UNIVERSITY OF CALIFORNIA
Los Angeles

Novel Efficient Implicit Methods for Elastic Solids And Cloth

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Mathematics

by

Yizhou Chen

2024

ABSTRACT OF THE DISSERTATION

Novel Efficient Implicit Methods
for Elastic Solids And Cloth

by

Yizhou Chen
Doctor of Philosophy in Mathematics
University of California, Los Angeles, 2024
Professor Joseph Michael Teran, Co-Chair
Professor Chenfanfu Jiang, Co-Chair

Physics-based simulation, coupled with the Finite Element Method (FEM), has emerged as a powerful tool in understanding and analyzing the complex behavior of elastic materials. Implicit Discretization is essential for efficiently and accurately simulating elastic solids and cloth.

In this thesis, we first explore ways of creating volumetric mesh for embedding surface mesh. The embedded surface mesh has many small self intersections. We devise an efficient and robust way of generating a hexahedron mesh to embed the triangle mesh, so that the self-intersecting regions are correctly duplicated. We then simulate the hexahedron mesh using Finite Element Method and interpolate to the embedded triangle mesh.

The second part explores different ways of improving the core simulation solver of Finite Element Method. We improve on the Position Based Dynamics (PBD)/Extended Position Based Dynamics (XPBD) framework. PBD/XPBD are known for their robustness under a very small computational budget. However, they have several limitations. PBD/XPBD

does not converge when the computational budget is sufficient. PBD/XPBD also supports limited hyperelastic constitutive models. We devise PXPBD (Primary Extended Position Based Dynamics) to address these issues. The PXPBD methods improves convergence rate of PBD/XPBD and support arbitrary hyperelastic models.

PBD/XPBD framework does not support quasistatic simulation either. We note quasistatic simulation is important for generating training data for machine learning based simulation approaches such as QNN (Quasistatic Neural Network). We also notice that the constraint-based Gauss-Seidel approach in PBD/XPBD causes loss of information on the node solve. So we design Position-Based Nonlinear Gauss Seidel (PBNG), where the hyperelastic energy from FEM is optimized per node, instead of per constraint. Doing so not only boosts convergence rate, but also enables high quality quasistatic simulation, which runs much faster than the existing methods such as Newton's method.

The last part of my thesis uses a machine learning-based approach to simulate human musculature effectively. We use a neural network to model the deformation of human soft tissues such as muscle, tendon, fat and skin with high fidelity. By deploying a biomechanics-based approach, we estimate the activation of single muscle during deformation. The activation parameters are then incorporated into an active neural network (ANN). which adds per-vertex deformation on top of Linear Blend Skinning (LBS). Our neural network achieves more than 1000X speed up as compared to traditional FEM approaches, but it has the same level of visual fidelity.

The dissertation of Yizhou Chen is approved.

Luminita Aura Vese

Christopher R. Anderson

Chenfanfu Jiang, Committee Co-Chair

Joseph Michael Teran, Committee Co-Chair

University of California, Los Angeles

2024

To my beloved wife and dear parents,

whose love, encouragement and endeavor

have been the foundation of my endeavors

TABLE OF CONTENTS

List of Figures

List of Tables

VITA

2018 B.S. (Mathematics) and B.A. (German) and M.A. (Mathematics), UCLA.

2019–present Teaching Assistant, Mathematics Department, UCLA.

2022–present Physics Programmer Intern, Epic Games, Inc.

PUBLICATIONS

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.

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), 1-25.

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.

Yizhou Chen, Yushan Han, Jingyu Chen, Zhan Zhang, Alex McAdams, and Joseph Teran.2024.

Position-Based Nonlinear Gauss-Seidel for Quasistatic Hyperelasticity. In ACM Transactions of Graphics. Association for Computing Machinery, New York, NY, USA, Article 115, 1-15.

Yushan Han, Yizhou Chen, Carmichael Ong, Jingyu Chen, Jeniffer Hicks and Joseph Teran.2024. A Neural Network Model for Efficient Musculoskeletal-Driven Skin Deformation. In ACM Transactions of Graphics. Association for Computing Machinery, New York, NY, USA, Article 118, 1-12.

CHAPTER 1

Introduction

Physics-based simulation is a crucial aspect of computer graphics. Finite Element Method is a fundamental framework for simulating object deformations efficiently. In this thesis, we explore various aspects of Finite Element Methods including mesh generation, numerical method for accelerating convergence rate and machine learning-based approach to model musculature deformation.

1.1 Grid-Meshing Algorithm for Embedding Self-Intersecting Surfaces

The creation of a volumetric mesh representing the interior of an input polygonal mesh is a common requirement in graphics and computational mechanics applications. Most mesh creation techniques assume that the input surface is not self-intersecting. However, due to numerical and/or user error, input surfaces are commonly self-intersecting to some degree. The removal of self-intersection is a burdensome task that complicates workflow and generally slows down the process of creating simulation-ready digital assets.

We present a method for the creation of a volumetric embedding hexahedron mesh from a self-intersecting input triangle mesh. Our method is designed for efficiency by minimizing use of computationally expensive exact/adaptive precision arithmetic. Although our approach allows for nearly no limit on the degree of self-intersection in the input surface, our focus is on efficiency in the most common case: many minimal self-intersections. The embedding

hexahedron mesh is created from a uniform background grid and consists of hexahedron elements that are geometrical copies of grid cells. Multiple copies of a single grid cell are used to resolve regions of self-intersection/overlap. Lastly, we develop a novel topology-aware embedding mesh coarsening technique to allow for user-specified mesh resolution as well as a topology-aware tetrahedralization of the hexahedron mesh.

The highlights of our method are the following:

- An efficient technique with reduced use of exact/adaptive precision arithmetic for building an embedding hexahedron mesh for an input self-intersecting triangle mesh from a uniform grid that is equivalent to pushing forward one unambiguously defined from a self-intersection-free state.
- A topology aware embedding mesh coarsening strategy to provide for flexible resolution/element count.
- A topology aware BCC approach for converting the embedding hexahedron mesh into an embedding tetrahedron mesh.

1.2 Toward More Accurate and Efficient Simulations of Elastic Solid and Cloth

Various methods have been proposed for solving the FEM-discretized equations of motions for large strain hyperelastic solids and cloth [LGL19, NOB16, BML14, TSI05, GSS15, MMC16, KYT06, ZBK18, ZLB16]. Thorough summaries of the state of the art are given by Zhu et al. [ZBK18] and Li et al. [LGL19]. The preferred approach in a given application generally depends on the relative importance of constitutive accuracy, robustness/stability and computational efficiency. There is no one method that is optimal in all computer graphics as different applications place different relative importance on these considerations. These equations are nonlinear, and an iterative solver must be used to improve the accuracy of

an initial guess by reducing the magnitude of the system residual. 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]. However, it is not always necessary to reduce the residual beyond a few orders of magnitude for satisfactory visual accuracy (see discussion in Liu et al. [LBO13], Bouaziz et al. [BML14], Zhu et al. [ZBK18]). In these cases, Newton's method is often outperformed by alternative techniques. 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] do not require computation of the exact energy Hessian and many of these simplifications leverage direct solvers based on pre-computed matrix factorizations of simplified discrete elliptic operators to allow for reduced per-iteration cost compared to Newton's method.

1.2.1 Primary Extended Position Based Dynamics

The Position Based Dynamics (PBD) approach of Müller et al. [MHH07] is remarkably powerful due to its robust and stable behavior in applications with minimal computational budgets. PBD has gained wide adoption since there are often no other methods that can provide comparably reliable behavior under extreme computation constraints. For elastic materials, PBD uses a constraint view of the material resistance to deformation and is similar to strain limiting [Pro95] and shape matching [MHT05] techniques. In the context of elasticity, this has been shown to be equivalent to a Gauss-Seidel minimization of an elastic potential that is quadratic in the constraints [LBO13, BML14, MMC16]. 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] addresses 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 (or hard strain constraints). In this case, the Lagrange multiplier terms can be interpreted as stress-like and associated with enforcing the constraints (e.g. pressure in an incompressible fluid [BBB07]). With this view, XPBD handles these terms correctly as weak constraints (e.g. as in weakly compressible materials [SSJ14, KSG09]) where PBD can be interpreted as a splitting scheme where non-stress based forces are first integrated, followed by a projection step.

Despite its many strengths, 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, e.g., for many biomechanical soft tissues. Furthermore, while XPBD is based on a Gauss-Seidel procedure for the Lagrange multiplier formulation of the backward Euler equations, it simplifies the system by omitting the Hessian of the constraints and the residual of the primary (position) equations. The omission of the primary equations is perfectly accurate in the first iteration, but as Macklin et al. [MMC16] point out, less so in latter iterations when constraint gradients vary significantly. We observe that this rapid variation occurs for many hyperelastic formulations and that its omission degrades residual reduction. However, the inclusion of this term introduces instabilities into XPBD.

We provide a modification to the XPBD position update that more accurately guarantees that the primary residual is zero and may be omitted. We call our approach Primal Extended Position Based Dynamics (PXPBD). It can be done in two ways. The first (B-PXPBD) uses fixed-point iteration to zero the primary residual after the Gauss-Seidel update of the Lagrange multiplier. The second (FP-PXPBD) is a reformulation of XPBD that allows for arbitrary hyperelastic models. We observe that the constraint Hessians and primary residual terms are exactly zero and can be omitted with no error if the first Piola-Kirchhoff stress [BW08] is used as the auxiliary unknown (in place of the Lagrange multipliers in the original

XPBD). We advocate for two models because each have relative strengths and weaknesses in their resolution of the primary residual omission in XPBD. B-PXPBD can be done with a simple modification to an existing XPBD code, however it requires the use of a blending parameter (see Section 4.1.3.1) since accurate fixed-point iteration is too costly. FP-PXPBD is a larger modification to an existing XPBD code and requires element-wise Newton solves, but it exactly resolves the the issues with both Hessian and residual omission in XPBD. Furthermore, FP-PXPBD allows for arbitrary hyperelastic models while B-PXPBD is based on constraint formulations as with XPBD.

We demonstrate our method with collision-intensive scenarios by applying it to the updated-Lagrangian formulation of hyperelasticity where the simulation mesh is embedded in a regular grid at each time step as in [JSS15]. We summarize our contributions as:

- B-PXPBD: A modification to the XPBD position update that improves residual reduction with hyperelasticity.
- FP-PXPBD: A first Piola-Kirchhoff formulation of the XPBD auxiliary variables that both guarantees zero primal residual for improved total residual reduction and generalizes XPBD to arbitrary hyperelastic models.
- A local affine transformation that decouples strain and translation variables in each FEM element for added efficiency with FP-PXPBD.
- A Sherman-Morrison rank-one quasi-Newton technique for each first Piola-Kirchhoff stress in FP-PXPBD.

1.2.2 Position-Based Nonlinear Gauss Seidel

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]. As noted in [CHC23], simply interpreting the square root of the hyperelastic potential as the constraint results in instability. 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 5.2).

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.

We summarize our contributions as:

- A position-based, rather than constraint-based, nonlinear Gauss-Seidel technique for hyperelastic implicit time stepping.

- A hyperelastic energy density Hessian projection to efficiently guarantee definiteness of linearized equations that does not require a singular value decomposition or symmetric eigen solves.
- A node coloring technique that allows for efficient parallel performance of our Gauss-Seidel updates.
- A novel multiresolution acceleration technique for reducing iteration counts at high resolution.

1.3 A Neural Network Model for Efficient Musculoskeletal-Driven Skin Deformation

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 [LSN13, FLP14, PLF14, MZS11, LST09, SGK18]. 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, LMR23, 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.

More specifically, 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, LMR23, SGO20, JHG22, CMM20]. Specifically, our network learns correctives 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.

CHAPTER 2

Continuum Mechanics and Finite Element Method

In this section, we review some fundamental ideas from continuum mechanics and derive the fundamental equations used for finite element method simulations. The equations are derived from a Lagrangian perspective. The derivations are based on Jiang et al. [JST16].

2.1 Continuum Mechanics

2.1.1 Kinematic Theory

We represent the deformation of the material using the undeformed positions $\mathbf{X}$, its deformed positions $\mathbf{x}$ and the deformation map $\boldsymbol{\phi}(\mathbf{X}, t)$. We usually call $\mathbf{x}$ world positions and $\mathbf{X}$ material positions. More specifically, at a given time t we have $\mathbf{x} = \boldsymbol{\phi}(\mathbf{X}, t)$. Differentiate with time we get:

$$\mathbf{V}(\mathbf{X}, t) = \frac{\partial \boldsymbol{\phi}}{\partial t}(\mathbf{X}, t) \tag{2.1}$$

$$\mathbf{A}(\mathbf{X}, t) = \frac{\partial^2 \boldsymbol{\phi}}{\partial t^2}(\mathbf{X}, t) = \frac{\partial \mathbf{V}}{\partial t}(\mathbf{X}, t) \tag{2.2}$$

where $\boldsymbol{\phi} : \Omega^0 \to \Omega^t; \Omega^0, \Omega^t \subset \mathbf{R}^d, d = 2$ or 3 is the dimension of the space. Let $\mathbf{v}(\mathbf{x}, t) = \mathbf{V}(\boldsymbol{\phi}^{-1}(\mathbf{x}, t), t)$ be the velocity in world space.

2.1.2 Deformation Gradient

We define the deformation gradient as the Jacobian of the deformation map $\boldsymbol{\phi}$ with respect to the material coordinates $\mathbf{X}$. It is useful in elastic simulations because most of the hyperelastic

models use this term for energy computation. We define deformation gradient $\mathbf{F}$ as:

$$\mathbf{F}(\mathbf{X},t) = \frac{\partial \boldsymbol{\phi}}{\partial \mathbf{X}}(\mathbf{X},t) = \frac{\partial \mathbf{x}}{\partial \mathbf{X}}(\mathbf{X},t) \tag{2.3}$$

In the discrete form $\mathbf{F}$ is a 2×2 matrix or 3×3 matrix depending on if the simulation is in 2D or 3D. $\mathbf{F}$ can also be a 3×2 matrix if the cloth is simulated because the material space of cloth is essentially 2D, and the object is being simulated in 3D. We define $J(\mathbf{X},t) = det(\mathbf{F})$ to be the determinant of $\mathbf{F}$. In the case $\mathbf{F}$ is 3×2, we define $\mathrm{J} = \sqrt{det(\mathbf{F}^T\mathbf{F})}$.

2.1.3 First Piola-Kirchoff Stress

For hyperelastic materials, the energy density Ψ is usually a function of deformation gradient $\mathbf{F}$. We define first Piola-Kirchoff stress to the derivative $\mathbf{P} = \frac{\partial \Psi}{\partial \mathbf{F}}$. Note that $\mathbf{P}$ is a matrix of the same dimension as $\mathbf{F}$. It is common in engineering literatures to relate $\mathbf{P}$ with Cauchy stress σ through the relationship $\sigma = \frac{1}{\mathrm{J}}\mathbf{P}\mathbf{F}^T$. Or in other words, $\mathbf{P} = \mathrm{J}\sigma\mathbf{F}^{-T}$.

2.1.4 Governing Equations

We derive the governing equations based on conservation of mass and conservation of momentum.

2.1.4.1 Conservation of Mass

Let $\mathrm{R}(\mathbf{X},t)$ be the density of the material at position $\mathbf{X}$ at time t and $\rho(\mathbf{x},t)$ be the density at world space such that $\rho(\boldsymbol{\phi}(\mathbf{X},t),t) = \mathrm{R}(\mathbf{X},t)$. For any particle $\mathbf{X}$ in the material space, given a ball B_ϵ^t or radius ϵ around $\mathbf{x}(\mathbf{X},t)$, the conservation of mass states that the mass of the ball B_ϵ^t is constant through out time. In other words,

$$\int_{\mathrm{B}_\epsilon^0} \mathrm{R}(\mathbf{X},0)d\mathbf{X} = \int_{\mathrm{B}_\epsilon^t} \rho(\mathbf{x},t)d\mathbf{x} = \int_{\mathrm{B}_\epsilon^0} \mathrm{R}(\mathbf{X},t)\mathrm{J}(\mathbf{X},t)d\mathbf{X} \tag{2.4}$$

Take limit of $\epsilon \to 0$ then we have $\mathrm{R}(\mathbf{X},0) = \mathrm{R}(\mathbf{X},t)\mathrm{J}(\mathbf{X},t)$.

2.1.4.2 Conservation of Momentum

We first define the traction field $\mathbf{t}(\cdot, t, \mathbf{n}) : \Omega^t \to \mathbb{R}^d$. Consider again the ball B_ϵ^t at time t. The force on the surface is $\int_{\partial \mathrm{B}_\epsilon^t} \mathbf{t}(\mathbf{x}, \mathbf{n}(\mathbf{x}))ds$. It can be shown that there is a matrix $\sigma(\mathbf{x}, t) \in \mathbb{R}^{d\times d}$ such that $\mathbf{t}(\mathbf{x}, \mathbf{n}(\mathbf{x})) = \sigma(\mathbf{x}, t)\mathbf{n}$. By Newton's second law we have the following:

$$\int_{\partial \mathrm{B}_\epsilon^t} \mathbf{t}(\mathbf{x}, \mathbf{n}(\mathbf{x}))ds + \int_{\mathrm{B}_\epsilon^t} \mathbf{f}^{\mathrm{ext}} d\mathbf{x} = \frac{d}{dt}\int_{\mathrm{B}_\epsilon^t} \rho(\mathbf{x}, t)\mathbf{v}(\mathbf{x}, t)d\mathbf{x} \tag{2.5}$$

$$= \frac{d}{dt}\int_{\mathrm{B}_\epsilon^0} \mathrm{R}(\mathbf{X}, t)\mathbf{V}(\mathbf{X}, t)Jd\mathbf{X} \tag{2.6}$$

$$= \frac{d}{dt}\int_{\mathrm{B}_\epsilon^0} \mathrm{R}(\mathbf{X}, 0)\mathbf{V}(\mathbf{X}, t)d\mathbf{X} \tag{2.7}$$

$$= \int_{\mathrm{B}_\epsilon^0} \mathrm{R}(\mathbf{X}, 0)\mathbf{A}(\mathbf{X}, t)d\mathbf{X} \tag{2.8}$$

For the left hand side of the equation, we change coordinates to material space:

$$\int_{\partial \mathrm{B}_\epsilon^t} \mathbf{t}(\mathbf{x}, \mathbf{n}(\mathbf{x}))ds + \int_{\mathrm{B}_\epsilon^t} \mathbf{f}^{\mathrm{ext}} d\mathbf{x} = \int_{\partial \mathrm{B}_\epsilon^0} \mathrm{J}(\mathbf{X}, t)\sigma(\boldsymbol{\phi}(\mathbf{X}, t), t)\mathbf{F}^{-T}(\mathbf{X}, t)\mathbf{N}ds + \int_{\mathrm{B}_\epsilon^0} \mathbf{F}^{\mathrm{ext}}\mathrm{J}(\mathbf{X}, t)d\mathbf{X} \tag{2.9}$$

$$= \int_{\partial \mathrm{B}_\epsilon^0} \mathbf{P}(\mathbf{X}, t)\mathbf{N}ds + \int_{\mathrm{B}_\epsilon^0} \mathbf{F}^{\mathrm{ext}}\mathrm{J}(\mathbf{X}, t)d\mathbf{X} \tag{2.10}$$

$$= \int_{\mathrm{B}_\epsilon^0} \nabla^{\mathbf{X}}\mathbf{P}(\mathbf{X}, t) + \mathbf{F}^{\mathrm{ext}}\mathrm{J}(\mathbf{X}, t)d\mathbf{X} \tag{2.11}$$

where $\mathbf{F}^{\mathrm{ext}}(\mathbf{X}, t) = \mathbf{f}^{\mathrm{ext}}(\boldsymbol{\phi}(\mathbf{X}, t), t)$. Then we have the equation $\mathrm{R}(\mathbf{X}, 0)\mathbf{A}(\mathbf{X}, t) = \nabla^{\mathbf{X}}\mathbf{P}(\mathbf{X}, t) + \mathbf{F}^{\mathrm{ext}}\mathrm{J}(\mathbf{X}, t)$.

CHAPTER 3

A Robust Grid-Based Meshing Algorithm for Embedding Self-Intersecting Surfaces

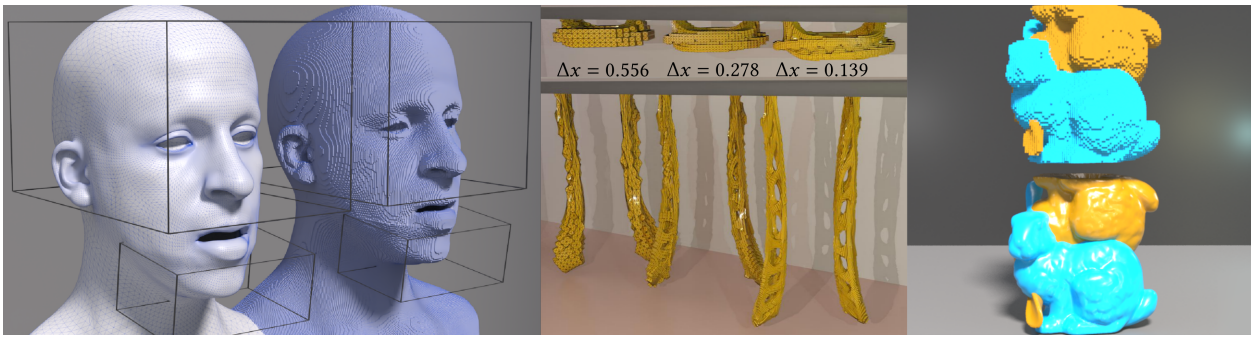
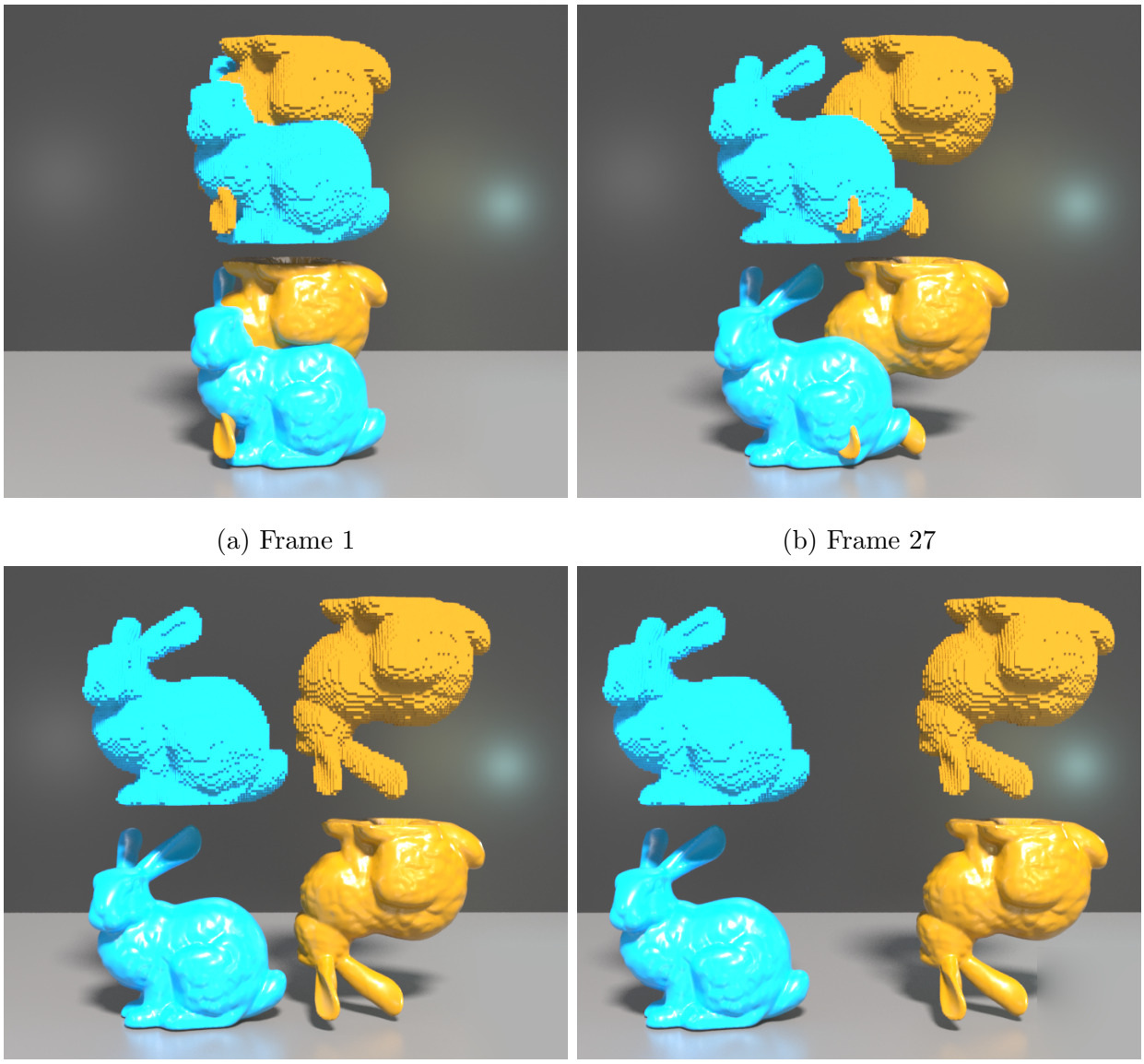


Figure 3.1: *(Left)* Our method can generate a consistent volumetric mesh for a facial geometry that contains self-intersections e.g. around the lips. *(Middle)* Two interlocking Möbius-strip-like bands separate freely at various spatial resolutions of the background grid, despite many near self-intersections in the surface geometry. *(Right)* Two bunny geometries can naturally separate despite significant initial overlaps.

3.1 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 3.8 for examples).



(a) Frame 1

(b) Frame 27

(c) Frame 54

(d) Frame 81

Figure 3.2: 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.

We briefly summarize the three main stages of our algorithm, as detailed in Figure 3.4. In the first stage, volumetric extension (Section 3.3), 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.

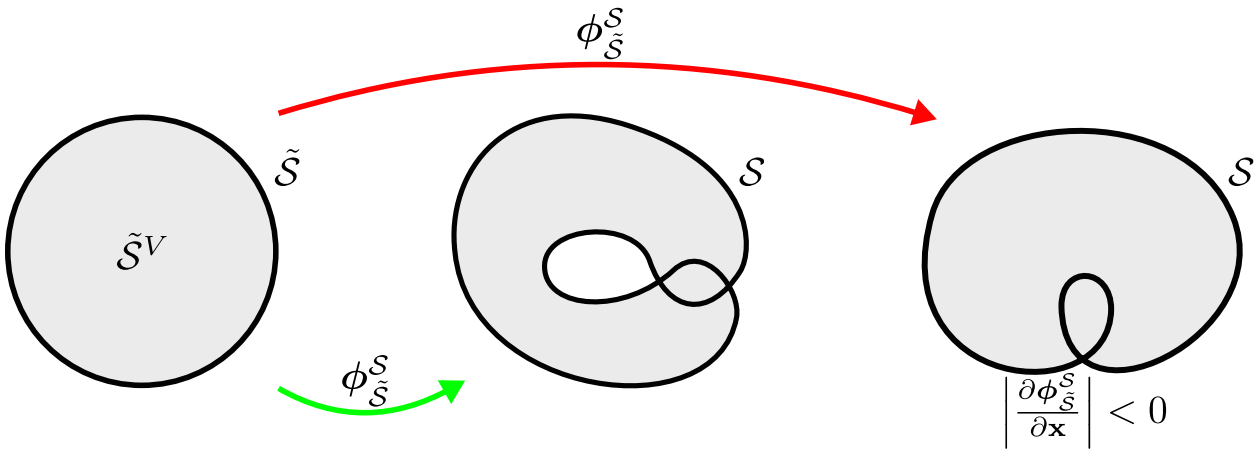


Figure 3.3: **Intersection-free mapping.** Two mappings from a non-self-intersecting region $\tilde{\mathcal{S}}^V$ to self-intersecting boundary $\mathcal{S}$ are shown. The second mapping (right) requires the existence of a negative Jacobian determinant.

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 3.4). 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 3.5), interior extension regions meshes $\mathcal{V}^{j^I,c}$ are sewn together and into the volumetric extension $\mathcal{V}^S$ to produce the final output mesh. We additionally provide a coarsening approach in Section 3.6 to provide user control over the embedding mesh resolution as well as a topologically-aware technique for converting the hexahedron mesh $\mathcal{V}$ into a tetrahedron mesh $\mathcal{T}$.

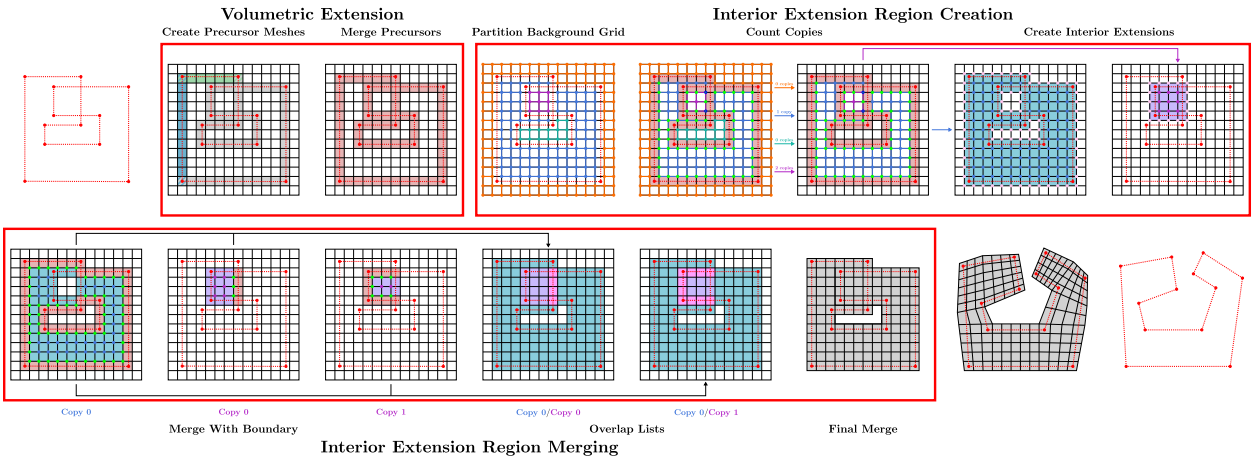


Figure 3.4: **Algorithm overview.** Given an initial input surface mesh $\mathcal{S}$, there are three major steps in the computation of the final volumetric extension mesh $\mathcal{V}$: Volumetric Extension, Interior Extension Region Creation, and Interior Extension Region Merging. *(Volumetric Extension)* 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.

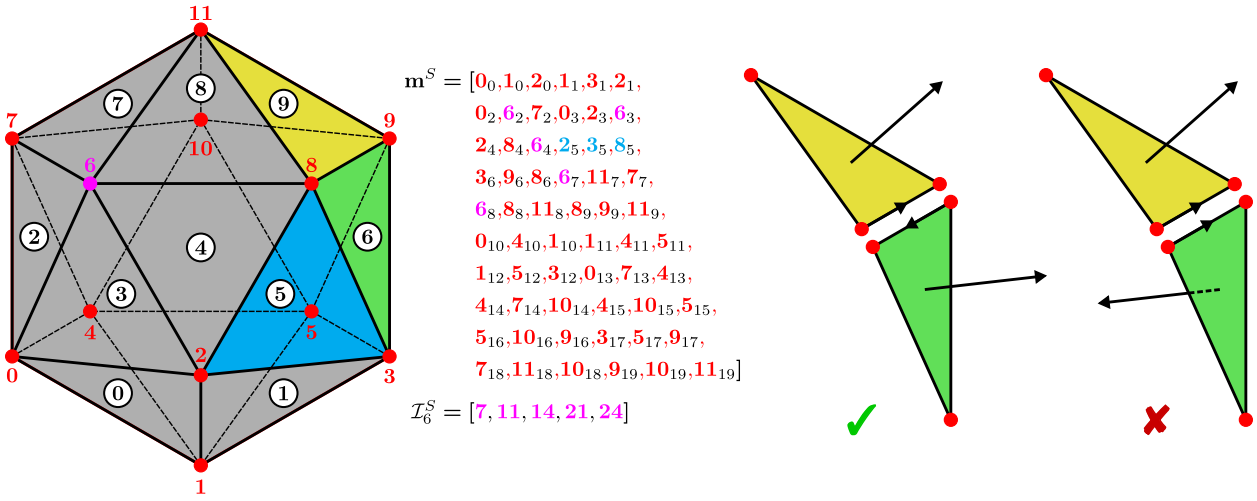


Figure 3.5: **Mesh conventions.** *(Left)* A sample triangle mesh is shown, along with the vector $\mathbf{m}^S$. The incident elements $\mathcal{I}_6^S$ for vertex 6 are also shown. The first 10 faces, visible from the front, have been labeled on the mesh. *(Right)* The left pair of triangles are consistently 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.

3.2 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 $t_{\lfloor \frac{j}{3} \rfloor}^S$, $0 \leq \lfloor \frac{j}{3} \rfloor < N_e^S$. For example, for the mesh $\mathcal{S}$ in Figure 3.5, triangle t_5^S is made up of vertices $\mathbf{x}_{m_j^S}^S$ 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 3.5 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 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

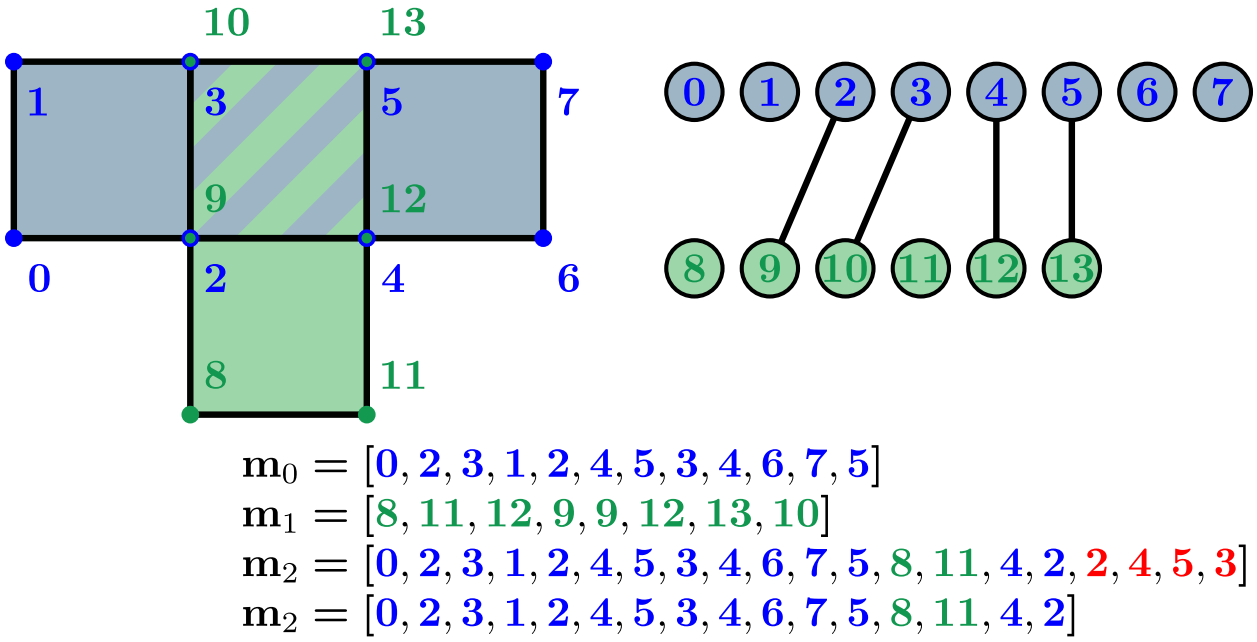


Figure 3.6: **Mesh merge.** An example of two meshes merging together. Vertices 2, 3, 4 and 5 merge with vertices 9, 10, 12 and 13, respectively. A new vector $\mathbf{m}_2$ is created to hold all of the hexahedron vertices post-merge, and the extra hexahedron (in red) is then removed.

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.

3.2.1 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 3.6.

3.3 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 $\boldsymbol{\phi}^S_{\tilde{S}}$.

3.3.1 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 $\mathcal{V}^S_e = (\mathbf{x}^V, \mathbf{m}^{V^S}_e)$ associated with each triangle t^S_e in $\mathcal{S}$. Note that all precursor meshes

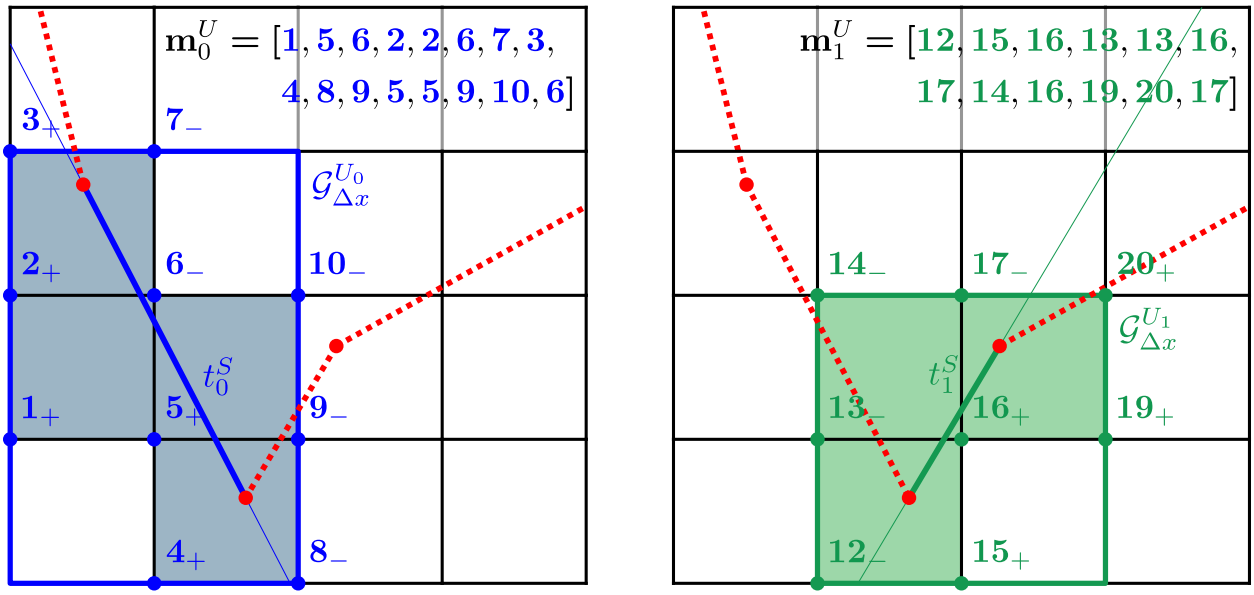


Figure 3.7: **Precursor meshes.** *(Left)* Surface element t_0^S creates quadrilateral mesh $\mathcal{V}_0^S$. *(Right)* Surface element t_1^S creates quadrilateral mesh $\mathcal{V}_1^S$. Each element creates copies of the grid cells it intersects by introducing new vertices which are geometrically coincident to grid nodes.

