UNIVERSITY OF CALIFORNIA

Los Angeles

Novel Implicit Discretization and Solutions for Elastic Solids and Fluids

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

by

Jingyu Chen

2024

ABSTRACT OF THE DISSERTATION

Novel Implicit Discretization and Solutions for Elastic Solids and Fluids

by

Jingyu Chen
Doctor of Philosophy in Mechanical Engineering
University of California, Los Angeles, 2024
Professor Joseph M. Teran, Co-Chair
Professor Hossein Pirouz Kavehpour, Co-Chair

Physics-based simulations are a powerful tool in both computer graphics and engineering applications. Implicit discretization is essential for accurate, stable, and efficient simulations of solids and fluids.

In this thesis, we first present a novel implicit Material Point Method (MPM) discretization of spatially varying surface energies. Our discretization is based on surface energy, enabling implicit time stepping and capturing surface gradients without explicitly resolving them as in traction-condition-based approaches. We include an implicit discretization of thermomechanical material coupling with novel particle-based enforcement of Robin boundary conditions. Lastly, we design a particle resampling approach for perfect conservations of linear and angular momentum with Affine-Particle-In-Cell (APIC) [JSS15].

The second part presents a novel deep-learning approach to approximate the solution of large, sparse, symmetric, positive-definite linear systems of equations. Our method is motivated by the conjugate gradients algorithm that iteratively selects search directions for minimizing the matrix norm of the approximation error. We use a deep neural network to

accelerate convergence via data-driven improvement of the search direction at each iteration. We demonstrate the efficacy of our approach on discretized Poisson equations with millions of degrees of freedom. Our algorithm can reduce the linear system residual to the target tolerance in a small number of iterations, independent of the problem size, and generalize effectively to various systems beyond those encountered during training.

Finally, we present improvements to Position Based Dynamics (PBD) [MHH07] and Extended Position Based Dynamics (XPBD) [MMC16] methods, which are variants of implicit time integrator. PBD/XPBD are powerful methods for the real-time simulation of elastic objects, but they do not always converge. We isolate the root cause in the approximate linearization of the nonlinear backward Euler systems utilized by XPBD. We provide two extensions to XPBD to address the non-convergence and support general hyperelastic models. The following chapter presents a novel position-based nonlinear Gauss-Seidel approach for quasistatic simulations of elastic objects. This approach retains the essential PBD feature of stable behavior with limited computational budgets and allows for convergent behavior when the budgets expand.

The dissertation of Jingyu Chen is approved.

Jeffrey D. Eldredge

Lihua Jin

Hossein Pirouz Kavehpour, Committee Co-Chair

Joseph M. Teran, Committee Co-Chair

University of California, Los Angeles

2024

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my advisor, Professor Joseph Teran, for his continuous guidance and support of my study at UCLA. Working with Joey in the past years has been the most wonderful adventure I have ever had. His insight and knowledge of this research area have truly helped me grow as a researcher. I would also like to thank him for providing me with many great opportunities which paved the way for my future career.

I would like to thank Professor Pirouz Kavehpour for serving as the co-chair of my thesis committee. I really appreciate his help and support over the past few years, which makes it possible for me to freely explore my research interests and pursue industry experience. I would also like to thank Professor Lihua Jin and Professor Jeffrey Eldredge for serving as members of my thesis committee. Taking classes with them many years ago has laid a solid foundation for my research. I would also like to thank all my committee members for their valuable advice for improving my work.

I would like to thank my colleagues and members of Joey's research group, David Hyde, Steven Gagniere, Alan Marquez-Razon, Elias Gueidon, Victoria Kala, Yizhou Chen, Yushan Han, Ayano Kaneda, and Osman Akar for their collaboration and help.

I would like to express gratitude to my mentors Miguel Otaduy, Guo Qi, and Chuyuan Fu for sharing their knowledge and experience with me during my internships at Meta Reality Labs, Amazon, and Everyday Robots.

I would like to thank my friends and family for all the joys and support they have given me in the past years. Special thanks go to my parents, who have always been supporting me on the other side of the Pacific Ocean.

My deepest gratitude goes to my partner, Tian Qiu, for her persistent love and companionship. Without her understanding and support, I wouldn't be able to finish this thesis.

VITA

2017	B.S. (Mechanical Engineering), Rensselaer Polytechnic Institute
2019	M.S. (Mechanical Engineering), UCLA
2019 - 2024	Research Assistant, UCLA
2021 - 2023	Teaching Assistant, UCLA
2021	Resident at Everyday Robot, X - The Moonshot Factory
2022	Summer Applied Scientist Intern, Amazon
2023	Summer Research Scientist Intern, Meta Reality Labs Research

PUBLICATIONS

Jingyu Chen, Victoria Kala, Alan Marquez-Razon, Elias Gueidon, David A. B. Hyde, and Joseph Teran. 2021. A momentum-conserving implicit material point method for surface tension with contact angles and spatial gradients. ACM Trans. Graph. 40, 4, Article 111 (August 2021), 16 pages. https://doi.org/10.1145/3450626.3459874

Ayano Kaneda, Osman Akar, Jingyu Chen, Victoria A. T. Kala, David Hyde, Joseph Teran. 2023. A Deep Conjugate Direction Method for Iteratively Solving Linear Systems. Proceedings of the 40th International Conference on Machine Learning, in Proceedings of Machine Learning Research 202:15720-15736

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, and Joseph Teran. 2023. Position-Based Nonlinear Gauss-Seidel for Quasistatic Hyperelasticity. arXiv preprint arXiv:2306.09021.

CHAPTER 1

Introduction

Physics-based simulations are ubiquitous in engineering and computer graphics applications. The advances in computing power have enabled industry and research laboratories to create new methodologies to understand the physics and attack complex problems in various engineering analyses. Simulations have demonstrated the strength in creating dramatic visual effects in film productions, generating realistic animations in video games, and achieving natural interactions in virtual reality.

To accurately reproduce the physical phenomena, these simulations are usually based on the numerical solutions of partial differential equations (PDEs) that describe the corresponding physics. The demands for efficient, robust, and realistic simulations have been increasing over the years. Among many simulation algorithms, implicit time stepping schemes have been widely used because of the accuracy, stability, and efficiency. Examples include but are not limited to computational fluid dynamics for studying the flow physics [Cho67, FPS19], nonlinear finite element analysis for computer-aided design [BLM13], computer animations of cloth and garments [BW98], smoke and fire simulations for visual effects [Sta99, FSJ01].

In this dissertation, we develop a novel implicit Material Point Method for simulating spatially varying surface tension phenomena, a deep learning-based preconditioning strategy for solving large sparse linear systems, and improvements to the family of Position-Based Dynamics methods [MHH07] for simulating the dynamics of elastic objects.

1.1 Material Point Method for Surface Tension

Surface tension driven flows like those in milk crowns [ZZK15], droplet coalescence [TWG10, WTG10, DHB16, YHW16, LLD20] and bubble formation [ZQC14, DBW15, HIK20] comprise some of the most visually compelling fluid motions. Indeed surface tension effects have been well examined in the computer graphics and broader computational physics literature. We design a novel approach for simulating surface tension driven phenomena that arise from spatial variations in cohesion and adhesion forces at the interface between two liquids. This is often called the Marangoni effect [SS60, VS15] and perhaps the most famous example is the tears of wine phenomenon [Tho55]. Other notable examples of the Marangoni effect include repulsive flows induced by a soap droplet on a water surface as well as the dynamics of molten waxes and metals [Lan02, FPF04].

We build on the work of Hyde et al. [HGM20] and show that efficient implicit time stepping with Marangoni effects is achievable with PIC. As in [HGM20] we observe that similarities with hyperelasticity suggest that the Material Point Method (MPM) [SCS94] is the appropriate version of PIC.

We generalize the MPM technique in [HGM20] and improve on its core functionality. Their method introduces temporary massless surface tension particles to represent the liquid interface Γ and its weighted boundary normals. However, this approach disrupts the perfect conservation of grid linear and angular momentum, particularly when surface tension forces act on nodes with no mass. To overcome this, we propose a novel mass and momentum resampling technique, introducing two new types of temporary particles that restore ideal conservation.

Since variations in surface energy are typically based on temperature and/or concentration gradients, we couple our surface tension coefficients with thermodynamically driven quantities. Furthermore, we resolve solid to liquid and liquid to solid phase changes as a function of temperature since many Marangoni effects arise from melting and cooling. No-

tably, we show that our novel conservative resampling naturally improves discretization of Robin and Neumann boundary conditions on the interface Γ needed for convection/diffusion of temperature and concentration.

In summary, our primary contributions are:

- A novel implicit MPM discretization of spatially varying surface tension forces.
- A momentum-conserving particle resampling technique for particles near the surface tension liquid interface.
- An implicit MPM discretization of the convection/diffusion evolution of temperature/concentration coupled to the surface tension coefficient including a novel particle-based Robin boundary condition.

1.2 Deep Conjugate Direction Method

The solution of large, sparse systems of linear equations is ubiquitous when partial differential equations (PDEs) are discretized to computationally simulate complex natural phenomena. We use the notation

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{1.1}$$

where the dimension n of the matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ and the vector $\mathbf{b} \in \mathbb{R}^n$ correlates with spatial fidelity of the computational domain. The quality and realism of a simulation are proportional to this spatial fidelity; typical modern applications of numerical PDEs require solving linear systems with millions of unknowns. In such applications, numerical approximation to the solution of these linear systems is typically the bottleneck in overall performance; accordingly, practitioners have spent decades devising specialized algorithms for their efficient solution [GL12, Saa03].

In the present work, we consider sparse linear systems that arise from discrete Poisson

equations in incompressible flow applications [Cho67, FSJ01, Bri08]. These equations yield discrete elliptic operators that are typically symmetric positive (semi) definite, which means that the preconditioned conjugate gradients method (PCG) can be used to minimize iteration counts [Saa03, HS52, Sti52].

Recently, data-driven approaches that leverage deep learning techniques have shown promise for solving linear systems. Various researchers have investigated machine learning estimation of multigrid parameters [GGB19, GSK16, LGM20]. Others have developed machine learning methods to estimate preconditioners [GA18, Sta20, IFH20] and initial guesses for iterative methods [LKB21, UBF20, ADP20]. [TSS17] and [YYX16] develop non-iterative machine learning approximations of the inverse of discrete Poisson equations from incompressible flow.

We develop a novel conjugate gradients-style iterative method, enabled by deep learning, for approximating the solution of SPD linear systems, which we call the deep conjugate direction method (DCDM).

We highlight the features of our method:

- We use a convolutional neural network (CNN) as an approximation of the inverse of the matrix in order to generate more efficient search directions.
- Our approach allows for efficient training and generalization to problems unseen (new matrices $\mathbf{A}$ and new right-hand sides $\mathbf{b}$).
- As an iterative method, our method can reduce the linear system residuals to the designated tolerance.

1.3 Efficient Simulations of Elastic Solids

When simulating large strain hyperelastic solids, the governing equations are commonly discretized in space with the finite element method (FEM) [SB12] and in time with implicit

backward Euler [BW98] or quasistatics [TSI05, BW08, ZBK18, RPP17, SA07]. Hyperelastic solid models define continuum stresses from a notion of elastic potential. In graphics applications, these models are commonly used for simulation-based enhancement of character flesh and musculature animation [MZS11, WZB20, SGK18, FLP14, MFJ21, TSB05]. In these applications, the constitutive control enabled by continuum models is essential for realism. Various methods have been proposed for solving the FEM-discretized equations of motions for these materials [LGL19, NOB16, BML14, TSI05, GSS15, MMC16, KYT06, ZBK18, ZLB16]. 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 [LBO13, BML14, ZBK18]. Real-time applications have very modest computational budgets and this restricts which techniques can be used. In these cases, Newton's method is often outperformed by alternative techniques.

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 budgets. 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. However, constitutive control over PBD behavior is challenging because the effective material stiffness varies 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. In this case, the La-

grange multiplier terms can be interpreted as stress-like and associated with enforcing the constraints.

1.3.1 Primal Extended Position Based Dynamics

XPBD can only discretize hyperelastic models that are quadratic in some notion of strain constraint [MMC16, MM21], which 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. Each formulation 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 5.1.3.1) 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.

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.

1.3.2 Position-Based Nonlinear Gauss-Seidel for Quasistatic Hyperelasticity

Despite its many strengths, PBD/XPBD has a few limitations that hinder its use in quasistatic applications. First, PBD/XPBD is designed for backward Euler, and omitting the inertial terms for quasistatics is not possible (it would require dividing by zero). Additionally, when PBD is viewed as the limit of infinite stiffness in XPBD, this limit incorrectly and irrevocably removes the external forcing terms. Lastly, the constraint-centric iteration in PBD/XPBD solves the positions involved in a single constraint at a time, ignoring the effects of adjacent constraints, which causes artifacts near vertices that appear in different types of constraints.

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. 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.

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.

1.4 Dissertation Overview

Chapter 2 provides a brief overview of continuum mechanics theories and the governing equations derivations. A brief review of the material point method is also included.

Chapter 3 presents a novel implicit Material Point Method (MPM) discretization of surface tension forces that arise from spatially varying surface energies. With our particle resampling approach, perfect conservation of linear and angular momentum is achieved with Affine-Particle-In-Cell (APIC) [JSS15]. This chapter is based on [CKM21].

Chapter 4 presents a deep learning method for approximating solutions to large, sparse, symmetric, positive-definite linear systems, common in numerical solutions of partial differential equations. Inspired by the conjugate gradients algorithm, the approach uses a deep neural network to enhance convergence by improving search directions through data-driven techniques. This chapter is based on [KAC23].

Chapter 5 discusses the root causes of the non-convergent behavior of the popular XPBD method for simulating elastic objects. To address this issue, we present B-PXPBD and FP-PXPBD. The former is a small modification to existing models addressable by the original XPBD. The latter is a more general formulation that extends XPBD to arbitrary hyperelasticity. This chapter is based on [CHC23a].

Chapter 6 presents a position-based nonlinear Gauss-Seidel (PBNG) method for robust and efficient quasistatic simulations of elastic objects. Compared to standard Newton's method, this approach maintains stability within constrained budgets. When compared to a constraint-based nonlinear Gauss-Seidel method, such as PBD [MHH07], PBNG achieves convergent behavior with expanded budgets. This chapter is based on [CHC23b].

CHAPTER 2

Continuum Mechanics and Material Point Method

2.1 Continuum Mechanics

In this section, we review three fundamental elements of continuum mechanics - kinematic theory, balance laws, and constitutive relations. The derivations are primarily based on [GS08, BW08, BLM13, MH94].

2.1.1 Kinematic Theory

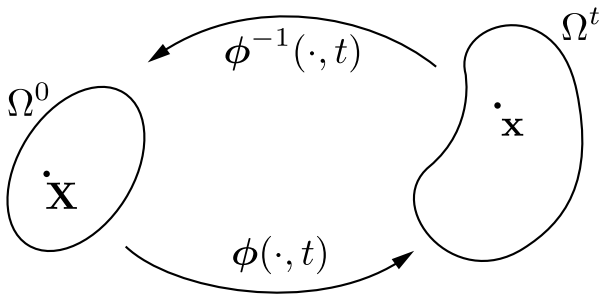


Figure 2.1: The object at time $t = 0$ is considered as the reference configuration. The reference configuration is the object at time t, which can be mapped from the reference configuration.

Continuum Motion Consider a body moving in space as in Figure 2.1. The domain of the body in the initial state is denoted as $\Omega^0 \subset \mathbb{R}^d$ with $\mathbb{R}^d$ being d-dimension Euclidean space. The domain of the body at time t is denoted as $\Omega^t \subset \mathbb{R}^d$. Each particle in a

continuum body can be expressed using two sets of coordinates: the material coordinates $\mathbf{X}$ in Ω^0 and the spatial coordinates $\mathbf{x}$ in Ω^t. The material motion can be described by a map $\boldsymbol{\phi}(\cdot, t) : \Omega^0 \to \Omega^t$. For a point $\mathbf{X}$ in the reference configuration Ω^0, under the deformation map $\boldsymbol{\phi}$, the point $\mathbf{X}$ is mapped to $\mathbf{x}$ in the deformed configuration Ω^t

$$\mathbf{x} = \mathbf{x}(\mathbf{X}, t) = \boldsymbol{\phi}(\mathbf{X}, t). \tag{2.1}$$

The displacement of a particle is given by

$$\mathbf{u}(\mathbf{X}, t) = \phi(\mathbf{X}, t) - \phi(\mathbf{X}, 0) = \mathbf{x}(\mathbf{X}, t) - \mathbf{X} \tag{2.2}$$

When describing the motion of a continuum body, we can either track the motion of a fixed particle, i.e., in the Lagrangian perspective, or measure the motion in a fixed spatial location, i.e., in the Eulerian perspective. For the velocity of a material point, if we compute the time rate of change of the position with $\mathbf{X}$ held constant, we have the Lagrangian velocity definition

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

The Lagrangian acceleration is

$$\mathbf{A}(\mathbf{X}, t) = \frac{\partial^2 \boldsymbol{\phi}}{\partial t^2}(\mathbf{X}, t) = \frac{\partial \mathbf{V}}{\partial t}(\mathbf{X}, t) \tag{2.4}$$

In Eulerian perspective, the velocity and acceleration are

$$\mathbf{v}(\mathbf{x}, t) = \mathbf{V}(\boldsymbol{\phi}^{-1}(\mathbf{x}, t), t) \tag{2.5}$$

$$\mathbf{a}(\mathbf{x}, t) = \frac{\partial \mathbf{V}}{\partial t}(\mathbf{X}, t) = \frac{\partial \mathbf{v}}{\partial t}(\mathbf{x}, t) + \frac{\partial \mathbf{v}}{\partial \mathbf{x}}(\mathbf{x}, t) \cdot \mathbf{v}(\mathbf{x}, t) \tag{2.6}$$

where $\frac{D(\cdot)}{Dt} = \frac{\partial(\cdot)}{\partial t} + \mathbf{v} \cdot \frac{\partial(\cdot)}{\partial \mathbf{x}}$ is the material derivatives.

Measures of Deformation In finite deformation theory, the deformation gradient $\mathbf{F}$ relates the quantities in the reference configuration and those after the deformation. The deformation gradient is defined as the Jacobian of the deformation mapping

$$\mathbf{F}(\mathbf{X}, t) = \frac{\partial \boldsymbol{\phi}}{\partial \mathbf{X}}(\mathbf{X}, t) = \frac{\partial \mathbf{x}}{\partial \mathbf{X}}(\mathbf{X}, t) \tag{2.7}$$

$\mathbf{F} \in \mathbb{R}^{d\times d}$ can be decomposed using polar decomposition

$$\mathbf{F} = \mathbf{RU} = \mathbf{VR} \tag{2.8}$$

where $\mathbf{R}$ is a rotation tensor. $\mathbf{U}$ and $\mathbf{V}$ are the right and left stretch tensor. $\mathbf{C} = \mathbf{F}^T\mathbf{F} = \mathbf{U}^2$ is called the right Cauchy-Green strain tensor. $\mathbf{b} = \mathbf{F}\mathbf{F}^T = \mathbf{V}^2$ is the left Cauchy-Green strain tensor. In computer graphics, the Green-Lagrange strain tensor $\mathbf{E} = \frac{1}{2}(\mathbf{C} - \mathbf{I})$ is also commonly used for defining elastic deformation.

The volume and area changes can be related to the deformation gradient. Consider an infinitesimal element in the reference configuration, $dV = dL_1 dL_2 dL_3$. In the deformed configuration, $dv = dl_1 dl_2 dl_3$. Each vector follows $d\boldsymbol{l}_i = \mathbf{F} d\boldsymbol{L}_i$. Then, the volume follows the following relation.

$$dv = \det(\mathbf{F})dV = JdV \tag{2.9}$$

It can also be shown that the infinitesimal area element can be related.

$$\mathbf{n}ds = J\mathbf{F}^{-T}\mathbf{N}dS \tag{2.10}$$

where $\mathbf{N}$, $\mathbf{n}$ are the surface normal before and after the deformation, and dS, ds are the corresponding surface areas. This relation is also known as Nanson's relation [BLM13].

2.1.2 Balance Laws

The conservation laws are the foundation of continuum mechanics. The conservation of mass, momentum, and energy can be derived from the Reynolds' Transport Theorem [BLM13]. For any quantity f (either scalar or vector), we have

$$\frac{D}{Dt}\int_{\Omega^t} f(\mathbf{x}, t)d\mathbf{x} = \int_{\Omega^t}\left(\frac{Df(\mathbf{x}, t)}{Dt} + f\nabla\cdot\mathbf{v}\right)d\mathbf{x} \tag{2.11}$$

For simulations that do not involve temperature change, usually, only the mass and momentum conservation are considered. Here, we derive the conservation laws from the Eulerian perspective.

Conservation of Mass The total mass m of the continuum must conserve throughout time.

$$\frac{Dm}{Dt} = \frac{D}{Dt}\int_{\Omega^t} \rho(\mathbf{x}, t) d\mathbf{x} = 0 \tag{2.12}$$

Apply Reynolds' transport theorem to obtain the integral form of the mass balance equations.

$$\int_{\Omega^t} \left(\frac{D\rho(\mathbf{x}, t)}{Dt} + \rho \nabla \cdot \mathbf{v} \right) d\mathbf{x} = 0 \tag{2.13}$$

The differential form of the mass balance can be obtained.

$$\frac{D\rho}{Dt} + \rho \nabla \cdot \mathbf{v} = 0 \tag{2.14}$$

Conservation of Linear Momemtum The linear momentum obeys the Newton's second law

$$\frac{D\boldsymbol{p}(t)}{Dt} = \mathbf{f}(t) \tag{2.15}$$

where $\boldsymbol{p}(t) = \int_{\Omega^t} \rho \mathbf{v}(\mathbf{x}, t) d\mathbf{x}$ and $\mathbf{f}(t) = \int_{\Omega^t} \rho \mathbf{b}(\mathbf{x}, t) d\mathbf{x} + \int_{\partial\Omega^t} \mathbf{t}(\mathbf{x}, t) d\boldsymbol{s}$. $\mathbf{b}$ is the body force and $\mathbf{t}$ is the surface traction. Then, apply the Reynolds' transport theorem and simplify by using the continuity equation (mass balance) (2.14).

$$\frac{D}{Dt}\int_{\Omega^t} \rho \mathbf{v}(\mathbf{x}, t) d\mathbf{x} = \int_{\Omega^t} \left(\frac{D(\rho \mathbf{v})}{Dt} + \rho \mathbf{v} (\nabla \cdot \mathbf{v}) \right) d\mathbf{x} \tag{2.16a}$$

$$= \int_{\Omega^t} \left[\rho \frac{D\mathbf{v}}{Dt} + \mathbf{v} \left(\frac{D\rho}{Dt} + \rho \nabla \cdot \mathbf{v} \right) \right] d\mathbf{x} \tag{2.16b}$$

$$= \int_{\Omega^t} \rho \frac{D\mathbf{v}}{Dt} d\mathbf{x} \tag{2.16c}$$

The momentum balance can be written as

$$\int_{\Omega^t} \rho \frac{D\mathbf{v}}{Dt} = \int_{\Omega^t} \rho \mathbf{b}(\mathbf{x}, t) d\mathbf{x} + \int_{\partial\Omega^t} \mathbf{t}(\mathbf{x}, t) d\mathbf{s}(\mathbf{x}) \tag{2.17}$$

Use Cauchy's relation and divergence theorem, the surface traction term can be written as

$$\int_{\partial\Omega^t} \mathbf{t} d\mathbf{s} = \int_{\partial\Omega^t} \boldsymbol{\sigma} \cdot \mathbf{n} d\mathbf{s} = \int_{\Omega^t} \nabla \cdot \boldsymbol{\sigma} d\mathbf{x} \tag{2.18}$$

The differential form of the momentum balance is

$$\rho \frac{D\mathbf{v}}{Dt} = \rho \mathbf{b} + \nabla \cdot \boldsymbol{\sigma} \tag{2.19}$$

Here, $\boldsymbol{\sigma}$ is the Cauchy stress tensor.

2.1.3 Constitutive Relations

Hyperelastic Solids In computer graphics, hyperelastic solids are commonly used for simulating deformable bodies. The work done by the stress only depends on the initial and final configurations. A strain energy density function Ψ is defined for the hyperelastic material. Instead of using the Cauchy stress $\boldsymbol{\sigma}$, the first Piola-Kirchhoff (PK1) stress $\mathbf{P} = J\boldsymbol{\sigma}\mathbf{F}^{-T}$ is used

$$\mathbf{P} = \frac{\partial \Psi}{\partial \mathbf{F}} \tag{2.20}$$

Newtonian Fluids For Newtonian fluids, the viscous stress is linear to the strain rate $\nabla \mathbf{v}$ [CMM90]. The Cauchy stress now becomes

$$\boldsymbol{\sigma} = -\left(p + \frac{2}{3}\mu \nabla \cdot \mathbf{v}\right)\mathbf{I} + 2\mu\left(\nabla \mathbf{v} + (\nabla \mathbf{v})^T\right) \tag{2.21}$$

where p is the hydrostatic pressure and μ is the dynamic viscosity. For incompressible Newtonian fluid, $\nabla \cdot \mathbf{v} = 0$. The momentum balance equation (2.19) becomes

$$\rho \frac{D\mathbf{v}}{Dt} = -\nabla p + \mu \nabla^2 \mathbf{v} + \rho \mathbf{b} \tag{2.22}$$

In computer graphics, for smoke and fire simulations, we usually ignore the viscosity term [FSJ01]. The incompressible Navier-Stokes equation (2.22) becomes the incompressible Euler equation

$$\rho \frac{D\mathbf{v}}{Dt} = -\nabla p + \rho \mathbf{b} \tag{2.23}$$

2.2 Material Point Method

In this section, we derive first provide a brief review of the material point method. Then, we derive the equations for implementing the material point method simulations. The derivations are based on [JST16].

2.2.1 MPM Algorithm

Material point method (MPM) is an extension of the Particle-In-Cell (PIC) method [Har64] and the FLuid-Implicit-Particle (FLIP) method [BR86], which allows particles to carry history-dependent information [SCS94]. In MPM, the continuum is discretized as particles. Both Lagrangian particles and Eulerian background grids are used in MPM. At each time step of MPM, each particle needs to transfer its mass and momentum to the background Eulerian grid nodes and retrieve the data after the grid solves. Each material point can carry history-dependent information. A MPM simulation step can be summarized as:

1. transferring the particle information to grid,
2. solving for the updated grid information,
3. transferring the grid information to particle,
4. updating particle information and advect particles.

2.2.2 Weak Form

Similar to the finite element method, MPM also formulates based on the weak form of the PDEs. Start from strong form (2.19).

$$\rho \frac{D\mathbf{v}}{Dt} = \rho \mathbf{a} = \rho \mathbf{b} + \nabla \cdot \boldsymbol{\sigma} \tag{2.24}$$

Apply an arbitrary test function $\boldsymbol{q}(\cdot, t)$ and write the weak form of the force balance in index notation.

$$\int_{\Omega^t} \rho a_i q_i d\mathbf{x} = \int_{\Omega^t} \rho b_i q_i d\mathbf{x} + \int_{\Omega^t} \sigma_{ij,j} q_i d\mathbf{x} \tag{2.25}$$

Use integration by parts and divergence theorem

$$\int_{\Omega^t} \sigma_{ij,j} q_i d\mathbf{x} = \int_{\Omega^t} (\sigma_{ij} q_i)_{,j} d\mathbf{x} - \int_{\Omega^t} \sigma_{ij} q_{i,j} d\mathbf{x} \tag{2.26a}$$

$$= \int_{\partial\Omega^t} \sigma_{ij} q_i n_j d\mathbf{s} - \int_{\Omega^t} \sigma_{ij} q_{i,j} d\mathbf{x} \tag{2.26b}$$

Again, use Cauchy's relation for the traction boundary condition, and we have the weak form of the force balance.

$$\int_{\Omega^t} \rho a_i q_i d\mathbf{x} = \int_{\Omega^t} \rho b_i q_i d\mathbf{x} - \int_{\Omega^t} \sigma_{ij} q_{i,j} d\mathbf{x} + \int_{\partial\Omega_t^N} t_i q_i d\mathbf{s} \tag{2.27}$$

2.2.3 Discretization

Interpolation Function Similar to the shape function of the finite element method [Hug00] or the kernel function of the smooth particle hydrodynamics (SPH) [Mon92], MPM uses a set of B-spline interpolation functions to determine how much information each grid node will receive from a particle depending on the relative distance. In the finite element perspective, the MPM particles are the quadrature points, while the background grid is the mesh [JST16].

Usually, MPM requires C^1 continuity for the interpolation function to avoid cell-crossing instability. Considering the computational cost, the following quadratic B-spline kernel is usually the preferred choice in practice.

$$N(x) = \begin{cases} \frac{3}{4} - |x|^2 & 0 \le |x| < \frac{1}{2} \\ \frac{1}{2}\left(\frac{3}{2} - |x|\right)^2 & \frac{1}{2} \le x < \frac{3}{2} \\ 0 & \frac{3}{2} \le |x| \end{cases} \tag{2.28}$$

In 3D, for a grid node with index $\mathbf{i} = (i, j, k)$ and p-th particle, the interpolation function is defined as [SKB08].

$$N_{\mathbf{i}}(\mathbf{x}_p) = N(\frac{1}{h}(x_p - x_{\mathbf{i}}))N(\frac{1}{h}(y_p - y_{\mathbf{i}}))N(\frac{1}{h}(z_p - z_{\mathbf{i}})) \tag{2.29}$$

For simplicity, denote $w_{\mathbf{i}p} = N_{\mathbf{i}}(\mathbf{x}_p)$ and $\nabla w_{\mathbf{i}p} = \nabla N_{\mathbf{i}}(\mathbf{x}_p)$.

Eulerian/Lagrangian Mass and Momentum In discretized formulation, the total mass of the particles and those transferred to the grid should match.

$$\sum_{\mathbf{i}} m_{\mathbf{i}} = \sum_p m_p \tag{2.30}$$

The same rule applies to the momentum.

$$\sum_{\mathbf{i}} m_{\mathbf{i}} \mathbf{v}_{\mathbf{i}} = \sum_{p} m_p \mathbf{v}_p \tag{2.31}$$

Deformation Gradient Update As shown in (2.7), $\mathbf{F}(\mathbf{X}, t) = \frac{\partial \mathbf{x}}{\partial \mathbf{X}}(\mathbf{X}, t)$. Time derivative of $\mathbf{F}$ is

$$\frac{\partial \mathbf{F}}{\partial t}(\mathbf{X}, t) = \frac{\partial \mathbf{v}}{\partial \mathbf{x}}(\phi(\mathbf{X}, t), t)\mathbf{F}(\mathbf{X}, t) \tag{2.32}$$

The Eulerian velocity $\mathbf{v}^{n+1}$ at time t^{n+1} is only a function of $\mathbf{x}$. Then, we can write $\mathbf{v}^{n+1}(\mathbf{x}) = \mathbf{V}(\phi^{-1}(\mathbf{x}, t^n), t^{n+1})$ or $\mathbf{v}^{n+1}(\phi(\mathbf{X}, t^n)) = \mathbf{V}(\mathbf{X}, t^{n+1})$. The time derivative can be approximated as

$$\frac{\partial \mathbf{F}}{\partial t}(\mathbf{X}_p, t^{n+1}) = \frac{\partial \mathbf{v}^{n+1}}{\partial \mathbf{x}}(\phi(\mathbf{X}_p, t^n))\mathbf{F}(\mathbf{X}_p, t^n) \approx \frac{\mathbf{F}_p^{n+1} - \mathbf{F}_p^n}{\Delta t} \tag{2.33}$$

Use the interpolation function, and the velocity gradient can be calculated by

$$\frac{\partial \mathbf{v}^{n+1}}{\partial \mathbf{x}}(\mathbf{x}_p^n) = \sum_{\mathbf{i}} \mathbf{v}_{\mathbf{i}}^{n+1} \left(\frac{\partial N_{\mathbf{i}}}{\partial \mathbf{x}}(\mathbf{x}) \right)^T = \sum_{\mathbf{i}} \mathbf{v}_{\mathbf{i}}^{n+1} (\nabla w_{\mathbf{i}p})^T \tag{2.34}$$

Then, the deformation gradient can be update as

$$\mathbf{F}_p^{n+1} = \left(\mathbf{I} + \Delta t \sum_{\mathbf{i}} \mathbf{v}_{\mathbf{i}}^{n+1} (\nabla w_{\mathbf{i}p})^T \right) \mathbf{F}_p^n \tag{2.35}$$

2.2.3.1 Explicit Formulation

The weak form balance equation (2.27) needs to be discretized both in time and space. Here, we first consider the explicit formulation (symplectic Euler) and write the spatially discretized equation, following [JST16]. Note that instead of using a single index i, a composite index is used. By convention, $\mathbf{i}, \mathbf{j}, \mathbf{k}$ denote the grid node index and Greek letters denote the components.

$$\int_{\Omega^{t\Delta x}} \rho q_\alpha a_\alpha d\mathbf{x} = \int_{\Omega^{t\Delta x}} \rho q_\alpha b_\alpha d\mathbf{x} - \int_{\Omega^{t\Delta x}} \sigma_{\alpha\beta} q_{\alpha,\beta} d\mathbf{x} + \int_{\partial\Omega_t^{\Delta x, N}} q_\alpha t_\alpha d\mathbf{s} \tag{2.36}$$

The acceleration term can be approximated as

$$\int_{\Omega^{t\Delta x}} \rho q_\alpha a_\alpha d\mathbf{x} = \frac{1}{\Delta t} \int_{\Omega^{t\Delta x}} \rho q_\alpha (v_\alpha^{n+1} - v_\alpha^n) d\mathbf{x} \tag{2.37}$$

Plug in the interpolation function $N_\mathbf{i} = N_\mathbf{i}(\mathbf{x}^n)$

$$\begin{aligned} \frac{1}{\Delta t} \int_{\Omega^{t\Delta x}} \rho^n q_{\mathbf{i}\alpha}^n N_\mathbf{i} (v_{\mathbf{j}\alpha}^{n+1} - v_{\mathbf{j}\alpha}^n) N_\mathbf{j} d\mathbf{x} = & \int_{\Omega^{t\Delta x}} \rho^n q_{\mathbf{i}\alpha}^n N_\mathbf{i} b_{\mathbf{j}\alpha}^n N_\mathbf{j} d\mathbf{x} \\ & - \int_{\Omega^{t\Delta x}} \sigma_{\alpha\beta}^n q_{\mathbf{i}\alpha}^n N_{\mathbf{i},\beta} d\mathbf{x} + \int_{\partial\Omega_t^{\Delta x,N}} q_{\mathbf{i}\alpha}^n N_\mathbf{i} t_\alpha^n d\mathbf{s} \end{aligned} \tag{2.38}$$

Since $q_{\mathbf{i}\alpha}^n$ is arbitrary, (2.38) must holds for any $q_{\mathbf{i}\alpha}^n$. The following equation must hold.

$$\begin{aligned} \int_{\Omega^{t\Delta x}} \rho^n N_\mathbf{i} \frac{(v_{\mathbf{j}\alpha}^{n+1} - v_{\mathbf{j}\alpha}^n)}{\Delta t} N_\mathbf{j} d\mathbf{x} = & \int_{\Omega^{t\Delta x}} \rho^n N_\mathbf{i} b_{\mathbf{j}\alpha}^n N_\mathbf{j} d\mathbf{x} \\ & - \int_{\Omega^{t\Delta x}} \sigma_{\alpha\beta}^n N_{\mathbf{i},\beta} d\mathbf{x} + \int_{\partial\Omega_t^{\Delta x,N}} N_\mathbf{i} t_\alpha^n d\mathbf{s} \end{aligned} \tag{2.39}$$

Use the lumped mass, and LHS can be approximated as follows. Note that in lumped mass approximation, $m_{\mathbf{ij}} \neq 0$ if and only if $\mathbf{i} = \mathbf{j}$. Therefore, we abbreviate $m_{\mathbf{ij}} v_{\mathbf{j}\alpha}$ as $(mv)_{\mathbf{i}\alpha}$.

$$\int_{\Omega^{t\Delta x}} \rho^n N_\mathbf{i} \frac{(v_{\mathbf{j}\alpha}^{n+1} - v_{\mathbf{j}\alpha}^n)}{\Delta t} N_\mathbf{j} d\mathbf{x} \approx \frac{(mv)_{\mathbf{i}\alpha}^{n+1} - (mv)_{\mathbf{i}\alpha}^n}{\Delta t} \tag{2.40}$$

The stress is estimated as $\boldsymbol{\sigma}_p = \boldsymbol{\sigma}(\mathbf{x}_p^n, t^n)$ at each particle

$$\int_{\Omega^{t\Delta x}} \sigma_{\alpha\beta}^n N_{\mathbf{i},\beta} d\mathbf{x} \approx \sum_p \boldsymbol{\sigma}_{p_{\alpha\beta}}^n N_{\mathbf{i},\beta}(\mathbf{x}_p^n) V_p^n \tag{2.41}$$

The particle volume can be determined by

$$V_p^n = J_p^n V_p^0 \tag{2.42}$$

Usually, the first Piola-Kirchhoff (PK1) stress is used in MPM because it is simply the derivative of the energy density ψ function with respect to the deformation gradient $\mathbf{F}$. The Cauchy stress can be related to PK1 through the following [BW08].

$$\boldsymbol{\sigma} = \frac{1}{J} \mathbf{P}\mathbf{F}^T \tag{2.43}$$

Plug (2.42) and (2.43) into (2.41)

$$\sum_p \boldsymbol{\sigma}^n_{p_{\alpha\beta}} N_{\mathbf{i},\beta}(\mathbf{x}^n_p) V^n_p = \sum_p \mathbf{P}^n_{p_{\alpha\gamma}} \mathbf{F}^n_{p_{\beta\gamma}} N_{\mathbf{i},\beta}(\mathbf{x}^n_p) V^0_p \tag{2.44}$$

Therefore, the discretized force balance can be written as

$$\frac{(mv)^{n+1}_{\mathbf{i}\alpha} - (mv)^n_{\mathbf{i}\alpha}}{\Delta t} = \int_{\Omega^{t\,\Delta x}} \rho^n N_{\mathbf{i}} b^n_{\mathbf{j}\alpha} N_{\mathbf{j}} d\mathbf{x} + \int_{\partial\Omega_t^{\Delta x,N}} N_{\mathbf{i}} t^n_\alpha d\mathbf{s} - \sum_p \mathbf{P}^n_{p_{\alpha\gamma}} \mathbf{F}^n_{p_{\beta\gamma}} N_{\mathbf{i},\beta}(\mathbf{x}^n_p) V^0_p \tag{2.45}$$

Now, consider the cases without the Neumann boundary condition, and the only body force is gravity. (2.45) can be simplified.

$$\frac{(mv)^{n+1}_{\mathbf{i}\alpha} - (mv)^n_{\mathbf{i}\alpha}}{\Delta t} = m^n_{\mathbf{i}} g_\alpha - \sum_p \mathbf{P}^n_{p_{\alpha\gamma}} \mathbf{F}^n_{p_{\beta\gamma}} N_{\mathbf{i},\beta}(\mathbf{x}^n_p) V^0_p \tag{2.46}$$

This can also be rewritten in a format that updates the momentum.

$$(mv)^{n+1}_{\mathbf{i}\alpha} = (mv)^n_{\mathbf{i}\alpha} + \Delta t \mathbf{f}_{\mathbf{i}\alpha}(\mathbf{x}^n_{\mathbf{i}}) \tag{2.47}$$

where the force term is $\mathbf{f}_{\mathbf{i}\alpha}(\mathbf{x}^n_{\mathbf{i}}) = m^n_{\mathbf{i}} g_\alpha - \sum_p \mathbf{P}^n_{p_{\alpha\gamma}} \mathbf{F}^n_{p_{\beta\gamma}} N_{\mathbf{i},\beta}(\mathbf{x}^n_p) V^0_p$. For simplicity, the sympletic Euler update scheme with single index $\mathbf{i}$ is

$$\mathbf{x}^{n+1}_{\mathbf{i}} = \mathbf{x}^n_{\mathbf{i}} + \Delta t \mathbf{v}^{n+1}_{\mathbf{i}} \tag{2.48a}$$

$$\mathbf{v}^{n+1}_{\mathbf{i}} = \mathbf{v}^n_{\mathbf{i}} + \Delta t \frac{\mathbf{f}_{\mathbf{i}}(\mathbf{x}^n_{\mathbf{i}})}{m_{\mathbf{i}}} \tag{2.48b}$$

2.2.3.2 Implicit Formulation

Similar to the sympletic Euler update (2.48), the backward Euler update scheme is

$$\mathbf{x}^{n+1}_{\mathbf{i}} = \mathbf{x}^n_{\mathbf{i}} + \Delta t \mathbf{v}^{n+1}_{\mathbf{i}} \tag{2.49a}$$

$$\mathbf{v}^{n+1}_{\mathbf{i}} = \mathbf{v}^n_{\mathbf{i}} + \Delta t \frac{\mathbf{f}_{\mathbf{i}}(\mathbf{x}^{n+1}_{\mathbf{i}})}{m_{\mathbf{i}}} \tag{2.49b}$$

To obtain the update of the velocity, a nonlinear system of equations needs to be solved. Newton's method is a common strategy to solve these equations.

