5.20, one end of a ribbon shape passes through the other, partitioning the surface into several components. These intersections are successfully resolved, and the mesh is allowed to move as in the previous example.

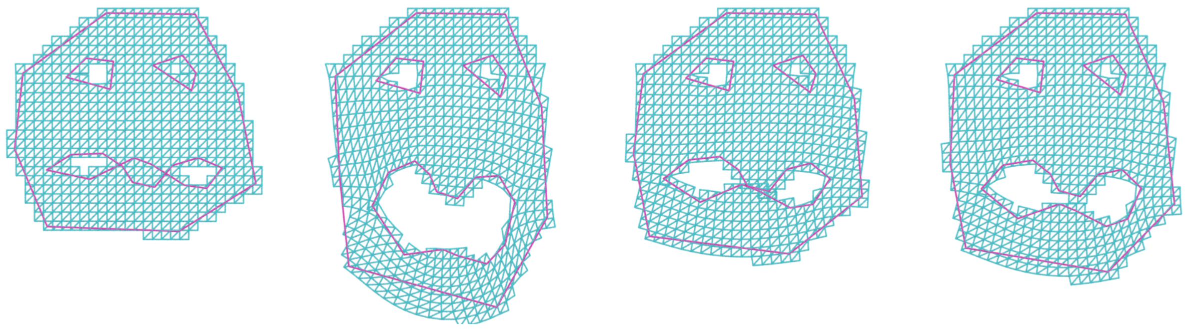
5.8.1.3 Face

Figure 5.21 demonstrates a similar scenario. In this case, the lips of the face geometry initially overlap; and, as an added challenge, the boundary of the input geometry consists



(a) Frame 0 (b) Frame 14 (c) Frame 59 (d) Frame 74

Figure 5.20: A ribbon with a more complicated initial self-intersection is also treated properly by our method.



(a) Frame 0 (b) Frame 8 (c) Frame 21 (d) Frame 92

Figure 5.21: A face with multiple boundary components and initially self-intersecting lips is successfully animated.

of multiple disconnected components. Our method successfully treats cases like these by design.

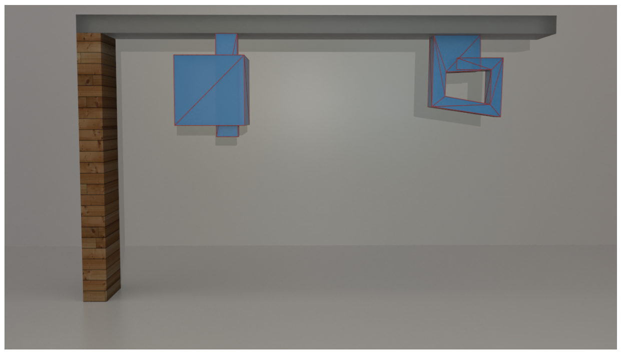
Table 5.1: Performance of generating volumetric meshes using our algorithm for various 3D examples. All times are in seconds and represent the total runtime of the algorithm.

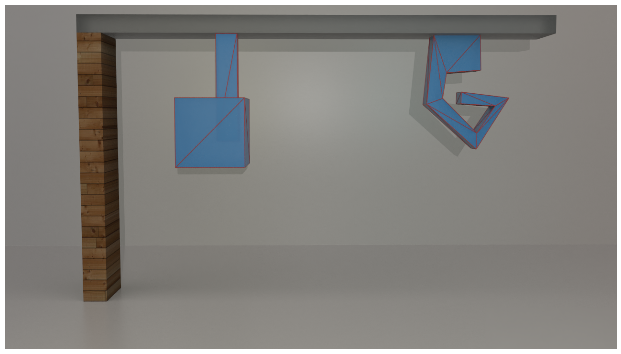
Example	Grid dim.	Δx	# Hex	Time (s)
Two Boxes	66×64×86	0.00955671	256368	2.80219
Simple Overlap	194×64×194	0.00328125	1606296	24.0179
Double Möbius	294×288×64	0.0347391	903653	33.6324
Twin Bunnies	162×166×128	0.0203027	1525821	31.1815
Dragon	512×690×520	0.0708709	20110457	303.301
Fancy Ball	130×132×128	2.82671	515400	25.8388
Head	512×830×718	0.000501962	62444819	839.951
Sacht	52×104×42	4.26331	112682	9.64888

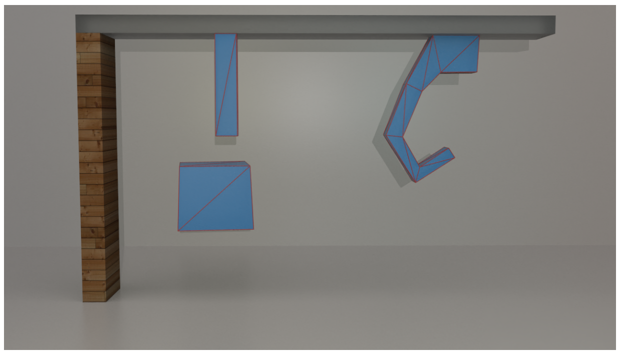
5.8.2 3D Examples

5.8.2.1 Two Boxes & Simple Overlap

We begin our 3D examples by demonstrating that our algorithm is able to quickly generate consistent meshes for simple self-intersecting geometries. In Figure 5.22, basic hand-made geometries are allowed to separate and unfurl from their initial self-intersecting states. The two boxes in the left-hand side of each subfigure were meshed using a background grid resolution of $66 \times 64 \times 86$ cells and $\Delta x = .00955671$, taking 2.80219s to generate the resulting 256,368 hexahedra in the output mesh. The simple overlapping shape in the right-hand side of each subfigure was meshed using a grid with $194 \times 64 \times 194$ cells and $\Delta x = .00328125$, resulting in 1,606,296 hexahedra in the output mesh.


(a) Frame 4


(b) Frame 9


(c) Frame 33

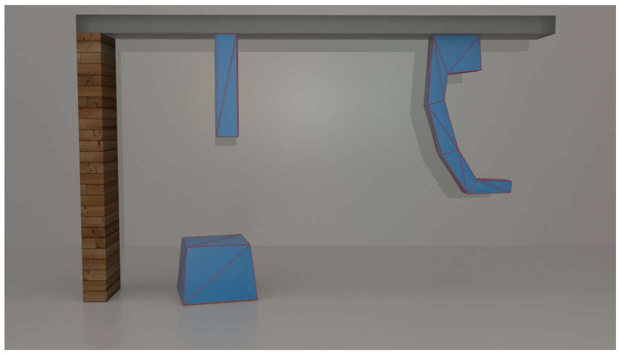

(d) Frame 48

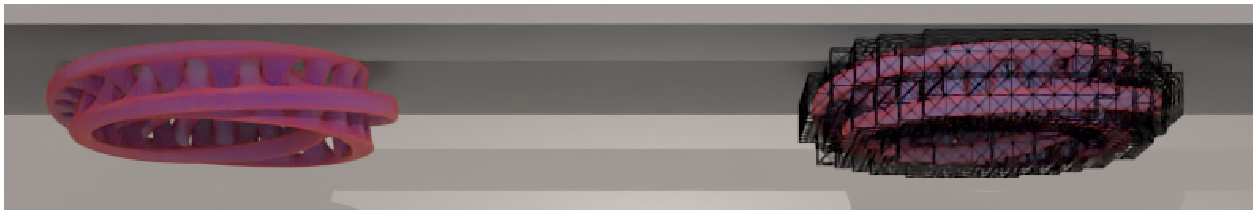
Figure 5.22: Simple self-intersecting 3D geometries are able to separate and unfurl with our algorithm.

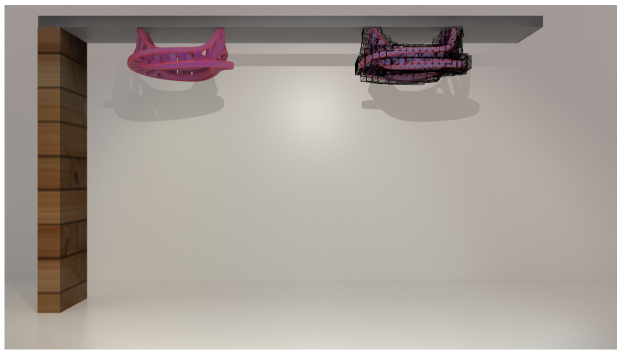
5.8.2.2 Double Möbius

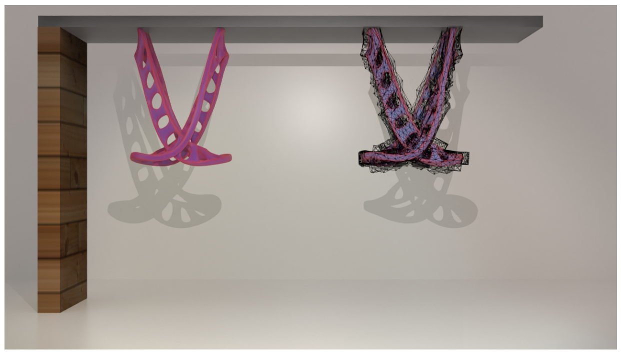
Figure 5.23 shows two Möbius-strip-like geometries[1] falling and separating under the effects of gravity, despite substantial intersections at the start of the simulation. This example was run using a background grid with $294 \times 288 \times 64$ cells and a Δx of 0.0347391. The resulting hexahedron mesh has 903,653 elements. Generating the volumetric mesh using our algorithm takes 33.6324s.

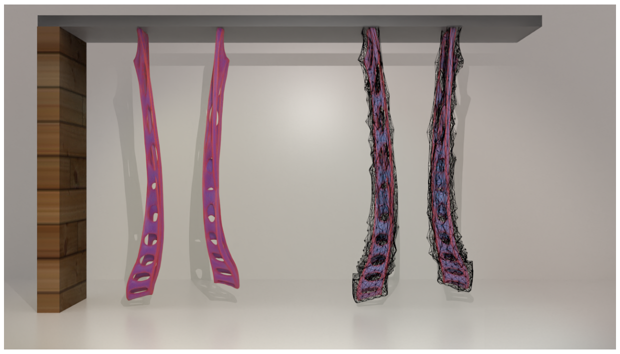
We also consider repeating this example at multiple spatial resolutions in order to demonstrate the effect of resolution on the quality of meshing results (see Figure 5.24). The coarsest

[1] "Mobius Bangle" by Creative_Hacker is licensed under CC BY 4.0.


(a) Frame 0


(b) Frame 20


(c) Frame 44


(d) Frame 78

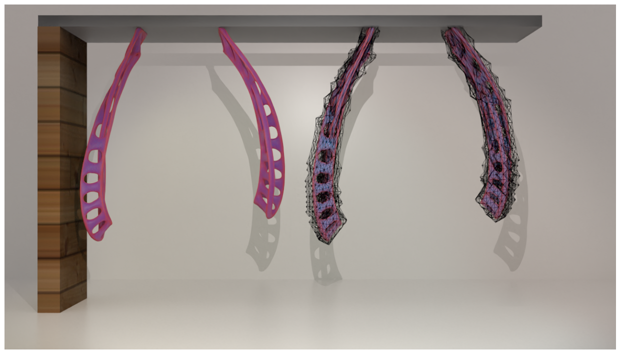
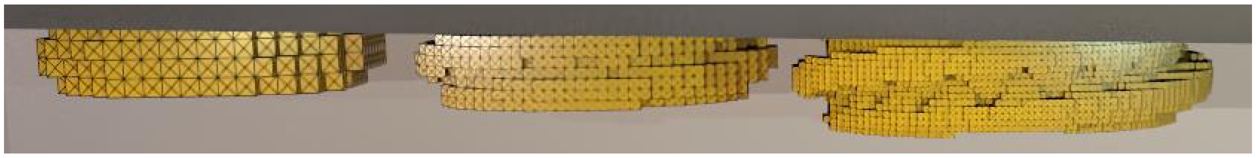
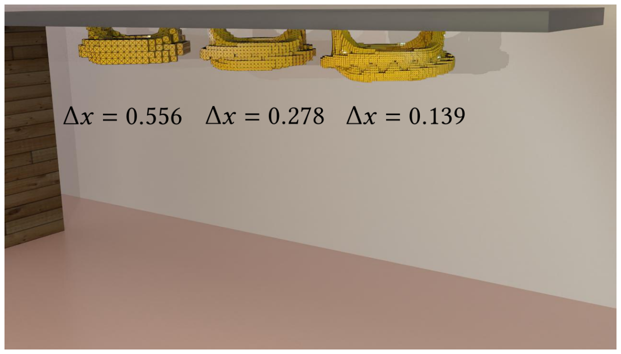

(e) Frame 110

Figure 5.23: Two intersecting Möbius-strip-like geometries (pink) naturally fall and separate under our method. The associated hexahedron meshes are shown in the right half of each frame.

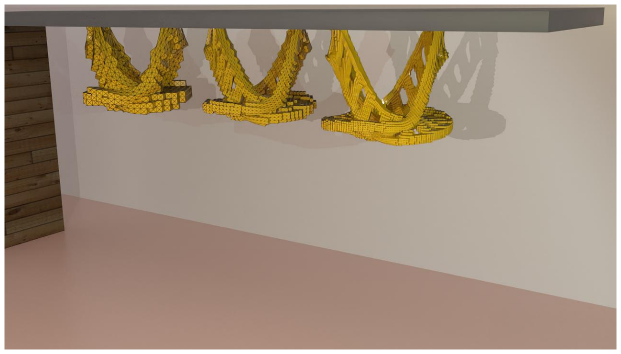
grid (corresponding to the leftmost meshes in each subfigure) is $21 \times 19 \times 5$ with $\Delta x = 0.556$. An intermediate grid resolution of $39 \times 37 \times 9$ cells with $\Delta x = 0.278$ corresponds to the middle meshes in each subfigure. The rightmost meshes in each subfigure come from using



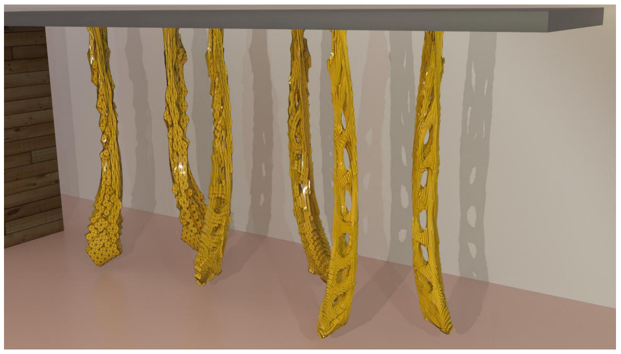
(a) Frame 0



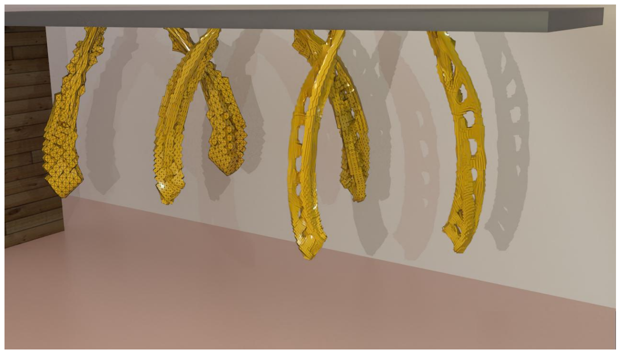
(b) Frame 16



(c) Frame 33



(d) Frame 84



(e) Frame 115

Figure 5.24: Running the example shown in Figure 5.23 at different spatial resolutions. In each frame, from left to right, the background grids have $\Delta x = 0.556$, 0.278, and 0.139.

a grid with $75 \times 73 \times 17$ cells with $\Delta x = 0.139$. Proper separation is achieved at all three of these tested resolutions, and in particular, our algorithm performs quite well on this example even at extremely low spatial resolution.

5.8.2.3 Twin Bunnies

Another standard example is the Stanford bunny. Figure 5.25 demonstrates that two almost completely overlapping bunny meshes can naturally separate under our method. No issues are encountered as different segments of the bunnies pass through one another. This example uses a grid resolution of $162 \times 166 \times 128$ cells with $\Delta x = 0.0203027$, resulting in a mesh with 1,525,821 hexahedra.

5.8.2.4 Dragon

The most complicated geometry we test our method on is the dragon[2] shown in Figure 5.26 (and also shown in Figure 5.7). Adequate resolution is required in order to resolve all the fine-scale features of this mesh; accordingly, we use a grid resolution of $512 \times 690 \times 520$ cells with $\Delta x = 0.0708709$. Our final mesh, generated in five minutes, contains just over 20 million hexahedra.

5.8.2.5 Fancy Ball

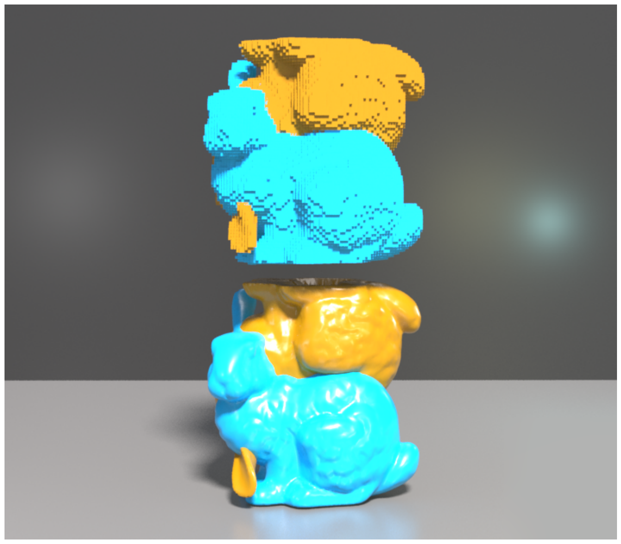
Figure 5.27 shows another interesting case where several ball-like geometries[3] deform and collide after being meshed with our algorithm. Each ball has a number of thin cuts and fine-scale features, which our algorithm is able to resolve using a grid with $130 \times 132 \times 128$ cells and $\Delta x = 2.82671$. The 515,400 resulting hexahedra are generated in 25.8388s.

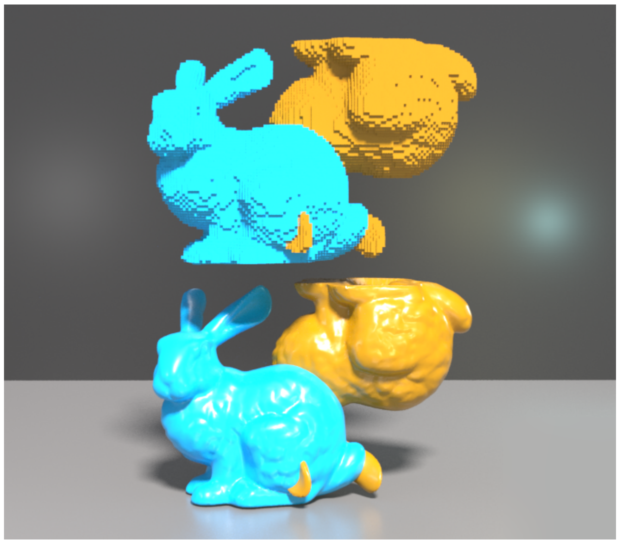
5.8.2.6 Head

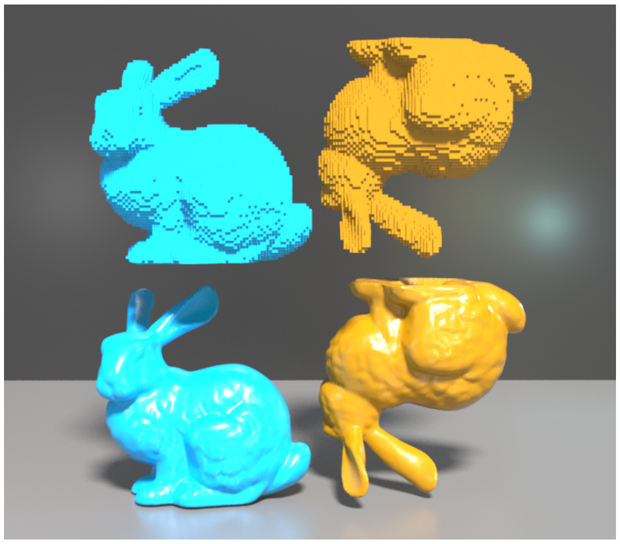
Modeling of the human body often gives rise to self-intersection. This is particularly common in the faces, where lip geometries often self-intersect. To that end, we consider a real-world

[2] "Asian Dragon" by Lalo-Bravo.

[3] "Abstract object" by sonic art.


(a) Frame 1


(b) Frame 27


(c) Frame 54

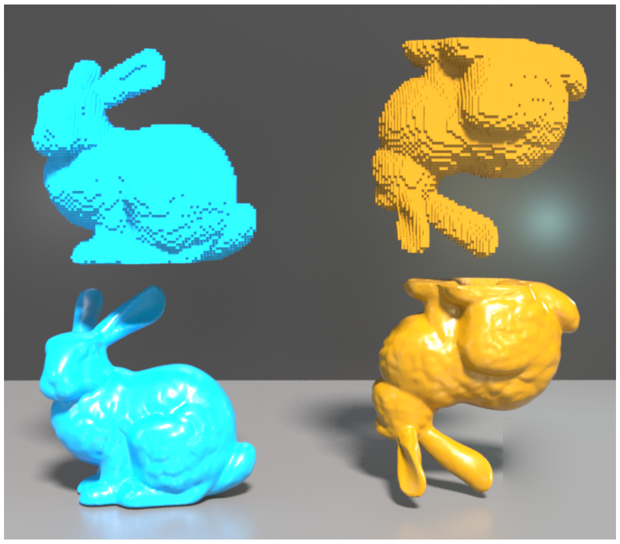
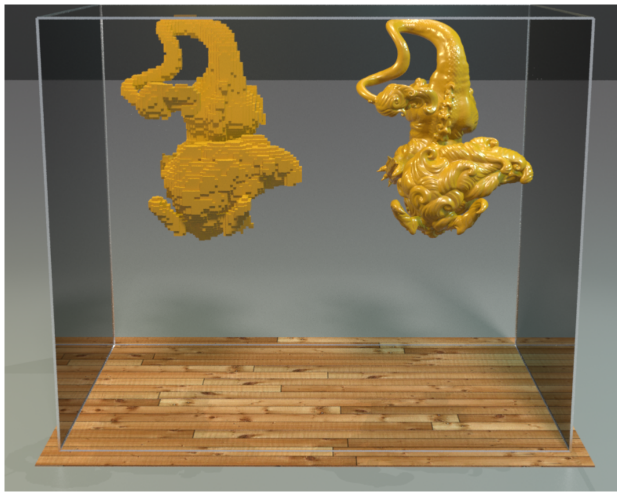
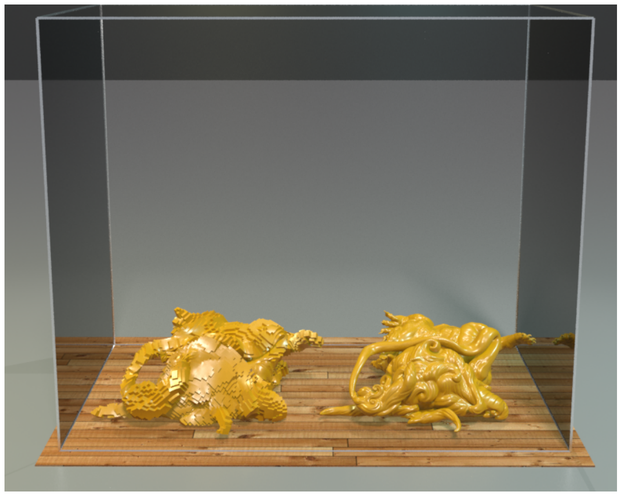

(d) Frame 81

Figure 5.25: Two overlapping bunnies naturally separate. The top part of each subfigure shows the meshes generated by our algorithm, while the bottom part of each subfigure shows the corresponding surface meshes.

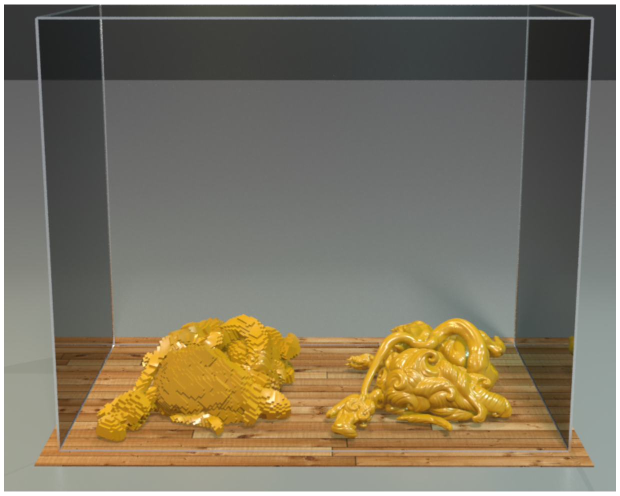
head geometry in Figure 5.28. Note that the lips separate effectively. This example results in a volumetric mesh with over 62 million elements, using a background grid resolution of $512 \times 830 \times 718$ cells and $\Delta x = 0.000501962$. Generating the hexahedron mesh takes 839.951s.