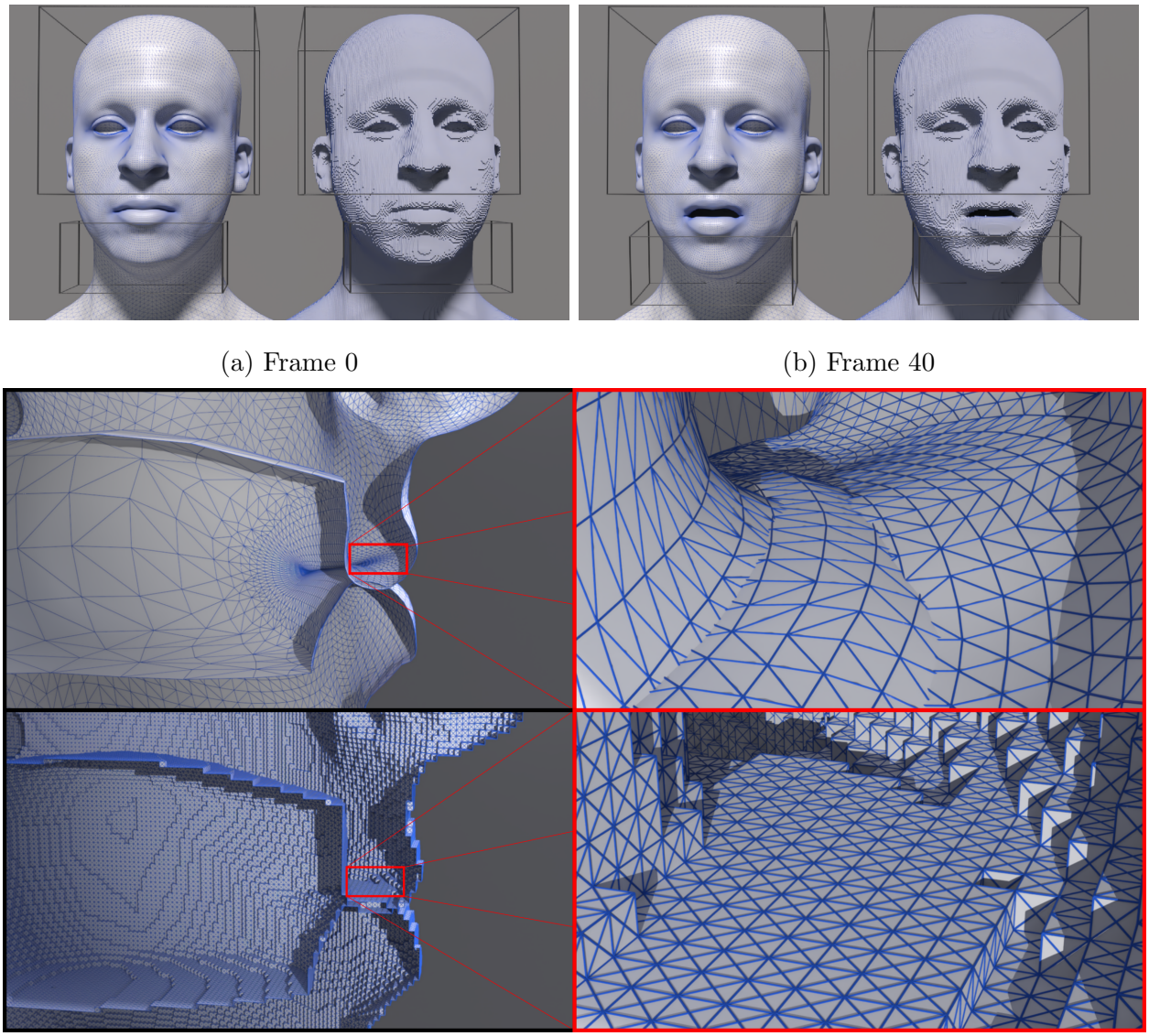
share the common vertex array $\mathbf{x}^V$ and that this process begins its evolution to the final $\mathcal{V}$ vertex array. For each triangle t_e^S in $\mathcal{S}$, we define a hexahedron mesh from the subgrid $\mathcal{G}_{\Delta x}^{V_e^S}$ of $\mathcal{G}_{\Delta x}$ defined by the grid-cell-aligned 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 3.7. 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 3.3.2; however, in many cases, they are equal to the final signs, and their comparably-low computational cost improves overall algorithm performance

3.3.2 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 3.2.1. 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 3.16). In other words, all geometrically coincident hexahedra in element precursor meshes associated with triangles that share a common vertex are merged (see Figure 3.9). 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 3.10), 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 $\mathbf{x}_i^V$ in the volumetric extension $\mathcal{V}^S$ is associated with some collection of precursor meshes $\mathcal{V}_{e_i}^S$ where $\mathbf{x}_i^V$ was created in the merge of vertices in the $\mathcal{V}_{e_i}^S$. This defines a local patch S_{i^V} of surface triangles $t_{e_i}^S$ in $\mathcal{S}$ associated with $\mathbf{x}_i^V$. When propagating signs from $\mathcal{S}$ to $\mathbf{x}_i^V$, only



(a) Frame 0

(b) Frame 40

(c) Interior view of lips

Figure 3.8: A face surface with self-intersecting lips is successfully meshed. The right-hand side of each of the first two frames shows the deformed hexahedron mesh, while each left-hand side shows the corresponding surface mesh. The wireframe boxes represent Dirichlet boundary condition regions. In the bottom four subfigures, lip intersection is visualized in the input surface and subsequent hexahedron mesh.

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 $\mathbf{x}_i^V$ 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 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_{iV} to $\mathbf{x}_i^V$. The closest facet calculation is performed by first storing S_{iV} 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_{iV} (if they are not already in it). If more triangles were added, we recompute the closest mesh facet. We illustrate this process in Figures 3.10 and 3.11. 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.

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.

3.4 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 in Section 3.2.1), 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 3.12. 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 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 3.13. 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 3.14. 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 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 3.5. 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 3.6) 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 $\mathcal{V}_{\mathbf{i}}^{j^I,0} = (\mathbf{x}^V, \mathbf{m}_{\mathbf{i}}^{Vj^I,0})$. Here $\mathbf{x_i}$ 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 $\mathbf{x_i}$, $\mathbf{m}_{\mathbf{i}}^{Vj^I,0}$ 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 3.2.1 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 3.15. We reemphasize that as described in Section 3.2.1 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.

3.5 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.

3.5.1 Merge With Boundary

Recall from Section 3.4 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 3.17, 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 3.2.1. 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 3.16. 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 3.4. 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 $\mathbf{x_j}$ 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 3.17. 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 3.2.1 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 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 3.18.

3.5.2 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

$$m^{V^{j_0^I,c_0}}_{8h_{e_0}^{V^{j_0^I,c_0}}+i^e} = m^{V^{j_1^I,c_1}}_{8h_{f_0}^{V^{j_1^I,c_1}}+i^e} = m^{V^S}_{8h_{g_0}^{V^S}+i^e},\ i^e \in \{0,1,\ldots,7\}\,. \tag{3.1}$$

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 $\mathbf{s}_0^{\mathbf{q}} = (h_{e_0}^{V^{j_0^I,c_0}}, h_{f_0}^{V^{j_1^I,c_1}})$ as a seed between the pair of region copies. Furthermore, we use $\mathbf{p}^{\mathbf{q}} = [\mathbf{s}_0^{\mathbf{q}}, \ldots, \mathbf{s}_{N_s^{\mathbf{q}}-1}^{\mathbf{q}}]$ 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 $\mathbf{p}^{\mathbf{q}}$ by first marking background grid cells geometrically coincident with hexahedra in the seeds as being visited. Then, starting with the seed $\mathbf{s}_0^{\mathbf{q}}$, 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 $\mathbf{p}^{\mathbf{q}}$ 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, $\mathbf{p}^{\mathbf{q}}$ is a list of overlapping hexahedra that will be used to sew the regions together. We illustrated this process in Figure 3.19.

3.5.3 Deduplication

As mentioned in Section 3.4, the number of copies is generally an over count. We use the overlap lists $\mathbf{p^q}$ 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 $\mathbf{p^q}$ is a copy of h_e^S (i.e. it uses the same vertices in $\mathbf{x}^S$ as in Equation (3.1)), 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 $\mathbf{p^q}$ are both copies of hexahedra from $\mathcal{V}^S$ in accordance with Equation (3.1), 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 3.5.1. This process is shown in Figure 3.20.

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 3.5.1 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.

3.5.4 Final Merge

We now merge the vertices of $\mathbf{x}^V$ using the pattern of Section 3.2.1 with adjacencies defined by the overlap lists. For each seed $\mathbf{s}$ in an overlap list, the geometrically coincident nodes of the two hexahedra in $\mathbf{s}$ 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 3.2.1 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.

3.6 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 3.2.1 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

$$m^M_{8h_0^M+k_{0\alpha}^V} = m^M_{8h_1^M+k_{1\alpha}^V},\ \alpha \in \{0,1,2,3\}\,. \tag{3.2}$$

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 procedure is illustrated in Figure 3.21.

3.7 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 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 3.22.

3.8 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 3.1. All experiments were run on a workstation with a single Intel® Core™ i9-10980XE CPU at 3.00GHz.

3.8.1 2D Examples

3.8.1.1 Single Overlap

Figure 3.23 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.

3.8.1.2 Ribbon

Our algorithm can also handle more complex self-intersections. In Figure 3.24, 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.

3.8.1.3 Face

Figure 3.25 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 of multiple disconnected components. Our method successfully treats cases like these by design.

3.8.2 3D Examples

3.8.2.1 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 3.26, 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.

Table 3.1: 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

3.8.2.2 Double Möbius

Figure 3.27 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 3.28). The coarsest 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 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.

3.8.2.3 Twin Bunnies

Another standard example is the Stanford bunny. Figure 3.2 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.

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

3.8.2.4 Dragon

The most complicated geometry we test our method on is the dragon[2] shown in Figure 3.29 (and also shown in Figure 3.11). 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.

3.8.2.5 Fancy Ball

Figure 3.30 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.

3.8.2.6 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 head geometry in Figure 3.8. 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.

3.8.2.7 Collection

Various objects from 3D examples are dropped in a tank in Figure 3.31. The objects naturally deform and collide without meshing or simulation issues.

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

[3] "Abstract object" by sonic art.

3.8.2.8 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 3.32 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).

3.9 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 3.3 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.

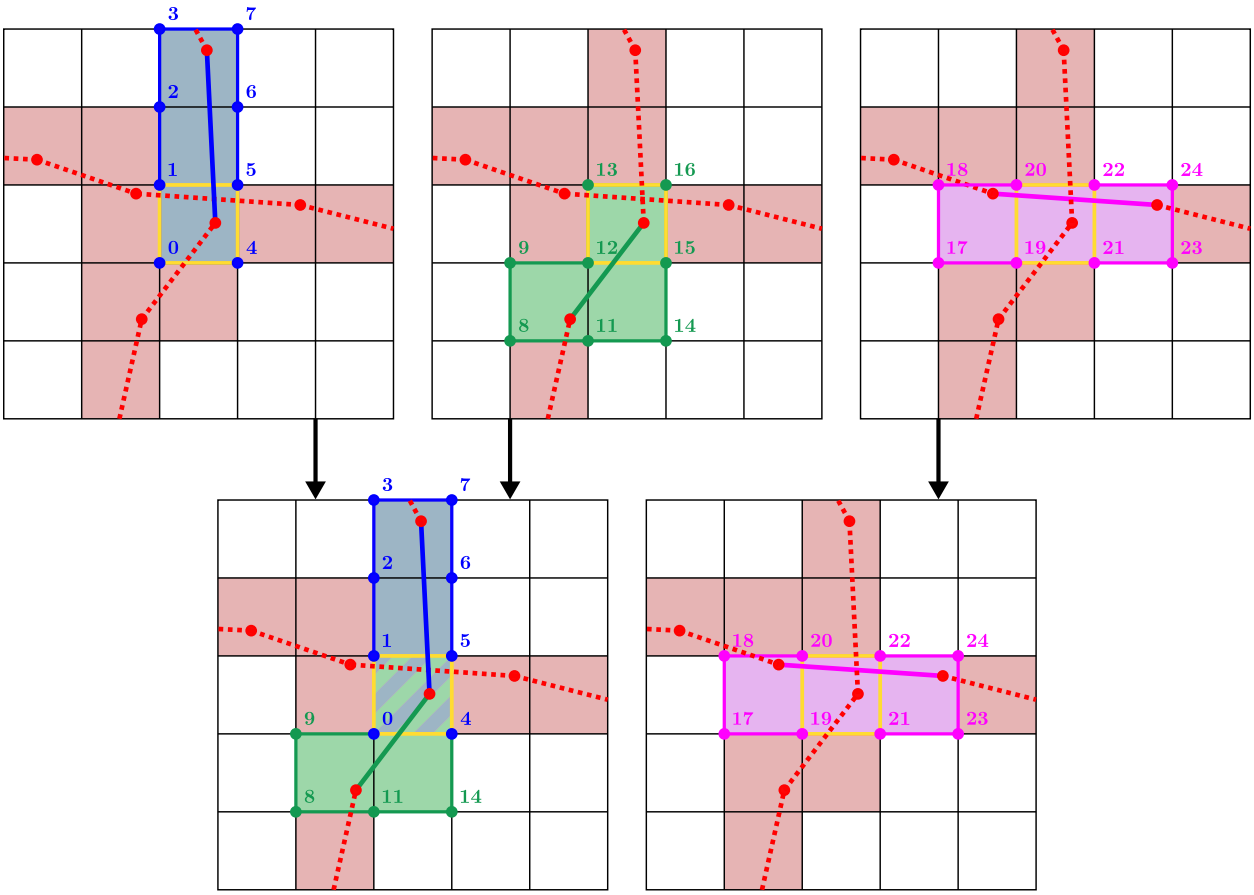


Figure 3.9: **Precursor merge.** The 12 vertices bordering the cell marked in yellow are merged into 8 resulting vertices. Blue vertices 0, 1, 4, 5 and green vertices 12, 13, 15, 16 are merged, respectively. However, magenta vertices 19, 20, 21, 22 do not merge with the blue or green vertices since their associated surface element is topologically distant.

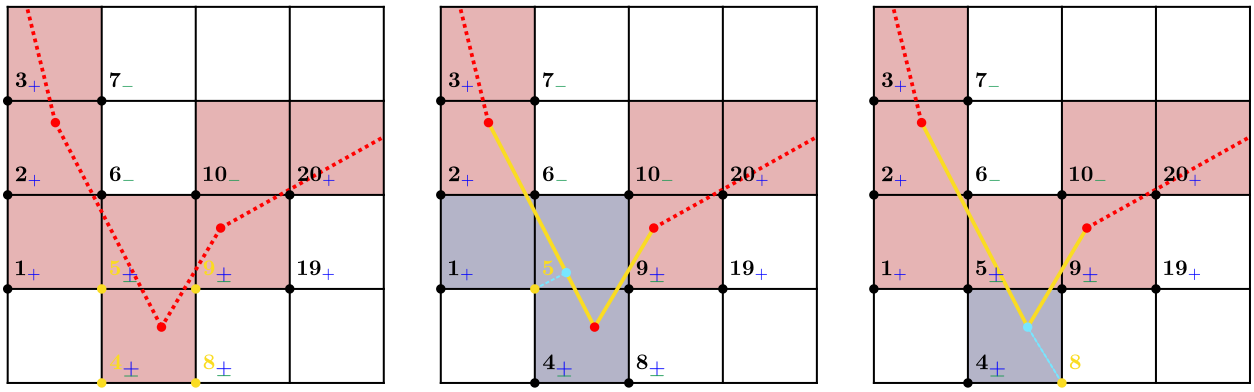


Figure 3.10: **Closest facet.** *(Left)* The four vertices in yellow all have ambiguous signs. *(Middle)* To sign vertex 5, we generate the local patch S_{5^V}, which are the segments shown in yellow. The closest facet (indicated in cyan) lies on a face. *(Right)* A similar process is illustrated for vertex 8, but here the closest facet is a vertex.

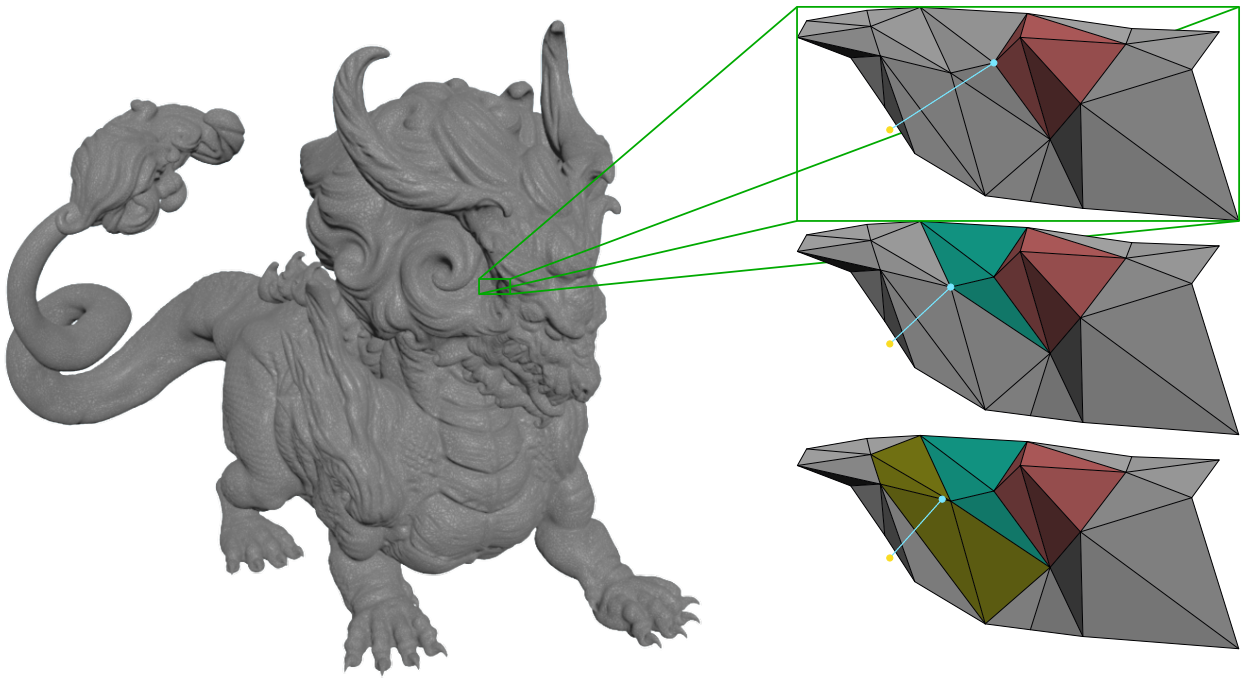


Figure 3.11: **Patch expansion.** The local patch S_{iv} corresponding to the yellow vertex is shown. The initial patch is indicated in red, and the closest facet is a vertex of the red patch. We add the missing incident triangles (turquoise) and recompute the closest facet. This is again a vertex with 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.

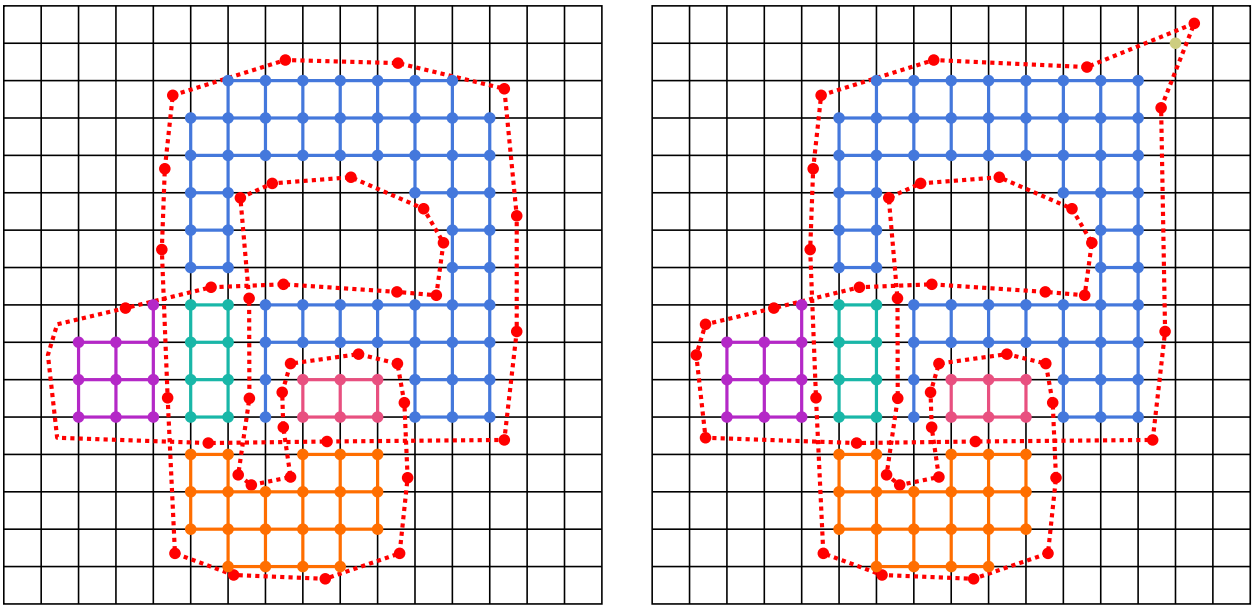


Figure 3.12: **Region over-count.** As the process of partitioning the grid only uses connectivity based on grid edges, it is possible for a contiguous region to be split into multiple regions. Shifting some of the vertices of $\mathcal{S}$ on the left results in the geometry on the right, which contains an additional region in the upper right corner since no edge connects this grid node to the larger blue region.

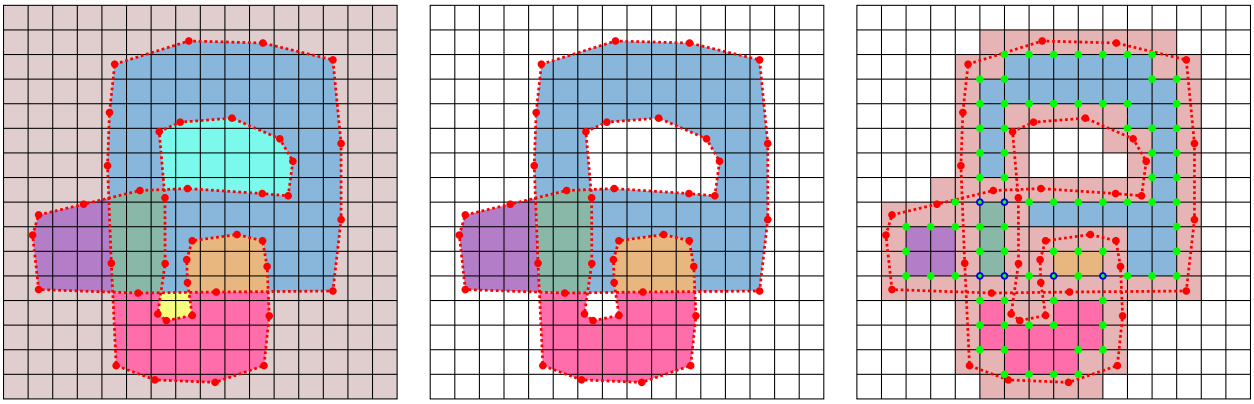


Figure 3.13: **Connected regions.** *(Left)* The surface partitions the background grid into contiguous regions. *(Middle)* The exterior regions are removed. *(Right)* The volumetric extension $\mathcal{V}^S$ is shown, along with the negatively signed vertices in green. Multiple geometrically coincident vertices are indicated using blue circles with green centers.

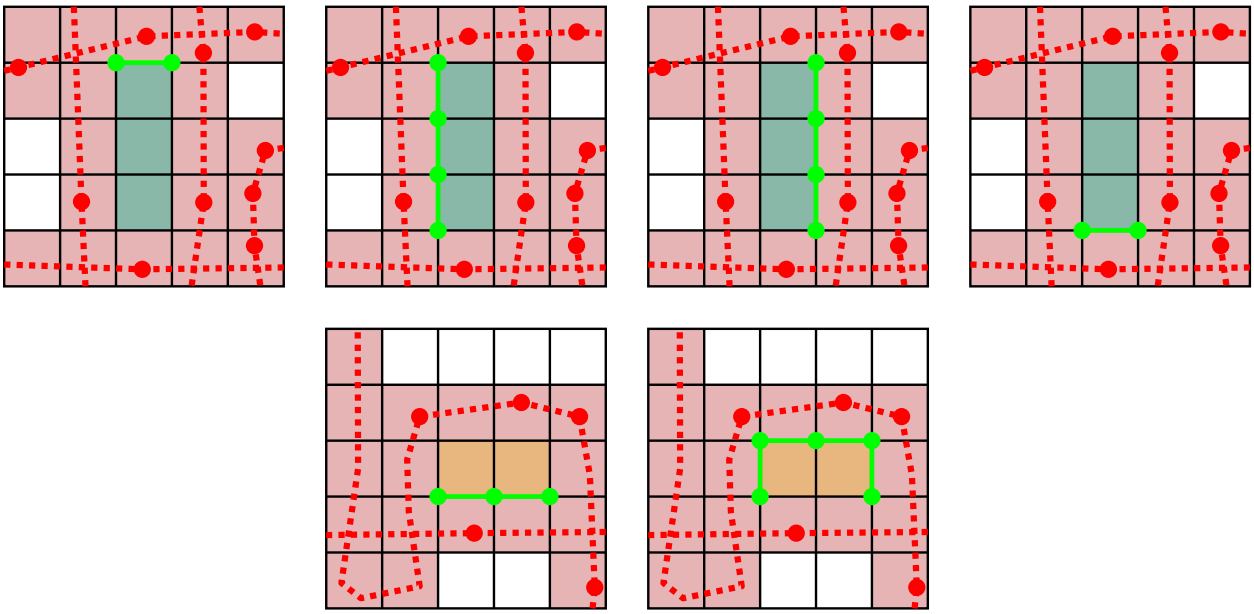


Figure 3.14: **Copy counting.** The two regions from Figure 3.13 having multiple copies are shown. Each copy is displayed with its corresponding connected component of vertices with negative sign.

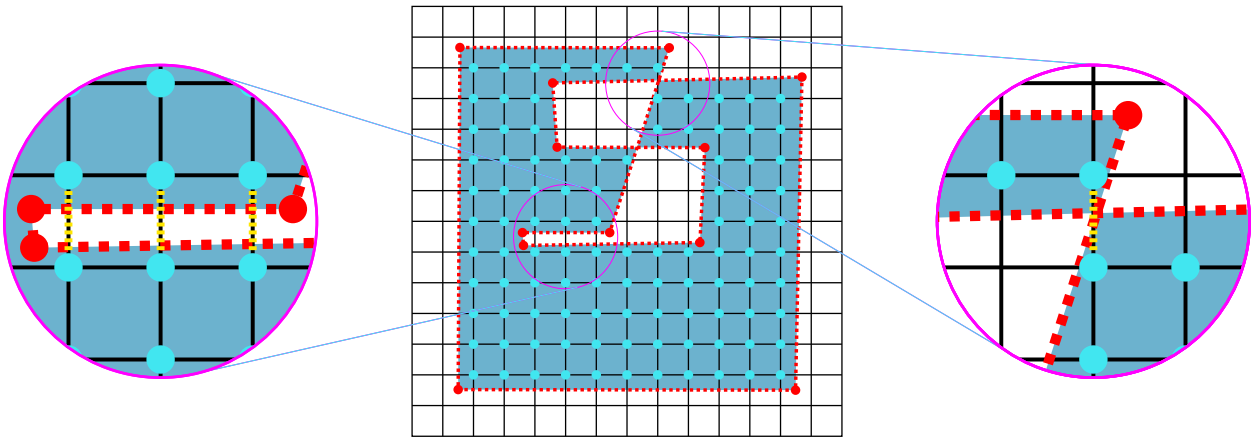


Figure 3.15: **Edge cut criterion.** Grid nodes $\mathbf{x_i}$ of a region are shown, along with two examples showing that adjacent grid nodes may have their common edge cut by a triangle (cut edges are indicated by the dashed yellow lines). In this case, adjacencies are not built between the corresponding vertices in $\mathcal{V}_{\mathbf{i}}^{j^I,0}$ to avoid unwanted sewing.

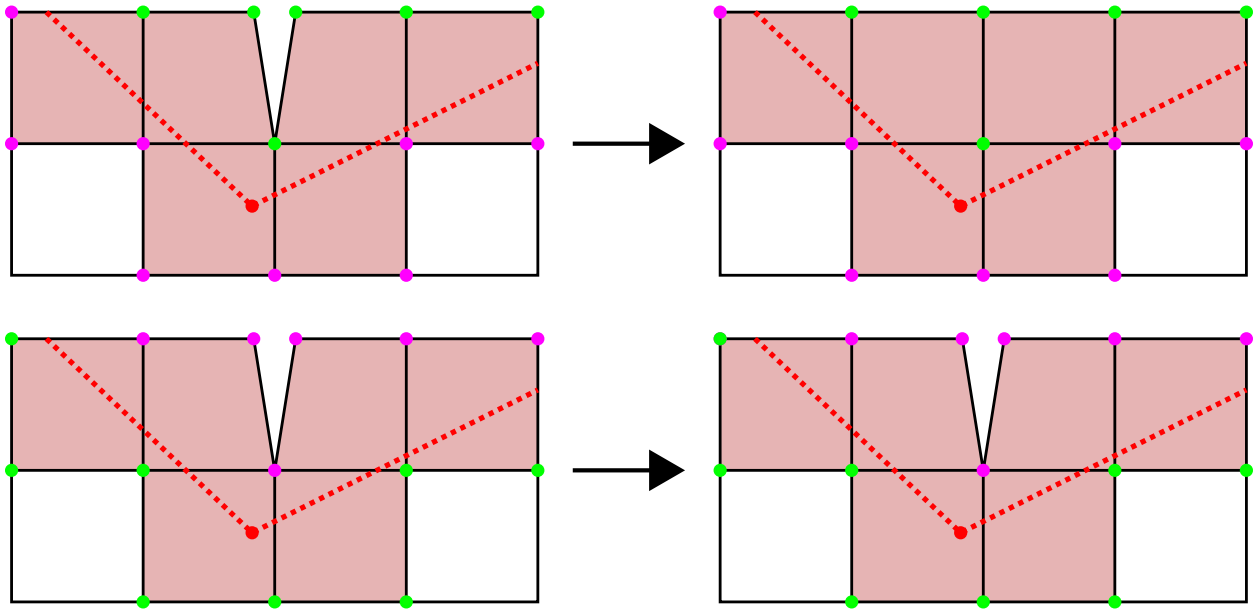


Figure 3.16: **Preliminary merge.** The construction of the volumetric extension $\mathcal{V}^S$ may result in geometrically coincident vertices which do not come from topologically distant parts of the mesh. Green vertices have negative signs, while purple vertices have positive sign. Above: The process in Section 3.5.1 merges these vertices into a single vertex. Below: We do not merge coincident positive vertices, to avoid unnecessarily sewing the exterior.

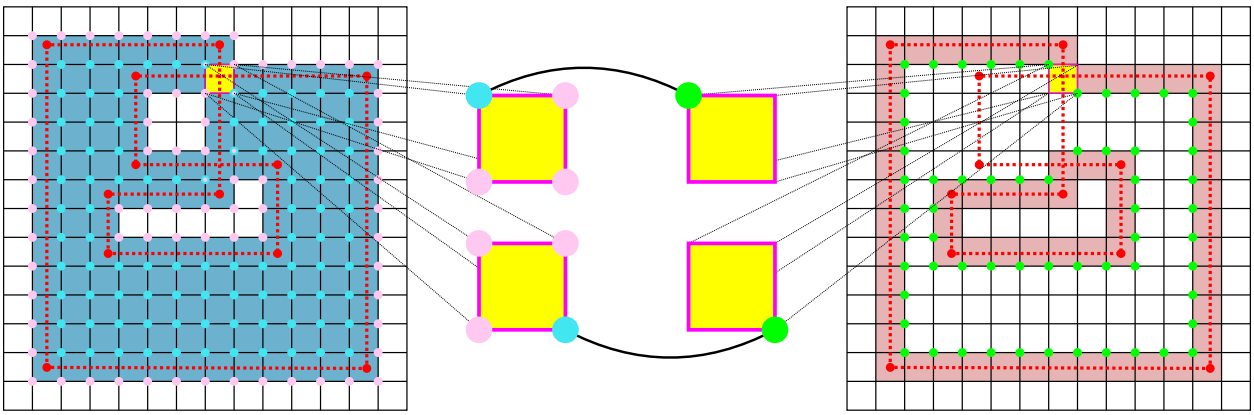


Figure 3.17: **Vertex adjacency.** The merge process between vertices of $\mathcal{V}^{j^I,c}$ and $\mathcal{C}_c^{j^I}$. For the cell highlighted in yellow, there are 2 hexahedra from $\mathcal{V}^{j^I,c}$ and therefore 4 pairs of geometrically coincident vertices. The two negatively signed vertices (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).

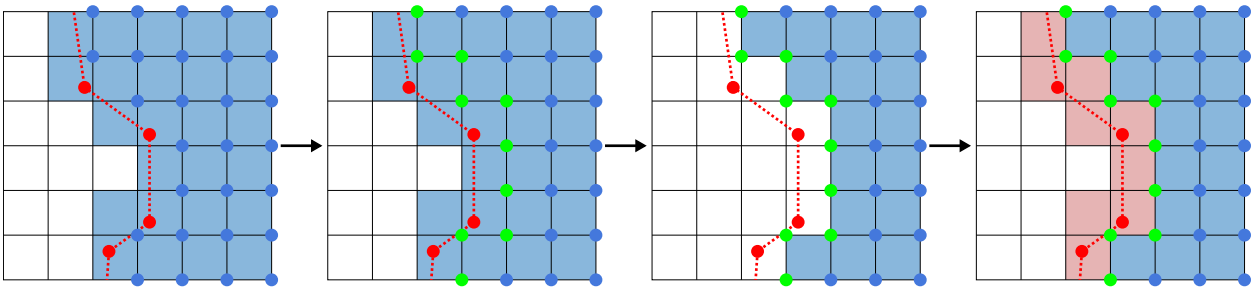


Figure 3.18: **Merge with boundary.** We illustrate the process of Section 3.5.1 following the preliminary merge of negatively signed vertices. First, specific vertices of $\mathcal{V}^{j^I,c}$ are merged with vertices of $\mathcal{C}_c^{j^I}$. Next, hexahedra to be replaced are removed from the $\mathcal{V}^{j^I,c}$. Finally, copies of hexahedra from $\mathcal{V}^S$ are added to this mesh.

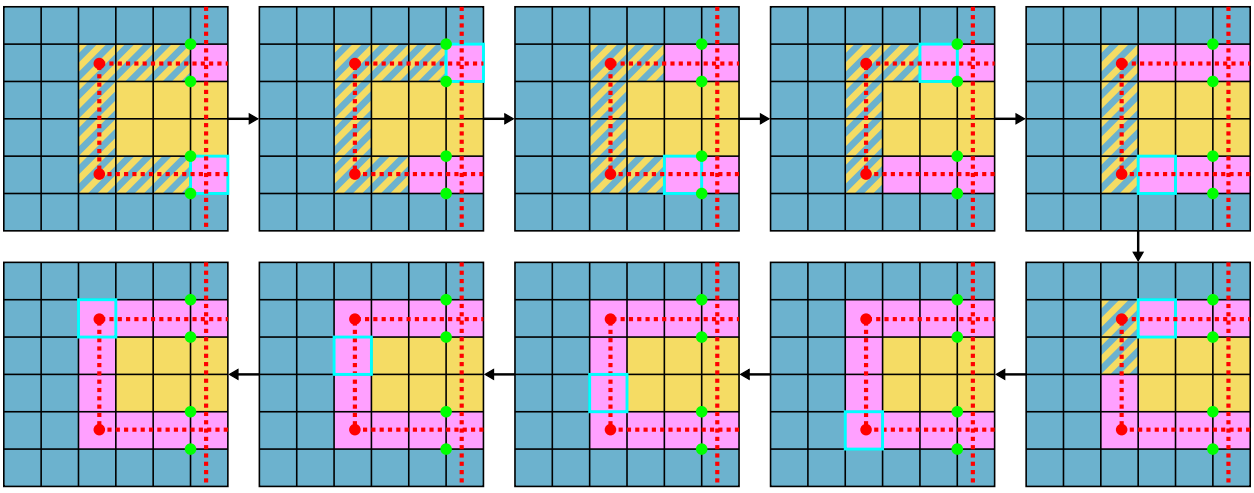


Figure 3.19: **Overlap lists.** A closeup of the overlap region from the geometry of Figure 3.17 is shown here. At the upper left, the seeds for the overlap between the two copies are shown in purple, as well as the incident negative vertices (green) to the seeds from each copy. At each step, the current 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.

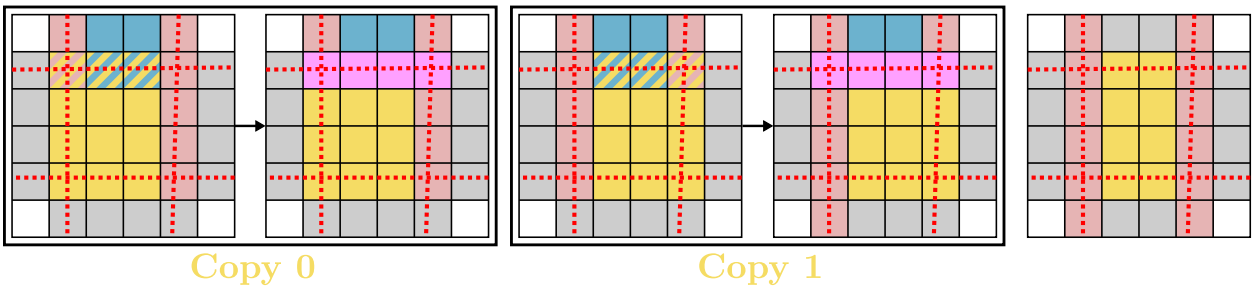


Figure 3.20: **Deduplication.** We show two of the four copies of the central region (yellow), corresponding to the right and left segments of $\mathcal{V}^S$. Each of copies 0 and 1 create an overlap list with the upper region (blue). The overlap list for copy 0 creates a pair between a non-boundary 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.

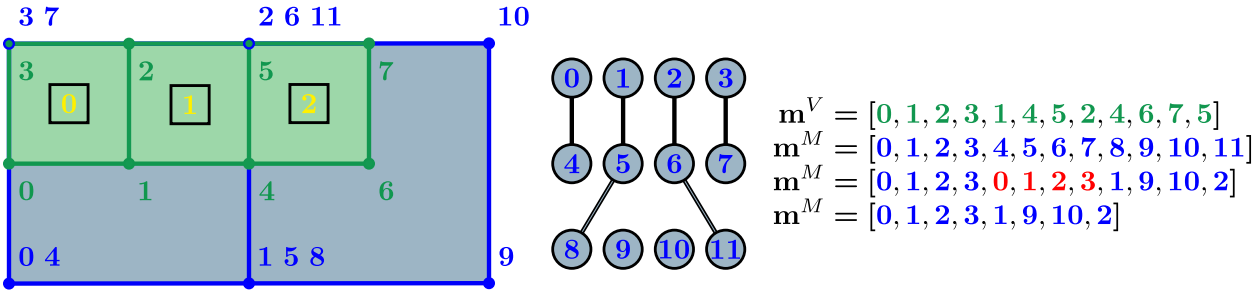


Figure 3.21: **Coarsening.** An example of fine mesh connections. Hexahedra 0 and 1 are totally connected, while hexahedra 1 and 2 are connected by a face. After merging the vertices of the coarse mesh (blue), the duplicated hexahedron (indicated in red) is removed.

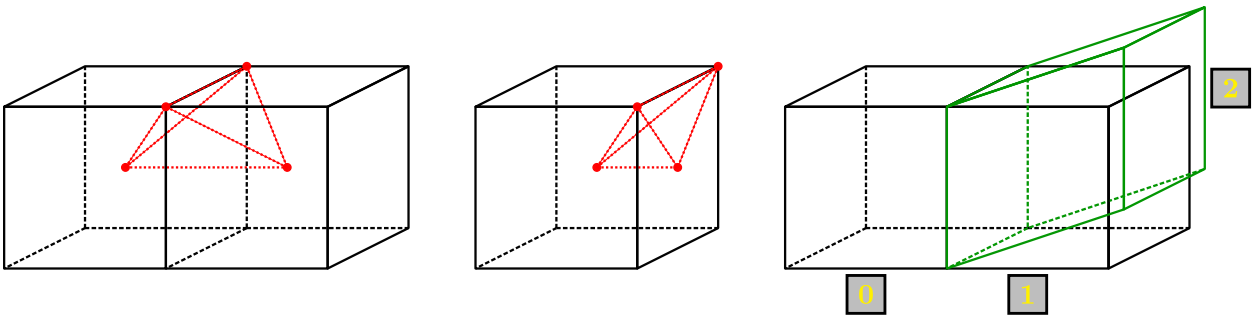
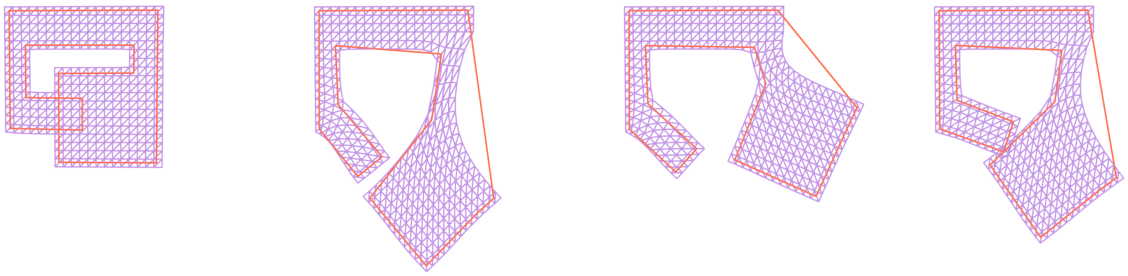
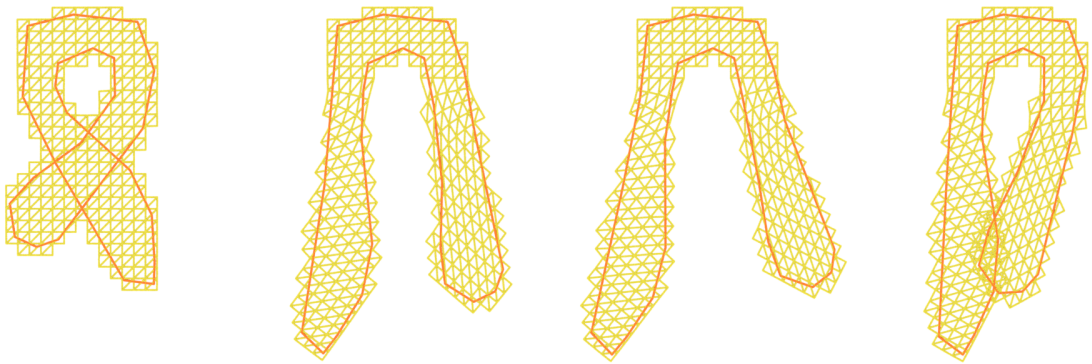


Figure 3.22: **Hexahedra tetrahedralization**. *(Left)* a standard interior face in $\mathcal{V}$. The centers of the two incident hexahedra are combined with two face vertices to form the tetrahedra (red). *(Middle)* a standard boundary face uses a face center instead of the missing incident hexahedron 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.