2.2.4 Transfer Schemes

PIC and FLIP Transfer schemes are essential to ensure minimum information loss during the momentum transfers and maintain the stability of the simulations. While PIC transfer is too dissipative and FLIP transfer tends to be unstable, [ZB05] use the interpolated PIC and FLIP to control the amount of damping for the particles. The velocity update is

$$\mathbf{v}_p^{n+1} = (1-\alpha)\mathbf{v}_{p_{\text{PIC}}}^{n+1} + \alpha\mathbf{v}_{p_{\text{FLIP}}}^{n+1} \tag{2.50}$$

The PIC and FLIP velocities are

$$\mathbf{v}_{p_{\text{PIC}}}^{n+1} = \sum_{\mathbf{i}} \mathbf{v}_{\mathbf{i}}^{n+1} w_{\mathbf{i}p}^n \tag{2.51}$$

$$\mathbf{v}_{p_{\text{FLIP}}}^{n+1} = \mathbf{v}_p^n + \sum_{\mathbf{i}} (\mathbf{v}_{\mathbf{i}}^{n+1} - \mathbf{v}_{\mathbf{i}}^n) w_{\mathbf{i}p}^n \tag{2.52}$$

APIC/PolyPIC The affine particle-in-cell (APIC) is an angular momentum-conserving transfer scheme. Details of proofs can be found in [JST17]. As in [JSS15, JST16], P2G transfer for APIC is

$$m_{\mathbf{i}} = \sum_p w_{\mathbf{i}p} m_p \tag{2.53}$$

$$(m\mathbf{v})_{\mathbf{i}} = \sum_p w_{\mathbf{i}p} m_p (\mathbf{v}_p + \mathbf{C}_p(\mathbf{x}_{\mathbf{i}} - \mathbf{x}_p)) \tag{2.54}$$

G2P transfer is

$$\mathbf{v}_p = \sum_{\mathbf{i}} w_{\mathbf{i}p} \mathbf{v}_{\mathbf{i}} \tag{2.55}$$

$$\mathbf{C}_p = \sum_{\mathbf{i}} \mathbf{v}_{\mathbf{i}} \left(\frac{\partial N_{\mathbf{i}}}{\partial \mathbf{x}}(\mathbf{x}_p) \right)^T \tag{2.56}$$

Matrix $\mathbf{C}_p^n = \mathbf{B}_p^n (\mathbf{D}_p^n)^{-1}$ is called affine velocity, where

$$\mathbf{D}_p^n = \sum_{\mathbf{i}} w_{\mathbf{i}p}^n m_p (\mathbf{x}_i - \mathbf{x}_p^n)(\mathbf{x}_i - \mathbf{x}_p^n)^T \tag{2.57}$$

$$\mathbf{B}_p^{n+1} = \sum_{\mathbf{i}} w_{\mathbf{i}p}^n \tilde{\mathbf{v}}_{\mathbf{i}}^{n+1} (\mathbf{x}_i - \mathbf{x}_p^n)^T \tag{2.58}$$

For quadratic interpolation function, $\mathbf{D}_p^n = \frac{1}{4}\Delta x^2 \mathbf{I}$.

APIC greatly extends PIC by allowing each particle to have degrees of freedom in rotation, dilation, and shearing. These types of motion are usually called velocity modes. In PIC, there are only 3 modes of translation. Rigid particle-in-cell (RPIC) has 6 and APIC has 12 [JST17].

PolyPIC [FGG17] is a natural extension of APIC by adding more velocity modes, which allows better energy conservation. Despite the great properties of PolyPIC, considering the computational cost, it is primarily used in smoke simulation which needs to achieve highly turbulent behaviors. For elastoplastic solids and water, the small damping that APIC provides is desired to obtain a natural appearance.

CHAPTER 3

A Momentum-Conserving Implicit Material Point Method for Surface Tension with Spatial Gradients

Figure 3.1: Our method enables the simulation of a wide variety of thermomechanical and surface-tension-driven effects. *(Top)* Letter-shaped candles melt and interact. *(Bottom)* A large melting candle; soap spreading on a water surface; water droplets falling and streaking on ramps; partial rebound of a water droplet impact; wine settling in a glass; a water droplet settling on a hydrophobic surface.

3.1 Governing Equations

We build on the work of Hyde et al. [HGM20] and use a energy-based formulation to model the surface tension. The spatial variation in the surface forces can be characterized in terms of the potential energy Ψ^s associated with surface tension:

$$\Psi^s = \int_\Gamma k^\sigma(\mathbf{x}) ds(\mathbf{x}). \tag{3.1}$$

Here the surface tension coefficient k^σ is proportionate to the relative cohesion and adhesion at the interface between the two fluids. Typically this coefficient is constant across the multi-material interface Γ, however with the Marangoni effect the coefficient varies with $\mathbf{x} \in \Gamma$. These variations are typically driven by temperature or concentration gradients and give rise to many subtle, but important visual behaviors where the variation typically causes fluid to flow away from low surface energy regions towards high surface energy regions. To capture the Marangoni effect, most approaches do not work with the potential energy Ψ^s in Equation (3.1) but instead base their discretization on its first variation. This variation results in the interfacial traction condition

$$\mathbf{t} = k^\sigma \kappa \mathbf{n} + \nabla^S k^\sigma. \tag{3.2}$$

Here $\mathbf{t}$ is the force per unit area due to surface tension at the interface Γ, ∇^S is the surface gradient operator at the interface and κ and $\mathbf{n}$ are the interfacial mean curvature and normal, respectively.

In this section, we first define the governing equations for thermomechanically driven phase change of hyperelastic solids and liquids with variable surface energy. As in [HGM20] we also cover the updated Lagrangian kinematics. Lastly, we provide the variational form of the governing equations for use in MPM discretization. We note that throughout the document Greek subscripts are assumed to run from $0, 1, \ldots, d-1$ for the dimension $d = 2, 3$ of the problem. Repeated Greek subscripts imply summation, while sums are explicitly indicated for Latin subscripts. Also, Latin subscripts in bold are used for multi-indices.

3.1.1 Kinematics

We adopt the continuum assumption [GS08] and updated Lagrangian kinematics [BLM13] used by Hyde et al. [HGM20]. At time t we associate our material with subsets $\Omega^t \subset \mathbb{R}^d$, $d = 2, 3$. We use Ω^0 to denote the initial configuration of material with $\mathbf{X} \in \Omega^0$ used to denote particles of material at time $t = 0$. A flow map $\boldsymbol{\phi} : \Omega^0 \times [0, T] \to \mathbb{R}^d$ defines the

material motion of particles $\mathbf{X} \in \Omega^0$ to their time t locations $\mathbf{x} \in \Omega^t$ as $\boldsymbol{\phi}(\mathbf{X}, t) = \mathbf{x}$. The Lagrangian velocity is defined by differentiating the flow map in time $\mathbf{V}(\mathbf{X}, t) = \frac{\partial \boldsymbol{\phi}}{\partial t}(\mathbf{X}, t)$.

3.1.1.1 Eulerian and Updated Lagrangian Representations

The Lagrangian velocity can be difficult to work with in practice since real world observations of material are made in Ω^t not Ω^0. The Eulerian velocity $\mathbf{v} : \Omega^t \to \mathbb{R}^d$ is what we observe in practice. The Eulerian velocity is defined in terms of the inverse flow map $\boldsymbol{\phi}^{-1}(\cdot, t) : \Omega^t \to \Omega^0$ as $\mathbf{v}(\mathbf{x}, t) = \mathbf{V}(\boldsymbol{\phi}^{-1}(\mathbf{x}, t), t)$ where $\boldsymbol{\phi}^{-1}(\mathbf{x}, t) = \mathbf{X}$. In general, we can use the flow map and its inverse to pullback quantities defined over Ω^t and pushforward quantities defined over Ω^0, respectively. For example, given $G : \Omega^0 \to \mathbb{R}$, its pushforward $g : \Omega^t \to \mathbb{R}$ is defined as $g(\mathbf{x}) = G(\boldsymbol{\phi}^{-1}(\mathbf{x}, t))$. This process is related to the material derivative operator $\frac{D}{Dt}$ where $\frac{Dg}{Dt}(\mathbf{x}, t) = \frac{\partial G}{\partial t}(\boldsymbol{\phi}^{-1}(\mathbf{x}, t)) = \frac{\partial g}{\partial t}(\mathbf{x}, t) + \sum_{\alpha=0}^{d-1} \frac{\partial g}{\partial x_\alpha}(\mathbf{x}, t) v_\alpha(\mathbf{x}, t)$ (see e.g., [GS08] for more detail).

In the updated Lagrangian formalism [BLM13] we write quantities over an intermediate configuration of material Ω^s with $0 \leq s < t$. For example, we can define $\hat{g}(\cdot, s) : \Omega^s \to \mathbb{R}$ as $\hat{g}(\tilde{\mathbf{x}}, s) = G(\boldsymbol{\phi}^{-1}(\tilde{\mathbf{x}}, s))$ for $\tilde{\mathbf{x}} \in \Omega^s$. As shown in [HGM20], this is particularly useful when discretizing momentum balance using its variational form. The key observation is that the updated Lagrangian velocity can be written as $\hat{\mathbf{v}}(\tilde{\mathbf{x}}, s, t) = \mathbf{V}(\boldsymbol{\phi}^{-1}(\tilde{\mathbf{x}}, s), t) = \mathbf{v}(\hat{\boldsymbol{\phi}}(\tilde{\mathbf{x}}, s, t), t)$ with $\hat{\boldsymbol{\phi}}(\tilde{\mathbf{x}}, s, t) = \boldsymbol{\phi}(\boldsymbol{\phi}^{-1}(\tilde{\mathbf{x}}, s), t)$ for $\tilde{\mathbf{x}} \in \Omega^s$. Intuitively, $\hat{\boldsymbol{\phi}}(\cdot, s, t) : \Omega^s \to \Omega^t$ is the mapping from the time s configuration to the time t configuration induced by the flow map. This has a simple relation to the material derivative as $\frac{\partial \hat{\mathbf{v}}}{\partial t}(\tilde{\mathbf{x}}, s, t) = \frac{\partial \mathbf{V}}{\partial t}(\boldsymbol{\phi}^{-1}(\tilde{\mathbf{x}}, s), t) = \frac{D\mathbf{v}}{Dt}(\hat{\boldsymbol{\phi}}(\tilde{\mathbf{x}}, s, t), t)$. As in [HGM20] we will generally use upper case for Lagrangian quantities, lower case for Eulerian quantities and hat superscripts for updated Lagrangian quantities.

3.1.1.2 Deformation Gradient

The deformation gradient $\mathbf{F} = \frac{\partial \boldsymbol{\phi}}{\partial \mathbf{X}}$ is defined by differentiating the flow map in space and can be used to quantify the amount of deformation local to a material point. We use

$J = \det(\mathbf{F})$ to denote the deformation gradient determinant. J represents the amount of volumetric dilation at a material point. Furthermore, it is used when changing variables with integration. We also make use of similar notation for the $\hat{\boldsymbol{\phi}}$ mapping from Ω^s to Ω^t, i.e. $\hat{\mathbf{F}} = \frac{\partial \hat{\boldsymbol{\phi}}}{\partial \bar{\mathbf{x}}}$, $\hat{J} = \det\left(\hat{\mathbf{F}}\right)$.

3.1.2 Conservation of Mass and Momentum

Our governing equations primarily consist of conservation of mass and momentum which can be expressed as

$$\rho \frac{D\mathbf{v}}{Dt} = \nabla \cdot \boldsymbol{\sigma} + \rho \mathbf{g}, \ \frac{D\rho}{Dt} = -\rho \nabla \cdot \mathbf{v}, \ \mathbf{x} \in \Omega^t \tag{3.3}$$

where ρ is the Eulerian mass density, $\mathbf{v}$ is the Eulerian material velocity, $\boldsymbol{\sigma}$ is the Cauchy stress and $\mathbf{g}$ is gravitational acceleration. Boundary conditions for these equations are associated with a free surface for solid material, surface tension for liquids and/or prescribed velocity conditions. We use $\partial\Omega_N^t$ to denote the portion of the time t boundary subject to free surface or surface tension conditions and $\partial\Omega_D^t$ to denote the portion of the boundary with Dirichlet velocity boundary conditions. Free surface conditions and surface tension boundary conditions are expressed as

$$\boldsymbol{\sigma}\mathbf{n} = \mathbf{t}, \mathbf{x} \in \partial\Omega_N^t \tag{3.4}$$

where $\mathbf{t} = \mathbf{0}$ for free surface conditions and $\mathbf{t} = k^\sigma \kappa \mathbf{n} + \nabla^S k^\sigma$ from Equation (3.2) for surface tension conditions. Velocity boundary conditions may be written as

$$\mathbf{v} \cdot \mathbf{n} = v_{\text{bc}}^n, \mathbf{x} \in \partial\Omega_D^t. \tag{3.5}$$

3.1.2.1 Constitutive Models

Each material point is either a solid or liquid depending on the thermomechanical evolution. For liquids, the Cauchy stress $\boldsymbol{\sigma}$ is defined in terms of pressure and viscous stress:

$$\boldsymbol{\sigma} = -p\mathbf{I} + \mu\left(\frac{\partial \mathbf{v}}{\partial \mathbf{x}} + \frac{\partial \mathbf{v}}{\partial \mathbf{x}}^T\right),\ p = -\frac{\partial \psi^p}{\partial J},$$

with $\psi^p(J) = \frac{\lambda^l}{2}(J-1)^2$. Here λ^l is the bulk modulus of the liquid and μ^l is its viscosity. For solids, the Cauchy stress is defined in terms of a hyperelastic potential energy density ψ^s as

$$\boldsymbol{\sigma} = \frac{1}{j}\frac{\partial \psi^s}{\partial \mathbf{F}}\mathbf{f}^T$$

where $\mathbf{f}(\mathbf{x},t) = \mathbf{F}(\boldsymbol{\phi}^{-1}(\mathbf{x},t),t)$ and $j(\mathbf{x},t) = J(\boldsymbol{\phi}^{-1}(\mathbf{x},t),t)$ are the Eulerian deformation gradient and its determinant, respectively. We use the fixed-corotated constitutive model from [SHS12] for ψ^h. This model is defined in terms of the polar SVD [ITF04] of the deformation gradient $\mathbf{F} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T$ with

$$\psi^h(\mathbf{F}) = \mu^h \sum_{\alpha=0}^{d-1}(\sigma_\alpha - 1)^2 + \frac{\lambda^h}{2}(J-1)^2,$$

where the σ_α are the diagonal entries of $\boldsymbol{\Sigma}$ and μ^h, λ^h are the hyperelastic Lamé coefficients.

3.1.3 Conservation of energy

We assume the internal energy of our materials consists of potential energy associated with surface tension, liquid pressure and hyperelasticity and thermal energy associated with material temperature. Conservation of energy together with thermodynamic considerations requires convection/diffusion of the material temperature [GS08] subject to Robin boundary conditions associated with convective heating by ambient material:

$$\begin{aligned} \rho c_p \frac{DT}{Dt} &= K\nabla^2 T + H \\ K\nabla T \cdot \mathbf{n} &= -h(T - \bar{T}) + b. \end{aligned} \tag{3.6}$$

Here c_p is specific heat capacity, T is temperature, K is thermal conductivity, H is a source function, $\bar{T}$ is the temperature of ambient material, and $\mathbf{n}$ is the surface boundary normal. h controls the rate of convective heating to the ambient temperature, and b represents the rate of boundary heating independent of the ambient material temperature $\bar{T}$.

The total potential energy Ψ in our material is as in [HGM20], however we include the spatial variation of the surface energy density in Equation (3.1) and the hyperelastic potential for solid regions:

$$\Psi(\boldsymbol{\phi}(\cdot,t)) = \Psi^{\sigma}(\boldsymbol{\phi}(\cdot,t)) + \Psi^{l}(\boldsymbol{\phi}(\cdot,t)) + \Psi^{h}(\boldsymbol{\phi}(\cdot,t)) + \Psi^{g}(\boldsymbol{\phi}(\cdot,t)).$$

Here Ψ^{σ} is the potential from surface tension, Ψ^{l} is the potential from liquid pressure, Ψ^{h} is the potential from solid hyperelasticity, and Ψ^{g} is the potential from gravity. As in typical MPM discretizations, our approach is designed in terms of these energies:

$$\Psi^{\sigma}(\boldsymbol{\phi}(\cdot,t)) = \int_{\partial\Omega^t} k^{\sigma}(\mathbf{x},t)ds(\mathbf{x}), \quad \Psi^{g}(\boldsymbol{\phi}(\cdot,t) = \int_{\Omega^0} R\mathbf{g}\cdot\boldsymbol{\phi}Jd\mathbf{X},$$
$$\Psi^{l}(\boldsymbol{\phi}(\cdot,t)) = \int_{\Omega^0} \frac{\lambda^l}{2}\left(J-1\right)^2 d\mathbf{X}, \quad \Psi^{h}(\boldsymbol{\phi}(\cdot,t)) = \int_{\Omega^0} \psi^{h}(\mathbf{F})d\mathbf{X}.$$

Here R is the pullback (see Section 3.1.1) of the mass density ρ. Note that in the expression for the surface tension potential it is useful to change variables using the updated Lagrangian view as in [HGM20]:

$$\Psi^{\sigma}(\boldsymbol{\phi}(\cdot,t)) = \int_{\partial\Omega^t} k^{\sigma}(\mathbf{x},t)ds(\mathbf{x}) = \int_{\partial\Omega^s} k^{\sigma}(\hat{\boldsymbol{\phi}}(\tilde{\mathbf{x}},s,t),t)|\hat{J}\hat{\mathbf{F}}^{-T}\tilde{\mathbf{n}}|ds(\tilde{\mathbf{x}}). \tag{3.7}$$

Here $\tilde{\mathbf{n}}$ is the outward unit normal at a point on the boundary of Ω^s and the expression $ds(\mathbf{x}) = |\hat{J}\hat{\mathbf{F}}^{-T}\tilde{\mathbf{n}}|ds(\tilde{\mathbf{x}})$ arises by a change of variables from an integral over $\partial\Omega^t$ to one over $\partial\Omega^s$. Notably, the spatial variation in k^{σ} is a natural extension of the Hyde et al. [HGM20] approach.

3.1.4 Variational Form of Momentum Balance

The strong form of momentum balance in Equation (3.3), together with the traction (Equation (3.4)) and Dirichlet velocity boundary conditions (Equation (3.5)), is equivalent to a

variational form that is useful when discretizing our governing equations using MPM. To derive the variational form, we take the dot product of Equation (3.3) with an arbitrary function $\mathbf{w} : \Omega^t \to \mathbb{R}^d$ satisfying $\mathbf{w} \cdot \mathbf{n} = 0$ for $\mathbf{x} \in \partial\Omega^t_D$ and integrate over the domain Ω^t, applying integration by parts where appropriate. Requiring that the Dirichlet velocity conditions in Equation (3.4) hold together with the following integral equations for all functions $\mathbf{w}$ is equivalent to the strong form, assuming sufficient solution regularity:

$$\int_{\Omega^t} \rho \frac{Dv_\alpha}{Dt} w_\alpha d\mathbf{x} = -\frac{d}{d\epsilon} PE(0; \mathbf{w}) - \mu^l \int_{\Omega^t} \epsilon^v_{\alpha\beta} \epsilon^w_{\alpha\beta} d\mathbf{x}. \tag{3.8}$$

Here $\epsilon^w = \frac{1}{2}\left(\frac{\partial w_\alpha}{\partial x_\beta} + \frac{\partial w_\beta}{\partial x_\alpha}\right)$ and $\epsilon^v = \frac{1}{2}\left(\frac{\partial v_\alpha}{\partial x_\beta} + \frac{\partial v_\beta}{\partial x_\alpha}\right)$ and

$$\mathrm{PE}(\epsilon; \mathbf{w}) = \Psi(\boldsymbol{\phi}(\cdot, t) + \epsilon \mathbf{W}).$$

Here $\mathbf{W}$ is the pullback of $\mathbf{w}$ (see Section 3.1.1). Note that this notation is rather subtle for the surface tension potential energy. For clarification,

$$\Psi^\sigma(\boldsymbol{\phi}(\cdot, t) + \epsilon \mathbf{W}) = \int_{\partial\Omega^s} k^\sigma(\hat{\boldsymbol{\phi}} + \epsilon \hat{\mathbf{w}}) |\hat{J}_{\epsilon,\hat{\mathbf{w}}} \hat{\mathbf{F}}^{-T}_{\epsilon,\hat{\mathbf{w}}} \tilde{\mathbf{n}}| ds(\tilde{\mathbf{x}}),$$

where $\hat{\mathbf{w}}(\tilde{\mathbf{x}}, s, t) = \mathbf{w}(\hat{\boldsymbol{\phi}}(\tilde{\mathbf{x}}, s, t))$ is the updated Lagrangian pullback of $\mathbf{w}$, $\hat{\mathbf{F}}_{\epsilon,\hat{\mathbf{w}}} = \frac{\partial \hat{\boldsymbol{\phi}} + \epsilon \hat{\mathbf{w}}}{\partial \tilde{\mathbf{x}}}$ is the deformation gradient of the mapping $\hat{\boldsymbol{\phi}} + \epsilon \hat{\mathbf{w}}$ and $\hat{J}_{\epsilon,\hat{\mathbf{w}}} = \det\left(\hat{\mathbf{F}}_{\epsilon,\hat{\mathbf{w}}}\right)$ is its determinant. Lastly, for discretization purposes, in practice we change variables in the viscosity term

$$\mu^l \int_{\Omega^t} \epsilon^v_{\alpha\beta} \epsilon^w_{\alpha\beta} d\mathbf{x} = \mu^l \int_{\Omega^s} \hat{\epsilon}^v_{\alpha\beta} \hat{\epsilon}^w_{\alpha\beta} \hat{J} d\tilde{\mathbf{x}}. \tag{3.9}$$

We can similarly derive a variational form of the temperature evolution in Equation (3.6) by requiring

$$\begin{aligned} \int_{\Omega^t} \rho c_p \frac{DT}{Dt} q d\mathbf{x} = &- \int_{\partial\Omega^t} qhT d\mathbf{S}(\mathbf{x}) + \int_{\partial\Omega^t} q\left(h\bar{T} + b\right) d\mathbf{S}(\mathbf{x}) \\ &+ \int_{\Omega^t} Hq d\mathbf{x} - \int_{\Omega^t} \nabla q \cdot K \nabla T d\mathbf{x} \end{aligned} \tag{3.10}$$

for all functions $q : \Omega^t \to \mathbb{R}$.

3.1.5 Thermomechanical Material Dependence and Phase Change

The thermomechanical material dependence is modeled by allowing the surface tension coefficient k^σ, the liquid bulk modulus λ^l, the liquid viscosity μ^l and the hyperelastic Lamé coefficients μ^h, λ^h to vary with temperature. When T exceeds a user-specified melting point T_{melt}, the solid phase is changed to liquid and the deformation gradient determinant J is set to 1. Similarly, if the liquid temperature drops below T_{melt}, the phase is updated to be hyperelastic solid, and the deformation gradient $\mathbf{F}$ is set to the identity matrix. In practice, resetting the deformation gradient and its determinant helps to prevent nonphysical popping when the material changes from solid to liquid and vice versa. We remark that incorporating a more sophisticated phase change model, such as a latent heat buffer, is potentially useful in future work [SSJ14].

3.1.6 Contact Angle

The contact angle between a liquid, a solid boundary and the ambient air is governed by the Young equation [You05]. This expression relates the resting angle θ (measured through the liquid) of a liquid in contact with a solid surface to the surface tension coefficients between the liquid, solid and air phases:

$$k^\sigma_{SG} = k^\sigma_{SL} + k^\sigma_{LG}\cos(\theta). \tag{3.11}$$

The surface tension coefficients are between the solid and gas phases, solid and liquid phases, and liquid and gas phases, respectively. As in [CWS13], we assume k^σ_{SG} is negligible since we are using a free surface assumption and do not explicitly model the air. Under this assumption, the solid-liquid contact angle is determined by the surface tension ratio $-k^\sigma_{SL}/k^\sigma_{LG}$. We note that, while one would expect surface tension coefficients/energies to be positive, this ratio can be negative under the assumption of zero solid-gas surface tension. Furthermore, we note that utilizing this expression requires piecewise constant surface tension coefficients where the variation along the liquid boundary is based on which portion is in contact with

the air and which is in contact with the solid. The distinct surface tension coefficients on different interfaces provide controllability of the spreading behavior of the liquid on the solid surface.

3.2 Discretization

As in [HGM20], we use MPM [SCS94] and APIC [JSS15] to discretize the governing equations. The domain Ω^{t^n} at time t^n is sampled using material points $\mathbf{x}_p^n$. These points also store approximations of the deformation gradient determinant J_p^n, constant velocity $\mathbf{v}_p^n$, affine velocity $\mathbf{A}_p^n$, volume V_p^0, mass $m_p = \rho(\mathbf{x}_p^0, t^0)V_p^0$, temperature T_p^n, and temperature gradient ∇T_p^n. We also make use of a uniform background grid with spacing Δx when discretizing momentum updates. To advance our state to time t^{n+1}, we use the following steps:

1. Resample particle boundary for surface tension and Robin boundary temperature conditions.
2. P2G: Conservative transfer of momentum and temperature from particles to grid.
3. Update of grid momentum and temperature.
4. G2P: Conservative transfer of momentum and temperature from grid to particles.

3.2.1 Conservative Surface Particle Resampling

The integrals associated with the surface tension energy in Equation (3.7) and the Robin temperature condition in Equation (3.10) are done over the boundary of the domain. We follow Hyde et al. [HGM20] and introduce special particles to cover the boundary in order to serve as quadrature points for these integrals. As in [HGM20] these particles are temporary and are removed at the end of the time step. However, while Hyde et al. [HGM20] used massless surface particles, we design a novel conservative mass and momentum resampling

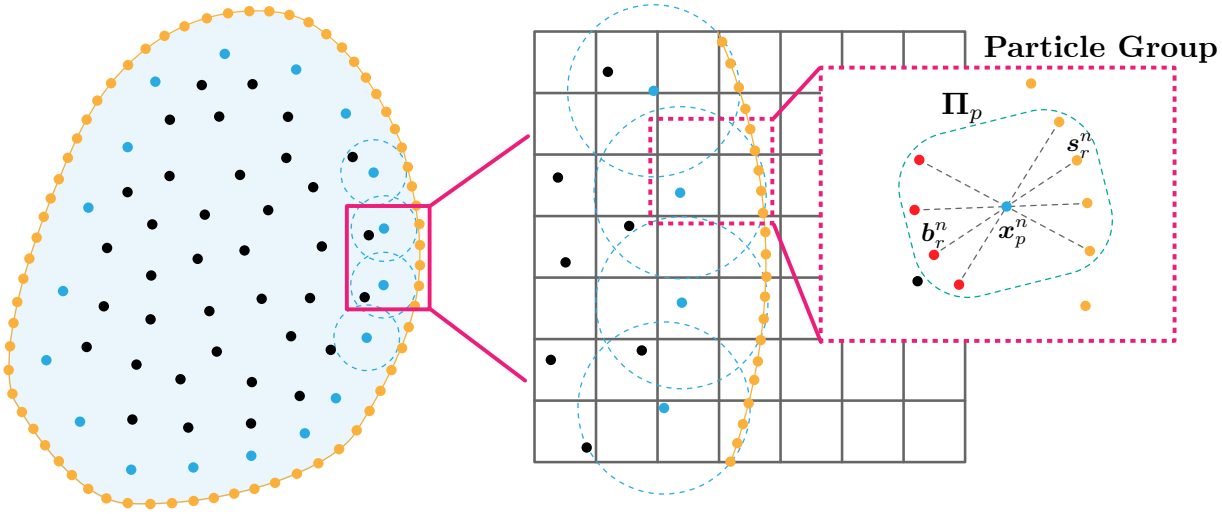


Figure 3.2: A portion of an MPM fluid in the simulation domain. Surface particles (yellow) are sampled on faces of the zero isocontour of the level set formed by unioning spherical level sets around each MPM particle. Each surface particle generates an associated balance particle (red) such that the closest MPM particle (blue) to a boundary particle lies on the midpoint of a line segment between the surface particle and balance particle. A single blue particle at $\boldsymbol{x}_p$ may be paired with multiple surface particles and balance particles, and they are considered to be in a particle group $\mathbf{\Pi}_p$. MPM particles that are not associated with any surface tension particles are marked as black.

for surface particles. Massless particles easily allow for momentum conserving transfers from particles to the grid and vice versa; however, they can lead to loss of conservation in the grid momentum update step. This occurs when there is a grid cell containing only massless particles. In this case, there are grid nodes with no mass that receive surface tension forces. These force components are then effectively thrown out since only grid nodes with mass will affect the end of time step particle momentum state (see Section 3.3.1).

We resolve this issue by assigning mass to each of the surface particles. However, to conserve total mass, some mass must be subtracted from interior MPM particles. Furthermore, changing the mass of existing particles also changes their momentum, which may lead to

violation of conservation. In order to conserve mass, linear momentum and angular momentum, we introduce a new particle for each surface particle. We call these balance particles, and like surface particles they are temporary and will be removed at the end of the time step. We show that the introduction of these balance particles naturally allows for conservation both when they are created at the beginning of the time step and when they are removed at the end of the time step.

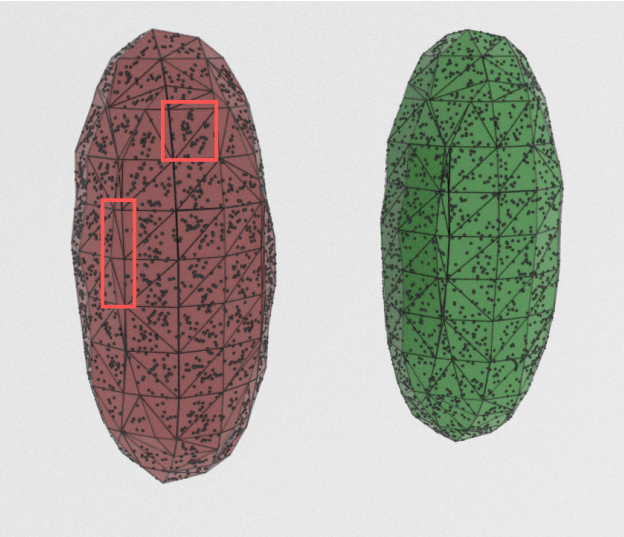


Figure 3.3: Isocontour and sampled boundary particles for an ellipsoid. *(Left)* Using the method of Hyde et al. [HGM20]. Note how low-quality triangles are undersampled and how sample points often clump near triangle centers. *(Right)* The present method, which does not suffer from similar issues.

3.2.1.1 Surface Particle Sampling

We first introduce surface particles using the approach in [HGM20]. A level set enclosing the interior MPM particles is defined as the union of spherical level sets defined around each interior MPM particle. Unlike Hyde et al. [HGM20], we do not smooth or shift the unioned level set. We compute the zero isocontour of the level set using marching cubes [Che95] and randomly sample surface particles along this explicit representation. In [HGM20], three-

dimensional boundaries were sampled using a number of sample points proportional to the surface area of each triangle. Sample points were computed using uniform random barycentric weights, which leads to a non-uniform distribution of points in each triangle. Instead, we employ a strategy of per-triangle Monte Carlo sampling using a robust Poisson distribution, as described in Figure 1 of [CCS12] (not their blue noise algorithm); uniform triangle sample points are generated as in [OFC02]. We found that this gave better coverage of the boundary without generating particles in a biased, mesh-dependent fashion (see Figure 3.3). We note that radii for the particle level sets are taken to be $0.73\Delta x$ (slightly larger than $\frac{\sqrt{2}}{2}\Delta x$) in 2D and $0.867\Delta x$ (slightly larger than $\frac{\sqrt{3}}{2}\Delta x$) in 3D, where Δx is the MPM grid spacing. This guarantees that even a single particle in isolation will always generate a level set zero isocontour that intersects the grid and will therefore always generate boundary sample points. Note also that as in [HGM20], we use the explicit marching cubes mesh of the zero isocontour to easily and accurately generate samples of area weighted normals $d\mathbf{A}_r^n$ where $\sum |d\mathbf{A}_r^n| \approx \int_{\partial\Omega^{t^n}} d\mathbf{s}$ are chosen with direction from the triangle normal and magnitude based on the number of samples in a given triangle and the triangle area.

3.2.1.2 Balance Particle Sampling

For each surface particle $\mathbf{s}_r^n$, we additionally generate a balance particle $\mathbf{b}_r^n$. First, we compute the closest interior MPM particle for each surface particle $\mathbf{x}_{p(\mathbf{s}_r^n)}^n$. Then we introduce the corresponding balance particle as

$$\mathbf{b}_r^n = \mathbf{s}_r^n + 2\left(\mathbf{x}_{p(\mathbf{s}_r^n)}^n - \mathbf{s}_r^n\right). \tag{3.12}$$

3.2.1.3 Mass and Momentum Splitting

After introducing the surface $\mathbf{s}_r^n$ and balance $\mathbf{b}_r^n$ particles, we assign them mass and momentum (see Figure 3.4). To achieve this in a conservative manner, we first partition the surface

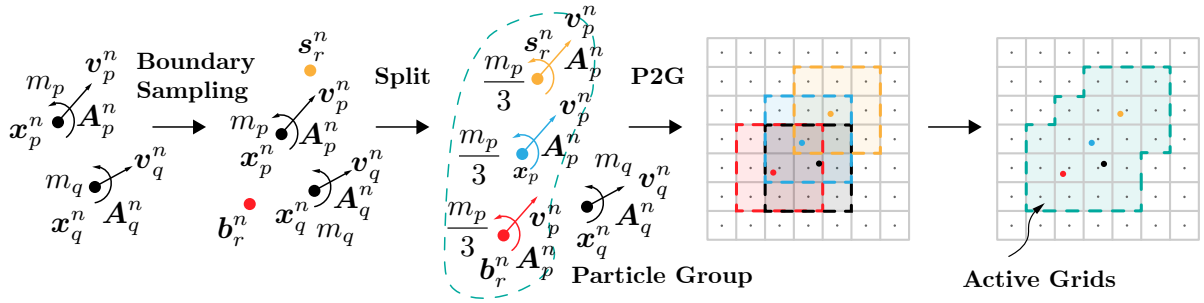


Figure 3.4: Splitting. After surface particles (yellow) are created, the mass and momentum of the interior MPM particles (blue) that are closest to the surface particles are immediately distributed. Particles in each particle group are assigned equal mass. MPM particles (black) that are not paired with any surface particles remain intact for the splitting process. Surface particles (yellow) and balance particles (red) are assigned the same linear velocity and affine velocity of the original particle (blue).

particles into particle groups Π_p defined as the set of surface particle indices r such that $\mathbf{x}_p^n$ is the closest interior MPM particle to $\mathbf{s}_r^n$ (see Figure 3.2). We assign the mass m_p of the particle $\mathbf{x}_p^n$ to the collection of $\mathbf{x}_p^n$, $\mathbf{s}_r^n$ and $\mathbf{b}_r^n$ for $r \in \Pi_p$ uniformly by defining a mass of $\tilde{m}_p = \frac{m_p}{2|\Pi_p|+1}$ to each surface and balance point as well as to $\mathbf{x}_p^n$. Here $|\Pi_p|$ is the number of elements in the set. This operation is effectively a split of the original particle $\mathbf{x}_p^n$ with mass m_p into a new collection of particles $\mathbf{x}_p^n, \mathbf{s}_r^n, \mathbf{b}_r^n,\ r \in \Pi_p$ with masses $\tilde{m}_p$. This split trivially conserves the mass. Importantly, by construction of the balance particles (Equation (3.12)) we ensure that the center of mass of the collection is equal to the original particle $\mathbf{x}_p^n$:

$$\frac{1}{m_p}\left(\tilde{m}_p \mathbf{x}_p^n + \sum_{r\in\Pi_p} \tilde{m}_p \mathbf{s}_r^n + \tilde{m}_p \mathbf{b}_r^n\right) = \mathbf{x}_p^n. \tag{3.13}$$

With this particle distribution, conservation of linear and angular momentum can be achieved by simply assigning each new particle in the collection the velocity $\mathbf{v}_p^n$ and affine velocity $\mathbf{A}_p^n$ of the original particle $\mathbf{x}_p^n$. We note that the conservation of the center of mass (Equation (3.13)) is essential for this simple constant velocity split to conserve linear and angular

momentum (see Appendix A).

3.2.2 Transfer: P2G

After the addition of the surface and balance particles, we transfer mass and momentum to the grid in the standard APIC [JSS15] way using their conservatively remapped mass and velocity state

$$
\begin{aligned}
m_{\mathbf{i}}^n &= \sum_p \tilde{m}_p \left(N_{\mathbf{i}}(\mathbf{x}_p^n) + \sum_{r \in \Pi_p} N_{\mathbf{i}}(\mathbf{s}_r^n) + N_{\mathbf{i}}(\mathbf{b}_r^n) \right), \\
m_{\mathbf{i}}^n \mathbf{v}_{\mathbf{i}}^n &= \sum_p \tilde{m}_p N_{\mathbf{i}}(\mathbf{x}_p^n) \left(\mathbf{v}_p^n + \mathbf{A}_p^n (\mathbf{x}_{\mathbf{i}} - \mathbf{x}_p^n) \right) \\
&\quad + \sum_p \tilde{m}_p \sum_{r \in \Pi_p} N_{\mathbf{i}}(\mathbf{s}_r^n) \left(\mathbf{v}_p^n + \mathbf{A}_p^n (\mathbf{x}_{\mathbf{i}} - \mathbf{s}_r^n) \right) \\
&\quad + \sum_p \tilde{m}_p \sum_{r \in \Pi_p} N_{\mathbf{i}}(\mathbf{b}_r^n) \left(\mathbf{v}_p^n + \mathbf{A}_p^n (\mathbf{x}_{\mathbf{i}} - \mathbf{b}_r^n) \right).
\end{aligned}
$$

Here $\mathbf{N}_{\mathbf{i}}(\mathbf{x}) = \mathbf{N}(\mathbf{x} - \mathbf{x}_{\mathbf{i}})$ are quadratic B-splines defined over the uniform grid with $\mathbf{x}_{\mathbf{i}}$ living at cell centers [SSC13]. Note that for interior MPM particles far enough from the boundary that $\Pi_p = \varnothing$. This reduces to the standard APIC [JSS15] splat since $\tilde{m}_p = m_p^n$. We also transfer temperature from particles to grid using

$$
T_{\mathbf{i}}^n = \sum_p m_p N_{\mathbf{i}}(\mathbf{x}_p^n)(T_p^n + (x_{\mathbf{i}\alpha} - x_{p\alpha}^n) \nabla T_{p\alpha}^n).
$$

Note that for the temperature transfer, we only use surface particles to properly apply the thermal boundary conditions, and we do not use these particles to transfer mass-weighted temperature to the grid.

3.2.3 Grid Momentum and Temperature Update

We discretize the governing equations in the standard MPM manner by using the particles as quadrature points in the variational forms. The interior MPM particles $\mathbf{x}_p^n$ are used

for volume integrals and the surface particles $\mathbf{s}_r^n$ are used for surface integrals. By choosing $s = t^n$, $t = t^{n+1}$ and by using grid discretized versions of $\hat{\mathbf{w}}(\tilde{\mathbf{x}}) = \sum_{\mathbf{j}} \mathbf{w}_{\mathbf{j}} N_{\mathbf{j}}(\tilde{\mathbf{x}})$, $\hat{\mathbf{v}}(\tilde{\mathbf{x}}, t^n, t^{n+1}) = \sum_{\mathbf{i}} \hat{\mathbf{v}}_{\mathbf{i}}^{n+1} N_{\mathbf{i}}(\tilde{\mathbf{x}})$, $\hat{q}(\tilde{\mathbf{x}}) = \sum_{\mathbf{j}} q_{\mathbf{j}} N_{\mathbf{j}}(\tilde{\mathbf{x}})$ and $\hat{T}(\tilde{\mathbf{x}}) = \sum_{\mathbf{i}} \hat{T}_{\mathbf{i}} N_{\mathbf{i}}(\tilde{\mathbf{x}})$.