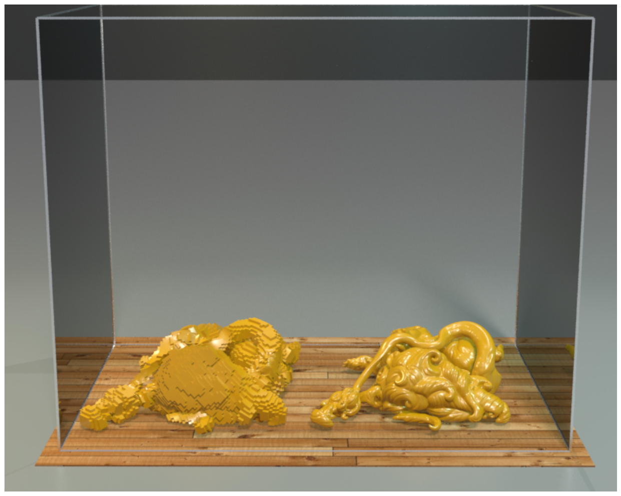
(a) Frame 0



(b) Frame 100

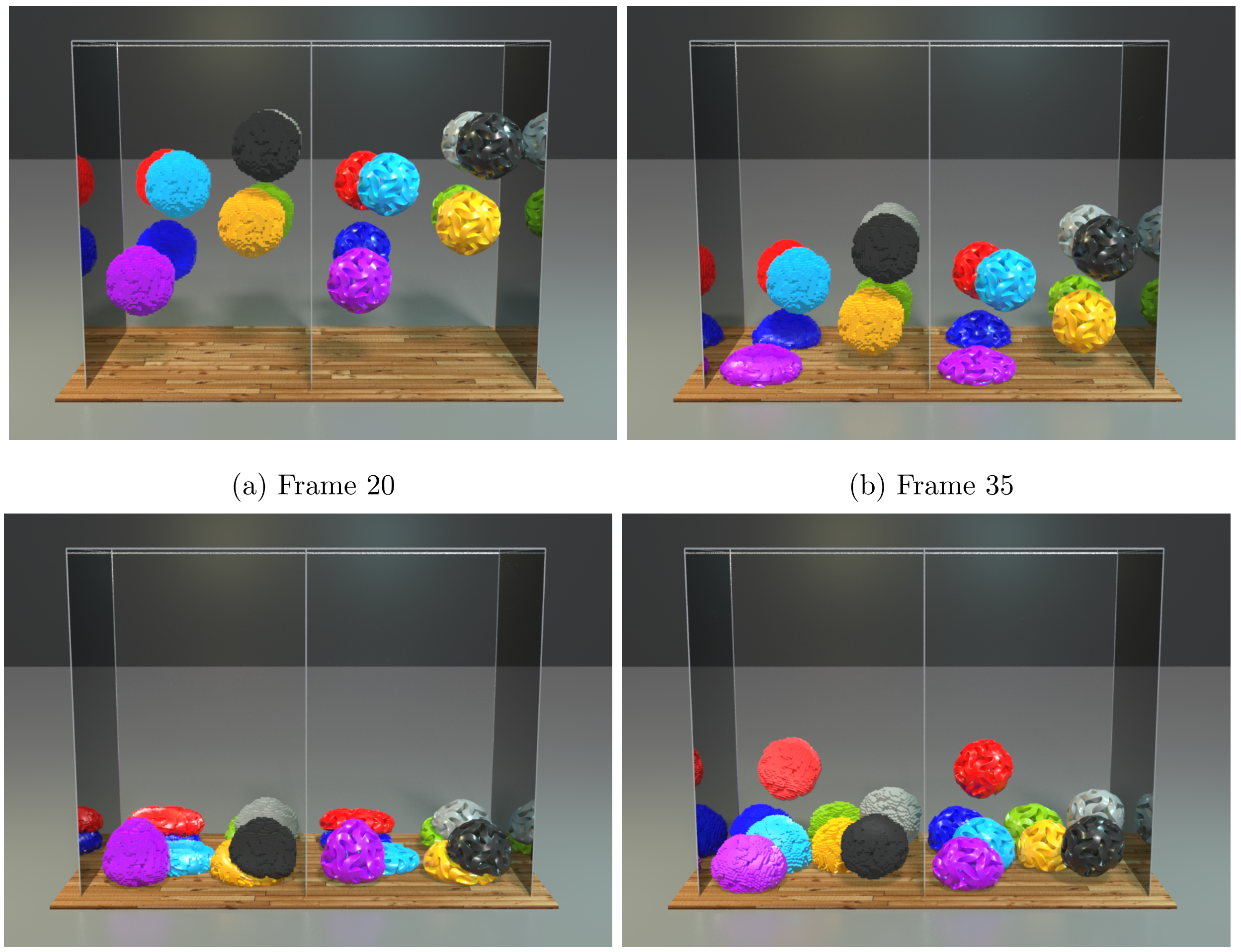


(c) Frame 200



(d) Frame 300

Figure 5.26: A complex mesh of a dragon is allowed to fall under gravity. The left-hand side of each subfigure shows the deforming mesh we generate, and each right-hand side shows the corresponding surface mesh.



(a) Frame 20 (b) Frame 35

(c) Frame 45 (d) Frame 80

Figure 5.27: Several ball-like geometries with intricate slices and holes are successfully meshed with our algorithm and then deform and collide under an FEM simulation.

5.8.2.7 Collection

Various objects from 3D examples are dropped in a tank in Figure 5.29. The objects naturally deform and collide without meshing or simulation issues.

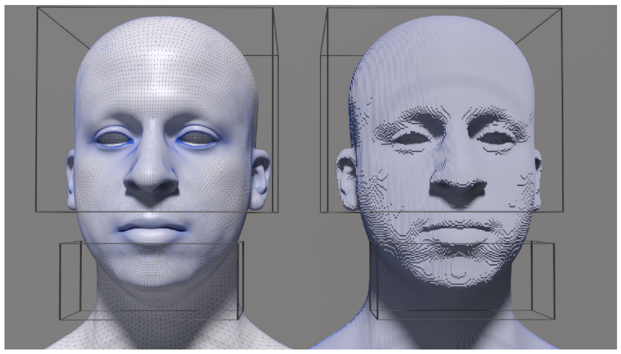
5.8.2.8 Sacht et al. Mesh

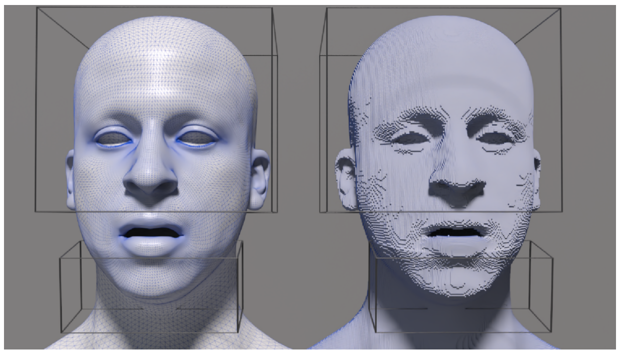
Finally, we demonstrate that our method, like that of Li and Barbič [LB18], can successfully separate the geometry shown in Figure 5.30 that is not supported by the method of Sacht et al. [SJP13]. In [SJP13], the bristles in this geometry get locked by the surrounding torus. However, both our method and [LB18] properly resolve all self-intersections. Of note, for a similar number of output mesh elements (112,682 vs. 112,554), our method runs noticeably faster than that of Li and Barbič [LB18] (9.65s vs. 22.5s).

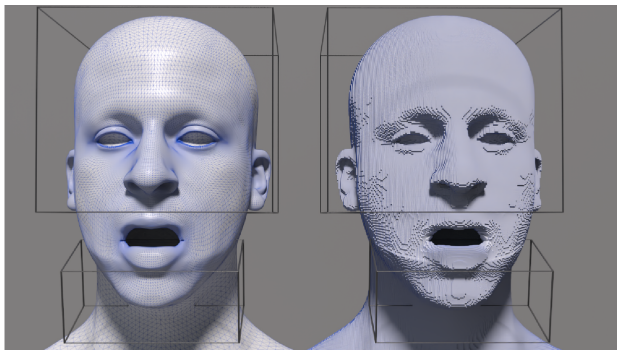
5.9 Discussion and Limitations

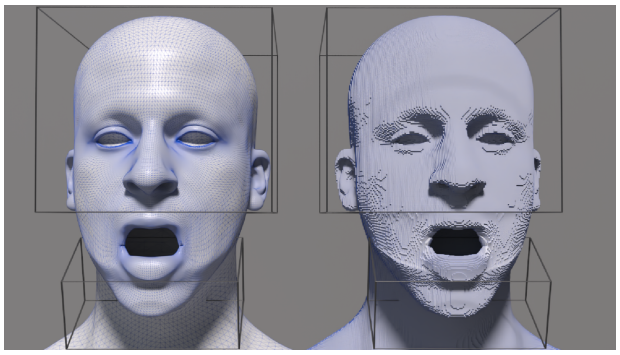
Our method has various limitations, most of which are attributed to our reduced use of exact/adaptive precision arithmetic. The most prominent limitations of our approach are in the types of input surface mesh $\mathcal{S}$ that we support. Fine-scale features, e.g., thin parallel sheets, can cause negatively signed vertices to be located in regions of the grid corresponding to an incorrect region. This may result in exterior regions erroneously generating copies, or interior regions creating extra copies which will not be correctly merged or deduplicated. In these pathological cases, the output mesh will have undesirable extraneous collections of hexahedra. We resolve these issues by refining the background grid, but very fine features may require refinement to an unreasonable resolution. However, our coarsening approach is designed to mitigate this. Even using added resolution and subsequent coarsening, our methodological simplifications prevent us from handling certain classes of cases that Li and Barbič [LB18] can handle, e.g., we cannot resolve non-simple immersions. It would be interesting to investigate whether our minimal-exact-arithmetic approach could be extended to handle non-simple immersions as well. Other future work includes improvements to the algorithm to handle known pathological cases without the need for refinement and subsequent coarsening, as well as improved detection mechanisms for such cases.

Lastly, Figure 1.1 illustrates an interesting case which neither our approach, that of Li and Barbič [LB18] nor that of Sacht et al. [SJP13] can handle. In this case, which is common near e.g. elbows and even shoulders in an upper torso, a portion of the domain overlaps in such a way that $\boldsymbol{\phi}_{\tilde{S}}^{S}$ must have negative Jacobian determinant in some regions. Our approach returns a mesh for this case, but it does not properly copy the overlap region and one of the two copies that would be required is rejected. I.e. our approach does not give a result consistent with creating a mesh in $\tilde{\mathcal{S}}^V$ and pushing it forward under $\boldsymbol{\phi}_{\tilde{S}}^{S}$. In Li and Barbič [LB18], this is noted as a case for which an immersion does not exist and Sacht et al. [SJP13] explicitly require the Jacobian determinant of $\boldsymbol{\phi}_{\tilde{S}}^{S}$ to be non-negative. However, this is a commonly occurring case which would be beneficial to resolve.


(a) Frame 0


(b) Frame 40


(c) Frame 80


(d) Frame 120

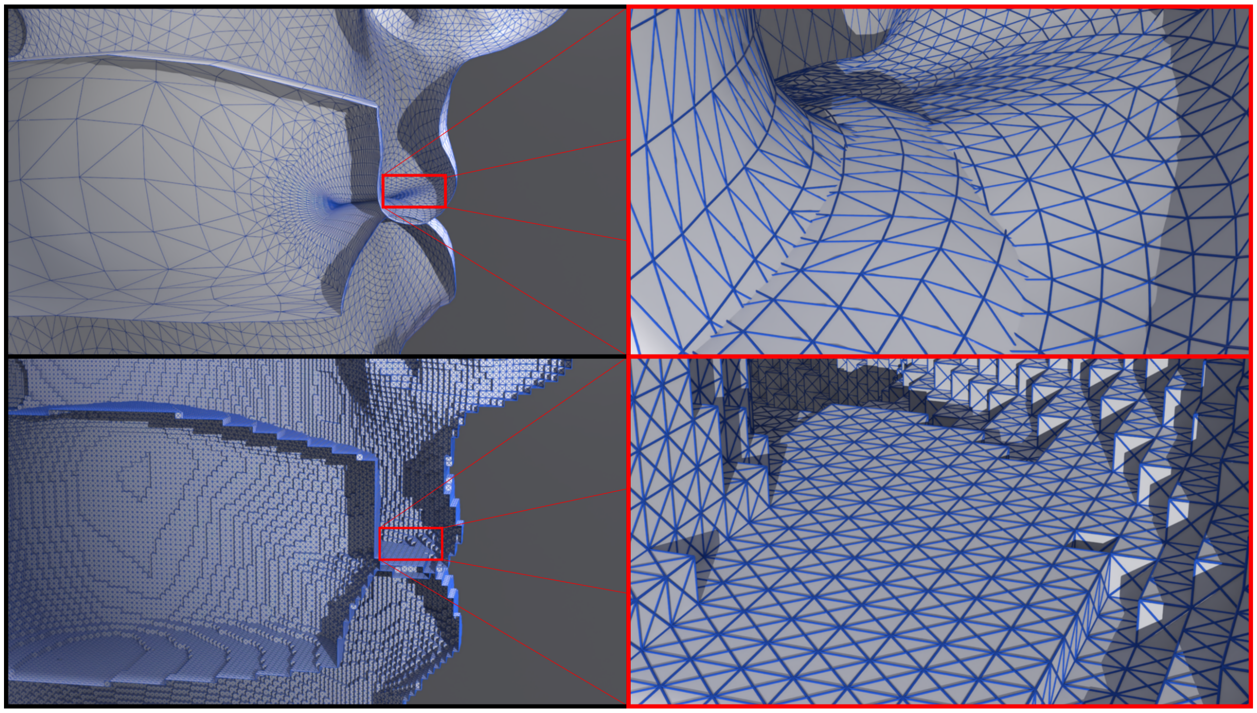
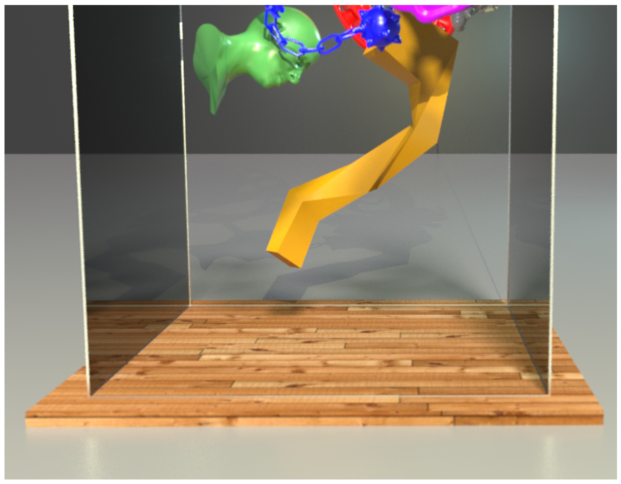
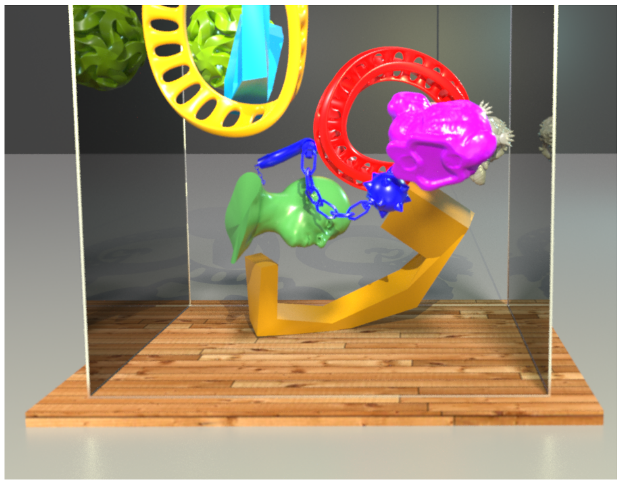

(e) Interior view of lips

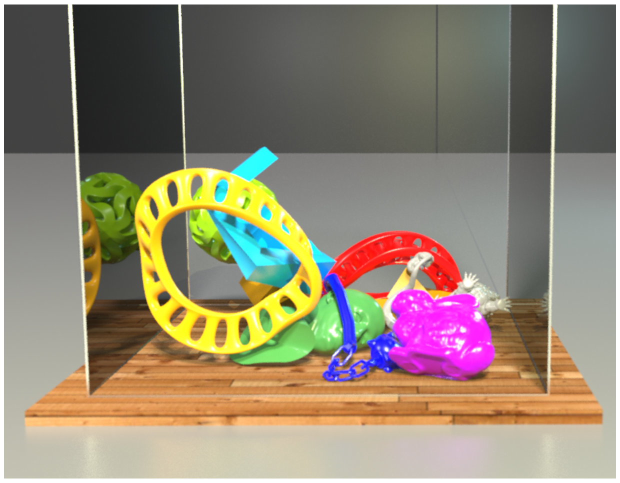
Figure 5.28: A face surface with self-intersecting lips is successfully meshed. The right-hand side of each of the first four frames shows the deformed hexahedron mesh, while each left-hand side shows the corresponding surface mesh. The wireframe boxes represent Dirichlet



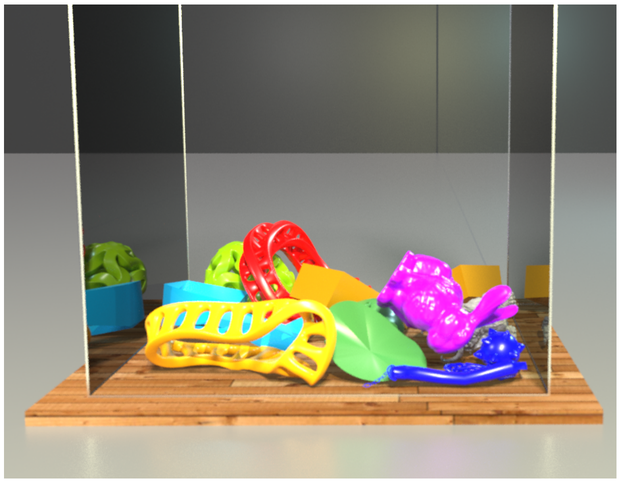
(a) Frame 60



(b) Frame 80

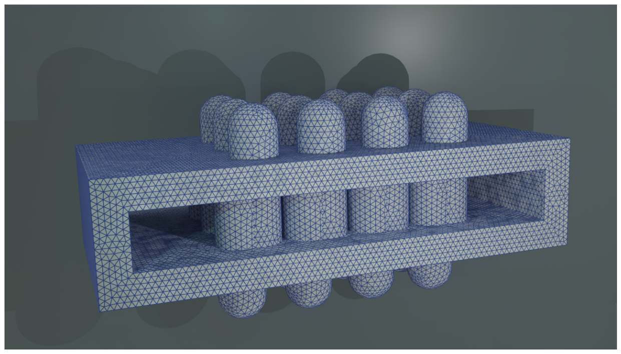


(c) Frame 100



(d) Frame 200

Figure 5.29: We simulated dropping our 3D examples into a box with a FEM sim.


(a) Initial State

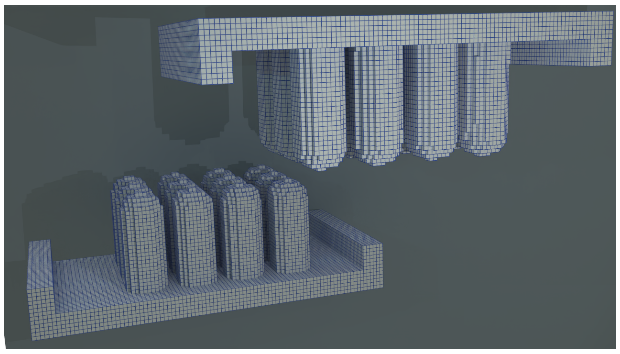

(b) Separation

Figure 5.30: Our method can successfully separate the torus and bristle geometry proposed in [SJP13].

CHAPTER 6

Position-Based Nonlinear Gauss-Seidel for Quasistatic Hyperelasticity

6.1 Previous work

Baraff and Witkin first demonstrated that implicit time stepping with elasticity is essential for efficiency [BW98]. Many approaches characterize implicit time stepping with hyperelasticity as a minimization of an incremental potential [MTG11, GSS15, LBO13, SD06, BML14, NOB16]. This is often referred to as variational implicit Euler [SD06, MTG11] or optimization implicit Euler [LBO13]. Quasistatic time stepping is an extreme case where inertia terms are ignored and only the strain energy is minimized [TSI05, SA07, LZX08, RPP17, KGL16]. Minimizers are usually found by setting the gradient of the energy to zero and solving the associated nonlinear system of equations with Newton's method. While Newton's method [NW06] generally requires the fewest iterations to reach a desired tolerance (often achieving quadratic convergence), each iteration can be costly and a line search is typically required for stability [GSS15]. There are many techniques that are less costly than Newton, but that can only reduce the system residual by a few orders of magnitude. However, many are satisfactory for visual accuracy. See discussion in Liu et al. [LBO13], Bouaziz et al. [BML14] and Zhu et al. [ZBK18].

Hyperelastic potentials must be rotationally invariant, non-negative and have global minima equal to zero at rotations. These considerations make the energy minimization non-convex with potentially non-unique solutions in quasistatic problems [BW08]. The non-

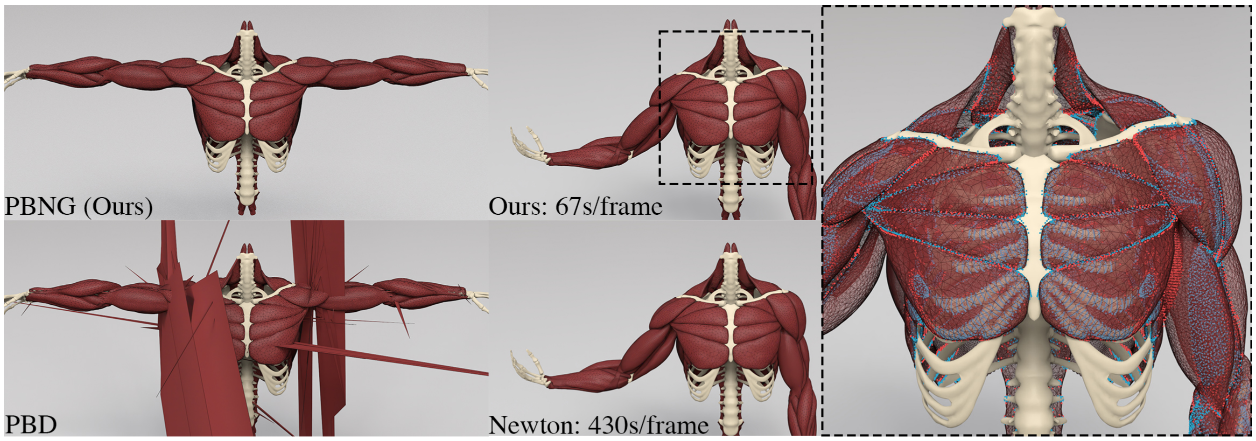


Figure 6.1: **Quasistatic Muscle Simulation with Collisions**. Our method (PBNG) produces high-quality results visually comparable to Newton's method but with a 6x speedup. In this hyperelastic simulation of muscles, we use weak constraints to bind muscles together and resolve collisions. The rightmost image visualizes these constraints. *Red* indicates a vertex involved in a contact constraint. *Blue* indicates a vertex is bound with connective tissues. PBD (lower left) becomes unstable with this quasistatic example after a few iterations.

convexity yields indefinite energy Hessians that can prevent convergence. Quasi-Newton methods can be used to approximate the Hessian with a symmetric semi-definite counterpart [TSI05, NW06, ZBK18, SGK19, LGL19]. Many methods avoid the indefiniteness issue with the inclusion of auxiliary (or secondary) variables. Narain et al. [NOB16], Bouaziz et al. [BML14], Liu et al. [LBO13] are recent examples of this, but similar approaches have been used in graphics since the local/global approach with ARAP by Sorkine et al. [SA07]. Rabinovich et al. [RPP17] generalize this approach to a wider range of distortion energies.

Methods like the Alternating Direction Method of Multipliers (ADMM) [BPC11, NOB16], the limited-memory Broyden-Fletcher-Goldfarb-Shanno algorithm (L-BFGS) [Ber97, ZBK18, LBK17, WWD21] and Sobolev preconditioned gradient descent (SGD) [Neu85, BML14, LBO13, SA07] require the inversion of a constant discrete elliptic operator (component wise-Laplacian) which can be pre-factored for efficiency. While this discrete operator does not suffer from indefiniteness issues, various authors note that SGD approaches may converge initially faster than Newton but will often taper off [ZBK18, LBO13, BML14, Wan15]. Zhu et al. [ZBK18] tailor their approach to this observation and use SGD initially and then combine with L-BFGS to incorporate more second-order information. Liu et al. [LBK17] and Witemeyer et al. [WWD21] also use combinations of SGD and L-BFGS. Kovalsky et al. [KGL16] add Nesterov acceleration to SGD. Hecht et al. [HLS12] develop efficient updates for a pre-factored Hessian with corotated materials. Wang [Wan15] discusses the challenges of using direct solution/pre-factoring in the SGD-style approaches of Narain et al. [NOB16], Bouaziz et al. [BML14] and Liu et al. [LBO13] and develops a Chebyshev acceleration technique as an alternative. In particular, they show that pre-factored discrete elliptic operators are memory-intensive (particularly at high-resolution) and limited since forward and backward substitutions do not parallelize. Moreover, [Wan15] show that simply replacing the direct solver with an iterative solver with reduced iteration count can lead to visually implausible or even unstable behaviors. However, Fratarcangeli et al. [FVP16] show that Gauss-Seidel iteration does not suffer from the same limitations in this context, although it does require degree of freedom coloring to facilitate parallel computation.

Tournier et al. [MNG15] develop a technique for bridging elasticity and constraint-based approaches that is robust to large stiffness. They use a similar primal/dual setup to XPBD. However, unlike XPBD, their approach solves the entire system at once, rather than iterating over individual constraints. Wang and Yang [WY16] use a Chebyshev accelerated gradient descent approach for general hyperelasticity and FEM.