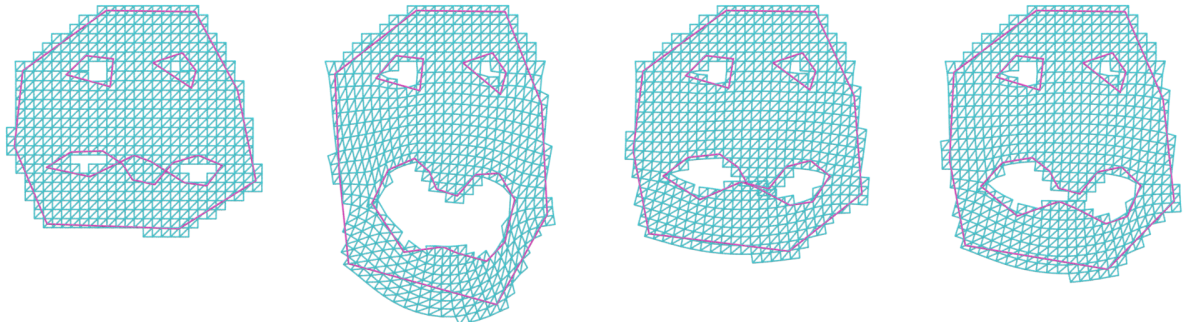
(a) Frame 0 (b) Frame 11 (c) Frame 27 (d) Frame 60

Figure 3.23: A self-intersecting shape is suspended from a ceiling. The geometry deforms under gravity, and both sides freely move regardless of the initial overlap.



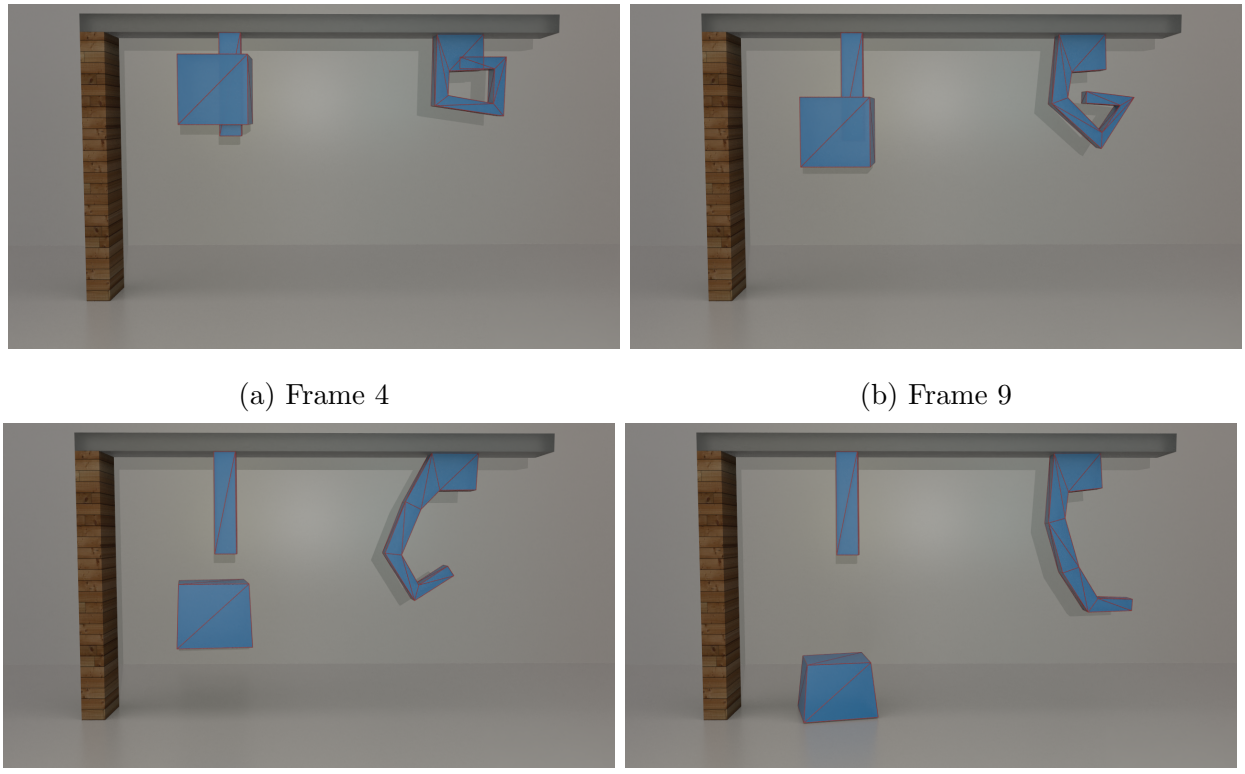
(a) Frame 0 (b) Frame 14 (c) Frame 59 (d) Frame 74

Figure 3.24: 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 3.25: A face with multiple boundary components and initially self-intersecting lips is successfully animated.



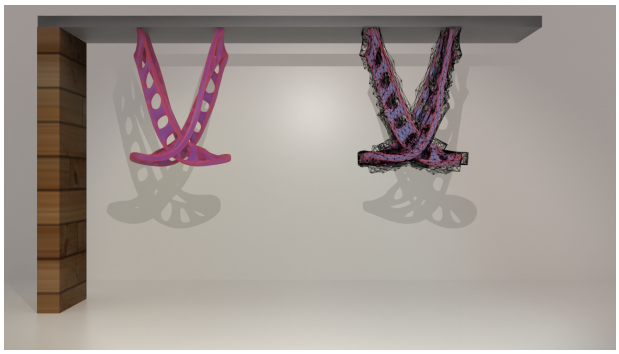
(a) Frame 4 (b) Frame 9

(c) Frame 33 (d) Frame 48

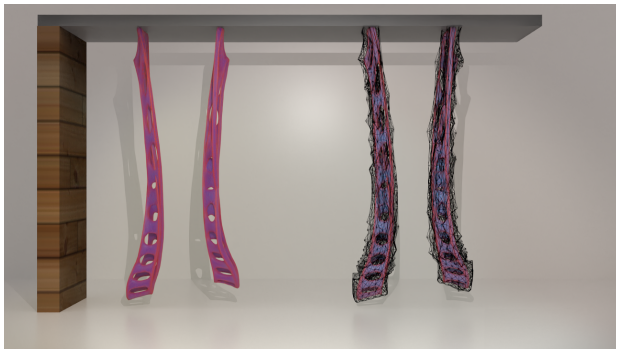
Figure 3.26: Simple self-intersecting 3D geometries are able to separate and unfurl with our algorithm.

(a) Frame 0

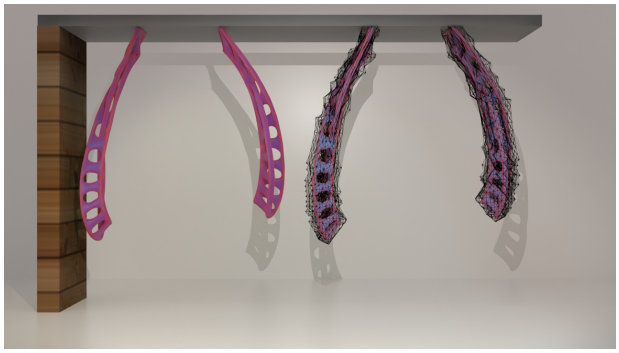
(b) Frame 20



(c) Frame 44

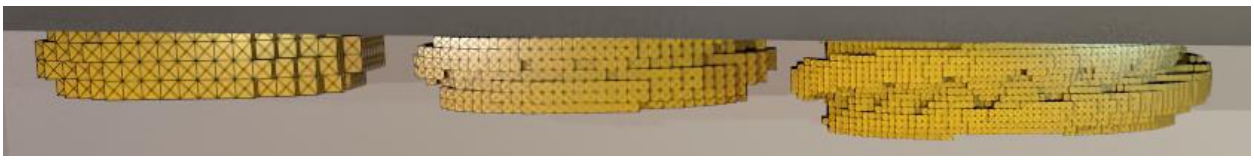


(d) Frame 78

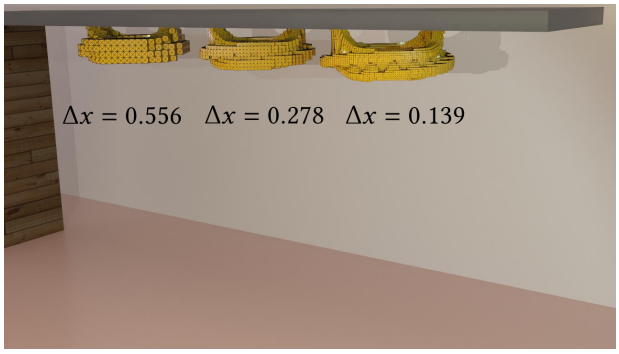


(e) Frame 110

Figure 3.27: 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.

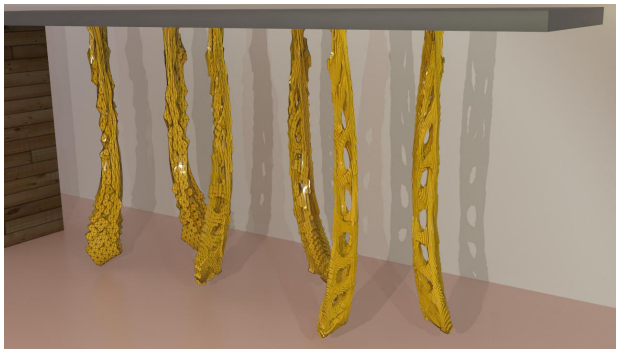


(a) Frame 0



(b) Frame 16

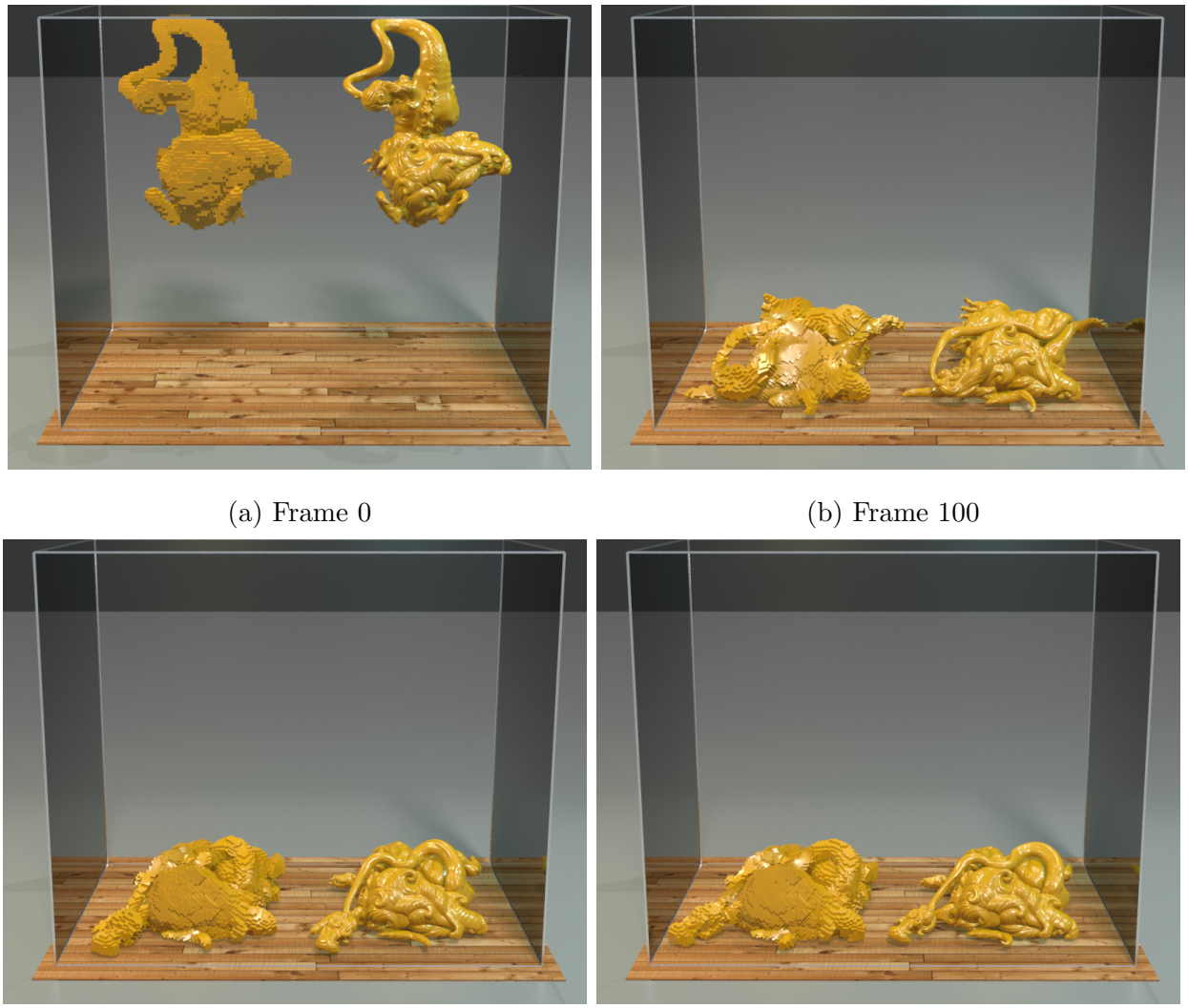
(c) Frame 33



(d) Frame 84

(e) Frame 115

Figure 3.28: Running the example shown in Figure 3.27 at different spatial resolutions. In each frame, from left to right, the background grids have $\Delta x = 0.556$, 0.278, and 0.139.



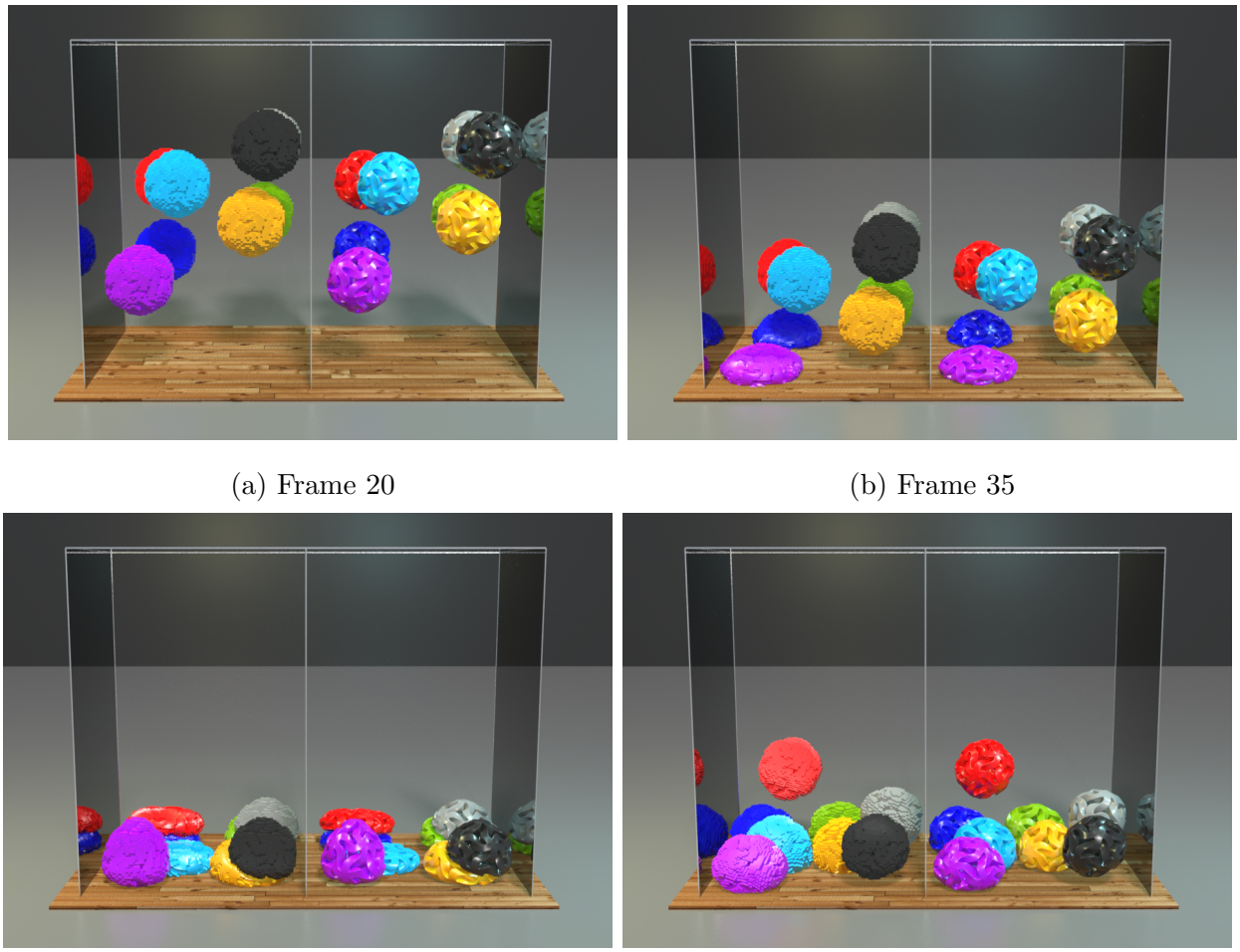
(a) Frame 0

(b) Frame 100

(c) Frame 200

(d) Frame 300

Figure 3.29: 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.



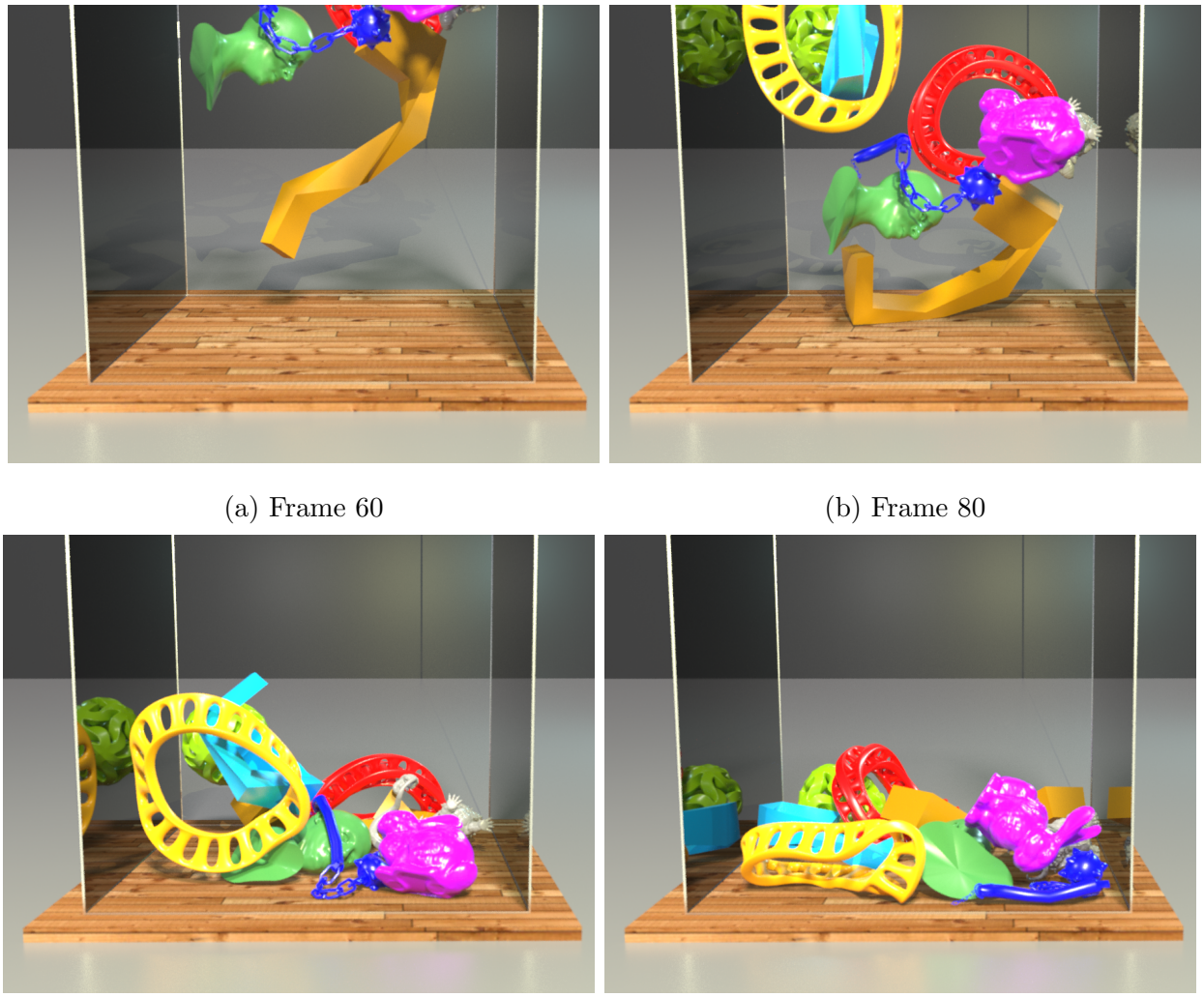
(a) Frame 20

(b) Frame 35

(c) Frame 45

(d) Frame 80

Figure 3.30: Several ball-like geometries with intricate slices and holes are successfully meshed with our algorithm and then deform and collide under an FEM simulation.



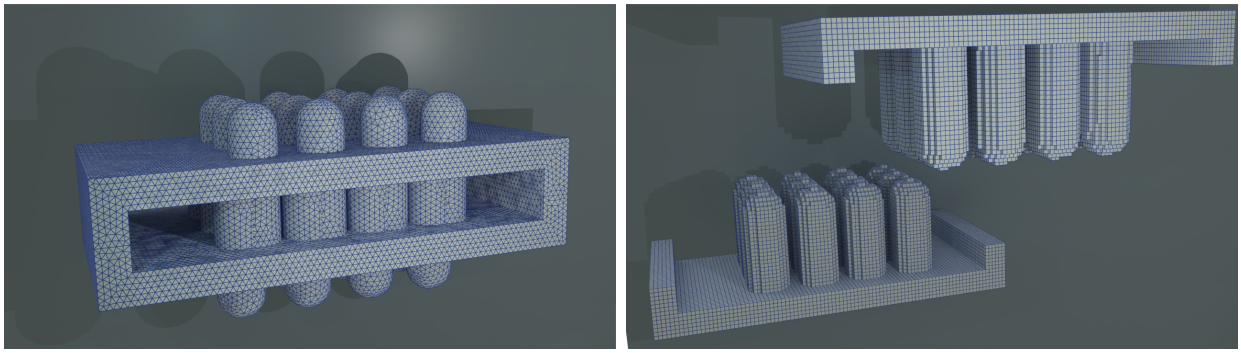
(a) Frame 60

(b) Frame 80

(c) Frame 100

(d) Frame 200

Figure 3.31: We simulated dropping our 3D examples into a box with a FEM sim.



(a) Initial State (b) Separation

Figure 3.32: Our method can successfully separate the torus and bristle geometry proposed in [SJP13].

CHAPTER 4

Primal Extended Position Based Dynamics for Hyperelasticity

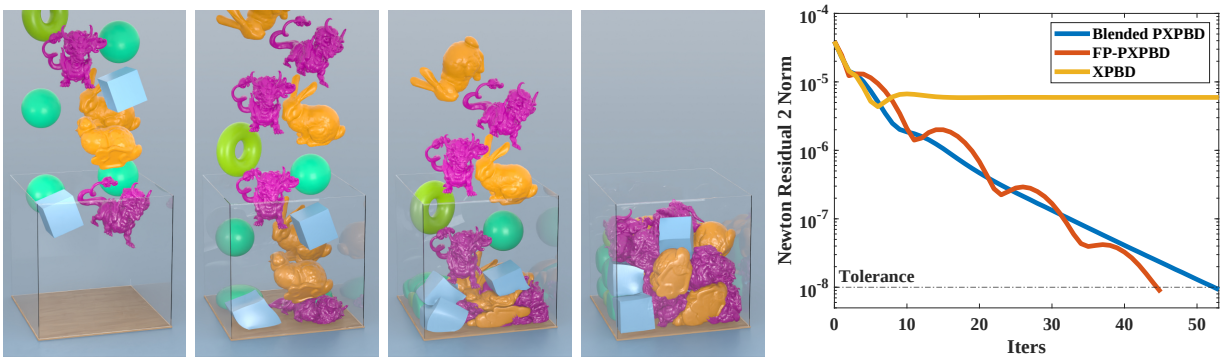


Figure 4.1: **30 Objects Dropping (left)**. Our Blended PXPBD (B-PXPBD) approach robustly handles large elastic deformations. **FEM Residual Comparison (right)**. B-PXPBD and FP-PXPBD reduce the backward Euler residual while XPBD stagnate in a representative step of a hyperelasticity simulation.

4.1 Methods

4.1.1 Equations

We consider implicit time stepping methods for integrating the FEM-discretized partial differential equations (PDEs) describing momentum balance with hyperelastic materials

$$\mathbf{M}\frac{\partial^2 \mathbf{x}}{\partial t^2} = -\frac{\partial PE}{\partial \mathbf{x}} + \mathbf{f}^{ext},\ PE(\mathbf{x}) = \sum_e \Psi(\mathbf{F}^e(\mathbf{x}))V^e. \tag{4.1}$$

Here $\mathbf{M} \in \mathbb{R}^{3N_p \times 3N_p}$ is a lumped (diagonal) mass matrix, $\mathbf{x} \in \mathbb{R}^{3N_p}$ are the deformed positions of the FEM mesh and the potential energy PE in the system is related to the hyperelastic potential energy density as Ψ. We use linear interpolation over tetrahedron (3D) or triangle (2D) meshes in our FEM formulation. V^e is the volume (3D) or area (2D) of the undeformed e^{th} element arising from the piecewise constant terms in an integrands associated with linear interpolation. $\mathbf{f}^{ext}$ are external forces (gravity etc.). The hyperelastic potential Ψ is a function of the deformation gradient in the e^{th} element ($\mathbf{F}^e$) which is related to deformed positions as

$$\mathbf{F}^e_{\alpha\beta}(\mathbf{x}) = \sum_i x_{i\alpha}\frac{\partial N_i}{\partial \mathbf{X}_\beta}(\mathbf{X}^e) \tag{4.2}$$

where N_i are the piecewise linear interpolation functions in the FEM formulation and $\mathbf{X}^e$ is the centroid of the undeformed element. We refer the reader to the Bonet and Wood [BW08] and Barbič and Sifakis [SB12] for more details.

4.1.1.1 Hyperelastic Energy Density

The hyperelastic potential defines the constitutive response of the material. We demonstrate our method with the fixed corotated potential from Stomakhin et al. [SHS12]

$$\Psi^{cor}(\mathbf{F}) = \mu|\mathbf{F} - \mathbf{R}(\mathbf{F})|_F^2 + \frac{\lambda}{2}(\det(\mathbf{F}) - 1)^2. \tag{4.3}$$

Here $\mathbf{R}(\mathbf{F})$ is the closest rotation to $\mathbf{F}$ which we compute from the polar singular value decomposition [GFJ16] and μ and λ are the Lamé coefficients. XPBD assumes that the

potential is of the form

$$PE(\mathbf{x}) = \sum_c \frac{1}{2} C_c(\mathbf{x}) \frac{1}{a_c} C_c(\mathbf{x}) \tag{4.4}$$

The corotated potential Ψ^{cor} can be adapted to this from in terms of the following constraints (in element) on the deformation gradient

$$\hat{C}_1(\mathbf{F}) = |\mathbf{F} - \mathbf{R}(\mathbf{F})|_F, \hat{C}_2(\mathbf{F}) = \det(\mathbf{F}) - 1. \tag{4.5}$$

The gradient of $\hat{C}_1$ is not defined when $\mathbf{F} = \mathbf{R}(\mathbf{F})$ (a common occurrence) and we use the modification $\tilde{C}_1 = \sqrt{\hat{C}_1^2 + \epsilon}$, where ϵ is an arbitrary positive constant to ensure that the gradient is always defined. We use constraints $C_1^e(\mathbf{x}) = \tilde{C}_1(\mathbf{F}^e(\mathbf{x}))$ and $C_2^e(\mathbf{x}) = \hat{C}_2(\mathbf{F}^e(\mathbf{x}))$ with weighting μ and λ respectively (in element e). This is equivalent to using the hyperelastic potential $\Psi^{cor} + \epsilon$ so it produces the same behavior as the corotated model.

We also demonstrate our method with an anisotropic model for muscle contraction (see Figure 4.4). Here the potential is

$$\Psi^{aniso}(\mathbf{F}) = \Psi^{cor} + \frac{\sigma_{max}}{\lambda_{ofl}}(f_a + \alpha_{act} f_p) \tag{4.6}$$

where the parameter $\alpha_{act} \in [0, 1]$ controls the degree of active contractile tension and f_a and f_p are based on the anisotropic fiber terms in Blemker et al. [BPD05]. More specifically, f_a and f_p are computed as the following:

$$f_p = \begin{cases} 0.0076\lambda_{ofl} e^{6.6(\frac{l^e}{\lambda_{ofl}} - 1)} - 0.05(l^e - \lambda_{ofl}) & l^e > \lambda_{ofl} \\ 0 & \text{otherwise} \end{cases} \tag{4.7}$$

$$f_a = \begin{cases} 0 & l^e \leq 0.4\lambda_{ofl} \\ 3\lambda_{ofl}(\frac{l^e}{\lambda_{ofl}} - 0.4)^3 & 0.6\lambda_{ofl} > l^e > 0.4\lambda_{ofl} \\ -0.6612\lambda_{ofl} + l^e - 1.33\lambda_{ofl}(\frac{l^e}{\lambda_{ofl}} - 1)^3 & 1.4\lambda_{ofl} \geq l^e \geq 0.6\lambda_{ofl} \\ 0.6774\lambda_{ofl} + 3\lambda_{ofl}(\frac{l^e}{\lambda_{ofl}} - 1.6)^3 & 1.6\lambda_{ofl} \geq l^e \geq 1.4\lambda_{ofl} \\ 0.6774\lambda_{ofl} & l^e > 1.6\lambda_{ofl} \end{cases} \tag{4.8}$$

where $l^e = \mathbf{F}^e \mathbf{v}^e$ and $\mathbf{v}^e$ is the fiber direction of the element e.

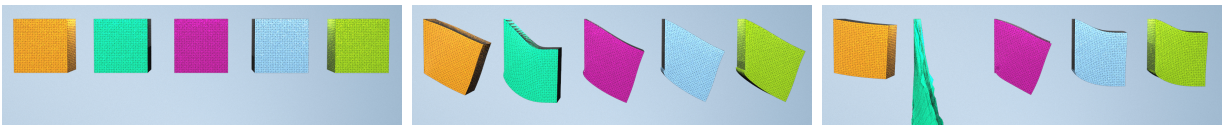


Figure 4.2: **Equal Budget Comparison**. From left to right: Newton (converged), Newton, FP-PXPBD, B-PXPBD, XPBD. With a limited budget, XPBD-style methods are stable, whereas the Newton's method suffers from instability. Frame 0, 10, 60 are shown in the figure.

4.1.1.2 Implicit Time Stepping

We consider both backward Euler and quasistatic time stepping schemes

$$\mathbf{M}\left(\frac{\frac{\mathbf{x}^{n+1}-\mathbf{x}^n}{\Delta t}-\mathbf{v}^n}{\Delta t}\right)=-\frac{\partial PE}{\partial \mathbf{x}}(\mathbf{x}^{n+1})+\mathbf{f}^{ext}. \tag{4.9}$$

Here $\mathbf{x}^n, \mathbf{v}^n$ represent the time $t^n = n\Delta t$ position and velocities. Quasistatic time stepping is the same but with the left hand side of Equation (4.9) replaced with $\mathbf{0}$. Note that we also may constrain some vertices $\mathbf{x}_i^n, 0 \leq i < N_p$ in practice to enforce boundary conditions and these equations are removed from Equation (4.9), however we omit the explicit representation of this for concise exposition.

4.1.2 XPBD

Macklin et al. [MMC16] solve Equation (4.9) with the introduction of a Lagrange multiplier λ_c associated with each constraint C_c. They assume the potential energy gradient is of the form

$$\Delta t^2 \frac{\partial PE}{\partial x_{i\alpha}} = -\sum_c \frac{\partial C_c}{\partial x_{i\alpha}}(\mathbf{x})\lambda_c \tag{4.10}$$

where they introduce $\lambda_c = -\frac{\Delta t^2}{a_c} C_c$ as an additional unknown which converts Equation (4.9) into the system

$$\mathbf{g}(\mathbf{x}^{n+1}, \boldsymbol{\lambda}) = \mathbf{M}\left(\mathbf{x}^{n+1} - \tilde{\mathbf{x}}\right) - \sum_c \lambda_c^T \nabla \mathbf{C}_c(\mathbf{x}^{n+1}) = \mathbf{0} \tag{4.11}$$

$$\mathbf{h}(\mathbf{x}^{n+1}, \boldsymbol{\lambda}) = \mathbf{C}(\mathbf{x}^{n+1}) + \frac{\mathbf{A}}{\Delta t^2}\boldsymbol{\lambda} = \mathbf{0}. \tag{4.12}$$

Here $\tilde{\mathbf{x}} = \mathbf{x}^n + \Delta t(\mathbf{v}^n + \mathbf{M}^{-1}\mathbf{f}^{ext})$ are the positions updated under the influence of inertia and external forces, $\boldsymbol{\lambda}$ is the vector of all Lagrange multipliers and $\mathbf{A}$ is a diagonal matrix with entries equal to a_c. The solution is approximated iteratively with $\mathbf{x}_k^{n+1}$ and $\boldsymbol{\lambda}_k$ denoting the k^{th} iterates. $\mathbf{g}(\mathbf{x}_k^{n+1}, \boldsymbol{\lambda}_k)$ is used to denote the residual of the position (primary) unknowns and $\mathbf{h}(\mathbf{x}_k^{n+1}, \boldsymbol{\lambda}_k)$ to denote the residual of the Lagrange multiplier (secondary) unknowns.

XPBD uses a nonlinear Gauss-Seidel procedure based on the linearization

$$\begin{pmatrix} \mathbf{M} + \sum_c \lambda_{ck} \frac{\partial^2 C_c}{\partial \mathbf{x}^2}(\mathbf{x}_k^{n+1}) & -\nabla C_c^T(\mathbf{x}_k^{n+1}) \\ \nabla \mathbf{C}(\mathbf{x}_k^{n+1}) & \frac{\mathbf{A}}{\Delta t^2} \end{pmatrix} \begin{pmatrix} \Delta \mathbf{x}_{k+1} \\ \Delta \boldsymbol{\lambda}_{k+1} \end{pmatrix} = - \begin{pmatrix} \mathbf{g}(\mathbf{x}_k^{n+1}, \boldsymbol{\lambda}_k) \\ \mathbf{h}(\mathbf{x}_k^{n+1}, \boldsymbol{\lambda}_k) \end{pmatrix}. \tag{4.13}$$

In XPBD, the red terms are omitted to enable the update

$$\left(\mathbf{C}^T(\mathbf{x}_k^{n+1})\mathbf{M}^{-1}\mathbf{C}(\mathbf{x}_k^{n+1}) + \frac{\mathbf{A}}{\Delta t^2}\right)\Delta \boldsymbol{\lambda}_{k+1} = -\mathbf{h}(\mathbf{x}_k^{n+1}, \boldsymbol{\lambda}_k) \tag{4.14}$$

$$\Delta \mathbf{x}_{k+1} = \mathbf{M}^{-1}\nabla \mathbf{C}(\mathbf{x}_k^{n+1})\Delta \boldsymbol{\lambda}_{k+1}. \tag{4.15}$$

Furthermore, Equation (4.14) is updated in a Gauss-Seidel fashion where the d^{th} Lagrange multiplier is updated via

$$\Delta \tilde{\lambda}_{k+1d} = \frac{-h_d(\mathbf{x}_k^{n+1}, \lambda_{kd})}{\nabla C_d^T(\mathbf{x}_k^{n+1})\mathbf{M}^{-1}\nabla C_d(\mathbf{x}_k^{n+1}) + \frac{a_d}{\Delta t^2}\lambda_{kd}}, \;\; \lambda_{k+1d} = \lambda_{kd} + \Delta \tilde{\lambda}_{k+1d}. \tag{4.16}$$

Note that we distinguish $\Delta \tilde{\lambda}_{k+1d}$ in Equation (4.16) from from $\Delta \lambda_{k+1d}$ in Equation (4.14) since only one one step of Gauss-Seidel iteration is performed on the linear system. Then the positions associated with the constraint are updated via Equation (4.15) to create

$$\mathbf{x}_{k+1}^{n+1} = \mathbf{x}_k^{n+1} + \mathbf{M}^{-1}\nabla \mathbf{C}_d(\mathbf{x}_k^{n+1})\Delta \tilde{\lambda}_{k+1d}. \tag{4.17}$$

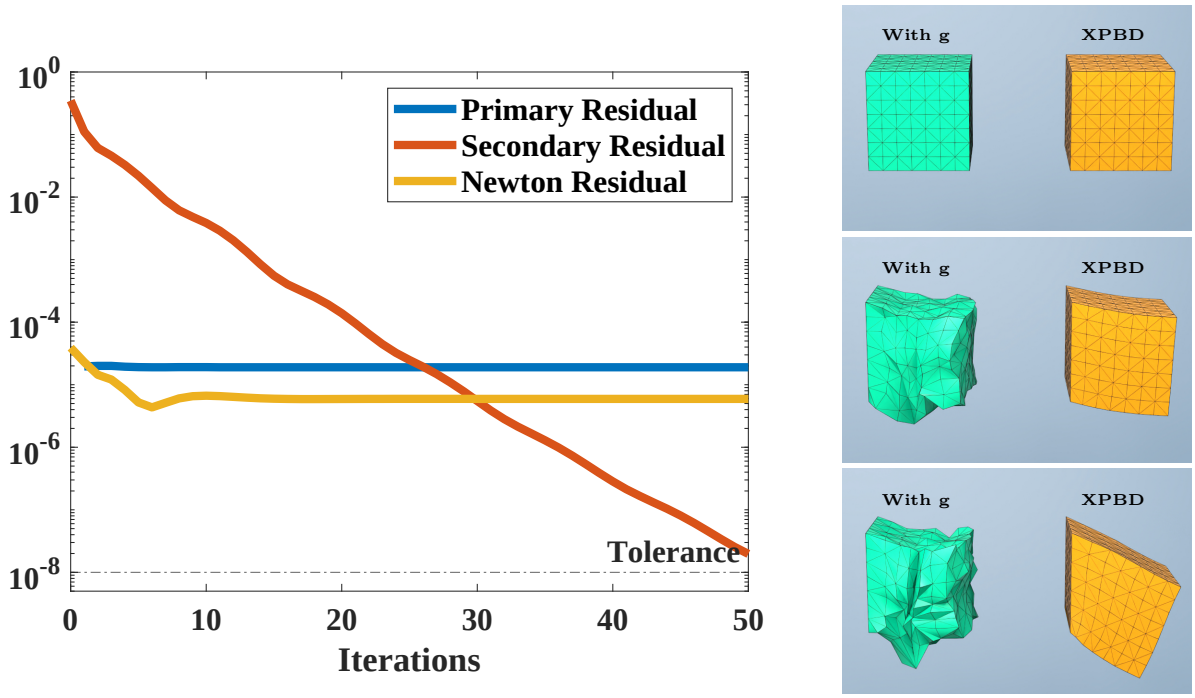
The system (Equations (4.11)-(4.12)) is then re-linearized (Equation (4.13)) and the process (Equations (4.16)-(4.17)) is repeated iteratively.