3.2.3.1 Momentum Update

As in [HGM20], the grid momentum update is derived from Equation (3.8):

$$m_{\mathbf{i}}^n \frac{\hat{\mathbf{v}}_{\mathbf{i}}^{n+1} - \mathbf{v}_{\mathbf{i}}^n}{\Delta t} = \mathbf{f}_{\mathbf{i}}(\mathbf{x} + \Delta t \hat{\mathbf{q}}) + m_{\mathbf{i}}^n \mathbf{g}, \tag{3.14}$$

$$\mathbf{f}_{\mathbf{i}}(\hat{\mathbf{x}}) = -\frac{\partial e}{\partial \hat{\mathbf{x}}_{\mathbf{i}}}(\hat{\mathbf{x}}) - \mu^l \sum_p \boldsymbol{\epsilon}^v(\hat{\mathbf{x}}; \mathbf{x}_p^n) \left(\frac{\partial N_{\mathbf{i}}}{\partial \mathbf{x}}(\mathbf{x}_p^n) \right)^T V_p^n, \tag{3.15}$$

where $\mathbf{f}_{\mathbf{i}}$ is the force on grid node $\mathbf{i}$ from potential energy and viscosity, $\epsilon^v(\hat{\mathbf{x}}; \mathbf{x}_p^n) = \frac{1}{2}\left(\sum_{\mathbf{j}} \hat{\mathbf{x}}_{\mathbf{j}} \frac{\partial N_{\mathbf{j}}}{\partial \mathbf{x}}(\mathbf{x}_p^n) + \left(\hat{\mathbf{x}}_{\mathbf{j}} \frac{\partial N_{\mathbf{j}}}{\partial \mathbf{x}}(\mathbf{x}_p^n) \right)^T \right)$ is the strain rate at $\mathbf{x}_p^n$, $\mathbf{g}$ is gravity, and $\hat{\mathbf{q}}$ is either 0 (for explicit time integration) or $\hat{\mathbf{v}}^{n+1}$ (for backward Euler time integration). $\mathbf{x}$ represents the vector of all unmoved grid node positions $\mathbf{x}_{\mathbf{i}}$. We use $e(\mathbf{y})$ to denote the discrete potential energy Ψ where MPM and surface particles are used as quadrature points:

$$\begin{aligned} e(\mathbf{y}) = &\sum_p \left(\psi^h(\mathbf{F}_p(\hat{\mathbf{y}})) + \frac{\lambda^l}{2}(J_p(\hat{\mathbf{y}}) - 1)^2 \right) V_p^0 \\ &+ \sum_r k^\sigma(\mathbf{s}_r^n) |\hat{J}_r(\hat{\mathbf{y}}) \hat{\mathbf{F}}_r^{-T}(\hat{\mathbf{y}}) d\mathbf{A}_r^n|, \end{aligned}$$

where, as in [SSC13], $\mathbf{F}_p(\hat{\mathbf{y}}) = \sum_{\mathbf{i}} \mathbf{y}_{\mathbf{i}} \frac{\partial N_{\mathbf{i}}}{\partial \mathbf{x}}(\mathbf{x}_p^n) \mathbf{F}_p^n$ and as in [HGM20], $J_p(\hat{\mathbf{y}}) = \left(1 - d + y_\alpha \frac{\partial N_{\mathbf{i}}}{\partial x_\alpha}(\mathbf{x}_p^n)\right) J_p^n$ and $\hat{\mathbf{F}}_p(\mathbf{y}) = \sum_{\mathbf{i}} \mathbf{y}_{\mathbf{i}} \frac{\partial N_{\mathbf{i}}}{\partial \mathbf{x}}(\mathbf{x}_p^n)$. With these conventions, the α component of the energy-based force on grid node $\mathbf{i}$ is of the form

$$\begin{aligned} -\frac{\partial e}{\partial x_{\mathbf{i}\alpha}}(\mathbf{y}) = &- \sum_p \frac{\partial \psi^h}{\partial F_{\alpha\delta}}(\mathbf{F}_p(\hat{\mathbf{y}})) F_{p\gamma\delta}^n \frac{\partial N_{\mathbf{i}}}{\partial x_\gamma}(\mathbf{x}_p^n) V_p^0 \\ &- \sum_p \lambda^l (J_p(\mathbf{y}) - 1) \frac{\partial N_{\mathbf{i}}}{\partial x_\alpha}(\mathbf{x}_p^n) J_p^n V_p^0 \\ &- \sum_r k^\sigma(\mathbf{s}_r^n) \frac{\partial |\det(\hat{\mathbf{F}}_r) \hat{\mathbf{F}}_r^{-T} d\mathbf{A}_r^n|}{\partial \hat{F}_{\alpha\delta}} (\hat{\mathbf{F}}_r(\hat{\mathbf{y}})) \frac{\partial N_{\mathbf{i}}}{\partial x_\delta}(\mathbf{x}_p^n). \end{aligned} \tag{3.16}$$

We note that the viscous contribution to the force in Equation (3.15) is the same as in [RGJ15]. We would expect V_p^{n+1} in this term when deriving from Equation (3.9), however we approximate it as V_p^n. This is advantageous since it makes the term linear; and since $\hat{J}_p(\hat{\mathbf{x}}) \approx 1$ from the liquid pressure and hyperelastic stress, it is not a poor approximation. Lastly, we note that the surface tension coefficient $k^\sigma(\mathbf{s}_r^n)$ will typically get its spatial dependence from composition with a function of temperature $k^\sigma(\mathbf{s}_r^n) = \tilde{k}^\sigma(\hat{T}(\mathbf{s}_r^n)) = \tilde{k}^\sigma(\mathbf{T}_r^{s,n})$.

In the case of implicit time stepping with backward Euler ($\hat{\mathbf{q}} = \hat{\mathbf{v}}^{n+1}$), we use Newton's method to solve the nonlinear systems of equations. This requires linearization of the grid forces associated with potential energy in Equation (3.16). We refer the reader to [SSC13] and [HGM20] for the expressions for these terms, as well as the definiteness fix used for surface tension contributions.

3.2.3.2 Temperature Update

We discretize Equation (3.10) in a similar manner which results in the following equations for the grid temperatures $T_\mathbf{i}$:

$$\begin{aligned} c_p m_\mathbf{i} \frac{\hat{T}_\mathbf{i}^{n+1} - T_\mathbf{i}^n}{\Delta t} = &- \sum_p K \frac{\partial N_\mathbf{i}}{\partial x_\alpha}(\mathbf{x}_p^n) \hat{T}_\mathbf{j}^{n+1} \frac{\partial N_\mathbf{j}}{\partial x_\alpha}(\mathbf{x}_p^n) V_p^n \\ &- \sum_r h N_\mathbf{i}(\mathbf{s}_r^n) \hat{T}_\mathbf{j}^{n+1} N_\mathbf{j}(\mathbf{s}_r^n) |d\mathbf{A}_r^n| \\ &+ \sum_r N_\mathbf{i}(\mathbf{s}_r^n) \left[h\bar{T}(\mathbf{s}_r^n) + b(\mathbf{s}_r^n) \right] |d\mathbf{A}_r^n| \\ &+ \sum_p h N_\mathbf{i}(\mathbf{x}_p^n) H(\mathbf{x}_p^n) V_p^n. \end{aligned}$$

Note that by using the surface particles $\mathbf{s}_r^n$ as quadrature points in the variational form, the Robin boundary condition can be discretized naturally with minimal modification to the Laplacian and time derivative terms. Also, note that the inclusion of this term modifies both the matrix and the right side in the linear system for $\hat{T}_\mathbf{i}^{n+1}$. We found that performing constant extrapolation of interior particle temperatures to the surface particles provided

better initial guesses for the linear solver.

3.2.4 Transfer: G2P

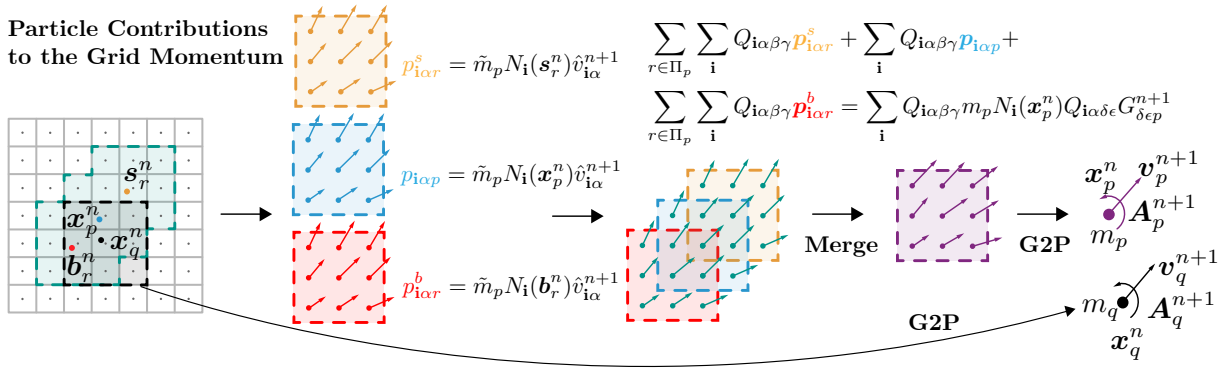


Figure 3.5: Merging. The merging process is a modified version of G2P. For the particles that are not associated with surface particles (black), a regular G2P is performed. Among each particle group, we calculate each particle's contribution to the grid momentum and the generalized affine moments of their summed momenta about their center of mass. Then, we restore the mass of the original particle associated with the group prior to the split and compute its generalized affine inertia tensor from its grid mass distribution. Using the affine inertia tensor of the original particle, we compute generalized velocity of the particle after the merging from the generalized moments of the group.

Once grid momentum and temperature have been updated, we transfer velocity and temperature back to the particles. For interior MPM particles with no associated surface or balance particles ($\Pi_p = \varnothing$), we transfer velocity, affine velocity and temperature from the grid to particles in the standard APIC [JSS15] way:

$$\mathbf{v}_p^{n+1} = \sum_{\mathbf{i}} N_{\mathbf{i}}(\mathbf{x}_p^n)\hat{\mathbf{v}}_{\mathbf{i}}^{n+1}, \quad \mathbf{A}_p^{n+1} = \frac{4}{\Delta x^2}\sum_{\mathbf{i}} N_{\mathbf{i}}(\mathbf{x}_p^n)\hat{\mathbf{v}}_{\mathbf{i}}^{n+1}(\mathbf{x}_{\mathbf{i}} - \mathbf{x}_p^n)^T.$$

For interior MPM particles that were split with a collection of surface and balance particles ($\Pi_p \neq \varnothing$), more care must be taken since surface and balance particles will be deleted

at the end of the time step. First, the particle is reassigned its initial mass m_p. Then we compute the portion of the grid momentum associated with each surface and balance particle associated with p as

$$\mathbf{p}^s_{\mathbf{i}r} = \tilde{m}_p N_{\mathbf{i}}(\mathbf{s}^n_r)\hat{\mathbf{v}}^{n+1}_{\mathbf{i}},\ \mathbf{p}^b_{\mathbf{i}r} = \tilde{m}_p N_{\mathbf{i}}(\mathbf{b}^n_r)\hat{\mathbf{v}}^{n+1}_{\mathbf{i}},\ r \in \Pi_p.$$

We then sum this with the split particle's share of the grid momentum to define the merged particle's share of the grid momentum

$$\mathbf{p}_{\mathbf{i}p} = \tilde{m}_p N_{\mathbf{i}}(\mathbf{x}^n_p)\hat{\mathbf{v}}^{n+1}_{\mathbf{i}} + \sum_{r\in\Pi_p} \mathbf{p}^s_{\mathbf{i}r} + \mathbf{p}^b_{\mathbf{i}r}.$$

Note that the $\mathbf{p}_{\mathbf{i}p}$ may be nonzero for more grid nodes than the particle would normally splat to (see Figure 3.5). We define the particle velocity from the total momentum by dividing by the mass $\mathbf{v}^{n+1}_p = \frac{1}{m_p}\sum_{\mathbf{i}} \mathbf{p}_{\mathbf{i}p}$. To define the affine particle velocity, we use a generalization of [FGG17] and first compute the generalized affine moments $t_{p\beta\gamma} = \sum_{\mathbf{i}} Q_{\mathbf{i}\alpha\beta\gamma} p_{\mathbf{i}p\alpha}$ of the momentum distribution $p_{\mathbf{i}p\alpha}$ where $Q_{\mathbf{i}\alpha\beta\gamma} = r_{\mathbf{i}p\gamma}\delta_{\alpha\beta}$ is the α component of the $\beta\gamma$ linear mode at grid node $\mathbf{i}$. Here $\mathbf{r}_{\mathbf{i}p} = \mathbf{x}_{\mathbf{i}} - \mathbf{x}^n_p$ is the displacement from the center of mass of the distribution to the grid node $\mathbf{x}_{\mathbf{i}}$. We note that these moments are the generalizations of angular momentum to affine motion, as was observed in [JSS15], however in our case we compute the moments from a potentially wider distribution of momenta $p_{\mathbf{i}p\alpha}$. Lastly, to conserve angular momenta (see Appendix A for details), we define the affine velocity by inverting the generalized affine inertia tensor $\sum_{\mathbf{i}} Q_{\mathbf{i}\alpha\gamma\delta} m_p N_{\mathbf{i}}(\mathbf{x}^n_p) Q_{\mathbf{i}\alpha\epsilon\tau}$ of the point $\mathbf{x}^n_p$ using its merged mass distribution $m_p N_{\mathbf{i}}(\mathbf{x}^n_p)$. However, as noted in [JSS15], the generalized inertia tensor $\frac{m_p\Delta x^2}{4}\mathbf{I}$ is constant diagonal when using quadratic B-splines for $N_{\mathbf{i}}(\mathbf{x}^n_p)$ and therefore the final affine velocity is $\mathbf{A}^{n+1}_p = \frac{4}{m_p\Delta x^2}\mathbf{t}_p$.

Temperature and temperature gradients are transferred in the same way whether or not a MPM particle was split or not:

$$T^{n+1}_p = \sum_{\mathbf{i}} \hat{T}^{n+1}_{\mathbf{i}} N(\mathbf{x}_{\mathbf{i}}),\ \nabla T^{n+1}_p = \sum_{\mathbf{i}} \hat{T}^{n+1}_{\mathbf{i}} \nabla N(\mathbf{x}_{\mathbf{i}}).$$

Table 3.1: Summary of the simulation parameters. Example-specific variable k^{σ}_{LG} can be found in the corresponding section. The unit for the thermal conductivity K is $\mathrm{W}/(\mathrm{m}\cdot\mathrm{K})$; the convective heat transfer coefficient h has a unit of $\mathrm{W}/(\mathrm{m}^2\cdot\mathrm{K})$; the unit of the boundary heating rate is W/m^2. The number of particles/cells used in each example is listed in Table 3.2.

Example	Δt [s]	CFL	Δx [m]	Bulk Modulus [Pa]	Density [kg/m^3]	k^{σ} [N/m]	k^{σ} Variation	Heat Transfer Coefficients
Conservation (2D, Explicit)	1×10^{-5}	0.6	1/63	4166.67	10	0.1	constant	N/A
Two Spheres (2D)	1×10^{-2} to 5×10^{-5}	0.6	1/127	833333.33	1	0.1	constant	$K=0.0025/h=10,0/b=0$
Rotating Heat Flux (2D)	1×10^{-2} to 5×10^{-5}	0.6	1/127	833333.33	1	0.1	constant	$K=0.01/h=0.1,0/h=5$
Droplet Impact	1×10^{-2} to 5×10^{-5}	0.3, 0.6	1/127	83333.33	10	$20,15,5,1,0.1,0.05$	constant	N/A
Droplets on Ramps	1×10^{-2} to 1×10^{-4}	0.6	1/63	16666.67	10	$1.0(k^{\sigma}_{SL}), k^{\sigma}_{LG}$ varies	piecewise-constant	N/A
Contact Angles	0.0333 to 1×10^{-4}	0.6	1/127	83333.33	1	$2.0(k^{\sigma}_{SL}), k^{\sigma}_{LG}$ varies	piecewise-constant	N/A
Soap Droplet in Water	1×10^{-2} to 5×10^{-5}	0.6	1/127	16666.67	1	0.5 (water), 0.01 (soap)	piecewise-constant	N/A
Wine Glass	1×10^{-2} to 1×10^{-5}	0.6	1/127	16666.67	1	$0.05(k^{\sigma}_{SL}), 0.015(k^{\sigma}_{LG})$	piecewise-constant	N/A
Candles	1×10^{-2} to 1×10^{-4}	0.6	1/127	833333.33	10	0.05, 0.1, 0.2, 0.4	constant	$K=0.1/h=0.5/b=50$
Candles (Letters)	1×10^{-2} to 1×10^{-4}	0.6	1/127	83333.33	10	0.05	constant	$K=0.1/h=2.5/b=100$
Lid-Driven Cavity (2D)	1×10^{-3}	0.6	1/63	416.67	10	1.0	temperature-dependent	$K=0.1/h=0/b=0$
Droplet with Marangoni Effect	1×10^{-2} to 5×10^{-5}	0.6	1/127	83333.33	1	$0.5\sim2.0$	temperature-dependent	$K=0.1/h=0.1/b=50$

3.3 Examples

3.3.1 Conservation

To demonstrate our method's ability to fully conserve momentum and center of mass, we simulate a two-dimensional ellipse that oscillates under zero gravity due to surface tension forces in a 1m $\times$ 1m domain. The ellipse has semiaxes of 0.3m and 0.1m. We compare these results to those obtained using the method of Hyde et al. [HGM20].

The total linear momentum $\sum_{\mathbf{i}} m_{\mathbf{i}} v_{\mathbf{i}}$ and the total angular momentum about the origin $\sum_{\mathbf{i}} x_{\mathbf{i}} \times m_{\mathbf{i}} v_{\mathbf{i}}$ are calculated on the grid. We also compute the center of mass velocity error $\frac{\sum_{\mathbf{i}} m_{\mathbf{i}} v_{\mathbf{i}}}{\sum_p m_p}$ and the center of mass drift relative to the domain size. As shown in Figure 3.6 and 3.7, the present technique perfectly conserves total linear momentum, total angular momentum, and the center of mass of the ellipse, unlike the approach of Hyde et al. [HGM20].

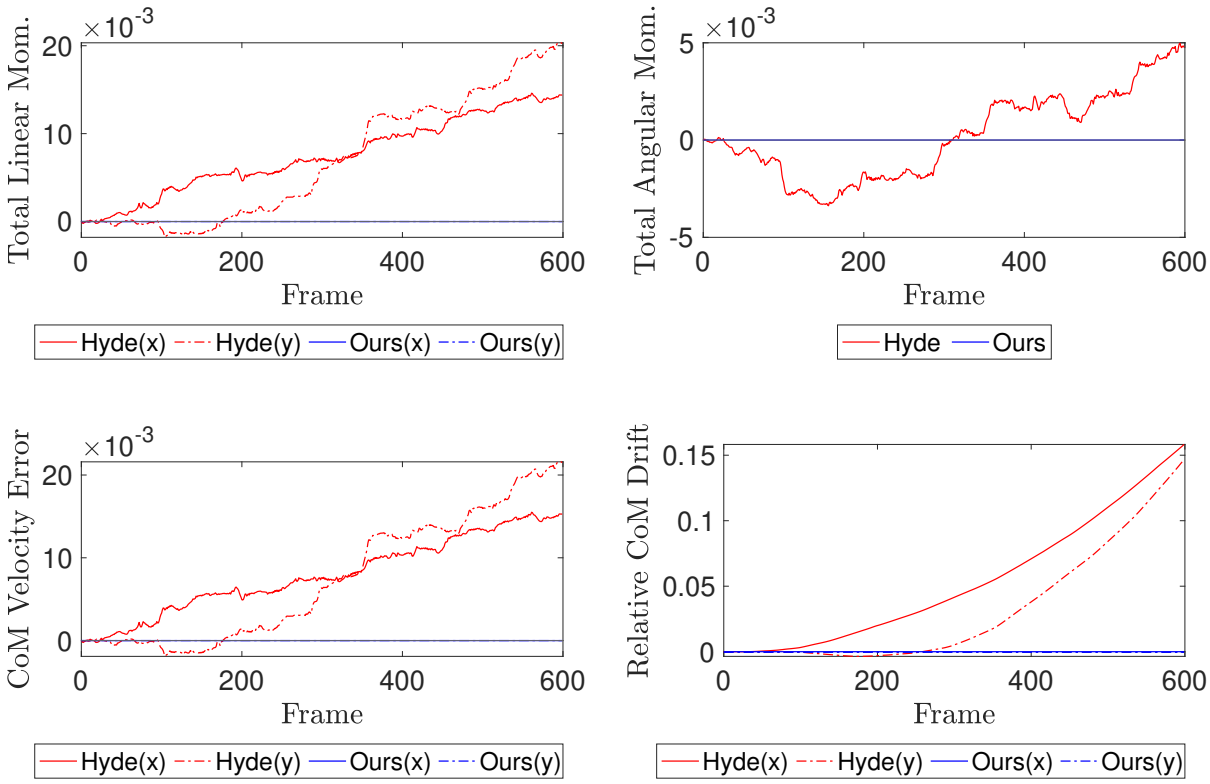
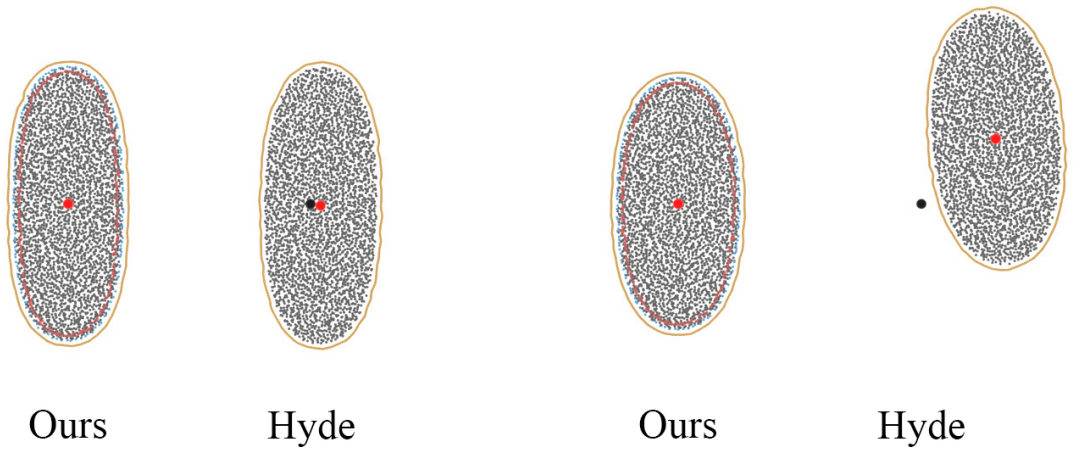


Figure 3.6: The present method (blue) conserves total mass, total linear and angular momentum, and center of mass, unlike the method of Hyde et al. [HGM20] (red).

3.3.2 Thermal Boundary Conditions

The Robin boundary condition allows for the realistic convective heat transfer between the object and the environment. While such effects may be approximated by a simplified Dirichlet or Neumann boundary condition, the Robin boundary condition simplifies the process, and it is crucial for simulating temperature-dependent effects, such as the resolidification of the liquid wax in Section 3.3.9.

To demonstrate this effect, we initialize two discs side-by-side with radii of 0.15m. The domain size is $1\text{m} \times 1\text{m}$, and the ambient temperature is 295K. The center of the left disc is at $(0.3, 0.5)$, and the temperature increases linearly from 265K at $x = 0.15$ to 295K at $x = 0.45$.

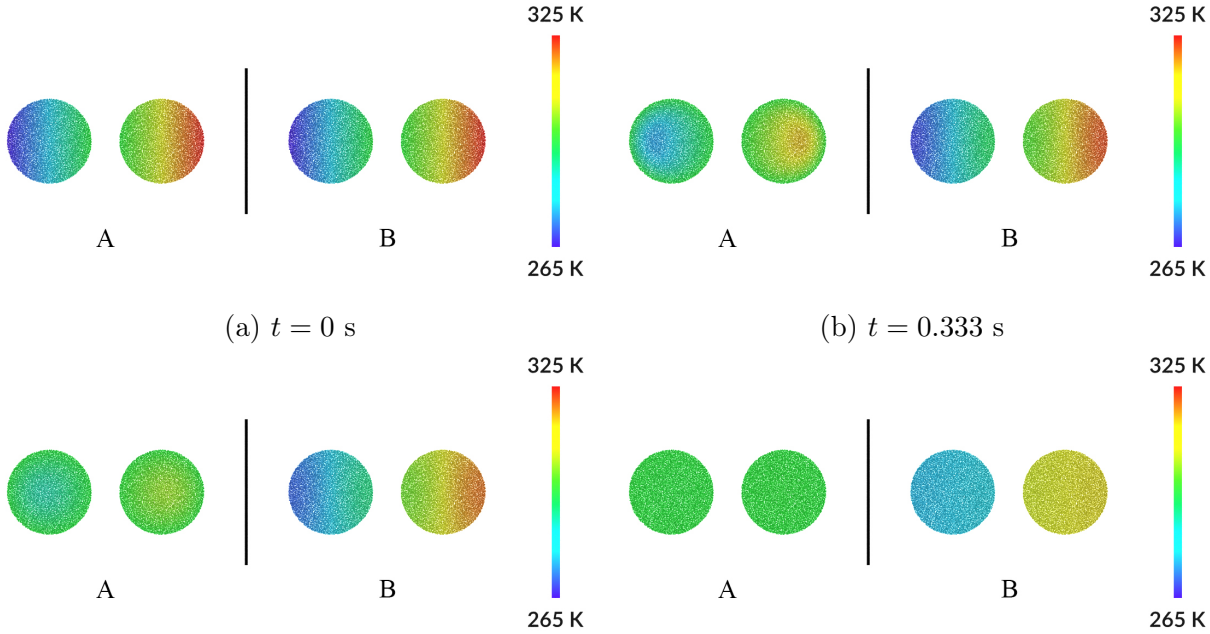


(a) $t = 6.53$ s (b) $t = 18.9$ s

Figure 3.7: An elliptical droplet oscillates under surface tension. The black dot indicates the initial location of the particles' center of mass, while the red dot is the position of the current center of mass. The drops in (a) are after 6 oscillation cycles, and the drops in (b) are after 18 cycles. The method of Hyde et al. [HGM20] does not conserve the momentum, so the center of mass drifts. Our method is conservative and preserves the center of mass even over a long period of time.

The center of the right one is at $(0.7, 0.5)$, and the temperature increases linearly from 295K at $x = 0.55$ to 325K at $x = 0.85$. The thermal conductivity for both simulation A and B is $0.0025\text{W}/(\text{m} \cdot \text{K})$. Simulation A has $h = 10\text{W}/(\text{m}^2 \cdot \text{K})$, while $h = 0$ for simulation B. As shown in Figure 3.8, the Robin boundary condition equilibrates the temperature of the discs in simulation A to the ambient temperature, while in simulation B, only the temperature in each disc reaches equilibrium.

Our method allows complex time-dependent boundary conditions to be applied. We simulate in Figure 3.9 a solid disc with a radius of 0.15m with a heat flux of $b = 5\text{W}/\text{m}^2$ applied on a section of its boundary. The disc is in a 1m $\times$ 1m domain with the ambient temperature of 295K. The location heat flux rotates about the center of the disc at 2π rad/s. Simulation A has the Robin boundary condition $h = 0.1\text{W}/(\text{m} \cdot \text{K})$ applied, so the region



(a) $t = 0$ s (b) $t = 0.333$ s

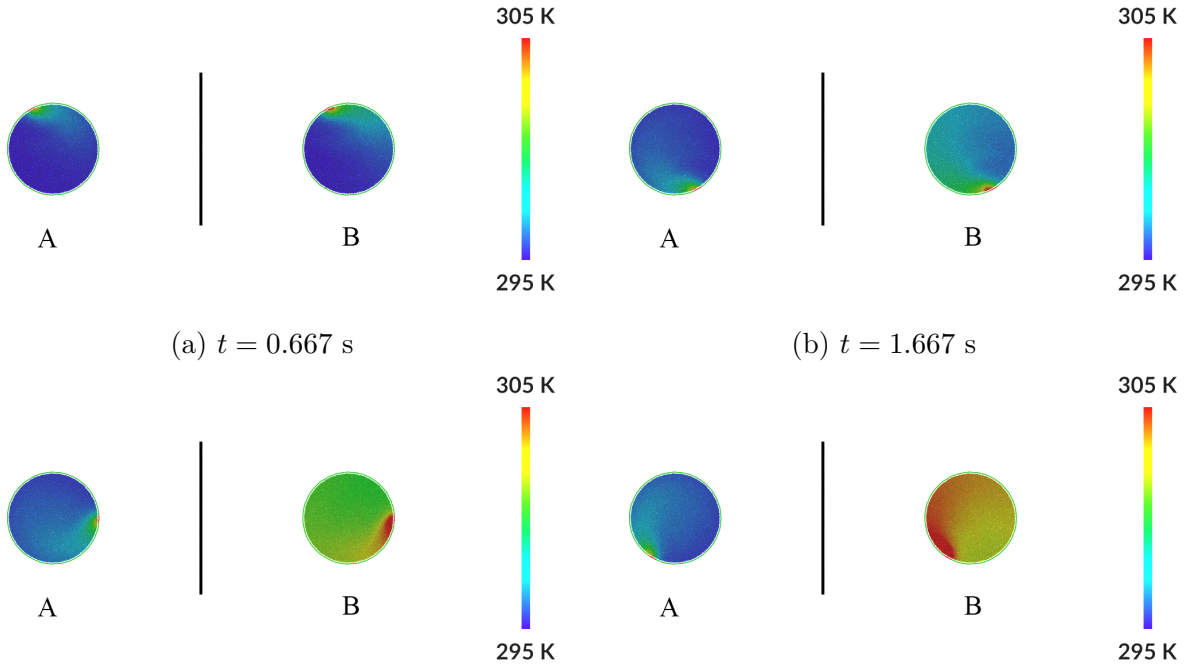
(c) $t = 1$ s (d) $t = 8$ s

Figure 3.8: Heat transfer in two discs. The discs initially have linear temperature distribution. Simulation A has the Robin boundary condition applied, while simulation B has only internal thermal conduction. The temperature in each disc reaches equilibrium over time. With the Robin boundary condition, the temperature of each disc approaches the ambient temperature.

without heat flux applied cools the disc to the ambient temperature. Simulation B does not have the Robin boundary condition, and the heat accumulates inside the disc.

3.3.3 Droplet Impact on Dry Surface

Our method is able to handle highly dynamic simulations with a wide range of surface tension strengths. We simulate several inviscid spherical droplets with radii of 0.15m free fall and impact a dry, frictionless, hydrophobic surface. In the top comparison of Figure 3.10, droplets with different surface tension coefficients drop from a height of 2.5m. The size of the simulation domain is 1m $\times$ 3m $\times$ 1m. With different surface tension coefficients k^{σ},



(a) $t = 0.667$ s (b) $t = 1.667$ s

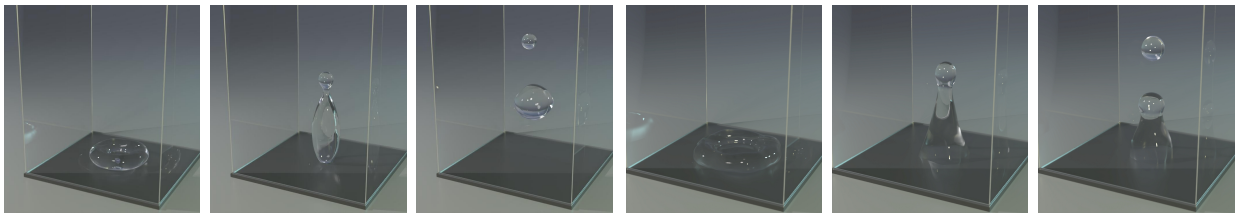
(c) $t = 4$ s (d) $t = 5.333$ s

Figure 3.9: Constant heat flux is applied to a small section of the disc boundary. The location of the heat flux rotates about the center of the disc at a constant speed. Robin boundary condition is enabled in simulation A and disabled in simulation B.

the droplets display distinct behaviors upon impact, as shown in Figure 3.10.

We also capture the partial rebound and the full rebound behaviors of the droplet after the impact. The middle and bottom rows of Figure 3.10 show the footage of a droplet with $k^\sigma = 15\text{N/m}$ dropped from a height of 3.5m (in $1\text{m} \times 4\text{m} \times 1\text{m}$ domain) and a droplet with $k^\sigma = 5\text{N/m}$ dropped from a height of 2.5m (in $1\text{m} \times 3\text{m} \times 1\text{m}$ domain), respectively. With a higher surface tension coefficient and a higher impact speed, the droplet is able to completely leave the surface after the impact. Our results qualitatively match the experiment outcomes from [RTM01].

(a)



(b) (c)

Figure 3.10: **(a)** Spherical droplets with different surface tension coefficients free fall from the same height. In the top figure, from left to right, the surface tension coefficients are $k^\sigma = 20, 5, 1, 0.1, 0.05$N/m. **(b)** full rebound of the droplet (initial height: 3.5m and $k^\sigma = 15$N/m). **(c)** partial rebound of the droplet (initial height: 2.5m and $k^\sigma = 5$N/m).

3.3.4 Droplets on Ramps

As discussed in Section 3.1.6, our method allows for distinct k^σ values at solid-liquid and liquid-air interfaces. Tuning the ratio between k^σ at these interfaces allows simulating different levels of hydrophilicity/hydrophobicity. Figure 3.11 shows an example of several liquid drops with different k^σ ratios falling on ramps of 5.5° angle. The length of the ramp is 3m,

and the domain size is $3\text{m} \times 0.5\text{m} \times 1\text{m}$. Coulomb friction with a friction coefficient of 0.2 was used for the ramp surface, and the drop has no viscosity. When there is a larger difference between solid-liquid and liquid-air surface tension coefficients (i.e., a smaller k^σ ratio), the liquid tends to drag more on the surface and undergo more separation and sticking. The leftmost example, with a k^σ ratio of 1.0, exhibits hydrophobic behavior.

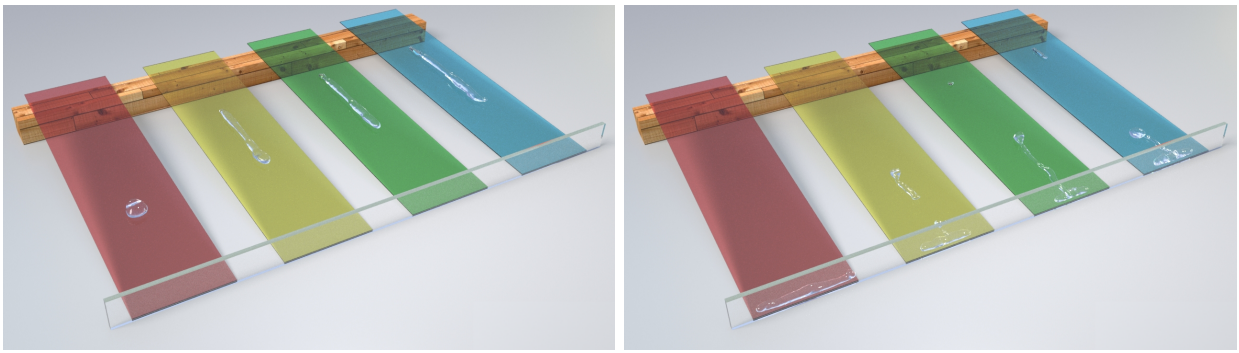


Figure 3.11: Liquid drops fall on a ramp with varying ratios between the solid-liquid and liquid-air surface tension coefficients. From left to right: ratios of 1.0, 0.6, 0.3, 0.05. *(Left)* Frame 60. *(Right)* Frame 100.

3.3.5 Lid-Driven Cavity

The two-dimensional lid-driven cavity is a classic example in the engineering literature of the Marangoni effect [FSK06, HSN18]. Inspired by works like these, we simulate a square unit domain and fill the domain with particles up to height $1 - 4\Delta x$ ($\Delta x = 1/63$), which results in a free surface near the top of the domain. A linear temperature gradient from 1 on the left to 0 on the right is initialized on the particles. To achieve the Marangoni effect, the surface tension coefficient k^σ is set to depend linearly on temperature: $k^\sigma = 1 - T_p$. k^σ is clamped to be in $[0, 1]$ to avoid artifacts due to numerical precision. Gravity is set to zero, dynamic viscosity is set to $1 \times 10^{-6}\text{Pa} \cdot \text{m}$ and implicit MPM is used with a maximum Δt of 0.001. Results are shown in Figure 3.12. We note that the center of the circulation drifts to the right over the course of the simulation due to uneven particle distribution resulting from

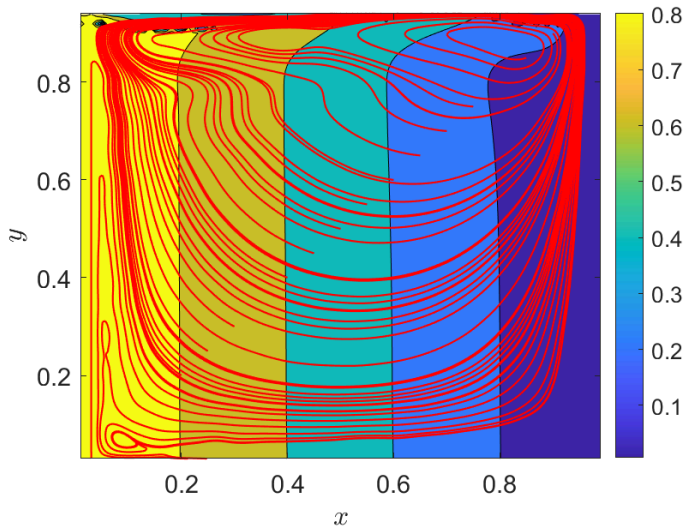


Figure 3.12: Frame 500 of a two-dimensional lid-driven cavity simulation. The simulation is initially stationary, but velocity streamlines (red) show the flow pattern characteristic of Marangoni convection that develops due to a temperature-dependent surface tension coefficient. The contour plot shows the evolving temperature field (initially a linear horizontal distribution).

the circulation of the particles; this drifting behavior is not observed in works like [FSK06] or [HSN18]. Investigating particle reseeding strategies to stabilize the flow is interesting future work.

3.3.6 Contact Angles

Figure 3.13 shows that our method enables simulation of various contact angles, emulating various degrees of hydrophobic or hydrophilic behavior as a droplet settles on a surface. We adjust the contact angles by assigning one surface tension coefficient, k^{σ}_{LG}, to the surface particles on the liquid-gas interface, and another one, k^{σ}_{SL}, to those on the solid-liquid interface. Following the Young equation (Equation (3.11)) and our assumption that k^{σ}_{SG} is negligible, the contact angle is given by $\theta = \arccos\big(-k^{\sigma}_{SL}/k^{\sigma}_{LG}\big)$. Note that the effect of gravity will

result in contact angles slightly smaller than targeted.

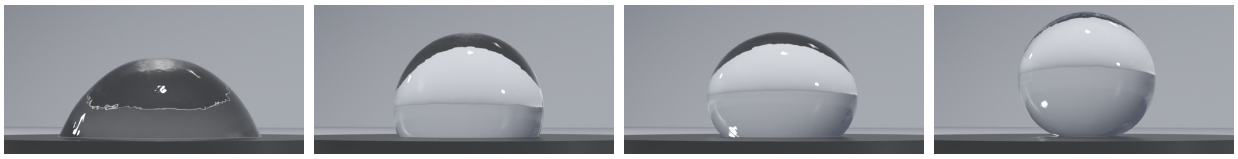


Figure 3.13: As our droplets settle, we are able to obtain contact angles of approximately 45, 90, 135 and 180 degrees, using a $k_{SL}^{\sigma}/k_{LG}^{\sigma}$ ratio of $-\sqrt{2}/2$, 0, $\sqrt{2}/2$ and 1, respectively.