6.2 Gauss-Seidel Notation

Our approach, PBD and XPBD all use nonlinear Gauss-Seidel to iteratively improve an approximation to the solution $\mathbf{x}^{n+1} \in \mathbb{R}^{dN^N}$ of Equation (2.11). We use l to denote the l^{th} Gauss-Seidel iteration $\mathbf{x}^{n+1,l} \approx \mathbf{x}^{n+1}$. During the course of one iteration, degrees of freedom in the approximate solution will be updated in sub-iterates which we denote as $\mathbf{x}_{(k)}^{n+1,l}$ with $0 \leq k < N^{\mathrm{GS}}$. Here $\mathbf{x}_{(0)}^{n+1,l} = \mathbf{x}^{n+1,l}$ and $\mathbf{x}_{(N^{\mathrm{GS}}-1)}^{n+1,l} = \mathbf{x}^{n+1,l+1}$. For example, with PBD/XPBD, in the k^{th} sub-iterate, the nodes in the k^{th} constraint will be projected/solved for. In our position-based approach, in the k^{th} sub-iterate, only a single node i_k will be updated. It is important to introduce this notation, since unlike with Jacobi-based approaches, the update of the k^{th} sub-iterate will depend on the contents of the $k-1^{\mathrm{th}}$ sub-iterate.

6.3 Position-Based Dynamics: Constraint-Based Nonlinear Gauss-Seidel

Macklin et al. [MMC16] show that PBD [MHH07] can be seen to be the extreme case of a numerical method for the approximation of the backward Euler temporal discretization of the FEM spatial discretization of Equation (2.1)

$$\sum_{j=0}^{N^N-1} m_{ij} \left(\frac{\mathbf{x}_j^{n+1} - 2\mathbf{x}_j^n + \mathbf{x}_j^{n-1}}{\Delta t^2} \right) = \mathbf{f}_i(\mathbf{x}^{n+1}) + \mathbf{f}_i^{\mathrm{ext}}, \ \mathbf{X}_i \notin \Omega_D^0. \tag{6.1}$$

Here $m_{ii} = \int_{\Omega^0} R^0 N_i d\mathbf{X}$ and $m_{ij} = 0$, $j \neq i$ are entries in the mass matrix. However, they require that the discrete potential energy in Equation (2.15) is of the form

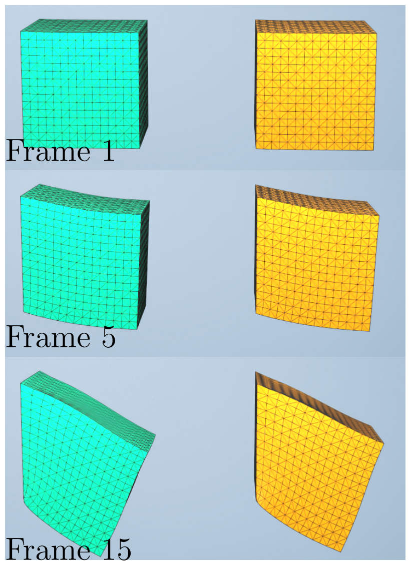
$$\hat{PE}^{\Psi}(\mathbf{y}) = \sum_{c=0}^{2N^E-1} \frac{1}{2\alpha_c} C_c^2(\mathbf{y}), \ \mathbf{y} \in \mathbb{R}^{dN^E}. \tag{6.2}$$

To demonstrate the connection between Equation (B.14) and PBD, Macklin et al. [MMC16] develop XPBD. It is based on the total Lagrange multiplier formulation

$$\sum_{j=0}^{N^N-1} m_{ij} \left(\mathbf{x}_j^{n+1} - \hat{\mathbf{x}}_j \right) - \sum_{c=0}^{P-1} \frac{\partial C_c}{\partial \mathbf{x}_i}(\mathbf{x}^{n+1}) \lambda_c^{n+1} = 0, \ \mathbf{X}_i \notin \Omega_D^0 \tag{6.3}$$

$$C_c(\mathbf{x}^{n+1}) + \frac{\alpha_c}{\Delta t^2} \lambda_c^{n+1} = 0, \ 0 \leq c < P \tag{6.4}$$

where $\hat{\mathbf{x}}_j = 2\mathbf{x}_j^n - \mathbf{x}_j^{n-1} - \frac{\Delta t^2}{m_{jj}} \mathbf{f}_j^{\mathrm{ext}}$ and $\boldsymbol{\lambda}^{n+1} \in \mathbb{R}^P$ is introduced as an additional unknown. The $\mathbf{x}^{n+1} \in \mathbb{R}^{dN^N}$ in Equations (B.18)-(B.19) is the same in the solution to Equation (B.14).



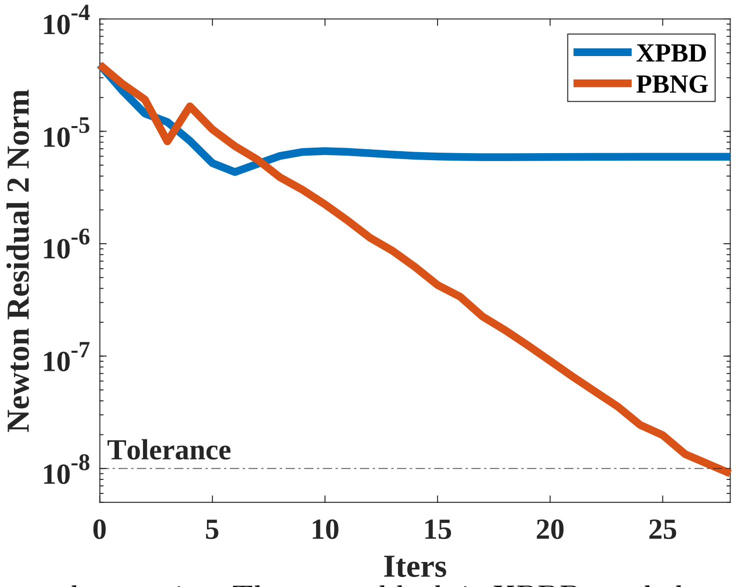


Figure 6.2: **Left**. Clamped blocks under gravity. The green block is XPBD, and the yellow one is PBNG. **Right**. PBNG is able to reduce the Newton residual to the tolerance, whereas XPBD's residual stagnates.

Macklin et al. [MMC16] use a per-constraint Gauss-Seidel update of Equations (B.18)-(B.19)

$$\mathbf{x}_{i(k+1)}^{n+1,l} = \mathbf{x}_{i(k)}^{n+1,l} + \Delta\mathbf{x}_{i(k+1)}^{n+1,l}, \quad \mathbf{X}_i \notin \Omega_D^0 \tag{6.5}$$

$$\Delta\mathbf{x}_{i(k+1)}^{n+1,l} = \frac{\Delta\lambda_{(k+1)c_k}^{n+1,l}}{m_{ii}} \frac{\partial C_{c_k}}{\partial \mathbf{x}_i}(\mathbf{x}_{(k)}^{n+1,l}) \tag{6.6}$$

$$\Delta\lambda_{(k+1)c_k}^{n+1,l} = \frac{-C_{c_k}(\mathbf{x}_{(k)}^{n+1,l}) + \frac{\alpha_{c_k}}{\Delta t^2} C_{c_k}(\mathbf{x}_{(k)}^{n+1,l})}{\sum_{j=0}^{N^N-1} \frac{1}{m_{jj}} \sum_{\beta=0}^{d-1} \left(\frac{\partial C_{c_k}}{\partial x_{j\beta}}(\mathbf{x}_{(k)}^{n+1,l})\right)^2 + \frac{\alpha_{c_k}}{\Delta t^2}}. \tag{6.7}$$

Here the $k+1^{\text{th}}$ sub-iterate in iteration l is generated by solving for the change in a single Lagrange multiplier $\Delta\lambda_{(k+1)c_k}^{n+1,l}$ associated with a constraint c_k that varies from sub-iteration to sub-iteration. However, as pointed out in [CHC23], this Gauss-Seidel procedure does not converge to a solution of Equation (B.14). Chen et al. [CHC23] isolate the root cause of this as the omission of the residual of Equation (B.18) in the update of the Lagrange multiplier in Equation (B.22) and moreover that inclusion of the residual in the update leads to unstable behavior. We demonstrate this behavior and contrast with our approach in Figure 6.2.

6.3.1 Quasistatics

As noted by Macklin et al. [MMC16], the XPBD update in Equations (B.20)-(B.22) is the same as in the original PBD [MHH07] in the limit $\alpha_c \to 0$. By choosing a stiffness inversely proportionate to a parameter $s \geq 0$ and examining the limiting behavior of the equations being approximated, we see that the original PBD approach generates an approximation to the quasistatic problem (Equations (2.5)), albeit with the external forcing terms omitted. More precisely, define $\boldsymbol{\phi}_s$ to be a solution of the problem

$$sR^0 \frac{\partial^2 \boldsymbol{\phi}_s}{\partial t^2} = \nabla^{\mathbf{X}} \cdot \mathbf{P} + s\mathbf{f}^{\text{ext}}. \tag{6.8}$$

subject to the same boundary conditions in Equations (2.2)-(2.3). This is equivalent to scaling the α_c that would appear in Equation (2.1) (through $\mathbf{P}$) by s. The α_c are inversely proportionate to the Lamé parameters, so as $s \to 0$, the material stiffness increases. Since the inertia and external force terms in Equation (B.23) vanish as $s \to 0$, it is clear then that the original PBD formulation generates an approximation to the solution of a quasistatic problem with the external forcing $\mathbf{f}^{\text{ext}}$ omitted. Note that PBD does include the external forcing term in its initial guess $\mathbf{x}_i^{n+1} = \mathbf{x}_i^n + \Delta t \mathbf{v}_i^n + \frac{\Delta t^2}{m_{ii}} \mathbf{f}_i^{\text{ext}}$. However, the effect of the initial guess vanishes as the iteration count is increased. We demonstrate this in Section 6.8.4. Also, note that this is not the case in the XPBD formulation where $\alpha_c > 0$.

Unfortunately, XPBD cannot be trivially modified to run quasistatic problems. For example, omitting the mass terms on the left-hand side of Equation (B.18) makes the Gauss-Seidel update in Equations(B.20)-(B.22) impossible since there would be a division by zero. The simplest fix for quasistatic problems with XPBD is to run to steady state using a pseudo-time iteration as in [CMM20]. This prevents the need for scaling the α_c which inherently removes the external forcing terms and does not introduce a divide by zero in Equation (B.21). However, this is very costly since each quasistatic time step is essentially the cost of an entire XPBD simulation. We refer to this technique as XPBD-QS (see Section 6.8.4). In addition to the excessive cost of this approach, we also observe severe iteration-order dependent behavior of XPBD-QS in the presence of spatially varying constraints and where constraints of different types affect the same vertices (see Figure 1.2). We believe the omission of the primary residual noted by Chen et al. [CHC23] is the cause of this iteration-dependent behavior. Intuitively, the Gauss-Seidel update would have information about adjacent constraints if this it could be added stably.

6.4 Position-Based Nonlinear Gauss-Seidel

To fix the issues with PBD/XPBD and quasistatics, we abandon the Lagrange-multiplier formulation and approximate the solution of Equation (2.11) using position-centric, rather than constraint-centric nonlinear Gauss-Seidel. This update takes into account each constraint that the position participates in. Visual intuition for this is illustrated in top of Figure 6.4(a). More specifically, in the k^{th} sub-iterate of iteration l, we modify a single node i_k with $\mathbf{X}_{i_k} \notin \partial\Omega_D^0$ as

$$\mathbf{x}^{n+1,l}_{(k+1)i_k} = \mathbf{x}^{n+1,l}_{(k)i_k} + \Delta\mathbf{x}^{n+1,l}_{(k+1)i_k} \tag{6.9}$$

$$\Delta\mathbf{x}^{n+1,l}_{(k+1)i_k} = \underset{\Delta\mathbf{y}\in\mathbb{R}^d}{\text{argmin}} \ \hat{PE}(\mathbf{x}^{n+1,l}_{(k)} + \tilde{\mathbf{C}}^{i_k}\Delta\mathbf{y}) - \Delta\mathbf{y}\cdot\hat{\mathbf{f}}^{\text{ext}}_{i_k}.$$

Here $\tilde{\mathbf{C}}^{i_k} \in \mathbb{R}^{dN^E\times d}$ is a selection matrix that isolates the degrees of freedom on the node i_k and has entries $\tilde{C}^{i_k}_{j\alpha\beta} = \delta_{ji_k}\delta_{\alpha\beta}$. We solve this minimization by setting the gradient to zero

$$\mathbf{0} = \mathbf{f}_{i_k}(\mathbf{x}^{n+1,l}_{(k)} + \tilde{\mathbf{C}}^{i_k}\Delta\mathbf{x}^{n+1,l}_{(k+1)i_k}) + \hat{\mathbf{f}}^{\text{ext}}_{i_k}. \tag{6.10}$$

We use a single step of a modified Newton's method to approximate the solution of Equation (6.10) for $\Delta\mathbf{x}^{n+1,l}_{(k+1)i_k} \in \mathbb{R}^d$. We use $\Delta\mathbf{x}^{n+1,l}_{(k+1)i_k} = \mathbf{0}$ as the initial guess. We found that using more than one iteration did not significantly improve robustness or convergence. Our update is of the form

$$\Delta\mathbf{x}^{n+1,l}_{(k+1)i_k} = \left(\mathbf{A}^{n+1,l}_{(k+1)i_k}\right)^{-1}\left(\mathbf{f}_{i_k}(\mathbf{x}^{n+1,l}_{(k)}) + \hat{\mathbf{f}}^{\text{ext}}_{i_k}\right). \tag{6.11}$$

Here $\mathbf{A}^{n+1,l}_{(k+1)i_k} \approx -\frac{\partial \mathbf{f}_{i_k}}{\partial \mathbf{y}_{i_k}}(\mathbf{x}^{n+1,l}_{(k)}) \in \mathbb{R}^{d\times d}$ is an approximation to the potential energy Hessian/negative force gradient.

6.4.1 Modified Hessian

We choose the modified energy Hessian $\mathbf{A}^{n+1,l}_{(k+1)i_k}$ to minimize its computational cost. The true Hessian $\frac{\partial \mathbf{f}_{i_k}}{\partial \mathbf{y}_{i_k}} \in \mathbb{R}^{d\times d}$ has entries

$$\frac{\partial f_{i_k\alpha}}{\partial y_{i_k\beta}}(\mathbf{y}) = -\sum_{e=0}^{N^E-1}\sum_{\delta,\gamma=0}^{d-1} \mathcal{C}^e_{\alpha\gamma\beta\delta}(\mathbf{y})\frac{\partial N^e_{i_k}}{\partial X_\gamma}\frac{\partial N^e_{i_k}}{\partial X_\delta}V^0_e - \sum_{c=0}^{N^{\mathrm{wc}}-1}\left(w^c_{0i_k} - w^c_{1i_k}\right)^2 K_{c\alpha\beta},\ 0 \le \alpha,\beta < d \tag{6.12}$$

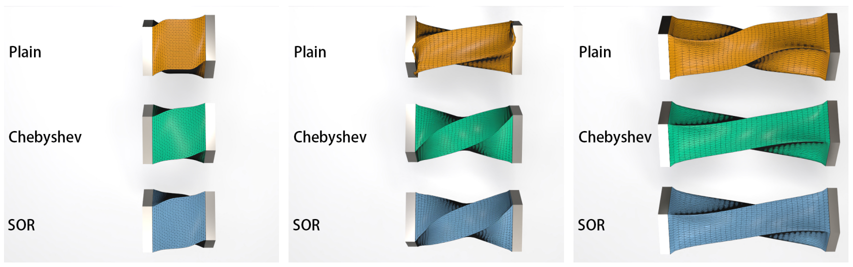
where $\mathcal{C}^e_{\alpha\gamma\beta\delta}(\mathbf{y}) = \frac{\partial^2\Psi}{\partial F_{\beta\delta}\partial F_{\alpha\gamma}}(\sum_{j=0}^{N^N-1}\mathbf{y}_j\frac{\partial N^e_j}{\partial \mathbf{X}})$ is the Hessian of the potential energy density evaluated at the deformation gradient in element e. This follows since the potential force on the node i_k is

$$\mathbf{f}_{i_k}(\mathbf{y}) = -\sum_{e=0}^{N^E-1}\hat{\mathbf{P}}_e(\mathbf{y})\frac{\partial N_{i_k}}{\partial \mathbf{X}}(\mathbf{X}^e)V^0_e - \sum_{c=0}^{N^{\mathrm{wc}}-1}\left(w^c_{0i_k} - w^c_{1i_k}\right)\mathbf{K}_c\mathbf{C}_c(\mathbf{y}) \tag{6.13}$$

where $\hat{\mathbf{P}}_e(\mathbf{y}) = \frac{\partial\Psi}{\partial \mathbf{F}}(\sum_{j=0}^{N^N-1}\mathbf{y}_j\frac{\partial N^e_j}{\partial \mathbf{X}})$ is the first Piola-Kirchhoff stress in the element.

The primary cost in Equation (6.12) is the evaluation of the Hessian of the energy density $\mathcal{C}^e_{\alpha\gamma\beta\delta}(\mathbf{y})$ which is a symmetric fourth order tensor with $d^2 \times d^2$ entries. Furthermore, this tensor can be indefinite, which would complicate the convergence of the Newton procedure. We use a definiteness projection as in [TSI05] and [SGK19]. However we use a very simple symmetric positive definite approximation instead of their approaches which require the singular value decomposition of the element deformation gradient $\sum_{j=0}^{N^N-1}\mathbf{y}_j\frac{\partial N^e_j}{\partial \mathbf{X}}$. Teran et al. [TSI05] also require the solution of a 3×3 and three 2×2 symmetric eigenvalue problems, our approach does not require this. Our simple approximation is $\tilde{\mathcal{C}}^e_{\alpha\gamma\beta\delta}(\mathbf{y}) \approx \mathcal{C}^e_{\alpha\gamma\beta\delta}(\mathbf{y})$ with

$$\tilde{\mathcal{C}}^e_{\alpha\gamma\beta\delta}(\mathbf{y}) = 2\mu\delta_{\alpha\beta}\delta_{\gamma\delta} + \lambda J {F^{e}}^{-1}{}_{\alpha\gamma}(\mathbf{y}) J {F^{e}}^{-1}{}_{\beta\delta}(\mathbf{y}). \tag{6.14}$$



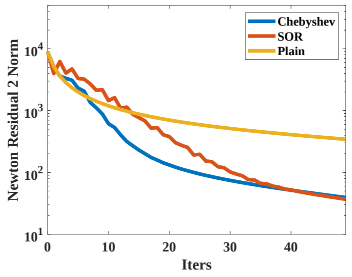


Figure 6.3: **Acceleration Techniques**. The convergence rate of PBNG may slow down as the iteration count increases. Chebyshev semi-iterative method and SOR effectively accelerate the Newton residual reduction.

Here $J\mathbf{F}^e(\mathbf{y}) = \det(\mathbf{F}^e(\mathbf{y}))\mathbf{F}^{e-T}(\mathbf{y})$ is the cofactor matrix of the element deformation gradient $\mathbf{F}^e(\mathbf{y}) = \sum_{j=0}^{N^N-1} \mathbf{y}_j \frac{\partial N_j^e}{\partial \mathbf{X}}$. We note that the cofactor matrix is defined for all deformation gradients $\mathbf{F}^e$, singular, inverted (negative determinant) or otherwise. This is essential for robustness to large deformation. We discuss the motivation for this simplification in Section 6.6, but note here that it is clearly positive definite since it is a scaled version of the identity with a rank-one update from the cofactor matrix (with positive $\lambda > 0$ scaling). With this convention, our symmetric positive definite modified nodal Hessian is of the form

$$A^{n+1}_{(k+1)i_k\alpha\beta} = \sum_{e=0}^{N^E-1} \sum_{\delta,\gamma=0}^{d-1} \tilde{C}^e_{\alpha\gamma\beta\delta}(\mathbf{x}^{n+1,l}_{(k)}) \frac{\partial N^e_{i_k}}{\partial X_\gamma} \frac{\partial N^e_{i_k}}{\partial X_\delta} V^0_e + \tag{6.15}$$

$$\sum_{c=0}^{N^{\mathrm{wc}}-1} \left(w^c_{0i_k} - w^c_{1i_k}\right)^2 K_{c\alpha\beta},\ 0 \le \alpha, \beta < d \tag{6.16}$$

6.4.2 Collision against kinematic bodies

We add support for hard collision constraints against kinematic geometry. At the beginning of each time step, each vertex $\mathbf{x}_i$ detects its closest point $\bar{\mathbf{x}}_i$ on the kinematic body. We use $\mathbf{n}_i$ to denote the unit outward normal to the collision body at the closest point. $\mathbf{x}_i$ is then classified as penetrating if $(\mathbf{x}_i - \bar{\mathbf{x}}_i) \cdot \mathbf{n}_i < 0$. For each penetrating $\mathbf{x}_i$, we project it to $\bar{\mathbf{x}}_i$

before the simulation. Then for each PBNG iteration, we check if $\Delta \mathbf{x}^{n,l}_{(k)i} \cdot \mathbf{n}_i < 0$. If so, we project $\Delta \mathbf{x}^{n,l}_{(k)i}$ to $\Delta \bar{\mathbf{x}}^{n,l}_{(k)i} = (\mathbf{I} - \mathbf{n}_i \times \mathbf{n}_i)\Delta \mathbf{x}^{n,l}_{(k)i}$.

6.4.3 SOR and Chebyshev Iteration

PBNG is remarkably stable and gives visually plausible results when the computational budget is limited, but it is also capable of producing numerically accurate results as the budget is increased. However, as shown in Figure 6.3 and as with most Gauss-Seidel approaches, the convergence rate of PBNG may decrease with iteration count. We investigated two simple acceleration techniques to help mitigate this: the Chebyshev semi-iterative method (as in [Wan15]) and SOR (as in [FVP16]). The Chebyshev method uses the update

$$\mathbf{x}^{n+1,l+1} = \omega_{l+1}(\gamma(\mathbf{x}^{n+1,l+1}_{\text{PBNG}} - \mathbf{x}^{n+1,l}) + \mathbf{x}^{n+1,l} - \mathbf{x}^{n+1,l-1}) + \mathbf{x}^{n+1,l-1} \tag{6.17}$$

where $\mathbf{x}^{n+1,l+1}$ denotes the accelerated update and $\mathbf{x}^{n+1,l+1}_{\text{PBNG}}$ denotes the standard PBNG update. Here $\omega_{l+1} = \frac{4}{4-\rho^2\omega_l}$ for $l > 2$, $\frac{2}{2-\rho^2}$ for $l = 2$ and 1 for $l < 2$. γ is an under-relaxation parameter that stabilizes the algorithm. For our examples, we set $\rho = .95$. PBNG is very stable, and this allows for the use of over-relaxation as well. We set $\gamma = 1.7$.

The SOR method uses a similar, but simpler update

$$\mathbf{x}^{n+1,l+1} = \omega(\mathbf{x}^{n+1,l+1}_{\text{PBNG}} - \mathbf{x}^{n+1,l-1}) + \mathbf{x}^{n+1,l-1}. \tag{6.18}$$

We use $\omega = 1.7$ for this under-relaxation parameter. As shown in Figure 6.3, Chebyshev and SOR behave similarly in terms of residual reduction and visual appearance.

6.5 Cloth Simulation

Our method can also naturally handle cloth simulation by adding a surface mesh contribution $\tilde{\text{PE}}(\mathbf{y})$ directly to the potential energy in Equation (2.17). We use the sum of a membrane hyperelastic potential and a bending term

$$\tilde{\mathrm{PE}}(\mathbf{y}) = \sum_{\hat{t}} \Psi^{\mathrm{cm}}(\mathbf{F}^{\mathrm{mem}}_{\hat{t}}(\mathbf{y}))A_{\hat{t}} + \frac{1}{2}\sum_{e} \theta_e(\mathbf{y})^2 A_e. \tag{6.19}$$

The membrane term is a simple generalization of the fixed corotated model [SHS12] to the case of surfaces

$$\begin{aligned}\Psi^{\mathrm{cm}}(\mathbf{F}^{\mathrm{mem}}) = \mu^{\mathrm{cm}}|\mathbf{F}^{\mathrm{mem}} - \mathbf{R}(\mathbf{F}^{\mathrm{mem}})|_F^2 + \\ \frac{\lambda^{\mathrm{cm}}}{2}(J(\mathbf{F}^{\mathrm{mem}}) - 1)^2.\end{aligned} \tag{6.20}$$

Here $\mathbf{F}^{\mathrm{mem}}_{\hat{t}}(\mathbf{y}) = \sum_i \mathbf{y}_i \frac{\partial \hat{N}_i}{\partial \mathbf{X}}(\mathbf{X}_{\hat{t}}) \in \mathbb{R}^{3\times 2}$ is the deformation gradient computed over the triangle $\hat{t}$ in practice where $\hat{N}_i$ are piecewise linear interpolating functions over the triangles. $A_{\hat{t}}$ is the reference area of the triangle $\hat{t}$ and $J(\mathbf{F}^{\mathrm{mem}}) = \sqrt{\det(\mathbf{F}^{\mathrm{mem}T}\mathbf{F}^{\mathrm{mem}})}$ measures the change in triangle area under motion defined by $\mathbf{y}$. $\mathbf{F} = \mathbf{R}(\mathbf{F})\mathbf{S}(\mathbf{F})$ is the polar decomposition of $\mathbf{F}$ with the convention that $\mathbf{R}(\mathbf{F}) \in \mathbb{R}^{3\times 2}$ has orthogonal columns and $\mathbf{S}(\mathbf{F}) \in \mathbb{R}^{2\times 2}$ is symmetric. For bending resistance we adopt a similar approach to Baraff and Witkin [BW98]. For each edge e with vertices $\mathbf{y}_e^0, \mathbf{y}_0^1$ that is incident to two triangles with unit normals $\mathbf{n}_e^1(\mathbf{y}), \mathbf{n}_e^2(\mathbf{y})$, we define $\theta_e(\mathbf{y}) \in [0, \pi)$ as the bending angle where $\theta_e(\mathbf{y}) = \mathrm{atan}(\frac{(\mathbf{n}_e^1(\mathbf{y})\times\mathbf{n}_e^2(\mathbf{y}))\cdot(\mathbf{y}_e^1 - \mathbf{y}_e^0)}{\mathbf{n}_e^1(\mathbf{y})\cdot\mathbf{n}_e^2(\mathbf{y})})$. We define an area to each edge A_e as one third of the sum of the areas of the two triangles incident to it (in the reference configuration of the triangles).