4.1.3 Primary residual XPBD (PXPBD)

The motivation for the omission of the residual and constraint Hessian terms (red) in Equation (4.13) is natural. The constraint Hessian is non-diagonal and its retention would preclude the decoupling of primary variables from the Lagrange multipliers in Equation (4.14). Furthermore, the primary residual term $\mathbf{g}(\mathbf{x}_k^{n+1}, \boldsymbol{\lambda}_k)$ requires more floating point operations and generally a gather operation for efficient parallel evaluation. As Macklin et al. [MMC16] point out, the initial guess of $\boldsymbol{\lambda}_0 = \mathbf{0}$ and $\mathbf{x}_0^{n+1} = \tilde{\mathbf{x}}$ means that $\mathbf{g}(\mathbf{x}_0^{n+1}, \boldsymbol{\lambda}_0) = \mathbf{0}$. However, its omission is harder to justify in latter iterates, though Macklin et al. [MMC16] argue that it is justified when the constraint gradients vary slowly and further that its omission makes the approach similar to that of Goldenthal et al. [GHF07]. While omission of secondary information is commonly done in quasi-Newton approaches, we observe that omission of the primary residual terms can lead to stagnation in residual reduction (see Figure 4.3(a)). Unfortunately, we also notice that inclusion of this term can cause XPBD to lose its favorable stability properties (see Figure 4.3(b)). We note though that if the global system in Equation (4.13) is solved with sufficient accuracy (e.g. with a Krylov method and without omission of the red terms), then stability and residual reduction can be achieved, however this is more costly than Newton's method for Equation (4.9) since the system size is larger with the inclusion of the $\boldsymbol{\lambda}$ unknowns.

4.1.3.1 Blended Primal XPBD (B-PXPBD)

We believe that the stability of XPBD is due to the omission of this primary residual term $\mathbf{g}(\mathbf{x}_k^{n+1}, \boldsymbol{\lambda}_k)$. We observe that this omission can be done without any error if the position update is chosen to guarantee that the primary residual is zero. This can be done by solving Equation (4.11) for $\mathbf{x}_{k+1}^{n+1}$ with $\boldsymbol{\lambda}_{k+1}$ fixed after the update (of a single Lagrange multiplier λ_{k+1d}) in Equation (4.16). We again note that in this context, the Lagrange multipliers $\boldsymbol{\lambda}_{k+1}$ are similar to stresses. Indeed as the a_c are taken to infinity we can see similarities between



(a) (b)

Figure 4.3: **(a) Primal Residual Comparison: Stagnation**. While XPBD reliably reduces the secondary residual, its omission of the primary residual in the linearization causes its primary residual to stagnate, making its true (Newton) residual stagnate as well. **(b) Primal Residual Inclusion: Instability**. XPBD is unstable when the primal residual term is not omitted.

Equations (4.11)-(4.12) and the discretized equations for incompressible fluids and for finite values of a_c the formulation is similar to the compressible formulations in Stomakhin et al. [SSJ14] and Kwatra et al. [KGF10]. Therefore, the process of solving Equation (4.11) for $\mathbf{x}_{k+1}^{n+1}$ with $\boldsymbol{\lambda}_{k+1}$ fixed is akin to solving for the change in positions given a fixed stress state (that does not depend on positions).

Unfortunately, solving Equation (4.11) for $\mathbf{x}_{k+1}^{n+1}$ is complicated by the dependence of the constraint gradient $\nabla\mathbf{C}(\mathbf{x}_{k+1}^{n+1})$ on positions and solving it accurately would be nearly as difficult as solving the original system in Equation (4.9). Furthermore, this dependence of the constraint gradient on positions means changing the stress in one constraint propagates

to changes in positions in adjacent constraints and therefore throughout the mesh. For example if fixed point iteration were used to solve for $\mathbf{x}_{k+1}^{n+1}$ given $\boldsymbol{\lambda}_{k+1}$ where the only change to $\boldsymbol{\lambda}_k$ was in a single constraint d (as in Equation (4.16)), then first only the positions of the vertices in the constraint would be changed, but then in the second iteration, any other constraint gradients with dependence on these positions would change, and all positions associated with those constraints would change, and so on. This would quickly become computationally inefficient, however performing one iteration results in an update that only changes the positions involved in the constraint associated with the Lagrange multiplier update in Equation (4.16)

$$\mathbf{x}_{k+1}^{n+1} = \tilde{\mathbf{x}} + \sum_c \lambda_{kc}\mathbf{M}^{-1}\nabla\mathbf{C}_c(\mathbf{x}_k^{n+1}) + \mathbf{M}^{-1}\nabla\mathbf{C}_d(\mathbf{x}_k^{n+1})\Delta\tilde{\lambda}_{k+1d}. \tag{4.18}$$

Note that when the residual $\mathbf{g}(\mathbf{x}_k^{n+1}, \boldsymbol{\lambda}_k) = \mathbf{0}$ is zero this update coincides with that of Equation (4.15). We found that even using this first fixed point iterate was enough to improve residual reduction, however we also found that it reduced the stability compared to Equation (4.15). We remedy this by taking a linear combination of the updates in Equations (4.15) and (4.18)

$$\mathbf{x}_{k+1}^{n+1} = \zeta\left(\mathbf{M}^{-1}\nabla C_d(\mathbf{x}_k^{n+1})\Delta\tilde{\lambda}_{k+1d}\right) + (1-\zeta)\Delta\mathbf{x}_{k+1}^{fp} + \mathbf{x}_k^{n+1} \tag{4.19}$$

$$\Delta\mathbf{x}_{k+1}^{fp} = \tilde{\mathbf{x}} + \sum_c \lambda_{kc}\mathbf{M}^{-1}\nabla C_c(\mathbf{x}_k^{n+1}) + \Delta\tilde{\lambda}_{k+1d}\mathbf{M}^{-1}\nabla C_d(\mathbf{x}_k^{n+1}) - \mathbf{x}_k^{n+1}. \tag{4.20}$$

The parameter ζ can usually chosen to be 0.5. We increase it if we observe instability and raise it if we see residual stagnation.

4.1.3.2 Implementation

Blended P-XPBD can be implemented as a small modification to XPBD. At the m^{th} iteration, we store $\Delta\mathbf{x}_d^{total} = \sum_{k<m}\Delta\mathbf{x}_{kd}$ at the d^{th} constraint where

$$\Delta\mathbf{x}_{kd} = \zeta\left(\mathbf{M}^{-1}\nabla C_d(\mathbf{x}_k^{n+1})\Delta\tilde{\lambda}_{kd}\right) + (1-\zeta)\Delta\mathbf{x}_k^{fp}. \tag{4.21}$$

Then we can obtain $\Delta \mathbf{x}_k^{fp}$ easily by

$$\Delta \mathbf{x}_k^{fp} = \lambda_{kd} \mathbf{M}^{-1} \nabla C_d(\mathbf{x}_k^{n+1}) - \Delta \mathbf{x}_d^{total}. \tag{4.22}$$

Pseudo-code similar to that of XPBD in Macklin et al. [MMC16] is provided in Algorithm 1.

ALGORITHM 1: B-PXPBD Simulation Loop

Initialize $\Delta \mathbf{x}_d^{total} = \mathbf{0}$.

while *not reached maximal iterations* **do**

  for *constraint d* **do**

    1. Compute $\Delta \tilde{\lambda}_{kd}$ as in Equation 4.16. ;
    2. Compute $\Delta \mathbf{x}_k^{fp}$ using Equation 4.22. ;
    3. Update $\mathbf{x}^{n+1}$ using Equation 4.20. ;
    4. Update λ_{k+1d} using Equation 4.16. ;
    5. Update $\Delta \mathbf{x}_d^{total} \leftarrow \Delta \mathbf{x}_d^{total} + \Delta \mathbf{x}_{kd}$;

  end

end

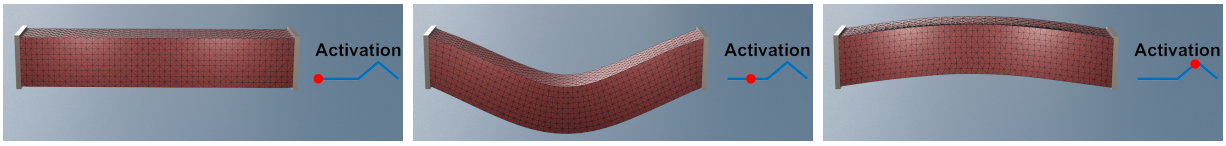


Figure 4.4: **Muscle Box Activation**. A rectangular bar with both ends clamped falls under gravity. Two seconds later, the muscle box is activated and contracts along the horizontal direction. The level of activation is shown on the right side of the images. $t = 0.0333, 1.2, 2.9$ seconds are shown in the footage.

ALGORITHM 2: FP-PXPBD Simulation Loop

while *not reached maximal iterations* **do**
  for *element e* **do**
    while *not converged or reached maximal iterations* **do**
      begin Solve Newton system
        1. Compute Newton residual via Equation (4.29);
        2. Compute $\mathbf{b}^e_{k+1l}$ via Equation 4.33;
        3. Compute $\delta\mathbf{F}^e_{k+1l}$ via Equation 4.34 with the approximation in Equation 4.48;
        4. Compute $\delta\mathbf{x}^e_{k+1l}$ as in Equation 4.32 ;
        5. Update the nodes on the element with $\mathbf{x}^{n+1}_{i^e k+1l+1} = \mathbf{x}^{n+1}_{i^e k+1l} + \delta\mathbf{x}^e_{i^e k+1l}$;
        6. Update $\mathbf{P}^e_{k+1l+1} = \frac{\partial\Psi}{\partial\mathbf{F}}(\mathbf{F}^e(\mathbf{x}^{n+1}_{i^e k+1l+1}))$;
      end
    end
  end
end

4.1.3.3 First Piola-Kirchhoff Primal XPBD (FP-PXPBD)

Noting that the auxiliary Lagrange multiplier variables are similar to stresses, we observe some convenient properties that arise from choosing an alternative stress measure in an analogous primary/secondary formulation of Equation 4.9. In a general FEM-discretized hyperelastic formulation (see Barbič and Sifakis [SB12]), the potential energy gradient has the expression

$$\frac{\partial PE}{\partial x_{i\alpha}}(\mathbf{x}) = \sum_{e,\beta,\gamma} P_{\beta\gamma}(\mathbf{F}^e(\mathbf{x}))\delta_{\alpha\beta}\frac{\partial N_i}{\partial X_\gamma}(\mathbf{X}^e)V^e \tag{4.23}$$

where $\delta_{\alpha\beta}$ is the Kronecker delta tensor and $\mathbf{P} = \frac{\partial\Psi}{\partial\mathbf{F}}$ is the gradient of the hyperelastic potential energy density with respect to the deformation gradient. This is the first Piola-

Kirchhoff stress [BW08]. If we introduce it as an unknown (analogous to λ_c), then tensor $B^e_{i\alpha\beta\gamma} = \delta_{\alpha\beta}\frac{\partial N_i}{\partial X_\gamma}(\mathbf{X}^e)V^e$ is analogous to the ∇C_c terms in XPBD since they convert the auxiliary (stress) terms to force in the expression in Equation (4.10). With this formulation, an analogous method consists of

$$\mathbf{g}(\mathbf{x}^{n+1}, \mathbf{P}^{n+1}) = \mathbf{M}\left(\mathbf{x}^{n+1} - \tilde{\mathbf{x}}\right) + \Delta t^2 \mathbf{B}\mathbf{P} = \mathbf{0} \tag{4.24}$$

$$\mathbf{h}^e(\mathbf{x}^{n+1}, \mathbf{P}^{n+1}) = \frac{\partial \Psi}{\partial \mathbf{F}}(\mathbf{F}^e(\mathbf{x}^{n+1})) - \mathbf{P}^e = \mathbf{0}. \tag{4.25}$$

Note that with this expression, the tensor $\mathbf{B}$ does not depend on the positional unknowns $\mathbf{x}^{n+1}$. In contrast, the analogous expression $\nabla\mathbf{C}(\mathbf{x}^{n+1})$ in Equations (4.11) does have this dependence, and it is precisely this issue that leads to the red terms in the linearization in Equation (4.13). Therefore, a formulation based on Equations (4.24) and (4.25) rather than Equations (4.11) and (4.12) will automatically satisfy the constraint that $\mathbf{g} = \mathbf{0}$ at each Gauss-Newton iteration and will not require the omission of the constraint Hessian since it is exactly zero. We adopt this strategy and iteratively solve Equations (4.24) and (4.25) for primary position unknowns $\mathbf{x}^{n+1}_k$ and secondary element stresses $\mathbf{P}^e_k$ in a Gauss-Seiedel manner analogous to that of the original XPBD. We observe that this retains the favorable stability properties of XPBD, while allowing for accurate residual reduction and application to arbitrary hyperelastic constitutive models.

This approach shifts the difficulty from the primary Equation (4.24) to the secondary Equation (4.25). It is trivial to maintain a zero primary residual, which simply requires plugging the current guess for the element stresses $\mathbf{P}_k$ into Equation (4.24) to define the current guess for $\mathbf{x}^{n+1}_k$. We update this guess iteratively by solving for the positions $\mathbf{x}^{n+1,e}_{k+1}$ in element e that satisfy Equation (4.25). This is equivalent to solving the nonlinear system equations for one element with the stresses in all adjacent elements held fixed, with their dependence on the element positions ignored. We use Ω^e to denote set of the mesh vertices

i^e in element e and solve

$$\mathbf{M}^e(\mathbf{x}_{k+1}^{n+1,e} - \tilde{\mathbf{x}}^e) + \Delta t^2 \mathbf{B}^e \mathbf{P}_{k+1}^e = \mathbf{f}^e \tag{4.26}$$

$$\frac{\partial \Psi}{\partial \mathbf{F}}(\mathbf{F}^e(\mathbf{x}_{k+1}^{n+1})) - \mathbf{P}_{k+1}^e = \mathbf{0} \tag{4.27}$$

where $f_{i^e\alpha k}^e = \Delta t^2 (f_{i^e\alpha}^{ext} - \sum_{\tilde{e}\neq e,\gamma,\delta} B_{i^e\alpha\gamma\delta}^{\tilde{e}} P_{k\gamma\delta}^{\tilde{e}}), \mathbf{f}^{ext}$ is the external force. In index notations Equation 4.26 can be written as:

$$\sum_{j^e,\beta} m_{i^e j^e} \delta_{\alpha\beta} \left(x_{k+1 j^e \beta}^{n+1} - \tilde{x}_{j^e\beta} \right) + \Delta t^2 \sum_{\gamma,\delta} B_{i^e\alpha\gamma\delta}^e P_{k+1\gamma\delta}^e = f_{i^e\alpha k}^e \tag{4.28}$$

Here Equation (4.27) can be satisfied trivially by setting $\mathbf{P}_{k+1}^e = \frac{\partial \Psi}{\partial \mathbf{F}}(\mathbf{F}^e(\mathbf{x}_{k+1}^{n+1}))$. With this simplification, Equations (4.27)-(4.28) can be rewritten as

$$\mathbf{M}^e(\mathbf{x}_{k+1}^{n+1,e} - \tilde{\mathbf{x}}^e) + \Delta t^2 \mathbf{B}^e \frac{\partial \Psi}{\partial \mathbf{F}}(\mathbf{F}^e(\mathbf{x}_{k+1}^{n+1,e})) - \mathbf{f}^e = \mathbf{0} \tag{4.29}$$

where $M_{i^e\alpha j^e\beta}^e = m_{i^e}\delta_{i^e j^e}\delta_{\alpha\beta}$ is portion of the mass matrix (where m_{i^e} is the mass of node i^e) composed of entries only in the element and $\mathbf{x}_k^{n+1,e}$ and $\tilde{\mathbf{x}}^e$ are extractions of element-wise positions from $\mathbf{x}_k^{n+1}$ and $\tilde{\mathbf{x}}$ respectively. Note that $\frac{\partial \Psi}{\partial \mathbf{F}}(\mathbf{F}^e(\mathbf{x}_{k+1}^{n+1})) = \frac{\partial \Psi}{\partial \mathbf{F}}(\mathbf{F}^e(\mathbf{x}_{k+1}^{n+1,e}))$ since the element deformation gradient only depends on the nodes of the element. Lastly, $\mathbf{B}^e$ has entries $B_{i^e\alpha\gamma\delta}^e = \delta_{\alpha\gamma}\frac{\partial N_{i^e}}{\partial X_\delta}(\mathbf{X}^e)V^e$ from Equation (4.23).

We use Netwon's method to solve Equation (4.29). $\mathbf{x}_{i^e k+1 l}^{n+1,e}$ denotes the l^{th} iteration of the local Newton procedure for computing the $k+1^{\text{th}}$ global iteration, which modifies the nodes i^e of element e. These nodes are updated in Newton's method as $\mathbf{x}_{i^e k+1 l+1}^{n+1,e} = \mathbf{x}_{i^e k+1 l}^{n+1,e} + \delta\mathbf{x}_{i^e k+1 l}^e$. To solve for $\delta\mathbf{x}_{i^e k+1 l}^e$, we make the following approximation:

$$\frac{\partial \Psi}{\partial \mathbf{F}}(\mathbf{F}^e(\mathbf{x}_{k+1 l+1}^{n+1,e})) \approx \frac{\partial^2 \Psi}{\partial \mathbf{F}^2}(\mathbf{F}^e(\mathbf{x}_{k+1 l}^{n+1,e})) : \delta\mathbf{F}_{k+1 l}^e + \mathbf{P}_{k+1 l}^e \tag{4.30}$$

and solve the system

$$\mathbf{M}^e \delta\mathbf{x}_{k+1 l}^e + \Delta t^2 \mathbf{B}^e \frac{\partial^2 \Psi}{\partial \mathbf{F}^2}(\mathbf{F}^e(\mathbf{x}_{k+1}^{n+1,e}))\delta\mathbf{F}_{k+1 l}^e = -\mathbf{g}_{k+1 l} \tag{4.31}$$

where $\mathbf{g}^e_{k+1l} = \mathbf{M}^e(\mathbf{x}^{n+1,e}_{k+1l} - \tilde{\mathbf{x}}^e) + \Delta t^2 \mathbf{B}^e \frac{\partial \Psi}{\partial \mathbf{F}}(\mathbf{F}^e(\mathbf{x}^{n+1,e}_{k+1l})) - \mathbf{f}^e$. This is a linear system of size 12×12 (6×6 in 2D). To reduce the size of the system, we use an affine basis for the change in positions determined by a Newton step:

$$\delta \mathbf{x}^e_{i^e k+1l} = \delta \mathbf{F}^e_{k+1l}(\mathbf{X}^e_{i^e} - \mathbf{X}^e_{\text{com}}) + \mathbf{b}^e_{k+1l} \tag{4.32}$$

where $\delta \mathbf{F}^e_{k+1l}$ are distortional degrees of freedom in the element, $\mathbf{b}^e_{k+1l}$ are translational degrees of freedom and $\mathbf{X}^e_{\text{com}}$ is the center of mass of the element in the undeformed configuration. Similarly, $\mathbf{X}^e_{i^e}$ refer to the undeformed positions of the vertices in the element. Distortional and translational degrees of freedom can be decoupled for more efficient solution. This can be seen by first summing over $i^e \in \Omega^e$ in Equation 4.31 and noting that $\sum_{i^e \in \Omega^e} B^e_{i^e \alpha\gamma\delta} = 0$ (when the interpolating functions N_{i^e} satisfy the partition of unity property)

$$\begin{aligned}
&\sum_{i^e, j^e, \beta} M_{i^e \alpha j^e \beta} \delta x^e_{j^e \beta k+1l} \\
&+ \sum_{i^e, \gamma, \delta, \sigma, \nu} \Delta t^2 B^e_{i^e \alpha\gamma\delta} \frac{\partial^2 \Psi}{\partial F_{\gamma\delta} \partial F_{\sigma\nu}}(\mathbf{F}^e(\mathbf{x}^{n+1,e}_{k+1l})) \delta F^e_{\sigma\nu k+1l} = \\
&\sum_{i^e} m_{i^e} \delta x^e_{i^e \alpha k+1l}.
\end{aligned}$$

Next, by defining tensor $D^e_{i^e \alpha\epsilon\tau} = \delta_{\alpha\epsilon}(X^e_{i^e \tau} - X^{e\tau}_{\text{com}})$ relating the original variables to the affine with $\delta x^e_{i^e \alpha k+1l} = \sum_{\epsilon,\tau} D^e_{i^e \alpha\epsilon\tau} \delta F^e_{\epsilon\tau k+1l} + b^e_{\alpha k+1l}$ we see that

$$\begin{aligned}
\sum_{i^e} m_{i^e} \delta x^e_{i^e \alpha k+1l} &= \sum_{i^e, \epsilon, \tau} m_{i^e} D^e_{i^e \alpha\epsilon\tau} \delta F^e_{\epsilon\tau k+1l} + \sum_{i^e} m_{i^e} b^e_{\alpha k+1l} \\
&= \sum_{\tau} (m_e X^e_{\tau \text{com}} - m_e X^e_{\tau \text{com}}) \delta F^e_{\alpha\tau k+1l} + m_e b^e_{\alpha k+1l} = m_e b^e_{\alpha k+1l}
\end{aligned}$$

where $m^e = \sum_{i^e \in \Omega^e} m_{i^e}$ is the element mass and the undeformed element center of mass is defined as $\mathbf{X}^e = \frac{1}{m^e} \sum_{i^e} m_{i^e} \mathbf{X}_{i^e}$. Therefore the translational degrees of freedom can easily be obtained from

$$\mathbf{b}^e_{k+1l} = \frac{-1}{m_e} \sum_{i^e \in \Omega^e} \mathbf{g}_{i^e k+1l}. \tag{4.33}$$

Once the translational component $\mathbf{b}^e_{k+1l}$ is determined from Equation (4.33), we multiply Equation 4.31 on the left side by $\mathbf{D}^{eT}$ to reveal a decoupled system of equations for the distortional degrees of freedom $\delta\mathbf{F}^e_{k+1l}$

$$\left(\tilde{\mathbf{M}}^e + \Delta t^2 V^e \frac{\partial^2 \Psi}{\partial \mathbf{F}^2}(\mathbf{F}_e(\mathbf{x}^{n+1}_{k+1l}))\right) \delta\mathbf{F}^e_{k+1l} = -\mathbf{G}^e_{k+1l} \tag{4.34}$$

where

$$G^e_{\eta\nu k+1l} = \sum_{i^e,\alpha} D^e_{i^e\alpha\eta\nu}\left(g_{i^e\alpha k+1l} + m_{i^e} b^e_{\alpha k+1l}\right) \tag{4.35}$$

$$\tilde{M}^e_{\eta\nu\epsilon\tau} = \sum_{i^e,j^e,\alpha} D^e_{i^e\alpha\eta\nu} M^e_{i^e\alpha j^e\beta} D^e_{j^e\alpha\epsilon\tau}. \tag{4.36}$$

For efficiency note that the matrix $\tilde{\mathbf{M}}^e$ is block diagonal with the structure $\tilde{M}^e_{\alpha\beta\gamma\delta} = \delta_{\alpha\gamma}\hat{M}^e_{\beta\delta}$, where interestingly $\hat{\mathbf{M}}^e = \sum_{i^e} m_{i^e}(\mathbf{X}_{i^e} - \mathbf{X}^e_{\text{com}})(\mathbf{X}_{i^e} - \mathbf{X}^e_{\text{com}})^T$ is the affine inertia tensor used in [JSS15]. This can be seen from

$$\begin{aligned}
\tilde{M}^e_{\eta\nu\epsilon\tau} &= \sum_{i^e,j^e,\alpha,\beta} D^e_{i^e\alpha\eta\nu} m_{i^e}\delta_{i^e j^e}\delta_{\alpha\beta} D^e_{j^e\beta\epsilon\tau} \\
&= \sum_{i^e,j^e,\alpha,\beta} \delta_{\alpha\eta}(X^e_{i^e\nu} - X^e_{\nu\text{com}}))m_{i^e}\delta_{i^e j^e}\delta_{\alpha\beta}\delta_{\alpha\epsilon}(X^e_{j^e\tau} - X^e_{\tau\text{com}})) \\
&= \delta_{\eta\epsilon}\sum_{i^e} m_{i^e}(X^e_{i^e\nu} - X^e_{\nu\text{com}})(X^e_{i^e\tau} - X^e_{\tau\text{com}}).
\end{aligned}$$

However note that unlike in [JSS15] , the matrix $\hat{\mathbf{M}}^e$ is not in general diagonal.

4.1.3.4 Partition of unity and reproduction of linear functions

We note that the expression in Equation 4.34 relies on the identity $\mathbf{D}^T\mathbf{B} = V^e\mathbf{I}$. This can be shown when the interpolating functions N_i form a partition of unity $\sum_i N_i = 1$ and

reproduce linear functions as $\mathbf{X} = \sum_i \mathbf{X}_i N_i(\mathbf{X})$. In particular, we show

$$
\begin{aligned}
(\mathbf{D}^T\mathbf{B})_{\epsilon\tau\beta\nu} &= D^e_{i^e\alpha\epsilon\tau} B^e_{i^e\alpha\beta\nu} && (4.37)\\
&= \sum_{i^e,\alpha} \delta_{\alpha\beta}\delta_{\alpha\epsilon}(X^e_{i^e\tau} - X^e_{\tau\mathrm{com}}) \frac{\partial N_{i^e}}{\partial X_\nu}(\mathbf{X}^e) V^e && (4.38)\\
&= \delta_{\epsilon\beta} V^e \sum_{i^e} (X^e_{i^e\tau} - X^e_{\tau\mathrm{com}}) \frac{\partial N_{i^e}}{\partial X_\nu}(\mathbf{X}^e) && (4.39)\\
&= \delta_{\epsilon\beta} V^e (\sum_{i^e} \frac{\partial N_{i^e}}{\partial X_\nu}(\mathbf{X}^e) X^e_{i^e\tau} - \sum_{i^e} \frac{\partial N_{i^e}}{\partial X_\nu}(\mathbf{X}^e) X^e_{\tau\mathrm{com}}) && (4.40)\\
&= \delta_{\epsilon\beta} V^e (\sum_{i^e} \frac{\partial N_{i^e}}{\partial X_\nu}(\mathbf{X}^e) X^e_{i^e\tau} - (\sum_{i^e} \frac{\partial N_{i^e}}{\partial X_\nu}(\mathbf{X}^e)) X^e_{\tau\mathrm{com}}) && (4.41)\\
&= \delta_{\epsilon\beta} V^e (\sum_{i^e} \frac{\partial N_{i^e}}{\partial X_\nu}(\mathbf{X}^e) X^e_{i^e\tau} - \frac{\partial (\sum_{i^e} N_{i^e})}{\partial X_\nu}(\mathbf{X}^e) X^e_{\tau\mathrm{com}}) && (4.42)\\
&= \delta_{\epsilon\beta} V^e (\sum_{i^e} \frac{\partial N_{i^e}}{\partial X_\nu}(\mathbf{X}^e) X^e_{i^e\tau} - \frac{\partial 1}{\partial X_\nu}(\mathbf{X}^e) X^e_{\tau\mathrm{com}}) && (4.43)\\
&= \delta_{\epsilon\beta} V^e \sum_{i^e} \frac{\partial N_{i^e}}{\partial X_\nu}(\mathbf{X}^e) \mathbf{X}^e_{i^e\tau} && (4.44)\\
&= V^e \delta_{\epsilon\beta}\delta_{\tau\nu}. && (4.45)
\end{aligned}
$$

4.1.3.5 Quasi-Newton

In general, solving Equation (4.34) for the distortional $\delta\mathbf{F}^e_l$ requires the solution of a 9×9 linear system (4×4 in 2D). However, we generally know (or can compute with minimal effort) the eigen decomposition of $\frac{\partial^2\Psi}{\partial\mathbf{F}^2}(\mathbf{F}_e(\mathbf{x}^{n+1,e}_{k+1l}))$ [TSI05, SGK19]. Since $\tilde{\mathbf{M}}^e$ is constant and block diagonal, its inverse can be precomputed with minimal storage and the inverse of $\tilde{\mathbf{M}}^e + \Delta t^2 \frac{\partial^2\Psi}{\partial\mathbf{F}^2}(\mathbf{F}_e(\mathbf{x}^{n+1}_{k+1l}))$ can be approximated using the Sherman-Morrison rank-1 update formula [Hag89]. However, if all eigen modes are used, this computation can be costly. We therefore use just a few modes in a quasi-Newton strategy, where the cost of the slow down in Newton convergence must be balanced against higher computational time per iteration, brought by using more modes in the Sherman-Morrison formula. In the case of the corotated

model, we can use

$$\frac{\partial^2 \Psi}{\partial \mathbf{F}^2}(\mathbf{F}^e) = 2\mu \mathbf{I} + 2\mu \frac{\partial \mathbf{R}(\mathbf{F})}{\partial \mathbf{F}} + \lambda \frac{\partial \det(\mathbf{F}^e)}{\partial \mathbf{F}^e} \otimes \frac{\partial \det(\mathbf{F}^e)}{\partial \mathbf{F}^e} \tag{4.46}$$

$$+ \lambda \det(\mathbf{F}^e) \frac{\partial^2 \det(\mathbf{F}^e)}{\partial (\mathbf{F}^e)^2} \tag{4.47}$$

$$\approx 2\mu \mathbf{I} + \lambda \frac{\partial \det(\mathbf{F}^e)}{\partial \mathbf{F}^e} \otimes \frac{\partial \det(\mathbf{F}^e)}{\partial \mathbf{F}^e} \tag{4.48}$$

where $\mathbf{I}$ is the 9×9 (4×4 in 2D) identity matrix. Note the term $2\mu \frac{\partial \mathbf{R}(\mathbf{F})}{\partial \mathbf{F}} + \lambda \det(\mathbf{F}^e) \frac{\partial^2 \det(\mathbf{F}^e)}{\partial (\mathbf{F}^e)^2}$ is omitted in the approximation. With this approximation, we can use the Sherman-Morrison formula for

$$\left(\tilde{\mathbf{M}}^e + \Delta t^2 V^e \frac{\partial^2 \Psi}{\partial \mathbf{F}^2}(\mathbf{F}_e(\mathbf{x}_{k+1l}^{n+1})) \right)^{-1} \approx \mathbf{Z}^e - \frac{\mathbf{Z}^e \left(\mathbf{W}^e \otimes \mathbf{W}^e \right) \mathbf{Z}^e}{1 + \mathbf{W}^e : (\mathbf{Z}^e \mathbf{W}^e)} \tag{4.49}$$

where $\mathbf{W}^e = \frac{\partial \det(\mathbf{F}^e)}{\partial \mathbf{F}^e}$ and $\mathbf{Z}^e = (\tilde{\mathbf{M}}^e + 2V^e \Delta t^2 \mu \mathbf{I})^{-1}$. Note that $\mathbf{Z}^e$ is constant and has the same symmetric block diagonal structure as $\tilde{\mathbf{M}}^e$ so its inverse can be precomputed and stored with only 6 floats (3 in 2D).

While the procedure outlined in Section 4.1.3.5 requires some elaborate notation, we note that it is effectively a standard Newton's method for FEM-discretized hyperelasticity on a single element. The only difference is that the stresses from the neighboring elements do not change when the element nodal positions change. This is inherent in the introduction of the stresses $\mathbf{P}^e$ as additional variables in Equations (4.24)-(4.25). The stresses from the neighboring elements just contribute forces that effect the right hand side of the Newton procedure, but not the matrix in the linearization. We summarize the process in Algorithm 2. In general, we run with 1-5 Newton iterations. As discussed in Section 4.1.3.5, with our novel approximation of the Hessian, the cost of solving the linear system becomes trivial. The major cost of the computation time for both XPBD and FP-PXPBD is computing the singular value decomposition of $\mathbf{F}^e$. As shown in Section 4.3.1, 4.3.2, 4.3.3 and 4.3.4 the speed of FP-PXPBD is comparable to XPBD.

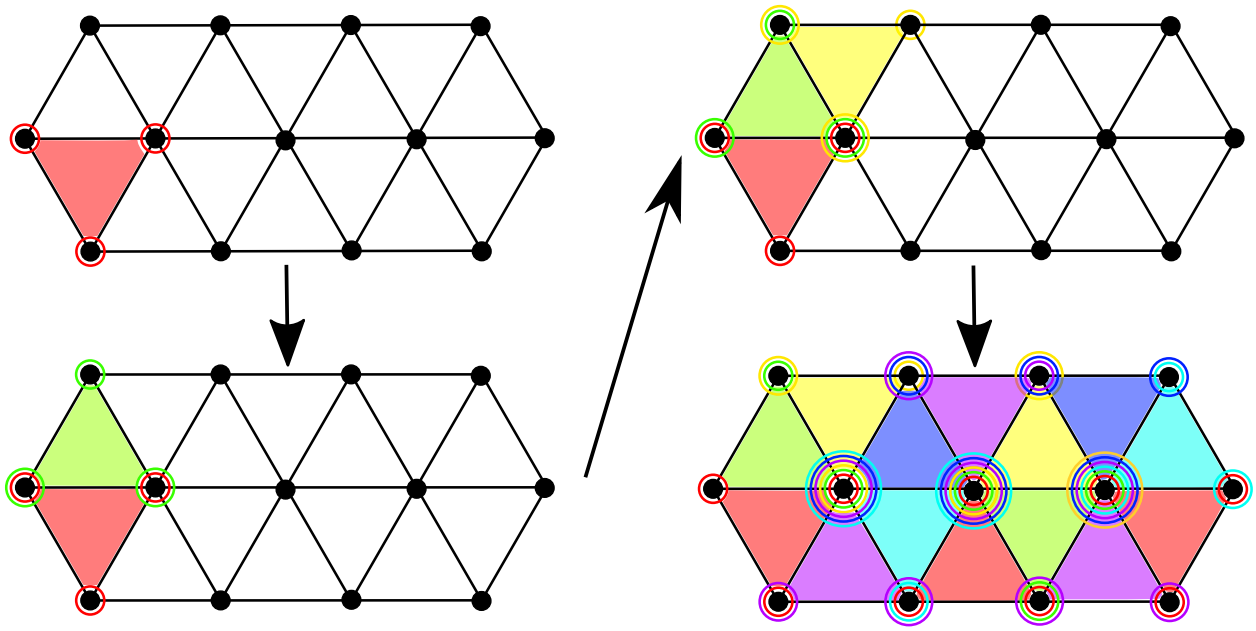


Figure 4.5: **Triangle Mesh Coloring**. A step-by-step illustration of coloring a mesh in 2D is shown. Each node registers the colors used by its incident triangles. We go over each triangle to determine its color as the minimal color unregistered by its incident nodes. All its incident nodes then register the triangle's color as used.

4.2 Parallelism

Computation of the element-wise updates must be done in parallel for optimal efficiency. Even though we use a Gauss-Seidel (as opposed to Jacobi) approach, we can achieve this with careful ordering of element-wise updates. This was discussed by Macklin and Müller [MM21], however their approach is limited to tetrahedral meshes created from hexahedral meshes. We provide a simple coloring scheme that works for all tetrahedron meshes. The coloring is done so that elements in the same color do not share vertices and can be updated in parallel without 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 elements. For each mesh element e, we find the minimal color that is not contained in the set $\cup_{\mathbf{x}_{i^e} \in e} S_{\mathbf{x}_{i^e}}$. Then, we register the color by adding it into $S_{\mathbf{x}_{i^e}}$ for each $\mathbf{x}_{i^e}$ in element e. This coloring strategy does not depend on the

topology of the mesh and requires only a one-time cost at the beginning of the simulation. The process is illustrated in Figure 4.5.

For the grid-based variation mentioned in Section 4.3.5, we do a similar process as the coloring scheme for the mesh, except the incident points of an element are now a subset of the grid nodes. Since the particle positions are interpolated by grid nodes, an element would be incident to all the grid nodes that interpolate to its incident particles on the mesh. So for each element, we choose its color as the the color with the least color index such that it is not yet registered by the incident grid nodes. The process is illustrated in Figure 4.6. Since grid nodes incident to an element change every timestep, the elements have to be recolored every timestep. We speed up the coloring process by using the coloring results from previous timestep as an initial guess. We note that Fratarcangeli et al. [FVP16] develop a randomized and effective ordering technique that could be used here as well.

Table 4.1: Timing Comparisons: runtime is measured for each frame (averaged over the course of the simulation). Each frame is run after advancing time .033.

Example	# Vertices	# Elements.	XPBD Runtime	B-PXPBD Runtime	FP-PXPBD Runtime	XPBD # iter	B-PXPBD # iter	FP-PXPBD # iter
Residual Reduction (Figure 4.3(b))	4K	17K	200ms	200ms	216ms	40	40	40
Equal Budget Comparison (Figure 4.2)	33K	149K	210ms	210ms	200ms	7	7	5
XPBD Hyperelastic (Figure 4.7)	4K	17K	22ms	-	44ms	4	-	4
XPBD Neohookean (Figure 4.8)	4K	17K	795ms	-	345ms	400	-	40
Simple Muscle (Figure 4.4)	5k	20k	-	-	160ms	-	-	4

4.3 Examples

We demonstrate our methods in a variety of representative scenarios with elastic deformation. Our approach has comparable computational complexity to XPBD, so we only provide limited run-time statistics in Table 4.1. Examples run with the corotated model (Equation 4.3) use the algorithm from [GFJ16] for its accuracy and efficiency. All the examples were run on an AMD Ryzen Threadripper PRO 3995WX CPU with 64 cores and 128 threads.

In each of the examples, we compute the mass m_i of node $\mathbf{x}_i$ from a user-specified density

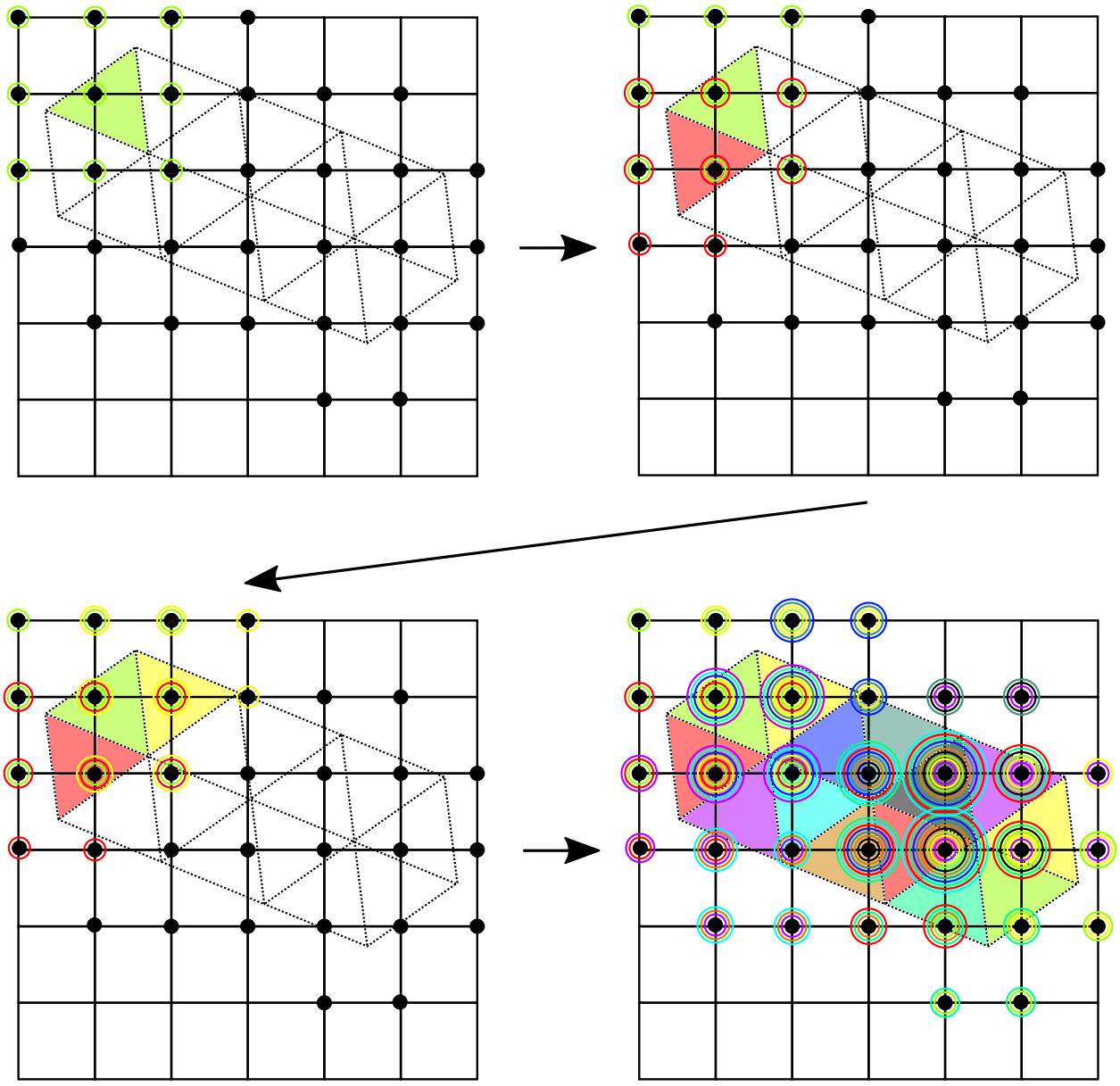


Figure 4.6: **Grid-based Mesh Coloring**. A step-by-step grid-based simplex mesh coloring scheme for 2D is shown. The illustration assumes the grid uses linear interpolation where interpolation over a cell only requires the 4 grid nodes on its corners. An element can have at maximum 12 incident grid nodes. After the first element is colored green, 9 grid nodes that are incident will register green as a used color. The elements incident to those nodes then cannot be labeled green.