A droplet of radius 0.1m is placed right above the ground in a 0.5m $\times$ 0.5m $\times$ 0.5m domain. Each droplet is discretized using 230k interior particles and 250k surface particles. The surface tension k_{LG}^{σ} is set to 2N/m, and we approximate k_{SL}^{σ} based on the desired contact angle θ. A dynamic viscosity of 0.075Pa $\cdot$ s is used to stabilize the simulations.

3.3.7 Soap Droplet in Water

We demonstrate a surface tension driven flow by simulating soap reducing the surface tension of the water. We initialize a 1m $\times$ 0.05m $\times$ 1m rectangular water pool, set a 0.075m-radius and 0.025m-height cylindrical region at the center of the pool to be liquid soap and identify particles in this region to be soap particles. We set the surface tension coefficients for the surface particles based on the type of its closest MPM particles. Surface particles associated with the water particles have higher surface tension than those associated with the soap particles. The viscosity is set to 0 in this example.

In order to visualize the effect of the surface tension driven flow, we randomly selected marker particles on the top surface of the pool. Due to the presence of the soap, the center of the pool has lower surface tension than the area near the edge of the container. The surface tension gradient drives the particles to flow from the center to the edge. Figure 3.14 shows footage of this process.

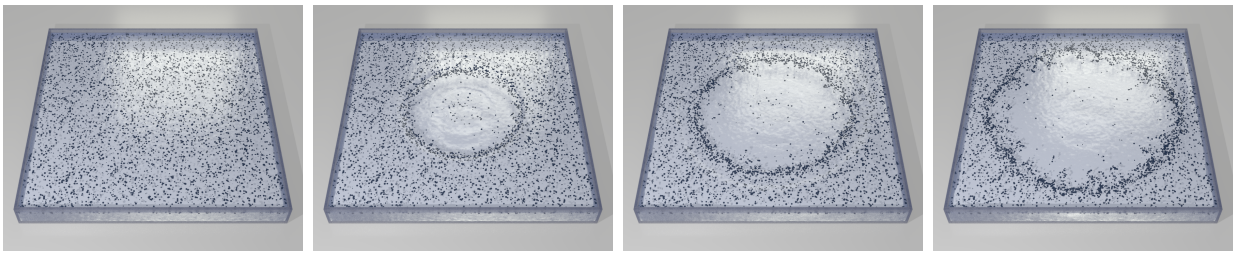


Figure 3.14: The soap in the center of the pool surface reduces the surface tension. The surface tension gradient drives the markers towards the walls of the container. Frames 0, 10, 20, 40 are shown in this footage.

3.3.8 Wine Glass

We consider an example of wine flowing on the surface of a pre-wetted glass. The glass is an ellipsoid centered at $(0.5\text{m}, 0.7\text{m}, 0.5\text{m})$ with characteristic dimensions $a = 0.4\text{m}$, $b = 0.6\text{m}$, and $c = 0.4\text{m}$. We initialize a thin band of particles with the thickness of $2\Delta x\text{m}$ on the surface of the wine glass and observe the formation of ridges and fingers as the particles settle toward the bulk fluid in the glass. We set the surface tension coefficient on the liquid-gas interface to be $k^{\sigma}_{LG} = 0.05\text{N/m}$ and the one on the solid-liquid interface to be $k^{\sigma}_{SL} = 0.015\text{N/m}$. The piecewise constant surface tension leads to a more prominent streaking behavior of the liquid on the glass wall. The results are shown in Figure 3.15.

3.3.9 Candles

We simulate several scenarios with wax candles. The height of the candle is 0.6m and the radius is 0.1m. The domain size is $1\text{m} \times 1\text{m} \times 1\text{m}$. In these examples, wax melts due to a heat source (candle flame) and resolidifies when it flows away from the flame. Ambient temperature $\hat{T}$ is taken to be 298K, and the melting point is 303K. Thermal conductivity K is taken to be $0.1\text{W/m} \cdot \text{K}$, and specific heat capacity c_p is set to 1J/K. No internal heat source is used ($H = 0$); instead, heating and cooling are applied only via the boundary conditions.

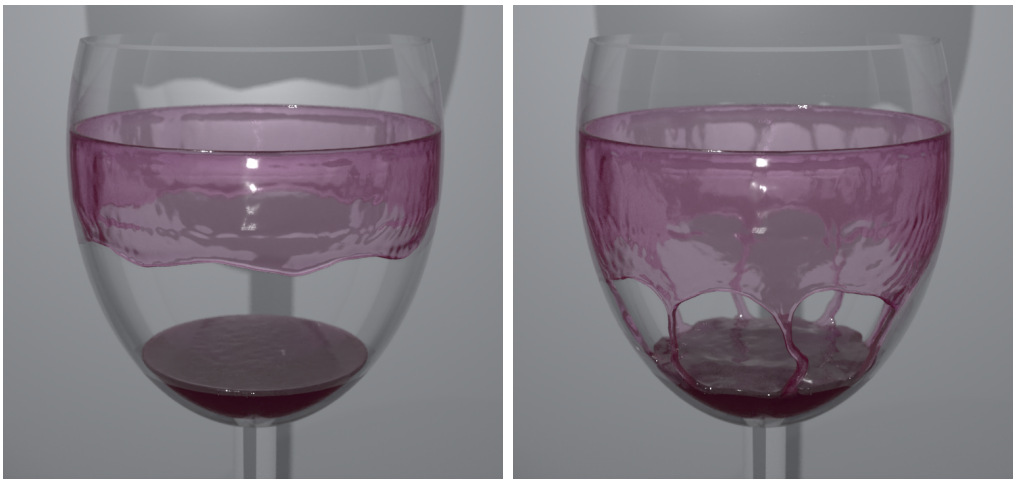


Figure 3.15: Wine is initialized in a glass with part of the interior pre-wetted. The falling wine forms tears and ridges, and the tears eventually connect with the bulk fluid. Frames 30 and 90 are shown.

To simulate the candle wicks, we manually construct and sample points on cubic splines. As the simulation progresses, we delete particles from the wick that are too far above the highest (y-direction) liquid particle within a neighborhood of the wick. The flames are created by running a separate FLIP simulation as a postprocess and anchoring the result to the exposed portion of each wick. We rendered these scenes using Arnold [GIF18] and postprocessed the renders using the NVIDIA OptiX denoiser (based on [CKS17]).

We consider the effect of varying k^σ on the overall behavior of the flow. Figure 3.16 compares k^σ values of 0.05N/m, 0.1N/m, 0.2N/m, and 0.4N/m. In these examples, a grid resolution of $\Delta x = 1/127$m was used, along with boundary condition parameters $h = 0.5\text{W/m}^2 \cdot \text{K}$ and $b = 50\text{W/m}^2$. The figure demonstrates that as surface tension increases, the molten wax spreads significantly less. As the wax cools and resolidifies, visually interesting layering behavior is observed.

Figure 3.17 shows an example of several candle letters melting in a container. Wicks follow generally curved paths inside the letters. Melt pools from the different letters seamlessly interact. This simulation used a surface tension coefficient $k^\sigma = 0.05\text{N/m}$, $h = 2.5\text{W/m}^2 \cdot \text{K}$,

Figure 3.16: Various k^σ values (0.05, 0.1, 0.2, 0.4N/m) are simulated in the case of a melting candle. Frame 1202 is shown.

$b = 100\text{W/m}^2$, dynamic viscosity of $0.01\text{Pa} \cdot \text{s}$, $\Delta x = 1/127\text{m}$, and a bulk modulus of 83333.33Pa for liquid and solid phases.

3.3.10 Droplet with Marangoni Effect

We simulate an inviscid liquid metal droplet that moves under the Marangoni effect, i.e., due to a temperature-induced surface tension gradient. A spherical drop with radius of 0.1m is initialized at position $(0.5\text{m}, (0.1 + 3.5\Delta x)\text{m}, 0.5\text{m})$ inside a $2\text{m} \times 1\text{m} \times 1\text{m}$ domain. We then turn on the heating 1 second after the simulation starts (while the droplet is still spreading). The Neumann boundary condition is applied to heat the particles in the region with $x \leq 0.5\text{m}$. The thermal conductivity K is set to be $0.1\text{W}/(\text{m} \cdot \text{K})$, and the convective

heat transfer coefficient h is $0.1\text{W}/(\text{m}^2 \cdot \text{K})$. The boundary heating rate $b = 50\text{W}/\text{m}^2$ is much higher than the conduction and convection, so the heat transfer inside the droplet and the heat exchange between the droplet and the environment are less prominent.

We define the surface tension coefficient as a function of temperature: $k^\sigma = \min(0.09(T - \bar{T}) + 0.5, 2)\text{N}/\text{m}$, where $\bar{T} = 50\text{K}$ is the ambient temperature. At its original temperature, the surface tension coefficient k^σ of the droplet is $0.5\text{N}/\text{m}$. As the temperature increases, k^σ increases linearly with the temperature. The maximum allowable surface tension strength is $2\text{N}/\text{m}$. After the heating, surface particles on the hotter side of the droplet have higher surface tension. The stronger surface tension penalizes the area changes and drives the particles to flow to the colder side, as shown in Figure 3.18. This surface tension gradient results in an interesting self-propelled behavior of the liquid metal droplet.

3.3.11 Performance

Table 3.2 shows average per-timestep runtime details for several of our examples. For this table, all experiments were run on a workstation equipped with 128GB RAM and with dual Intel® Xeon® E5-2687W v4 CPUs at 3.00Ghz.

3.4 Discussion and Future Work

Our method allows for simulation of surface tension energies with spatial gradients, including those driven by variation in temperature. Our MPM approach to the problem resolves many interesting characteristic phenomena associated with these variations. However, while we provide perfect conservation of linear and angular momentum, our approach to the thermal transfers is not perfectly conservative. Developing a thermally conservative transfer strategy is interesting future work. Also, although we simulate tears of wine on the walls of a glass, we did not simulate the effect of alcohol evaporation on the surface energy variation. Adding in a mixture model as in [DHW19] would be interesting future work. Lastly, although our

Table 3.2: Performance measurements for one time step of several of our 3D examples, broken down by (1) sampling: generating surface and balance particles and conservative momentum splitting, (2) conservative momentum merging, (3) single particle-to-grid transfer, (4) single grid-to-particle transfer, (5) total time of the linear solve, (6) total time of one time step. Note that each linear solve involves several particle-to-grid and grid-to-particle transfers, and each time step requires several linear solves. All times are in milliseconds.

Example	# Cells	# Int. Part.	# Surf. Part.	Sampling	Merging	Part.→Grid	Grid→Part.	Linear Solve	Time Step
Droplet Impact ($k^\sigma = 5$)	2M	794K	100K	2224	20	95	39	1422	10065
Droplets on Ramps ($k^\sigma_{SL}/k^\sigma_{LG} = 0.05$)	1.5M	70K	100K	258	6	28	9	199	1434
Contact Angles ($k^\sigma_{SL}/k^\sigma_{LG} = 0$)	256K	230K	250K	492	17	73	38	647	4286
Soap Droplet in Water	1M	4M	200K	2166	33	575	257	5599	35304
Wine Glass	2M	1.6M	500K	1549	52	163	91	2030	12440
Candle ($k^\sigma = 0.1$)	2M	618K	50K	1420	7	122	44	2646	29162
Candle Letters	256K	3.1M	100K	4601	15	574	187	8445	172787
Droplet with Marangoni Effect	4.1M	235K	200K	2991	18	82	51	1636	12056

approach was designed for MPM, SPH is more commonly used for the simulation of liquids. However, SPH and MPM have many similarities, as recently shown by the work of Gissler et al. [GHB20], and it would be interesting future work to generalize our approach to SPH.

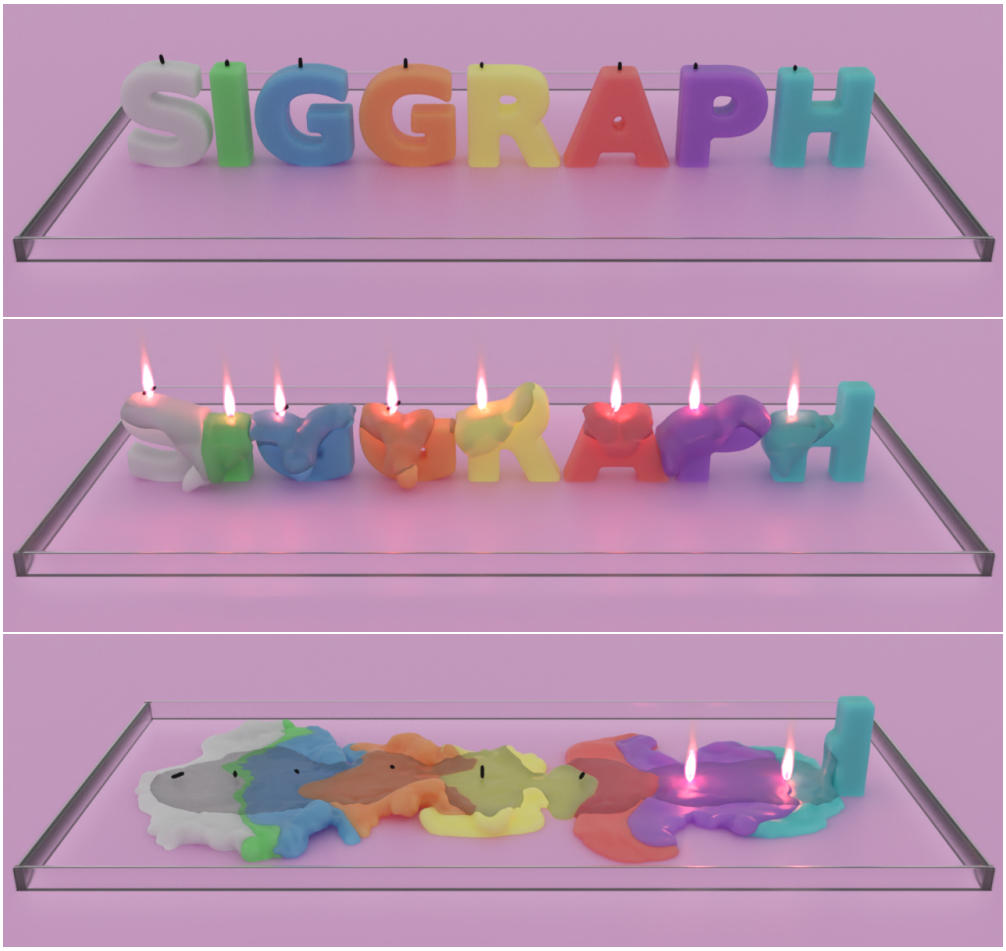


Figure 3.17: Letter-shaped candles melt inside a container. *(Top)* Frame 1, before flames are lit. *(Middle)* Frame 60, in the middle of melting. *(Bottom)* Frame 200, as flames are extinguished and wax pools resolidify.

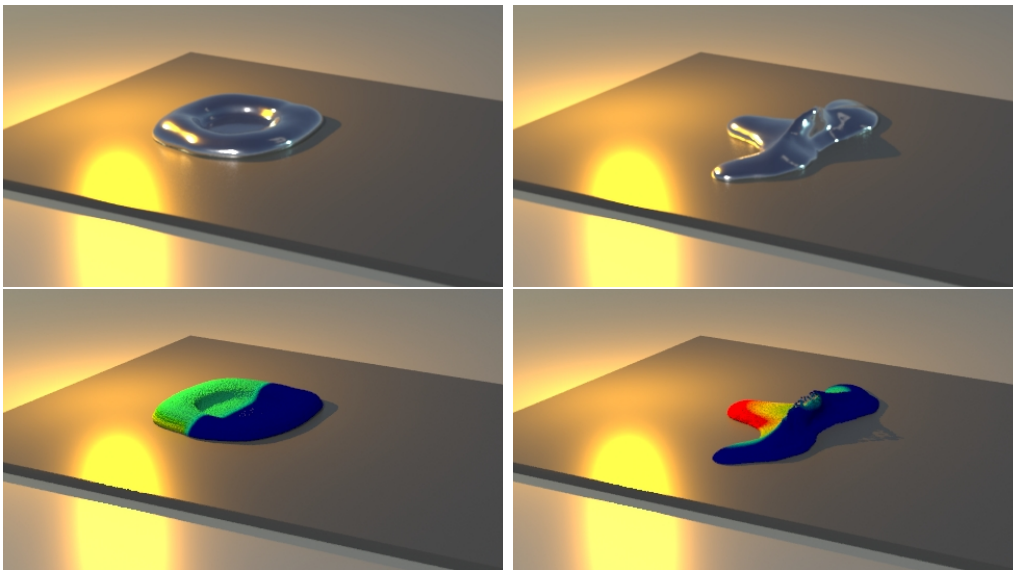


Figure 3.18: A liquid metal droplet subjected to heating on one side. The surface tension coefficient increases as the temperature increases. *(Top)* the liquid metal at frame 45 and frame 130. *(Bottom)* the particle view of temperature distribution at frame 45 and frame 130. The red color indicates higher temperature.

CHAPTER 4

A Deep Conjugate Direction Method for Iteratively Solving Linear Systems

4.1 Motivation: Incompressible Flow

We demonstrate the efficacy of our approach with the linear systems that arise in incompressible flow applications. Specifically, we use our algorithm to solve the Poisson equation discretized on a regular grid, following the pressure projection equations that arise in Chorin's splitting technique [Cho67] for the inviscid, incompressible Euler equations. These equations are

$$\rho\left(\frac{\partial \mathbf{u}}{\partial t}+\frac{\partial \mathbf{u}}{\partial \boldsymbol{x}}\mathbf{u}\right)+\nabla p=\mathbf{f}^{ext}, \qquad \nabla\cdot\mathbf{u}=0 \tag{4.1}$$

where $\mathbf{u}$ is fluid velocity, p is pressure, ρ is density, and $\mathbf{f}^{ext}$ accounts for external forces like gravity. The equations are assumed at all positions $\boldsymbol{x}$ in the spatial fluid flow domain Ω and for time $t>0$. The first equation in Equation 4.1 enforces conservation of momentum in the absence of viscosity, and the second enforces incompressibility and conservation of mass. These equations are subject to initial conditions $\rho(\boldsymbol{x},0)=\rho^0$ and $\mathbf{u}(\boldsymbol{x},0)=\mathbf{u}^0(\boldsymbol{x})$, as well as boundary conditions $\mathbf{u}(\boldsymbol{x},t)\cdot\mathbf{n}(\boldsymbol{x})=u^{\partial\Omega}(\boldsymbol{x},t)$ on the boundary of the domain $\boldsymbol{x}\in\partial\Omega$ (where $\mathbf{n}$ is the unit outward pointing normal at position $\boldsymbol{x}$ on the boundary).

Equation 4.1 is discretized in both time and space. Temporally, we split the advection $\frac{\partial \mathbf{u}}{\partial t}+\frac{\partial \mathbf{u}}{\partial \boldsymbol{x}}\mathbf{u}=0$ and body forces terms $\rho\frac{\partial \mathbf{u}}{\partial t}=\mathbf{f}^{ext}$, and finally enforce incompressibility via the pressure projection $\frac{\partial \mathbf{u}}{\partial t}+\frac{1}{\rho}\nabla p=\mathbf{0}$ such that $\nabla\cdot\mathbf{u}=0$; this is the standard advection-

projection scheme proposed by [Cho67]. Using finite differences in time, we can summarize this as

$$\rho^0 \left(\frac{\mathbf{u}^* - \mathbf{u}^n}{\Delta t} + \frac{\partial \mathbf{u}^n}{\partial \boldsymbol{x}} \mathbf{u}^n \right) = \mathbf{f}^{ext} \tag{4.2}$$

$$-\nabla \cdot \frac{1}{\rho^0} \nabla p^{n+1} = -\nabla \cdot \mathbf{u}^* \tag{4.3}$$

$$-\frac{1}{\rho^0} \nabla p^{n+1} \cdot \mathbf{n} = \frac{1}{\Delta t} \left(u^{\partial\Omega} - \mathbf{u}^* \cdot \mathbf{n} \right). \tag{4.4}$$

For the spatial discretization, we use a regular marker-and-cell (MAC) grid [HW65] with cubic voxels whereby velocity components are stored on the face of voxel cells, and scalar quantities (e.g., pressure p or density ρ) are stored at voxel centers. We use backward semi-Lagrangian advection [Sta99, FSJ01, GHM20] for Equation 4.2. All spatial partial derivatives are approximated using finite differences. Equations 4.3 and 4.4 describe the pressure Poisson equation with Neumann conditions on the boundary of the flow domain. We discretize the left-hand side of Equation 4.3 using a standard 7-point finite difference stencil. The right-hand side is discretized using the MAC grid discrete divergence finite difference stencils as well as contributions from the boundary condition terms in Equation 4.4. We refer the reader to [Bri08] for more in-depth implementation details. Equation 4.4 is discretized by modifying the Poisson stencil to enforce Neumann boundary conditions. We do this using a simple labeling of the voxels in the domain. For simplicity, we assume $\Omega \subset (0,1)^3$ is a subset of the unit cube, potentially with internal boundaries. We label cells in the domain as either liquid or boundary. This simple classification is enough to define the discrete Poisson operators (with appropriate Neumann boundary conditions at domain boundaries) that we focus on in the present work; we illustrate the details in Figure 4.1.

We use the following notation to denote the discrete Poisson equations associated with Equations 4.3–4.4:

$$\mathbf{A}^{\Omega} \mathbf{x} = \mathbf{b}^{\nabla \cdot \mathbf{u}^*} + \mathbf{b}^{u^{\partial\Omega}}, \tag{4.5}$$

where $\mathbf{A}^{\Omega}$ is the discrete Poisson matrix associated with the voxelized domain, $\mathbf{x}$ is the vector of unknown pressure, and $\mathbf{b}^{\nabla \cdot \mathbf{u}^*}$ and $\mathbf{b}^{u^{\partial\Omega}}$ are the right-hand side terms from Equations 4.3

and 4.4, respectively. $\mathbf{A}^{\Omega}$ in Equation 4.5, is a large, sparse, SPD linear system. The computational complexity of solving Equation 4.5 strongly depends on data (e.g., internal boundary conditions in the flow domain, see Figure 4.1).

We define a special case of the matrix involved in this discretization to be the Poisson matrix $\mathbf{A}^{\text{train}}$ associated with $\Omega = (0, 1)^3$, i.e., a full fluid domain with no internal boundaries. We use this matrix for training, yet demonstrate that our network generalizes to all other matrices arising from more complicated flow domains. To be clear, the implication of this is that by training DCDM *one time*—which we have already done, and we release our pre-trained models and source code along with this paper—practitioners can immediately apply DCDM to *any* Poisson system (regardless of internal boundary conditions, etc.). Although there is a clear limitation that we only train our network to solve Poisson problems, this is a major advantage over state-of-the-art methods like FluidNet [TSS17], which require highly diverse training data (matrices from many fluid simulations, all with different types of obstacles and boundary conditions) in order to train a network with sufficient generalization; we only ever leverage a single training matrix (i.e., a single set of boundary conditions) $\mathbf{A}^{\text{train}}$.

4.2 Deep Conjugate Direction Method

We present our method for the deep learning acceleration of iterative approximations to the solution of linear systems of the form seen in Equation 4.5. We first briefly discuss relevant details of search direction methods, particularly the choice of line search directions[1]. We then present a deep learning technique for improving the quality of these search directions that ultimately reduces iteration counts required to achieve satisfactory residual reduction. Lastly, we outline the training procedures for our deep CNN.

Our approach iteratively improves approximations to the solution $\mathbf{x}$ of Equation 4.5. We build on the method of CG, which requires the matrix $\mathbf{A}^{\Omega}$ in Equation 4.5 to be SPD. SPD

[1]For a comprehensive background on CG, see Appendix B.1.

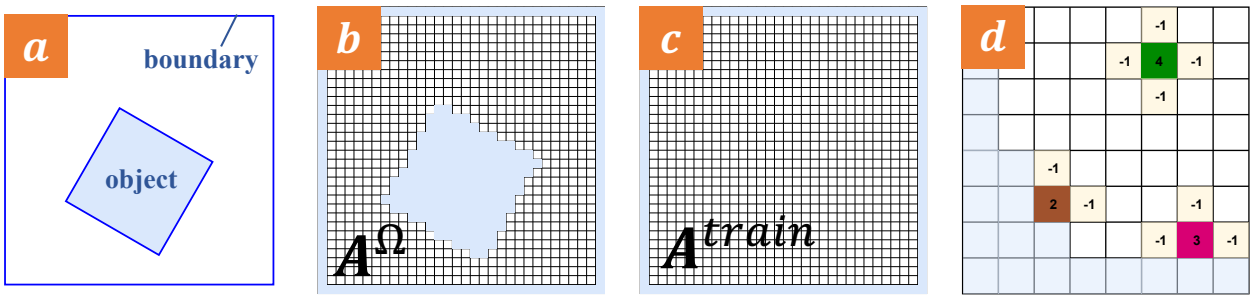


Figure 4.1: **(a)** We illustrate a sample flow domain $\Omega \subset (0,1)^2$ (in 2D for ease of illustration) with internal boundaries (blue lines). **(b)** We voxelize the domain with a regular grid: white cells represent interior/fluid, and blue cells represent boundary conditions. **(c)** We train using the matrix $\mathbf{A}^{\text{train}}$ from a discretized domain with *no* interior boundary conditions, where d is the dimension. This creates linear system with $n = (n_c + 1)^d$ unknowns, where n_c is the number of grid cells on each direction. **(d)** We illustrate the non-zero entries in an example matrix $\mathbf{A}^{\Omega}$ from the voxelized and labeled (white vs. blue) grid for three example interior cells (green, magenta, and brown). Each case illustrates the non-zero entries in the row associated with the example cell. All entries of $\mathbf{A}^{\Omega}$ in rows corresponding to boundary/blue cells are zero.

matrices $\mathbf{A}^{\Omega}$ give rise to the matrix norm $\|\mathbf{y}\|_{\mathbf{A}^{\Omega}} = \sqrt{\mathbf{y}^T \mathbf{A}^{\Omega} \mathbf{y}}$. CG can be derived in terms of iterative line search improvement based on optimality in this norm. That is, an iterate $\mathbf{x}_{k-1} \approx \mathbf{x}$ is updated along search direction $\mathbf{d}_k$ by a step size α_k that is chosen to minimize the matrix norm of the error between the updated iterate and $\mathbf{x}$:

$$\begin{aligned} \alpha_k &= \arg\min_{\alpha} \frac{1}{2} \|\mathbf{x} - (\mathbf{x}_{k-1} + \alpha \mathbf{d}_k)\|_{\mathbf{A}^{\Omega}}^2 \\ &= \frac{\mathbf{r}_{k-1}^T \mathbf{d}_k}{\mathbf{d}_k^T \mathbf{A}^{\Omega} \mathbf{d}_k}, \end{aligned} \tag{4.6}$$

where $\mathbf{r}_{k-1} = \mathbf{b} - \mathbf{A}^{\Omega} \mathbf{x}_{k-1}$ is the $(k-1)^{\text{th}}$ residual (see Appendix B.2 for details). Different search directions $\mathbf{d}_k$ result in different algorithms. A natural choice is the negative gradient of the matrix norm of the error (evaluated at the current iterate), $\mathbf{d}_k = -\frac{1}{2} \nabla \|\mathbf{x}_{k-1}\|_{\mathbf{A}^{\Omega}}^2 = \mathbf{r}_{k-1}$, since this will point in the direction of steepest decrease. This is the gradient descent method

(GD). Unfortunately, this approach requires many iterations in practice. CG modifies GD into a more effective strategy by instead choosing directions that are $\mathbf{A}$-orthogonal (i.e., $\mathbf{d}_i^T \mathbf{A}^\Omega \mathbf{d}_j = 0$ for $i \neq j$). More precisely, the search direction $\mathbf{d}_k$ is chosen as follows:

$$\mathbf{d}_k = \mathbf{r}_{k-1} - \sum_{i=1}^{k-1} h_{ik} \mathbf{d}_i, \qquad h_{ik} = \frac{\mathbf{d}_i^T \mathbf{A}^\Omega \mathbf{r}_{k-1}}{\mathbf{d}_i^T \mathbf{A}^\Omega \mathbf{d}_i},$$

which guarantees A-orthogonality. The magic of CG is that $h_{ik} = 0$ for $i < k-1$, hence this iteration can be performed without the need to store all previous search directions $\mathbf{d}_i$ and without the need for computing all previous h_{ik}.

While the residual is a natural choice for generating $\mathbf{A}$-orthogonal search directions (since it points in the direction of the steepest local decrease), it is not the optimal search direction. Optimality is achieved when $\mathbf{d}_k$ is parallel to $(\mathbf{A}^\Omega)^{-1}\mathbf{r}_{k-1}$, whereby $\mathbf{x}_k$ will be equal to $\mathbf{x}$ since α_k (computed from Equation 4.6) will step directly to the solution. We can see this by considering the residual and its relation to the search direction:

$$\begin{aligned} \mathbf{r}_k &= \mathbf{b} - \mathbf{A}^\Omega \mathbf{x}_k = \mathbf{b} - \mathbf{A}^\Omega \mathbf{x}_{k-1} - \alpha_k \mathbf{A}^\Omega \mathbf{d}_k \\ &= \mathbf{r}_{k-1} - \alpha_k \mathbf{A}^\Omega \mathbf{d}_k. \end{aligned}$$

In light of this, we use deep learning to create an approximation $\mathbf{f}(\mathbf{c}, \mathbf{r})$ to $(\mathbf{A}^\Omega)^{-1}\mathbf{r}$, where $\mathbf{c}$ denotes the network weights and biases. This is analogous to using a preconditioner in PCG; however, our network is not SPD (nor even a linear function). We simply use this data-driven approach as our means of generating better search directions $\mathbf{d}_k$. Furthermore, we only need to approximate a vector parallel to $(\mathbf{A}^\Omega)^{-1}\mathbf{r}$ since the step size α_k will account for any scaling in practice. In other words, $\mathbf{f}(\mathbf{c}, \mathbf{r}) \approx s_{\mathbf{r}} (\mathbf{A}^\Omega)^{-1}\mathbf{r}$, where the scalar $s_{\mathbf{r}}$ is not defined globally; it only depends on $\mathbf{r}$, and the model does not learn it. Lastly, as with CG, we enforce $\mathbf{A}$-orthogonality, yielding search directions

$$\mathbf{d}_k = \mathbf{f}(\mathbf{c}, \mathbf{r}_{k-1}) - \sum_{i=1}^{k-1} h_{ik} \mathbf{d}_i, h_{ik} = \frac{\mathbf{f}(\mathbf{c}, \mathbf{r}_{k-1})^T \mathbf{A}^\Omega \mathbf{d}_i}{\mathbf{d}_i^T \mathbf{A}^\Omega \mathbf{d}_i}.$$

We summarize our approach in Algorithm 1. Note that we introduce the variable i_{start}. To guarantee $\mathbf{A}$-orthogonality between all search directions, we must have $i_{\text{start}} = 1$. However,

this requires storing all prior search directions, which can be costly. We found that using $i_{\text{start}} = k - 2$ worked nearly as well as $i_{\text{start}} = 1$ in practice (in terms of our ability to iteratively reduce the residual of the system). We demonstrate this in Figure 4.4c.

Algorithm 1: DCDM

$\mathbf{r}_0 = \mathbf{b} - \mathbf{A}^{\Omega}\mathbf{x}_0$;
$k = 1$;
while $\|\mathbf{r}_{k-1}\| \geq \epsilon$ **do**
  $\mathbf{d}_k = \mathbf{f}(\mathbf{c}, \frac{\mathbf{r}_{k-1}}{\|\mathbf{r}_{k-1}\|})$;
  for $i_{start} \leq i < k$ **do**
    $h_{ik} = \frac{\mathbf{d}_k^T \mathbf{A}^{\Omega} \mathbf{d}_i}{\mathbf{d}_i^T \mathbf{A}^{\Omega} \mathbf{d}_i}$;
    $\mathbf{d}_k$-=$h_{ik}\mathbf{d}_i$;
  end
  $\alpha_k = \frac{\mathbf{r}_{k-1}^T \mathbf{d}_k}{\mathbf{d}_k^T \mathbf{A}^{\Omega} \mathbf{d}_k}$;
  $\mathbf{x}_k = \mathbf{x}_{k-1} + \alpha_k \mathbf{d}_k$;
  $\mathbf{r}_k = \mathbf{b} - \mathbf{A}^{\Omega}\mathbf{x}_k$;
  $k = k + 1$;
end

4.3 Model Architecture, Datasets, and Training

Efficient performance of our method requires effective training of our deep convolutional network for weights and biases $\mathbf{c}$ such that $\mathbf{f}(\mathbf{c}, \mathbf{r}) \approx s_{\mathbf{r}}(\mathbf{A}^{\Omega})^{-1}\mathbf{r}$ (for arbitrary scalar $s_{\mathbf{r}}$). We design a model architecture, loss function, and self-supervised training approach to achieve this. Our approach has modest training requirements and allows for effective residual reduction while generalizing well to problems not seen in the training data.

4.3.1 Loss Function and Self-supervised Learning

Although we generalize to arbitrary matrices $\mathbf{A}^{\Omega}$ from Equation 4.5 that correspond to domains $\Omega \subset (0,1)^3$ that have internal boundaries (see Figure 4.1), we train using just the matrix $\mathbf{A}^{\text{train}}$ from the full cube domain $(0,1)^3$. "the full cube domain $(0,1)^3$" is just the unit cube discretized on regular intervals, see e.g. Figure 4.1(c).

In contrast, other similar approaches [TSS17, YYX16] train using matrices $\mathbf{A}^{\Omega}$ and right-hand sides $\mathbf{b}^{\nabla \cdot \mathbf{u}^*} + \mathbf{b}^{u^{\partial\Omega}}$ that arise from flow in many domains with internal boundaries. We train our network by minimizing the L^2 difference $\|\mathbf{r} - \alpha \mathbf{A}^{\text{train}} \mathbf{f}(\mathbf{c}, \mathbf{r})\|_2$, where $\alpha = \frac{\mathbf{r}^T \mathbf{f}(\mathbf{c},\mathbf{r})}{\mathbf{f}(\mathbf{c},\mathbf{r})^T \mathbf{A}^{\text{train}} \mathbf{f}(\mathbf{c},\mathbf{r})}$ from Equation 4.6. This choice of α accounts for the unknown scaling in the approximation of $\mathbf{f}(\mathbf{c}, \mathbf{r})$ to $\left(\mathbf{A}^{\text{train}}\right)^{-1} \mathbf{r}$. We use a self-supervised approach and train the model by minimizing

$$\text{Loss}(\mathbf{f}, \mathbf{c}, \mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{\mathbf{r} \in \mathcal{D}} \|\mathbf{r} - \frac{\mathbf{r}^T \mathbf{f}(\mathbf{c}, \mathbf{r})}{\mathbf{f}(\mathbf{c}, \mathbf{r})^T \mathbf{A}^{\text{train}} \mathbf{f}(\mathbf{c}, \mathbf{r})} \mathbf{A}^{\text{train}} \mathbf{f}(\mathbf{c}, \mathbf{r})\|_2$$

for a given dataset $\mathcal{D}$ consisting of training vectors $\mathbf{b}^i$. In Algorithm 1, the normalized residuals $\frac{\mathbf{r}_k}{\|\mathbf{r}_k\|}$ are passed as inputs to the model. Unlike in e.g. FluidNet [TSS17], only the first residual $\frac{\mathbf{r}_0}{\|\mathbf{r}_0\|}$ is directly related to the problem-dependent original right-hand side $\mathbf{b}$. Hence we consider a broader range of training vectors than those expected in a given problem of interest, e.g., incompressible flows. We observe that generally the residuals $\mathbf{r}_k$ in Algorithm 1 are skewed to the lower end of the spectrum of the matrix $\mathbf{A}^{\Omega}$. Since $\mathbf{A}^{\Omega}$ is a discretized elliptic operator, lower end modes are of lower frequency of spatial oscillation. We create our training vectors $\mathbf{b}^i \in \mathcal{D}$ using $m \ll n$ approximate eigenvectors of the training matrix $\mathbf{A}^{\text{train}}$. We use the Rayleigh-Ritz method to create approximate eigenvectors $\mathbf{q}_i$, $0 \leq i < m$. This approach allows us to effectively approximate the full spectrum of $\mathbf{A}^{\text{train}}$ without computing the full eigendecomposition, which can be expensive ($O(n^3)$) at high resolution. Note that generating the dataset has $O(m^2 N)$ complexity, N being the resolution (e.g., 64^3 or 128^3), due to reorthogonalization of Lanczos vectors (see Appendix B.4). Hence we tried values like m = 1K, 5K, 10K, and 20K, and chose the smallest value (m = 10,000) that gave a viable

model after training.

The Rayleigh-Ritz vectors are orthonormal and satisfy $\mathbf{Q}_m^T \mathbf{A}^{\text{train}} \mathbf{Q}_m = \mathbf{\Lambda}_m$, where $\mathbf{\Lambda}_m$ is a diagonal matrix with nondecreasing diagonal entries λ_i referred to as Ritz values (approximate eigenvalues) and $\mathbf{Q}_m = [\mathbf{q}_0, \mathbf{q}_1, \ldots, \mathbf{q}_{m-1}] \in \mathbb{R}^{n \times m}$. We pick $\mathbf{b}^i = \frac{\sum_{j=0}^{m-1} c_j^i \mathbf{q}_j}{\left\| \sum_{j=0}^{m-1} c_j^i \mathbf{q}_j \right\|}$, where the coefficients c_j^i are picked from a standard normal distribution

$$c_j^i = \begin{cases} 9 \cdot \mathcal{N}(0,1) & \text{if } \tilde{j} \leq j \leq \frac{m}{2} + \theta \\ \mathcal{N}(0,1) & \text{otherwise} \end{cases}$$

where θ is a small number (we used $\theta = 500$), and $\tilde{j}$ is the first index that $\lambda_{\tilde{j}} > 0$. This choice creates 90% of $\mathbf{b}^i$ from the lower end of the spectrum, with the remaining 10% from the higher end. The Riemann-Lebesgue Lemma states the Fourier spectrum of a continuous function will decay at infinity, so this specific choice of $\mathbf{b}_i$'s is reasonable for the training set. In practice, we also observed that the right-hand sides of the pressure system that arose in flow problems (in the empty domain) tended to be at the lower end of the spectrum. Notably, even though this dataset only uses Rayleigh-Ritz vectors from the training matrix $\mathbf{A}^{\text{train}}$, our network can be effectively generalized to flows in irregular domains, e.g., smoke flowing past a rotating box and flow past a bunny (see Figure 4.3).