6.5.0.1 Modified Hessian

Similar to Section 6.4.1, we modify the Hessians of the above models to ensure semi-positive definiteness. We make the simple approximation

$$\frac{\partial^2 \Psi^{\mathrm{cm}}}{\partial F^{\mathrm{mem}}_{\alpha\gamma}\partial F^{\mathrm{mem}}_{\beta\delta}}(\mathbf{y}) \approx 2\mu^{\mathrm{cm}}\delta_{\alpha\beta}\delta_{\gamma\delta} + \frac{\lambda^{\mathrm{cm}}}{4}J^2 L^{\mathrm{mem}}_{\alpha\gamma}L^{\mathrm{mem}}_{\beta\delta} \tag{6.21}$$

$$\mathbf{L}^{\mathrm{mem}} = \mathbf{F}^{\mathrm{mem}}((\mathbf{F}^{\mathrm{mem}})^T\mathbf{F}^{\mathrm{mem}})^{-T} + \mathbf{F}^{\mathrm{mem}}((\mathbf{F}^{\mathrm{mem}})^T\mathbf{F}^{\mathrm{mem}})^{-1} \tag{6.22}$$

For the bending model we use the rank one approximation

$$\frac{1}{2}\frac{\partial^2 \theta_e^2}{\partial \mathbf{x}^2} \approx \frac{\partial \theta_e}{\partial \mathbf{x}} \otimes \frac{\partial \theta_e}{\partial \mathbf{x}}. \tag{6.23}$$

See Appendix B for more detail.

6.6 Lamé Coefficients

The parameters of an isotropic constitutive model are often based on Lamé coefficients μ and λ which are themselves set from Young's modulus E and Poisson's ratio ν according to Equation (2.9). This relationship is based on the assumption of linear dependence of stress on strain, or quadratic potential energy density

$$\Psi^{\mathrm{le}}(\mathbf{F}) = \mu \mathrm{tr}(\boldsymbol{\epsilon}^2(\mathbf{F})) + \frac{\lambda}{2}\mathrm{tr}(\boldsymbol{\epsilon}(\mathbf{F}))^2 \tag{6.24}$$

$$\boldsymbol{\epsilon} = \frac{1}{2}(\mathbf{F} + \mathbf{F}^T) - \mathbf{I}. \tag{6.25}$$

Furthermore, Equation (2.9) is derived from the model in Equation (6.24) by holding one end of a cuboidal domain fixed and applying a displacement at its opposite end. The remaining faces of the domain are assumed to be traction-free. Young's modulus is the scaling in a linear relationship between the traction exerted by the material in resistance to the displacement. The Poisson's ratio correlates with the degree of volume preservation via deformation in the directions orthogonal to the applied displacement.

The use of Lamé coefficients with nonlinear models is not directly analogous since the relation between displacement and traction is not a linear scaling in the cuboid example. When using Lamé coefficients with nonlinear problems, the cuboid derivation should hold if the model were linearized around $\mathbf{F} = \mathbf{I}$. All isotropic hyperelastic constitutive models can be written in terms of the isotropic invariants $I_\alpha : \mathbb{R}^{d\times d} \to \mathbb{R}$, $0 \leq \alpha < d$

$$I_0(\mathbf{F}) = \mathrm{tr}(\mathbf{F}^T\mathbf{F}),\ I_1(\mathbf{F}) = \mathrm{tr}((\mathbf{F}^T\mathbf{F})^2),\ I_2(\mathbf{F}) = \det(\mathbf{F}) \tag{6.26}$$

$$\Psi(\mathbf{F}) = \hat{\Psi}(I_0(\mathbf{F}), I_1(\mathbf{F}), I_2(\mathbf{F})). \tag{6.27}$$

See [GS08] for more detailed derivation. Note, when $d = 2$, $I_1(\mathbf{F}) = \mathrm{tr}((\mathbf{F}^T\mathbf{F})^2)$ is not used.

With this convention, the Hessian of the potential energy density is of the form

$$\frac{\partial^2 \Psi}{\partial \mathbf{F}^2} = \sum_{\alpha=0}^{d-1} \frac{\partial \hat{\Psi}}{\partial I_\alpha} \frac{\partial^2 I_\alpha}{\partial \mathbf{F}^2} + \sum_{\alpha,\beta=0}^{d-1} \frac{\partial^2 \hat{\Psi}}{\partial I_\alpha \partial I_\beta} \frac{\partial I_\alpha}{\partial \mathbf{F}} \otimes \frac{\partial I_\beta}{\partial \mathbf{F}}. \tag{6.28}$$

If Lamé parameters are to be used with a nonlinear model, the Hessian $\frac{\partial^2 \Psi}{\partial \mathbf{F}^2}(\mathbf{F})$ should match that of linear elasticity when evaluated at $\mathbf{F} = \mathbf{I}$. For example, this is why we adjust the Lamé parameters used in [MM21] in Equation (2.8). See Appendix B for derivation detail.

We choose our approximate Hessian in Equation (6.14) based on this fact. That is, by omitting all but the first and last terms in Equation (6.28), our approximate Hessian is both symmetric positive definite and consistent with any model that is set from Lamé coefficients (e.g. from Young's modulus and Poisson's ratio)

$$\tilde{\mathcal{C}} = \mu \frac{\partial^2 I_0}{\partial \mathbf{F}^2} + \lambda \frac{\partial I_{d-1}}{\partial \mathbf{F}} \otimes \frac{\partial I_{d-1}}{\partial \mathbf{F}}. \tag{6.29}$$

Again, see Appendix B for more detail.

6.7 Coloring and Parallelism

Parallel implementation of Gauss-Seidel techniques is complicated by data dependencies in the updates. This can be alleviated by careful ordering of sub-iterate position updates. We provide simple color-based orderings for both PBD and PBNG techniques. For PBD, colors are assigned to constraints so that those in the same color do not share incident nodes. Constraints in the same color can then be solved at the same time with no race conditions. For each vertex $\mathbf{x}_i$ in the mesh, we maintain a set $S_{\mathbf{x}_i}$ that stores the colors used by its incident constraints. For each constraint c, we find the minimal color as the least integer that is not contained in the set $\cup_{\mathbf{x}_i \in c} S_{\mathbf{x}_i}$. We then register the color by adding it into $S_{\mathbf{x}_i}$ for each $\mathbf{x}_i$ in constraint c. With PBNG, we color the nodes so that those in the same color do not share any mesh element or weak constraint. For each element or weak constraint c, we maintain a set S_c that stores the colors used by its incident nodes. For a position

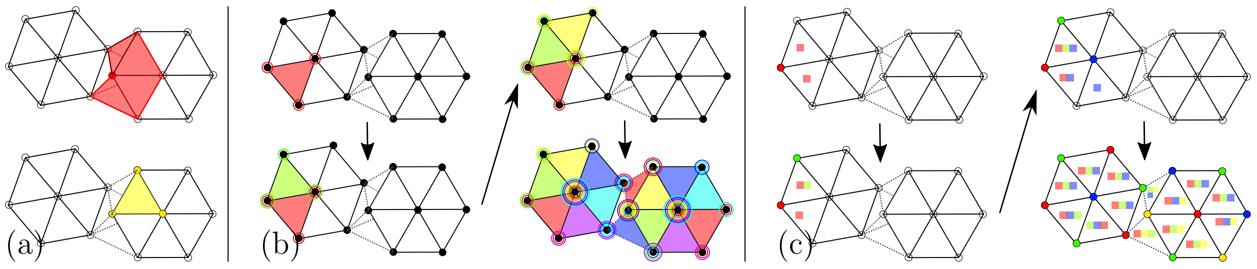


Figure 6.4: **(a) Dual Coloring** . Node based coloring (top) is contrasted with constraint based coloring (bottom). When a node is colored as red, its incident elements register red as used colors. When a constraint is colored yellow, its incident particles register yellow as used colors. **(b) Constraints-Based Coloring**. A step-by-step constraint mesh coloring scheme is shown. The dotted line indicates two weak constraints between the elements. The first constraint is colored red, all its incident points will register red as a used color. Other constraints incident to the first constraint have to choose other colors. **(c) Node-Based Coloring**. A step-by-step node coloring scheme is shown. The constraint register the colors used by its incident particles. The first particle is colored red, so all its incident constraints will register red as used. Other particles incident to the constraints have to choose other colors.

$\mathbf{x}_i$, we compute its color as the minimal one not contained in the set $\cup_{\mathbf{x}_i \in c} S_c$. Then we register the color by adding it into S_c for each element or weak constraint $\mathbf{x}_i$ is incident to. The coloring process is illustrated in Figures 6.4(b) and 6.4(c). We observe that coloring the nodes instead of the constraints gives fewer colors. This makes simulations run faster since more work can be done without race conditions. In Table 6.1, we demonstrate this performance observation. Note that we use the omp parallel directive from Intel's OpenMP library for parallelizing the updates.

Example	# Vertices	# Elements.	# Particle Colors	# Constraint Colors	PBNG Runtime/Iter	PBD Runtime/Iter
Res 32 Box Stretching	32K	150K	5	39	28ms	26.8ms
Muscles Without Collisions	284k	1097K	13	82	131ms	140ms
Res 64 Box Stretching	260K	1250K	5	39	65ms	137ms
Res 128 Box Stretching	2097K	10242K	5	40	1520ms	1080ms
Dropping Simple Shapes Into Box	256K	1069K	11	52	270ms	140ms
Res 16 Box Dropping	4.1K	17K	5	39	3.6ms	4.1ms

Table 6.1: Number of Colors Comparison: runtime is measured per iteration (averaged over the first 200 iterations). PBNG does more work per-iteration than PBD, but has comparable speed due to improved scaling resulting from a smaller number of colors.

6.7.1 Collision Coloring

For simulations with static weak constraints, the coloring is a one-time cost. Otherwise, the colors have to be updated every time the weak constraint structure changes, e.g. from self-collision (Figures 6.5 and 6.10). We propose a simple coloring scheme for this purpose: only nodes incident to the newly added weak constraints need recoloring. We first compute all nodes $\mathbf{x}_i^{\text{extra}}$ that are incident to newly added weak constraints. For each $\mathbf{x}_i^{\text{extra}}$, we compute the used color set $\cup_{\mathbf{x}_i^{\text{extra}} \in c} S_c$. We use the color of $\mathbf{x}_i^{\text{extra}}$ from the previous time step as an initial guess. If it already exists in the used color set, then we find the minimal color that is not used. This is generally of moderate cost, e.g. in the muscle examples with collisions (Figures 6.1, 1.2 and 6.5), our algorithm takes less than 680ms/frame for recoloring, while the actual simulation takes a total of 67s to run.

6.8 Examples

We demonstrate the versatility and robustness of PBNG with a number of representative simulations of quasistatic (and dynamic) hyperelasticity. Examples run with the corotated model (Equation (2.7)) use the algorithm from [GFJ16] for its accuracy and efficiency. All the examples use Poisson's ratio $\nu = 0.3$. We compare PBNG, PBD, XPBD, XPBD-QS

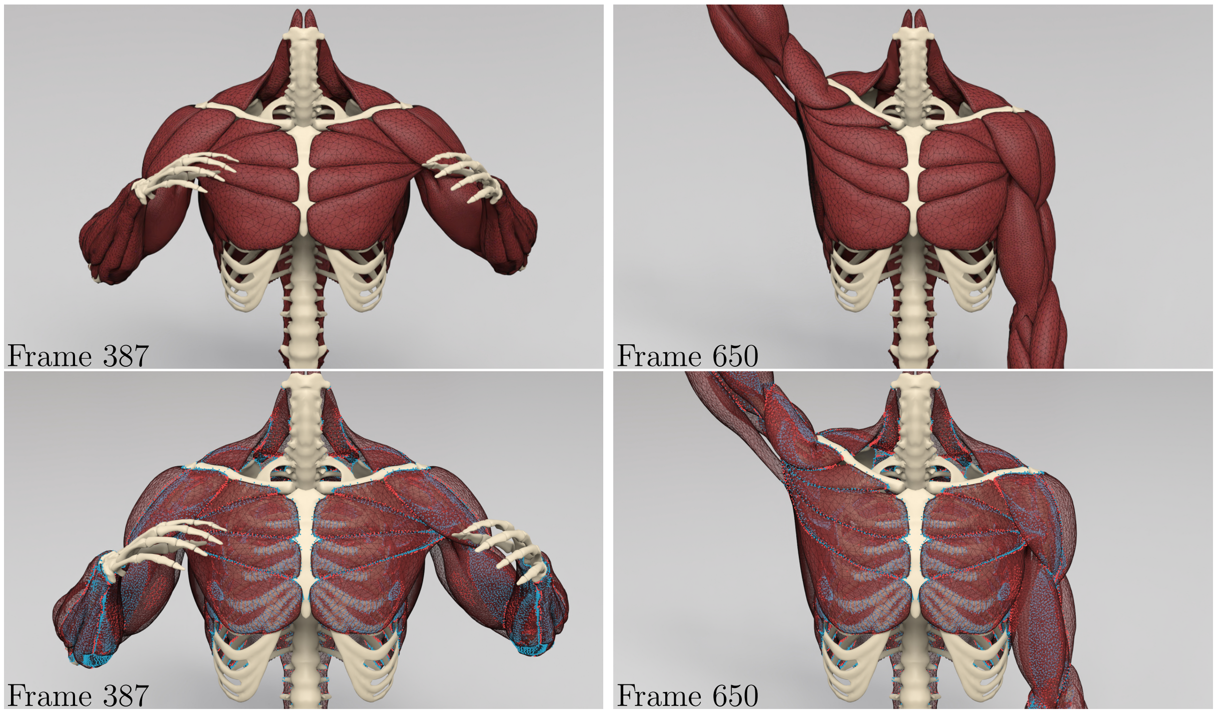


Figure 6.5: **PBNG Muscle Simulation**. The top row shows simulation results while the bottom row visualizes the vertex constraint status. *Red* indicates a vertex involved in contact, weak constraints are dynamically built to resolve the collisions. *Blue* represents the vertex positions of connective tissue bindings.

and XPBD-QS (Flipped). For XPBD-QS we do the hyperelastic constraints first, followed by weak constraints. For XPBD-QS (Flipped) the order is swapped. All the examples were run on an AMD Ryzen Threadripper PRO 3995WX CPU using 8 threads. In Table 6.4, we provide comprehensive performance statistics for PBNG. In Table 6.2, we provide runtime comparisons between PBNG and the other methods.

6.8.1 Stretching Block

We stretch and twist a simple block in a simple scenario. The block has 32K particles and 150K elements. Both ends of the block are clamped. They are stretched, squeezed and twisted in opposite directions. The block has $R^0 = 10kg/m^3$ and Young's modulus

Example	# Vertices	# Elements.	PBNG Runtime	Newton Runtime	PBD Runtime	PBNG # iter	PBD # iter	Newton # iter
Box Stretching (low budget)	32K	150K	170ms	170ms	170ms	6	6	2 (7 CGs)
Box Stretching (big budget)	32K	150K	1.3s	1.3s	1.3s	40	40	11 (10 CGs)
Muscle with collisions	284k	1097K	67s	430s	-	510	-	34 (200CGs)

Table 6.2: Methods Comparisons: We show runtime per frame for different methods for some of the examples. Each frame is run after advancing time .033.

$E = 10^5 Pa$. There is no gravity. The simulation is quasistatic. We compare performance between Newton's method, PBD, PBNG and XPBD as described in Section 6.3. In Figure 6.6, these methods are run under a fixed budget. Every method has a runtime of 1.3s/frame. With an ample budget, PBNG converges to ground truth, while PBD and XPBD do not. In Figure 6.6, we show a simulation where every method has a runtime of 170ms/frame. Newton's method is remarkably unstable. PBNG looks visually plausible. PBD and XPBD-QS have visual artifacts and fail to converge. Residual plots vs. time are shown at the bottom of Figure 6.6.

6.8.1.1 Resolution Comparison

In this example, we demonstrate PBNG's versatility by running the block stretching and twisting with various resolutions. As shown in Figure 6.7, the top block has 32K particles and 150K elements. The middle block has 260K particles and 1250K elements. The bottom block has 2097K particles and 10242K elements. Even at high-resolution (bottom block), PBNG is visually plausible after only 40 iterations and 61 seconds/frame of runtime.

6.8.1.2 Constitutive Model Comparison

In this example, we apply PBNG to various constitutive models on the same block examples. All three blocks have 32K particles and 150K elements. Frames are shown in Figure 2.1. The blocks from top to bottom are run with corotated (Equation 2.7), stable Neo-Hookean

(Equation 2.10) and Neo-Hookean (Equation 2.8) models respectively. With 40 iterations per frame, they are all visually plausible.

6.8.1.3 Comparison with Linear Gauss-Seidel

In this example, we show the superior performance of the nonlinear Gauss-Seidel strategy in PBNG against Newton's method with linear Gauss-Seidel used at each iteration. We compare on a representative block example with 32K particles and 150K elements. The simulation is run with both low iteration counts and a high iteration counts. Note that we match the iteration count instead of runtime, because computation of the explicit matrix and residual for linear Gauss Seidel once (620ms) already exceeds the total simulation cost of PBNG (170ms) in the low iteration count setting. For low iteration counts PBNG runs with 6 iterations/frame. Linear Gauss Seidel uses 2 Newton iterations with 3 Gauss Seidel iteration each. For high iteration counts PBNG runs with 42 iterations/frame and Newton + linear Gauss Seidel runs 6 Newton iterations with 7 Gauss Seidel iteration each.

We observe that PBNG has several advantages. First it only uses the diagonal blocks on the hessian with local solves, resulting in a much lower per iteration cost than Linear Gauss Seidel, as shown in Table 6.3. Also it does not have the overhead of computing the global Hessian and global residual, which are typically more costly than the entire simulation budget in real-time applications. Also PBNG achieves clearly superior nonlinear system residual reduction, as shown in Figure 6.8. Also, we observe that linear Gauss Seidel requires a smaller SOR ω because it is less stable than PBNG in practice. For this example $\omega = 1.3$ for linear Gauss Seidel and $\omega = 1.7$ for PBNG.

6.8.1.4 Approximate Hessian Comparison

In this example we demonstrate the efficacy of our Hessian approximation (Equation 6.14). All four blocks have 4K particles and 20K elements (see Figure 6.9). The top three bars are

Iteration Count	Newton Overhead	Linear GS Runtime/Iter	PBNG Runtime/Iter	Linear GS SOR	PBNG SOR	PBNG Runtime/Frame	Linear GS Runtime/Frame
6	620ms	35ms	27ms	1.3	1.7	170ms	1345ms
40	620ms	35ms	27ms	1.3	1.7	1080ms	5358ms

Table 6.3: Runtime Breakdown: We show runtime breakdown for linear Gauss Seidel and PBNG. Newton Overhead refers to the cost of computing Newton residual and explicit hessian every Newton iteration, a cost which PBNG does not require.

simulated using Newton's method with the exact hessian and different linear solvers. The top bar uses an exact solve (QR decomposition). The second bar uses an iterative solver (BICGSTAB since the true Hessian is not positive definite) and the third bar uses linear Gauss Seidel. The bottom bar is simulated using the approximate Hessian in Equation 6.14. All approaches using the exact hessian lead to unstable results, while our approximation leads to a correct converged result.

6.8.1.5 Acceleration Comparison

In this example, we compare the effects of the Chebyshev semi-iterative method and the SOR method. In Figure 6.3, we stretch and twist the same block with 32K particles and 150K elements. The top bar is simulated with plain PBNG. The middle bar is simulated with PBNG with Chebyshev semi-iterative method with $\gamma = 1.7, \rho = .95$. The bottom bar is simulated with PBNG with SOR and $\omega = 1.7$. 10 iterations are used for each time step. With a limited budget, plain PBNG is less converged than accelerated techniques. Figure 6.3 shows the convergence rate of the three methods vs. the number of iterations at the first time step. We can see that the acceleration techniques boost the convergence rate.

6.8.2 Collisions

We support collisions by dynamically adding weak constraints as discussed in Section 2.4. We use a time step of $\Delta t = .002s$ and detect collision every time step.

6.8.2.1 Two Blocks Colliding

We demonstrate the generation of dynamic weak constraints with a simple example. We take two blocks with one side fixed and drive them toward each other. This is a dynamic/backward Euler simulation. The blocks have $R^0 = 10kg/m^3$ and Young's modulus $E = 1000Pa$. The weak constraints have stiffness $k_n = 10^8$ and $k_\tau = 0$. The dynamic weak constraints are visualized in Figure 6.10 as red nodes in the mesh.

6.8.2.2 Muscles

We quasistatically simulate a large-scale musculature with collision and connective tissue weak constraints. The mesh has a total of 284K particles and 1097K elements. The muscles have $R^0 = 1000kg/m^3$, Young's modulus $E = 10^5 Pa$, connective tissue (blue) weak constraint stiffness is isotropic $k_n = k_\tau = 10^8$. Dynamic collision (red) weak constraint stiffness is anisotropic $k_n = 10^8$ and $k_\tau = 0$. We show several frames of muscles simulated with PBNG and dynamically generated weak constraints in Figure 6.5. PBNG takes 67 seconds to simulate a frame, while Newton's method takes 430s. In figure 6.1, we show that PBNG looks visually the same as Newton, while running 6-7 times faster. We also show that PBD and XPBD-QS fail to converge. In Figure 6.1, we show PBD becomes unstable. In Figure 1.2, we demonstrate sub-iteration order-dependent behavior with PBD. XPBD-QS has weak constraints processed last, which leads to excessive stretching of elements. XPBD-QS (Flipped) has weak constraints processed first, which degrades their enforcement and leaves a gap.

6.8.2.3 Dropping Objects

40 objects with simple shapes are dropped into a glass box. The objects have a total of 256K particles and 1069K elements. The simulation is run with dynamic/backward Euler. Some frames are shown in Figure 6.11. We show PBNG's capability of handling collision-intensive

scenarios. The example is run with $R^0 = 10kg/m^3$, Young's modulus $E = 3000Pa$ and weak constraint stiffness $k_n = 10^8$ and $k_\tau = 0$.

6.8.2.4 Dropping Armadillos

We showcase the capability of PBNG with a simulation coupling cloth with solids. In this example 20 armadillos are dropped onto a piece of cloth with four corners held fixed. Frames are shown at 6.12. After $3.33s$ one end of the cloth breaks free and armadillos drop into a glass box. Each armadillo has 17K vertices and 66K particles. The rectangular cloth has 4K particles and 8K triangles. We set $\Delta t = 0.004s, R^0 = 10kg/m^3, E = 1000Pa$. For the rectangular cloth we set $R^0 = R^0 = 10kg/m^2, E^{\text{cod}} = 10000Pa, k_b = .05$. We set Poisson ratio $\nu = 0.3$ for all objects and $k_n = 10^8$ for weak constraints. The average runtime is 101.2s/frame.

6.8.3 Varying Stiffness

In this example, we demonstrate that XPBD-QS fails to resolve quasistatic problems with varied stiffness. In Figure 6.13, we show the initial setup for the simulation. The simulation is quasistatic. Both block meshes have $R^0 = 10kg/m^3$ and Young's modulus $E = 1000Pa$. The first block mesh has its top boundary constrained. The second block is weakly constrained to the first block via weak constraints between them. The springs have stiffness $k_n = k_\tau = 10^8$. There is gravity in the scene with acceleration -9.8 in the $y-$direction. As we show in Figure 6.13, PBNG converges to a plausible state. XPBD-QS and XPBD-QS (Flipped) fail to converge. Depending on the order of the constraints, it either leaves a gap between the two blocks or a very stretched top layer of the bottom block. This example also serves as a simplified version of the connective bindings on the muscles, which are used in Figure 1.2. The residual plot is shown on the right of Figure 6.13.

Example	# Vertices	# Elements	# Triangles	PBNG Runtime / Frame	PBNG # Iter/Frame	# Substeps	Model
Box Stretching (low budget)	32K	150K	0	170ms	6	1	Corotated
Box Stretching (big budget)	32K	150K	0	1300ms	40	1	Corotated
Muscle with collisions	284k	1097K	0	67000ms	510	17	Corotated
Res 64 Box Stretching	260K	1250K	0	1300ms	20	1	Corotated
Res 128 Box Stretching	2097K	10242K	0	61000ms	40	1	Corotated
Dropping Simple Shapes Into Box	256K	1069K	0	49800ms	136	17	Corotated
Two moving blocks colliding	8.2K	33K	0	1630ms	136	17	Corotated
Box Stretching	32K	150K	0	1300ms	40	1	Stable Neo-Hookean
Box Stretching	32K	150K	0	825ms	40	1	Neo-Hookean
Armadillos Dropping	344K	1320K	8K	101200ms	360	9	Corotated

Table 6.4: Performance Table of PBNG: runtime is measured for each frame (averaged over the course of the simulation). Each frame is written after advancing time .033.