ρ. We denote $\mathcal{I}$ to be the set of elements that contain node i. We define

$$m_i = \sum_{e \in \mathcal{I}} \frac{V^e \rho}{n_e} \tag{4.50}$$

Then the mass matrix is set with $M_{i\alpha j\beta} = \delta_{ij}\delta_{\alpha\beta}m_i$. We compute Lamé parameters μ and λ with Poisson ratio ν and Young's modulus E. They are computed as following:

$$\mu = \frac{E}{2(1+\nu)}, \lambda = \frac{E\nu}{(1+\nu)(1-2\nu)} \tag{4.51}$$

For all the examples in this paper we set Poisson ratio $\nu = 0.3$ and density $\rho = 10$.

4.3.1 Residual Comparison

We compare the residual reduction between XPBD, B-PXPBD and FP-PXPBD. Figure 1 (right) shows the residual reduction in a representative time step of a simple elasticity simulation. While B-PXBD and FP-PXBD continually reduce the nonlinear backward Euler residual, XPBD stagnates. Note that XPBD effectively reduces the auxiliary residual, but not the primary residual and that it makes rapid initial progress when the omission of the primary residual is well-justified. The example setup is the same as the one shown in Figure 4.3(b). B-PXPBD has blending parameter $\zeta = 0.5$.

4.3.2 Equal Budget Comparison

In Figure 4.2 we compare methods when simulated with a restricted computational budget. At the left we show Newton's method run to full-convergence (residual of Equation (4.9) less than $1e^{-8}$), which is computationally expensive. Then, we compare (from left to right) Newton's method, FP-PXPBD, B-PXPBD and XPBD when only allowed 200ms of computation time per frame. Newton's method is remarkably unstable, but the XPBD-style methods are stable and visually plausible. Here we fix the left side of the tetrahedron mesh created from a $32 \times 32 \times 32$ grid and apply gravity. The Young's modulus is $E = 1000$ and the time step is $\Delta t = 0.01$. B-PXPBD has blending parameter $\zeta = 0.1$.

4.3.3 XPBD Hyperelastic

In this example we demonstrate that XPBD is incapable of dealing with certain hyperelastic models. The top bar is simulated with XPBD and the corotated model, where the constraint is reformulated as $C_e(\mathbf{x}) = \sqrt{\Psi^{cor}(\mathbf{F}^e)}$. The middle bar is simulated with FP-PXPBD and the bottom bar XPBD as formulated in Equation (4.5). As demonstrated in Figure 4.7, the top bar becomes unstable after a couple of time steps. The reformulation at the top is a simple means of addressing general hyperelasticity with a XPBD formulation, however it does not behave stably. For this example we take a rectangular mesh and clamp both ends which are then stretched and then squeezed. We set Young's modulus as $E = 1e^4$ and time step $\Delta t = 0.01$.

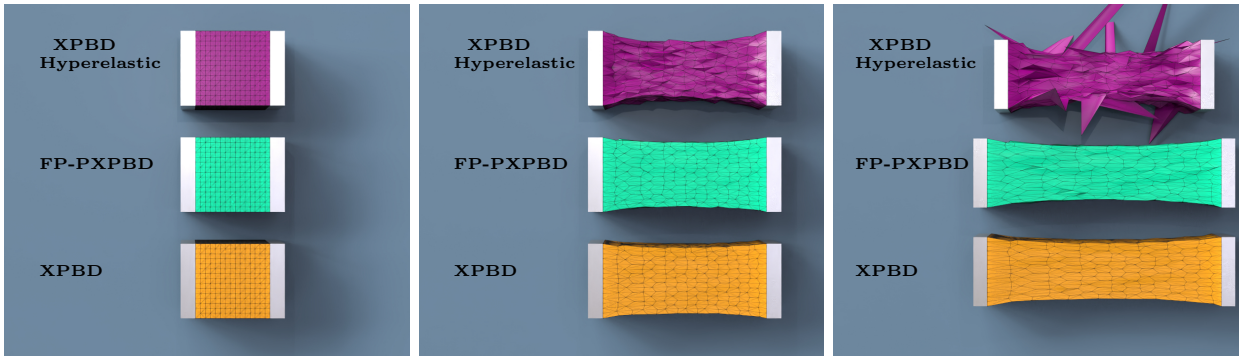


Figure 4.7: **XPBD Hyperelastic**. Defining the XPBD constraint as the square root of the hyperelastic potential is not stable (top). Results at frame 0, 10, 30 are shown.

4.3.4 XPBD Neohookean

In this example we compare XPBD and FP-PXPBD when used with the Neo-Hookean model proposed in Macklin et al.[MM21]. We generalize the low-rank approximation to the Hessian from Equation 4.48 to this model as

$$\frac{\partial^2 \Psi}{\partial \mathbf{F}^2}(\mathbf{F}_e) \approx \mu \mathbf{I} + \lambda \frac{\partial \det(\mathbf{F}^e)}{\partial \mathbf{F}^e} \otimes \frac{\partial \det(\mathbf{F}^e)}{\partial \mathbf{F}^e} \tag{4.52}$$

Similarly, we can approximate the Hessian inverse as in Equation 4.49 with $\mathbf{Z}^e = (\tilde{\mathbf{M}}^e + V^e \Delta t^2 \mu \mathbf{I})^{-1}$. The test scenario is similar to that in Section 4.3.3. We use Young's modulus $E = 1000$ and time step $\Delta t = 0.01$. Results are shown in Figure 4.8. The top bar is simulated with XPBD and is run with 100 iterations per time step. However, it does not converge to the ground truth run with Newton's method, which is shown in the bottom row. It is also visibly less volume conserving. On the other hand, FP-PXPBD converges to the ground truth with 10 iterations per time step.

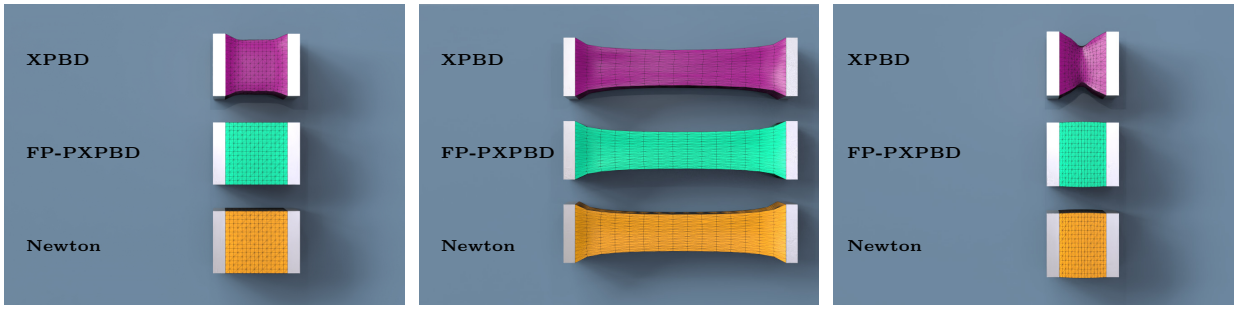
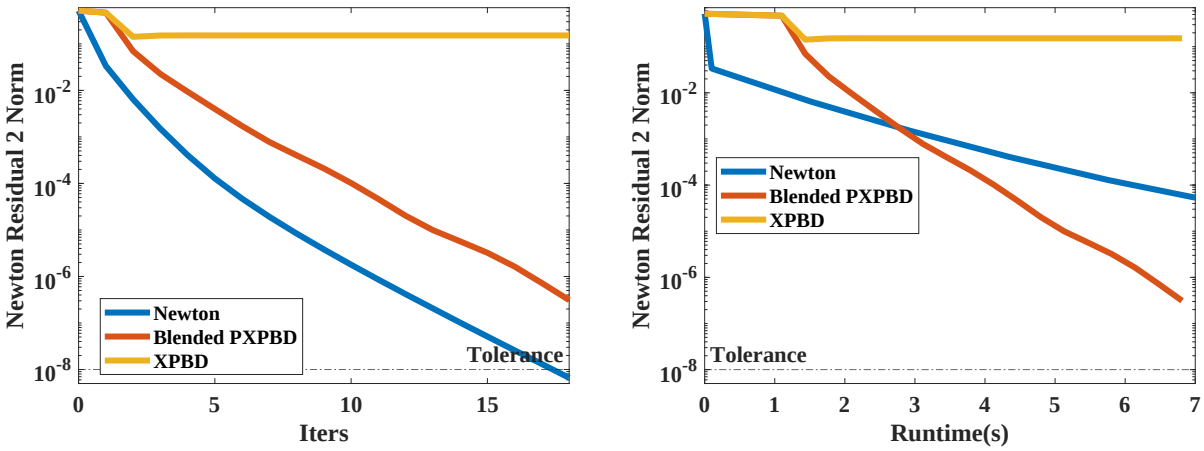


Figure 4.8: **XPBD Neohookean**. XPBD is less volume-conserving than FP-PXPBD when the cube is squeezed. Results at frame 1, 25, 52 are shown.

4.3.5 Grid-Based B-PXPBD Examples

We showcase the versatility and the robustness of B-PXPBD through a variety of collision intensive examples. We use the grid-based approach of Jiang et al. [JSS15] since this approach does not require modification of the potential energy to address collision/contact and therefore clearly demonstrates our solver performance. Here, the backward Euler degrees of freedom are over a regular grid where the tetrahedron mesh is embedded/interpolated from its motion. That is, we use B-PXPBD to solve the system of equations for implicit time stepping outlined in Jiang et al. [JSS15], but where the energy is written in the XPBD way using the constraints Equation (4.5). This requires a modification to the coloring



(a) (b)

Figure 4.9: **(a) Grid-Based Residual vs. Iterations**. The residual norm vs. iterations is plotted at a representative time step with grid-based collision. Newton's method and B-PXPBD reliably reduce the residual, but XPBD stagnates. **(b) Grid-based Residual vs. Runtime**. The residual norm vs. computation time is plotted at a representative time step. Grid-based B-PXPBD and grid-based XPBD take an extra 1 second at the beginning of each timestep to compute preprocessing data. Note that B-PXPBD achieves fasters convergence than Newton's method.

strategy used for parallel implementation (see Section 4.2 for specifics) but is otherwise a straightforward application of our techniques so we omit explicit detail.

In Figure 1, we drop 30 objects stacked on top of each other into a glass box. The objects include bunnies, dragons, balls, boxes and tori. The bunny mesh used is obtained from [TL94] The total vertex count of the mesh is around 800,000. We visualize the convergence behaviors of grid-based XPBD, grid-based B-PXPBD and Newton's method in Figure 4.9. While the residual of grid-based XPBD stagnates, grid-based B-PXPBD continually reduces the nonlinear residual. Though grid-based B-PXPBD has a convergence rate that is slower than Newton's method, it has a much lower computational budget than Newton's method. As the right plot in Figure 4.9 indicates, given the same computational budget, grid-based

B-PXPBD would reduce residual more than Newton's method. On average, the grid-based B-PXPBD takes 17.6s/frame, whereas Newton's method takes 58.9s/frame. We demonstrate more collision-intensive scenarios in Figures 4.10, 4.12 and Figure 4.11. For these examples, the Young's modulus is $E = 1000$ and CFL number is .4. B-PXPBD has blending parameter $\zeta = 0.5$.

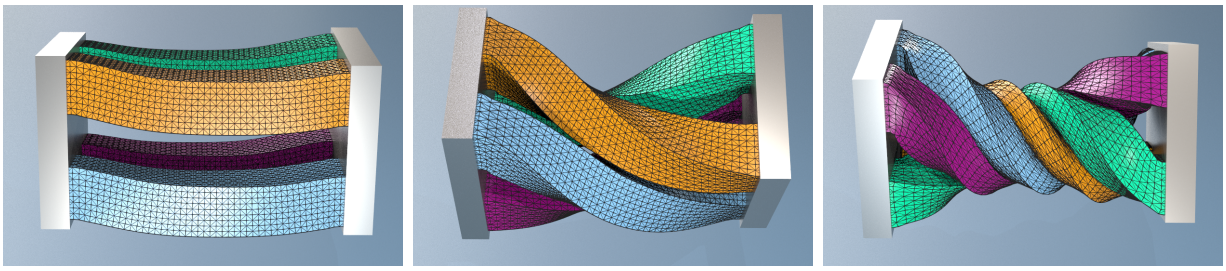


Figure 4.10: **Four Bars Twisting**. Grid-based B-PXPBD is capable of handling large deformation and complex collisions.

4.4 Discussion and Limitations

Our framework effectively addresses XPBD convergence issues with hyperelasticity and allows for generalization to arbitrary constitutive models. Furthermore, we attain the favorable stability and efficiency properties that make PBD and XPBD techniques so powerful. However, our approach does have limitations.

With B-PXPBD the blending parameter ζ can require numerous simulations to establish a useful value. FP-PXPBD is more general, but the local step may be more costly and the Sherman-Morrison formula must be applied on a case-by-case basis for different constitutive models. Lastly, while the grid-base approach of Jiang et al. [JSS15] is an easy way to handle collisions within our framework, it is not ideal for many applications since a grid-based CFL must still be used. Furthermore, it does not naturally allow for use with FP-PXBPD because the accelerations we apply in the local step are not directly applicable. We provide two approaches B-PXPBD and FP-PXPBD since both have different strengths and

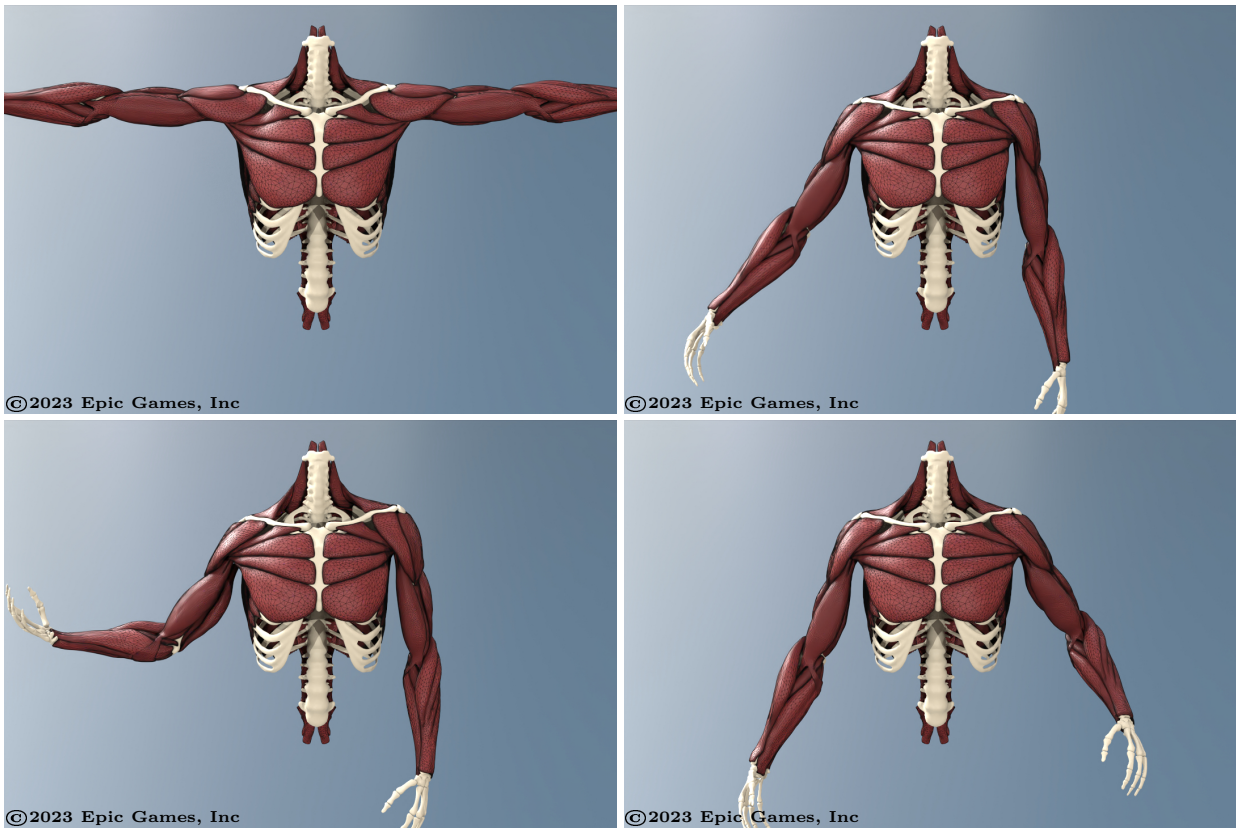


Figure 4.11: **Muscle**. Large-scale muscle simulation using grid-based B-PXPBD. Frames 0, 30, 60, 140 are shown.

weaknesses. While B-PXPBD is simple to implement (and requires only a small modificaiton to an existing XPBD code), it requires the tuning of the blending parameter and does not address general hyperelastic models models. Alternatively, FP-PXPBD can be applied to general models but it requires a larger deviation from an existing XPBD code. Furthermore, the number of iterations in the Newton method for each element effects the efficiency of the approach relative to B-PXPBD. In practice, these considerations must be weighed when deciding which approach to use.

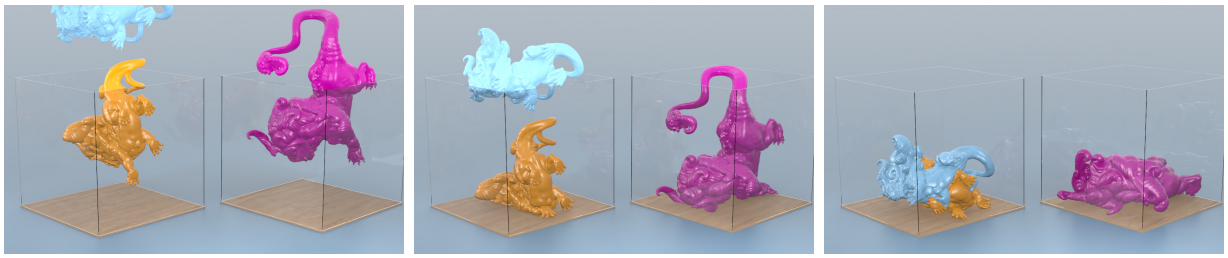


Figure 4.12: **Dropping Dragons**. Grid-based simulation with B-PXPBD exhibits many collision-driven large deformations.

CHAPTER 5

Position-Based Nonlinear Gauss-Seidel for Quasistatic Hyperelasticity

5.1 Equations

We consider continuum mechanics conceptions of the governing physics where a flow map $\boldsymbol{\phi} : \Omega^0 \times [0,T] \to \mathbb{R}^d$, $d = 2$ 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

$$R^0 \frac{\partial^2 \boldsymbol{\phi}}{\partial t^2} = \nabla^{\mathbf{X}} \cdot \mathbf{P} + \mathbf{f}^{\text{ext}} \tag{5.1}$$

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

$$\boldsymbol{\phi}(\mathbf{X},t) = \mathbf{x}_D(\mathbf{X},t),\ \mathbf{X} \in \partial\Omega^0_D \tag{5.2}$$

$$\mathbf{P}(\mathbf{X},t)\hat{\mathbf{N}}(\mathbf{X},t) = \mathbf{T}(\mathbf{X},t),\ \mathbf{X} \in \partial\Omega^0_{\hat{\mathbf{N}}} \tag{5.3}$$

where $\hat{\mathbf{N}}$ is the outward-pointing normal to the initial boundary $\partial\Omega^0$ and $\partial\Omega^0$ is split into Dirichlet ($\partial\Omega^0_D$) and Neumann ($\partial\Omega^0_{\hat{\mathbf{N}}}$) regions where the deformation and applied traction respectively are specified. Here $\mathbf{T}$ denotes externally applied traction boundary conditions. For hyperelastic materials, the first Piola-Kirchhoff stress is related to a notion of potential

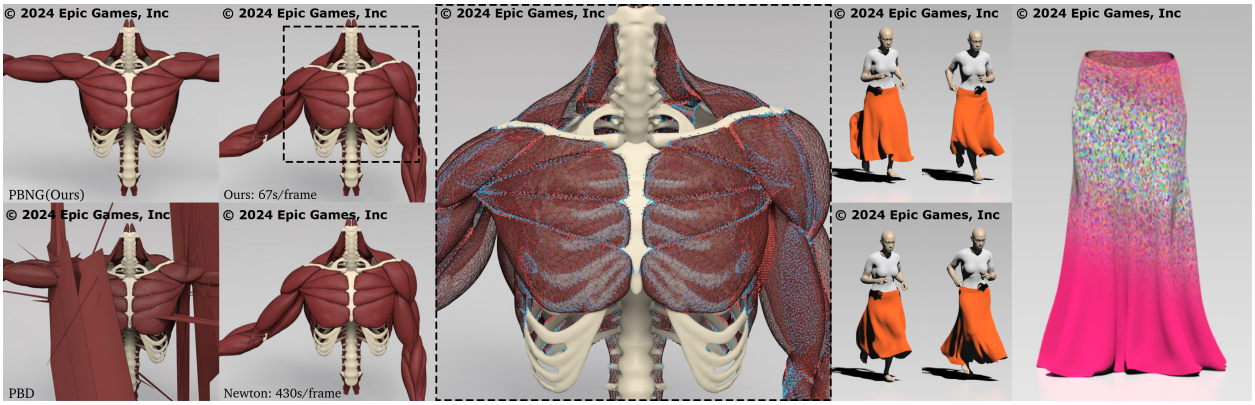


Figure 5.1: **Quasistatic Muscle Simulation with Collisions**. **Left.** Our method (PBNG) produces high-quality results visually comparable to Newton's method but with a 6x speedup. PBD (lower left) becomes unstable with this quasistatic example after a few iterations. **Middle.** In this hyperelastic simulation of muscles, we use weak constraints to bind muscles together and resolve collisions. *Red* indicates a vertex involved in a contact constraint. *Blue* indicates a vertex is bound with connective tissues. **Right.** A dress of 24K particles is simulated with MPBNG on a running mannequin. The rightmost image visualizes our multiresolution mesh.

energy density $\Psi : \mathbb{R}^{d\times d} \to \mathbb{R}$ as

$$\mathbf{P}(\mathbf{X}, t) = \frac{\partial \Psi}{\partial \mathbf{F}}(\frac{\partial \boldsymbol{\phi}}{\partial \mathbf{X}}(\mathbf{X}, t)),\ \mathrm{PE}(\boldsymbol{\phi}(\cdot, t)) = \int_{\Omega^0} \Psi(\frac{\partial \boldsymbol{\phi}}{\partial \mathbf{X}}) d\mathbf{X} \tag{5.4}$$

where $\mathrm{PE}(\boldsymbol{\phi}(\cdot, t))$ 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 (5.1)) can be neglected and the material motion is defined by a sequence of equilibrium problems

$$\mathbf{0} = \nabla^{\mathbf{X}} \cdot \mathbf{P} + \mathbf{f}^{\text{ext}} \tag{5.5}$$

subject to the boundary conditions in Equations (5.2)-(5.3). This is equivalent to the minimization problems

$$\boldsymbol{\phi}(\cdot, t) = \underset{\boldsymbol{\Upsilon} \in \mathcal{W}^t}{\text{argmin}}\ \mathrm{PE}(\boldsymbol{\Upsilon}) - \int_{\Omega_0} \mathbf{f}^{\text{ext}} \cdot \boldsymbol{\Upsilon} d\mathbf{X} - \int_{\partial\Omega^0_{\hat{\mathbf{N}}}} \mathbf{T} \cdot \boldsymbol{\Upsilon} ds(\mathbf{X}) \tag{5.6}$$

where $\mathcal{W}^t = \left\{ \boldsymbol{\Upsilon} : \Omega_0 \to \mathbb{R}^d \,|\, \boldsymbol{\Upsilon}(\mathbf{X}) = \mathbf{x}_D(\mathbf{X}, t),\ \mathbf{X} \in \partial\Omega^0_D \right\}$. We note that even though the velfocity does not affect the quasistatic equilibrium equations in Equation (5.5), the time dependence in the boundary conditions gives rise to solutions $\boldsymbol{\phi}(\mathbf{X}, t)$ that change with respect to time.

5.1.1 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]

$$\Psi^{\text{cor}}(\mathbf{F}) = \mu|\mathbf{F} - \mathbf{R}(\mathbf{F})|_F^2 + \frac{\lambda}{2}(\det(\mathbf{F}) - 1)^2. \tag{5.7}$$

Here $\mathbf{F} = \mathbf{R}(\mathbf{F})SS(\mathbf{F})$ is the polar decomposition of $\mathbf{F}$. Neo-Hookean models [BW08] have also been used since they do not require polar decomposition and recently some of them have been shown to have favorable behavior with nearly incompressible materials [SGK18].

$$\Psi^{\text{nh}}(\mathbf{F}) = \frac{1}{2}\mu|\mathbf{F}|_F^2 + \frac{\hat{\lambda}}{2}(\det(\mathbf{F}) - 1 - \frac{\mu}{\hat{\lambda}})^2. \tag{5.8}$$

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

$$\mu = \frac{E}{2(1+\nu)},\ \lambda = \frac{E\nu}{(1+\nu)(1-2\nu)}. \tag{5.9}$$

Note that we distinguish between the $\hat{\lambda}$ used in Macklin and Müeller [MM21] and the Lamé parameter λ; we discuss the reason for this in more detail in Section 5.7. We also support the stable Neo-Hookean model proposed in [SGK18]

$$\Psi^{\text{snh}}(\mathbf{F}) = \frac{1}{2}\mu(|\mathbf{F}|_F^2 - d) + \frac{1}{2}(\det(\mathbf{F}) - 1 - \frac{3\mu}{4\lambda})^2 - \frac{1}{2}\mu\log(1 + |\mathbf{F}|_F^2). \tag{5.10}$$

5.2 Discretization

We use the FEM discretization of the quasistatic problem in Equation (5.5)

$$\mathbf{f}_i(\mathbf{x}^{n+1}) + \hat{\mathbf{f}}_i^{\text{ext}} = \mathbf{0},\ \mathbf{X}_i \notin \Omega_D^0 \tag{5.11}$$

$$\mathbf{x}_i^{n+1} = \mathbf{x}_D(\mathbf{X}_i, t^{n+1}),\ \mathbf{X}_i \in \Omega_D^0. \tag{5.12}$$

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

are given as

$$\mathbf{f}_i(\mathbf{y}) = -\frac{\partial \hat{\mathrm{PE}}}{\partial \mathbf{y}_i}(\mathbf{y}) \tag{5.13}$$

$$\hat{\mathrm{PE}}(\mathbf{y}) = \hat{\mathrm{PE}}^{\Psi}(\mathbf{y}) + \hat{\mathrm{PE}}^{\mathrm{wc}}(\mathbf{y}) \tag{5.14}$$

$$\hat{\mathrm{PE}}^{\Psi}(\mathbf{y}) = \sum_{e=0}^{N^E-1} \Psi\Big(\sum_{j=0}^{N^V-1} \mathbf{y}_j \frac{\partial \chi_j}{\partial \mathbf{X}}(\mathbf{X}^e)\Big) V_e^0 \tag{5.15}$$

$$\hat{\mathbf{f}}_i^{\mathrm{ext}} = \int_{\Omega^0} \mathbf{f}^{\mathrm{ext}} \chi_i d\mathbf{X} + \int_{\partial\Omega^0_{\mathbf{N}}} \mathbf{T} \chi_i ds(\mathbf{X}) \tag{5.16}$$

where $\hat{\mathrm{PE}}^{\Psi} : \mathbb{R}^{dN^V} \to \mathbb{R}$ is the discretization of the potential energy, $\sum_{j=0}^{N^V-1} \mathbf{y}_j \frac{\partial \chi_j}{\partial \mathbf{X}}(\mathbf{X}^e)$ is the deformation gradient induced by nodal positions $\mathbf{y} \in \mathbb{R}^{dN^V}$ in tetrahedron ($d = 3$) or triangle ($d = 2$) element e with $0 \leq e < N^E$, $\frac{\partial \chi_i}{\partial \mathbf{X}}(\mathbf{X}^e)$ 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 $\hat{\mathrm{PE}}^{\mathrm{wc}} : \mathbb{R}^{dN^V} \to \mathbb{R}$ in Equation (5.14) to account for self-collisions and similar weak constraints (see Section 5.2.1). Similar to the non-discrete case, the constrained minimization problem

$$\mathbf{x}^{n+1} = \underset{\mathbf{y} \in \mathcal{W}_{\Delta x}^{n+1}}{\operatorname{argmin}} \ \hat{\mathrm{PE}}(\mathbf{y}) - \mathbf{y} \cdot \hat{\mathbf{f}}^{\mathrm{ext}} \tag{5.17}$$

where $\mathcal{W}_{\Delta x}^{n+1} = \left\{ \mathbf{y} \in \mathbb{R}^{dN^V} \,|\mathbf{y}_i = \mathbf{x}_D(\mathbf{X}_i, t^{n+1}),\ \mathbf{X}_i \in \partial\Omega_D^0 \right\}$ is equivalent to Equations (5.11)-(5.12).

5.2.1 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

$$\hat{\mathrm{PE}}^{\mathrm{wc}}(\mathbf{y}) = \frac{1}{2}\sum_{c=0}^{N^{\mathrm{wc}}-1} \mathbf{C}_c(\mathbf{y})^T \mathbf{K}_c \mathbf{C}_c(\mathbf{y}) \tag{5.18}$$

$$\mathbf{C}_c(\mathbf{y}) = \sum_{j=0}^{N^V-1} w_{0j}^c \mathbf{y}_{0j} - w_{1j}^c \mathbf{y}_{1j}. \tag{5.19}$$

Here the w_{0j}^c, w_{1j}^c are interpolation weights that sum to one and are non-negative. This creates constraints between the interpolated points $\sum_{j=0}^{N^V-1} w_{0j}^c \mathbf{y}_{0j}$ and $\sum_{j=0}^{N^V-1} w_{1j}^c \mathbf{y}_{1j}$. 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 $\mathbf{n}$. $\boldsymbol{\tau}_0^T \boldsymbol{\tau}_1 = 0$ and $||\boldsymbol{\tau}_i|| = 1, i = 0, 1$. 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. We note that, in most of our examples, the anisotropic model is used for collision constraints where $\mathbf{n}$ is the collision constraint direction.

5.3 Gauss-Seidel Notation

Our approach, PBD and XPBD all use nonlinear Gauss-Seidel to iteratively improve an approximation to the solution $\mathbf{x}^{n+1} \in \mathbb{R}^{dN^V}$ of Equation (5.11) (or equivalently, Equation (5.17)). Here we introduce detailed notation to help clarify the specific details of our method as well as its convergence behaviors. We refer to one Gauss-Seidel iteration as the process of updating all vertices once and use l to denote the iteration count as $\mathbf{x}^{n+1,l} \approx \mathbf{x}^{n+1}$. During the course of one Gauss-Seidel iteration, individual vertex degrees of freedom in the approximate solution will be updated in sub-iterates (indexed by k) which we denote as $\mathbf{x}_{(k)}^{n+1,l} \in \mathbb{R}^{dN^V}$ with $0 \leq k < N^{GS}$. For example, $\mathbf{x}_{(0)}^{n+1,l} = \mathbf{x}^{n+1,l}$ and $\mathbf{x}_{(N^V - N^D - 1)}^{n+1,l} = \mathbf{x}^{n+1,l+1}$

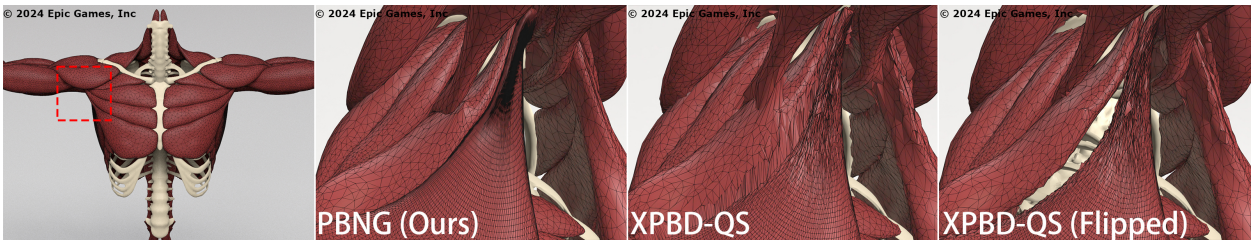


Figure 5.2: **PBNG vs XPBD**. Muscle simulation demonstrates iteration-order-dependent behavior with XPBD and quasistatics. A zoom-in view under the right armpit region is provided. Each method is run for 130 iterations. PBNG converges to the desired solution, binding the muscles closely together. XPBD-QS and XPBD-QS (Flipped) fail to converge, leaving either artifacts or gaps between the muscles. Details on XPBD-QS and XPBD-QS (Flipped) can be found in Section 5.9.

for PBNG. To further clarify, with PBD/XPBD in the k^{th} sub-iterate, the nodes in the k^{th} constraint will be projected/solved for and so N^{GS} will be equal to the total number of constraints. In our position-based approach, in the k^{th} sub-iterate, only the d components of 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 $k-1^{\text{th}}$ sub-iterate.

5.4 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 (5.1)

$$\sum_{j=0}^{N^V-1} m_{ij}\left(\frac{\mathbf{x}_j^{n+1}-2\mathbf{x}_j^n+\mathbf{x}_j^{n-1}}{\Delta t^2}\right)=\mathbf{f}_i(\mathbf{x}^{n+1})+\mathbf{f}_i^{\text{ext}},\ \mathbf{X}_i\notin\Omega_D^0. \tag{5.20}$$

Here $m_{ii} = \int_{\Omega^0} R^0 \chi_i d\mathbf{X}$ and $m_{ij} = 0$, $j \neq i$ are entries in the mass matrix. However, Macklin et al. [MMC16] require that the discrete potential energy in Equation (5.15) is of the form

$$\hat{PE}^{\Psi}(\mathbf{y}) = \sum_{c=0}^{2N^E-1} \frac{1}{2\alpha_c} C_c^2(\mathbf{y}),\ \mathbf{y} \in \mathbb{R}^{dN^E}. \tag{5.21}$$

To demonstrate the connection between Equation (5.20) and PBD, Macklin et al. [MMC16] develop XPBD. It is based on the total Lagrange multiplier formulation

$$\sum_{j=0}^{N^V-1} m_{ij} \left(\mathbf{x}_j^{n+1} - \hat{\mathbf{x}}_j\right) - \sum_{c=0}^{P-1} \frac{\partial C_c}{\partial \mathbf{x}_i}(\mathbf{x}^{n+1}) \lambda_c^{n+1} = 0,\ \mathbf{X}_i \notin \Omega_D^0 \tag{5.22}$$

$$C_c(\mathbf{x}^{n+1}) + \frac{\alpha_c}{\Delta t^2} \lambda_c^{n+1} = 0,\ 0 \leq c < P \tag{5.23}$$

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 $\mathbf{x}^{n+1} \in \mathbb{R}^{dN^V}$ in Equations (5.22)-(5.23) is the same in the solution to Equation (5.20). P is the number of constraints. Macklin et al. [MMC16] use a per-constraint Gauss-Seidel update of Equations (5.22)-(5.23)

$$\mathbf{x}_{i(k+1)}^{n+1,l} = \mathbf{x}_{i(k)}^{n+1,l} + \Delta \mathbf{x}_{i(k+1)}^{n+1,l},\ \mathbf{X}_i \notin \Omega_D^0 \tag{5.24}$$

$$\Delta \mathbf{x}_{i(k+1)}^{n+1,l} = \frac{\Delta \lambda_{(k+1)c_k}^{n+1,l}}{m_{ii}} \frac{\partial C_{c_k}}{\partial \mathbf{x}_i}(\mathbf{x}_{(k)}^{n+1,l}) \tag{5.25}$$

$$\Delta \lambda_{(k+1)c_k}^{n+1,l} = \frac{-C_{c_k}(\mathbf{x}_{(k)}^{n+1,l}) + \frac{\alpha_{c_k}}{\Delta t^2} C_{c_k}(\mathbf{x}_{(k)}^{n+1,l})}{\sum_{j=0}^{N^V-1} \frac{1}{m_{jj}} \sum_{\beta=0}^{d-1} \left(\frac{\partial C_{c_k}}{\partial x_{j\beta}}(\mathbf{x}_{(k)}^{n+1,l})\right)^2 + \frac{\alpha_{c_k}}{\Delta t^2}}. \tag{5.26}$$

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 (5.20). [CHC23] isolates the root cause of this as the omission of the residual of Equation (5.22) in the update of the Lagrange multiplier in Equation (5.26) 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 5.3.

5.4.1 Quasistatics

As noted by Macklin et al. [MMC16], the XPBD update in Equations (5.24)-(5.26) 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 (5.5)), albeit with the external forcing terms omitted. More precisely, define $\boldsymbol{\phi}_s$ to be a solution of the problem

$$sR^0 \frac{\partial^2 \boldsymbol{\phi}_s}{\partial t^2} = \nabla^{\mathbf{X}} \cdot \mathbf{P} + s\mathbf{f}^{\text{ext}}. \tag{5.27}$$

subject to the same boundary conditions in Equations (5.2)-(5.3). This is equivalent to scaling the α_c that would appear in Equation (5.1) (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 (5.27) 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 9.3 of the paper. Also, note that this is not the case in the XPBD formulation where $\alpha_c > 0$, as it is the inverse stiffness term.

Unfortunately, XPBD cannot be trivially modified to run quasistatic problems. For example, omitting the mass terms on the left-hand side of Equation (5.22) makes the Gauss-Seidel update in Equations (5.24)-(5.26) 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 (5.25). 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 5.9.3 for

example). 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 5.2). 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 it could be added stably.

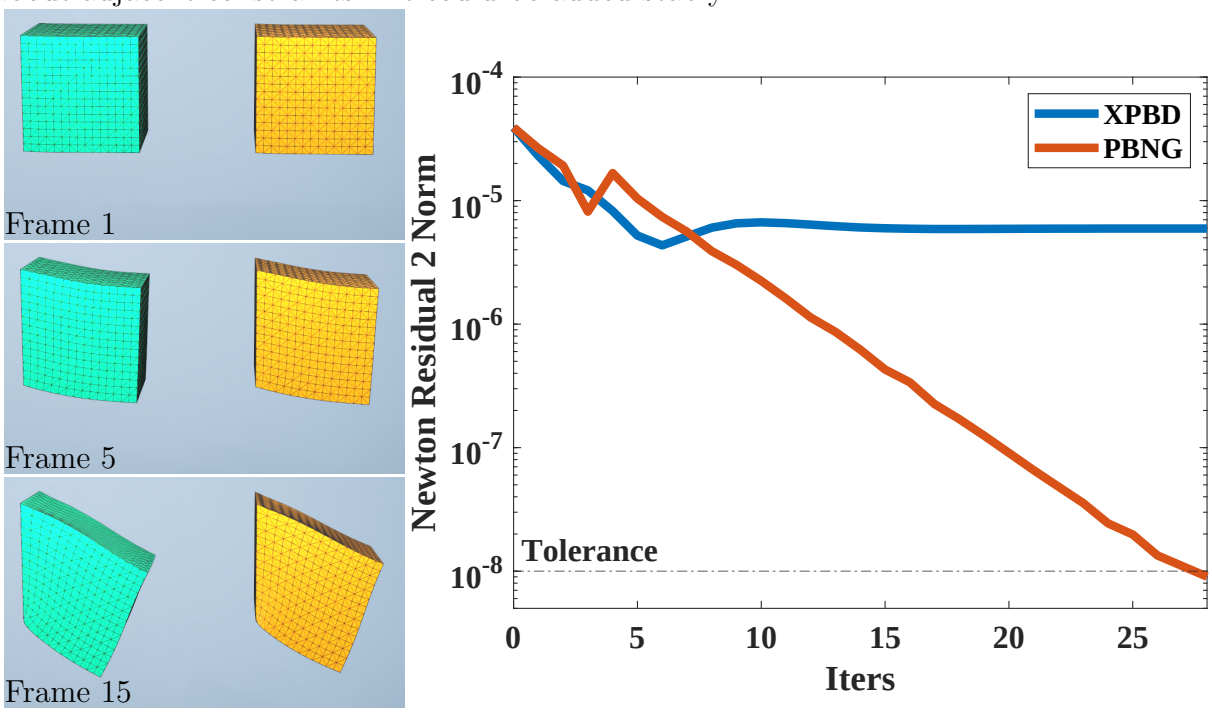


Figure 5.3: **Left**. Clamped blocks under gravity. The green block is XPBD, and the yellow one is PBNG. **Right**. PBNG is able to reduce the Newton residual to the tolerance, whereas XPBD's residual stagnates.