We generate the Rayleigh-Ritz vectors by first tridiagonalizing the training matrix $\mathbf{A}^{\text{train}}$ with m Lanczos iterations [Lan50] to form $\mathbf{T}^m = {\mathbf{Q}_m^L}^T \mathbf{A}^{\text{train}} \mathbf{Q}_m^L \in \mathbb{R}^{m \times m}$. We then diagonalize $\mathbf{T}^m = \hat{\mathbf{Q}}^T \mathbf{\Lambda}_m \hat{\mathbf{Q}}$. While asymptotically costly, we note that this algorithm is performed on the comparably small $m \times m$ matrix $\mathbf{T}^m$ (rather than on the $\mathbf{A}^{\text{train}} \in \mathbb{R}^{n \times n}$). This yields the Rayleigh-Ritz vectors as the columns of $\mathbf{Q}_m = \mathbf{Q}_m^L \hat{\mathbf{Q}}$. The Lanczos vectors are the columns of the matrix $\mathbf{Q}_m^L$ and satisfy a three-term recurrence whereby the next Lanczos vector can be iteratively computed from previous two as

$$\beta_j \mathbf{q}_{j+1}^L = \mathbf{A}^{\text{train}} \mathbf{q}_j^L - \beta_{j-1} \mathbf{q}_{j-1}^L - \alpha_j \mathbf{q}_j^L,$$

where α_j and β_j are diagonal and subdiagonal entries of $\mathbf{T}^k$. β_j is computed so that $\mathbf{q}_{j+1}^L$ is a unit vector, and $\alpha_{j+1} = \mathbf{q}_{j+1}^T \mathbf{A}^{\text{train}} \mathbf{q}_{j+1}$. We initialize the iteration with a random

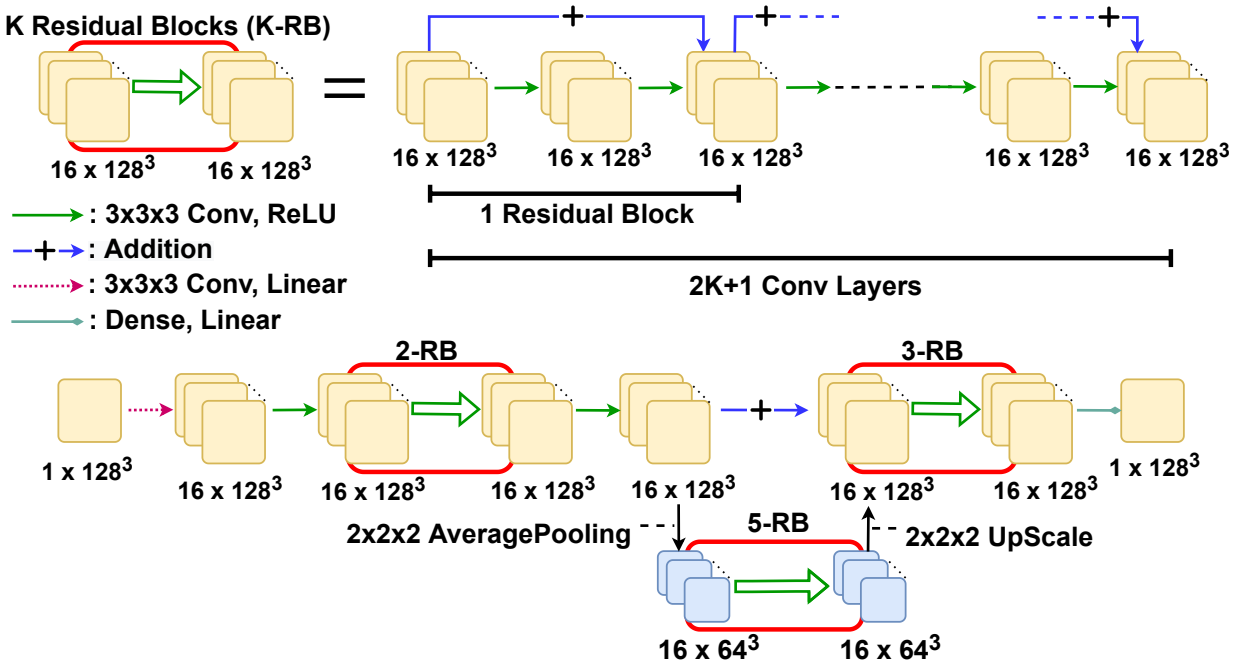


Figure 4.2: Architecture for training with $\mathbf{A}^{\text{train}}$ on a 128^3 grid.

$\mathbf{q}_0^L \in \text{span}(\mathbf{A}^{\text{train}})$. The Lanczos algorithm can be viewed as a modified Gram-Schmidt technique to create an orthonormal basis for the Krylov space associated with $\mathbf{q}_0^L$ and $\mathbf{A}^{\text{train}}$, and it therefore suffers from rounding error sensitivities manifested as loss of orthonormality with vectors that do not appear in the recurrence. We found that the simple strategy described in [Pai71] of orthogonalizing each iterate with respect to all previous Lanczos vectors to be sufficient for our training purposes. Dataset creation takes 5–7 hours for a 64^3 computational grid, and 2–2.5 days for a 128^3 grid (see Appendix B.4 for more detail).

We reiterate that since DCDM generalizes to various Poisson systems (see Sections 4.3.2 and 4.4) despite only using data corresponding to an empty fluid domain, practitioners do not need to generate new data in order to apply our method. Moreover, we show in the examples that it is possible to use trained model weights from a lower-resolution grid for higher-resolution problems, so practitioners may not need to generate new data even if running problems at different resolutions than what we consider.

4.3.2 Model Architecture

The internal structure of our CNN architecture for a 128^3 grid is shown in Figure 4.2. It consists of a series of convolutional layers with residual connections. The upper left of Figure 4.2 (K Residual Blocks) shows our use of multiple blocks of residually connected layers. Notably, within each block, the first layer directly affects the last layer with an addition operator. All non-input or output convolutions use a $3 \times 3 \times 3$ filter, and all layers consist of 16 feature maps. In the middle of the first level, a layer is downsampled (via the average pooling operator with $(2 \times 2 \times 2)$ pool size) and another set of convolutional layers is applied with residual connection blocks. The last layer in the second level is upscaled and added to the layer that is downsampled. The last layer in the network is dense with a identity function. The activation functions in all convolutional layers are ReLU, except for the first convolution, which uses a linear activation function.

Initially we tried a simple deep feedforward convolutional network with residual connections (motivated by [HZR16]). Although such a simple model works well for DCDM, it requires a high number of layers, which results in higher training and inference times. We found that creating parallel layers of CNNs with downsampling reduced the number of layers required. In summary, our goal was to first identify the simplest network architecture that provided adequate accuracy for our target problems, and subsequently, we sought to make architectural changes to minimize training and inference time. We are interested in a more thorough investigation of potential network architectures, filter sizes, etc., to better characterize the tradeoff curves between accuracy and efficiency; as a first step in this direction, we included a brief ablation study in Appendix B.4.

Differing resolutions use differing numbers of convolutions, but the fundamental structure remains the same. More precisely, the number of residual connections is changed for different resolutions. For example, a 64^3 grid uses one residual block on the left, two on the right on the upper level, and three on the lower level. Furthermore, the weights trained on a lower

resolution grid can be used effectively with higher resolutions. Figure 4.4d shows convergence results for a 256^3 grid, using a model trained for a 64^3 grid and a 128^3 grid. The model that we use for 256^3 grids in our final examples was trained on a 128^3 grid; however, as the shown in the figure, even training with a 64^3 grid allows for efficient residual reduction. Table 4.1 shows results for three different resolutions, where DCDM uses 64^3 and 128^3 trained models. Since we can use the same weights trained over a 64^d domain and/or 128^d domain, the number of parameters does not depend on the spatial fidelity. It depends on d for the kernel size.

4.3.3 Training

Using the procedure explained in Section 4.3.1, we create the training dataset $\mathcal{D} \in \text{span}(\mathbf{A}^{\text{train}}) \cap \mathcal{S}^{n-1}$ of size 20,000 generated from 10,000 Rayleigh-Ritz vectors. $\mathcal{S}^{n-1}$ is the unit sphere, i.e., all training vectors are scaled to have unit length. We train our model with TensorFlow [AAB15] on a single NVIDIA RTX A6000 GPU with 48GB memory. Training is done with standard deep learning techniques—more precisely, back-propagation and the ADAM optimizer [KB15] (with starting learning rate 0.0001). Training takes approximately 10 minutes and 1 hour per epoch for grid resolutions 64^3 and 128^3, respectively. We trained our model for 50 epochs; however, the model from the thirty-first epoch was optimal for 64^3, and the model from the third epoch was optimal for 128^3.

4.4 Results and Analysis

We demonstrate DCDM on three increasingly difficult examples and provide numerical evidence for the efficient convergence of our method. All examples were run on a workstation with dual stock AMD EPYC 75F3 processors, and an NVIDIA RTX A6000 GPU with 48GB memory. The grid resolutions we evaluate are the same as used in e.g. [TSS17] and are common for graphics papers.

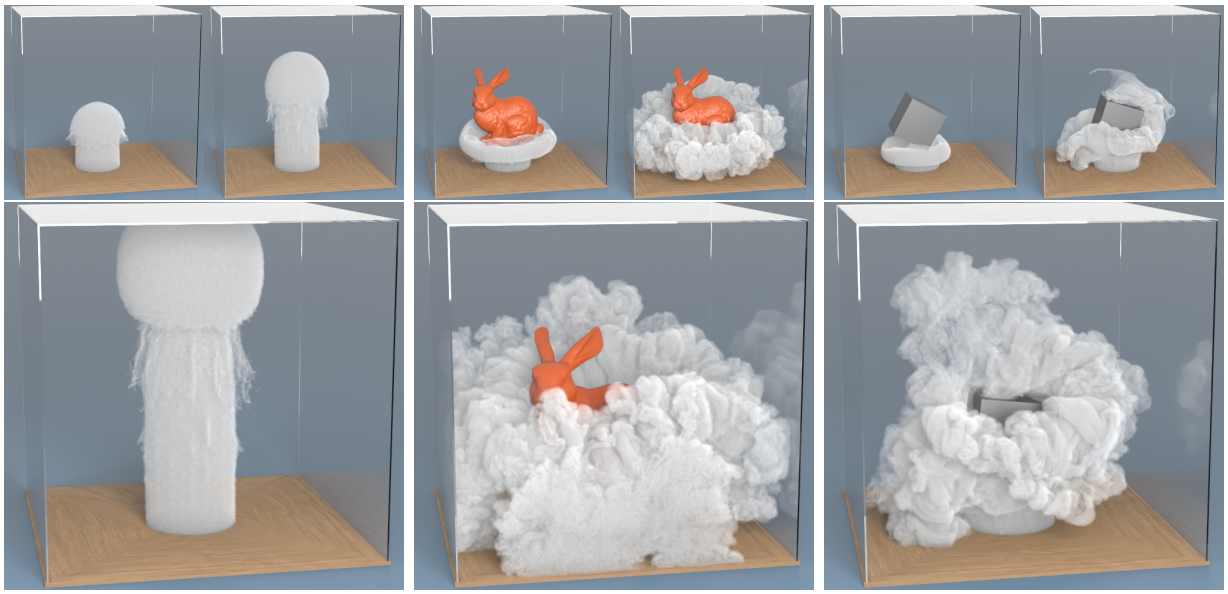


Figure 4.3: DCDM for simulating a variety of incompressible flow examples. Left: smoke plume at $t = 6.67, 13.33, 20$ seconds. Middle: smoke passing a bunny at $t = 5, 10, 15$ seconds. Right: smoke passing a spinning box (time-dependent Neumann boundary conditions) at $t = 2.67, 6, 9.33$ seconds.

Figure 4.3 showcases DCDM for incompressible smoke simulations. In each simulation, inlet boundary conditions are set in a circular portion of the bottom of the cubic domain, whereby smoke flows around potential obstacles and fills the domain. We show a smoke plume (no obstacles), flow past a complex static geometry (the Stanford bunny), and flow past a dynamic geometry (a rotating cube). Visually plausible and highly-detailed results are achieved for each simulation (see supplementary material for larger videos). The plume example uses a computational grid with resolution 128^3, while the other two uses grids with resolution 256^3 (representing over 16 million unknowns). For each linear solve, DCDM was run until the residual was reduced by four orders of magnitude[2]. In our experience,

[2]Computer graphics experts have found that solving Poisson equations until a four orders-of-magnitude reduction in residual is achieved is enough for visual realism (any further computational effort does not yield easily perceptible differences) [MST10, PGG23].

production-grade solvers (e.g., 3D smoke simulators for movie visual effects) use resolutions of 128^3 or more, and as computing resources improve we are seeing more problems solved at huge scales like 512^3 and above, where a learning-enhanced method like DCDM will have a more dramatic impact.

For the bunny example, Figures 4.4a–b demonstrate how residuals decrease over the course of a linear solve, comparing DCDM with other methods. Figure 4.4a shows the mean results (with standard deviations) over the course of 400 simulation frames, while in Figure 4.4b, we illustrate behavior on a particular frame (frame 150). For FluidNet, we use the optimized implementation provided by [flu22]. This implementation includes pre-trained models that we use without modification. In both subfigures, it is evident that the FluidNet residual never changes, since the method is not iterative; FluidNet reduces the initial residual by no more than one order of magnitude. On the other hand, with DCDM, we can continually reduce the residual (e.g., by four orders of magnitude) as we apply more iterations of our method, just as with classical CG. In Figure 4.4b, we also visualize the convergence of three other classical methods, CG, Deflated CG [SYE00], and incomplete Cholesky preconditioned CG (ICPCG); clearly, DCDM reduces the residual in the fewest number of iterations (e.g., approximately one order of magnitude fewer iterations than ICPCG). Since FluidNet is not iterative and lacks a notion of residual reduction, we treat $\mathbf{r}_0$ for FluidNet as though an initial guess of zero is used (as is done in our solver).

To clarify these results, Table 4.1 reports convergence statistics for DCDM compared to standard iterative techniques, namely, CG, Deflated CG, and ICPCG. For all 64^3, 128^3, and 256^3 grids with the bunny example, we measure the time t_r and the number of iterations n_r required to reduce the initial residual on a particular time step of the simulation by four orders of magnitude. DCDM achieves the desired results in by far the fewest number of iterations at all resolutions. At 256^3, DCDM performs approximately 6 times faster than CG, suggesting a potentially even wider performance advantage at higher resolutions. Inference is the dominant cost in an iteration of DCDM; the other linear algebra computations in an iter-

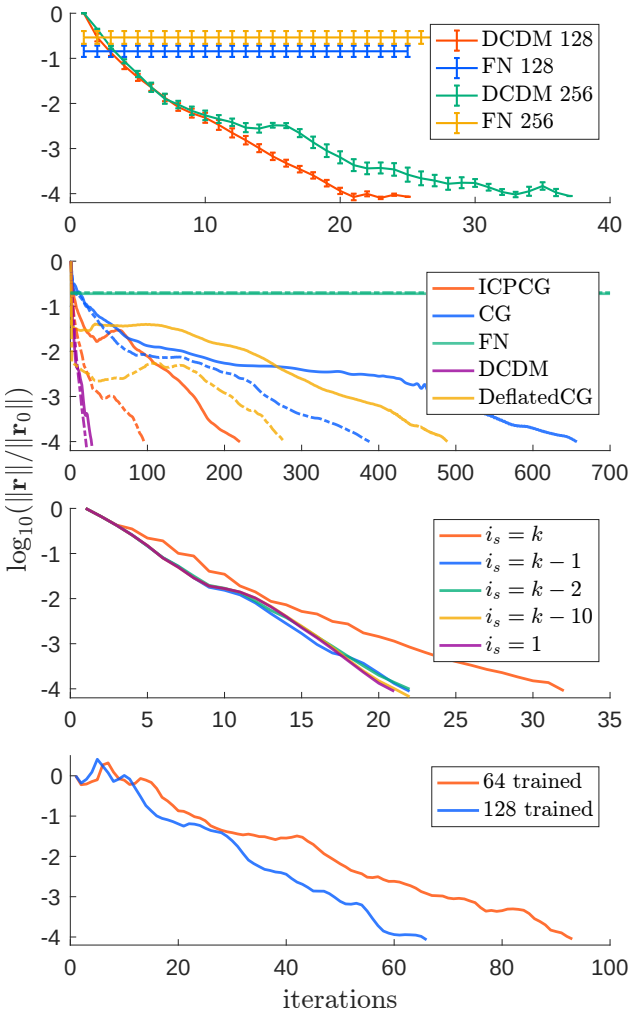


Figure 4.4: Convergence data for the bunny example (see also Table 4.1). **(a)** Mean and std. dev. (over all 400 frames in the simulation) of residual reduction during linear solves (with 128^3 and 256^3 grids) using FluidNet (FN) and DCDM. **(b)** Residual plots with ICPCG, CG, FN, DCDM, and Deflated CG at frame 150. Dashed and solid lines represent results for 128^3 and 256^3, respectively. **(c)** Decrease in residuals with varying degrees of **A**-orthogonalization ($i_s = i_{\text{start}}$) in the 128^3 case. **(d)** Reduction in residuals when the network is trained with a 64^3 or 128^3 grid for the 256^3 grid simulation shown in Figure 4.3 Middle.

ation of DCDM are comparable to those in CG. The nice result of our method is that despite the increased time per iteration, the number of required iterations is reduced so drastically

that DCDM materially outperforms classical methods like CG. Although ICPCG successfully reduces number of iterations (Figure 4.4b), we found the runtime to scale prohibitively with grid resolution. We used SciPy's [VGO20] `sparse.linalg.spsolve_triangular` function for forward and back substitution in our ICPCG implementation, and we also used a precomputed $\mathbf{L}$ that is not accounted for in the table results (though this took no more than 4 seconds at the highest resolution); Appendix B.3 includes further details on ICPCG.

Notably, even though Deflated CG and DCDM are based on approximate Ritz vectors, DCDM performs far better, indicating the value of using a neural network.

We performed three additional sets of tests. First, we tried low resolutions, 16^3 and 32^3, which are such small problems that we would expect CG to win due to the relatively high overhead of evaluating a neural network: indeed, DCDM and CG take 0.377sec/15iter and 0.008sec/48iter at 16^3, respectively, and 0.717sec/16iter and 0.063sec/53iter at 32^3. Note that we used the model (and parameters) tailored for 64^3 resolution to obtain these results; a lighter model, trained specifically for 16^3 and 32^3 resolutions, would give better timings, though likely still behind CG. Second, we tested cases where d = 2, at resolutions 256^2 and 512^2. For this setup, running the smoke plume test (2D analogue of 4.3 Left) at 256^2, DCDM and CG take 2.18sec/64iter and 0.59sec/536iter, respectively. Again, since the system for this resolution is much smaller than those reported in Table 4.1, we expect CG to be more efficient. However, at 512^2, the system is big enough where we actually do outperform CG in time as well: 3.87sec/126iter for DCDM vs. 5.60sec/1146iter for CG. Third, we performed comparisons between DCDM and a more recent work, [SSH19]. Since [SSH19] requires many asymptotically expensive computations, we expected a significant performance advantage with DCDM. For the 256^2 smoke plume example, using matrices from frame 10 of the simulation, [SSH19] requires 1024 iterations for convergence (15.41s), vs. only 50 for DCDM (1.50s).

Method	64^3 Grid			128^3 Grid			256^3 Grid		
	t_r	n_r	tp_r	t_r	n_r	tp_r	t_r	n_r	tp_r
DCDM-64	2.71s	**16**	0.169s	**22s**	27	0.814 s	**261s**	58	4.50s
DCDM-128	5.37s	19	0.283 s	26s	**25**	1.083s	267s	**44**	6.07s
CG	**1.77**s	168	0.0105s	26s	465	0.0559s	1548s	1046	1.479s
Deflated CG	771.6s	117	6.594s	3700s	277	13.357s	21030s	489	43.00s
ICPCG	164s	43	3.813s	2877s	94	30.60s	54714s	218	250.98s

Table 4.1: Timing and iteration comparison for different methods on the bunny example. t_r, n_r and tp_r represents time, iteration and time per iteration. DCDM-{64,128} calls a model whose parameters are trained over a $\{64^3, 128^3\}$ grid. All computations are done using only CPUs; model inference does not use GPUs. All implementation is done in Python. See Appendix B.3 for convergence plots.

4.5 Conclusions

We presented DCDM, incorporating CNNs into a CG-style algorithm that yields efficient, convergent behavior for solving linear systems. Our method effectively acts as a preconditioner, albeit a nonlinear one[3]. Our method is evaluated on linear systems with over 16 million degrees of freedom and converges to a desired tolerance in merely tens of iterations. Furthermore, despite training the underlying network on a single domain (per resolution) without obstacles, our network is able to successfully predict search directions that enable efficient linear solves on domains with complex and dynamic geometries. Moreover, the training data for our network does not require running fluid simulations or solving linear systems ahead of time; our Rayleigh-Ritz vector approach enables us to quickly generate

[3]Algebraically, any preconditioner P is attempting to learn an inverse of A^Ω, which is equivalent to what DCDM achieves for purposes of CG (learning the action of the inverse of the matrix on a vector $\mathbf{x}$). We initially tried learning a low-rank linear preconditioner, but our explorations were not successful; the approach was not efficient for higher resolutions because it required a large k.

very large training datasets, unlike other works. We release our code, data, and pre-trained models so users can immediately apply DCDM to Poisson systems without further dataset generation or training, especially due to the feasibility of pre-trained weights for inference at different grid resolutions: `https://github.com/ayano721/2023_DCDM`.

Our network was designed for and trained exclusively using data related to the discrete Poisson matrix, which likely limits the generalizability of our present *model*. However, we believe our *method* is readily applicable to other classes of PDEs (or general problems with graph structure) that give rise to large, sparse, symmetric linear systems. To that end, we briefly applied DCDM to matrices arising from discretized heat equations (a similar class of large, sparse matrices; hence expected to work well with DCDM). We found that we can achieve convergence (reducing the initial residual by four orders of magnitude) using DCDM trained only on Poisson matrices—even though our test heat equation used Dirichlet boundary conditions, unlike the Neumann boundary conditions used with the Poisson equation systems we solved before. For a heat equation matrix at $N = 64$, DCDM can converge in only 14 iterations. Future work will extend this analysis. We note that our method is unlikely to work well for matrices that have high computational cost to evaluate $\mathbf{A} * x$ (e.g., dense matrices), since training relies on efficient $\mathbf{A} * \mathbf{x}$ evaluations. An interesting question is how well our method and current models would apply to discrete Poisson matrices arising from non-uniform grids, e.g., quadtrees or octrees [LGF04].

CHAPTER 5

Primal Extended Position Based Dynamics for Hyperelasticity

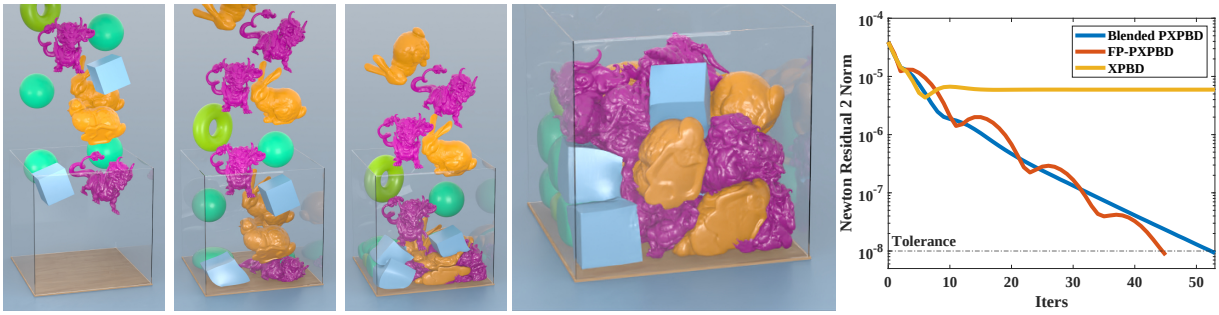


Figure 5.1: **30 Objects Dropping (left)**. Our Blended PXPBD (B-PXPBD) approach robustly handles large elastic deformations. **FEM Residual Comparison (right)**. B-PXPBD and FP-PXPBD reduce the backward Euler residual while XPBD stagnate in a representative step of a hyperelsaticity simulation.

5.1 Methods

5.1.1 Equations

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

$$\mathbf{M}\frac{\partial^2 \mathbf{x}}{\partial t^2} = -\frac{\partial PE}{\partial \mathbf{x}} + \mathbf{f}^{ext}, \; PE(\mathbf{x}) = \sum_e \Psi(\mathbf{F}^e(\mathbf{x}))V^e. \tag{5.1}$$

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

$$\mathbf{F}^e_{\alpha\beta}(\mathbf{x}) = \sum_i x_{i\alpha} \frac{\partial N_i}{\partial \mathbf{X}_\beta}(\mathbf{X}^e) \tag{5.2}$$

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.

5.1.1.1 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]

$$\Psi^{cor}(\mathbf{F}) = \mu|\mathbf{F} - \mathbf{R}(\mathbf{F})|_F^2 + \frac{\lambda}{2}(\det(\mathbf{F}) - 1)^2. \tag{5.3}$$

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

$$PE(\mathbf{x}) = \sum_c \frac{1}{2} C_c(\mathbf{x}) \frac{1}{a_c} C_c(\mathbf{x}) \tag{5.4}$$

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

$$\hat{C}_1(\mathbf{F}) = |\mathbf{F} - \mathbf{R}(\mathbf{F})|_F, \hat{C}_2(\mathbf{F}) = \det(\mathbf{F}) - 1. \tag{5.5}$$

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 5.3). Here the potential is

$$\Psi^{aniso}(\mathbf{F}) = \Psi^{cor} + \frac{\sigma_{max}}{\lambda_{ofl}}(f_a + \alpha_{act} f_p) \tag{5.6}$$

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]. We refer the reader to the supplemental technical document (Appendix C) for a detailed description of these terms.

5.1.1.2 Implicit Time Stepping

We consider both backward Euler and quasistatic time stepping schemes

$$\mathbf{M}\left(\frac{\frac{\mathbf{x}^{n+1}-\mathbf{x}^n}{\Delta t} - \mathbf{v}^n}{\Delta t}\right) = -\frac{\partial PE}{\partial \mathbf{x}}(\mathbf{x}^{n+1}) + \mathbf{f}^{ext}. \tag{5.7}$$

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 (5.7) 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 (5.7), however we omit the explicit representation of this for concise exposition.

5.1.2 XPBD

Macklin et al. [MMC16] solve Equation (5.7) 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

$$\Delta t^2 \frac{\partial PE}{\partial x_{i\alpha}} = -\sum_c \frac{\partial C_c}{\partial x_{i\alpha}}(\mathbf{x})\lambda_c \tag{5.8}$$

where they introduce $\lambda_c = -\frac{\Delta t^2}{a_c} C_c$ as an additional unknown which converts Equation (5.7) into the system

$$\mathbf{g}(\mathbf{x}^{n+1}, \boldsymbol{\lambda}) = \mathbf{M}\left(\mathbf{x}^{n+1} - \tilde{\mathbf{x}}\right) - \sum_c \lambda_c^T \nabla \mathbf{C}_c(\mathbf{x}^{n+1}) = \mathbf{0} \tag{5.9}$$

$$\mathbf{h}(\mathbf{x}^{n+1}, \boldsymbol{\lambda}) = \mathbf{C}(\mathbf{x}^{n+1}) + \frac{\mathbf{A}}{\Delta t^2}\boldsymbol{\lambda} = \mathbf{0}. \tag{5.10}$$

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

$$\begin{pmatrix} \mathbf{M} + \sum_c \lambda_{ck} \frac{\partial^2 C_c}{\partial \mathbf{x}^2}(\mathbf{x}_k^{n+1}) & -\nabla C_c^T(\mathbf{x}_k^{n+1}) \\ \nabla \mathbf{C}(\mathbf{x}_k^{n+1}) & \frac{\mathbf{A}}{\Delta t^2} \end{pmatrix} \begin{pmatrix} \Delta \mathbf{x}_{k+1} \\ \Delta \boldsymbol{\lambda}_{k+1} \end{pmatrix} = - \begin{pmatrix} \mathbf{g}(\mathbf{x}_k^{n+1}, \boldsymbol{\lambda}_k) \\ \mathbf{h}(\mathbf{x}_k^{n+1}, \boldsymbol{\lambda}_k) \end{pmatrix}. \tag{5.11}$$

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

$$\left(\mathbf{C}^T(\mathbf{x}_k^{n+1})\mathbf{M}^{-1}\mathbf{C}(\mathbf{x}_k^{n+1}) + \frac{\mathbf{A}}{\Delta t^2}\right)\Delta \boldsymbol{\lambda}_{k+1} = -\mathbf{h}(\mathbf{x}_k^{n+1}, \boldsymbol{\lambda}_k) \tag{5.12}$$

$$\Delta \mathbf{x}_{k+1} = \mathbf{M}^{-1}\nabla \mathbf{C}(\mathbf{x}_k^{n+1})\Delta \boldsymbol{\lambda}_{k+1}. \tag{5.13}$$

Furthermore, Equation (5.12) is updated in a Gauss-Seidel fashion where the d^{th} Lagrange multiplier is updated via

$$\Delta \tilde{\lambda}_{k+1d} = \frac{-h_d(\mathbf{x}_k^{n+1}, \lambda_{kd})}{\nabla C_d^T(\mathbf{x}_k^{n+1})\mathbf{M}^{-1}\nabla C_d(\mathbf{x}_k^{n+1}) + \frac{a_d}{\Delta t^2}\lambda_{kd}}, \quad \lambda_{k+1d} = \lambda_{kd} + \Delta \tilde{\lambda}_{k+1d}. \tag{5.14}$$

Note that we distinguish $\Delta \tilde{\lambda}_{k+1d}$ in Equation (5.14) from from $\Delta \lambda_{k+1d}$ in Equation (5.12) 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 (5.13) to create

$$\mathbf{x}_{k+1}^{n+1} = \mathbf{x}_{k}^{n+1} + \mathbf{M}^{-1}\nabla\mathbf{C}_d(\mathbf{x}_{k}^{n+1})\Delta\tilde{\lambda}_{k+1d}. \tag{5.15}$$

The system (Equations (5.9)-(5.10)) is then re-linearized (Equation (5.11)) and the process (Equations (5.14)-(5.15)) is repeated iteratively.

5.1.3 Primary residual XPBD (PXPBD)

The motivation for the omission of the residual and constraint Hessian terms (red) in Equation (5.11) is natural. The constraint Hessian is non-diagonal and its retention would preclude the decoupling of primary variables from the Lagrange multipliers in Equation (5.12). 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 5.2(a)). Unfortunately, we also notice that inclusion of this term can cause XPBD to lose its favorable stability properties (see Figure 5.2(b)). We note though that if the global system in Equation (5.11) 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 (5.7) since the system size is larger with the inclusion of the $\boldsymbol{\lambda}$ unknowns.

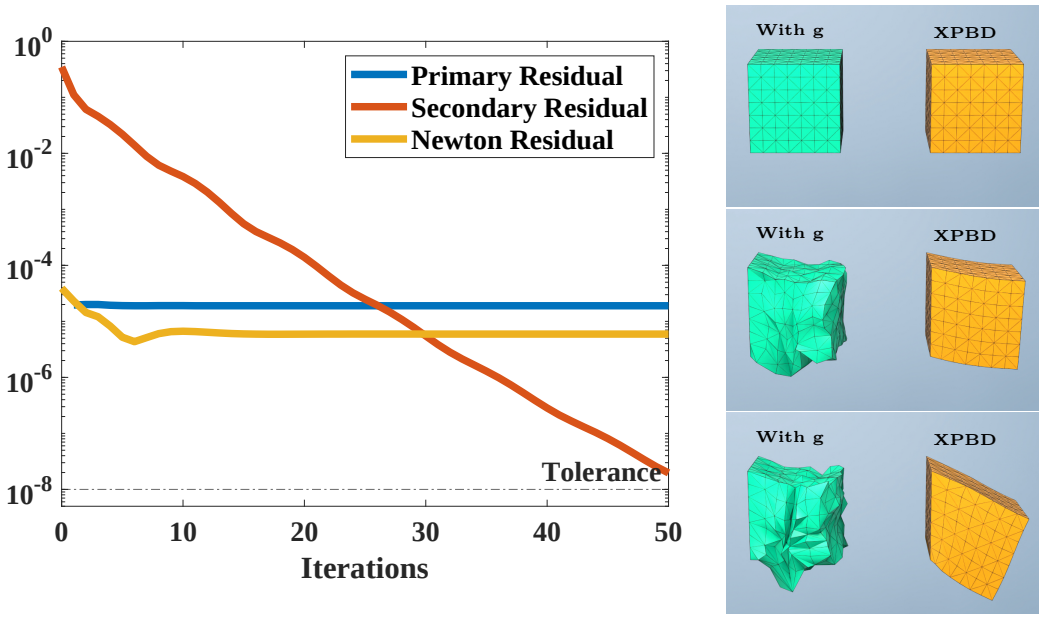


Figure 5.2: **(a) Primal Residual Comparison: Stagnation**. While XPBD reliably reduces the secondary residual, its omission of the primary residual in the linearization causes its primary residual to stagnate, making its true (Newton) residual stagnate as well. **(b) Primal Residual Inclusion: Instability**. XPBD is unstable when the primal residual term is not omitted.

5.1.3.1 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 (5.9) 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 (5.14). 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 Equations (5.9)-(5.10) 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 (5.9) 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 (5.9) 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 (5.7). 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 (5.14)), 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 (5.14)

$$\mathbf{x}_{k+1}^{n+1} = \tilde{\mathbf{x}} + \sum_c \lambda_{kc}\mathbf{M}^{-1}\nabla\mathbf{C}_c(\mathbf{x}_k^{n+1}) + \mathbf{M}^{-1}\nabla\mathbf{C}_d(\mathbf{x}_k^{n+1})\Delta\tilde{\lambda}_{k+1d}. \tag{5.16}$$

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 (5.13). 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 (5.13). We remedy this by taking a linear combination of the updates in Equations (5.13) and (5.16)

$$\mathbf{x}_{k+1}^{n+1} = \zeta\left(\mathbf{M}^{-1}\nabla C_d(\mathbf{x}_k^{n+1})\Delta\tilde{\lambda}_{k+1d}\right) + (1-\zeta)\Delta\mathbf{x}_{k+1}^{fp} + \mathbf{x}_k^{n+1} \tag{5.17}$$

$$\Delta\mathbf{x}_{k+1}^{fp} = \tilde{\mathbf{x}} + \sum_c \lambda_{kc}\mathbf{M}^{-1}\nabla C_c(\mathbf{x}_k^{n+1}) + \Delta\tilde{\lambda}_{k+1d}\mathbf{M}^{-1}\nabla C_d(\mathbf{x}_k^{n+1}) - \mathbf{x}_k^{n+1}. \tag{5.18}$$

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.

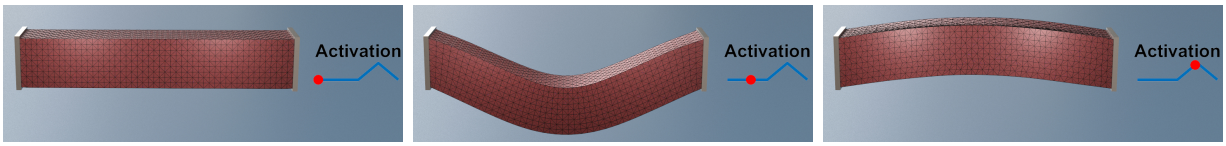


Figure 5.3: **Muscle Box Activation**. A rectangular bar with both ends clamped falls under gravity. Two seconds later, the muscle box is activated and contracts along the horizontal direction. The level of activation is shown on the right side of the images. $t = 0.0333, 1.2, 2.9$ seconds are shown in the footage.

5.1.3.2 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 5.7. In a general FEM-discretized hyperelastic formulation (see Barbič and Sifakis [SB12]), the potential energy gradient has the expression

$$\frac{\partial PE}{\partial x_{i\alpha}}(\mathbf{x}) = \sum_{e,\beta,\gamma} P_{\beta\gamma}(\mathbf{F}^e(\mathbf{x}))\delta_{\alpha\beta}\frac{\partial N_i}{\partial X_\gamma}(\mathbf{X}^e)V^e \tag{5.19}$$

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 (5.8). With this formulation, an analogous method consists of

$$\mathbf{g}(\mathbf{x}^{n+1}, \mathbf{P}^{n+1}) = \mathbf{M}\left(\mathbf{x}^{n+1} - \tilde{\mathbf{x}}\right) + \Delta t^2 \mathbf{B}\mathbf{P} = \mathbf{0} \tag{5.20}$$

$$\mathbf{h}^e(\mathbf{x}^{n+1}, \mathbf{P}^{n+1}) = \frac{\partial \Psi}{\partial \mathbf{F}}(\mathbf{F}^e(\mathbf{x}^{n+1})) - \mathbf{P}^e = \mathbf{0}. \tag{5.21}$$

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 (5.9) does have this

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 (5.25);
        2. Compute $\mathbf{b}^e_{k+1l}$ via Equation 5.28;
        3. Compute $\delta\mathbf{F}^e_{k+1l}$ via Equation 5.30 with the approximation in Equation 5.33;
        4. Compute $\delta\mathbf{x}^e_{k+1l}$ as in Equation 5.26 ;
        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

dependence, and it is precisely this issue that leads to the red terms in the linearization in Equation (5.11). Therefore, a formulation based on Equations (5.20) and (5.21) rather than Equations (5.9) and (5.10) 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 (5.20) and (5.21) 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 (5.20) to the secondary

Equation (5.21). 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 (5.20) to define the current guess for $\mathbf{x}_k^{n+1}$. We update this guess iteratively by solving for the positions in element e that satisfy Equation (5.21). 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

$$\mathbf{M}^e(\mathbf{x}_{k+1}^{n+1,e} - \tilde{\mathbf{x}}^e) + \Delta t^2 \mathbf{B}^e \mathbf{P}_{k+1}^e = \mathbf{f}^e \tag{5.22}$$

$$\frac{\partial \Psi}{\partial \mathbf{F}}(\mathbf{F}^e(\mathbf{x}_{k+1}^{n+1})) - \mathbf{P}_{k+1}^e = \mathbf{0} \tag{5.23}$$

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 5.22 can be written as:

$$\sum_{j^e,\beta} m_{i^e j^e} \delta_{\alpha\beta} \left(x_{k+1 j^e \beta}^{n+1} - \tilde{x}_{j^e\beta} \right) + \Delta t^2 \sum_{\gamma,\delta} B_{i^e\alpha\gamma\delta}^e P_{k+1\gamma\delta}^e = f_{i^e\alpha k}^e \tag{5.24}$$

Here Equation (5.23) 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 (5.23)-(5.24) can be rewritten as

$$\mathbf{M}^e(\mathbf{x}_{k+1}^{n+1,e} - \tilde{\mathbf{x}}^e) + \Delta t^2 \mathbf{B}^e \frac{\partial \Psi}{\partial \mathbf{F}}(\mathbf{F}^e(\mathbf{x}_{k+1}^{n+1,e})) - \mathbf{f}^e = \mathbf{0} \tag{5.25}$$

where $M_{i^e\alpha j^e\beta}^e = m_{i^e j^e}\delta_{\alpha\beta}$ is the element-wise mass matrix 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 $B_{i^e\alpha\gamma\delta}^e = \delta_{\alpha\gamma}\frac{\partial N_i}{\partial X_\delta}(\mathbf{X}^e)V^e$ from Equation (5.19).

We use Netwon's method to solve Equation (5.25). $\mathbf{x}_{i^e k+1 l}^{n+1,e}$ 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 need to solve 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:

$$\delta \mathbf{x}^e_{i^e k+1l} = \delta \mathbf{F}^e_{k+1l}(\mathbf{X}^e_{i^e} - \mathbf{X}^e_{\text{com}}) + \mathbf{b}^e_{k+1l} \tag{5.26}$$

where $\delta \mathbf{F}^e_{k+1l}$ are the distortional degrees of freedom in the element, $\mathbf{b}^e_{k+1l}$ are the translational degrees of freedom and $\mathbf{X}^e_{\text{com}}$ is the center of mass of the element in the undeformed configuration. We refer the readers to the technical document [YYJ23] for details. Similarly, $\mathbf{X}^e_{i^e}$ refer to the undeformed positions of the vertices in the element. The tensor $D^e_{i^e\alpha\epsilon\tau} = \delta_{\alpha\epsilon}(X^e_{i^e\tau} - X^{e\tau}_{\text{com}})$ relates these variables with $\delta x^e_{i^e\alpha k+1l} = \sum_{\epsilon,\tau} D^e_{i^e\alpha\epsilon\tau}\delta F^e_{\epsilon\tau k+1l} + b^e_{\alpha k+1l}$. It can be shown that with this choice of basis, the translational degrees of freedom decouple, reducing the dimensionality of the system to be solved. That is, by introducing the linearization

$$\frac{\partial \Psi}{\partial \mathbf{F}}(\mathbf{F}^e(\mathbf{x}^{n+1,e}_{k+1l+1})) \approx \frac{\partial^2 \Psi}{\partial \mathbf{F}^2}(\mathbf{F}^e(\mathbf{x}^{n+1,e}_{k+1l})) : \delta \mathbf{F}^e_{k+1l} + \mathbf{P}^e_{k+1l} \tag{5.27}$$

we can solve for the translational component as

$$\mathbf{b}^e_{k+1l} = \frac{-1}{m^e} \sum_{i^e \in \Omega^e} \mathbf{g}_{i^e k+1l} \tag{5.28}$$

$$g_{i^e\alpha k+1l} = m_{i^e}(\mathbf{x}^{n+1}_{i^e\alpha k+1l} - \tilde{\mathbf{x}}_{i^e\alpha}) - f^e_{i^e\alpha k} + \Delta t^2 \sum_{\gamma,\delta} B^e_{i^e\alpha\gamma\delta} P^e_{k+1l\gamma\delta} \tag{5.29}$$

where $\mathbf{P}^e_{k+1l} = \frac{\partial \Psi}{\partial \mathbf{F}}(\mathbf{F}^e(\mathbf{x}^{n+1,e}_{k+1l}))$. Furthermore, once the translational component $\mathbf{b}^e_l$ is determined from Equation (5.28), the distortional degrees of freedom $\delta \mathbf{F}^e_l$ can be determined from the linear system

$$\left(\tilde{\mathbf{M}}^e + \Delta t^2 V^e \frac{\partial^2 \Psi}{\partial \mathbf{F}^2}(\mathbf{F}_e(\mathbf{x}^{n+1}_{k+1l}))\right) \delta \mathbf{F}^e_{k+1l} = -\mathbf{G}^e_{k+1l} \tag{5.30}$$

where

$$G^e_{\eta\nu k+1l} = \sum_{i^e,\alpha} D^e_{i^e\alpha\eta\nu}\left(g_{i^e\alpha k+1l} + m_{i^e} b^e_{\alpha k+1l}\right) \tag{5.31}$$

$$\tilde{M}^e_{\eta\nu\epsilon\tau} = \sum_{i^e,j^e \in \Omega^3} D^e_{i^e\alpha\eta\nu} m_{i^e j^e} D^e_{j^e\alpha\epsilon\tau}. \tag{5.32}$$

Note that the choice of the center of mass for the center of the translation as well as the partition of unity/reproduction of linear functions properties of the FEM interpolation functions are the keys to this decoupling. Also 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]. Derivation details are provided in the supplemental technical document (Appendix C).