6.8.4 PBD

In this example, we show how PBD eliminates the effects of external forcing as the number of iterations increases. We clamp the left side of a simple bar mesh. We run a quasistatic simulation with gravity (acceleration $-9.8m/s^2$ in the $y-$direction). The bar has $R^0 = 10kg/m^3$ and Young's modulus $E = 1000Pa$. As shown in Figure 6.14, PBD converges to a rigid bar configuration. PBNG converges to a plausible solution. XPBD-QS appears to resolve the issues with PBD and quasistatics. However, XPBD-QS with 10 iterations per pseudo-time step appears more converged than XPBD-QS with 1 iteration per pseudo-time step.

6.8.5 XPBD

We run a simple dynamics example to show that XPBD does cannot reduce the backward Euler system residual in practice, as discussed in Chen et al. [CHC23]. We take a simple block with the left side clamped. It falls under gravity and oscillates. The simulation scene is shown on the top of Figure 6.2. The block has 4.1K particles and 17K elements. This simple but representative example demonstrates superior convergence behavior of PBNG over XPBD.

6.8.6 PBNG vs. PBD and Limited Newton

We run a simple quasistatic example to illustrate the convergence propagation behavior of PBNG compared to each conjugate gradient (CG) iteration in Newton's method as well as PBD. In Figure 6.15, a block has its two sides stretched and then clamped. We compute the quasistatic equilibrium using Newton's method with 1 Newton iteration, PBD and PBNG. PBD does not converge to the right solution. After 50 iterations, PBNG looks visually plausible, but Newton's method is visually not converged. The residual plots are presented in Figure 6.15. PBNG iterations are comparable to CG iterations in Newton's method, but they have more favorable deformation propagation behavior.

6.9 Discussion and Limitations

We show that a node-based Gauss-Seidel approach for the nonlinear equations of quasistatic and backward Euler time stepping has remarkably stable behavior. While we generate visually plausible behaviors with restricted computational budgets in a manner that surpasses the PBD and XPBD state-of-the-art for quasistatic problems, our approach (even with Chebyshev and SOR acceleration) will still lose (in terms of numerical residual reduction) to a standard Newton's method when the computational budget is expanded. The MPBNG approach reduces number of iterations, but each iteration is more expensive than the original PBNG iteration. One may explore optimizations of MPBNG to make it more efficient. A multigrid or domain decomposition approach could be combined with our approach to address this in future work.

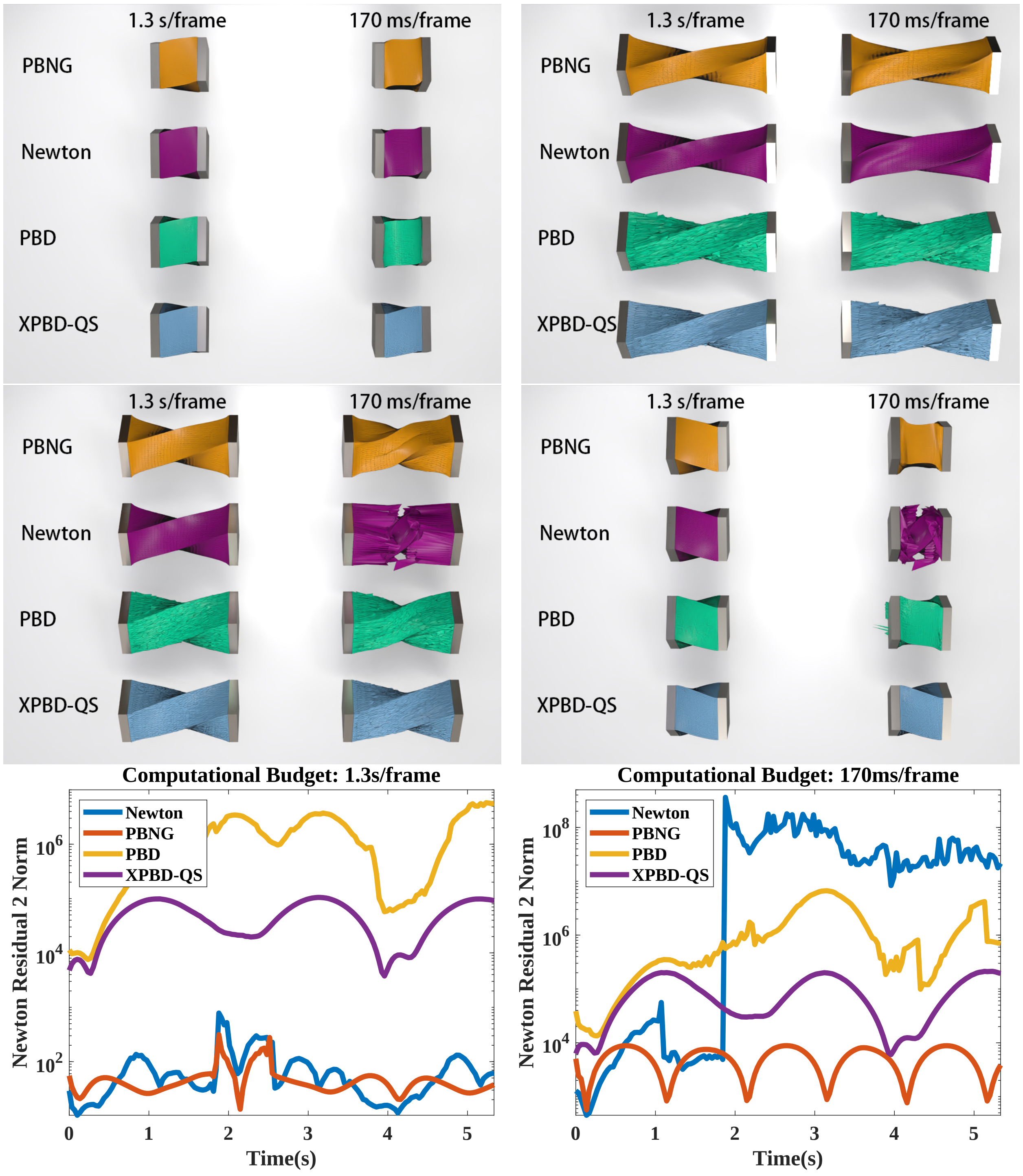


Figure 6.6: **Comparisons with Different Computational Budget**. A block is stretched/compressed while being twisted. With a sufficiently large computational budget, Newton's method is stable, but it becomes unstable when the computational budget is small. PBD and XPBD-QS do not significantly reduce the residual in the given computational time, resulting in noisy artifacts on the mesh. PBNG maintains relatively small residuals and generates visually plausible results of the deformable block even if the budget is limited.

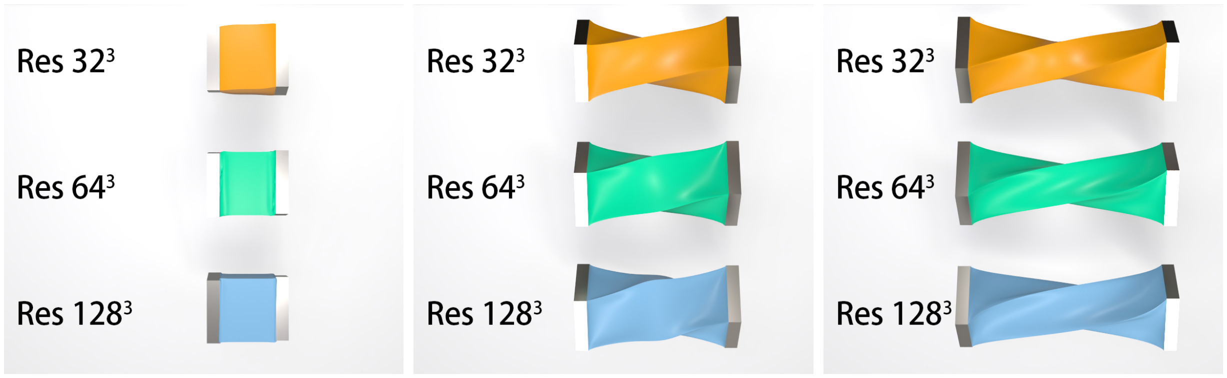


Figure 6.7: **Different Mesh Resolution**. PBNG produces consistent results when the mesh is spatially refined. The highest resolution mesh in this comparison has over 2M vertices and only requires 40 iterations to produce visually plausible results.

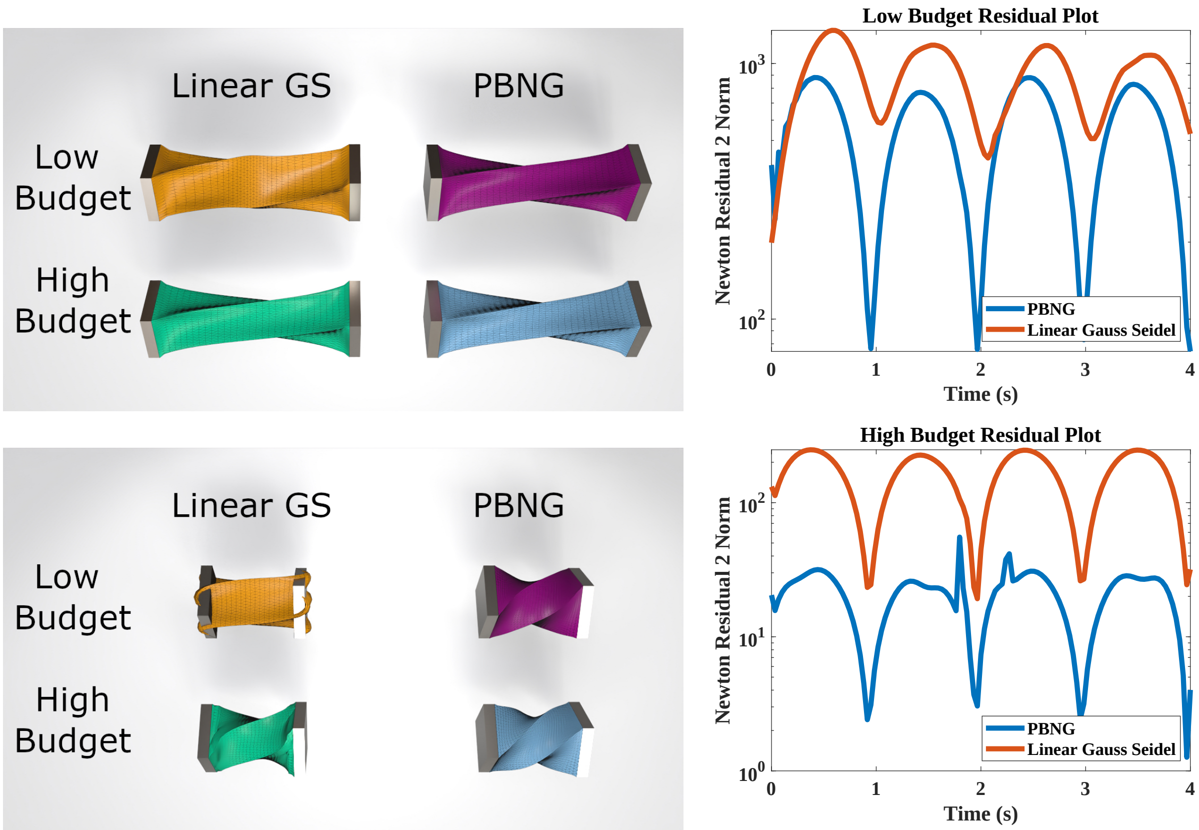


Figure 6.8: **Linear Gauss Seidel vs. PBNG**. The fact that PBNG relinearizes every

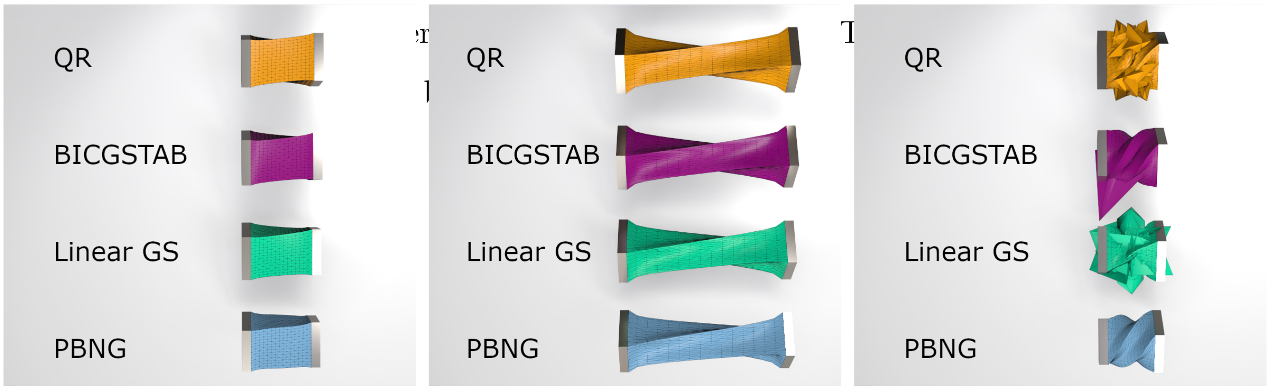


Figure 6.9: **Hessian Comparison**. The top three bars are simulated using Newton's method with different linear solvers (QR, BICGSTAB and linear Gauss Seidel respectively). The bottom bar is simulated using PBNG. The top bar uses the exact hessian and becomes unstable. The bottom bar uses our hessian projection and stays stable.

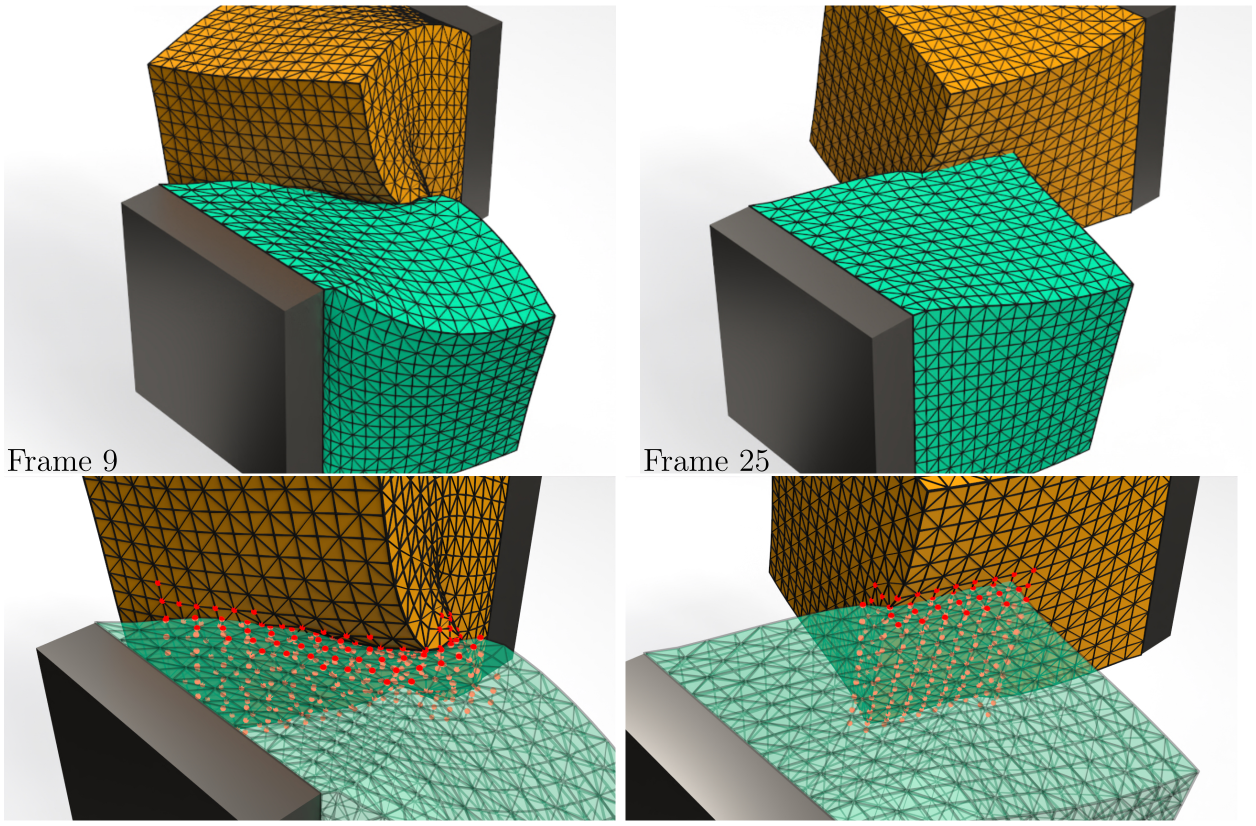


Figure 6.10: **Two Blocks Colliding**. Two blocks collide with each other with one face clamped. Red particles indicate that dynamic weak constraints have been built to resolve the collision of corresponding mesh vertices.

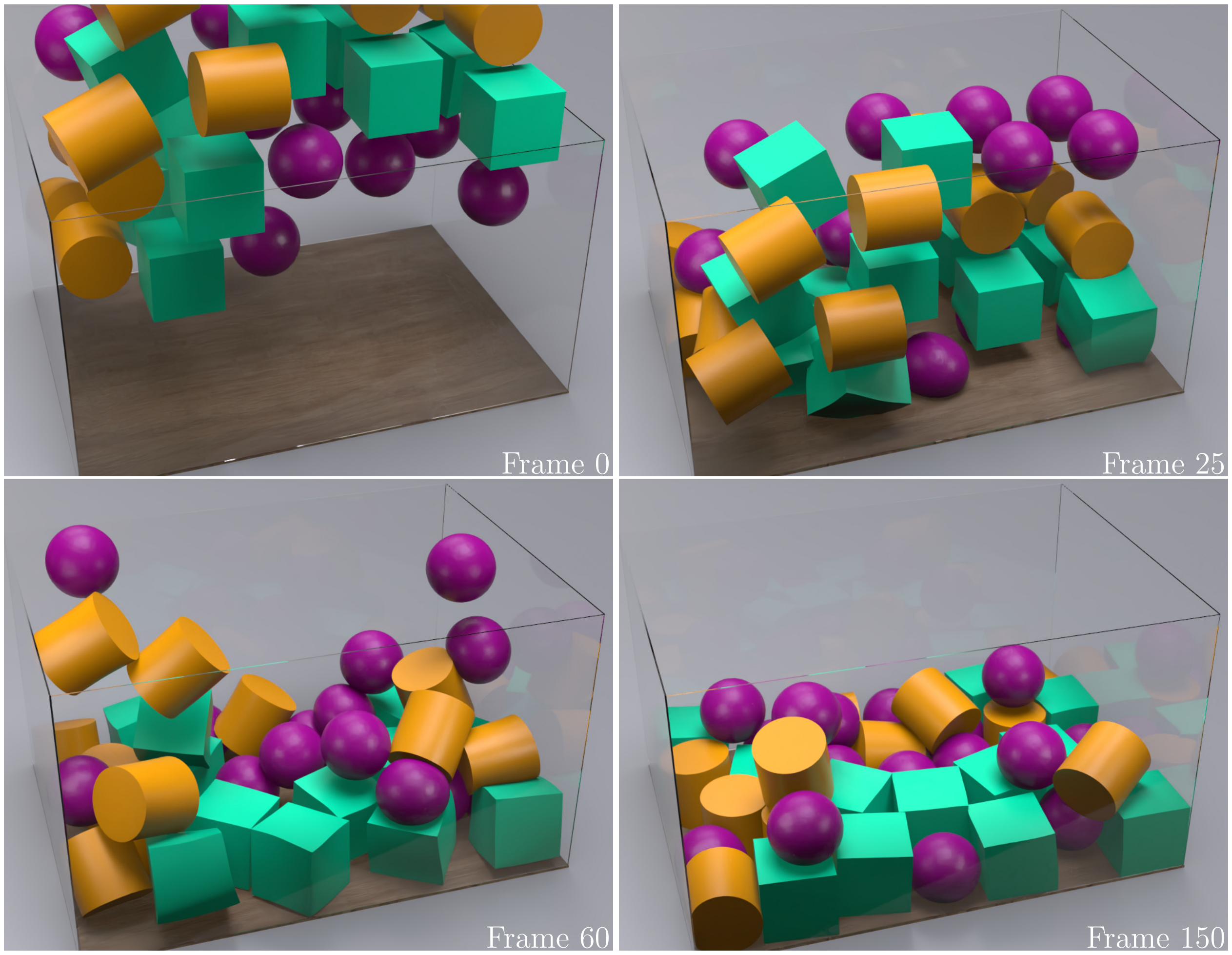


Figure 6.11: **Objects Dropping**. A variety of objects drop under gravity. Our method is able to robustly handle collisions between deformable objects through weak constraints.

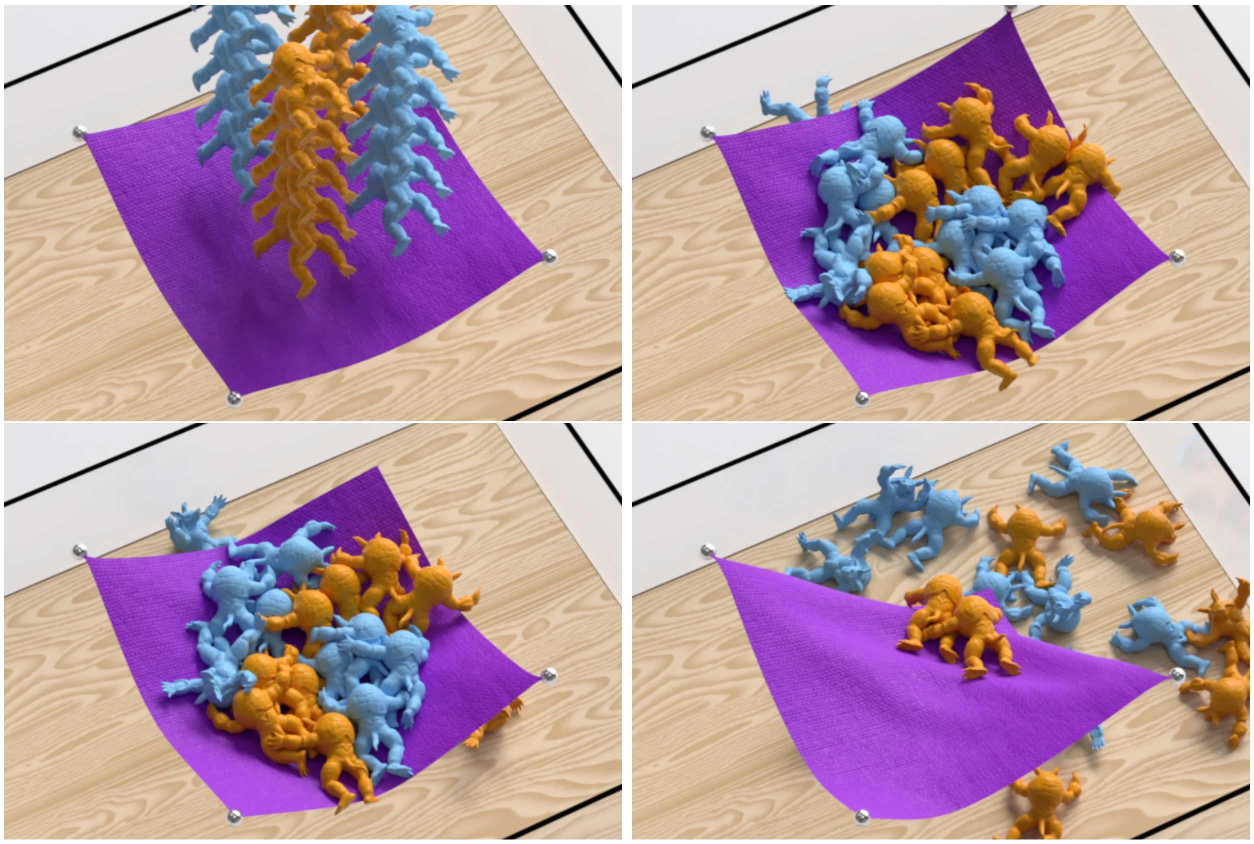


Figure 6.12: Armadillos Dropping: 20 armadillos drop onto a rectangular cloth. 3.33s later one end of cloth breaks loose and the armadillos fall into a glass box. Frame 1, 50, 102, 150 are shown.