5.5 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 (5.11) 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 Figure 5.6(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

$$\begin{aligned} \mathbf{x}_{(k+1)i_k}^{n+1,l} &= \mathbf{x}_{(k)i_k}^{n+1,l} + \Delta\mathbf{x}_{(k+1)i_k}^{n+1,l} \\ \Delta\mathbf{x}_{(k+1)i_k}^{n+1,l} &= \underset{\Delta\mathbf{y} \in \mathbb{R}^d}{\text{argmin}} \; \hat{PE}(\mathbf{x}_{(k)}^{n+1,l} + \tilde{\mathbf{C}}^{i_k}\Delta\mathbf{y}) - \Delta\mathbf{y} \cdot \hat{\mathbf{f}}_{i_k}^{\text{ext}}. \end{aligned} \tag{5.28}$$

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

$$\mathbf{0} = \mathbf{f}_{i_k}(\mathbf{x}_{(k)}^{n+1,l} + \tilde{\mathbf{C}}^{i_k}\Delta\mathbf{x}_{(k+1)i_k}^{n+1,l}) + \hat{\mathbf{f}}_{i_k}^{\text{ext}}. \tag{5.29}$$

We use a single step of a modified Newton's method to approximate the solution of Equation (5.29) for $\Delta\mathbf{x}_{(k+1)i_k}^{n+1,l} \in \mathbb{R}^d$. We use $\Delta\mathbf{x}_{(k+1)i_k}^{n+1,l} = \mathbf{0}$ 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

$$\Delta\mathbf{x}_{(k+1)i_k}^{n+1,l} = \left(\mathbf{A}_{(k+1)i_k}^{n+1,l}\right)^{-1}\left(\mathbf{f}_{i_k}(\mathbf{x}_{(k)}^{n+1,l}) + \hat{\mathbf{f}}_{i_k}^{\text{ext}}\right). \tag{5.30}$$

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

5.5.1 Modified Hessian

We choose the modified energy Hessian $\mathbf{A}_{(k+1)i_k}^{n+1,l}$ 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

$$\begin{aligned} \frac{\partial f_{i_k\alpha}}{\partial y_{i_k\beta}}(\mathbf{y}) = &-\sum_{e=0}^{N^E-1}\sum_{\delta,\gamma=0}^{d-1} \mathcal{C}_{\alpha\gamma\beta\delta}^e(\mathbf{y})\frac{\partial N_{i_k}^e}{\partial X_\gamma}\frac{\partial N_{i_k}^e}{\partial X_\delta}V_e^0 - \\ &\sum_{c=0}^{N^{\text{wc}}-1}\left(w_{0i_k}^c - w_{1i_k}^c\right)^2 K_{c\alpha\beta},\ 0 \le \alpha,\beta < d \end{aligned} \tag{5.31}$$

where $\mathcal{C}^e_{\alpha\gamma\beta\delta}(\mathbf{y}) = \frac{\partial^2\Psi}{\partial F_{\beta\delta}\partial F_{\alpha\gamma}}(\sum_{j=0}^{N^V-1}\mathbf{y}_j\frac{\partial N^e_j}{\partial \mathbf{X}})$ is the Hessian of the potential energy density evaluated at the deformation gradient in element e and $K_{c\alpha\beta}$ is the stiffness tensor associated with weak-constraints.

The primary cost in Equation (5.31) 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 that (unlike [TSI05, SGK19]) does not require the singular value decomposition of the element deformation gradient $\sum_{j=0}^{N^V-1}\mathbf{y}_j\frac{\partial N^e_j}{\partial \mathbf{X}}$. Also note that 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

$$\tilde{\mathcal{C}}^e_{\alpha\gamma\beta\delta}(\mathbf{y}) = 2\mu\delta_{\alpha\beta}\delta_{\gamma\delta} + \lambda J(F^e)^{-1}_{\alpha\gamma}(\mathbf{y})J(F^e)^{-1}_{\beta\delta}(\mathbf{y}). \tag{5.32}$$

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 $\mathbf{F}^e(\mathbf{y}) = \sum_{j=0}^{N^V-1}\mathbf{y}_j\frac{\partial N^e_j}{\partial \mathbf{X}}$. 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 note that this approximation works for any isotropic potential energy density Ψ where μ and λ are the associated Lamé parameters computed from Young's modulus E. We discuss the motivation for this simplification in Chapter 7, 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 this convention, our symmetric positive definite modified nodal Hessian is of the form

$$A^{n+1}_{(k+1)i_k\alpha\beta} = \sum_{e=0}^{N^E-1}\sum_{\delta,\gamma=0}^{d-1}\tilde{\mathcal{C}}^e_{\alpha\gamma\beta\delta}(\mathbf{x}^{n+1,l}_{(k)})\frac{\partial N^e_{i_k}}{\partial X_\gamma}\frac{\partial N^e_{i_k}}{\partial X_\delta}V^0_e + \tag{5.33}$$

$$\sum_{c=0}^{N^{\mathrm{wc}}-1}\left(w^c_{0i_k} - w^c_{1i_k}\right)^2 K_{c\alpha\beta},\ 0\le\alpha,\beta<d. \tag{5.34}$$

5.5.2 Collision against kinematic bodies

We add support for hard collision constraints against kinematic geometry (collision bodies that do not deform). 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 $\Delta\bar{\mathbf{x}}^{n,l}_{(k)i} = (\mathbf{I} - \mathbf{n}_i\mathbf{n}_i^T)\Delta\mathbf{x}^{n,l}_{(k)i}$ to allow for sliding tangential to the constraint surface.

5.5.3 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 with most Gauss-Seidel approaches the convergence rate of PBNG may decrease with iteration count (see Figure 5.12 for details). We investigated two simple acceleration techniques to help mitigate this: the Chebyshev semi-iterative method (as in [Wan15]) and SOR. The Chebyshev method uses the update $\mathbf{x}^{n+1,l+1} = \omega_{l+1}(\gamma(\mathbf{x}^{n+1,l+1}_{\text{PBNG}} - \mathbf{x}^{n+1,l}) + \mathbf{x}^{n+1,l} - \mathbf{x}^{n+1,l-1}) + \mathbf{x}^{n+1,l-1}$. 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 $\mathbf{x}^{n+1,l+1} = \omega(\mathbf{x}^{n+1,l+1}_{\text{PBNG}} - \mathbf{x}^{n+1,l-1}) + \mathbf{x}^{n+1,l-1}$. We use $\omega = 1.7$ for this under-relaxation parameter. Chebyshev and SOR behave similarly in terms of residual reduction and visual appearance (see Figure 5.12).

5.6 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 (5.17). We use the sum of a membrane hyperelastic potential and a bending term:

$$\tilde{\text{PE}}(\mathbf{y}) = \sum_{\hat{t}} \Psi^{\text{cm}}(\mathbf{F}^{\text{mem}}_{\hat{t}}(\mathbf{y}))A_{\hat{t}} + \frac{1}{2}k_b \sum_e \theta_e(\mathbf{y})^2. \tag{5.35}$$

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

$$\Psi^{\text{cm}}(\mathbf{F}^{\text{mem}}) = \mu^{\text{cm}}|\mathbf{F}^{\text{mem}} - \mathbf{R}(\mathbf{F}^{\text{mem}})|_F^2 + \frac{\lambda^{\text{cm}}}{2}(J(\mathbf{F}^{\text{mem}}) - 1)^2. \tag{5.36}$$

Here μ^{cm} and λ^{cm} are derived from Young's modulus E^{cm} in a similar fashion as μ and λ in Section 5.5. $\mathbf{F}^{\text{mem}}_{\hat{t}}(\mathbf{y}) = \sum_i \mathbf{y}_i \frac{\partial \hat{\chi}_i}{\partial \mathbf{X}}(\mathbf{X}_{\hat{t}}) \in \mathbb{R}^{3\times 2}$ is the deformation gradient computed over the triangle $\hat{t}$ where $\hat{\chi}_i$ are piecewise linear interpolating functions over the triangles. $A_{\hat{t}}$ is the reference area of the triangle $\hat{t}$ and $J(\mathbf{F}^{\text{mem}}) = \sqrt{\det(\mathbf{F}^{\text{mem}T}\mathbf{F}^{\text{mem}})}$ denotes the multiplicative change in the triangle area under motion defined by $\mathbf{y}$. The term $\mathbf{R}(\mathbf{F}^{\text{mem}})$ in Equation (5.36) is the rotational part of the polar decomposition of $\mathbf{F}^{\text{mem}} = \mathbf{R}(\mathbf{F}^{\text{mem}})SS(\mathbf{F}^{\text{mem}})$ with the convention that $\mathbf{R}(\mathbf{F}^{\text{mem}}) \in \mathbb{R}^{3\times 2}$ has orthogonal columns and $SS(\mathbf{F}^{\text{mem}}) \in \mathbb{R}^{2\times 2}$ is symmetric.

For bending resistance in Equation (5.35), we adopt a similar approach to Baraff and Witkin [BW98]. For each edge e with vertices $\mathbf{y}^0_e, \mathbf{y}^1_e$ that is incident to two triangles with unit normals $\mathbf{n}^1_e(\mathbf{y}), \mathbf{n}^2_e(\mathbf{y})$, we define $\theta_e(\mathbf{y}) \in [0, \pi)$ as the bending angle where $\theta_e(\mathbf{y}) = \text{atan}(\frac{(\mathbf{n}^1_e(\mathbf{y})\times\mathbf{n}^2_e(\mathbf{y}))\cdot(\mathbf{y}^1_e - \mathbf{y}^0_e)}{\mathbf{n}^1_e(\mathbf{y})\cdot\mathbf{n}^2_e(\mathbf{y})})$. k_b is the bending stiffness parameter. Note that as in [BW98], we do not use an area weighting on the bending term.

5.6.1 Modified Hessian

Similar to Section 5.5.1, we modify the Hessians of the above models to ensure positive semi-definiteness. We make the simple approximation $\frac{\partial^2 \Psi^{\text{cm}}}{\partial F^{\text{mem}}_{\alpha\gamma} \partial F^{\text{mem}}_{\beta\delta}}(\mathbf{y}) \approx 2\mu^{\text{cm}}\delta_{\alpha\beta}\delta_{\gamma\delta} +$

$\frac{\lambda^{\text{cm}}}{4} J^2 L^{\text{mem}}_{\alpha\gamma} L^{\text{mem}}_{\beta\delta}$ where
$\mathbf{L}^{\text{mem}} = \mathbf{F}^{\text{mem}}((\mathbf{F}^{\text{mem}})^T\mathbf{F}^{\text{mem}})^{-T} + \mathbf{F}^{\text{mem}}((\mathbf{F}^{\text{mem}})^T\mathbf{F}^{\text{mem}})^{-1}$. For the bending model, we use the rank one approximation

$$\frac{1}{2}\frac{\partial^2\theta_e^2}{\partial\mathbf{x}^2} \approx \frac{\partial\theta_e}{\partial\mathbf{x}} \otimes \frac{\partial\theta_e}{\partial\mathbf{x}}. \tag{5.37}$$

The bending potential in Equation (5.35) is quadratic in θ_e so a rank-one term consisting of the outer-product in Equation (5.37) is a simple PSD Hessian approximation. See the Chapter 7 for more details.

5.6.2 Multiresolution Acceleration

When a solid and a piece of cloth have the same particle count, the max topological distance between the particles in the cloth is bigger than that in the solid because a cloth is essentially a 2D object. Since Gauss-Seidel only does local updates, it needs to propagate information further for cloth to converge. In practice, cloth will look overly stretchy when propagation has not proceeded sufficiently (see Section 5.9.5.). This slowed convergence with increased resolution is the motivation for techniques like multigrid [Bra77, WLF18]. We develop a multiresolution techniques along these lines: MPBNG (or multilayer PBNG). For a given piece of cloth χ_0, we remesh it to create a nested mesh hierarchy $\chi_0 \supset \chi_1 \supset \chi_2 \supset \chi_n$. Usually the number of vertices is reduced by approximately a factor of 4. These layers equally divide the stiffness and mass density of the original material as in a multi-species continua model [AC76]. We then bind each two successive layers using weak constraints (Equation (5.15)) and define a multi-layer iteration as follows: 4-6 PBNG iterations are first applied on the coarsest level. Then all particles on $\cup_{i\neq n}\chi_i$ are iterated over in a standard PBNG manner. This is done in parallel, utilizing coloring that takes into account all layers and the constraints between them. After the multi-layer iterations have been applied, we do a final pass of 6-8 PBNG iterations on the finest layer. The weak constraints are visualized in Figure 5.4. Note that we found this three-pass approach to be more effective than a

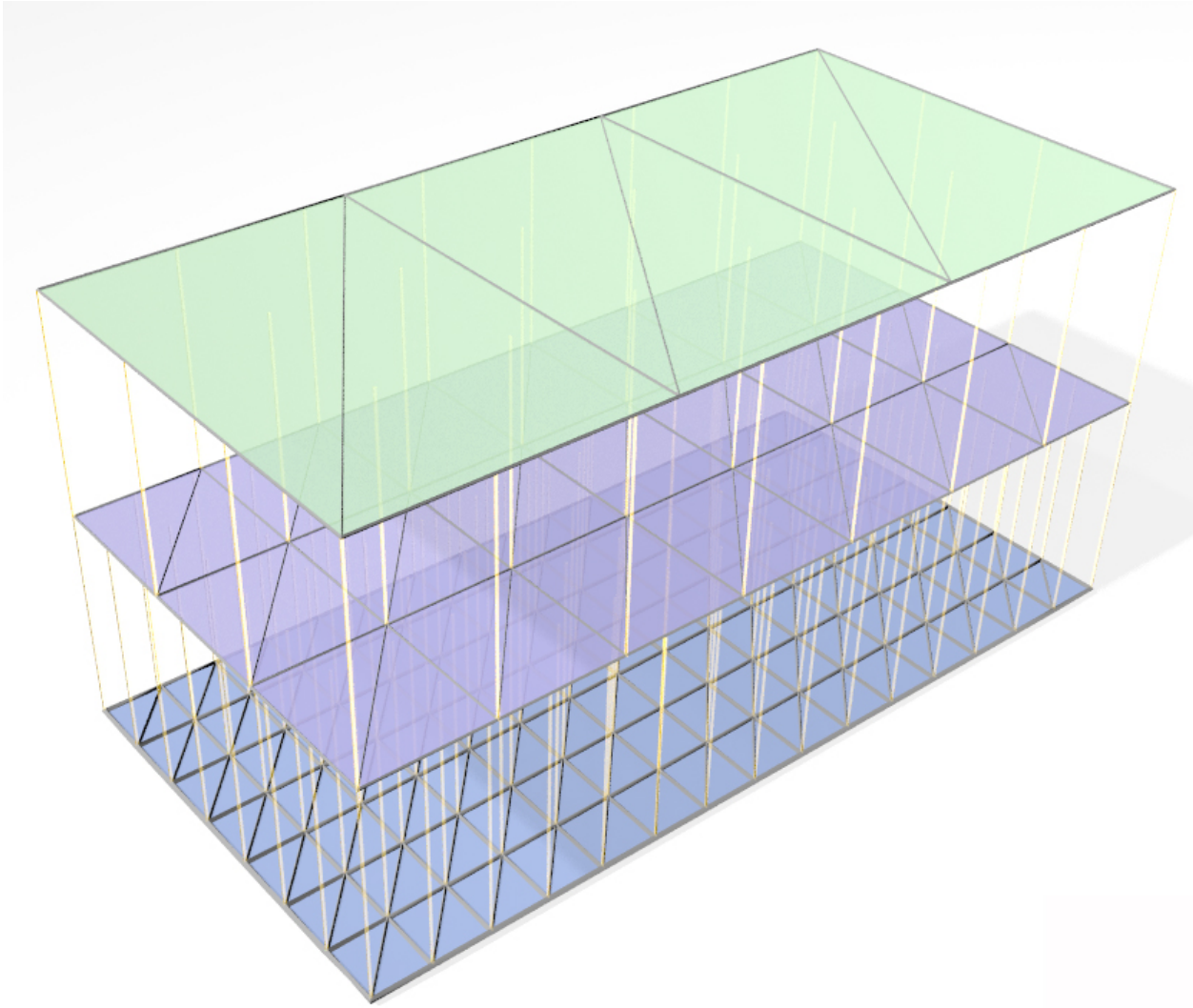


Figure 5.4: **MPBNG Illustration**. We visualize the cloth multi-resolution hierarchy. Straight lines illustrate constraints between vertices on the finer level to their targets on the coarser level.

standard multigrid V-cycle in practice.

After each multiresolution iteration l, we reduce the stiffness of the weak constraints and the mass on the coarse layers. We define a scaling factor $\kappa_l = \sqrt{1 - \frac{l^2}{r_{\text{des}}^2}}$ where r_{des} is a user-specified radius of descent. We choose $r_{\text{des}} = k_{\max} + r_{\text{pad}}$ where $k_{\max}$ is the maximal number of iterations and $r_{\text{pad}} \in [.05, 2]$. In practice, r_{pad} does require much tuning and $r_{\text{pad}} = .05$ typically suffices. This factor is multiplied to the mass of the nodes on the coarse layers (i.e.

all layers except χ_0) and k_c of all weak constraints. As l becomes larger, κ_l decreases to a small number, which means the problems converges to the original one. The same approach can be applied to solid simulation as well, however we observed that MPBNG achieves less residual reduction than PBNG in this case (with a fixed computational budget).

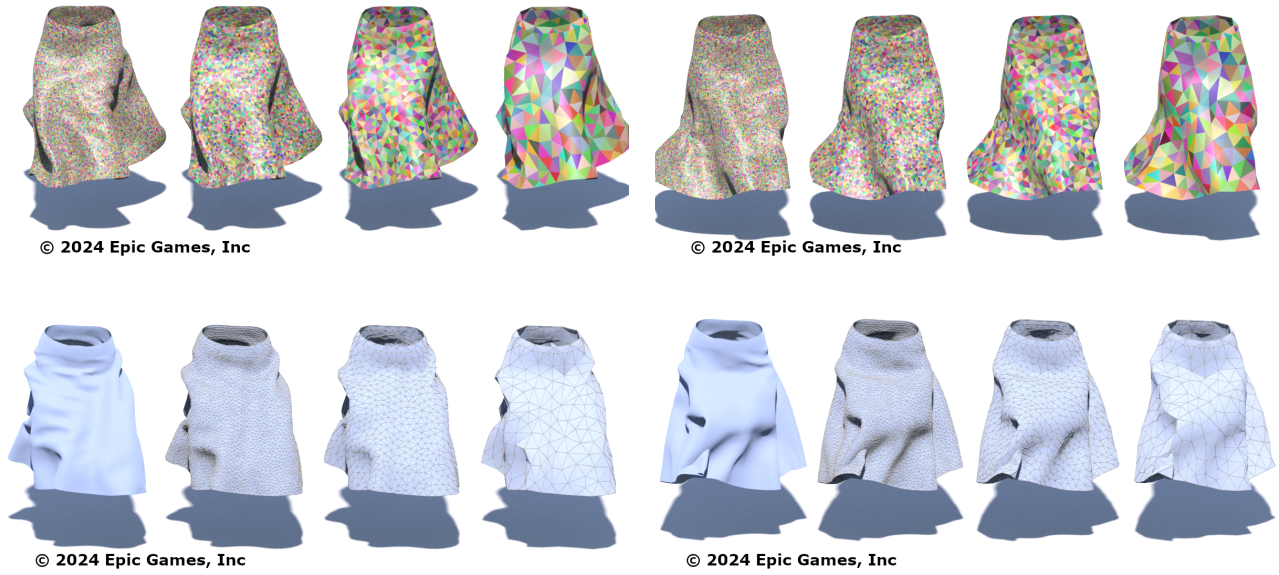


Figure 5.5: **Multiresolution Dress**. We illustrate the multiresolution meshes used with our MPBNG approach in a representative clothing simulation.

5.7 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 (5.9). This relationship is based on the assumption of linear dependence of stress on strain, or quadratic potential energy density

$$\Psi^{\text{le}}(\mathbf{F}) = \mu \text{tr}(\boldsymbol{\epsilon}^2(\mathbf{F})) + \frac{\lambda}{2}\text{tr}(\boldsymbol{\epsilon}(\mathbf{F}))^2 \tag{5.38}$$

$$\boldsymbol{\epsilon} = \frac{1}{2}(\mathbf{F} + \mathbf{F}^T) - \mathbf{I}. \tag{5.39}$$

Furthermore, Equation (5.9) is derived from the model in Equation (5.38) 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 $I_\alpha : \mathbb{R}^{d\times d} \to \mathbb{R}$, $0 \leq \alpha < d$

$$I_0(\mathbf{F}) = \operatorname{tr}(\mathbf{F}^T\mathbf{F}),\ I_1(\mathbf{F}) = \operatorname{tr}((\mathbf{F}^T\mathbf{F})^2),\ I_2(\mathbf{F}) = \det(\mathbf{F}) \tag{5.40}$$

$$\Psi(\mathbf{F}) = \hat{\Psi}(I_0(\mathbf{F}), I_1(\mathbf{F}), I_2(\mathbf{F})). \tag{5.41}$$

See [GS08] for more detailed derivation. Note, when $d = 2$, $I_1(\mathbf{F}) = \operatorname{tr}((\mathbf{F}^T\mathbf{F})^2)$ is not used. With this convention, the Hessian of the potential energy density is of the form

$$\frac{\partial^2\Psi}{\partial\mathbf{F}^2} = \sum_{\alpha=0}^{d-1} \frac{\partial\hat{\Psi}}{\partial I_\alpha}\frac{\partial^2 I_\alpha}{\partial\mathbf{F}^2} + \sum_{\alpha,\beta=0}^{d-1} \frac{\partial^2\hat{\Psi}}{\partial I_\alpha \partial I_\beta}\frac{\partial I_\alpha}{\partial\mathbf{F}} \otimes \frac{\partial I_\beta}{\partial\mathbf{F}}. \tag{5.42}$$

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 (5.8). See Chapter 7 for derivation details.

We choose our approximate Hessian in Equation 9 in the paper based on this fact. That is, by omitting all but the first and last terms in Equation (5.42), 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)

$$\tilde{\mathcal{C}} = \mu\frac{\partial^2 I_0}{\partial\mathbf{F}^2} + \lambda\frac{\partial I_{d-1}}{\partial\mathbf{F}} \otimes \frac{\partial I_{d-1}}{\partial\mathbf{F}}. \tag{5.43}$$

Again, see Chapter 7 for derivation details.

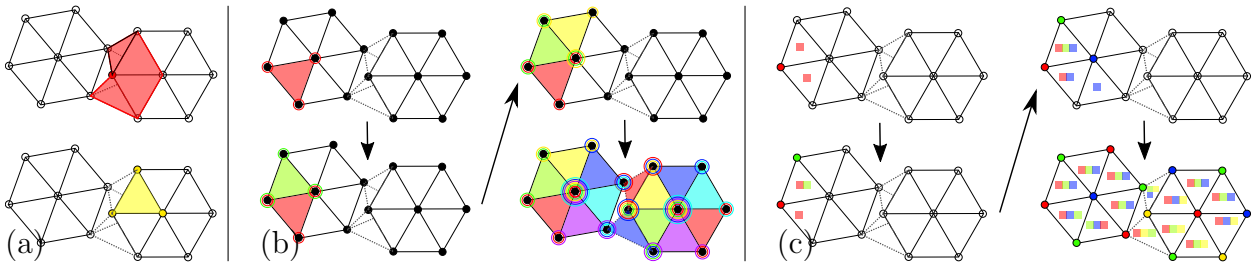


Figure 5.6: **(a) Dual Coloring** . Node based coloring (top) is contrasted with constraint based coloring (bottom). When a node is colored as red, its incident elements register red as used colors. When a constraint is colored yellow, its incident particles register yellow as used colors. **(b) Constraints-Based 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.

5.8 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

Table 5.1: **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.

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 5.2: **Methods Comparisons**. We show runtime per frame for different methods. Each frame is $\frac{1}{30}$ seconds.

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)

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 $\mathbf{x}_i$, 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. 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. We provide more details on the coloring process in Figure 5.6. The performance gain is listed in Table 5.1. Note that we use the omp parallel directive from Intel's OpenMP library for parallelizing the updates.

Table 5.3: **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.

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

5.8.1 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 (see Chapter 7). 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 5.1, 5.2 and 5.15), our algorithm takes less than 680ms/frame for recoloring, while the actual simulation takes a total of 67s to run.

5.9 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 used in the algorithm from [GFJ16] for its accuracy and efficiency. Example 5.9.2.4,

5.9.5 and 5.9.6 are dynamic simulations. The rest are quasistatic simulations. All the examples use Poisson's ratio $\nu = 0.3$. We compare PBNG, PBD, XPBD, XPBD-QS 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 5.3, we provide comprehensive performance statistics for PBNG. In Table 5.2, we provide runtime comparisons between PBNG and the other methods. Note that for efficiency we did not use a line search with Newton's method in our experiments. We note adding line search requires evaluating Newton residuals multiple times, which would further increase cost. As in Figure 5.7, line search would further reduce number of Newton iterations, making it less stable.

5.9.1 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 $E = 10^5 Pa$. There is no gravity. The simulation is quasistatic. We compare performance between Newton's method, PBD, PBNG and XPBD as described in Section 5.4. In Figure 5.7, 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 5.7, 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 5.7.

5.9.1.1 Resolution Comparison

In this example, we demonstrate PBNG's versatility by running the block stretching and twisting with various resolutions. As shown in Figure 5.8, 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.

5.9.1.2 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 5.9. The blocks from top to bottom are run with corotated (Equation 5.7), stable Neo-Hookean (Equation 5.10) and Neo-Hookean (Equation 5.8) models respectively. With 40 iterations per frame, they are all visually plausible.

5.9.1.3 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 (see Figure 5.10). 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 and 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.

Table 5.4: **Runtime Breakdown:** We compare the runtimes of linear Gauss-Seidel and PBNG. Newton Overhead refers to the cost of computing the Newton residual and explicit Hessian in each iteration (a cost which PBNG does not require).

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

We observe that PBNG has several advantages. 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 5.4. PBNG 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. PBNG achieves clearly superior nonlinear system residual reduction, as shown in Figure 5.10. Lastly, 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.

5.9.1.4 Approximate Hessian Comparison

In this example we demonstrate the efficacy of our Hessian approximation (Equation 5.32). All four blocks have 4K particles and 20K elements (see Figure 5.11). The top three bars are 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 5.32. All approaches using the exact Hessian lead to unstable results, while our approximation leads to a correct converged result.

5.9.1.5 Acceleration Comparison

In this example, we compare the effects of the Chebyshev semi-iterative method and the SOR method. In Figure 5.12, 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 5.12 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.

5.9.2 Collisions

Here we demonstrate our approach with examples that require collision resolution. For all examples in this section, we use a time step of $\Delta t = .002s$ and detect collision every time step. We illustrate how weak constraints are dynamically created for collision with a simple two-block colliding example.

5.9.2.1 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 5.13 as red nodes in the mesh.

5.9.2.2 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 5.14. We show PBNG's capability of handling collision-intensive scenarios. The example is run with $\Delta t = 0.002s$, $R^0 = 10kg/m^3$, Young's modulus $E = 3000Pa$ and weak constraint stiffness $k_n = 10^8$ and $k_\tau = 0$.

5.9.2.3 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 $E = 10^5 Pa$, and the 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 5.15. PBNG takes 67 seconds to simulate a frame, while Newton's method takes 430s. In Figure 5.1, 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 5.1, we show PBD becomes unstable. In Figure 5.2, we demonstrate sub-iteration order-dependent behavior with XPBD-QS. 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.

5.9.2.4 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 Figure 5.16. 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 $R^0 = 10kg/m^2, E^{\text{cod}} = 10000Pa, k_b = .05$. 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.

5.9.3 XPBD with Varying Stiffness

In this example, we demonstrate that XPBD-QS fails to resolve quasistatic problems with varied stiffness. In Figure 5.17, 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. The springs have stiffness $k_n = k_\tau = 10^8$. There is gravity in the scene with acceleration $-9.8m/s^2$ in the $y-$direction. As we show in Figure 5.17, 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 5.2. The residual plot is shown on the right of Figure 5.17.

5.9.4 Quasistatic PBD/XPBD and External Forcing

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 5.18, 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.

5.9.5 Multiresolution Test: Cloth Stretching

We demonstrate how our multi-layer (MPBNG) approach resolves excessive stretching with PBNG when the iteration count is low. A rectangular shape cloth with 8K vertices is suspended from two corners under gravity. The cloth has density $R^0 = 10kg/m^2$, Young's modulus $E = 1000Pa$, $k_b = 0$. We run both PBNG and MPBNG with a budget of 1.8s/frame of computation each. We take $dt = 0.005s$ and use 30 frames per second. PBNG is run with 44 iterations per timestep and MPBNG is run with 13 iterations per timestep with $r_{pad} = 0.05$. As shown in Figure 5.19, PBNG appears to be much stretchier than the converged simulation, while MPBNG is less stretched and almost indistinguishable from the well-converged ideal solution (despite MPBNG having the same computational run time as PBNG).

5.9.6 Multiresolution Test: Clothing on Mannequin

We demonstrate the ability of MPBNG to reduce excessive stretching with a dress example. The dress has 24K vertices, density $R^0 = 10kg/m^2$, Young's modulus $E = 1000Pa$ and $k_b = 0.01$. The simulation is run with $\Delta t = .012s$ and a fixed budget of 840ms/frame. PBNG has 84 iterations per frame with this budget and MPBNG has 24 iterations. We create a mesh hierachy of four layers, where the layers have .5K, 1.5K, 6K and 24K particles respectively (see Figure 5.5). As shown in Figure 5.20, PBNG is stretchier than MPBNG given the same computational budget. We further demonstrate MPBNG's capabilities when the mannequin runs (see Figure 5.1). We take $dt = 0.003s$ and use 20 iterations per time step with $r_{\text{pad}} = 2$.

5.9.7 XPBD

We run a simple dynamics example to show that XPBD 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 5.3. The block has 4.1K particles and 17K elements. This simple but representative example demonstrates superior convergence behavior of PBNG over XPBD.

5.10 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. We show that it is capable of reducing the nonlinear system residuals in practice, in contrast to PBD/XPBD. Furthermore, we show that our node-based Gauss-Seidel approach resolves fundamental issues with PBD/XPBD for quasistatic problems, particularly for applications that require efficient and reliable creation of training data. However, our approach is mostly tailored to isotropic hyperelasticity. In future work, generalizing to the case of transverse and general anisotropy is of interest.

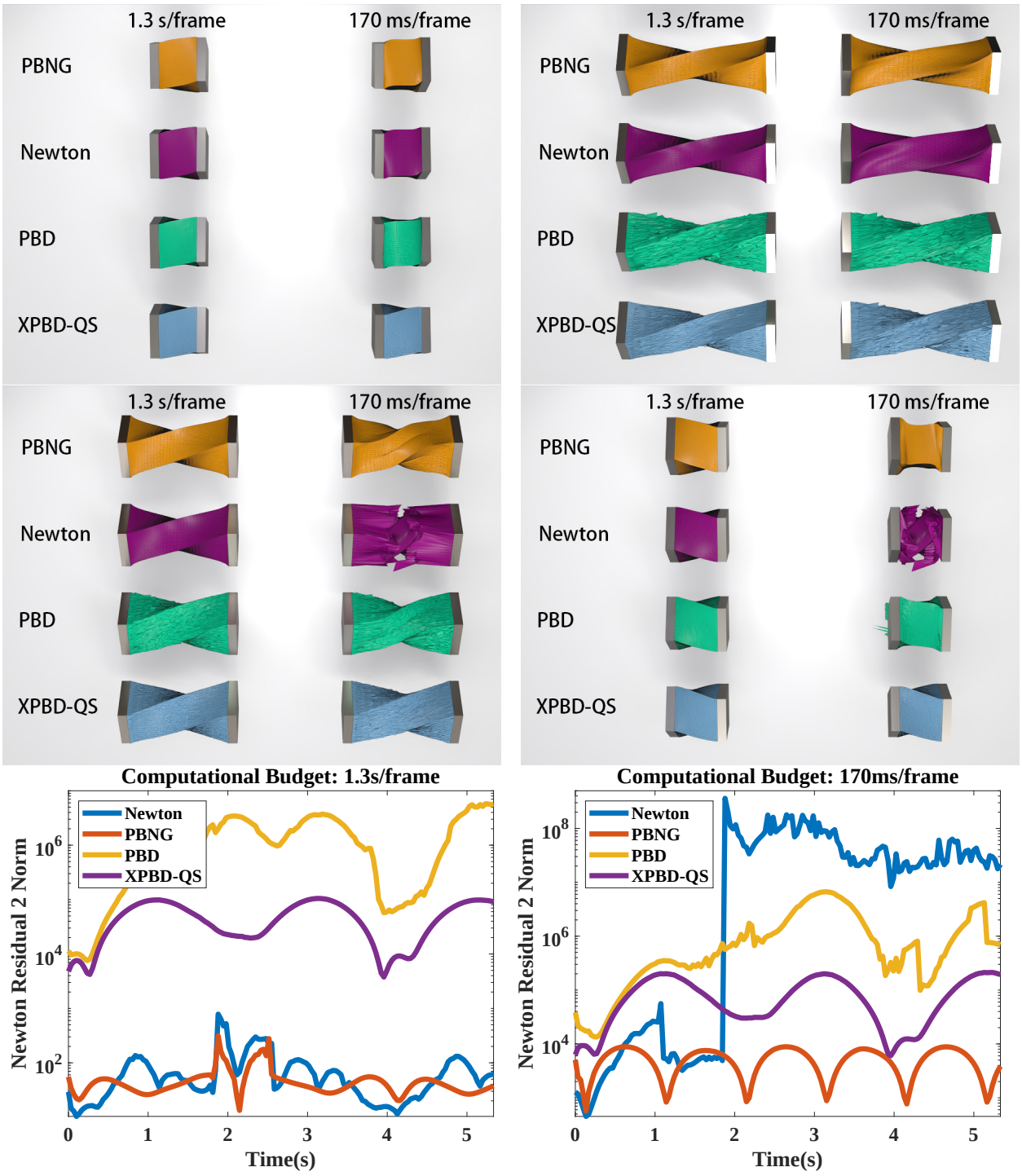


Figure 5.7: **Comparisons with Different Computational Budget**. A block is stretched/compressed while being twisted. With a sufficiently large computational budget, Newton's method is stable, but it becomes unstable when the computational budget is small. PBD and XPBD-QS do not significantly reduce 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.

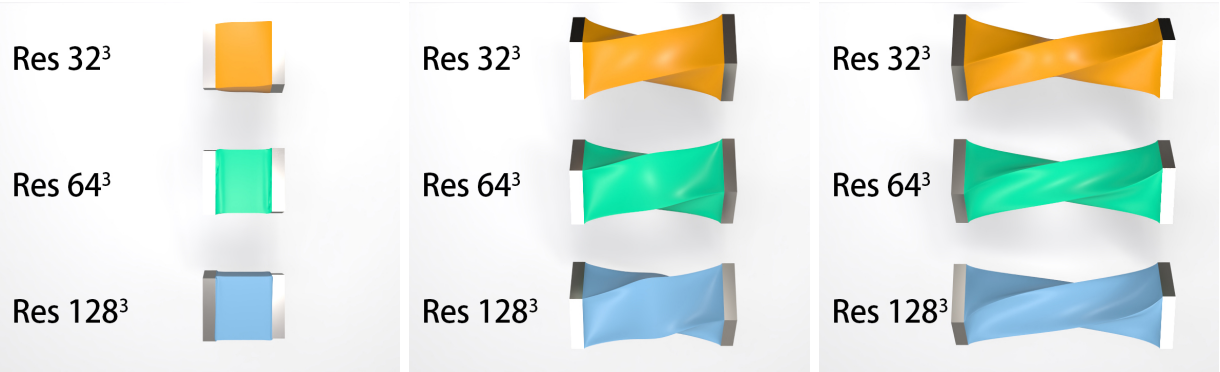


Figure 5.8: **Different Mesh Resolution**. PBNG produces consistent results when the mesh is spatially refined. The highest resolution mesh in this comparison has over 2M vertices and only requires 40 iterations to produce visually plausible results.

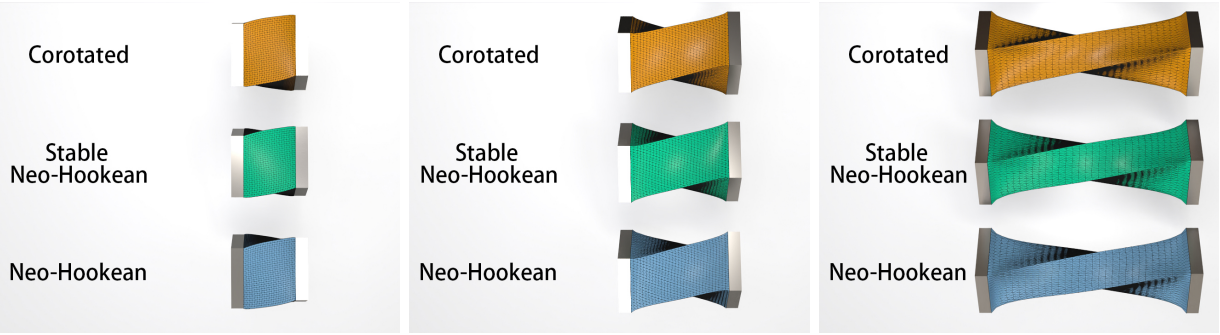


Figure 5.9: **Different Constitutive Models**. PBNG works with various constitutive models. We showcase the corotated, Neo-Hookean, and stable Neo-Hoookean models through a block twisting and stretching example.

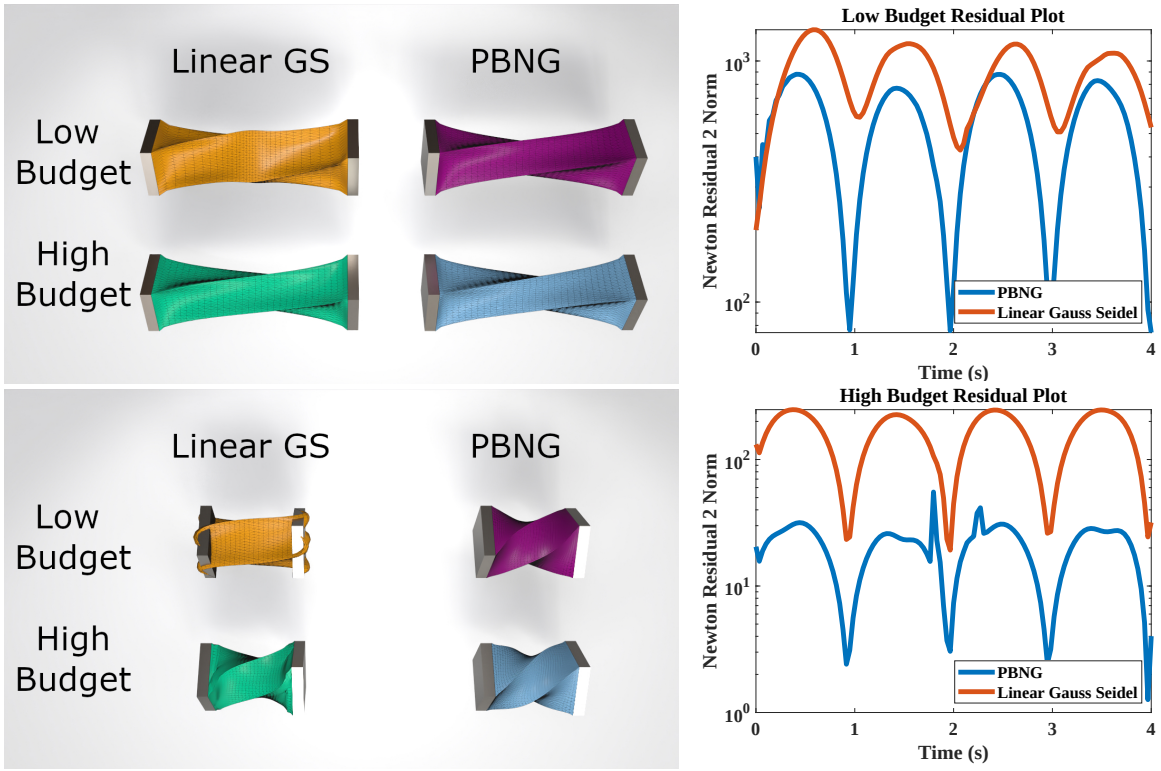


Figure 5.10: **Linear Gauss-Seidel vs. PBNG**. PBNG achieves superior residual reduction and visual quality compared to Newton's method with linear Gauss-Seidel.