5.1.3.3 Quasi-Newton

In general, solving Equation (5.30) 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

$$\frac{\partial^2\Psi}{\partial\mathbf{F}^2}(\mathbf{F}_e) \approx 2\mu\mathbf{I} + \lambda\frac{\partial\det(\mathbf{F}^e)}{\partial\mathbf{F}^e} \otimes \frac{\partial\det(\mathbf{F}^e)}{\partial\mathbf{F}^e} \tag{5.33}$$

where $\mathbf{I}$ is the 9×9 (4×4 in 2D) identity matrix (see Appendix C for more detail). With this approximation, we can use the Sherman-Morrison formula for

$$\left(\tilde{\mathbf{M}}^e + \Delta t^2 V^e \frac{\partial^2\Psi}{\partial\mathbf{F}^2}(\mathbf{F}_e(\mathbf{x}^{n+1}_{k+1l}))\right)^{-1} \approx \mathbf{Z}^e - \frac{\mathbf{Z}^e\,(\mathbf{W}^e \otimes \mathbf{W}^e)\,\mathbf{Z}^e}{1 + \mathbf{W}^e : (\mathbf{Z}^e\mathbf{W}^e)} \tag{5.34}$$

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 5.1.3.3 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 (5.20)-(5.21). 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 5.1.3.3, 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 5.3.1, 5.3.2, 5.3.3 and 5.3.4 the speed of FP-PXPBD is comparable to XPBD.

5.2 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. For the grid-based variation mentioned in Section 5.3.5, 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. The grid-based variation requires coloring at the beginning of every time step. For more detailed descriptions and illustrations we refer the readers to [YYJ23]. We note that Fratarcangeli et al. [FVP16] develop a randomized and effective ordering technique that could be used here as well.

Table 5.1: 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 5.2(b))	4K	17K	200ms	200ms	216ms	40	40	40
Equal Budget Comparison (Figure 5.4)	33K	149K	210ms	210ms	200ms	7	7	5
XPBD Hyperelastic (Figure 5.5)	4K	17K	22ms	-	44ms	4	-	4
XPBD Neohookean (Figure 5.6)	4K	17K	795ms	-	345ms	400	-	40
Simple Muscle (Figure 5.3)	5k	20k	-	-	160ms	-	-	4

5.3 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 5.1. Examples run with the corotated model (Equation 5.3) 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 ρ. We denote $\mathcal{I}$ to be the set of elements that contain node i. We define $m_i = \sum_{e \in \mathcal{I}} \frac{V^e \rho}{n_e}$, where $n_e = 4$ in 3D and 3 in 2D. 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 $\mu = \frac{E}{2(1+\nu)}, \lambda = \frac{E\nu}{(1+\nu)(1-2\nu)}$. For all the examples in this paper we set Poisson ratio $\nu = 0.3$ and density $\rho = 10$.

5.3.1 Residual Comparison

Residual reduction between XPBD, B-PXPBD and FP-PXPBD is compared. 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. XPBD effectively reduces the auxiliary residual, but not the primary residual. The example setup is the same as the one shown in Figure 5.2(b). B-PXPBD has blending parameter $\zeta = 0.5$.

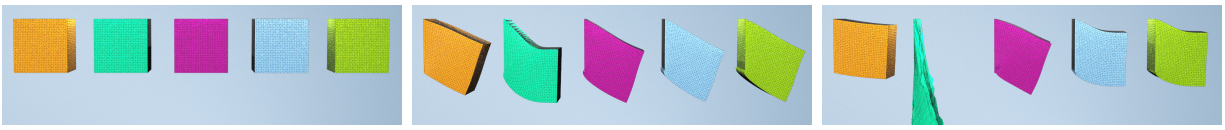


Figure 5.4: **Equal Budget Comparison**. From left to right: Newton (converged), Newton, FP-PXPBD, B-PXPBD, XPBD. With a limited budget, XPBD-style methods are stable, whereas the Newton's method suffers from instability. Frame 0, 10, 60 are shown in the figure.

5.3.2 Equal Budget Comparison

In Figure 5.4 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 (5.7) 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 $\zeta = 0.1$.

5.3.3 XPBD Hyperelastic

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 (5.5). As demonstrated in Figure 5.5, 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 $\Delta t = 0.01$.

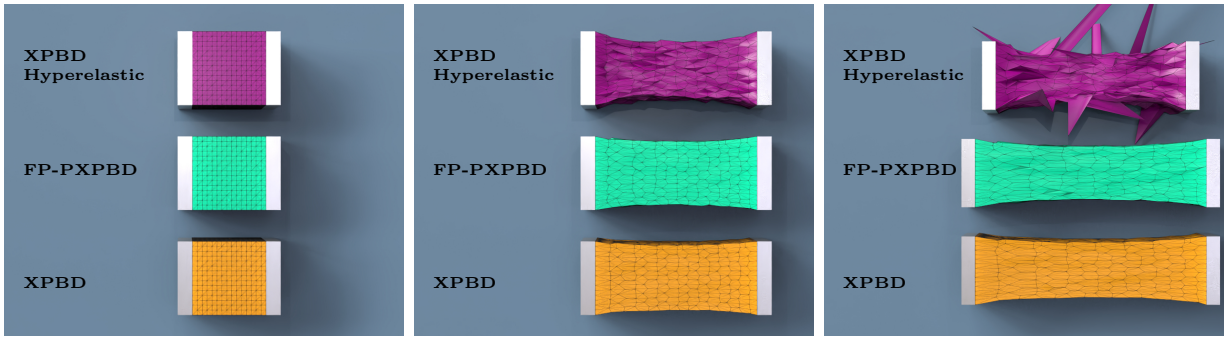


Figure 5.5: **XPBD Hyperelastic**. Defining the XPBD constraint as the square root of the hyperelastic potential is not stable (top). Results at frame 0, 10, 30 are shown.

5.3.4 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 5.33 to this model as $\frac{\partial^2 \Psi}{\partial \mathbf{F}^2}(\mathbf{F}_e) \approx \mu \mathbf{I} + \lambda \frac{\partial \det(\mathbf{F}^e)}{\partial \mathbf{F}^e} \otimes \frac{\partial \det(\mathbf{F}^e)}{\partial \mathbf{F}^e}$. Similarly, we can approximate the Hessian inverse as in Equation 5.34 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 5.3.3. We use Young's modulus $E = 1000$ and time step $\Delta t = 0.01$. Results are shown in Figure 5.6. 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.

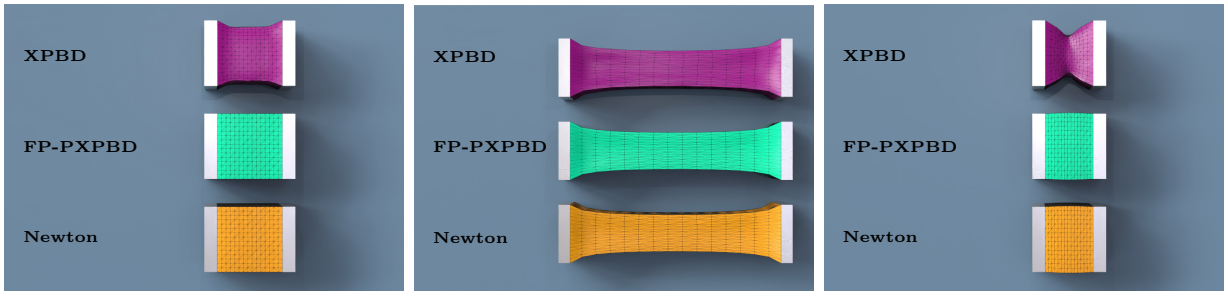
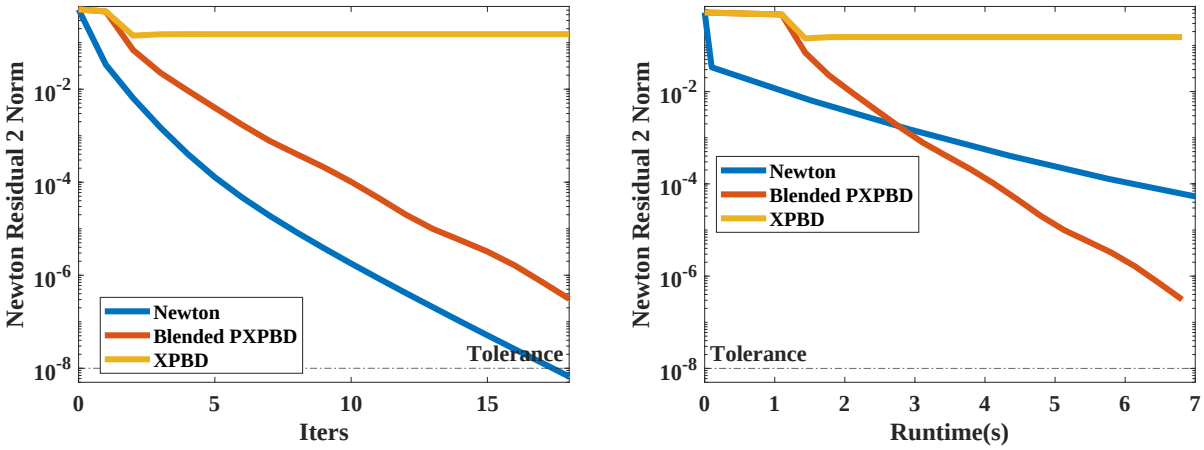


Figure 5.6: **XPBD Neohookean**. XPBD is less volume-conserving than FP-PXPBD when the cube is squeezed. Results at frame 1, 25, 52 are shown.



(a) (b)

Figure 5.7: **(a) Grid-Based Residual vs. Iterations**. Newton's method and B-PXPBD reliably reduce the residual, but XPBD stagnates. **(b) Grid-based Residual vs. Runtime**. 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 faster convergence than Newton's method.

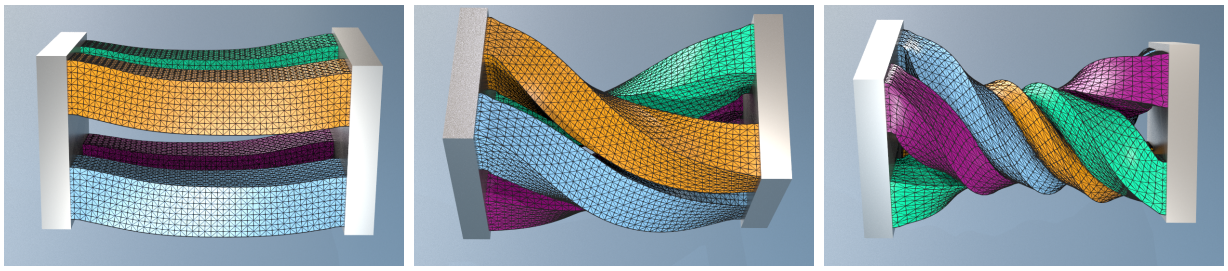


Figure 5.8: **Four Bars Twisting**. Grid-based B-PXPBD is capable of handling large deformation and complex collisions.

5.3.5 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. The backward Euler degrees of freedom are over a regular grid where the tetrahedron mesh is embedded/interpolated from its motion. We use B-PXPBD to solve the system of equations for implicit time stepping outlined in Jiang et al. [JSS15], but the energy is written in the XPBD way using the constraints Equation (5.5). This requires a modification to the coloring strategy used for parallel implementation (see Appendix C for specifics) but is otherwise a straightforward application of our techniques so we omit explicit detail.

Figure 5.9: **Muscle**. Large-scale muscle simulation using grid-based B-PXPBD. Frames 30, 60, 140 are shown. ©2023 Epic Games, Inc

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 5.7. 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 5.7 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 5.8, 5.10 and Figure 5.9. For these examples, the Young's modulus is $E = 1000$ and CFL number is .4. B-PXPBD has blending parameter $\zeta = 0.5$.

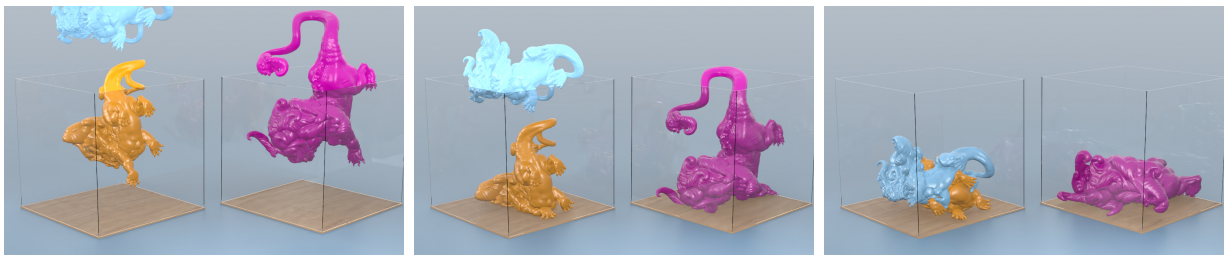


Figure 5.10: **Dropping Dragons**. Grid-based simulation with B-PXPBD exhibits many collision-driven large deformations.

5.4 Discussion and Limitations

Our framework effectively addresses XPBD convergence issues with hyperelasticity and allows for generalization to arbitrary constitutive models. 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. It also requires a larger deviation from an existing XPBD code. In practice, these considerations must be weighed when deciding which approach to use.

CHAPTER 6

Position-Based Nonlinear Gauss-Seidel for Quasistatic Hyperelasticity

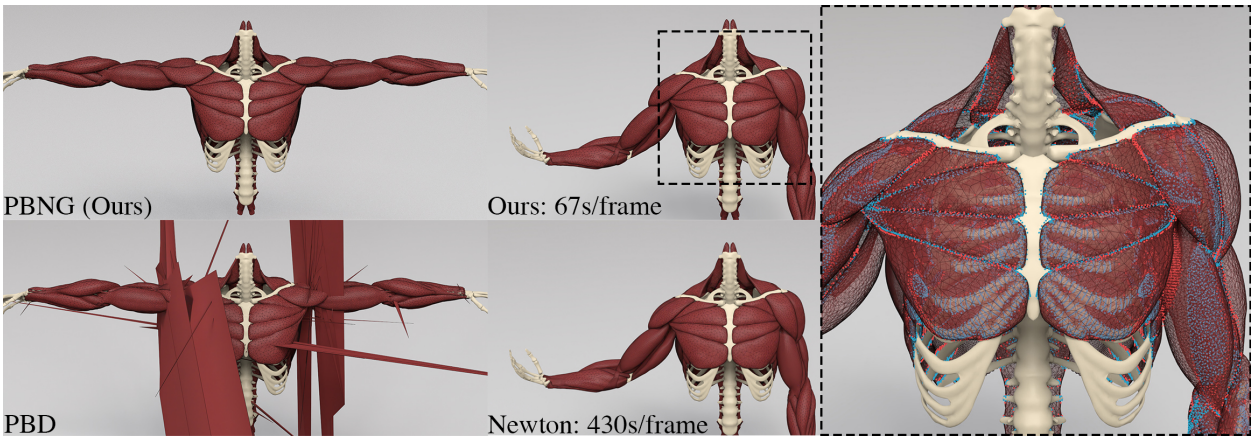


Figure 6.1: **Quasistatic Muscle Simulation with Collisions**. Our method (PBNG) produces high-quality results visually comparable to Newton's method but with a 6x speedup. In this hyperelastic simulation of muscles, we use weak constraints to bind muscles together and resolve collisions. The rightmost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

6.1 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

$$R^0 \frac{\partial^2 \boldsymbol{\phi}}{\partial t^2} = \nabla^{\mathbf{X}} \cdot \mathbf{P} + \mathbf{f}^{\text{ext}} \tag{6.1}$$

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

$$\boldsymbol{\phi}(\mathbf{X}, t) = \mathbf{x}_D(\mathbf{X}, t), \ \mathbf{X} \in \partial\Omega_D^0 \tag{6.2}$$

$$\mathbf{P}(\mathbf{X}, t)\mathbf{N}(\mathbf{X}, t) = \mathbf{T}_N(\mathbf{X}, t), \ \mathbf{X} \in \partial\Omega_N^0 \tag{6.3}$$

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

$$\mathbf{P}(\mathbf{X}, t) = \frac{\partial \Psi}{\partial \mathbf{F}}(\frac{\partial \boldsymbol{\phi}}{\partial \mathbf{X}}(\mathbf{X}, t)), \ \text{PE}(\boldsymbol{\phi}(\cdot, t)) = \int_{\Omega^0} \Psi(\frac{\partial \boldsymbol{\phi}}{\partial \mathbf{X}}) d\mathbf{X} \tag{6.4}$$

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

In quasistatic problems, the inertial terms in the momentum balance (Equation (6.1)) can be neglected and the material motion is defined by a sequence of equilibrium problems

$$\mathbf{0} = \nabla^{\mathbf{X}} \cdot \mathbf{P} + \mathbf{f}^{\text{ext}} \tag{6.5}$$

subject to the boundary conditions in Equations (6.2)-(6.3). This is equivalent to the minimization problems

$$\boldsymbol{\phi}(\cdot,t) = \underset{\boldsymbol{\Upsilon} \in \mathcal{W}^t}{\operatorname{argmin}} \ \mathrm{PE}(\boldsymbol{\Upsilon}) - \int_{\Omega_0} \mathbf{f}^{\mathrm{ext}} \cdot \boldsymbol{\Upsilon} d\mathbf{X} - \int_{\partial\Omega_N^0} \mathbf{T}_N \cdot \boldsymbol{\Upsilon} ds(\mathbf{X}) \tag{6.6}$$

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

6.1.1 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]

$$\Psi^{\mathrm{cor}}(\mathbf{F}) = \mu|\mathbf{F} - \mathbf{R}(\mathbf{F})|_F^2 + \frac{\lambda}{2}(\det(\mathbf{F}) - 1)^2. \tag{6.7}$$

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

$$\Psi^{\mathrm{nh}}(\mathbf{F}) = \frac{1}{2}\mu|\mathbf{F}|_F^2 + \frac{\hat{\lambda}}{2}(\det(\mathbf{F}) - 1 - \frac{\mu}{\hat{\lambda}})^2. \tag{6.8}$$

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

$$\mu = \frac{E}{2(1+\nu)},\ \lambda = \frac{E\nu}{(1+\nu)(1-2\nu)}. \tag{6.9}$$

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 6.6. We also support

the stable Neo-Hookean model proposed in [SGK18]

$$\Psi^{\text{snh}}(\mathbf{F}) = \frac{1}{2}\mu(|\mathbf{F}|_F^2 - d) + \frac{1}{2}(\det(\mathbf{F}) - 1 - \frac{3\mu}{4\lambda})^2 - \frac{1}{2}\mu\log(1 + |\mathbf{F}|_F^2). \tag{6.10}$$

Figure 6.2: **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.

6.2 Discretization

We use the FEM discretization of the quasistatic problem in Equation (6.5)

$$\mathbf{f}_i(\mathbf{x}^{n+1}) + \hat{\mathbf{f}}_i^{\text{ext}} = \mathbf{0}, \;\; \mathbf{X}_i \notin \Omega_D^0 \tag{6.11}$$

$$\mathbf{x}_i^{n+1} = \mathbf{x}_D(\mathbf{X}_i, t^{n+1}), \;\; \mathbf{X}_i \in \Omega_D^0. \tag{6.12}$$

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

are given as

$$\mathbf{f}_i(\mathbf{y}) = -\frac{\partial \hat{\mathrm{PE}}}{\partial \mathbf{y}_i}(\mathbf{y}) \tag{6.13}$$

$$\hat{\mathrm{PE}}(\mathbf{y}) = \hat{\mathrm{PE}}^{\Psi}(\mathbf{y}) + \hat{\mathrm{PE}}^{\mathrm{wc}}(\mathbf{y}) \tag{6.14}$$

$$\hat{\mathrm{PE}}^{\Psi}(\mathbf{y}) = \sum_{e=0}^{N^E-1} \Psi\Big(\sum_{j=0}^{N^N-1} \mathbf{y}_j \frac{\partial N_j^e}{\partial \mathbf{X}}\Big) V_e^0 \tag{6.15}$$

$$\hat{\mathbf{f}}_i^{\mathrm{ext}} = \int_{\Omega^0} \mathbf{f}^{\mathrm{ext}} N_i d\mathbf{X} + \int_{\partial\Omega_N^0} \mathbf{T}_N N_i ds(\mathbf{X}) \tag{6.16}$$

where $\hat{\mathrm{PE}}^{\Psi} : \mathbb{R}^{dN^N} \to \mathbb{R}$ is the discretization of the potential energy, $\sum_{j=0}^{N^N-1} \mathbf{y}_j \frac{\partial N_j^e}{\partial \mathbf{X}}$ is the deformation gradient induced by nodal positions $\mathbf{y} \in \mathbb{R}^{dN^N}$ in tetrahedron ($d = 3$) or triangle ($d = 2$) element e with $0 \leq e < N^E$, $\frac{\partial N_i^e}{\partial \mathbf{X}}$ is the derivative of the interpolating function in element e (which is constant since we use piecewise linear interpolation) and V_e^0 is the measure of the element. We refer the reader to [BW08, SB12] for more detail on the FEM derivation of potential energy terms in a hyperelastic formulation. Also, note that we add another term to the discrete potential energy $\hat{\mathrm{PE}}^{\mathrm{wc}} : \mathbb{R}^{dN^N} \to \mathbb{R}$ in Equation (6.14) to account for self-collisions and similar weak constraints (see Section 6.2.1). Similar to the non-discrete case, the constrained minimization problem

$$\mathbf{x}^{n+1} = \underset{\mathbf{y} \in \mathcal{W}_{\Delta x}^{n+1}}{\operatorname{argmin}} \ \hat{\mathrm{PE}}(\mathbf{y}) - \mathbf{y} \cdot \hat{\mathbf{f}}^{\mathrm{ext}} \tag{6.17}$$

where $\mathcal{W}_{\Delta x}^{n+1} = \left\{ \mathbf{y} \in \mathbb{R}^{dN^N} \,|\mathbf{y}_i = \mathbf{x}_D(\mathbf{X}_i, t^{n+1}),\ \mathbf{X}_i \in \partial\Omega_D^0 \right\}$ is equivalent to Equations (6.11)-(6.12).

6.2.1 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

$$\hat{\mathrm{PE}}^{\mathrm{wc}}(\mathbf{y}) = \frac{1}{2}\sum_{c=0}^{N^{\mathrm{wc}}-1} \mathbf{C}_c(\mathbf{y})^T \mathbf{K}_c \mathbf{C}_c(\mathbf{y}) \tag{6.18}$$

$$\mathbf{C}_c(\mathbf{y}) = \sum_{j=0}^{N^N-1} w_{0j}^c \mathbf{y}_j - w_{1j}^c \mathbf{y}_j. \tag{6.19}$$

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 $\sum_{j=0}^{N^N-1} w_{0j}^c \mathbf{y}_j$ and $\sum_{j=0}^{N^N-1} w_{1j}^c \mathbf{y}_j$. The stiffness of the constraint is represented in the matrix $\mathbf{K}_c$. This can allow for anisotropic responses where $\mathbf{K}_c = k_n \mathbf{n}\mathbf{n}^T + k_\tau \left(\boldsymbol{\tau}_0 \boldsymbol{\tau}_0^T + \boldsymbol{\tau}_1 \boldsymbol{\tau}_1^T\right)$. Here $\mathbf{n}^T \boldsymbol{\tau}_i = 0$, $i = 0, 1$ and k_n is the stiffness in the $\mathbf{n}$ direction while k_τ is the stiffness in response to the motion in the plane normal to $\mathbf{n}$. In the case of an isotropic constraint ($k_c = k_n = k_\tau$), we use the scalar k_c in place of $\mathbf{K}_c$ since $\mathbf{K}_c = k_c \mathbf{I}$ is diagonal. We note that, in most of our examples, the anisotropic model is used for collision constraints where $\mathbf{n}$ is the collision constrain direction (see Section 6.8.2).

6.3 Gauss-Seidel Notation

Our approach, PBD and XPBD all use nonlinear Gauss-Seidel to iteratively improve an approximation to the solution $\mathbf{x}^{n+1} \in \mathbb{R}^{dN^N}$ of Equation (6.11). We use l to denote the l^{th} Gauss-Seidel iteration $\mathbf{x}^{n+1,l} \approx \mathbf{x}^{n+1}$. During the course of one iteration, degrees of freedom in the approximate solution will be updated in sub-iterates which we denote as $\mathbf{x}_{(k)}^{n+1,l}$ with $0 \leq k < N^{\mathrm{GS}}$. Here $\mathbf{x}_{(0)}^{n+1,l} = \mathbf{x}^{n+1,l}$ and $\mathbf{x}_{(N^{\mathrm{GS}}-1)}^{n+1,l} = \mathbf{x}^{n+1,l+1}$. For example, with PBD/XPBD, in the k^{th} sub-iterate, the nodes in the k^{th} constraint will be projected/solved for. In our position-based approach, in the k^{th} sub-iterate, only a single node i_k will be updated. It is important to introduce this notation, since unlike with Jacobi-based approaches, the update

of the k^{th} sub-iterate will depend on the contents of the $k-1^{\text{th}}$ sub-iterate.

6.4 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 (6.1)

$$\sum_{j=0}^{N^N-1} m_{ij} \left(\frac{\mathbf{x}_j^{n+1} - 2\mathbf{x}_j^n + \mathbf{x}_j^{n-1}}{\Delta t^2} \right) = \mathbf{f}_i(\mathbf{x}^{n+1}) + \mathbf{f}_i^{\text{ext}},\ \mathbf{X}_i \notin \Omega_D^0. \tag{6.20}$$

Here $m_{ii} = \int_{\Omega^0} R^0 N_i d\mathbf{X}$ and $m_{ij} = 0,\ j \neq i$ are entries in the mass matrix. However, they require that the discrete potential energy in Equation (6.15) is of the form

$$\hat{PE}^{\Psi}(\mathbf{y}) = \sum_{c=0}^{2N^E-1} \frac{1}{2\alpha_c} C_c^2(\mathbf{y}),\ \mathbf{y} \in \mathbb{R}^{dN^E}. \tag{6.21}$$

For example, this can be done with the energy densities in Equations (6.7) and (6.8) using two constraints $c = 2e$ and $c = 2e+1$ per element e

$$C_{2e}^{\text{cor}}(\mathbf{y}) = |\mathbf{F}^e(\mathbf{y}) - \mathbf{R}(\mathbf{F}^e(\mathbf{y}))|_F,\ C_{2e+1}^{\text{cor}}(\mathbf{y}) = \det(\mathbf{F}^e(\mathbf{y})) - 1 \tag{6.22}$$

$$C_{2e}^{\text{nh}}(\mathbf{y}) = |\mathbf{F}^e(\mathbf{y})|_F,\ C_{2e+1}^{\text{nh}}(\mathbf{y}) = \det(\mathbf{F}^e(\mathbf{y})) - 1 - \frac{\mu}{\hat{\lambda}}. \tag{6.23}$$

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

To demonstrate the connection between Equation (6.20) and PBD, Macklin et al. [MMC16] develop XPBD. It is based on the total Lagrange multiplier formulation

$$\sum_{j=0}^{N^N-1} m_{ij} \left(\mathbf{x}_j^{n+1} - \hat{\mathbf{x}}_j \right) - \sum_{c=0}^{P-1} \frac{\partial C_c}{\partial \mathbf{x}_i}(\mathbf{x}^{n+1}) \lambda_c^{n+1} = 0,\ \mathbf{X}_i \notin \Omega_D^0 \tag{6.24}$$

$$C_c(\mathbf{x}^{n+1}) + \frac{\alpha_c}{\Delta t^2} \lambda_c^{n+1} = 0,\ 0 \leq c < P \tag{6.25}$$

where $\hat{\mathbf{x}}_j = 2\mathbf{x}_j^n - \mathbf{x}_j^{n-1} - \frac{\Delta t^2}{m_{jj}}\mathbf{f}_j^{\text{ext}}$ and $\boldsymbol{\lambda}^{n+1} \in \mathbb{R}^P$ is introduced as an additional unknown. The $\mathbf{x}^{n+1} \in \mathbb{R}^{dN^N}$ in Equations (6.24)-(6.25) is the same in the solution to Equation (6.20). Macklin et al. [MMC16] use a per-constraint Gauss-Seidel update of Equations (6.24)-(6.25)

$$\mathbf{x}_{i(k+1)}^{n+1,l} = \mathbf{x}_{i(k)}^{n+1,l} + \Delta\mathbf{x}_{i(k+1)}^{n+1,l},\ \mathbf{X}_i \notin \Omega_D^0 \tag{6.26}$$

$$\Delta\mathbf{x}_{i(k+1)}^{n+1,l} = \frac{\Delta\lambda_{(k+1)c_k}^{n+1,l}}{m_{ii}} \frac{\partial C_{c_k}}{\partial \mathbf{x}_i}(\mathbf{x}_{(k)}^{n+1,l}) \tag{6.27}$$

$$\Delta\lambda_{(k+1)c_k}^{n+1,l} = \frac{-C_{c_k}(\mathbf{x}_{(k)}^{n+1,l}) + \frac{\alpha_{c_k}}{\Delta t^2} C_{c_k}(\mathbf{x}_{(k)}^{n+1,l})}{\sum_{j=0}^{N^N-1} \frac{1}{m_{jj}} \sum_{\beta=0}^{d-1} \left(\frac{\partial C_{c_k}}{\partial x_{j\beta}}(\mathbf{x}_{(k)}^{n+1,l})\right)^2 + \frac{\alpha_{c_k}}{\Delta t^2}}. \tag{6.28}$$

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.

6.4.1 Quasistatics

As noted by Macklin et al. [MMC16], the XPBD update in Equations (6.26)-(6.28) 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 (6.5)), albeit with the external forcing terms omitted. More precisely, define $\boldsymbol{\phi}_s$ to be a solution of the problem

$$sR^0 \frac{\partial^2 \boldsymbol{\phi}_s}{\partial t^2} = \nabla^{\mathbf{X}} \cdot \mathbf{P} + s\mathbf{f}^{\text{ext}}. \tag{6.29}$$

subject to the same boundary conditions in Equations (6.2)-(6.3). This is equivalent to scaling the α_c that would appear in Equation (6.1) (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 (6.29) vanish as $s \to 0$, it is clear then that the original PBD formulation generates an approximation to the solution of a quasistatic problem with the external forcing $\mathbf{f}^{\text{ext}}$ omitted. Note that PBD does include the external

forcing term in its initial guess $\mathbf{x}_i^{n+1} = \mathbf{x}_i^n + \Delta t \mathbf{v}_i^n + \frac{\Delta t^2}{m_{ii}} \mathbf{f}_i^{\text{ext}}$. However, the effect of the initial guess vanishes as the iteration count is increased. We demonstrate this in Section 6.8.4 Also, note that this is not the case in the XPBD formulation where $\alpha_c > 0$.

Unfortunately, XPBD cannot be trivially modified to run quasistatic problems. For example, omitting the mass terms on the left-hand side of Equation (6.24) makes the Gauss-Seidel update in Equations(6.26)-(6.28) impossible since there would be a division by zero. The simplest fix for quasistatic problems we can conceive of in the PBD framework is to use XPBD run to steady state using a pseudo-time iteration. This prevents the need for scaling the α_c which inherently removes the external forcing terms and does not introduce a divide by zero in Equation (6.27). However, this is very costly since each quasistatic time step is essentially the cost of an entire XPBD simulation. Nevertheless, we compare our approach against this option (see Section 6.8.4) since it will at least allow for the correct representation of the forcing terms. We refer to this technique as XPBD-QS.

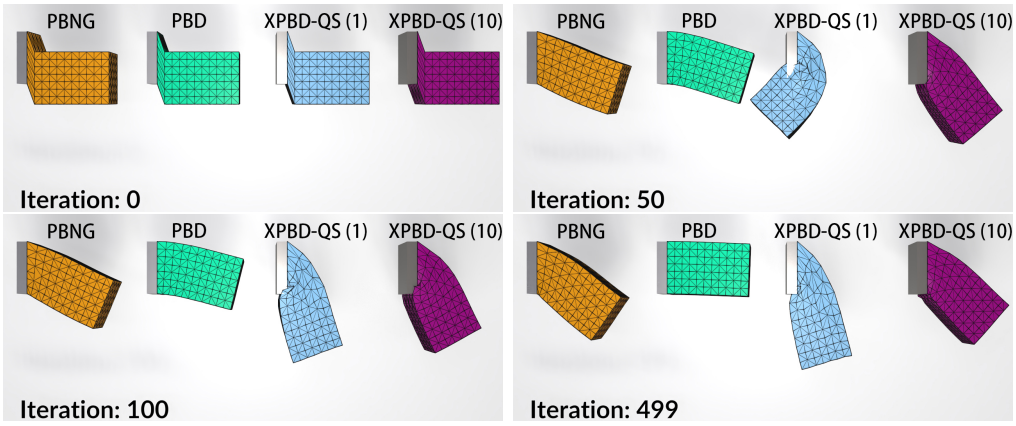


Figure 6.3: **Bar under Gravity**. A quasistatic simulation of a bar bending under gravity using different methods. The effect of external forcing vanishes in the PBD example as the number of iterations increases. More local iterations of XPBD-QS produces better results. PBNG converges to visually plausible results within fewer iterations than XPBD-QS.

6.4.2 XPBD Convergence

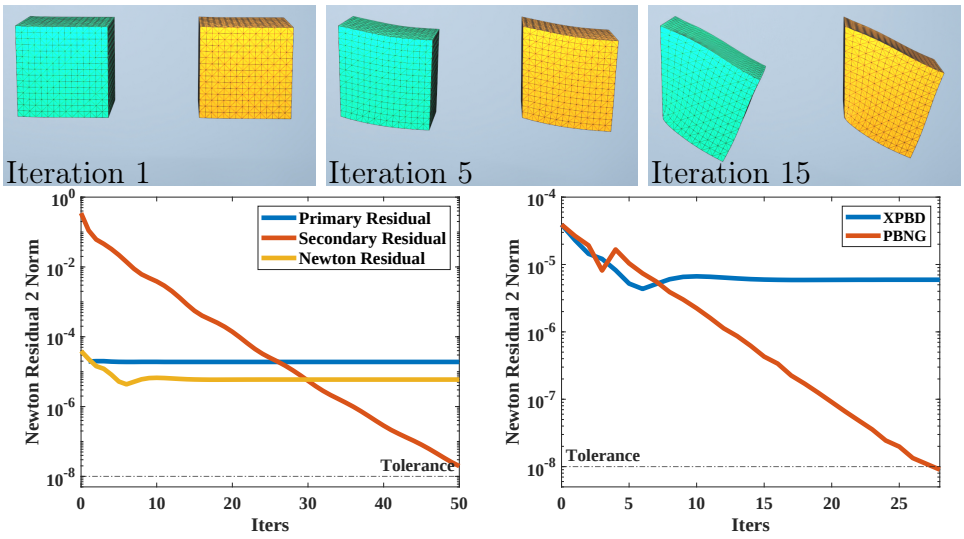
The linear system in Equations (6.24)-(6.25) that forms the basis for the XPBD Gauss-Seidel update is generated from the omission of two terms in their Lagrange multiplier formulation (shown in red)

$$\begin{pmatrix} \mathbf{M} + \sum_c \lambda_{ck} \frac{\partial^2 C_c}{\partial \mathbf{x}^2}(\mathbf{x}_k^{n+1}) & -\nabla C_c^T(\mathbf{x}_k^{n+1}) \\ \nabla \mathbf{C}(\mathbf{x}_k^{n+1}) & \frac{\mathbf{A}}{\Delta t^2} \end{pmatrix} \begin{pmatrix} \Delta \mathbf{x}_{k+1} \\ \Delta \boldsymbol{\lambda}_{k+1} \end{pmatrix} = \begin{pmatrix} \mathbf{g}(\mathbf{x}_k^{n+1}, \boldsymbol{\lambda}_k) \\ \mathbf{h}(\mathbf{x}_k^{n+1}, \boldsymbol{\lambda}_k) \end{pmatrix}. \tag{6.30}$$

The left-hand side term is second-order, but its omission is essential for the decoupling of position (primary) and Lagrange multipliers. However, the omission on the right-hand side is of the residual in the position (primary) equations. Without this term, residual reduction with XPBD stagnates after an iteration or two. However, the inclusion of this term leads to unstable behavior. We demonstrate this in Section 6.8.5 and Figure 6.4. We believe that the omission of the primary residual in the update causes iteration-dependent behavior and generally degraded convergence with XPBD since information about adjacent constraints would be included in this term if it could be stably added. Furthermore, the effect is more visually pronounced in quasistatic problems.