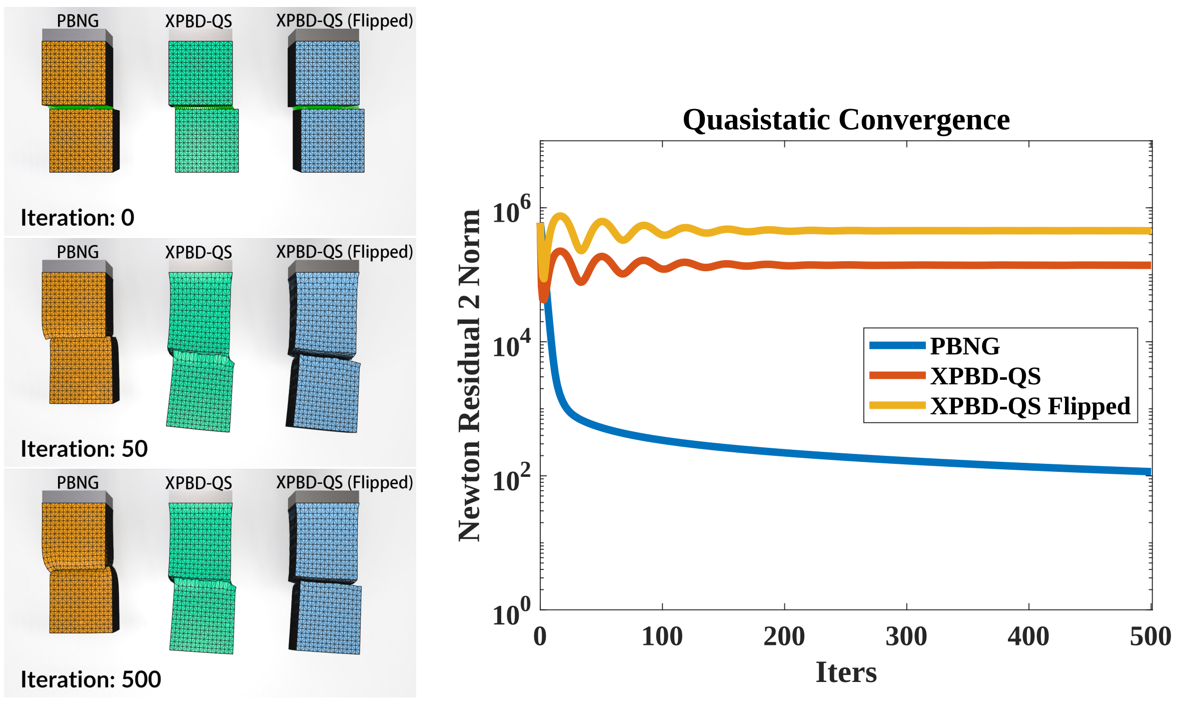


Figure 6.13: **Two Blocks Hanging**. Two identical blocks are bound together through weak constraints. Green line segments in iteration 0 indciate weak constraint springs. PBNG is able to reduce the residual by a few orders of magnitude and converges quickly. XPBD-QS methods demonstrate iteration-order-dependent behavior. Residuals oscillate and produce visually incorrect results.

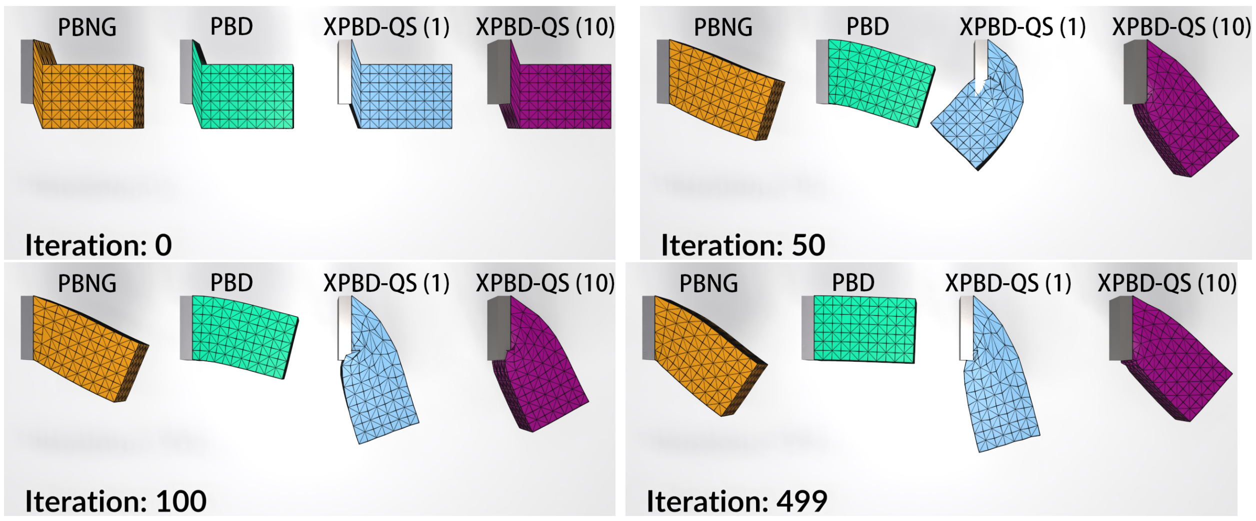


Figure 6.14: **Bar under Gravity**. A quasistatic simulation of a bar bending under gravity using different methods. The effect of external forcing vanishes in the PBD example as the number of iterations increases. More local iterations of XPBD-QS produces better results. PBNG converges to visually plausible results within fewer iterations than XPBD-QS.

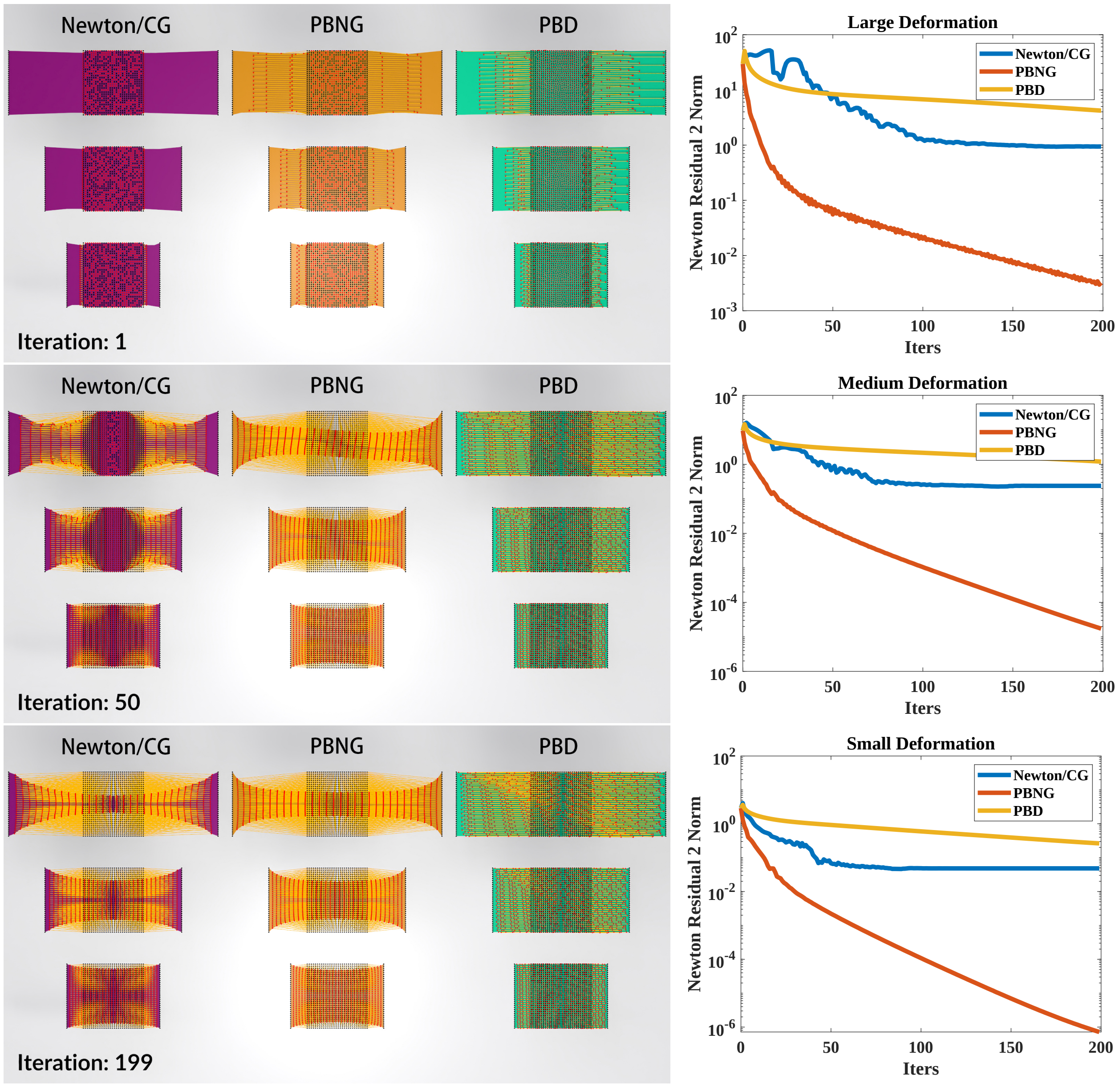


Figure 6.15: **Deformation Propagation Visualization**. A square block is initially stretched on its sides. **Left column**: visual results of the blocks after certain iterations. Black points are the initial positions. Red points are positions at the current iteration. Yellow line segments indicate the displacement of each node. Each method is color coded - purple is Newton, orange is PBNG, and green is PBD. Each row shows the results of large, medium, and small deformations respectively. PBNG converges to a visually plausible result in fewer iterations than one Newton step with increasing CG iterations. PBD fails to shrink in the transverse direction. **Right column**: 2-norm of the Newton residual vector. PBNG outperforms Newton's method and PBD.

APPENDIX A

Torque Equilibrium

We denote the spatial domain of each bone as $\Omega_b \subset \mathbb{R}^3$ and each individual bone mesh consists of locations $\mathbf{x} \in \Omega_b$ sampled in the bone domains. We denote the spatial domain consisting of all the bones in the skeleton as union $\Omega = \cup_b \Omega_b$. For any $\mathbf{x} \in \Omega_b$, a top-down joint rotation hierarchy $\{j_0, \ldots, j_i, \ldots, j_b\}$ originating from a root bone Ω_0 (the sternum/rib cage in our examples) defining the articulation kinematics on bone Ω_b is known. The articulated position $\boldsymbol{\phi}^b(\mathbf{x};\mathbf{q}) \in \mathbb{R}^3$ of $\mathbf{x} \in \Omega_b$ is composed of a combination of joint transforms $\mathbf{L}^{j_i}(\cdot; q_{j_i})$ defined on each joint rotation j_i as

$$\begin{aligned}
\boldsymbol{\phi}^b(\mathbf{x};\mathbf{q}) &= \mathbf{L}^{j_0}(\mathbf{L}^{j_1}(\mathbf{L}^{j_2}(\ldots); q_{j_1}); q_{j_0}) \\
\mathbf{L}^{j_i}(\mathbf{x}; q_{j_i}) &= \mathbf{R}^{j_i}(q_{j_i})(\mathbf{x} - \mathbf{x}^{j_i}) + \mathbf{x}^{j_i} \\
\mathbf{R}^j(q_j) &= \mathbf{U}^j \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(q_j) & -\sin(q_j) \\ 0 & \sin(q_j) & \cos(q_j) \end{pmatrix} \mathbf{U}^{j^T} \\
\mathbf{U}^j &= \begin{pmatrix} u_0^j, v_0^j, w_0^j \\ u_1^j, v_1^j, w_1^j \\ u_2^j, v_2^j, w_2^j \end{pmatrix} \in \mathbb{R}^{3\times 3}.
\end{aligned}$$

Here $\mathbf{R}^j$ is a rotation matrix of local joint rotation q_j radians around the associated pivot $\mathbf{x}^j$. $\mathbf{U}^j$ is an orthogonal matrix representing the rotation axis $\mathbf{u}^j \in \mathbb{R}^3$. Note that the vector $\mathbf{q} \in \mathbb{R}^{N_C}$ is made up of the components q_j. The Jacobian of the skeletal kinematics $\mathbf{J}^b(\mathbf{x};\mathbf{q}) \in \mathbb{R}^{3\times N_C}$ with respect to the joint state $\mathbf{q}$ is defined as $\mathbf{J}^b(\mathbf{x};\mathbf{q}) = \frac{\partial \boldsymbol{\phi}^b}{\partial \mathbf{q}}(\mathbf{x};\mathbf{q})$.

The principle of virtual work reveals the joint torques required to maintain the pose $\mathbf{q}$ in the presence of external forcing

$$\begin{aligned}\delta W = \int_\Omega \mathbf{f}(\mathbf{x}) \cdot \delta\boldsymbol{\phi}^b(\mathbf{x};\mathbf{q})d\mathbf{x} &= \int_\Omega \mathbf{f}(\mathbf{x}) \cdot (\mathbf{J}^b(\mathbf{x};\mathbf{q})\delta\mathbf{q})d\mathbf{x} \\ &= \delta\mathbf{q}^T \int_\Omega \mathbf{J}^{b^T}(\mathbf{x};\mathbf{q})\mathbf{f}(\mathbf{x})d\mathbf{x} = 0.\end{aligned}$$

Here $\delta\mathbf{q}$ is an arbitrary perturbation in the joint state. Intuitively, this states that the residual of the external and muscle forcing $\mathbf{f}(\mathbf{x}) = \rho(\mathbf{x})\mathbf{g} + \mathbf{f}^{\mathrm{m}}(\mathbf{x})$ can only be non-zero in components orthogonal to the articulation. This yields the torque constraints

$$\int_\Omega \sum_{\alpha=0}^{d-1} J^b_{\alpha j}(\mathbf{x};\mathbf{q}) f_\alpha(\mathbf{x}) d\mathbf{x} = 0 \tag{A.1}$$

where $J^b_{\alpha j}$ and f_α are the components of the Jacobian and force respectively. Also, $\hat{\Omega}_j \subset \Omega$ is defined to be all bodies affected by articulation of joint j.

After applying the chain rule in the Jacobian derivative, $\mathbf{J}^b{}_{:,j_i}$, the j_i^{th} column of the Jacobian, becomes:

$$\begin{aligned}\mathbf{J}^b{}_{:,j_i}(\mathbf{x};\mathbf{q}) = \frac{\partial\boldsymbol{\phi}^b}{\partial q_{j_i}}(\mathbf{x};\mathbf{q}) &= \frac{\partial\mathbf{L}^{j_0}}{\partial q_{j_0}}\frac{\partial q_{j_0}}{\partial q_{j_i}} + \frac{\partial\mathbf{L}^{j_0}}{\partial\mathbf{L}^{j_1}}\frac{\partial\mathbf{L}^{j_1}}{\partial q_{j_i}} \\ &= \delta_{j_i,j_0}\mathbf{R}^{j_0\prime}(q_{j_0})(\mathbf{L}^{j_1} - \mathbf{x}^{j_0}) + \mathbf{R}^{j_0}(q_{j_0})\frac{\partial\mathbf{L}^{j_1}}{\partial q_{j_i}} \\ &= \mathbf{R}^{j_0}(q_{j_0})\ldots\mathbf{R}^{j_{i-1}}(q_{j_{i-1}})\mathbf{R}^{j_i\prime}(q_{j_i})(\mathbf{L}^{j_{i+1}} - \mathbf{x}^{j_i})\end{aligned}$$

The key simplification is to consider the current state as the rest state $\mathbf{x} = \boldsymbol{\phi}^b(\mathbf{x};\mathbf{0}) = \mathbf{L}^{j_0} = \ldots = \mathbf{L}^{j_i}$, and $\mathbf{R}^{j_i}(0) = \mathbf{I}$ is the identity matrix.

$$\frac{\partial\boldsymbol{\phi}^b}{\partial q_{j_i}}(\mathbf{x};\mathbf{0}) = \mathbf{U}^{j_i}\begin{pmatrix}0 & 0 & 0\\ 0 & 0 & -1\\ 0 & 1 & 0\end{pmatrix}\mathbf{U}^{j_i T}(\mathbf{x} - \mathbf{x}^{j_i})$$

$$\frac{\partial\boldsymbol{\phi}^b_\alpha}{\partial q_{j_i}}(\mathbf{x};\mathbf{0}) = -\sum_{j,k,l=0}^{d-1} U^{j_i}_{\alpha j}\epsilon_{0jk}U^{j_i}{}_{lk}(x_l - x^{j_i}_l)$$

$$= \sum_{l=0}^{d-1}(w^{j_i}_\alpha v^{j_i}_l - v^{j_i}_\alpha w^{j_i}_l)(x_l - x^{j_i}_l) = -\sum_{p,l=0}^{d-1}\epsilon_{p\alpha l}u^{j_i}_p(x_l - x^{j_i}_l)$$

where ϵ is the Levi-Civita symbol. Note we use the property $\mathbf{u}^{j_i} = \mathbf{v}^{j_i} \times \mathbf{w}^{j_i}$ from the orthogonal matrix. Define $\hat{\Omega}_j \subset \Omega$ to be all bodies articulated by joint state q_j. Equation (A.1) becomes:

$$\begin{aligned}
\int_\Omega \sum_{\alpha=0}^{d-1} J^b_{\alpha j}(\mathbf{x};\mathbf{q}) f_\alpha(\mathbf{x}) d\mathbf{x} &= \int_{\hat{\Omega}_j} - \sum_{\alpha,p,l=0}^{d-1} \epsilon_{p\alpha l} u^j_p (x_l - x^{j_i}_l) f_\alpha(\mathbf{x}) d\mathbf{x} \\
&= \int_{\hat{\Omega}_j} \sum_{\alpha,p,l=0}^{d-1} \epsilon_{pl\alpha} (x_l - x^{j_i}_l) f_\alpha(\mathbf{x}) u^j_p d\mathbf{x} \\
&= \int_{\hat{\Omega}_j} ((\mathbf{x} - \mathbf{x}^j) \times \mathbf{f}(\mathbf{x})) \cdot \mathbf{u}^j d\mathbf{x} = 0
\end{aligned} \tag{A.2}$$

For further simplification of ball-and-socket joints, we can take $\mathbf{U}_j = \mathbf{I}$ and Equation (A.2) is simply

$$\int_{\hat{\Omega}_j} (\mathbf{x} - \mathbf{x}^j) \times \mathbf{f}(\mathbf{x}) d\mathbf{x} = \mathbf{0}$$

Knowing the set of muscles, we only find equilibrium on the joint articulations that can be controlled by the muscles. The remaining articulations, e.g. root and end joints, are implicitly in equilibrium.

APPENDIX B

Lamé Coefficients, Linear Elasticity and Hyperelasticity

B.1 Linear Elasticity

B.1.1 Potential

$$\Psi^{\mathrm{le}}(\mathbf{F}) = \mu \boldsymbol{\epsilon}(\mathbf{F}) : \boldsymbol{\epsilon}(\mathbf{F}) + \frac{\lambda}{2}\mathrm{tr}(\boldsymbol{\epsilon}(\mathbf{F}))^2 \tag{B.1}$$

$$\boldsymbol{\epsilon}(\mathbf{F}) = \frac{1}{2}\left(\mathbf{F} + \mathbf{F}^T\right) - \mathbf{I} \tag{B.2}$$

B.1.2 First-Piola-Kirchhoff Stress

$$\mathbf{P}^{\mathrm{le}}(\mathbf{F}) = \frac{\partial \Psi^{\mathrm{le}}}{\partial \mathbf{F}}(\mathbf{F}) = 2\mu\boldsymbol{\epsilon}(\mathbf{F}) + \lambda \mathrm{tr}(\boldsymbol{\epsilon}(\mathbf{F}))\mathbf{I} \tag{B.3}$$

B.1.3 Hessian

$$\frac{\partial^2 \Psi^{\mathrm{le}}}{\partial \mathbf{F}^2}(\mathbf{F}) = 2\mu \frac{\partial \boldsymbol{\epsilon}}{\partial \mathbf{F}}(\mathbf{F}) + \lambda \mathbf{I} \otimes \mathbf{I}. \tag{B.4}$$

The entries in $\frac{\partial \boldsymbol{\epsilon}}{\partial \mathbf{F}}(\mathbf{F})$ are given by $\frac{\partial \epsilon_{\alpha\beta}}{\partial F_{\gamma\delta}} = \frac{1}{2}\left(\delta_{\alpha\gamma}\delta_{\beta\delta} + \delta_{\beta\gamma}\delta_{\alpha\delta}\right)$. When viewed as a matrix, the Hessian has entries

$\frac{\partial^2 \Psi^{\text{le}}}{\partial F_{\sigma\tau} \partial F_{\delta\epsilon}}(\mathbf{I})$	00	11	22	01	10	12	21	02	20
00	$2\mu+\lambda$	λ	λ						
11	λ	$2\mu+\lambda$	λ						
22	λ	λ	$2\mu+\lambda$						
01				μ	μ				
10				μ	μ				
12						μ	μ		
21						μ	μ		
02								μ	μ
20								μ	μ

B.1.4 General Isotropic Elasticity Modified Hessian

We use the modified Hessian

$$\tilde{\mathcal{C}}(\mathbf{F}) = \mu \frac{\partial^2 I_0}{\partial \mathbf{F}^2} + \lambda \frac{\partial I_{d-1}}{\partial \mathbf{F}} \otimes \frac{\partial I_{d-1}}{\partial \mathbf{F}}. \tag{B.5}$$

where $I_0(\mathbf{F}) = \mathbf{F} : \mathbf{F}$ and $I_{d-1}(\mathbf{F}) = \det(\mathbf{F})$. $\frac{\partial^2 I_0}{\partial \mathbf{F}^2}$ is the twice the identity. Furthermore, when $\mathbf{F} = \mathbf{I}$, we get $\tilde{\mathcal{C}}(\mathbf{I})$ has entries

$\tilde{\mathcal{C}}_{\alpha\beta\gamma\delta}(\mathbf{I})$	00	11	22	01	10	12	21	02	20
00	$2\mu+\lambda$	λ	λ						
11	λ	$2\mu+\lambda$	λ						
22	λ	λ	$2\mu+\lambda$						
01				2μ	0				
10				0	2μ				
12						2μ	0		
21						0	2μ		
02								2μ	0
20								0	2μ

While this is not exactly equal to the Hessian of the potential for linear elasticity, the bottom three 2×2 blocks have the same eigenvalues as in the linear elasticity Hessian, where the 2μ mode is repeated and the null mode for the linear elasticity Hessian associated with linear rotations are removed. We keep this simplification since it maintains the meaning of the Lamé coefficients and since we found it to work as a modified Hessian in practice.

B.2 Neo-Hookean

B.2.1 Neo-Hookean Potential

$$\Psi(\mathbf{F}) = \frac{\mu}{2}\mathbf{F} : \mathbf{F} + \frac{\hat{\lambda}}{2}(\det(\mathbf{F}) - 1 - \frac{\mu}{\hat{\lambda}})^2 \tag{B.6}$$

B.2.2 First-Piola-Kirchhoff Stress

$$\mathbf{P}(\mathbf{F}) = \mu\mathbf{F} + \hat{\lambda}(\det(\mathbf{F}) - 1 - \frac{\mu}{\hat{\lambda}})J\mathbf{F}^{-T} \tag{B.7}$$

B.2.3 Hessian

$$\frac{\partial^2\Psi}{\partial\mathbf{F}}(\mathbf{F}) = \mu\mathbf{I} + \hat{\lambda}J\mathbf{F}^{-T} \otimes J\mathbf{F}^{-T} + \hat{\lambda}(\det(\mathbf{F}) - 1 - \frac{\mu}{\hat{\lambda}})\frac{\partial^2 J}{\partial\mathbf{F}^2}(\mathbf{F}) \tag{B.8}$$

B.2.3.1 Determinant Hessian

The determinant can be written in terms of the permutation tensor $\tilde{\boldsymbol{\epsilon}}_{\alpha\beta\gamma}$ as

$$J = \det(\mathbf{F}) = \tilde{\boldsymbol{\epsilon}}_{\alpha\beta\gamma} F_{0\alpha} F_{1\beta} F_{1\gamma} \tag{B.9}$$

$$\frac{\partial J}{\partial F_{\delta\boldsymbol{\epsilon}}}(\mathbf{F}) = J F^{-1}_{\boldsymbol{\epsilon}\delta} \tag{B.10}$$

$$= \tilde{\boldsymbol{\epsilon}}_{\boldsymbol{\epsilon}\beta\gamma}\delta_{0\delta}F_{1\beta}F_{2\gamma} + \tilde{\boldsymbol{\epsilon}}_{\alpha\epsilon\gamma}\delta_{1\delta}F_{0\alpha}F_{2\gamma} + \tilde{\boldsymbol{\epsilon}}_{\alpha\beta\epsilon}\delta_{2\delta}F_{0\alpha}F_{1\beta} \tag{B.11}$$

$$\frac{\partial^2 J}{\partial F_{\sigma\tau}\partial F_{\delta\boldsymbol{\epsilon}}}(\mathbf{F}) = \tilde{\boldsymbol{\epsilon}}_{\boldsymbol{\epsilon}\tau\gamma}\delta_{0\delta}\delta_{1\sigma}F_{2\gamma} + \tilde{\boldsymbol{\epsilon}}_{\tau\boldsymbol{\epsilon}\gamma}\delta_{0\sigma}\delta_{1\delta}F_{2\gamma} + \tilde{\boldsymbol{\epsilon}}_{\tau\beta\epsilon}\delta_{0\sigma}\delta_{2\delta}F_{1\beta} + \tag{B.12}$$

$$\tilde{\boldsymbol{\epsilon}}_{\epsilon\beta\tau}\delta_{0\delta}\delta_{2\sigma}F_{1\beta} + \tilde{\boldsymbol{\epsilon}}_{\alpha\boldsymbol{\epsilon}\tau}\delta_{1\delta}\delta_{2\sigma}F_{0\alpha} + \tilde{\boldsymbol{\epsilon}}_{\alpha\tau\boldsymbol{\epsilon}}\delta_{1\sigma}\delta_{2\delta}F_{0\alpha}. \tag{B.13}$$

The determinant Hessian evaluated at $\mathbf{F} = \mathbf{I}$ is

$\frac{\partial^2 J}{\partial F_{\sigma\tau}\partial F_{\delta\boldsymbol{\epsilon}}}(\mathbf{I})$	00	11	22	01	10	12	21	02	20
00		1	1						
11	1		1						
22	1	1							
01					-1				
10				-1					
12							-1		
21						-1			
02									-1
20								-1	

B.2.4 Lamé Coefficients

$\frac{\partial^2 \Psi^{\mathrm{nh}}}{\partial F_{\sigma\tau} \partial F_{\delta\epsilon}}(\mathbf{I})$	00	11	22	01	10	12	21	02	20
00	$\mu + \hat{\lambda}$	$-\mu + \hat{\lambda}$	$-\mu + \hat{\lambda}$						
11	$-\mu + \hat{\lambda}$	$\mu + \hat{\lambda}$	$-\mu + \hat{\lambda}$						
22	$-\mu + \hat{\lambda}$	$-\mu + \hat{\lambda}$	$\mu + \hat{\lambda}$						
01				μ	μ				
10				μ	μ				
12						μ	μ		
21						μ	μ		
02								μ	μ
20								μ	μ

This is only consistent with linear elasticity if we have $-\mu + \hat{\lambda} = \lambda$, note that then $\mu + \hat{\lambda} = 2\mu + \lambda$.