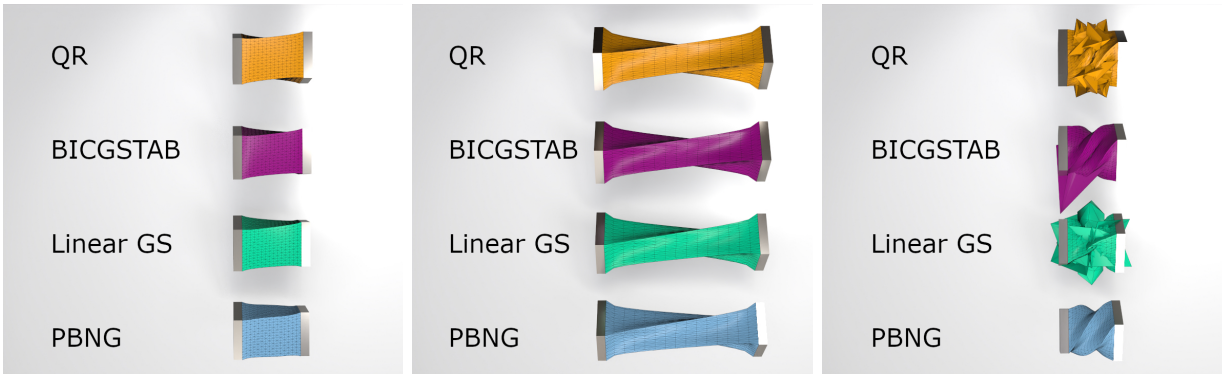


Figure 5.11: **Hessian Comparison**. The top three bars are simulated using Newton's method with different linear solvers (QR, BICGSTAB and linear Gauss-Seidel respectively). The bottom bar is simulated using PBNG. The top bar uses the exact Hessian and becomes unstable. The bottom bar uses our Hessian projection and stays stable.

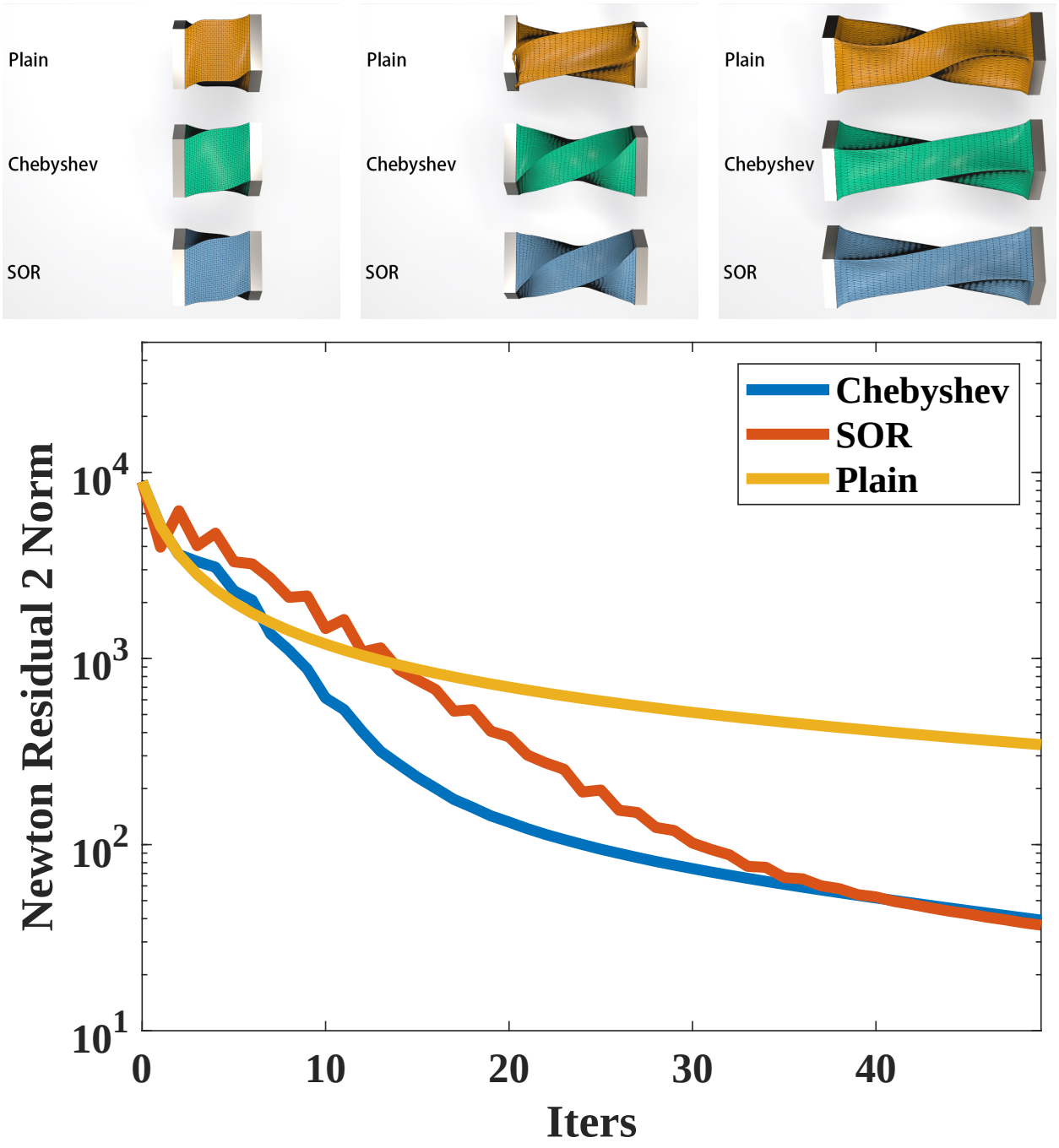


Figure 5.12: **Acceleration Techniques**. The convergence rate of PBNG may slow down as the iteration count increases. Chebyshev semi-iterative method and SOR effectively accelerate the Newton residual reduction.

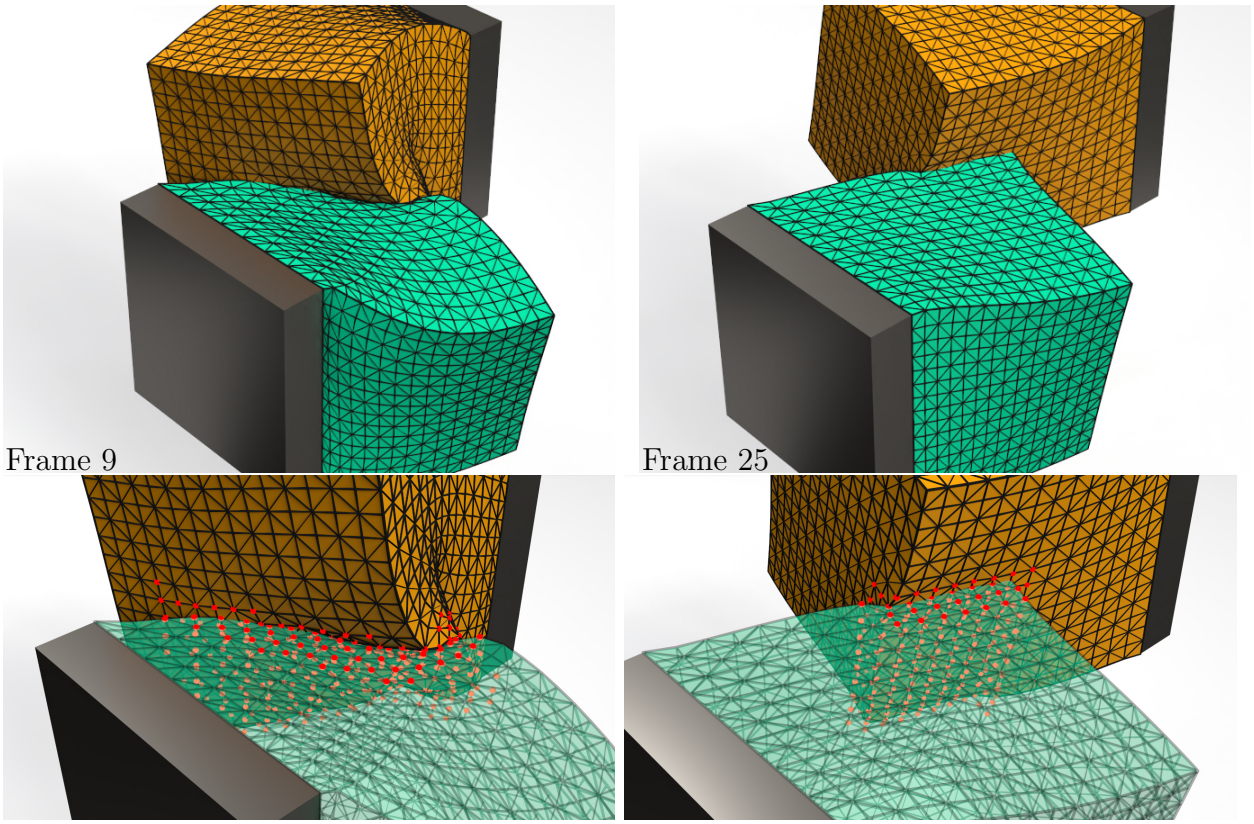


Figure 5.13: **Two Blocks Colliding**. Two blocks collide with each other with one face clamped. Red particles indicate that dynamic weak constraints have been built to resolve the collision of corresponding mesh vertices.

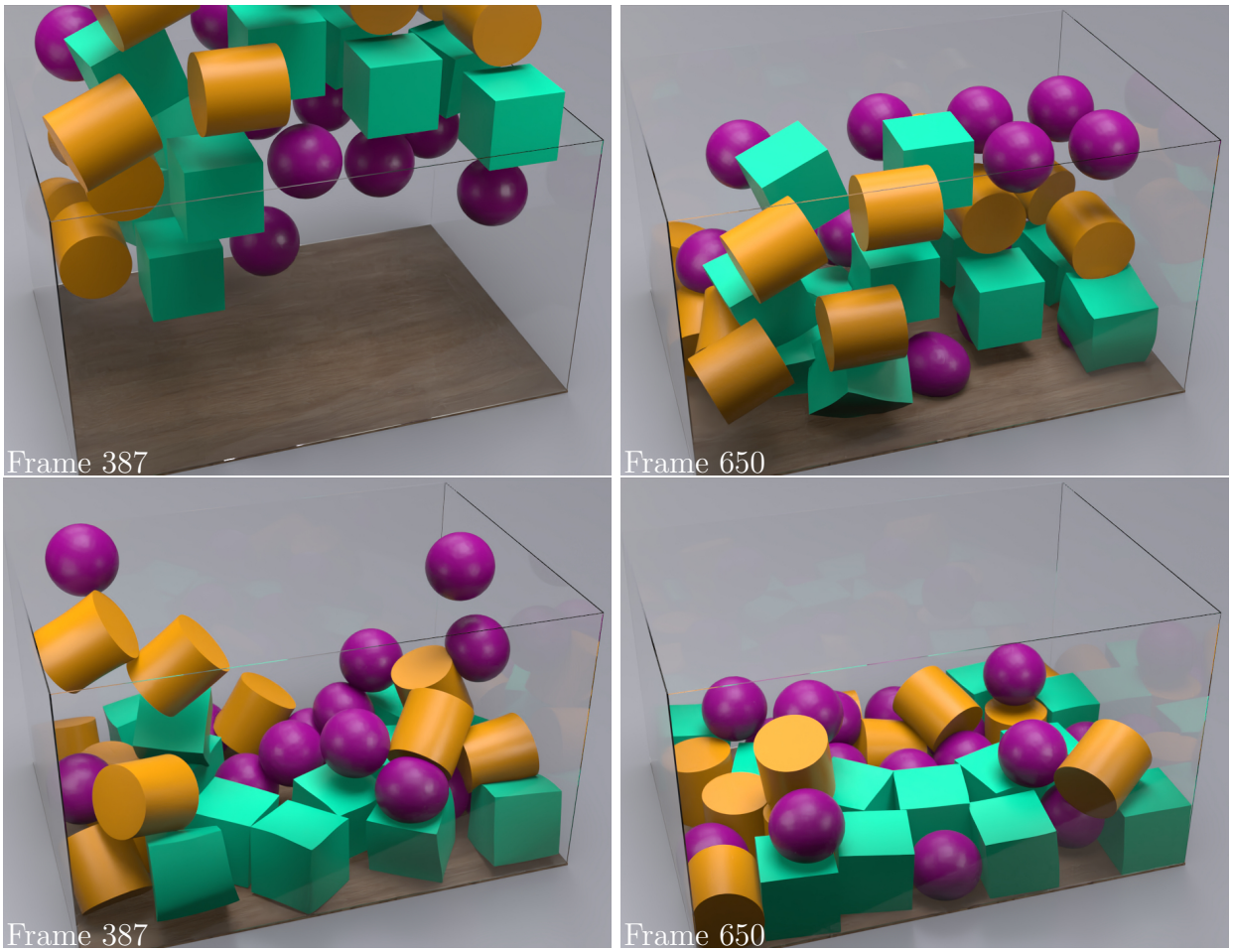


Figure 5.14: **Objects Dropping**. A variety of objects drop under gravity. Our method is able to robustly handle collisions between deformable objects through weak constraints.

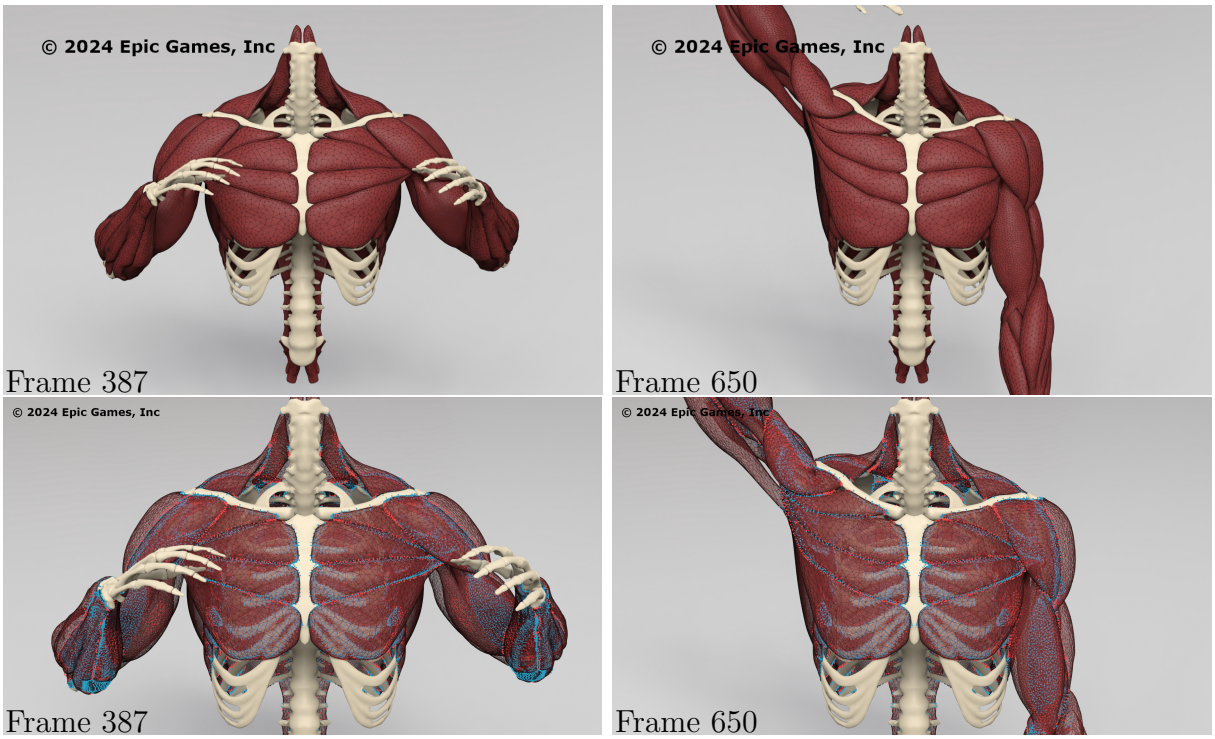


Figure 5.15: **PBNG Muscle Simulation**. The top row shows simulation results while the bottom row visualizes the vertex constraint status. *Red* indicates a vertex involved in contact, weak constraints are dynamically built to resolve the collisions. *Blue* represents the vertex positions of connective tissue bindings.

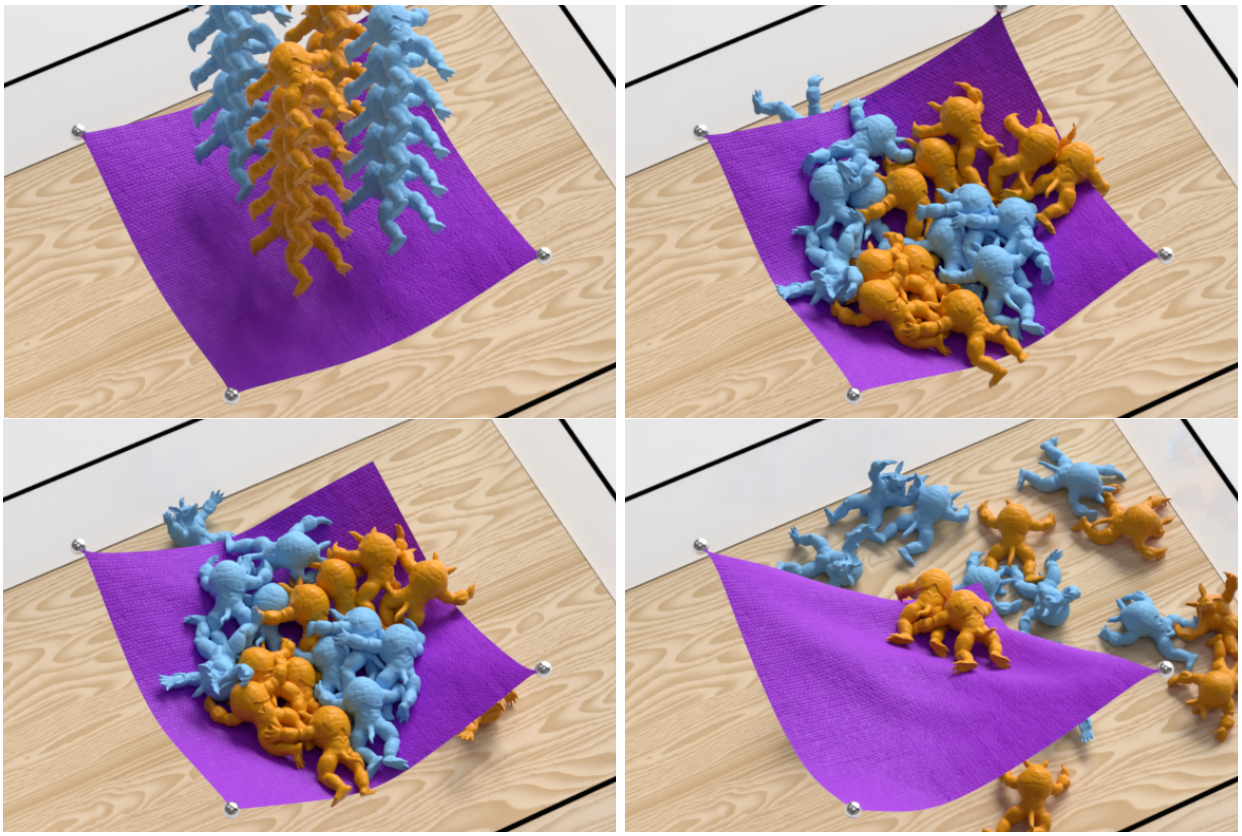


Figure 5.16: **Armadillos Dropping**. We demonstrate that PBNG can handle a large-scale simulation involving many collision-driven deformations and with clothing and solids coupled together.

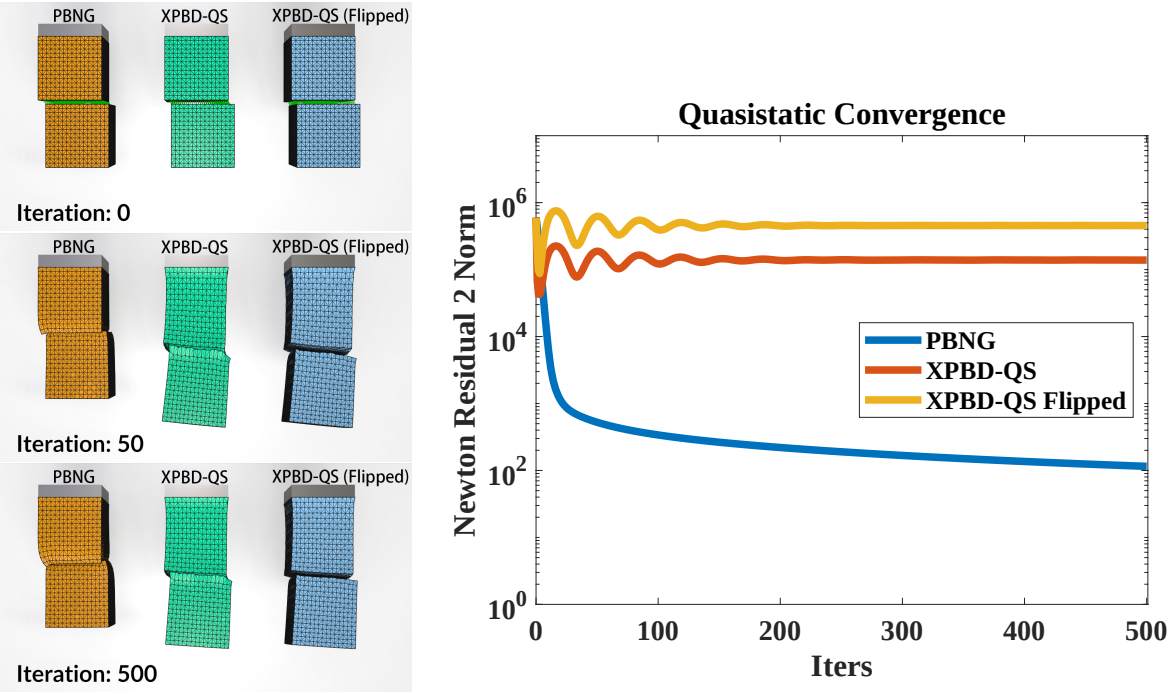


Figure 5.17: **Two Blocks Hanging**. Two identical blocks are bound together through weak constraints. Green line segments in iteration 0 indicate weak constraint springs. PBNG is able to reduce the residual by a few orders of magnitude and converges quickly. XPBD-QS methods demonstrate iteration-order-dependent behavior. Residuals oscillate and produce visually incorrect results.

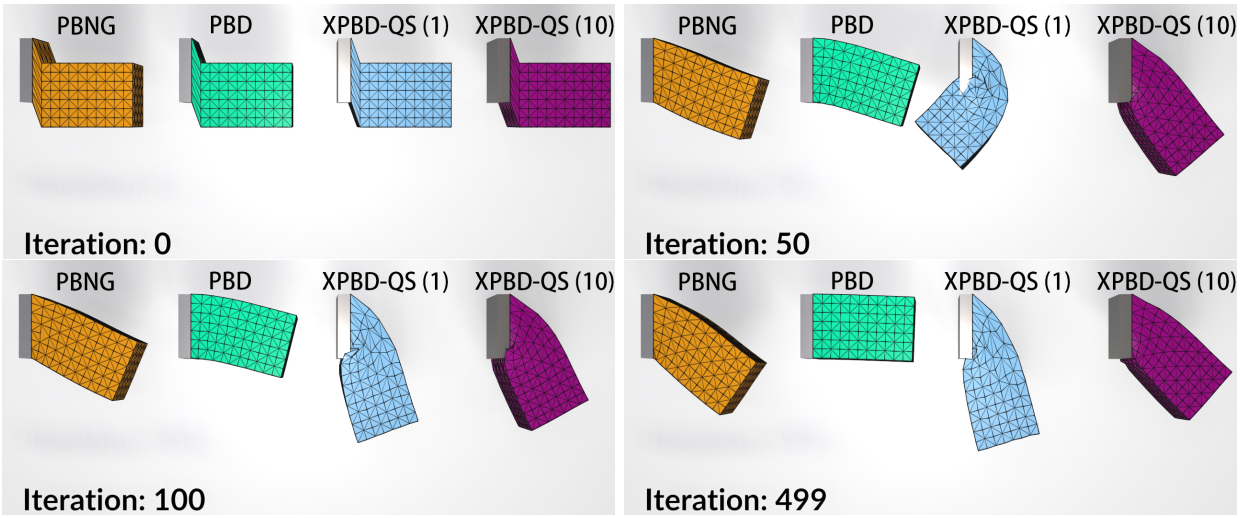


Figure 5.18: **Bar under Gravity**. A quasistatic simulation of a bar bending under gravity using different methods. The effect of external forcing vanishes in the PBD example as the number of iterations increases. More local iterations of XPBD-QS produces better results. PBNG converges to visually plausible results within fewer iterations than XPBD-QS.

Figure 5.19: **Draping Cloth**. A piece of rectangular cloth with two corners clamped swings under gravity. With a fixed computational time of 1.8s/frame, MPBNG is much more similar to the converged look of the cloth than PBNG which appears too stretchy.

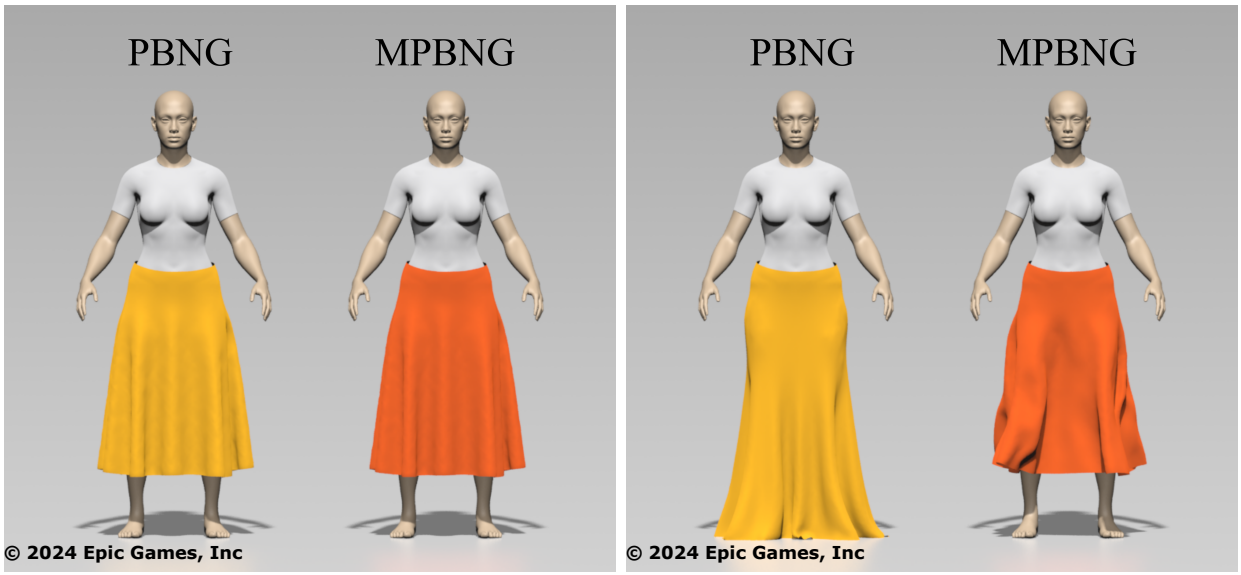


Figure 5.20: **Draping Dress**. With a fixed computational time of 840ms/frame, MPBNG cloth appears much less stretchy than PBNG alone.

CHAPTER 6

A Neural Network Model for Efficient Musculoskeletal-Driven Skin Deformation

6.1 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 6.2 (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 $\mathbf{LBS}(\mathbf{Y};\boldsymbol{\Theta}) = \sum_{i=0}^{N_B-1} w_i^{LBS}(\mathbf{Y})\,(\mathbf{R}_i(\boldsymbol{\Theta})\mathbf{Y} + \mathbf{t}_i(\boldsymbol{\Theta}))$ 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 $\boldsymbol{\phi}^m(\mathbf{X}^m;\boldsymbol{\Theta},\mathbf{a})$ similarly in terms of

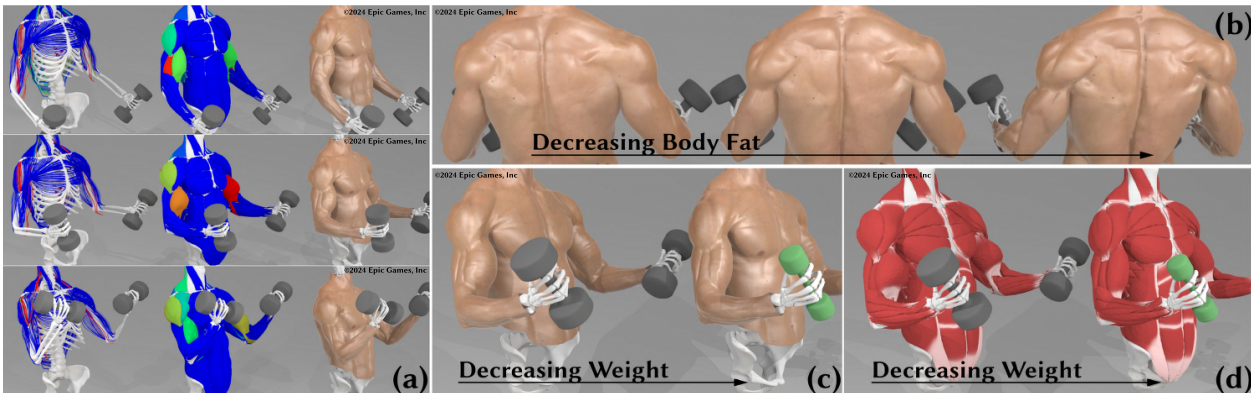


Figure 6.1: **Overview**. **(a)** Muscle activations are estimated (left and right) from a biceps curl motion using fiber streamlines embedded in muscles and then drive deformation in an ANN to improve realism. **(b)** Our approach naturally allows for variation in body fat percentage. **(c)** The effect of increased weight in the biceps curl is naturally added with our approach. **(d)** Note the difference in muscle deformation due to added weight.

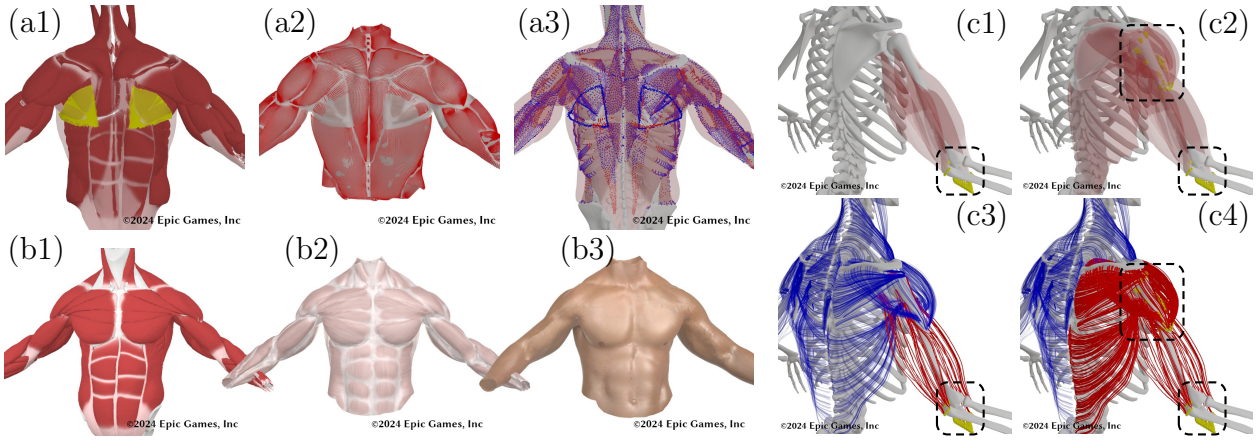


Figure 6.2: **Simulation Setup**. **(a1)**: Connective tissue membrane (in yellow). **(a2)**: Fascia with constrained vertices in red and simulated membrane in grey. **(a3)**: Muscle weak constraint visualization: fascia constraints (in blue) and contact constraints (in red). **(b1)-(b3)**: Reference 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).

passive and active neural networks $\mathbf{PNN}^m$ and $\mathbf{ANN}^m$ respectively. $\mathbf{X}^m$ is the location of the musculotendon vertex in the A-pose (see Figure 6.2 (b1)).

We provide the details for the creation of passive simulation data of the muscles, tendons, fascia and fat in Section 6.2 as well as the training process and neural network architecture in Section 6.3. 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 6.2 (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 $\boldsymbol{\Theta}$. In this case we write $\mathbf{a} = \mathbf{a}(\boldsymbol{\Theta}, \mathbf{f}^{\text{ext}})$ and the skin/fat kinematics take on the form

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

We detail the activation solution process in Section 6.4.

6.2 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 $\hat{\mathbf{x}}^m(\boldsymbol{\Theta}, \mathbf{a}) \in \mathbb{R}^{3N_{M^v}}$ 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 $\boldsymbol{\Theta}$. 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 $\hat{\mathbf{x}}^f(\hat{\mathbf{x}}^m(\boldsymbol{\Theta}, \mathbf{a}))$ 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 $\hat{\mathbf{x}}^s(\hat{\mathbf{x}}^f(\hat{\mathbf{x}}^m(\boldsymbol{\Theta}, \mathbf{a})))$. We train the muscle ($\mathbf{PNN}^m$) and fat/skin ($\mathbf{PNN}^s$) passive neural networks on $\boldsymbol{\Theta}$ using equilibrium states $\hat{\mathbf{x}}^m(\boldsymbol{\Theta}, \mathbf{0})$ and $\hat{\mathbf{x}}^s(\hat{\mathbf{x}}^f(\hat{\mathbf{x}}^m(\boldsymbol{\Theta}, \mathbf{0})))$ 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.

6.2.1 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

$$\begin{aligned} \mathrm{PE}^m(\mathbf{x}^m, \mathbf{x}^c; \boldsymbol{\Theta}, \mathbf{a}) = &\sum_t \Psi(\mathbf{F}_t(\mathbf{x}^m; \mathbf{a}, \mathbf{D}_t))V_t + \sum_{\hat{t}} \hat{\mu}|\mathbf{F}^{\mathrm{cod}}_{\hat{t}}(\mathbf{x}^m)|^2_F A_{\hat{t}} \\ &+ \sum_j \frac{1}{2}\mathbf{r}_j(\mathbf{x}^m, \mathbf{x}^c)^T \mathbf{K}_j(\mathbf{x}^m, \mathbf{x}^c)\mathbf{r}_j(\mathbf{x}^m, \mathbf{x}^c) \end{aligned} \tag{6.1}$$

subject to Dirichlet constraints

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

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 $\mathbf{R}_{i_j}(\boldsymbol{\Theta})\mathbf{X}^m_i + \mathbf{t}_{i_j}(\boldsymbol{\Theta})$ is the transformation of the constrained vertices $\mathbf{x}^m_i(\boldsymbol{\Theta})$ under bone transform i_j based on joint state $\boldsymbol{\Theta}$. Ψ is the hyperelastic potential energy density (see Section 6.2.1.1). We also couple in extra connective tissue with vertices we denote as $\mathbf{x}^c \in \mathbb{R}^{3N_c}$ (see Section 6.2.1.3). 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 6.2.1.2). Note that the minimization of Equation (6.1) 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, V_t is the volume of the tetrahedron and $\mathbf{F}_t \in \mathbb{R}^{3\times3}$ is the deformation gradient in the

tetrahedron. $\hat{t}$ refers to triangles in connective tissue meshes, $A_{\hat{t}}$ is the area of the triangle and $\mathbf{F}_{\hat{t}}^{\text{cod}} \in \mathbb{R}^{3\times 2}$ 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.

6.2.1.1 Musculotendon Fiber Fields

Although muscles are anisotropic and deform under activation state $\mathbf{a}$, we use the isotropic fixed corotated potential from [SHS12]

$$\Psi(\mathbf{F}) = \mu|\mathbf{F} - \mathbf{R}(\mathbf{F})|_F^2 + \frac{\lambda}{2}(\det(\mathbf{F}) - 1)^2 \tag{6.2}$$

where $\mathbf{F} = \mathbf{R}(\mathbf{F})SS(\mathbf{F})$ is the polar decomposition of $\mathbf{F}$ in terms of rotation $\mathbf{R}(\mathbf{F})$ and symmetric $SS(\mathbf{F})$. 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 6.4). 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 6.2 (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 6.3, we show effects of different choices of α and γ on fully contracting biceps.