6.5 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 (6.11) using position-centric, rather than constraint-centric nonlinear Gauss-Seidel., This update takes into account each constraint that the position participates in. Visual intuition for this is illustrated in top of Figure 6.7(a). More specifically, in the k^{th} sub-iterate of iteration l, we modify a single node



(a) (b)

Figure 6.4: **Top**. Clamped blocks under gravity. The green block is XPBD, and the yellow one is PBNG. **(a) Primary 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) Convergence**. PBNG is able to reduce the Newton residual to the tolerance, whereas XPBD's residual stagnates.

i_k with $\mathbf{X}_{i_k} \notin \partial\Omega_D^0$ as

$$\mathbf{x}_{(k+1)i_k}^{n+1,l} = \mathbf{x}_{(k)i_k}^{n+1,l} + \Delta\mathbf{x}_{(k+1)i_k}^{n+1,l} \tag{6.31}$$

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

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

$$\mathbf{0} = \mathbf{f}_{i_k}(\mathbf{x}_{(k)}^{n+1,l} + \tilde{\mathbf{C}}^{i_k}\Delta\mathbf{x}_{(k+1)i_k}^{n+1,l}) + \hat{\mathbf{f}}_{i_k}^{\text{ext}}. \tag{6.32}$$

We use a single step of a modified Newton's method to approximate the solution of Equation (6.32) 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

$$\Delta \mathbf{x}^{n+1,l}_{(k+1)i_k} = \left(\mathbf{A}^{n+1,l}_{(k+1)i_k}\right)^{-1} \left(\mathbf{f}_{i_k}(\mathbf{x}^{n+1,l}_{(k)}) + \hat{\mathbf{f}}^{\text{ext}}_{i_k}\right). \tag{6.33}$$

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

6.5.1 Modified Hessian

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

$$\begin{aligned}\frac{\partial f_{i_k\alpha}}{\partial y_{i_k\beta}}(\mathbf{y}) = &- \sum_{e=0}^{N^E-1} \sum_{\delta,\gamma=0}^{d-1} \mathcal{C}^e_{\alpha\gamma\beta\delta}(\mathbf{y}) \frac{\partial N^e_{i_k}}{\partial X_\gamma} \frac{\partial N^e_{i_k}}{\partial X_\delta} V^0_e - \\ &\sum_{c=0}^{N^{\text{wc}}-1} \left(w^c_{0i_k} - w^c_{1i_k}\right)^2 K_{c\alpha\beta},\ 0 \le \alpha,\beta < d\end{aligned} \tag{6.34}$$

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

$$\mathbf{f}_{i_k}(\mathbf{y}) = -\sum_{e=0}^{N^E-1} \hat{\mathbf{P}}_e(\mathbf{y}) \frac{\partial N_{i_k}}{\partial \mathbf{X}}(\mathbf{X}^e) V^0_e - \sum_{c=0}^{N^{\text{wc}}-1} \left(w^c_{0i_k} - w^c_{1i_k}\right) \mathbf{K}_c \mathbf{C}_c(\mathbf{y}) \tag{6.35}$$

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

The primary cost in Equation (6.34) is the evaluation of the Hessian of the energy density $\mathcal{C}^e_{\alpha\gamma\beta\delta}(\mathbf{y})$ which is a symmetric fourth order tensor with $d^2 \times d^2$ entries. Furthermore, this tensor can be indefinite, which would complicate the convergence of the Newton procedure. We use a definiteness projection as in [TSI05] and [SGK19]. However we use a very simple symmetric positive definite approximation instead of their approaches which

require the singular value decomposition of the element deformation gradient $\sum_{j=0}^{N^N-1} \mathbf{y}_j \frac{\partial N_j^e}{\partial \mathbf{X}}$. Teran et al. [TSI05] also require the solution of a 3×3 and three 2×2 symmetric eigenvalue problems, our approach does not require this. Our the simple approximation is $\tilde{\mathcal{C}}^e_{\alpha\gamma\beta\delta}(\mathbf{y}) \approx \mathcal{C}^e_{\alpha\gamma\beta\delta}(\mathbf{y})$ with

$$\tilde{\mathcal{C}}^e_{\alpha\gamma\beta\delta}(\mathbf{y}) = 2\mu\delta_{\alpha\beta}\delta_{\gamma\delta} + \lambda JF^{e^{-1}}{}_{\alpha\gamma}(\mathbf{y}) JF^{e^{-1}}{}_{\beta\delta}(\mathbf{y}). \tag{6.36}$$

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

$$A^{n+1}_{(k+1)i_k\alpha\beta} = \sum_{e=0}^{N^E-1} \sum_{\delta,\gamma=0}^{d-1} \tilde{\mathcal{C}}^e_{\alpha\gamma\beta\delta}(\mathbf{x}^{n+1,l}_{(k)}) \frac{\partial N^e_{i_k}}{\partial X_\gamma} \frac{\partial N^e_{i_k}}{\partial X_\delta} V^0_e + \tag{6.37}$$

$$\sum_{c=0}^{N^{\mathrm{wc}}-1} \left(w^c_{0i_k} - w^c_{1i_k}\right)^2 K_{c\alpha\beta},\ 0 \le \alpha, \beta < d \tag{6.38}$$

6.5.2 Acceleration Techniques

PBNG is remarkably stable and gives visually plausible results when the computational budget is limited, but it is also capable of producing numerically accurate results as the budget is increased. However, as shown in Figure 6.5 and as with most Gauss-Seidel approaches, the convergence rate of PBNG may decrease with iteration count. We investigated two simple acceleration techniques to help mitigate this: the Chebyshev semi-iterative method (as in [Wan15]) and SOR (as in [FVP16]). The Chebyshev method uses the update

$$\mathbf{x}^{n+1,l+1} = \omega_{l+1}(\gamma(\mathbf{x}^{n+1,l+1}_{\mathrm{PBNG}} - \mathbf{x}^{n+1,l}) + \mathbf{x}^{n+1,l} - \mathbf{x}^{n+1,l-1}) + \mathbf{x}^{n+1,l-1} \tag{6.39}$$

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

The SOR method uses a similar, but simpler update

$$\mathbf{x}^{n+1,l+1} = \omega(\mathbf{x}^{n+1,l+1}_{\text{PBNG}} - \mathbf{x}^{n+1,l-1}) + \mathbf{x}^{n+1,l-1}. \tag{6.40}$$

We use $\omega = 1.7$ for this under-relaxation parameter. As shown in Figure 6.5, Chebyshev and SOR behave similarly in terms of residual reduction and visual appearance.

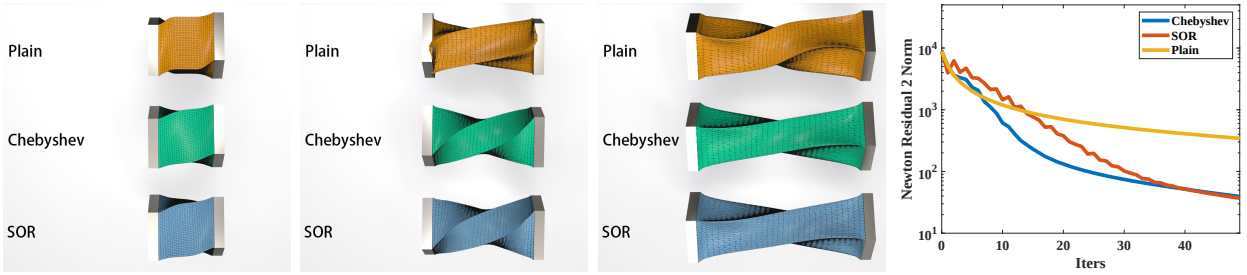


Figure 6.5: **Acceleration Techniques**. The convergence rate of PBNG may slow down as the iteration count increases. Chebyshev semi-iterative method and SOR effectively accelerate the Newton residual reduction.

6.6 Lamé Coefficients

The parameters of an isotropic constitutive model are often based on Lamé coefficients μ and λ which are themselves set from Young's modulus E and Poisson's ratio ν according to Equation (6.9). This relationship is based on the assumption of linear dependence of stress

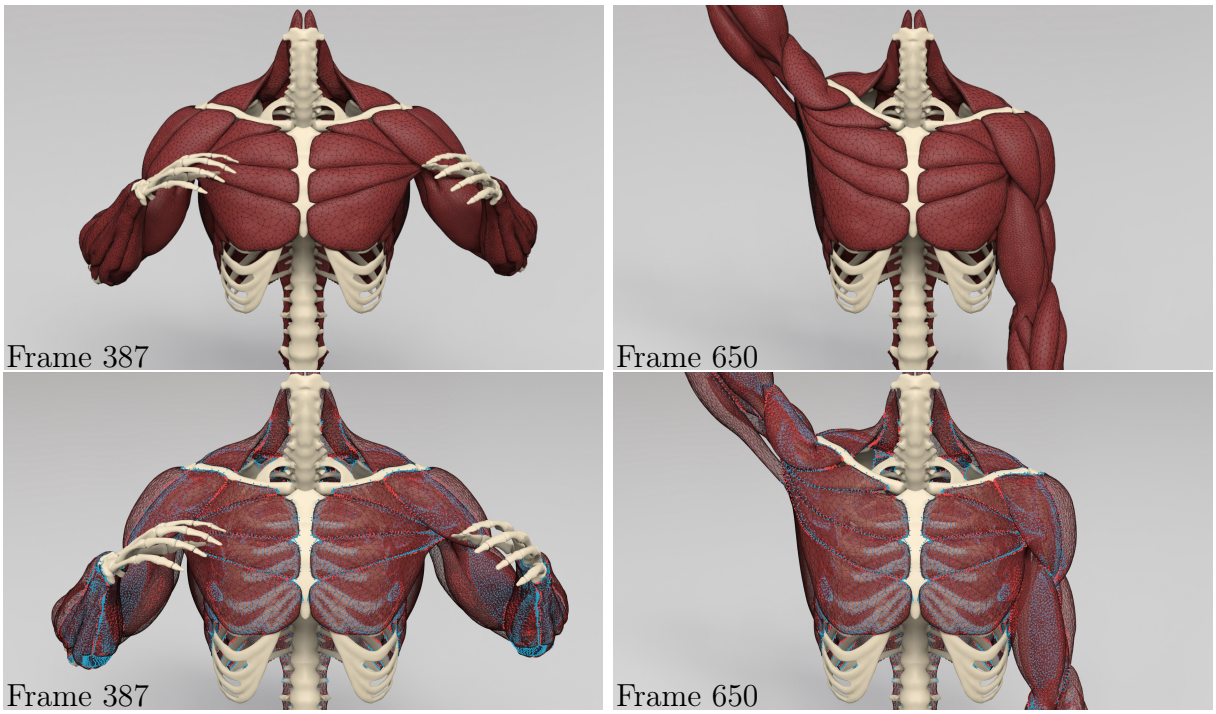


Figure 6.6: **PBNG Muscle Simulation**. The top row shows simulation results while the bottom row visualizes the vertex constraint status. *Red* indicates a vertex involved in contact, weak constraints are dynamically built to resolve the collisions. *Blue* represents the vertex positions of connective tissue bindings.

on strain, or quadratic potential energy density

$$\Psi^{\text{le}}(\mathbf{F}) = \mu \text{tr}(\boldsymbol{\epsilon}^2(\mathbf{F})) + \frac{\lambda}{2}\text{tr}(\boldsymbol{\epsilon}(\mathbf{F}))^2 \tag{6.41}$$

$$\boldsymbol{\epsilon} = \frac{1}{2}(\mathbf{F} + \mathbf{F}^T) - \mathbf{I}. \tag{6.42}$$

Furthermore, Equation (6.9) is derived from the model in Equation (6.41) 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 $I_\alpha : \mathbb{R}^{d\times d} \to \mathbb{R}$, $0 \leq \alpha < d$

$$I_0(\mathbf{F}) = \text{tr}(\mathbf{F}^T\mathbf{F}),\ I_1(\mathbf{F}) = \text{tr}((\mathbf{F}^T\mathbf{F})^2),\ I_2(\mathbf{F}) = \det(\mathbf{F}) \tag{6.43}$$

$$\Psi(\mathbf{F}) = \hat{\Psi}(I_0(\mathbf{F}), I_1(\mathbf{F}), I_2(\mathbf{F})). \tag{6.44}$$

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

$$\frac{\partial^2\Psi}{\partial\mathbf{F}^2} = \sum_{\alpha=0}^{d-1} \frac{\partial\hat{\Psi}}{\partial I_\alpha}\frac{\partial^2 I_\alpha}{\partial\mathbf{F}^2} + \sum_{\alpha,\beta=0}^{d-1} \frac{\partial^2\hat{\Psi}}{\partial I_\alpha \partial I_\beta}\frac{\partial I_\alpha}{\partial\mathbf{F}} \otimes \frac{\partial I_\beta}{\partial\mathbf{F}}. \tag{6.45}$$

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 (6.8). See the supplementary technical document (Appendix D) for derivation details.

We choose our approximate Hessian in Equation (6.36) based on this fact. That is, by omitting all but the first and last terms in Equation (6.45), our approximate Hessian is both symmetric positive definite and consistent with any model that is set from Lamé coefficients (e.g. from Young's modulus and Poisson's ratio)

$$\tilde{\mathcal{C}} = \mu\frac{\partial^2 I_0}{\partial\mathbf{F}^2} + \lambda\frac{\partial I_{d-1}}{\partial\mathbf{F}} \otimes \frac{\partial I_{d-1}}{\partial\mathbf{F}}. \tag{6.46}$$

Again, see the supplementary technical document (Appendix D) for more details.

6.7 Coloring and Parallelism

Parallel implementation of Gauss-Seidel techniques is complicated by data dependencies in the updates. This can be alleviated by careful ordering of sub-iterate position updates. We provide simple color-based orderings for both PBD and PBNG techniques. For PBD, colors are assigned to constraints so that those in the same color do not share incident nodes. Constraints in the same color can then be solved at the same time with no race conditions. For each vertex $\mathbf{x}_i$ in the mesh, we maintain a set $S_{\mathbf{x}_i}$ that stores the colors used by its incident constraints. For each constraint c, we find the minimal color as the least integer that is not contained in the set $\cup_{\mathbf{x}_i \in c} S_{\mathbf{x}_i}$. We then register the color by adding it into $S_{\mathbf{x}_i}$ for each $\mathbf{x}_i$ in constraint c. With PBNG, we color the nodes so that those in the same color do not share any mesh element or weak constraint. For each element or weak constraint c, we maintain a set S_c that stores the colors used by its incident nodes. For a position $\mathbf{x}_i$, we compute its color as the minimal one not contained in the set $\cup_{\mathbf{x}_i \in c} S_c$. Then we register the color by adding it into S_c for each element or weak constraint $\mathbf{x}_i$ is incident to. The coloring process is illustrated in Figures 6.7(b) and 6.7(c). We observe that coloring the nodes instead of the constraints gives fewer colors. This makes simulations run faster since more work can be done without race conditions. In Table 6.1, we demonstrate this performance observation. Note that we use the omp parallel directive from Intel's OpenMP library for parallelizing the updates.

6.7.1 Collision Coloring

For simulations with static weak constraints, the coloring is a one-time cost. Otherwise, the colors have to be updated every time the weak constraint structure changes, e.g. from self-collision (Figures 6.6 and 6.10). We propose a simple coloring scheme for this purpose: only nodes incident to the newly added weak constraints need recoloring. We first compute all nodes $\mathbf{x}_i^{\text{extra}}$ that are incident to newly added weak constraints. For each $\mathbf{x}_i^{\text{extra}}$, we compute

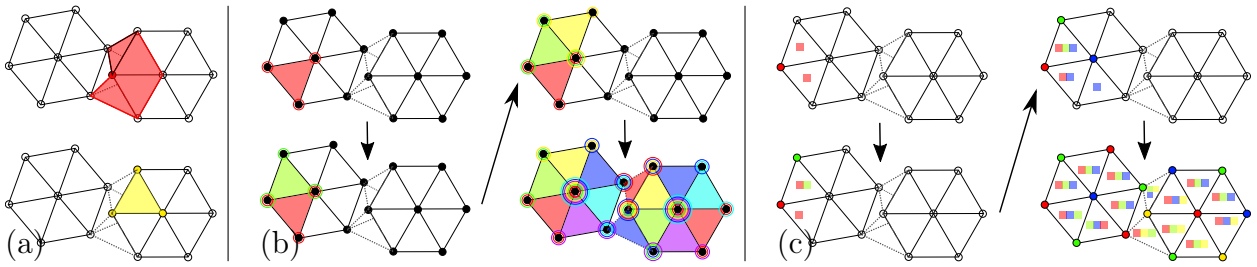


Figure 6.7: **(a) Dual Coloring** . Node based coloring (top) is contrasted with constraint based coloring (bottom). When a node is colored as red, its incident elements register red as used colors. When a constraint is colored yellow, its incident particles register yellow as used colors. **(b) Constraints-Based 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.

the used color set $\cup_{\mathbf{x}_i^{\text{extra}} \in c} S_c$. We use the color of $\mathbf{x}_i^{\text{extra}}$ from the previous time step as an initial guess. If it already exists in the used color set, then we find the minimal color that is not used. This is generally of moderate cost, e.g. in the muscle examples with collisions (Figures 6.1, 6.11 and 6.6), our algorithm takes less than 680ms/frame for recoloring, while the actual simulation takes a total of 67s to run.

6.8 Examples

We demonstrate the versatility and robustness of PBNG with a number of representative simulations of quasistatic (and dynamic) hyperelasticity. Examples run with the corotated model (Equation (6.7)) use the algorithm from [GFJ16] for its accuracy and efficiency. All

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

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

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 with 64 cores and 128 threads. In Table 6.3, we provide comprehensive performance statistics for PBNG. In Table 6.2, we provide runtime comparisons between PBNG and the other methods.

6.8.1 Stretching Block

We stretch and twist a simple block in a simple scenario. The block has 32K particles and 150K elements. Both ends of the block are clamped. They are stretched, squeezed and twisted in opposite directions. The block has $R^0 = 10$ and Young's modulus $E = 10^5$. There is no gravity. The simulation is quasistatic. We compare performance between Newton's

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

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

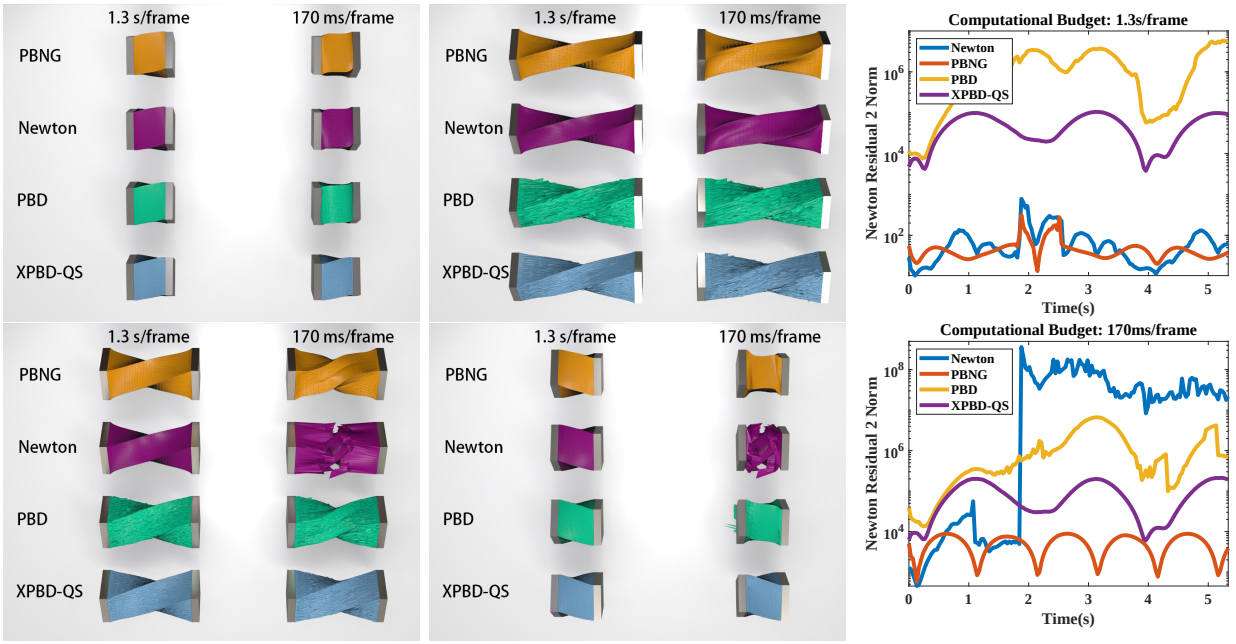


Figure 6.8: **Comparisons with Different Computational Budget**. A block is stretched/compressed while being twisted. With a sufficiently large computational budget, Newton's method is stable, but it becomes unstable when the computational budget is small. PBD and XPBD-QS do not significantly reduce 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.

method, PBD, PBNG and XPBD as described in Section 6.4. In Figure 6.8, these methods are run under a fixed budget. Every method has a runtime of 1.3s/frame. With an ample budget, PBNG converges to ground truth, while PBD and XPBD do not. In Figure 6.8, we show a simulation where every method has a runtime of 170ms/frame. Newton's method is remarkably unstable. PBNG looks visually plausible. PBD and XPBD-QS have visual artifacts and fail to converge. Residual plots vs. time are shown at the bottom of Figure 6.8.

6.8.1.1 Different Resolution

In this example, we demonstrate PBNG's versatility by running the block stretching and twisting with various resolutions. As shown in Figure 6.9, 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.

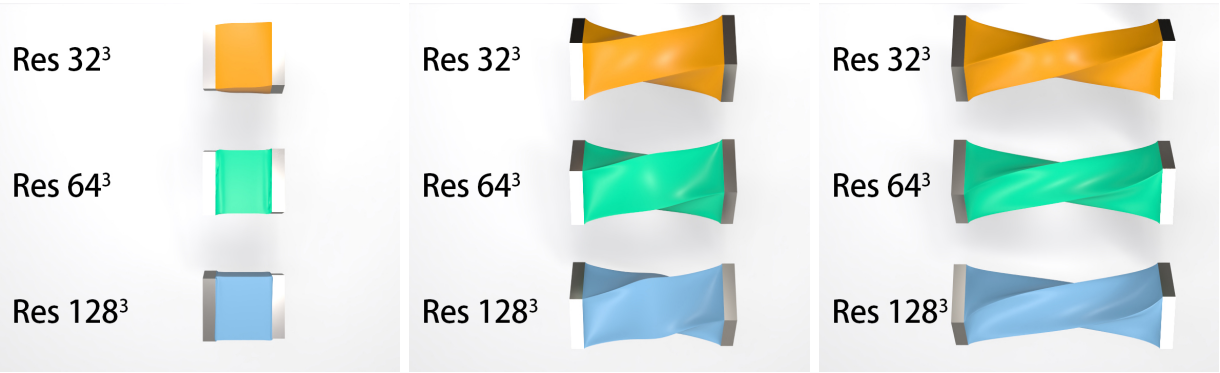


Figure 6.9: **Different Mesh Resolution**. PBNG produces consistent results when the mesh is spatially refined. The highest resolution mesh in this comparison has over 2M vertices and only requires 40 iterations to produce visually plausible results.

6.8.1.2 Different Constitutive Models

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 6.2. The blocks from top to bottom are run with corotated (Equation 6.7), stable Neo-Hookean (Equation 6.10) and Neo-Hookean (Equation 6.8) models respectively. With 40 iterations per frame, they are all visually plausible.

6.8.1.3 Acceleration Comparison

In this example, we compare the effects of the Chebyshev semi-iterative method and the SOR method. In Figure 6.5, we stretch and twist the same block with 32K particles and 150K elements. The top bar is simulated with plain PBNG. The middle bar is simulated with PBNG with Chebyshev semi-iterative method with $\gamma = 1.7, \rho = .95$. The bottom bar is simulated with PBNG with SOR and $\omega = 1.7$. 10 iterations are used for each time step. With a limited budget, plain PBNG is less converged than accelerated techniques. Figure 6.5 shows the convergence rate of the three methods vs. the number of iterations at the first time step. We can see that the acceleration techniques boost the convergence rate.

6.8.2 Collisions

We support collisions by dynamically adding weak constraints as discussed in Section 6.2.1. We use a time step of $\Delta t = .002$ and detect collision every time step.

6.8.2.1 Two Blocks Colliding

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

6.8.2.2 Muscles

We quasistatically simulate a large-scale musculature with collision and connective tissue weak constraints. The mesh has a total of 284K particles and 1097K elements. The muscles have $R^0 = 1000$, Young's modulus $E = 10^5$, connective tissue (blue) weak constraint stiffness

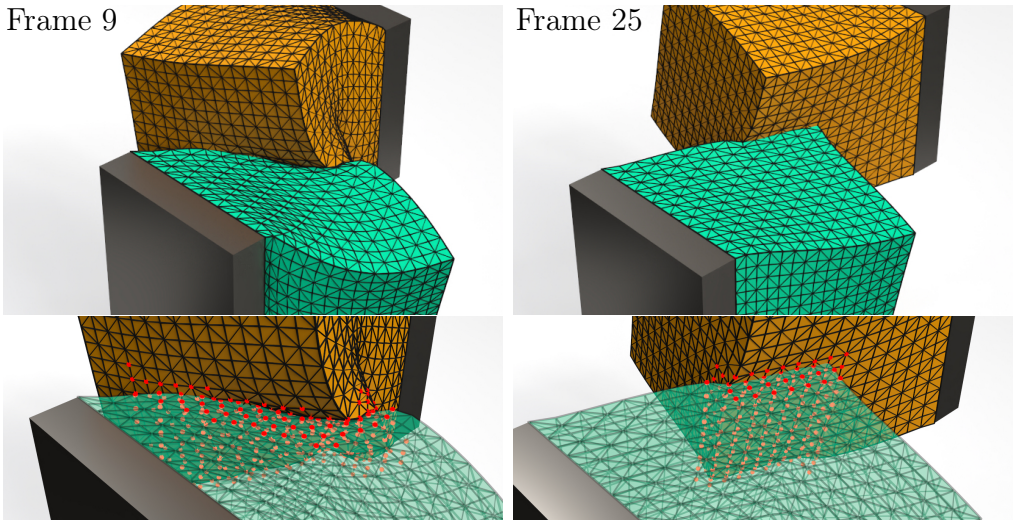


Figure 6.10: **Two Blocks Colliding**. Two blocks collide with each other with one face clamped. Red particles indicate that dynamic weak constraints have been built to resolve the collision of corresponding mesh vertices.

is isotropic $k_n = k_\tau = 10^8$. Dynamic collision (red) weak constraint stiffness is anisotropic $k_n = 10^8$ and $k_\tau = 0$. We show several frames of muscles simulated with PBNG and dynamically generated weak constraints in Figure 6.6. PBNG takes 67 seconds to simulate a frame, while Newton's method takes 430s. In figure 6.1, we show that PBNG looks visually the same as Newton, while running 6-7 times faster. We also show that PBD and XPBD-QS fail to converge. In Figure 6.1, we show PBD becomes unstable. In Figure 6.11, we demonstrate sub-iteration order-dependent behavior with PBD. XPBD-QS has weak constraints processed last, which leads to excessive stretching of elements. XPBD-QS (Flipped) has weak constraints processed first, which degrades their enforcement and leaves a gap.

6.8.2.3 Dropping Objects

40 objects with simple shapes are dropped into a glass box. The objects have a total of 256K particles and 1069K elements. The simulation is run with dynamic/backward Euler. Some frames are shown in Figure 6.12. We show PBNG's capability of handling collision-

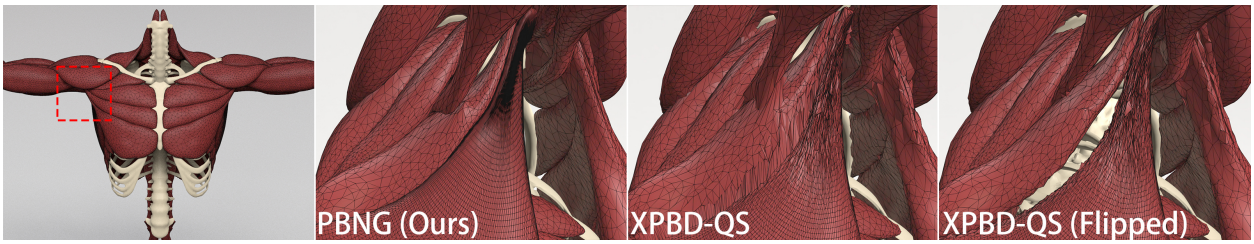


Figure 6.11: **PBNG vs XPBD**. Muscle simulation demonstrates iteration-order-dependent behavior with XPBD and quasistatics. A zoom-in view under the right armpit region is provided. Each method is run 130 iterations. PBNG converges to the desired solution, binding the muscles closely together. XPBD-QS and XPBD-QS (Flipped) fail to converge, leaving either artifacts or gaps between the muscles.

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

6.8.3 Varying Stiffness

In this example, we demonstrate that XPBD-QS fails to resolve quasistatic problems with varied stiffness. In Figure 6.13, we show the initial setup for the simulation. The simulation is quasistatic. Both block meshes have $R^0 = 10$ and Young's modulus $E = 1000$. The first block mesh has its top boundary constrained. The second block is weakly constrained to the first block via weak constraints between them. The springs have stiffness $k_n = k_\tau = 10^8$. There is gravity in the scene with acceleration -9.8 in the $y-$direction. As we show in Figure 6.13, PBNG converges to a plausible state. XPBD-QS and XPBD-QS (Flipped) fail to converge. Depending on the order of the constraints, it either leaves a gap between the two blocks or a very stretched top layer of the bottom block. This example also serves as a simplified version of the connective bindings on the muscles, which are used in Figure 6.11. The residual plot is shown on the right of Figure 6.13.

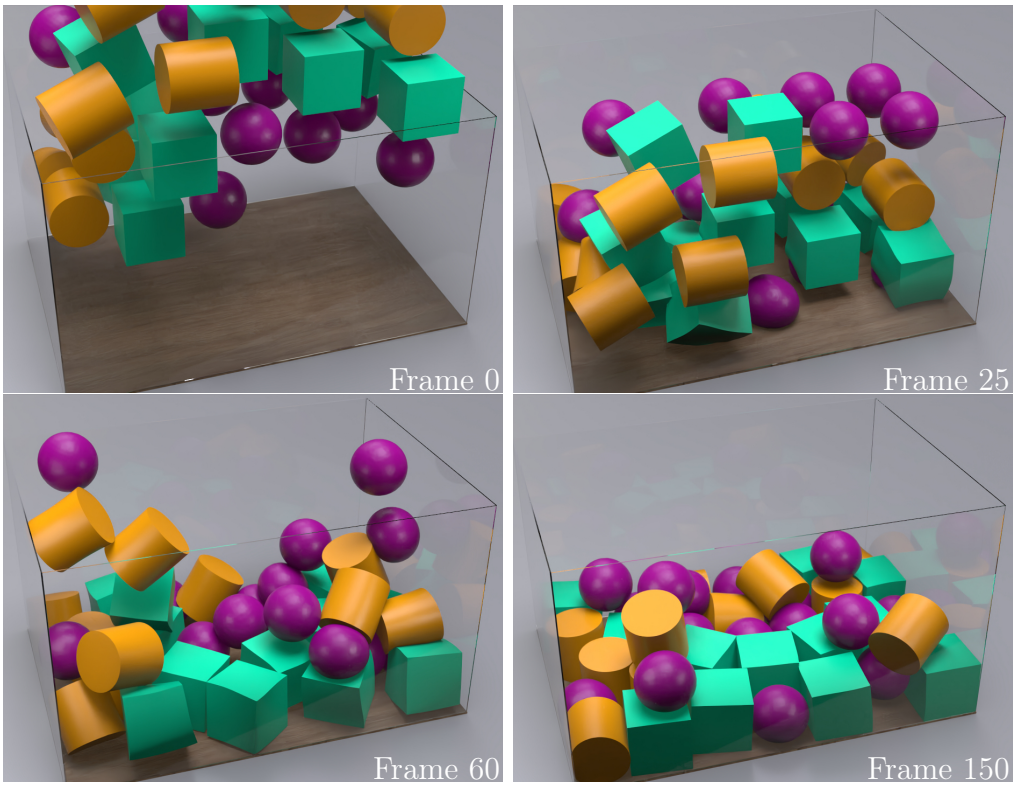


Figure 6.12: **Objects Dropping**. A variety of objects drop under gravity. Our method is able to robustly handle collisions between deformable objects through weak constraints.

6.8.4 PBD

In this example, we show how PBD eliminates the effects of external forcing as the number of iterations increases. We clamp the left side of a simple bar mesh. We run a quasistatic simulation with gravity (acceleration -9.8 in the $y-$direction). The bar has $R^0 = 10$ and Young's modulus $E = 1000$. As shown in Figure 6.3, 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.

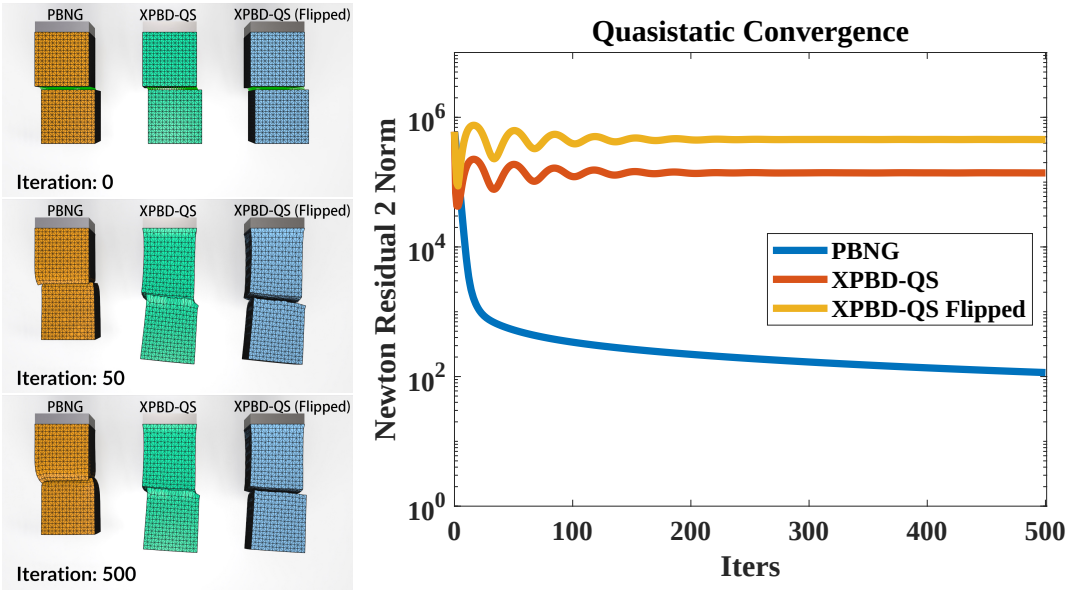


Figure 6.13: **Two Blocks Hanging**. Two identical blocks are bound together through weak constraints. Green line segments in iteration 0 indicate weak constraint springs. PBNG is able to reduce the residual by a few orders of magnitude and converges quickly. XPBD-QS methods demonstrate iteration-order-dependent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

6.8.5 XPBD

We run a simple dynamics example to show that XPBD does not converge numerically, as discussed in Section 6.4.2. We take a simple block with the left side clamped. It falls under gravity and oscillates. The simulation scene is shown on the top of Figure 6.4. The block has 4.1K particles and 17K elements. In this simple simulation, we compare the convergence behavior between PBNG and XPBD. As shown in Figure 6.4(b), XPBD stagnates, while PBNG converges to the tolerance. We demonstrate the reason for XPBD's stagnation in Figure 6.4(a). XPBD omits the primary residual terms, which results in the stagnation of residual reduction. Though XPBD reduces the secondary residual, the true residual stagnates.

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

Table 6.3: 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.

6.8.6 PBNG vs. PBD and Limited Newton

We run a simple quasistatic example to illustrate the convergence propagation behavior of PBNG compared to each conjugate gradient (CG) iteration in Newton's method as well as PBD. In Figure 6.14, a block has its two sides stretched and then clamped. We compute the quasistatic equilibrium using Newton's method with 1 Newton iteration, PBD and PBNG. PBD does not converge to the right solution. After 50 iterations, PBNG looks visually plausible, but Newton's method is visually not converged. The residual plots are presented in Figure 6.14. PBNG iterations are comparable to CG iterations in Newton's method, but they have more favorable deformation propagation behavior.

6.9 Discussion and Limitations

We show that a node-based Gauss-Seidel approach for the nonlinear equations of quasistatic and backward Euler time stepping has remarkably stable behavior. While we generate visually plausible behaviors with restricted computational budgets in a manner that surpasses the PBD and XPBD state-of-the-art for quasistatic problems, our approach (even with Chebyshev and SOR acceleration) will still lose (in terms of numerical residual reduction) to a

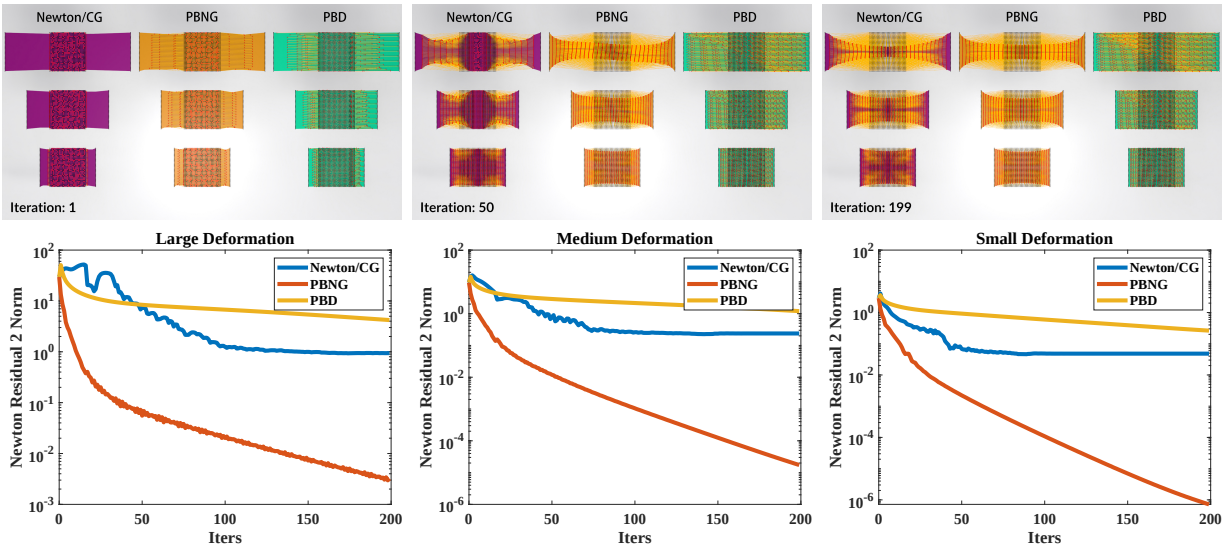


Figure 6.14: **Deformation Propagation Visualization**. A square block is initially stretched on its sides. **Top row**: visual results of the blocks after certain iterations. Black points are the initial positions. Red points are positions at the current iteration. Yellow line segments indicate the displacement of each node. Each method is color coded - purple is Newton, orange is PBNG, and green is PBD. Each row shows the results of large, medium, and small deformations respectively. PBNG converges to a visually plausible result in fewer iterations than one Newton step with increasing CG iterations. PBD fails to shrink in the transverse direction. **Bottom row**: 2-norm of the Newton residual vector. PBNG outperforms Newton's method and PBD.

standard Newton's method when the computational budget is expanded. A multigrid or domain decomposition approach could be combined with our approach to address this in future work.

APPENDIX A

Supplementary Material for Surface Tension

This supplementary material provides detailed derivations of the momentum-conserving remapping algorithm. Section A.1 generalizes the APIC momentum [JSS15] and introduces our notation. Section A.2 proves that assigning the linear and angular velocities of "center of mass" particles to the rest in the particle group (see Figure A.1) conserves linear and angular momentum. Finally, Section A.3 provides the strategy we choose for conservative momentum merging.

A.1 Preliminaries

In order to consider conservation of momentum on the MPM grid, we first need to define mass and velocity on a grid node $\mathbf{i}$. In the APIC perspective, the mass of a grid node $\mathbf{i}$ is

$$m_{\mathbf{i}} = \sum_p m_p w_{\mathbf{i}p}, \tag{A.1}$$

and the momentum of grid node $\mathbf{i}$ is

$$m_{\mathbf{i}}\mathbf{v}_{\mathbf{i}} = \sum_p w_{\mathbf{i}p} m_p (\mathbf{v}_p + \mathbf{A}_p(\mathbf{x}_{\mathbf{i}} - \mathbf{x}_p)), \tag{A.2}$$

where $w_{\mathbf{i}p} = N_{\mathbf{i}}(\mathbf{x}_p)$ is the interpolation weight from particle p to grid node $\mathbf{i}$. We can substitute (A.1) into (A.2) to obtain

$$\left(\sum_p m_p w_{\mathbf{i}p}\right)\mathbf{v}_{\mathbf{i}} = \sum_p w_{\mathbf{i}p} m_p (\mathbf{v}_p + \mathbf{A}_p(\mathbf{x}_{\mathbf{i}} - \mathbf{x}_p)). \tag{A.3}$$

If we denote $m_{\mathbf{i}p} \equiv m_p w_{\mathbf{i}p}$, each particle p's contribution to grid node $\mathbf{i}$'s momentum is

$$m_{\mathbf{i}p}\mathbf{v}_{\mathbf{i}} = m_{\mathbf{i}p} \sum_p \left(\mathbf{v}_p + \mathbf{A}_p(\mathbf{x}_{\mathbf{i}} - \mathbf{x}_p)\right), \tag{A.4}$$

and therefore, velocity at grid node $\mathbf{i}$ can be expressed as

$$\mathbf{v}_{\mathbf{i}} = \sum_p \left(\mathbf{v}_p + \mathbf{A}_p \mathbf{r}_{\mathbf{i}p}\right), \tag{A.5}$$

where $\mathbf{r}_{\mathbf{i}p} \in \mathbb{R}^d$ is the distance from the grid node $\mathbf{i}$ to the center of mass of the all grid nodes associated with particle p. The center of mass location is $\mathbf{x}_p$.