B.3 XPBD and Gauss-Seidel

Macklin et al. [MMC16] show that PBD [MHH07] can be seen to be the extreme case of a numerical method for the approximation of the backward Euler temporal discretization of the FEM spatial discretization of Equation (2.1)

$$\sum_{j=0}^{N^N-1} m_{ij} \left(\frac{\mathbf{x}_j^{n+1} - 2\mathbf{x}_j^n + \mathbf{x}_j^{n-1}}{\Delta t^2} \right) = \mathbf{f}_i(\mathbf{x}^{n+1}) + \mathbf{f}_i^{\mathrm{ext}}, \ \mathbf{X}_i \notin \Omega_D^0. \tag{B.14}$$

Here $m_{ii} = \int_{\Omega^0} R^0 N_i d\mathbf{X}$ and $m_{ij} = 0,\ j \neq i$ are entries in the mass matrix. However, they require that the discrete potential energy in Equation (2.15) is of the form

$$\hat{PE}^{\Psi}(\mathbf{y}) = \sum_{c=0}^{2N^E-1} \frac{1}{2\alpha_c} C_c^2(\mathbf{y}), \ \mathbf{y} \in \mathbb{R}^{dN^E}. \tag{B.15}$$

For example, this can be done with the energy densities in Equations (2.7) and (2.8) using two constraints $c = 2e$ and $c = 2e + 1$ per element e

$$C_{2e}^{\text{cor}}(\mathbf{y}) = |\mathbf{F}^e(\mathbf{y}) - \mathbf{R}(\mathbf{F}^e(\mathbf{y}))|_F, \; C_{2e+1}^{\text{cor}}(\mathbf{y}) = \det(\mathbf{F}^e(\mathbf{y})) - 1 \tag{B.16}$$

$$C_{2e}^{\text{nh}}(\mathbf{y}) = |\mathbf{F}^e(\mathbf{y})|_F, \; C_{2e+1}^{\text{nh}}(\mathbf{y}) = \det(\mathbf{F}^e(\mathbf{y})) - 1 - \frac{\mu}{\hat{\lambda}}. \tag{B.17}$$

In this case, $\alpha_{2e}^{\text{cor}} = \frac{1}{2\mu V_e^0}$, $\alpha_{2e+1}^{\text{cor}} = \frac{1}{\lambda V_e^0}$, $\alpha_{2e}^{\text{nh}} = \frac{1}{\mu V_e^0}$, $\alpha_{2e+1}^{\text{nh}} = \frac{1}{\hat{\lambda} V_e^0}$

To demonstrate the connection between Equation (B.14) and PBD, Macklin et al. [MMC16] develop XPBD. It is based on the total Lagrange multiplier formulation

$$\sum_{j=0}^{N^N-1} m_{ij} \left(\mathbf{x}_j^{n+1} - \hat{\mathbf{x}}_j\right) - \sum_{c=0}^{P-1} \frac{\partial C_c}{\partial \mathbf{x}_i}(\mathbf{x}^{n+1})\lambda_c^{n+1} = 0, \; \mathbf{X}_i \notin \Omega_D^0 \tag{B.18}$$

$$C_c(\mathbf{x}^{n+1}) + \frac{\alpha_c}{\Delta t^2}\lambda_c^{n+1} = 0, \; 0 \le c < P \tag{B.19}$$

where $\hat{\mathbf{x}}_j = 2\mathbf{x}_j^n - \mathbf{x}_j^{n-1} - \frac{\Delta t^2}{m_{jj}}\mathbf{f}_j^{\text{ext}}$ and $\boldsymbol{\lambda}^{n+1} \in \mathbb{R}^P$ is introduced as an additional unknown. The $\mathbf{x}^{n+1} \in \mathbb{R}^{dN^N}$ in Equations (B.18)-(B.19) is the same in the solution to Equation (B.14). Macklin et al. [MMC16] use a per-constraint Gauss-Seidel update of Equations (B.18)-(B.19)

$$\mathbf{x}_{i(k+1)}^{n+1,l} = \mathbf{x}_{i(k)}^{n+1,l} + \Delta\mathbf{x}_{i(k+1)}^{n+1,l}, \; \mathbf{X}_i \notin \Omega_D^0 \tag{B.20}$$

$$\Delta\mathbf{x}_{i(k+1)}^{n+1,l} = \frac{\Delta\lambda_{(k+1)c_k}^{n+1,l}}{m_{ii}} \frac{\partial C_{c_k}}{\partial \mathbf{x}_i}(\mathbf{x}_{(k)}^{n+1,l}) \tag{B.21}$$

$$\Delta\lambda_{(k+1)c_k}^{n+1,l} = \frac{-C_{c_k}(\mathbf{x}_{(k)}^{n+1,l}) + \frac{\alpha_{c_k}}{\Delta t^2} C_{c_k}(\mathbf{x}_{(k)}^{n+1,l})}{\sum_{j=0}^{N^N-1} \frac{1}{m_{jj}} \sum_{\beta=0}^{d-1} \left(\frac{\partial C_{c_k}}{\partial x_{j\beta}}(\mathbf{x}_{(k)}^{n+1,l})\right)^2 + \frac{\alpha_{c_k}}{\Delta t^2}}. \tag{B.22}$$

Here the $k + 1^{\text{th}}$ sub-iterate in iteration l is generated by solving for the change in a single Lagrange multiplier $\Delta\lambda_{(k+1)c_k}^{n+1,l}$ associated with a constraint c_k that varies from sub-iteration to sub-iteration.

B.3.1 Quasistatics

As noted by Macklin et al. [MMC16], the XPBD update in Equations (B.20)-(B.22) is the same as in the original PBD [MHH07] in the limit $\alpha_c \to 0$. By choosing a stiffness inversely proportionate to a parameter $s \geq 0$ and examining the limiting behavior of the equations being approximated, we see that the original PBD approach generates an approximation to the quasistatic problem (Equations (2.5)), albeit with the external forcing terms omitted. More precisely, define $\boldsymbol{\phi}_s$ to be a solution of the problem

$$sR^0 \frac{\partial^2 \boldsymbol{\phi}_s}{\partial t^2} = \nabla^{\mathbf{X}} \cdot \mathbf{P} + s\mathbf{f}^{\text{ext}}. \tag{B.23}$$

subject to the same boundary conditions in Equations (2.2)-(2.3). This is equivalent to scaling the α_c that would appear in Equation (2.1) (through $\mathbf{P}$) by s. The α_c are inversely proportionate to the Lamé parameters, so as $s \to 0$, the material stiffness increases. Since the inertia and external force terms in Equation (B.23) vanish as $s \to 0$, it is clear then that the original PBD formulation generates an approximation to the solution of a quasistatic problem with the external forcing $\mathbf{f}^{\text{ext}}$ omitted. Note that PBD does include the external forcing term in its initial guess $\mathbf{x}_i^{n+1} = \mathbf{x}_i^n + \Delta t \mathbf{v}_i^n + \frac{\Delta t^2}{m_{ii}} \mathbf{f}_i^{\text{ext}}$. However, the effect of the initial guess vanishes as the iteration count is increased. We demonstrate this in Section 6.8.4 Also, note that this is not the case in the XPBD formulation where $\alpha_c > 0$.

Unfortunately, XPBD cannot be trivially modified to run quasistatic problems. For example, omitting the mass terms on the left-hand side of Equation (B.18) makes the Gauss-Seidel update in Equations(B.20)-(B.22) impossible since there would be a division by zero. The simplest fix for quasistatic problems we can conceive of in the PBD framework is to use XPBD run to steady state using a pseudo-time iteration. This prevents the need for scaling the α_c which inherently removes the external forcing terms and does not introduce a divide by zero in Equation (B.21). However, this is very costly since each quasistatic time step is essentially the cost of an entire XPBD simulation. Nevertheless, we compare our approach against this option (see Section 6.8.4) since it will at least allow for the correct representation of the forcing terms. We refer to this technique as XPBD-QS.

REFERENCES

[ACW06] A. Angelidis, M.-P. Cani, G. Wyvill, and S. King. "Swirling-sweepers: Constant-volume modeling." Graph. Models, **68**(4):324–332, 2006.

[ANF11] Brian F Allen, Michael Neff, and Petros Faloutsos. "Analytic proportional-derivative control for precise and compliant motion." In 2011 IEEE International Conference on Robotics and Automation, pp. 6039–6044. IEEE, 2011.

[ARM19] B. Angles, D. Rebain, M. Macklin, B. Wyvill, L. Barthe, J. Lewis, J. Von Der Pahlen, S. Izadi, J. Valentin, S. Bouaziz, and A. Tagliasacchi. "VIPER: Volume Invariant Position-based Elastic Rods." Proc. ACM Comput Graph Interact Tech, **2**(2), 2019.

[ASK05] D. Anguelov, P. Srinivasan, D. Koller, S. Thrun, J. Rodgers, and J. Davis. "Scape: shape completion and animation of people." In ACM SIGGRAPH 2005 Papers, pp. 408–416. 2005.

[AST10] E. de Aguiar, L. Sigal, A. Treuille, and J. Hodgins. "Stable Spaces for Real-time Clothing." ACM Trans. Graph., **29**(4):106:1–106:9, 2010.

[Att10] M. Attene. "A lightweight approach to repairing digitized polygon meshes." The visual computer, **26**(11):1393–1406, 2010.

[Ber97] D. Bertsekas. "Nonlinear programming." J Op Res Soc, **48**(3):334–334, 1997.

[BFG20] H. Brönnimann, A. Fabri, G.-J. Giezeman, S. Hert, M. Hoffmann, L. Kettner, S. Pion, and S. Schirra. "2D and 3D Linear Geometry Kernel." In CGAL User and Reference Manual. CGAL Editorial Board, 5.2 edition, 2020.

[BME21] H. Bertiche, M. Madadi, and S. Escalera. "PBNS: Physically Based Neural Simulator for Unsupervised Garment Pose Space Deformation.", 2021.

[BML14] S. Bouaziz, S. Martin, T. Liu, L. Kavan, and M. Pauly. "Projective Dynamics: Fusing Constraint Projections for Fast Simulation." ACM Trans Graph, **33**(4):154:1–154:11, 2014.

[BOD18] S. Bailey, D. Otte, P. Dilorenzo, and J. O'Brien. "Fast and Deep Deformation Approximations." ACM Trans Graph, **37**(4):119:1–12, August 2018.

[BOD20] Stephen W Bailey, Dalton Omens, Paul Dilorenzo, and James F O'Brien. "Fast and deep facial deformations." ACM Transactions on Graphics (TOG), **39**(4):94–1, 2020.

[BPC11] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. "Distributed optimization and statistical learning via the alternating direction method of multipliers." Foundations and Trends in Machine Learning, **3**(1):1–122, 2011.

[BPD05] S. Blemker, P. Pinsky, and S. Delp. "A 3D model of muscle reveals the causes of nonuniform strains in the biceps brachii." J. Biomech, **38**(4):657–665, 2005.

[BSM20] M. Botsch, D. Sieger, P. Moeller, and A. Fabri. "Surface Mesh." In CGAL User and Reference Manual. CGAL Editorial Board, 5.2 edition, 2020.

[BW98] D. Baraff and A. Witkin. "Large Steps in Cloth Simulation." In Proc ACM SIGGRAPH, SIGGRAPH '98, pp. 43–54, 1998.

[BW08] J. Bonet and R. Wood. Nonlinear continuum mechanics for finite element analysis. Cambridge University Press, 2008.

[CB81] Roy D. Crowninshield and Richard A. Brand. "A physiologically based criterion of muscle force prediction in locomotion." Journal of Biomechanics, **14**(11):793–801, 1981.

[CBE15] M. Cong, M. Bao, J. E, K. Bhat, and R. Fedkiw. "Fully automatic generation of anatomical face simulation models." In Proc ACM SIGGRAPH/Eurographics Symp Comp Anim, pp. 175–183, 2015.

[CBF16] M. Cong, L. Bhat, and R. Fedkiw. "Art-Directed Muscle Simulation for High-End Facial Animation." In Proc 2016 ACM SIGGRAPH/Eurographics Symp Comp Anim, pp. 119–127. Eurographics Association, 2016.

[CGC02] Steve Capell, Seth Green, Brian Curless, Tom Duchamp, and Zoran Popović. "Interactive skeleton-driven dynamic deformations." ACM transactions on graphics (TOG), **21**(3):586–593, 2002.

[CHC23] Y. Chen, Y. Han, J. Chen, MS. a, R. Fedkiw, and J. Teran. "Primal Extended Position Based Dynamics for Hyperelasticity." In Proceedings of the 16th ACM SIGGRAPH Conference on Motion, Interaction and Games, MIG '23, 2023.

[CHC24] Y. Chen, Y. Han, J. Chen, Z. Zhang, A. McAdams, and J. Teran. "Position-Based Nonlinear Gauss-Seidel for Quasistatic Hyperelasticity." ACM Trans. Graph., 2024.

[CLZ13] Y. Chen, Z. Liu, and Z. Zhang. "Tensor-Based Human Body Modeling." In Proc IEEE CVPR, 2013.

[CMM20] N. Chentanez, M. Macklin, M. Müller, S. Jeschke, and T. Kim. "Cloth and Skin Deformation with a Triangle Mesh Based Convolutional Neural Network." Comp Graph Forum, **39**(8):123–134, 2020.

[CPS10] I. Chao, U. Pinkall, P. Sanan, and P. Schröder. "A Simple Geometric Model for Elastic Deformations." ACM Trans Graph, **29**(4), 2010.

[CYJ18] Lan Chen, Juntao Ye, Liguo Jiang, Chengcheng Ma, Zhanglin Cheng, and Xiaopeng Zhang. "Synthesizing cloth wrinkles by CNN-based geometry image superresolution." Computer Animation and Virtual Worlds, **29**(3-4):e1810, 2018.

[CZ92] D. Chen and D. Zeltzer. "Pump it up: Computer animation of a biomechanically based model of muscle using the finite element method." In Proc 19th SIGGRAPH, pp. 89–98, 1992.

[DAA07] S. Delp, F. Anderson, A. Arnold, P. Loan, A. Habib, C. John, E. Guendelman, and D. Thelen. "OpenSim: open-source software to create and analyze dynamic simulations of movement." IEEE Trans Biomed Eng, **54**(11):1940–1950, 2007.

[DDS11] Suvranu De, Dhannanjay Deo, Ganesh Sankaranarayanan, and Venkata S Arikatla. "A physics-driven neural networks-based simulation system (phynness) for multimodal interactive virtual environments involving nonlinear deformable objects." Presence, **20**(4):289–308, 2011.

[DLH90] S. Delp, J. Loan, M. Hoy, F. Zajac, E. Topp, and J. Rosen. "An interactive graphics-based model of the lower extremity to study orthopaedic surgical procedures." IEEE Trans Biomed Eng, **37**(8):757–767, 1990.

[DMH20] Congyue Deng, Tai-Jiang Mu, and Shi-Min Hu. "Alternating convlstm: Learning force propagation with alternate state updates." arXiv preprint arXiv:2006.07818, 2020.

[DSZ05] Alessandro De Luca, Bruno Siciliano, and Loredana Zollo. "PD control with online gravity compensation for robots with elastic joints: Theory and experiments." automatica, **41**(10):1809–1819, 2005.

[DT18] T. Dao and M. Tho. "A systematic review of continuum modeling of skeletal muscles: current trends, limitations, and recommendations." App bionic biomech, **2018**, 2018.

[EGS03] O. Etzmuss, J. Gross, and W. Strasser. "Deriving a particle system from continuum mechanics for the animation of deformable objects." IEEE Trans Vis Comp Graph, **9**(4):538–550, October 2003.

[Epi21] Epic Games. "MetaHuman Creator.", 2021.

[FB81] Martin A Fischler and Robert C Bolles. "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography." Communications of the ACM, **24**(6):381–395, 1981.

[FLP14] Y. Fan, J. Litven, and D. Pai. "Active Volumetric Musculoskeletal Systems." ACM Trans Graph, **33**(4), 2014.

[FMD19] Lawson Fulton, Vismay Modi, David Duvenaud, David IW Levin, and Alec Jacobson. "Latent-space Dynamics for Reduced Deformable Simulation." In Computer graphics forum, volume 38, pp. 379–391. Wiley Online Library, 2019.

[FTS06] W. Von Funck, H. Theisel, and H.-P. Seidel. "Vector field based shape deformations." ACM Trans. Graph., **25**(3):1118–1125, 2006.

[FVP16] M. Fratarcangeli, T. Valentina, and F. Pellacini. "Vivace: a practical gauss-seidel method for stable soft body dynamics." ACM Trans Graph, **35**(6):1–9, Nov 2016.

[FYK10] W. Feng, Y. Yu, and B. Kim. "A Deformation Transformer for Real-time Cloth Animation." ACM Trans. Graph., **29**(4):108:1–108:9, 2010.

[GD01] J. Gain and N. Dodgson. "Preventing self-intersection under free-form deformation." IEEE Trans Viz Comp Grap, **7**(4):289–298, 2001.

[GFJ16] T. Gast, C. Fu, C. Jiang, and J. Teran. "Implicit-shifted Symmetric QR Singular Value Decomposition of 3x3 Matrices." Technical report, University of California Los Angeles, 2016.

[GHC24] S. Gagniere, Y. Han, Y. Chen, D. Hyde, A. Marquez-Razon, J. Teran, and R. Fedkiw. "A Robust Grid-Based Meshing Algorithm for Embedding Self-Intersecting Surfaces." Computer Graphics Forum, 2024.

[GJF20] Z. Geng, D. Johnson, and R. Fedkiw. "Coercing machine learning to output physically accurate results." J Comp Phys, **406**:109099, 2020.

[GRH12] Peng Guan, Loretta Reiss, David A Hirshberg, Alexander Weiss, and Michael J Black. "Drape: Dressing any person." ACM Transactions on Graphics (TOG), **31**(4):1–10, 2012.

[GS08] O. Gonzalez and A. Stuart. A first course in continuum mechanics. Cambridge University Press, 2008.

[GSS15] T. Gast, C. Schroeder, A. Stomakhin, C. Jiang, and J. Teran. "Optimization Integrator for Large Time Steps." IEEE Trans Vis Comp Graph, **21**(10):1103–1115, 2015.

[GTH98] Radek Grzeszczuk, Demetri Terzopoulos, and Geoffrey Hinton. "Neuroanimator: Fast neural network emulation and control of physics-based models." In Proceedings of the 25th annual conference on Computer graphics and interactive techniques, pp. 9–20, 1998.

[HCO24] Y. Han, Y. Chen, C. Ong, J. Chen, J. Hicks, and J. Teran. "A Neural Network Model for Efficient Musculoskeletal-Driven Skin Deformation." ACM Trans. Graph., 2024.

[HDD19] Daniel Holden, Bang Chi Duong, Sayantan Datta, and Derek Nowrouzezahrai. "Subspace neural physics: Fast data-driven interactive simulation." In Proceedings of the 18th annual ACM SIGGRAPH/Eurographics Symposium on Computer Animation, pp. 1–12, 2019.

[HLS12] F. Hecht, Y. Lee, J. Shewchuk, and J. O'Brien. "Updated Sparse Cholesky Factors for Corotational Elastodynamics." ACM Trans Graph, **31**(5), 2012.

[Hol92] John H Holland. Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. MIT press, 1992.

[HPS11] D. Harmon, D. Panozzo, O. Sorkine, and D. Zorin. "Interference-aware geometric modeling." ACM Transactions on Graphics (TOG), **30**(6):1–10, 2011.

[HT89] W. Horn and D. Taylor. "A theorem to determine the spatial containment of a point in a planar polyhedron." Comp Vis Graph Imag Proc, **45**(1):106–116, 1989.

[HTC14] Fabian Hahn, Bernhard Thomaszewski, Stelian Coros, Robert W Sumner, Forrester Cole, Mark Meyer, Tony DeRose, and Markus Gross. "Subspace clothing simulation using adaptive bases." ACM Transactions on Graphics (TOG), **33**(4):1–9, 2014.

[Hug00] T. Hughes. The finite element method : linear static and dynamic finite elment analysis. Dover, 2000.

[HUS15] Jennifer L Hicks, Thomas K Uchida, Ajay Seth, Apoorva Rajagopal, and Scott L Delp. "Is my model good enough? Best practices for verification and validation of musculoskeletal models and simulations of movement." Journal of biomechanical engineering, **137**(2):020905, 2015.

[HWB95] Jessica K Hodgins, Wayne L Wooten, David C Brogan, and James F O'Brien. "Animating human athletics." In Proceedings of the 22nd annual conference on Computer graphics and interactive techniques, pp. 71–78, 1995.

[HZG18] Y. Hu, Q. Zhou, X. Gao, A. Jacobson, D. Zorin, and D. Panozzo. "Tetrahedral Meshing in the Wild." ACM Trans. Graph., **37**(4):60:1–60:14, July 2018.

[IHB15] J. Inouye, G. Handsfield, and S. Blemker. "Fiber Tractography for Finite-Element Modeling of Transversely Isotropic Biological Tissues of Arbitrary Shape Using Computational Fluid Dynamics." In Proc Conf Summer Comp Sim, p. 1?6. Soc Comp Sim Int, 2015.

[IKK17] A. Ichim, P. Kadleček, L. Kavan, and M. Pauly. “Phace: Physics-Based Face Modeling and Animation.” ACM Trans Graph, **36**(4), 2017.

[JF22] Daniel Johnson and Ronald Fedkiw. “Smoothing Discontinuous Root-Finding for Subsequent Differentiability in Learning, Inverse Problems, and Control.” preprint, 2022.

[JHG22] Y. Jin, Y. Han, Z. Geng, J. Teran, and R. Fedkiw. “Analytically Integratable Zero-Restlength Springs for Capturing Dynamic Modes Unrepresented by Quasistatic Neural Networks.” In ACM SIGGRAPH 2022 Conf Proc, SIGGRAPH ’22, New York, NY, USA, 2022. ACM.

[JWG19] Y. Jiang, T. Van Wouwe, F. De Groote, and K. Liu. “Synthesis of Biologically Realistic Human Motion Using Joint Torque Actuation.” ACM Trans Graph, **38**(4), 2019.

[JZG20] N. Jin, Y. Zhu, Z. Geng, and R. Fedkiw. “A pixel-based framework for data-driven clothing.” In Comp Graph Forum, volume 39, pp. 135–144, 2020.

[KBT17] D. Koschier, J. Bender, and N. Thuerey. “Robust eXtended Finite Elements for complex cutting of deformables.” ACM Trans Graph, **36**(4):55:1–55:13, 2017.

[KCO09] L. Kavan, S. Collins, and C. O’Sullivan. “Automatic Linearization of Nonlinear Skinning.” In Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games, I3D ’09, p. 49?56. ACM, 2009.

[KCv07] L. Kavan, S. Collins, J. Žára, and C. O’Sullivan. “Skinning with Dual Quaternions.” In Proc 2007 Symp Int 3D Graph Games, I3D ’07, p. 39?46. ACM, 2007.

[KGB11] L. Kavan, D. Gerszewski, A. Bargteil, and P. Sloan. “Physics-inspired Upsampling for Cloth Simulation in Games.” ACM Trans Graph, **30**(4):93:1–93:10, 2011.

[KGL16] S. Kovalsky, M. Galun, and Y. Lipman. “Accelerated Quadratic Proxy for Geometric Optimization.” ACM Trans Graph, **35**(4), 2016.

[KPP17] Meekyoung Kim, Gerard Pons-Moll, Sergi Pujades, Seungbae Bang, Jinwook Kim, Michael J Black, and Sung-Hee Lee. “Data-driven physics for human soft tissue animation.” ACM Transactions on Graphics (TOG), **36**(4):1–12, 2017.

[KRA23] J. Kneifl, D. Rosin, O. Avci, O. Röhrle, and J. Fehr. “Low-dimensional data-based surrogate model of a continuum-mechanical musculoskeletal system based on non-intrusive model order reduction.” Arch App Mech, **93**(9):3637–3663, 2023.

[KS12] L. Kavan and O. Sorkine. "Elasticity-Inspired Deformers for Character Articulation." ACM Trans Graph, **31**(6), 2012.

[KZ05] L. Kavan and J. Žára. "Spherical blend skinning: a real-time deformation of articulated models." In Proc 2005 Symp Int 3D Grap Games, pp. 9–16, 2005.

[LAH21] P. Li, K. Aberman, R. Hanocka, L. Liu, O. Sorkine-Hornung, and B. Chen. "Learning Skeletal Articulations with Neural Blend Shapes." ACM Trans Graph, **40**(4), 2021.