Tendon attaches to bones in the skeleton at origin (proximal) and insertion (distal) locations as shown in Figure 6.4. We manually select origin and insertion vertices in the

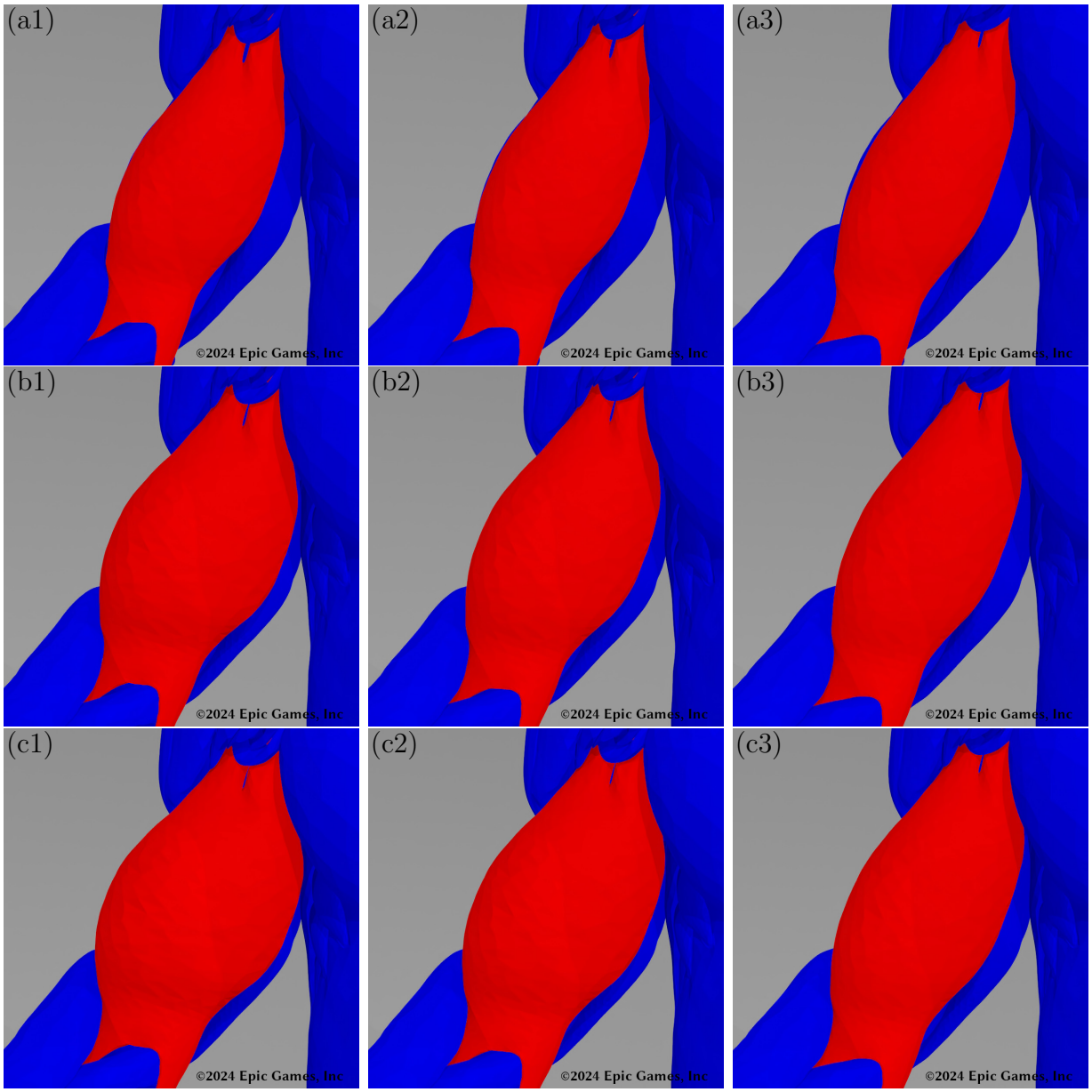


Figure 6.3: **Fiber Compression and Volume Preservation Parameters**. **Row (a-c)**: Volume preservation parameter $\alpha = 0.5, 1, 1.2$. **Column (1-3)**: Fiber compression parameter $\gamma = 0.3, 0.4, 0.6$.

musculotendon meshes as in [TSB05]. These are used to define fixed vertex boundary conditions when minimizing Equation (6.1). However, we also use origin/insertion points to define fiber directions in each tetrahedron in each musculotendon mesh as in [IHB15]. We

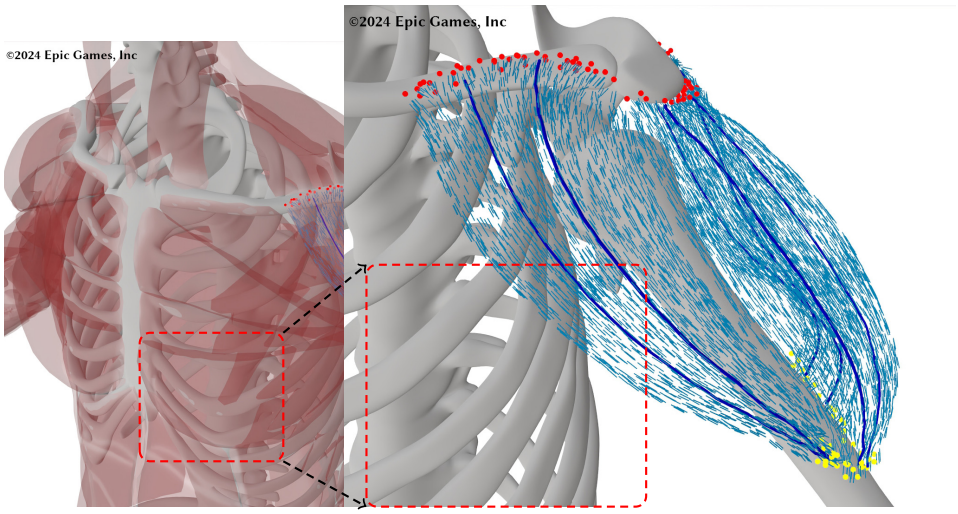


Figure 6.4: **Muscle Fibers and Streamlines**. Muscle fiber directions $\mathbf{D}_t$ are shown in blue. Origin points are shown in red and insertion points are shown in yellow. A few representative streamlines are shown as solid blue curves.

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 6.2 (c3-c4)). We illustrate this process in Figure 6.4 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).

6.2.1.2 Contact Model

We adopt the model of McAdams et al. [MZS11] for the weak constraints used in Equation (6.1) to model the effects of contact between muscles and connective tissue like fascia. Fascia 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

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

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 6.2 (a3).

As in McAdams et al. [MZS11] we model the weak interactions in an anisotropic manner. This is done through the stiffness term $\mathbf{K}_j$ in Equation (6.1). The anistropy is defined in terms of contact (k_n) and sliding/friction (k_{τ_0}, k_{τ_1}) stiffnesses respectively as $\mathbf{K}_j = k_n\mathbf{n}_j\mathbf{n}_j^T + k_{\tau_0}\boldsymbol{\tau}_{j_0}\boldsymbol{\tau}_{j_0}^T + k_{\tau_1}\boldsymbol{\tau}_{j_1}\boldsymbol{\tau}_{j_1}^T$ where $\mathbf{n}_j$ is the triangle normal/contact direction and $\mathbf{n}_j^T\boldsymbol{\tau}_{j_i} = 0$, $i = 0, 1$. For isotropic constraints ($k_c = k_n = k_{\tau_0} = k_{\tau_1}$), we use the scalar k_c in place of $\mathbf{K}_j$ since it is diagonal in this case. 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 6.5.3.

6.2.1.3 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 (6.1). The membrane is put under tension using the quadratic potential $\sum_{\hat{t}} \hat{\mu} |\mathbf{F}^{\text{cod}}_{\hat{t}}(\mathbf{x}^c)|^2_F A_{\hat{t}}$. 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 triangles 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 $\mathbf{x}^c_{j_0}$. We illustrate this connective tissue in yellow in Figure 6.2 (a1).

6.2.2 Fascia Layer

Given the equilibrium configuration of the musculotendon geometry $\hat{\mathbf{x}}^m(\boldsymbol{\Theta})$ for a given joint state $\boldsymbol{\Theta}$, we solve for the fascia layer $\hat{\mathbf{x}}^f(\hat{\mathbf{x}}^m(\boldsymbol{\Theta}))$. 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 6.2 (a2)); we simulate the remaining 10% by putting the membrane under tension with the same model as the connective tissue membrane in Section 6.2.1.3. We also collide these against the boundary of the musculotendon meshes. This serves as the inner layer of the fat/skin mesh.

6.2.3 Fat and Skin Layer

In the last layer of our approach, we solve for the equilibrium configuration $\hat{\mathbf{x}}^s(\hat{\mathbf{x}}^f(\hat{\mathbf{x}}^m(\boldsymbol{\Theta})))$ 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

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

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 is the again the isotropic potential from Equation (6.2). The inner fascia vertices are fixed based on the joint state through $\mathbf{x}^f(\mathbf{x}^m(\boldsymbol{\Theta}))$. 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 6.8.

6.3 Neural Network

Our passive ($\mathbf{PNN}(\boldsymbol{\Theta}; \mathbf{X}) \in \mathbb{R}^3$) 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 Jin et al. [JZG20] and utilize a PCA final layer so that the network output is simply tens of PCA coefficients. The vertex displacements then can be 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 6.5. Our PNN input consists of $N_J = 23$ joint local rotation matrices 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 6.6. 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

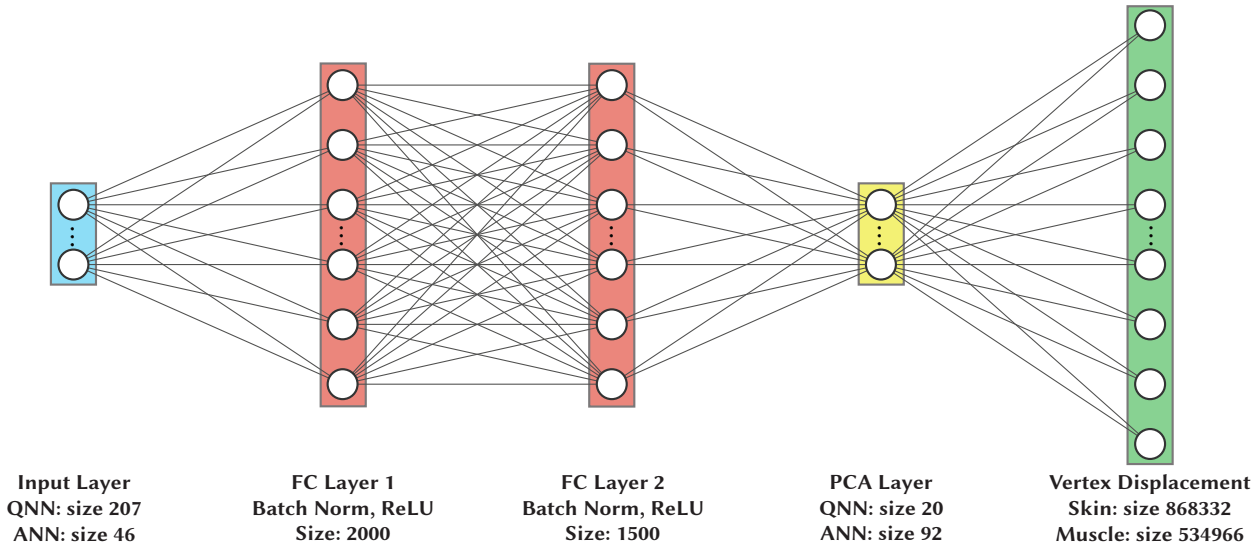


Figure 6.5: **Architecture for PNN and ANN**. We use two fully connected (FC) hidden layers with batch normalization and ReLU activation function.

1000 epochs and stochastic gradient descent. The training and evaluation loss as well as training time are provided in Table 6.1.

6.4 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 $\hat{\mathbf{a}} \in \mathbb{R}^{N_F}$ ($N_F = 2624$ in our examples). 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 6.2.1.1 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.

6.4.1 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 the supplemental material in Chapter 8 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

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

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

$$\int_\Omega \sum_{\alpha=0}^{d-1} J^b_{\alpha j}(\mathbf{x};\mathbf{q})f_\alpha(\mathbf{x})d\mathbf{x} = \int_{\hat{\Omega}_j}((\mathbf{x}-\mathbf{x}^j)\times\mathbf{f}(\mathbf{x}))\cdot\mathbf{u}^j d\mathbf{x} = 0 \tag{6.3}$$

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 the supplemental material in Chapter 8 for more detail.

6.4.2 Active Muscle Force Model

Each fiber streamline in each muscle originates on a bone and inserts on another bone (see Figure 6.2 (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

$$\mathbf{f}^j(l_j; \hat{a}_j) = \sigma_j^{\max}(f_p(l_j) + \hat{a}_j f_a(l_j))\mathbf{n}_j \tag{6.4}$$

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 Seth et al. [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].

6.4.3 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 (6.3)). The inverse activation problem can be formulated as minimizing the quadratic total energy spent subject to linear torque equilibrium equations. Specifically,

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

$$\text{subject to}$$

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

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. $\tilde{\mathbf{x}}_{j_i}^{SL} = \mathbf{x}_{j_i}^{SL} - \mathbf{x}^j$ is the local position of endpoints $\mathbf{x}_{j_i}^{SL}$ 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 (6.4), we can represent the Equation (6.3) 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 k^{th} iteration is

$$\begin{aligned}\hat{\boldsymbol{\lambda}}^k &= (\mathbf{W}(\mathbf{D}^{k-1})^{-1}\mathbf{W}^T)^{-1}(\mathbf{W}(\mathbf{D}^{k-1})^{-1}\mathbf{M}^{p,k-1}\hat{\mathbf{a}}_{\text{lb}} + \mathbf{b}) \\ \hat{\mathbf{a}}^k &= -\mathbf{D}^{k-1^{-1}}(\mathbf{M}^{p,k-1}\hat{\mathbf{a}}_{\text{lb}} - \mathbf{W}^T\hat{\boldsymbol{\lambda}}^k).\end{aligned} \tag{6.5}$$

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 (6.5) since the size is negligible.

6.5 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.

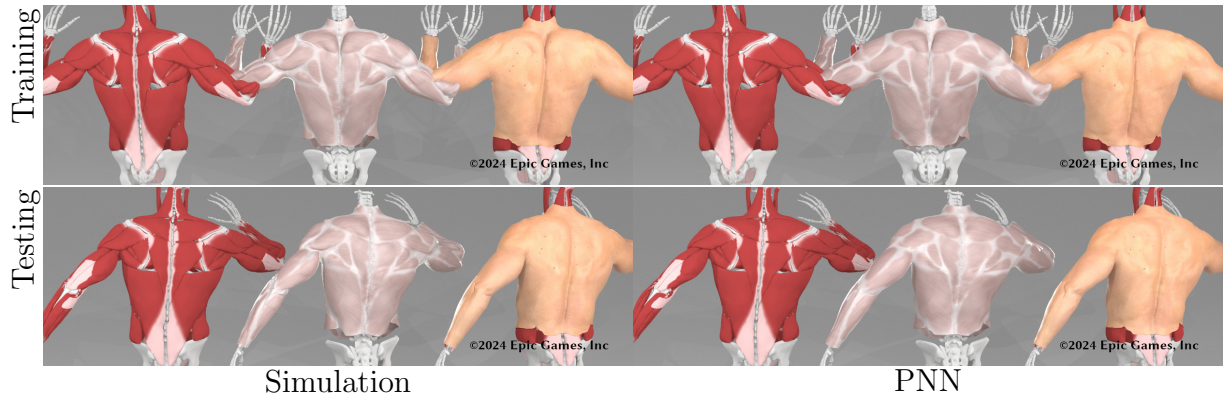


Figure 6.6: **PNN Fitting and Generalization**. **Top Row**: Muscle and fat/skin PNNs fit training data effectively. Simulation (left) is compared to the PNNs (right). **Bottom Row**: Muscle and fat/skin PNNs generalize effectively to unseen testing data. Simulation (left) is compared to PNNs (right).

6.5.1 Network Deformation Demonstrations

We first show the per-vertex mean squared error (MSE) loss of the PNN and ANN on the training data and evaluation data in Table 6.1. 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 6.6. 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 biceps curl animation in Figure 6.7. Note that this motion is not in the training data set. Our model shows more realistic muscle contraction in the biceps, 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 6.8. 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 6.9. As expected, increased

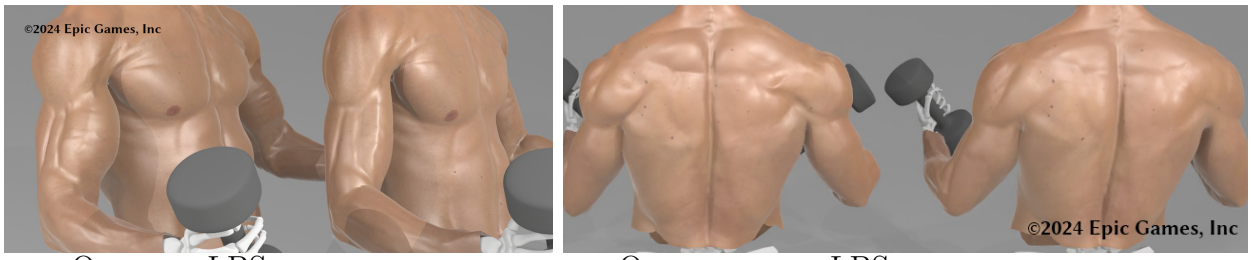


Figure 6.7: **Active neural network deformation**. In each image, the left body illustrates the benefit of our ANN and PNN enhancement of LBS by comparing it to standard LBS in the right body counterpart.

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 6.10. We note that the activations are asymmetric due to 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.

6.5.2 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] and compared their estimated activations to the EMG data to validate their model. For the shoulder abduction and flexion tasks shown in Figure 6.11, our model captures many of the characteristics of the observed EMG patterns, such as the ramp in superior trapezius

Table 6.1: **Training and Evaluation Loss**. We show that the trained PNN and ANN are able to generalize to the evaluation data. The networks were trained using an 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

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).

6.5.3 Simulation Parameters and Runtime

We use relatively few physical parameters in our simulations. We set our Lamé parameters μ, $\hat{\mu}$ 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

Table 6.2: **Runtime Comparison**. We compare the runtime of our approach against simulation. Times are averaged over the testing EMG animation. Examples were run using an AMD Ryzen PRO 3995WX CPU (128 threads).

Task	# Vertices	Runtime (ms/frame)
$\mathbf{PNN}^m + \mathbf{ANN}^m$	182K	36
$\mathbf{PNN}^s + \mathbf{ANN}^s$	289K	52
Inverse Activation	2624 curves	180
Simulation, Muscle+Fascia+Skin	182K+145K+145K	283K+25K+39K

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 6.2. Our method is more than one thousand times faster than the simulation and can be run in real time if slightly fewer curves and mesh vertices are used.

6.6 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.

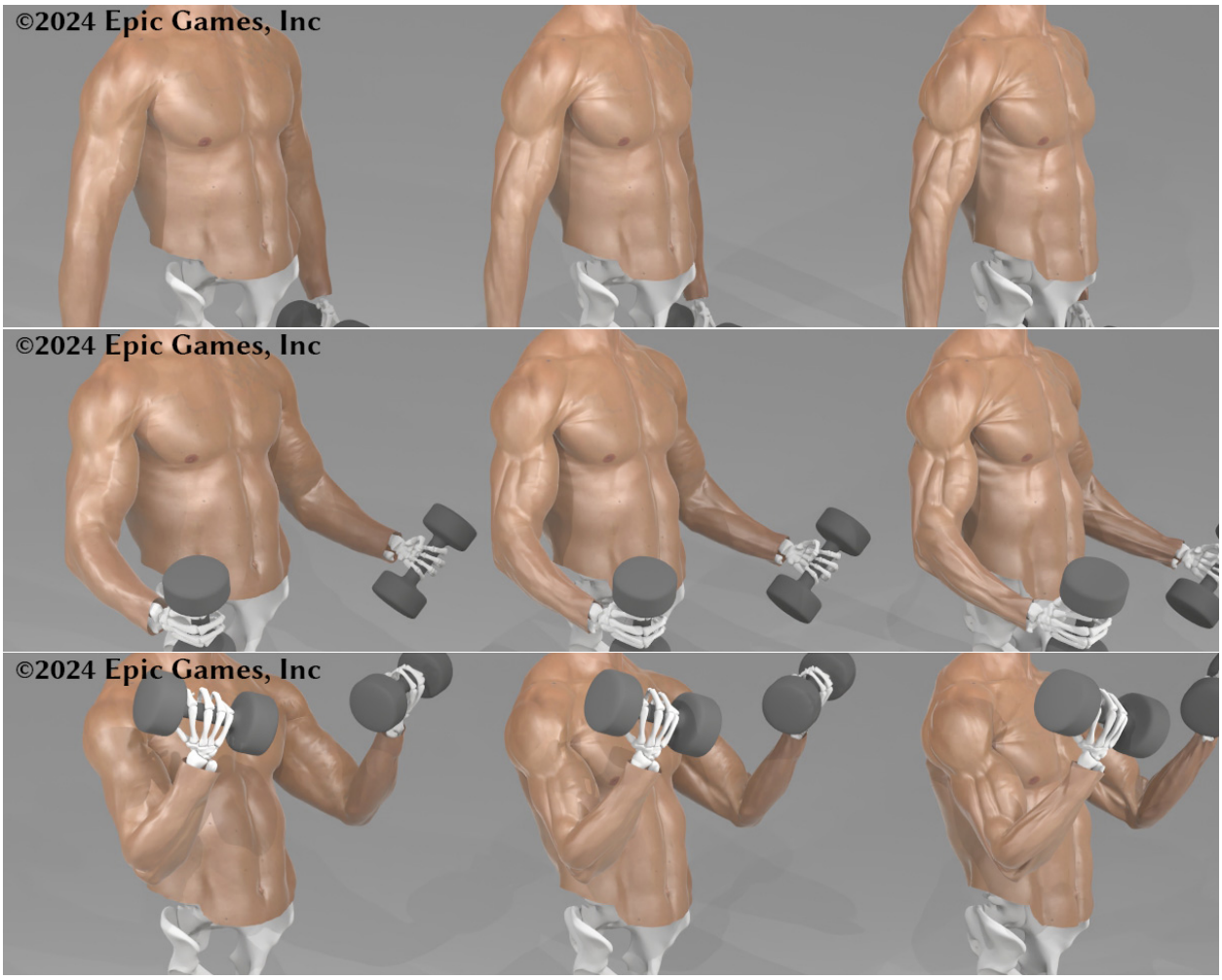


Figure 6.8: **Variation in Body Fat Percentage**. Left: unmodified outer skin surface, middle: halfway between skin and fascia, right: 0% body fat/fascia.

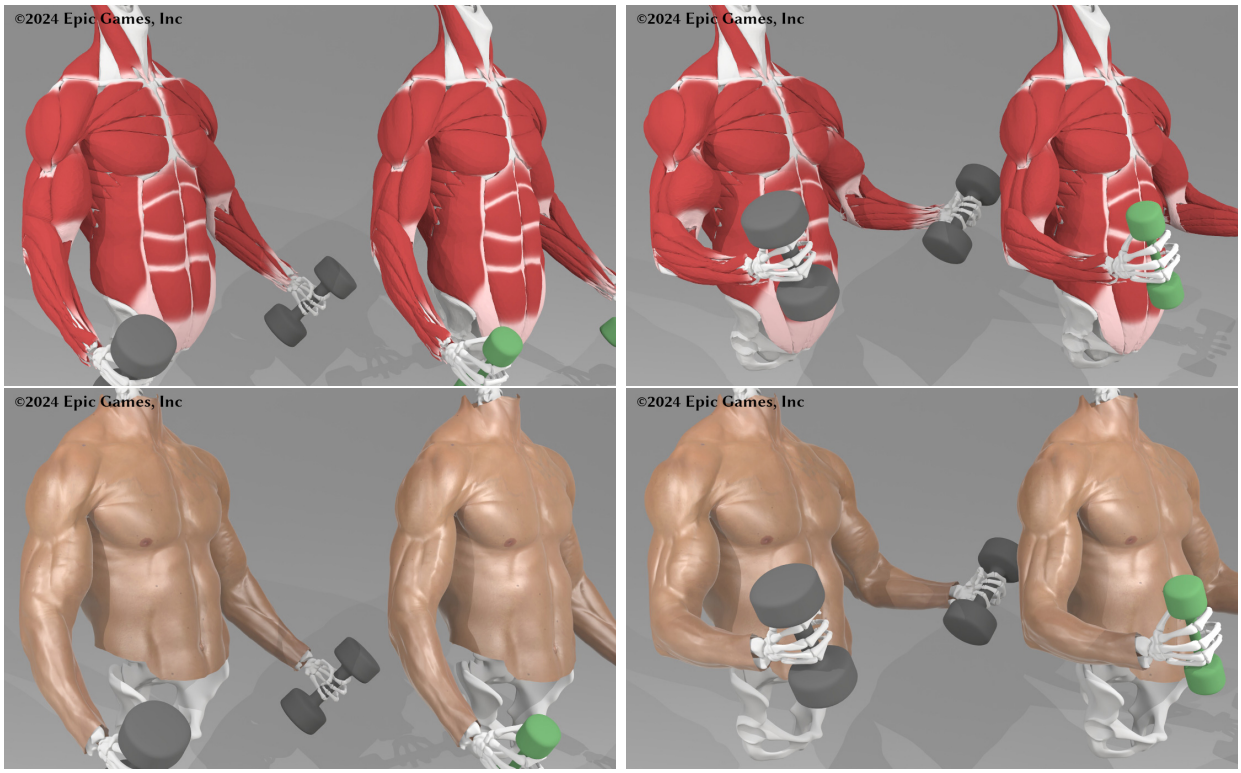


Figure 6.9: **Effect of Increased Weights on Muscles and Skin**. Heavy dumbbell (gray) v.s. light dumbbell (green).

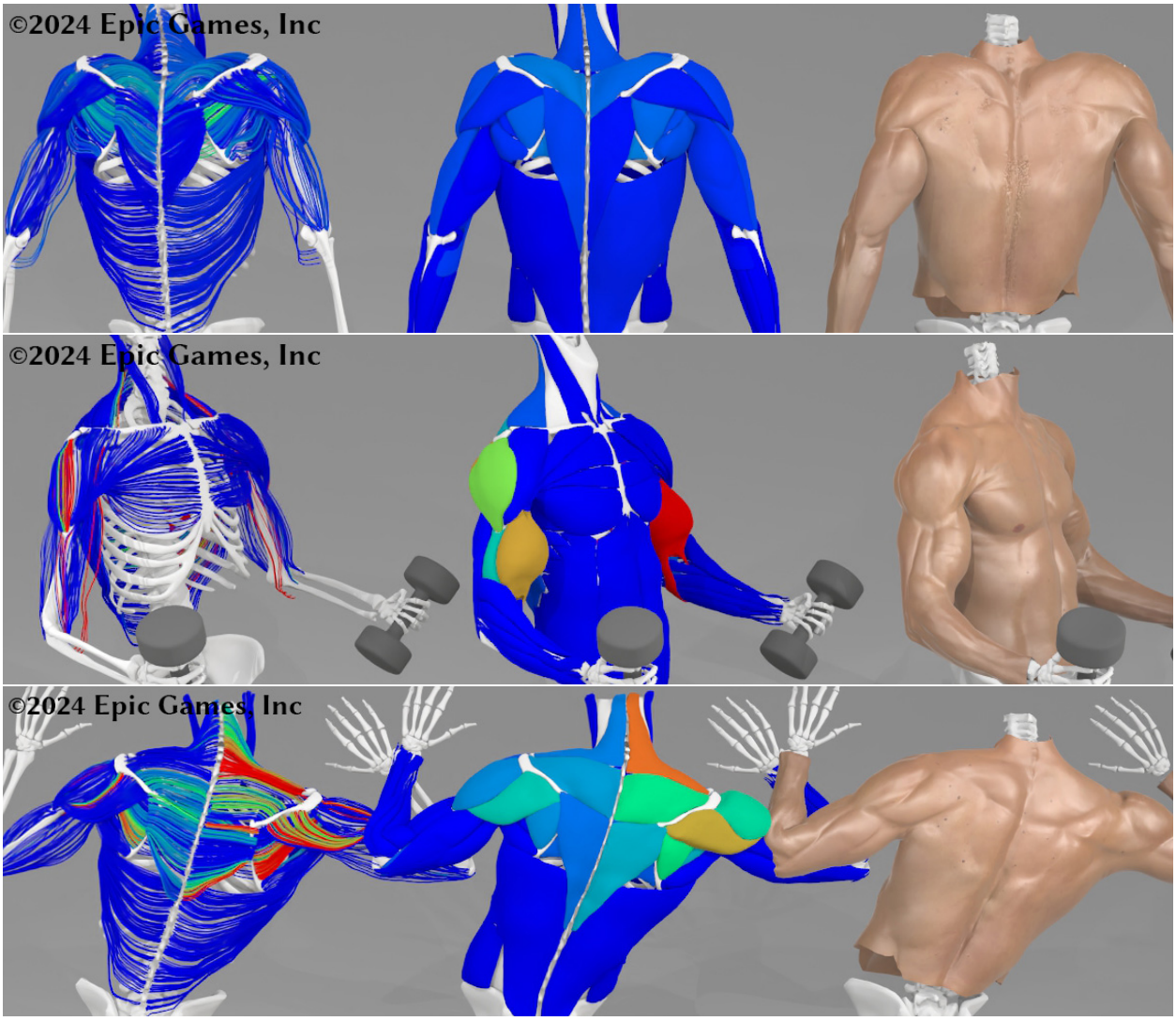


Figure 6.10: **Streamline Activations on Various Poses**. **Left to right**: streamline activations, muscle activations with active network muscle contraction and skin with active network deformation. **Top**: shoulder shrug at time $t = 0.4$s. **Middle**: biceps curl at time $t = 0.53$s. **Bottom**: motion capture at time $t = 48.1$s.

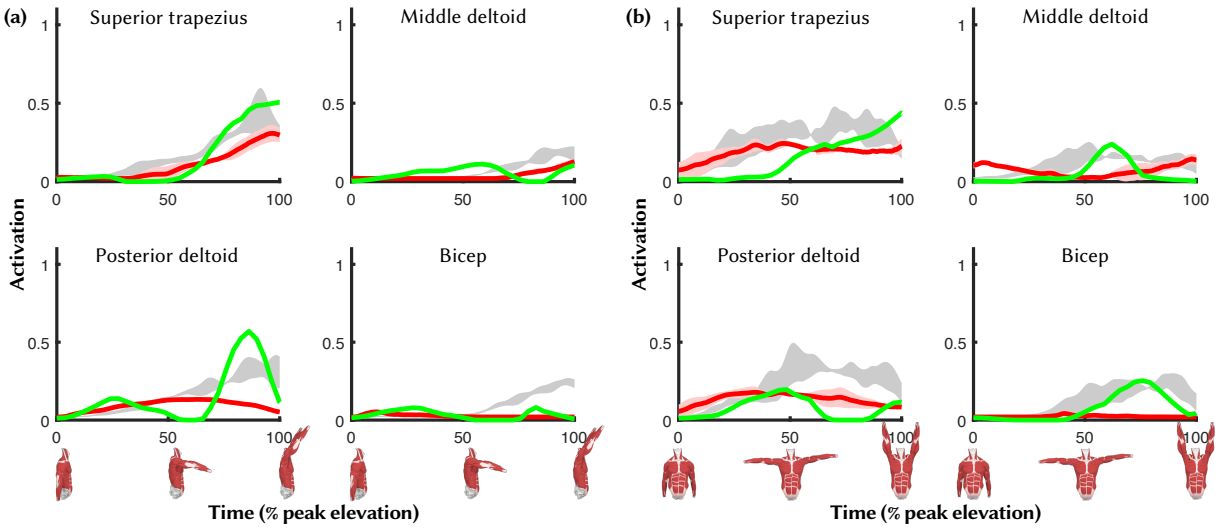


Figure 6.11: **Activation Comparison**. We compare our computed muscle activations with the state-of-the-art approach in Seth et al. [SHU18]. Ground-truth, experimentally observed EMG Data is provided. Our comparisons to EMG data are similar and in some cases improved over Seth et al. [SHU18]. **Green**: Ours. **Red**: Seth et al. [SHU18]. **Gray**: EMG data. **(a)**: Shoulder flexion, **(b)**: Shoulder abduction.

CHAPTER 7

Supplementary Material for Position-Based Nonlinear Gauss Seidel

7.1 Linear Elasticity

7.1.1 Potential

$$\Psi^{\text{le}}(\mathbf{F}) = \mu\boldsymbol{\epsilon}(\mathbf{F}) : \boldsymbol{\epsilon}(\mathbf{F}) + \frac{\lambda}{2}\text{tr}(\boldsymbol{\epsilon}(\mathbf{F}))^2 \tag{7.1}$$

$$\boldsymbol{\epsilon}(\mathbf{F}) = \frac{1}{2}\left(\mathbf{F} + \mathbf{F}^T\right) - \mathbf{I} \tag{7.2}$$

7.1.2 First-Piola-Kirchhoff Stress

$$\mathbf{P}^{\text{le}}(\mathbf{F}) = \frac{\partial \Psi^{\text{le}}}{\partial \mathbf{F}}(\mathbf{F}) = 2\mu\boldsymbol{\epsilon}(\mathbf{F}) + \lambda\text{tr}(\boldsymbol{\epsilon}(\mathbf{F}))\mathbf{I} \tag{7.3}$$

7.1.3 Hessian

$$\frac{\partial^2 \Psi^{\text{le}}}{\partial \mathbf{F}^2}(\mathbf{F}) = 2\mu\frac{\partial \boldsymbol{\epsilon}}{\partial \mathbf{F}}(\mathbf{F}) + \lambda\mathbf{I} \otimes \mathbf{I}. \tag{7.4}$$

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

7.1.4 General Isotropic Elasticity Modified Hessian

We use the modified Hessian

$$\tilde{\mathcal{C}}(\mathbf{F}) = \mu \frac{\partial^2 I_0}{\partial \mathbf{F}^2} + \lambda \frac{\partial I_{d-1}}{\partial \mathbf{F}} \otimes \frac{\partial I_{d-1}}{\partial \mathbf{F}}. \tag{7.5}$$

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.

7.2 Neo-Hookean

7.2.1 Neo-Hookean Potential

$$\Psi(\mathbf{F}) = \frac{\mu}{2}\mathbf{F} : \mathbf{F} + \frac{\hat{\lambda}}{2}(\det(\mathbf{F}) - 1 - \frac{\mu}{\hat{\lambda}})^2 \tag{7.6}$$

7.2.2 First-Piola-Kirchhoff Stress

$$\mathbf{P}(\mathbf{F}) = \mu\mathbf{F} + \hat{\lambda}(\det(\mathbf{F}) - 1 - \frac{\mu}{\hat{\lambda}})J\mathbf{F}^{-T} \tag{7.7}$$

7.2.3 Hessian

$$\frac{\partial^2 \Psi}{\partial \mathbf{F}}(\mathbf{F}) = \mu\mathbf{I} + \hat{\lambda}J\mathbf{F}^{-T} \otimes J\mathbf{F}^{-T} + \hat{\lambda}(\det(\mathbf{F}) - 1 - \frac{\mu}{\hat{\lambda}})\frac{\partial^2 J}{\partial \mathbf{F}^2}(\mathbf{F}) \tag{7.8}$$

7.2.3.1 Determinant Hessian

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

$$J = \det(\mathbf{F}) = \tilde{\boldsymbol{\epsilon}}_{\alpha\beta\gamma} F_{0\alpha} F_{1\beta} F_{1\gamma} \tag{7.9}$$

$$\frac{\partial J}{\partial F_{\delta\boldsymbol{\epsilon}}}(\mathbf{F}) = J F^{-1}_{\boldsymbol{\epsilon}\delta} \tag{7.10}$$

$$= \tilde{\boldsymbol{\epsilon}}_{\boldsymbol{\epsilon}\beta\gamma}\delta_{0\delta} F_{1\beta} F_{2\gamma} + \tilde{\boldsymbol{\epsilon}}_{\alpha\boldsymbol{\epsilon}\gamma}\delta_{1\delta} F_{0\alpha} F_{2\gamma} + \tilde{\boldsymbol{\epsilon}}_{\alpha\beta\boldsymbol{\epsilon}}\delta_{2\delta} F_{0\alpha} F_{1\beta} \tag{7.11}$$

$$\frac{\partial^2 J}{\partial F_{\sigma\tau}\partial F_{\delta\boldsymbol{\epsilon}}}(\mathbf{F}) = \tilde{\boldsymbol{\epsilon}}_{\boldsymbol{\epsilon}\tau\gamma}\delta_{0\delta}\delta_{1\sigma} F_{2\gamma} + \tilde{\boldsymbol{\epsilon}}_{\tau\boldsymbol{\epsilon}\gamma}\delta_{0\sigma}\delta_{1\delta} F_{2\gamma} + \tilde{\boldsymbol{\epsilon}}_{\tau\beta\boldsymbol{\epsilon}}\delta_{0\sigma}\delta_{2\delta} F_{1\beta} + \tag{7.12}$$

$$\tilde{\boldsymbol{\epsilon}}_{\boldsymbol{\epsilon}\beta\tau}\delta_{0\delta}\delta_{2\sigma} F_{1\beta} + \tilde{\boldsymbol{\epsilon}}_{\alpha\boldsymbol{\epsilon}\tau}\delta_{1\delta}\delta_{2\sigma} F_{0\alpha} + \tilde{\boldsymbol{\epsilon}}_{\alpha\tau\boldsymbol{\epsilon}}\delta_{1\sigma}\delta_{2\delta} F_{0\alpha}. \tag{7.13}$$

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	

7.2.4 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 $\mu + \hat{\lambda} = 2\mu + \lambda$.

CHAPTER 8

Supplementary Material for Inverse Activation

8.1 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

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

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

$$\int_\Omega \sum_{\alpha=0}^{d-1} J^b_{\alpha j}(\mathbf{x};\mathbf{q}) f_\alpha(\mathbf{x}) d\mathbf{x} = 0 \tag{8.1}$$

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 (8.1) becomes:

$$\begin{aligned} \int_\Omega \sum_{\alpha=0}^{d-1} J^b_{\alpha j}(\mathbf{x}; \mathbf{q}) f_\alpha(\mathbf{x}) d\mathbf{x} &= \int_{\hat{\Omega}_j} -\sum_{\alpha,p,l=0}^{d-1} \epsilon_{p\alpha l} u^j_p (x_l - x^{j_i}_l) f_\alpha(\mathbf{x}) d\mathbf{x} \\ &= \int_{\hat{\Omega}_j} \sum_{\alpha,p,l=0}^{d-1} \epsilon_{pl\alpha} (x_l - x^{j_i}_l) f_\alpha(\mathbf{x}) u^j_p d\mathbf{x} \\ &= \int_{\hat{\Omega}_j} ((\mathbf{x} - \mathbf{x}^j) \times \mathbf{f}(\mathbf{x})) \cdot \mathbf{u}^j d\mathbf{x} = 0 \end{aligned} \tag{8.2}$$

For further simplification of ball-and-socket joints, we can take $\mathbf{U}_j = \mathbf{I}$ and Equation (8.2) 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.

References

[AC76] R. Atkin and R. Craine. "Continuum theories of mixtures: basic theory and historical development." *Quart J Mech App Math*, **29**(2):209–244, 1976.

[BBB07] C. Batty, F. Bertails, and R. Bridson. "A fast variational framework for accurate solid-fluid coupling." *ACM Trans Graph*, **26**(3), 2007.

[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.

[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.

[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.

[Bra77] A. Brandt. "Multi-level adaptive solutions to boundary-value problems." *Math Comp*, **31**(138):333–390, 1977.

[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.

[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.

[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.

[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.

[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.

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

[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.

[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.

[GHF07] R. Goldenthal, D. Harmon, R. Fattal, M. Bercovier, and E. Grinspun. "Efficient Simulation of Inextensible Cloth." *ACM Trans Graph*, **26**(3), July 2007.

[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.

[Hag89] W. Hager. "Updating the inverse of a matrix." *SIAM review*, **31**(2):221–239, 1989.

[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.

[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.

[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.

[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.

[JSS15] C. Jiang, C. Schroeder, A. Selle, J. Teran, and A. Stomakhin. "The Affine Particle-In-Cell Method." *ACM Trans Graph*, **34**(4):51:1–51:10, 2015.

[JST16] C. Jiang, C. Schroeder, J. Teran, A. Stomakhin, and A. Selle. "The Material Point Method for Simulating Continuum Materials." In *ACM SIGGRAPH 2016 Course*, pp. 24:1–24:52, 2016.

[JZG20] N. Jin, Y. Zhu, Z. Geng, and R. Fedkiw. "A Pixel-Based Framework for Data-Driven Clothing." In *Proc ACM SIGGRAPH/Eurographics Symp Comp Anim*, SCA '20. Eurographics Association, 2020.

[KGF10] N. Kwatra, J. Gretarsson, and R. Fedkiw. "Practical Animation of Compressible Flow for ShockWaves and Related Phenomena." In *Symp Comp Anim*, pp. 207–215, 2010.

[KSG09] N. Kwatra, J. Su, J. Grétarsson, and R. Fedkiw. "A method for avoiding the acoustic time step restriction in compressible flow." *J Comp Phys*, **228**(11):4146–4161, 2009.

[KYT06] L. Kharevych, W. Yang, Y. Tong, E. Kanso, J.E. Marsden, P. Schröder, and M. Desbrun. "Geometric, Variational Integrators for Computer Animation." In *Proc 2006 ACM SIGGRAPH/Eurograph Symp Comp Anim*, SCA '06, p. 43?51. Eurographics Association, 2006.

[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.

[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.

[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.

[LMR23] M. Loper, N. Mahmood, J. Romero, G. Pons-Moll, and M. Black. *SMPL: A Skinned Multi-Person Linear Model*. ACM, 1 edition, 2023.

[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.

[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.

[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.

[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.

[MHT05] M. Müller, B. Heidelberger, M. Teschner, and M. Gross. "Meshless deformations based on shape matching." In *ACM transactions on graphics (TOG)*, volume 24, pp. 471–478. ACM, 2005.

[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.

[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.

[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.

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

[Pro95] X. Provot. "Deformation constraints in a mass-spring model to describe rigid cloth behaviour." In *Graph Int*, pp. 147–155. Canadian Information Processing Society, 1995.

[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.

[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.

[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.

[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.

[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.

[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.

[SSJ14] A. Stomakhin, C. Schroeder, C. Jiang, L. Chai, J. Teran, and A. Selle. "Augmented MPM for phase-change and varied materials." *ACM Trans Graph*, **33**(4):138:1–138:11, 2014.

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

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

[TL94] G. Turk and M. Levoy. "Stanford Bunny.", 1994. Stanford University Computer Graphics Laboratory.

[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.

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

[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." volume 2, p. 18. ACM, 2019.

[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.

[WLF18] Z. Wang, L.Wu, M. Fraftarcangeli, M. Tang, and H. Wang. "Parallel Multigrid for nonlinear cloth simulation." *Computer Graphics Forum*, **37**(7):131–141, 2018.

[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.

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

[ZLB16] D. Zhao, Y. Li, and J. Barbič. "Asynchronous Implicit Backward Euler Integration." 2016.