[LB18] Y. Li and J. Barbič. "Immersion of Self-Intersecting Solids and Surfaces." ACM Trans. Graph., **37**(4), July 2018.

[LBK17] T. Liu, S. Bouaziz, and L. Kavan. "Quasi-Newton Methods for Real-Time Simulation of Hyperelastic Materials." ACM Trans Graph, **36**(4), 2017.

[LBO13] T. Liu, A. Bargteil, J. O'Brien, and L. Kavan. "Fast Simulation of Mass-Spring Systems." ACM Trans Graph, **32**(6):209:1–7, 2013.

[LCF00] J. Lewis, M. Cordner, and N. Fong. "Pose Space Deformation: A Unified Approach to Shape Interpolation and Skeleton-Driven Deformation." In Proc 27th SIGGRAPH, SIGGRAPH '00, p. 165?172. ACM Press/Addison-Wesley Publishing Co., 2000.

[LCT18] Zorah Lahner, Daniel Cremers, and Tony Tung. "Deepwrinkles: Accurate and realistic clothing modeling." In Proceedings of the European Conference on Computer Vision (ECCV), pp. 667–684, 2018.

[LGL19] M. Li, M. Gao, T. Langlois, C. Jiang, and D. Kaufman. "Decomposed Optimization Time Integrator for Large-Step Elastodynamics." ACM Trans Graph, **38**(4), jul 2019.

[LHE20] Haolin Liu, Ye Han, Daniel Emerson, Houriyeh Majditehran, Qi Wang, Yoed Rabin, and Levent Burak Kara. "Real-time Prediction of Soft Tissue Deformations Using Data-driven Nonlinear Presurgical Simulations." arXiv preprint arXiv:2010.13823, 2020.

[LJL23] S. Lee, Y. Jiang, and K. Liu. "Anatomically Detailed Simulation of Human Torso." ACM Trans Graph, **42**(4), 2023.

[LJS15] L'ubor Ladickỳ, SoHyeon Jeong, Barbara Solenthaler, Marc Pollefeys, and Markus Gross. "Data-driven fluid simulations using regression forests." ACM Transactions on Graphics (TOG), **34**(6):1–9, 2015.

[LMR15] M. Loper, N. Mahmood, J. Romero, G. Pons-Moll, and M. Black. "SMPL: A Skinned Multi-Person Linear Model." ACM Trans Graph, **34**(6), 2015.

[LPK14] Y. Lee, M. Park, T. Kwon, and J. Lee. "Locomotion Control for Many-Muscle Humanoids." ACM Trans Graph, **33**(6), 2014.

[LPL19] S. Lee, M. Park, K. Lee, and J. Lee. "Scalable Muscle-Actuated Human Simulation and Control." ACM Trans Graph, **38**(4), 2019.

[LRT20] S. Loriot, M. Rouxel-Labbé, J. Tournois, and I. Yaz. "Polygon Mesh Processing." In CGAL User and Reference Manual. CGAL Editorial Board, 5.2 edition, 2020.

[LS07] F. Labelle and J. Shewchuk. "Isosurface Stuffing: Fast Tetrahedral Meshes with Good Dihedral Angles." In ACM SIGGRAPH 2007, SIGGRAPH '07, pp. 57–es, New York, NY, USA, 2007. ACM.

[LSN13] D. Li, S. Sueda, D. Neog, and D. Pai. "Thin Skin Elastodynamics." ACM Trans Graph, **32**(4):49:1–49:9, 2013.

[LST09] S. Lee, E. Sifakis, and D. Terzopoulos. "Comprehensive Biomechanical Modeling and Simulation of the Upper Body." ACM Trans Graph, **28**(4), sep 2009.

[LSW20] R. Luo, T. Shao, H. Wang, W. Xu, X. Chen, K. Zhou, and Y. Yang. "NNWarp: Neural Network-Based Nonlinear Deformation.", 2020.

[LT06] S. Lee and D. Terzopoulos. "Heads up! Biomechanical Modeling and Neuromuscular Control of the Neck." ACM Trans Graph, **25**(3):1188?1198, 2006.

[LYP18] S. Lee, R. Yu, J. Park, M. Aanjaneya, E. Sifakis, and J. Lee. "Dexterous Manipulation and Control with Volumetric Muscles." ACM Trans Graph, **37**(4), 2018.

[LZJ22] Y. Li, L. Zhang nd Z. Qiu, Y. Jiang, N. Li, Y. Ma, Y. Zhang, L. Xu, and J. Yu. "NIMBLE: A Non-Rigid Hand Model with Bones and Muscles." ACM Trans Graph, **41**(4), 2022.

[LZX08] L. Liu, L. Zhang, Y. Xu, C. Gotsman, and S. Gortler. "A Local/Global Approach to Mesh Parameterization." In Proc Symp Geom Proc, SGP '08, p. 1495?1504. Eurograph Assoc, 2008.

[MBT03] N. Molino, R. Bridson, J. Teran, and R. Fedkiw. "A Crystalline, Red Green Strategy for Meshing Highly Deformable Objects with Tetrahedra." In Int Mesh Round, pp. 103–114. Citeseer, 2003.

[MCC11] T. McLaughlin, L. Cutler, and D. Coleman. "Character Rigging, Deformations, and Simulations in Film and Game Production." In ACM SIGGRAPH 2011 Courses, SIGGRAPH '11, New York, NY, USA, 2011. ACM.

[MCH22] Alan Marquez, Yizhou Chen, Yushan Han, Steven Gagniere, Michael Tupek, and Joseph Teran. "A Momentum Conserving Hybrid Particle/Grid Iteration for Volumetric Elastic Contact." preprint, 2022.

[MDM02] M. Müller, J. Dorsey, L. McMillan, R. Jagnow, and B. Cutler. "Stable real-time deformations." In Proc 2002 ACM SIGGRAPH/Eurograph Symp Comp Anim, pp. 49–54, 2002.

[MDR14] J. Mancewicz, M. Derksen, H. Rijpkema, and C. Wilson. "Delta Mush: Smoothing Deformations While Preserving Detail." In Proc Fourth Symp Digital Prod, DigiPro '14, p. 7?11. ACM, 2014.

[MFJ21] V. Modi, L. Fulton, A. Jacobson, S. Sueda, and D. Levin. "Emu: Efficient muscle simulation in deformation space." In Comp Graph Forum, volume 40, pp. 234–248. Wiley Online Library, 2021.

[MFS21] Luke Metz, C Daniel Freeman, Samuel S Schoenholz, and Tal Kachman. "Gradients are Not All You Need." arXiv preprint arXiv:2111.05803, 2021.

[MG03] A. Mohr and M. Gleicher. "Building Efficient, Accurate Character Skins from Examples." ACM Trans Graph, **22**(3):562?568, 2003.

[MG04] M. Müller and M. Gross. "Interactive virtual materials." In Proc Graph Int, pp. 239–246. Canadian Human-Computer Communications Society, 2004.

[MHH07] M. Müller, B. Heidelberger, M. Hennix, and J. Ratcliff. "Position based dynamics." J Vis Comm Im Rep, **18**(2):109–118, 2007.

[MLT89] N. Magnenat-Thalmann, R. Laperrière, and D. Thalmann. "Joint-Dependent Local Deformations for Hand Animation and Object Grasping." In ProcGraph Int '88, pp. 26–33. Canadian Information Processing Society, 1989.

[MM21] M. Macklin and M. Muller. "A Constraint-based Formulation of Stable Neo-Hookean Materials." In Motion, Interaction and Games, pp. 1–7. ACM, Nov 2021.

[MMC16] M. Macklin, M. Müller, and N. Chentanez. "XPBD: Position-Based Simulation of Compliant Constrained Dynamics." In Proc 9th Int Conf Motion Games, MIG '16, p. 49?54. ACM, 2016.

[MMC20] Andrea Mendizabal, Pablo Márquez-Neila, and Stéphane Cotin. "Simulation of hyperelastic materials in real-time using deep learning." Medical image analysis, **59**:101569, 2020.

[MMG06] B. Merry, P. Marais, and J. Gain. "Animation Space: A Truly Linear Framework for Character Animation." ACM Trans Graph, **25**(4):1400–1423, 2006.

[MNG15] M.Tournier, M. Nesme, B. Gilles, and F. Faure. "Stable Constrained Dynamics." ACM Trans Graph, pp. 1–10, 2015.

[MPM20] F. Meister, T. Passerini, V. Mihalef, A. Tuysuzoglu, A. Maier, and T. Mansi. "Deep learning acceleration of Total Lagrangian Explicit Dynamics for soft tissue mechanics." Comp Meth App Mech Eng, **358**:112628, 2020.

[MTG11] S. Martin, B. Thomaszewski, E. Grinspun, and M.Gross. "Example-Based Elastic Materials." In ACM SIGGRAPH 2011, SIGGRAPH '11. ACM, 2011.

[MZS11] A. McAdams, Y. Zhu, A. Selle, M. Empey, R. Tamstorf, J. Teran, and E. Sifakis. "Efficient Elasticity for Character Skinning with Contact and Collisions." ACM Trans Graph, **30**(4):37:1–37:12, 2011.

[Neu85] J. Neuberger. "Steepest descent and differential equations." J Math Soc Japan, **37**(2):187–195, 1985.

[Ng 98] V. Ng-Thow-Hing. "Anatomically-based models for physical and geometrical reconstruction of animals." 1998.

[NOB16] R. Narain, M. Overby, and G. Brown. "ADMM Projective Dynamics: Fast Simulation of General Constitutive Models." In Proc ACM SIGGRAPH/Eurograph Symp Comp Anim, SCA '16, p. 21?28. Eurograph Assoc, 2016.

[NW06] J. Nocedal and S. Wright. "Conjugate gradient methods." Num Opt, pp. 101–134, 2006.

[OF03] S. Osher and R. Fedkiw. Level set methods and dynamic implicit surfaces. Applied mathematical science. Springer, New York, N.Y., 2003.

[PFS20] Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter Battaglia. "Learning Mesh-Based Simulation with Graph Networks." In International Conference on Learning Representations, 2020.

[PLF14] D. Pai, D. Levin, and Y. Fan. "Eulerian Solids for Soft Tissue and More." In ACM SIGGRAPH 2014 Courses, SIGGRAPH '14. ACM, 2014.

[PLP20] Chaitanya Patel, Zhouyingcheng Liao, and Gerard Pons-Moll. "Tailornet: Predicting clothing in 3d as a function of human pose, shape and garment style." In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 7365–7375, 2020.

[PRM15] G. Pons-Moll, J. Romero, N. Mahmood, and M. Black. "Dyna: A Model of Dynamic Human Shape in Motion." ACM Trans Graph, **34**(4), 2015.

[PRW19] Micha Pfeiffer, Carina Riediger, Jürgen Weitz, and Stefanie Speidel. "Learning soft tissue behavior of organs for surgical navigation with convolutional neural networks." International journal of computer assisted radiology and surgery, **14**(7):1147–1155, 2019.

[RCC22] C. Romero, D. Casas, M. Chiaramonte, and M. Otaduy. "Contact-Centric Deformation Learning." ACM Trans Graph, **41**(4), 2022.

[RKL21] H. Ryu, M. Kim, S. Lee, M. Park, K. Lee, and J. Lee. "Functionality-Driven Musculature Retargeting." Comp Graph Forum, **40**(1):341–356, 2021.

[RKS18] Francois Roewer-Despres, Najeeb Khan, and Ian Stavness. "Towards finite element simulation using deep learning." In 15th international symposium on computer methods in biomechanics and biomedical engineering, 2018.

[RPK19] M. Raissi, P. Perdikaris, and G. E. Karniadakis. "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations." Journal of Computational physics, **378**:686–707, 2019.

[RPP17] M. Rabinovich, R. Poranne, D. Panozzo, and O. Sorkine-Hornung. "Scalable Locally Injective Mappings." ACM Trans Graph, **36**(2), 2017.

[SA07] O. Sorkine and M. Alexa. "As-Rigid-As-Possible Surface Modeling." In EUROGRAPHICS SYMPOSIUM ON GEOMETRY PROCESSING, 2007.

[SB12] E. Sifakis and J. Barbic. "FEM simulation of 3D deformable solids: a practitioner's guide to theory, discretization and model reduction." In ACM SIGGRAPH 2012 Courses, SIGGRAPH '12, pp. 20:1–20:50. ACM, 2012.

[SD06] A. Stern and M. Desbrun. "Discrete Geometric Mechanics for Variational Time Integrators." In ACM SIGGRAPH 2006 Courses, SIGGRAPH '06, p. 75?80. ACM, 2006.

[SDF07] E. Sifakis, K. Der, and R. Fedkiw. "Arbitrary cutting of deformable tetrahedralized objects." In Proc ACM SIGGRAPH/Eurograph Symp Comp Anim, pp. 73–80, 2007.

[SDM19] A. Seth, M. Dong, R. Matias, and S. Delp. "Muscle contributions to upper-extremity movement and work from a musculoskeletal model of the human shoulder." Frontiers in neurorobotics, **13**:90, 2019.

[SG21] Yasmin Salehi and Dennis Giannacopoulos. "PhysGNN: A Physics-Driven Graph Neural Network Based Model for Predicting Soft Tissue Deformation in Image-Guided Neurosurgery." arXiv preprint arXiv:2109.04352, 2021.

[SGK18] B. Smith, F. De Goes, and T. Kim. "Stable neo-hookean flesh simulation." ACM Trans Grap (TOG), **37**(2):1–15, 2018.

[SGK19] B. Smith, F. Goes, and T. Kim. "Analytic eigensystems for isotropic distortion energies." ACM Trans Graph (TOG), **38**(1):1–15, 2019.

[SGO20] I. Santesteban, E. Garces, M. Otaduy, and D. Casas. "SoftSMPL: Data-driven Modeling of Nonlinear Soft-tissue Dynamics for Parametric Humans." Comp Graph Forum, **39**(2):65–75, 2020.

[SGP20] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. "Learning to simulate complex physics with graph networks." In International Conference on Machine Learning, pp. 8459–8468. PMLR, 2020.

[She98] Jonathan Richard Shewchuk. "Tetrahedral Mesh Generation by Delaunay Refinement." In Proceedings of the Fourteenth Annual Symposium on Computational Geometry, p. 86–95, 1998.

[SHS12] A. Stomakhin, R. Howes, C. Schroeder, and J. Teran. "Energetically consistent invertible elasticity." In Proc Symp Comp Anim, pp. 25–32, 2012.

[SHU18] A. Seth, J. Hicks, T. Uchida, A. Habib, C. Dembia, J. Dunne, C. Ong, M. DeMers, A. Rajagopal, M. Millard, S. Hamner, E. Arnold, J. Yong, S. Lakshmikanth, M. Sherman, J. Ku, and S. Delp. "OpenSim: Simulating musculoskeletal dynamics and neuromuscular control to study human and animal movement." PLOS Computational Biology, **14**(7):1–20, 07 2018.

[Si15] H. Si. "TetGen, a Delaunay-Based Quality Tetrahedral Mesh Generator." ACM Trans. Math. Softw., **41**(2), February 2015.

[SJP13] L. Sacht, A. Jacobson, D. Panozzo, C. Schüller, and O. Sorkine-Hornung. "Consistent Volumetric Discretizations inside Self-Intersecting Surfaces." In Proceedings of the Eleventh Eurographics/ACMSIGGRAPH Symposium on Geometry Processing, SGP '13, pp. 147–156, Goslar, DEU, 2013. Eurographics Association.

[SKP08] S. Sueda, A. Kaufman, and D. Pai. "Musculotendon Simulation for Hand Animation." ACM Trans Graph, **27**(3), 2008.

[SLP21] Seung Heon Sheen, Egor Larionov, and Dinesh K Pai. "Volume Preserving Simulation of Soft Tissue with Skin." Proceedings of the ACM on Computer Graphics and Interactive Techniques, **4**(3):1–23, 2021.

[SNF05] E. Sifakis, I. Neverov, and R. Fedkiw. "Automatic Determination of Facial Muscle Activations from Sparse Motion Capture Marker Data." ACM Trans Graph, **24**(3):417?425, 2005.

[SOC19] I. Santesteban, M. Otaduy, and D. Casas. "Learning-Based Animation of Clothing for Virtual Try-On." Comp Graph Forum, **38**(2):355–366, 2019.

[SOW15] J. Selinger, S. O'Connor, J. Wong, and J. Donelan. "Humans Can Continuously Optimize Energetic Cost during Walking." Current Biology, **25**(18):2452–2456, 2015.

[SPC97] F. Scheepers, R. Parent, W. Carlson, and S. May. "Anatomy-Based Modeling of the Human Musculature." In Proc 24th SIGGRAPH, SIGGRAPH '97, p. 163?172, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[SSR20] S. Song, W. Shi, and M. Reed. "Accurate Face Rig Approximation with Deep Differential Subspace Reconstruction." ACM Trans Graph, **39**(4), 2020.

[ST08] R. Schmedding and M. Teschner. "Inversion handling for stable deformable modeling." Vis Comp, **24**(7-9):625–633, 2008.

[SWR21] S. Srinivasan, Q. Wang, J. Rojas, G. Klár, L. Kavan, and E. Sifakis. "Learning Active Quasistatic Physics-Based Models from Data." ACM Trans Graph, **40**(4), 2021.

[SZC21] Hyewon Seo, Kaifeng Zou, and Frederic Cordier. "DSNet: Dynamic Skin Deformation Prediction by Recurrent Neural Network." In Computer Graphics International Conference, pp. 365–377. Springer, 2021.

[SZK15] S. Saito, Z. Zhou, and L. Kavan. "Computational Bodybuilding: Anatomically-based Modeling of Human Bodies." ACM Trans Graph, **34**(4), 2015.

[TA06] Darryl G. Thelen and Frank C. Anderson. "Using computed muscle control to generate forward dynamic simulations of human walking from experimental data." Journal of biomechanics, **39 6**:1107–15, 2006.

[TBF19] M. Tao, C. Batty, E. Fiume, and D. Levin. "Mandoline: Robust Cut-Cell Generation for Arbitrary Triangle Meshes." ACM Trans. Graph., **38**(6), November 2019.

[TGL18] Qingyang Tan, Lin Gao, Yu-Kun Lai, Jie Yang, and Shihong Xia. "Mesh-based autoencoders for localized deformation component analysis." In Proceedings of the AAAI Conference on Artificial Intelligence, volume 32, 2018.

[The20] The CGAL Project. CGAL User and Reference Manual. CGAL Editorial Board, 5.2 edition, 2020.

[TPG20] Qingyang Tan, Zherong Pan, Lin Gao, and Dinesh Manocha. "Realtime simulation of thin-shell deformable materials using CNN-based mesh embedding." IEEE Robotics and Automation Letters, **5**(2):2325–2332, 2020.

[TSB05] J. Teran, E. Sifakis, S. Blemker, V. Ng-Thow-Hing, C. Lau, and R. Fedkiw. "Creating and simulating skeletal muscle from the visible human data set." IEEE Trans Vis Comp Graph, **11**(3):317–328, 2005.

[TSI05] J. Teran, E. Sifakis, G. Irving, and R. Fedkiw. "Robust quasistatic finite elements and flesh simulation." In Proc 2005 ACM SIGGRAPH/Eurograph Symp Comp Anim, pp. 181–190, 2005.

[VSP18] J. Valentin, M. Sprenger, D. Pflüger, and O. Röhrle. "Gradient-based optimization with b-splines on sparse grids for solving forward-dynamics simulations of three-dimensional, continuum-mechanical musculoskeletal system models." Int J Num Meth Biomed Eng, **34**(5):e2965, 2018.

[Wan15] H. Wang. "A Chebyshev Semi-Iterative Approach for Accelerating Projective and Position-Based Dynamics." ACM Trans Graph, **34**(6), nov 2015.

[Wan21] Huamin Wang. "GPU-based simulation of cloth wrinkles at submillimeter levels." ACM Transactions on Graphics (TOG), **40**(4):1–14, 2021.

[WBT19] S. Wiewel, M. Becher, and N. Thuerey. "Latent space physics: Towards learning the temporal evolution of fluid flow." In Computer graphics forum, volume 38, pp. 71–82. Wiley Online Library, 2019.

[WCC21] Nannan Wu, Qianwen Chao, Yanzhen Chen, Weiwei Xu, Chen Liu, Dinesh Manocha, Wenxin Sun, Yi Han, Xinran Yao, and Xiaogang Jin. "AgentDress: Realtime Clothing Synthesis for Virtual Agents using Plausible Deformations." IEEE Transactions on Visualization and Computer Graphics, **27**(11):4107–4118, 2021.

[WDG19] S. Wang, M. Ding, T. Gast, L. Zhu, S. Gagniere, C. Jiang, and J. Teran. "Simulation and Visualization of Ductile Fracture with the Material Point Method." Proc. ACM Comput. Graph Int Tech, **2**(2), 2019.

[WG97] J. Wilhelms and A. Van Gelder. "Anatomically based modeling." In Proc 24th SIGGRAPH, pp. 173–180, 1997.

[WGF07] Rachel Weinstein, Eran Guendelman, and Ronald Fedkiw. "Impulse-based control of joints and muscles." IEEE transactions on visualization and computer graphics, **14**(1):37–46, 2007.

[WGZ20] Jane Wu, Zhenglin Geng, Hui Zhou, and Ronald Fedkiw. “Skinning a parameterization of three-dimensional space for neural network cloth.” arXiv preprint arXiv:2006.04874, 2020.

[WHD12] J. Wang, S. Hamner, S. Delp, and V. Koltun. “Optimizing Locomotion Controllers Using Biologically-Based Actuators and Objectives.” ACM Trans Graph, **31**(4), 2012.

[WHR10] H. Wang, F. Hecht, R. Ramamoorthi, and J. O’Brien. “Example-based Wrinkle Synthesis for Clothing Animation.” ACM Trans. Graph., **29**(4):107:1–107:8, 2010.

[WJG21] Jane Wu, Yongxu Jin, Zhenglin Geng, Hui Zhou, and Ronald Fedkiw. “Recovering geometric information with learned texture perturbations.” Proceedings of the ACM on Computer Graphics and Interactive Techniques, **4**(3):1–18, 2021.

[WJS14] Y. Wang, C. Jiang, C. Schroeder, and J. Teran. “An adaptive virtual node algorithm with robust mesh cutting.” In Proc ACM SIGGRAPH/Eurograph Symp Comp Anim, pp. 77–85. Eurographics Association, 2014.

[WMB19] B. Wang, G. Matcuk, and J. Barbič. “Hand Modeling and Simulation Using Stabilized Magnetic Resonance Imaging.” ACM Trans Graph, **38**(4), 2019.

[WMB21] B. Wang, G. Matcuk, and J. Barbič. “Modeling of Personalized Anatomy using Plastic Strains.” ACM Trans Graph, **40**(2), 2021.

[WP02] C. Wang and C. Phillips. “Multi-Weight Enveloping: Least-Squares Approximation Techniques for Skin Animation.” In Proc 2002 ACM SIGGRAPH/Eurograph Symp Comp Anim, SCA ’02, p. 129?138. ACM, 2002.

[WVY22] Y. Wang, J. Verheul, S. Yeo, N. Kalantari, and S. Sueda. “Differentiable Simulation of Inertial Musculotendons.” ACM Trans Graph, **41**(6), 2022.

[WWD21] B. Witemeyer, N. Weidner, T. Davis, T. Kim, and S. Sueda. “QLB: Collision-Aware Quasi-Newton Solver with Cholesky and L-BFGS for Nonlinear Time Integration.” In Proc 14th ACM SIGGRAPH Conf Mot Int Games, MIG ’21. ACM, 2021.

[WY16] H. Wang and Y. Yang. “Descent methods for elastic body simulation on the GPU.” ACM Trans Graph, **35**(6):1–10, Nov 2016.

[WZB20] B. Wang, M. Zheng, and J. Barbič. “Adjustable Constrained Soft-Tissue Dynamics.” Pac Graph 2020 and Comp Graph Forum, **39**(7), 2020.

[XB16] Hongyi Xu and Jernej Barbič. “Pose-space subspace dynamics.” ACM Transactions on Graphics (TOG), **35**(4):1–14, 2016.

[XUC14] Weiwei Xu, Nobuyuki Umetani, Qianwen Chao, Jie Mao, Xiaogang Jin, and Xin Tong. "Sensitivity-optimized rigging for example-based real-time clothing synthesis." ACM Trans. Graph., **33**(4):107–1, 2014.

[ZBK18] Y. Zhu, R. Bridson, and D. Kaufman. "Blended Cured Quasi-Newton for Distortion Optimization." ACM Trans Graph, **37**(4), jul 2018.

[ZBL20] Jiayi Eris Zhang, Seungbae Bang, David I.W. Levin, and Alec Jacobson. "Complementary Dynamics." ACM Transactions on Graphics, 2020.

[ZBO13] J. Zurdo, J. Brito, and M. Otaduy. "Animating Wrinkles by Example on Non-Skinned Cloth." IEEE Trans Vis Comp Grap, **19**(1):149–158, 2013.

[ZWC21] Meng Zhang, Tuanfeng Y Wang, Duygu Ceylan, and Niloy J Mitra. "Dynamic neural garments." ACM Transactions on Graphics (TOG), **40**(6):1–15, 2021.

[ZWH22] M. Zheng, B. Wang, J. Huang, and J. Barbič. "Simulation of Hand Anatomy Using Medical Imaging." ACM Trans Graph, **41**(6), 2022.

[ZZC21] Mianlun Zheng, Yi Zhou, Duygu Ceylan, and Jernej Barbic. "A Deep Emulator for Secondary Motion of 3D Characters." In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 5932–5940, 2021.