Remark. *The repeated grid index (eg.* $\mathbf{i}$*) and particle index (eg.* p*) does not suggest summing over the grid nodes or particles unless we explicitly write the summation notation.*

Definition 1. *Based on* (A.5)*, we define the generalized particle velocity as*

$$\mathbf{G}_p \equiv [\mathbf{v}_p \mathbf{A}_p]. \tag{A.6}$$

$\mathbf{G}_p$ is a $d \times (d+1)$ matrix, which is a combination of the linear velocity $\mathbf{v}_p$ and the affine velocity $\mathbf{A}_p$, where $d = 2, 3$ is the dimension. Alternatively, $\mathbf{G}_p$ can be written using index notation $G_{\beta\gamma p}$, where $\beta = 1, \ldots, d$ and $\gamma = 0, 1, \ldots, d$. $\mathbf{G}_p$ for particle p can be compacted into a column vector. For 3D scenarios,

$$\mathbf{G}_p = [v_{1p}, v_{2p}, v_{3p}, A_{11p}, A_{12p}, A_{13p}, A_{21p}, A_{22p}, A_{23p}, A_{31p}, A_{32p}, A_{33p}]^T. \tag{A.7}$$

Definition 2. *With the generalized velocity* $G_{\beta\gamma p}$*, we can write the grid velocity as*

$$v_{\mathbf{i}\alpha} = Q_{\mathbf{i}p\alpha\beta\gamma} G_{\beta\gamma p}. \tag{A.8}$$

where

$$Q_{\mathbf{i}p\alpha\beta\gamma} = \begin{cases} \delta_{\alpha\beta}, & \gamma = 0, \\ r_{\mathbf{i}p\gamma}\delta_{\alpha\beta}, & \gamma > 0 \end{cases} \tag{A.9}$$

$Q_{\mathbf{i}p\alpha\beta\gamma}$ can be written in matrix form as

$$\mathbf{Q}_{\mathbf{i}p} = \begin{bmatrix} 1 & 0 & 0 & r_{\mathbf{i}p1} & r_{\mathbf{i}p2} & r_{\mathbf{i}p3} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & r_{\mathbf{i}p1} & r_{\mathbf{i}p2} & r_{\mathbf{i}p3} & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & r_{\mathbf{i}p1} & r_{\mathbf{i}p2} & r_{\mathbf{i}p3} \end{bmatrix}. \tag{A.10}$$

Note that since $Q_{\mathbf{i}p\alpha\beta\gamma}$ only relies on the node $\mathbf{i}$ and the center of mass location, for simplicity, we define

$$Q_{\mathbf{i}\alpha\beta\gamma} \equiv Q_{\mathbf{i}p\alpha\beta\gamma}. \tag{A.11}$$

Then, particle p's contribution to the grid momentum of node $\mathbf{i}$ is

$$p_{\mathbf{i}\alpha p} = m_{\mathbf{i}p} v_{\mathbf{i}\alpha} = m_{\mathbf{i}p} Q_{\mathbf{i}\alpha\beta\gamma} G_{\beta\gamma p}. \tag{A.12}$$

Definition 3. *The generalized moment on the grid due to particle p is*

$$t_{\beta\gamma p} \equiv \sum_{\mathbf{i}} \sum_{\alpha=1}^{d} Q_{\mathbf{i}\alpha\beta\gamma} p_{\mathbf{i}\alpha p}, \tag{A.13}$$

which is a generalization of the linear momentum and the angular momentum about the center of mass.

(A.12) and (A.13) lead to

$$\sum_{\mathbf{i}} \sum_{\alpha=1}^{d} Q_{\mathbf{i}\alpha\beta\gamma} p_{\mathbf{i}\alpha p} = \sum_{\mathbf{i}} \sum_{\alpha=1}^{d} Q_{\mathbf{i}\alpha\beta\gamma} m_{\mathbf{i}p} Q_{\mathbf{i}\alpha\beta\gamma} G_{\beta\gamma p}, \tag{A.14}$$

which is also the equation for the grid-to-particle transfer [JST17].

Definition 4. *The generalized inertia tensor is defined as*

$$\mathbb{I}_{\beta\gamma\delta\epsilon p} \equiv \sum_{\mathbf{i}} \sum_{\alpha=1}^{d} Q_{\mathbf{i}\alpha\beta\gamma} m_{\mathbf{i}p} Q_{\mathbf{i}\alpha\beta\gamma} \tag{A.15}$$

For APIC with quadratic B-spline interpolation, the inertia tensor is diagonal [JSS15]:

$$\mathbb{I}_{\beta\gamma\delta\epsilon p} = m_p \mathbb{D}_{\beta\gamma\delta\epsilon}, \tag{A.16}$$

where $\mathbb{D}_{\beta\gamma\delta\epsilon}$ is constant and diagonal. In 3D,

$$\mathbb{D}_{\beta\gamma\delta\epsilon} = \text{diag}\{1, 1, 1, \frac{1}{4}\Delta x^2, \frac{1}{4}\Delta x^2, \frac{1}{4}\Delta x^2, \frac{1}{4}\Delta x^2, \frac{1}{4}\Delta x^2, \frac{1}{4}\Delta x^2, \frac{1}{4}\Delta x^2, \frac{1}{4}\Delta x^2\}. \quad \text{(A.17)}$$

We can rewrite the right-hand side of the (A.14) as the product of the generalized inertia tensor and the generalized velocity

$$\sum_{\mathbf{i}} \sum_{\alpha=1}^{d} Q_{\mathbf{i}\alpha\beta\gamma} m_{\mathbf{i}p} Q_{\mathbf{i}\alpha\beta\gamma} G_{\beta\gamma p} = \mathbb{I}_{\beta\gamma\delta\epsilon p} G_{\delta\epsilon p}. \quad \text{(A.18)}$$

Remark. *Up to this point, our discussion still falls inside the APIC framework (no resampling involved). Starting from the next section, we introduce the resampled surface particles* $\mathbf{s}_r$ *and balance particles* $\mathbf{b}_r$.

A.2 Conservative Splitting

We now prove our splitting method is conservative. The conservation is both global (total linear and angular momentum conserved over the background grid) and local (linear and angular momentum conserved in each particle group).

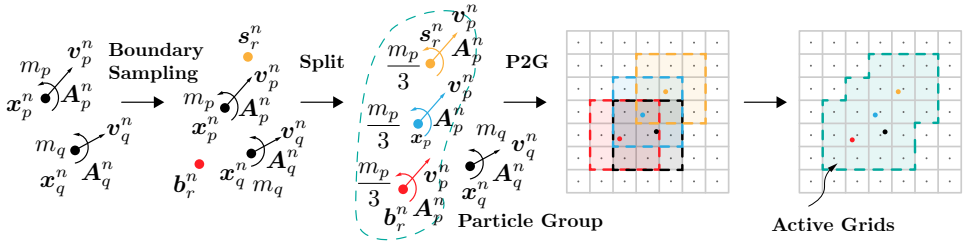


Figure A.1: Splitting algorithm demonstration. The blue particle $\mathbf{x}_p$ is the center of mass of the particle group. Center of mass particles have their linear and angular velocities assigned to the rest of the particles in the group.

Claim 1. *Let* $t_{\beta\gamma}$ *be the total generalized moment before the splitting, and* $\tilde{t}_{\beta\gamma}$ *be the one after the splitting. For each particle* r *in particle* p*'s group* Π_p*, setting* $\mathbf{v}_r = \mathbf{v}_p$ *and* $\mathbf{A}_r = \mathbf{A}_p$

conserves the total linear and angular momentum:

$$t_{\beta\gamma} = \tilde{t}_{\beta\gamma}. \tag{A.19}$$

$\mathbf{v}_p$ *and* $\mathbf{A}_p$ *are the linear and affine velocities of* $\mathbf{x}_p$*, which is the center of mass of the particle group.*

Proof. Before the splitting, only $\mathbf{x}_p$ contributes to the generalized moment (on the grid) about the center of mass.

$$t_{\beta\gamma} = t_{\beta\gamma p} = \sum_{\mathbf{i}} \sum_{\alpha=0}^{d-1} Q_{\mathbf{i}\alpha\beta\gamma} p_{\mathbf{i}\alpha p} \tag{A.20}$$

After the splitting, each particle ($\mathbf{b}_r$, $\mathbf{s}_r$ and $\mathbf{x}_p$) will have its own contribution to the generalized moment. Note that the center of mass location for the particle group is $\mathbf{x}_p$ before and after the splitting, and $Q_{\mathbf{i}\alpha\beta\gamma}$ only depends on grid node $\mathbf{i}$ and the center of mass $\mathbf{x}_p$. Therefore, we can write the new total generalized moment as

$$\tilde{t}_{\beta\gamma} = \tilde{t}_{\beta\gamma p} + \sum_{r\in\Pi_p} \left(\tilde{t}^s_{\beta\gamma r} + \tilde{t}^b_{\beta\gamma r}\right) \tag{A.21}$$

$$= \sum_{\mathbf{i}} \sum_{\alpha=1}^{d} Q_{\mathbf{i}\alpha\beta\gamma} \tilde{p}_{\mathbf{i}\alpha p} + \sum_{r\in\Pi_p} \left(\sum_{\mathbf{i}} \sum_{\alpha=1}^{d} Q_{\mathbf{i}\alpha\beta\gamma} \tilde{p}^s_{\mathbf{i}\alpha r} + \sum_{\mathbf{i}} \sum_{\alpha=1}^{d} Q_{\mathbf{i}\alpha\beta\gamma} \tilde{p}^b_{\mathbf{i}\alpha r} \right) \tag{A.22}$$

$$= \sum_{\mathbf{i}} \sum_{\alpha=1}^{d} Q_{\mathbf{i}\alpha\beta\gamma} \tilde{m}_{\mathbf{i}p} Q_{\mathbf{i}\alpha\delta\epsilon} \tilde{G}_{\delta\epsilon p} +$$
$$\sum_{r\in\Pi_p} \left(\sum_{\mathbf{i}} \sum_{\alpha=1}^{d} Q_{\mathbf{i}\alpha\beta\gamma} \tilde{m}^s_{\mathbf{i}r} Q_{\mathbf{i}\alpha\delta\epsilon} \tilde{G}^s_{\delta\epsilon r} + \sum_{\mathbf{i}} \sum_{\alpha=1}^{d} Q_{\mathbf{i}\alpha\beta\gamma} \tilde{m}^b_{\mathbf{i}r} Q_{\mathbf{i}\alpha\delta\epsilon} \tilde{G}^b_{\delta\epsilon r} \right) \tag{A.23}$$

Since the mass is evenly distributed to each particle,

$$\tilde{m}_p = \tilde{m}^s_r = \tilde{m}^b_r = \frac{m_p}{2|\Pi_p| + 1}. \tag{A.24}$$

If we assign the same linear and affine velocity to all the particles, that is,

$$\tilde{G}_{\beta\gamma p} = \tilde{G}^s_{\beta\gamma r} = \tilde{G}^b_{\beta\gamma r} = G_{\beta\gamma p}, \tag{A.25}$$

then (A.23) can be written as

$$\tilde{t}_{\beta\gamma} = \sum_{\mathbf{i}} \sum_{\alpha=1}^{d} Q_{\mathbf{i}\alpha\beta\gamma} w_{\mathbf{i}p} \tilde{m}_p Q_{\mathbf{i}\alpha\delta\epsilon} \tilde{G}_{\delta\epsilon p} +$$
$$\sum_{r \in \Pi_p} \left(\sum_{\mathbf{i}} \sum_{\alpha=1}^{d} Q_{\mathbf{i}\alpha\beta\gamma} w^s_{\mathbf{i}r} \tilde{m}^s_{\mathbf{i}r} Q_{\mathbf{i}\alpha\delta\epsilon} \tilde{G}^s_{\delta\epsilon r} + \sum_{\mathbf{i}} \sum_{\alpha=1}^{d} Q_{\mathbf{i}\alpha\beta\gamma} w^b_{\mathbf{i}r} \tilde{m}^b_{\mathbf{i}r} Q_{\mathbf{i}\alpha\delta\epsilon} \tilde{G}^b_{\delta\epsilon r} \right) \tag{A.26}$$
$$= \tilde{\mathbb{I}}_{\beta\gamma\delta\epsilon p} \tilde{G}_{\delta\epsilon p} + \sum_{r \in \Pi_p} \left(\tilde{\mathbb{I}}^s_{\beta\gamma\delta\epsilon r} \tilde{G}^s_{\delta\epsilon r} + \tilde{\mathbb{I}}^b_{\beta\gamma\delta\epsilon r} \tilde{G}^b_{\delta\epsilon r} \right) \tag{A.27}$$
$$= \left[\tilde{m}_p + \sum_{r \in \Pi_p} (\tilde{m}^s_r + \tilde{m}^b_r) \right] \mathbb{D}_{\beta\gamma\delta\epsilon} G_{\delta\epsilon p} \tag{A.28}$$

Based on (A.25), we can compute the original mass as

$$m_p = \left[\tilde{m}_p + \sum_{r \in \Pi_p} (\tilde{m}^s_r + \tilde{m}^b_r) \right] \tag{A.29}$$

Substituting (A.29) into (A.28), we recover the generalized moment before the splitting:

$$\tilde{t}_{\beta\gamma} = m_p \mathbb{D}_{\beta\gamma\delta\epsilon} G_{\delta\epsilon p} = t_\beta \gamma. \tag{A.30}$$

Since our way of building surface and balance particles and distributed mass does not alter the center of mass location, this leads to the conclusion that the generalized moment on the grid about the center of mass is conserved during the splitting. In other words, our splitting method conserves linear and angular momentum. □

A.3 Conservative Merging

During the merge process, we collect mass and momentum from the surface particles and balance particles, and we remove these temporary particles. Here, we show a momentum-conserving way to set the linear and affine velocity of $\mathbf{x}_p$.

Claim 2. *There exists a way to set* $\mathbf{v}_p^{n+1}$ *and* $\mathbf{A}_p^{n+1}$ *so that the generalized moment before*

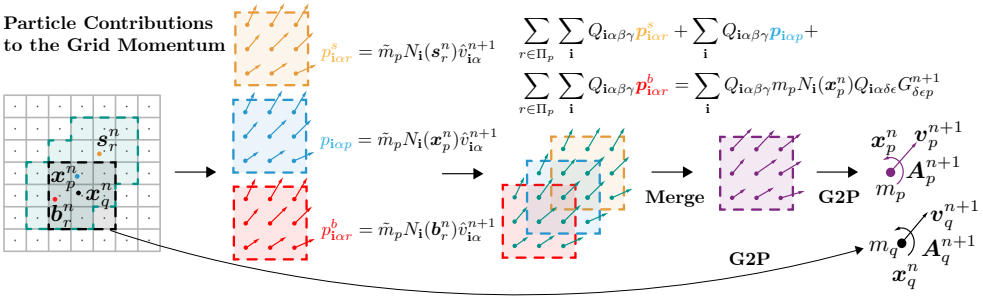


Figure A.2: Merging algorithm demonstration.

the merge $t_{\beta\gamma}$ and the generalized moment after the merge $\hat{t}_{\beta\gamma}$ match:

$$t_{\beta\gamma} = \hat{t}_{\beta\gamma}. \tag{A.31}$$

Proof. Recall that for a single particle group Π_p, the total generalized moment on the grid can be computed by each particle's contributions:

$$t_{\beta\gamma} = t_{\beta\gamma p} + \sum_{r} t_{\beta\gamma r}. \tag{A.32}$$

Before the merge, we have

$$t_{\beta\gamma} = \sum_{\mathbf{i}} \sum_{\alpha=1}^{d} Q_{\mathbf{i}\alpha\beta\gamma} p_{\mathbf{i}\alpha p} + \sum_{r\in\Pi_p} \sum_{\mathbf{i}} \sum_{\alpha=1}^{d} Q_{\mathbf{i}\alpha\beta\gamma} p^s_{\mathbf{i}\alpha r} + \sum_{r\in\Pi_p} \sum_{\mathbf{i}} \sum_{\alpha=1}^{d} Q_{\mathbf{i}\alpha\beta\gamma} p^b_{\mathbf{i}\alpha r}. \tag{A.33}$$

After the merge, only particle $\mathbf{x}_p$ contributes to the generalized moment:

$$\hat{t}_{\beta\gamma} = \sum_{\mathbf{i}} \sum_{\alpha=1}^{d} Q_{\mathbf{i}\alpha\beta\gamma} \hat{m}_{\mathbf{i}p} Q_{\mathbf{i}\alpha\delta\epsilon} \hat{G}_{\delta\epsilon p}. \tag{A.34}$$

In order to have $\hat{t}_{\beta\gamma} = t_{\beta\gamma}$, we need $\hat{G}_{\delta\epsilon p}$ to satisfy the follow equation:

$$\begin{aligned} \sum_{\mathbf{i}} \sum_{\alpha=1}^{d} Q_{\mathbf{i}\alpha\beta\gamma} \hat{m}_{\mathbf{i}p} Q_{\mathbf{i}\alpha\delta\epsilon} \hat{G}_{\delta\epsilon p} = & \sum_{\mathbf{i}} \sum_{\alpha=1}^{d} Q_{\mathbf{i}\alpha\beta\gamma} p_{\mathbf{i}\alpha p} \\ & + \sum_{r\in\Pi_p} \sum_{\mathbf{i}} \sum_{\alpha=1}^{d} Q_{\mathbf{i}\alpha\beta\gamma} p^s_{\mathbf{i}\alpha r} + \sum_{r\in\Pi_p} \sum_{\mathbf{i}} \sum_{\alpha=1}^{d} Q_{\mathbf{i}\alpha\beta\gamma} p^b_{\mathbf{i}\alpha r}. \end{aligned} \tag{A.35}$$

Thus, (A.35) leads to an expression for $\hat{G}_{\delta\epsilon p}$ which conserves the linear and angular momentum during the merging:

$$\hat{G}_{\delta\epsilon p} = (\mathbb{I}_{\beta\gamma\delta\epsilon p})^{-1} \left[\sum_{\mathbf{i}} \sum_{\alpha=1}^{d} Q_{\mathbf{i}\alpha\beta\gamma} p_{\mathbf{i}\alpha p} + \sum_{r \in \Pi_p} \left(\sum_{\mathbf{i}} \sum_{\alpha=1}^{d} Q_{\mathbf{i}\alpha\beta\gamma} p^{s}_{\mathbf{i}\alpha r} + \sum_{\mathbf{i}} \sum_{\alpha=1}^{d} Q_{\mathbf{i}\alpha\beta\gamma} p^{b}_{\mathbf{i}\alpha r} \right) \right]. \tag{A.36}$$

From $\hat{G}_{\delta\epsilon p}$, we can determine $\mathbf{v}_p^{n+1}$ and $\mathbf{A}_p^{n+1}$. Since our derivation of $\hat{G}_{\delta\epsilon p}$ is based on $\hat{t}_{\beta\gamma} = t_{\beta\gamma}$, the merged linear and affine velocity guarantees the conservation of linear and angular momentum. □

APPENDIX B

Supplementary Material for DCDM

B.1 Conjugate Gradients Method

In this appendix, we provide a review of the conjugate gradients (CG) method, which is the inspiration for DCDM as presented in this paper. The conjugate gradients method is a special case of the line search method, where the search directions are **A**-orthogonal to each other. It can also be viewed as a modification of gradient descent (GD) where the search direction is chosen as the component of the residual (equivalently, the negative gradient of the matrix norm of the error) that is **A**-orthogonal to all previous search directions:

$$\mathbf{d}_k = \mathbf{r}_{k-1} - \sum_{i=1}^{k-1} h_{ik}\mathbf{d}_i, \qquad h_{ik} = \frac{\mathbf{d}_i^T \mathbf{A}\mathbf{r}_{k-1}}{\mathbf{d}_i^T \mathbf{A}\mathbf{d}_i}.$$

With this choice, the search directions form a basis for $\mathbb{R}^n$ so that the initial error can be written as $\mathbf{e}_0 = \mathbf{x} - \mathbf{x}_0 = \sum_{i=1}^{n} e_i \mathbf{d}_i$, where e_i are the components of the initial error written in the basis. Furthermore, since the search directions are **A**-orthogonal, the optimal step sizes α_k at each iteration satisfy

$$\begin{aligned}\alpha_k &= \frac{\mathbf{r}_{k-1}^T \mathbf{d}_k}{\mathbf{d}_k^T \mathbf{A}\mathbf{d}_k} = \frac{\mathbf{d}_k^T \mathbf{A}\mathbf{e}_{k-1}}{\mathbf{d}_k^T \mathbf{A}\mathbf{d}_k} \\ &= \frac{\mathbf{d}_k^T \mathbf{A}\left(\sum_{i=1}^{n} e_i \mathbf{d}_i - \sum_{j=1}^{k-1} \alpha_j \mathbf{d}_j\right)}{\mathbf{d}_k^T \mathbf{A}\mathbf{d}_k} = e_k.\end{aligned}$$

That is, the optimal step sizes are chosen to precisely eliminate the components of the error on the basis defined by the search directions. Thus, convergence is determined by the (at most n) non-zero components e_i in the initial error. Although rounding errors prevent this

from happening exactly in practice, this property greatly reduces the number of required iterations [GL12].

Furthermore, $h_{ik} = 0$ for $i < k-1$, and thus iteration can be performed without the need to store all previous search directions $\mathbf{d}_i$ and without the need for computing all previous h_{ik}. To see this, it is sufficient to show $\mathbf{d}_i^T \mathbf{A}\mathbf{r}_{k-1} = 0$.

Lemma In the CG method, residuals are orthogonal, i.e., $\mathbf{r}_k^T \mathbf{r}_j = 0$ for all $j < k$.

Proof

$$\begin{aligned}
\mathbf{r}_k^T \mathbf{r}_j &= (\mathbf{r}_{k-1} - \alpha_k \mathbf{A}\mathbf{d}_k)^T \mathbf{r}_j \\
&= \mathbf{r}_{k-1}^T \mathbf{r}_j - \alpha_k \mathbf{d}_k^T \mathbf{A}\mathbf{r}_j \\
&= \mathbf{r}_{k-1}^T \mathbf{r}_j - \alpha_k \mathbf{d}_k^T \mathbf{A} \left(\mathbf{d}_{j+1} + \sum_{i=1}^{j} h_{i(j+1)} \mathbf{d}_i \right)
\end{aligned}$$

For $j < k-1$, $\mathbf{r}_{k-1}^T \mathbf{r}_j = 0$ follows from induction. $\mathbf{d}_k^T \mathbf{A}(\mathbf{d}_{j+1} + \sum_{i=1}^{j} h_{i(j+1)} \mathbf{d}_i) = \mathbf{d}_k^T \mathbf{A}\mathbf{d}_{j+1} + \sum_{i=1}^{j} h_{i(j+1)} \mathbf{d}_k^T \mathbf{A}\mathbf{d}_i$=0 because $\mathbf{d}_i$ are $\mathbf{A}$-orthogonal by their definition. For $j = k-1$,

$$\begin{aligned}
\mathbf{r}_k^T \mathbf{r}_{k-1} &= \mathbf{r}_{k-1}^T \mathbf{r}_{k-1} - \alpha_k \mathbf{d}_k^T \mathbf{A}\mathbf{r}_{k-1} \\
&= \mathbf{r}_{k-1}^T \mathbf{r}_{k-1} - \frac{\mathbf{r}_{k-1}^T \mathbf{d}_k}{\mathbf{d}_k^T \mathbf{A}\mathbf{d}_k} \mathbf{d}_k^T \mathbf{A}\mathbf{r}_{k-1} \\
&= \mathbf{r}_{k-1}^T \mathbf{r}_{k-1} - \frac{\mathbf{r}_{k-1}^T \mathbf{d}_k}{\mathbf{d}_k^T \mathbf{A}\mathbf{d}_k} \mathbf{d}_k^T \mathbf{A} \left(\mathbf{d}_k + \sum_{i=1}^{k-1} h_i \mathbf{d}_k \right) \\
&= \mathbf{r}_{k-1}^T \mathbf{r}_{k-1} - \mathbf{r}_{k-1}^T \mathbf{d}_k \qquad \text{(by } \mathbf{A}\text{-orthogonality of } \mathbf{d}_k) \\
&= \mathbf{r}_{k-1}^T (\mathbf{r}_{k-1} - \mathbf{d}_k) \\
&= \mathbf{r}_{k-1}^T \left(\sum_{i=1}^{k-1} h_{ik} \mathbf{d}_i \right) \\
&= \sum_{i=1}^{k-1} h_{ik} \mathbf{r}_{k-1}^T \mathbf{d}_i
\end{aligned}$$

So proving $\mathbf{r}_{k-1}^T \mathbf{d}_i = 0$ for $i < k$ would finish the proof. However, by the definition of $\mathbf{d}_i = \mathbf{r}_{i-1} - \sum_{j=1}^{i-1} h_{ij} \mathbf{d}_k$, induction proves $\mathbf{d}_i \in \text{span}(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_{i-1})$. Hence, $\mathbf{r}_{k-1}^T \mathbf{d}_i = 0$ for

all $i \leq k-1$, which proves the lemma.

Claim In the CG method, search directions are **A**-orthogonal to all previous residuals, i.e., $\mathbf{d}_i^T \mathbf{A}\mathbf{r}_{k-1} = 0$ for all $i < k-1$.

Proof $\mathbf{r}_i^T \mathbf{r}_{k-1} = (\mathbf{r}_{i-1}^T - \alpha_i \mathbf{A}\mathbf{d}_i)^T \mathbf{r}_{k-1}$, hence $\mathbf{d}_i \mathbf{A}\mathbf{r}_{k-1} = \mathbf{r}_{i-1}^T \mathbf{r}_{k-1} - \mathbf{r}_i^T \mathbf{r}_{k-1} = 0$ for all $i < k-1$, using the lemma above.

This proves the sparsity of the h_{ik}. As discussed in the main body of the paper, this "memoryless" property of CG is inherited by DCDM and enables the efficiency of our method.

B.2 Choice of α

Line search is an iterative method to find a local minimum of an objective function $h : \mathbb{R}^n \to \mathbb{R}$. In the context of variationally solving $\mathbf{A}\mathbf{x} = \mathbf{b}$, $h(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T \mathbf{A}\mathbf{x} - \mathbf{x}^T \mathbf{b}$, and the k^{th} iterate is computed by

$$\mathbf{x}_k = \mathbf{x}_{k-1} + \alpha_k \mathbf{d}_k.$$

One desires a step size α_k that yields $h(\mathbf{x}_k) < h(\mathbf{x}_{k-1})$. More specifically, the optimal choice is

$$\alpha_k = \arg\min_{\alpha} h(\mathbf{x}_{k-1} + \alpha \mathbf{d}_k) = \frac{\mathbf{r}_{k-1}^T \mathbf{d}_k}{\mathbf{d}_k^T \mathbf{A}\mathbf{d}_k},$$

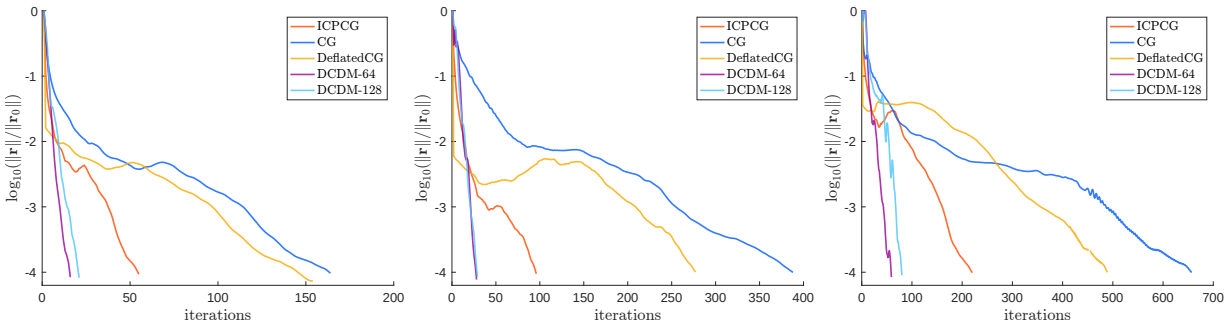
where $\mathbf{r}_{k-1} = \mathbf{b} - \mathbf{A}\mathbf{x}_{k-1}$ is the $(k-1)^{\text{th}}$ residual. To see that this choice of α_k is indeed the minimizer, we can define the objective function as $g(\alpha)$ and write

$$\begin{aligned}
g(\alpha) &= h(\mathbf{x}_{k-1} + \alpha \mathbf{d}_k) \\
&= \frac{1}{2}(\mathbf{x}_{k-1} + \alpha \mathbf{d}_k)^T \mathbf{A}(\mathbf{x}_{k-1} + \alpha \mathbf{d}_k) - \mathbf{b}^T(\mathbf{x}_{k-1} + \alpha \mathbf{d}_k) \\
&= \frac{1}{2}\alpha^2 \mathbf{d}_k^T \mathbf{A}\mathbf{d}_k + \alpha(\mathbf{d}_k^T \mathbf{A}\mathbf{x}_{k-1} - \mathbf{d}_k^T \mathbf{b}) + \left(\frac{1}{2}\mathbf{x}_{k-1}^T \mathbf{A}\mathbf{x}_{k-1} + \mathbf{x}_{k-1}^T \mathbf{b}\right) \\
&= \frac{1}{2}\alpha^2 \mathbf{d}_k^T \mathbf{A}\mathbf{d}_k - \alpha \mathbf{d}_k^T \underbrace{\mathbf{r}_{k-1}}_{\mathbf{b} - \mathbf{A}\mathbf{x}_{k-1}} + (\text{constant terms}).
\end{aligned}$$

Taking the derivative with respect to α, we have $g'(\alpha) = \alpha \mathbf{d}_k^T \mathbf{A}\mathbf{d}_k - \mathbf{d}_k^T \mathbf{r}_{k-1} = 0$, yielding $\alpha = \frac{\mathbf{r}_{k-1}^T \mathbf{d}_k}{\mathbf{d}_k^T \mathbf{A}\mathbf{d}_k}$ as the minimizer of $g(\alpha)$.

B.3 Additional Convergence Results

We include additional convergence results, similar to those shown in Figure 4.4b, in Figure B.1. Specifically, these plots show the convergence of all the methods reported in Table 4.1 at each of the resolutions reported there. The figure visually demonstrates the significant reduction in iteration count achieved by DCDM.



(a) $N = 64$ (b) $N = 128$ (c) $N = 256$

Figure B.1: Convergence of different methods on the 3D bunny example for $N = 64$, 128, 256; summary results, as well as timings, are reported in Table 4.1. DCDM-{64,128} calls a model whose parameters are trained over a $\{64^3, 128^3\}$ grid.

We remark on ICPCG since it is a popular preconditioner and closest in performance to DCDM. When using ICPCG with matrices that arise in a domain with moving internal boundaries (such as our bunny examples), the approximate factorization of $\mathbf{A}$ must be re-computed. Recomputation is also required in the approach of [TSS17] in examples like these. Moreover, as Figure 4.4d shows, DCDM does not require full A-orthogonality. Hence the algorithm only stores two previous vectors, just like CG, and unlike the much more significant memory requirements of ICPCG. For example, the L and D matrices for 128^3 take about 18.7MB in `scipy.sparse` format, while our network can be stored in less than 500KB.

B.4 Ablation Study and Runtime Analysis

Here, we provide results of a small ablation study on network architecture in order to justify some of the architectural choices we made in constructing the DCDM network. We considered a few different models (Figure B.4a to Figure B.4e), several of which are modifications of the model we ultimately used to generate our results (Figure B.4a). The models we considered include one without ResNet connections (Figure B.4b), one with simple downsampling and upsampling (a U-Net-like structure) (Figure B.4c), a minimal CNN (Figure B.4d), and a model with different filter sizes of the blocks (Figure B.4e). We compared how these models perform on the same bunny example considered in the main part of the paper (at resolution 64^3). Figure B.2 shows that the architecture we ultimately selected for DCDM yields the best results.

Method	DCDM	Model 1	Model 2	Model 3	Model 4	U-Net
Number of Parameters	97,457	97,457	97,457	97,457	24,537	3,527,505

Table B.1: Number of parameters for each network architecture considered in the ablation study.

Each model's parameter count is listed in Table B.1. Compared to a basic CNN or U-Net architecture (like the one used in [TSS17]), our DCDM network is actually quite light. For example, the U-Net architecture in [TSS17] uses 3,527,505 parameters (at $N = 64$ in 3D), while our network (at the same resolution) requires only 97,457 parameters (a 36x reduction). In addition, one advantage of our method is that DCDM only needs to be trained once (and data only generated once) per problem class (and possibly size). So if a user desired to solve Poisson systems (which are quite common in computer graphics and engineering), they could use our pre-trained models off the shelf; though we readily concede that new classes of matrices or new resolutions could require new data generation or retraining.

Dataset generation is a key step in using the DCDM model we selected. We found that we

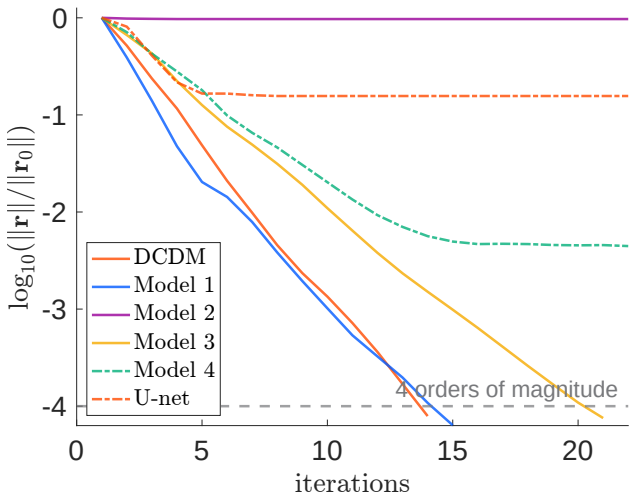
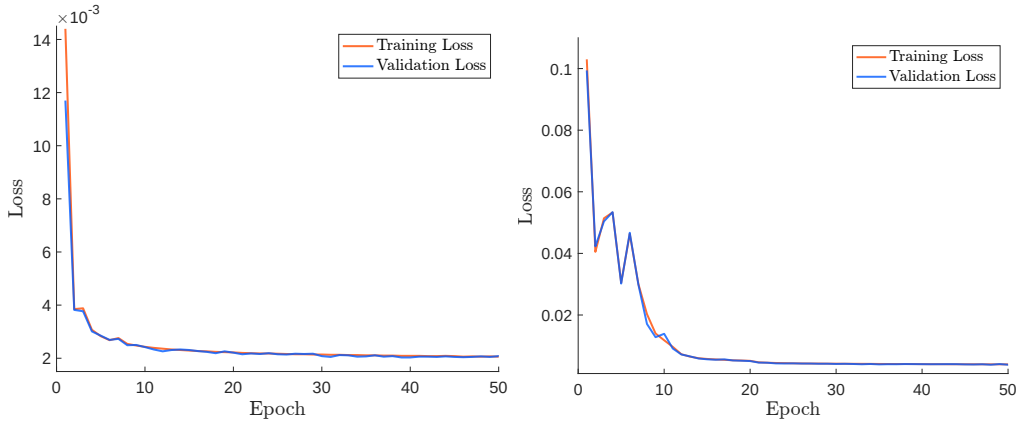


Figure B.2: Residual plot for the bunny example at $N = 64$ with each trained model. The dashed line represents a four-orders-of-magnitude reduction in residual, which is the convergence criterion we use throughout our examples.

needed to include orthogonalization to previous vectors in the Lanczos problem in practice (a well-known limitation of the method). This causes the creation of a dataset (cf. Section 4.3.1) to take $O(n^3m^2)$ time, where m is the number of Lanczos vectors to be created and n is the resolution. Hence increasing resolution from 64^3 to 128^3 increases the time by a factor of 8, which scales 5–7 hours to 2–2.5 days. (However, since we can use low resolution models on higher resolution problems, this scaling can be mitigated, cf. Section 4.3.2.) In addition, the orthogonalization step makes dataset generation have complexity $O(n^3m^2)$, instead of the $O(n^3m)$ complexity of classical Lanczos processes. If we can find any other solution for the numerical problems of classical Lanczos iteration besides orthogonalization, we can drastically reduce the time to generate the dataset (such a task is outside the scope of the present work). We note that storing the training dataset has asymptotic cost $O(kn^3)$; for instance, the dataset of $k = 20{,}000$ synthetic data takes 23GB and 159GB of storage for resolutions 64^3 and 128^3, respectively.

B.5 Model training

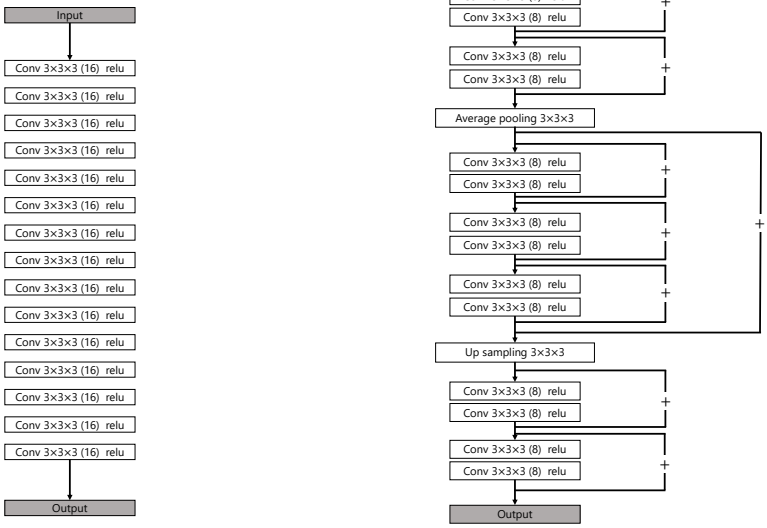
Figure B.3 shows the decrease in training and validation losses observed when training the neural network used for DCDM. As mentioned in Section 4.3.3, for DCDM, we selected the model after epoch 31 for $N = 64$ and epoch 3 for $N = 128$. The plots clearly demonstrate that training and validation loss seem to decrease after these epochs. However, we found that our epoch selections yielded the best performance on our test data, namely, the examples we showed in Section 4.4. Accordingly, we conjecture that our model overfit relatively quickly to both training and validation data, and that perhaps training and validation data were much more similar to each other compared to the test data. We are interested in exploring this further in future work. Of course, philosophically, choosing a model by comparing its performance from different epochs on test data essentially makes that test data part of the validation data, but this is a broader discussion for the learning community.



(a) $N = 64$ (b) $N = 128$

Figure B.3: Training and validation loss for the networks used in DCDM at resolutions $N = 64$ and $N = 128$.

(a) DCDM (our model) (b) Model 1 (c) Model 2



(d) Model 3 (e) Model 4

Figure B.4: Network architectures considered for our ablation study.

APPENDIX C

Supplementary Material for XPBD

C.1 First Piola-Kirchhoff XPBD System

As described in the paper, we want to solve the following system with Newton's method:

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

Let $\mathbf{x}_{i^e k+1 l}^{n+1,e}$ 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 $\mathbf{x}_{i^e k+1 l+1}^{n+1,e} = \mathbf{x}_{i^e k+1 l}^{n+1,e} + \delta \mathbf{x}_{i^e k+1 l}^{e}$. We define the following linearization

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

where we use an affine basis for the change in positions determined by a Newton step:

$$\delta \mathbf{x}_{i^e k+1 l}^{e} = \delta \mathbf{F}_{k+1 l}^{e}(\mathbf{X}_{i^e}^{e} - \mathbf{X}_{\text{com}}^{e}) + \mathbf{b}_{k+1 l}^{e} \tag{C.3}$$

In index notations we have:

$$\delta \mathbf{x}_{i^e \alpha k+1 l}^{e} = \ \delta_{\alpha\epsilon} \delta \mathbf{F}_{\epsilon\tau k+1 l}^{e}(\mathbf{X}_{i^e \tau}^{e} - \mathbf{X}_{\tau\ \text{com}}^{e}) + \mathbf{b}_{\alpha k+1 l}^{e} \tag{C.4}$$

To solve for $\delta \mathbf{x}_{i^e k+1 l}^{e}$, we need to solve the following system.

$$\mathbf{M}^e \delta \mathbf{x}_{i^e k+1 l}^{e} + \Delta t^2 \mathbf{B}^e \frac{\partial^2 \Psi}{\partial \mathbf{F}^2}(\mathbf{F}^e(\mathbf{x}_{k+1}^{n+1,e})) \delta \mathbf{F}_{k+1 l}^{e} = -\mathbf{g}_{k+1 l} \tag{C.5}$$

where $\mathbf{g}^e_{k+1l} = \mathbf{M}^e(\mathbf{x}^{n+1,e}_{k+1l} - \tilde{\mathbf{x}}^e) + \Delta t^2 \mathbf{B}^e \frac{\partial \Psi}{\partial \mathbf{F}}(\mathbf{F}^e(\mathbf{x}^{n+1,e}_{k+1l})) - \mathbf{f}^e$.

Plug Equation C.3 into Equation C.5 we have:

$$\sum_{i^e \in \Omega^e} \left(M_{i^e \alpha j^e \beta} \delta x^e_{j^e \beta k+1l} + \Delta t^2 B^e_{i^e \alpha \gamma \delta} \frac{\partial^2 \Psi}{\partial F_{\gamma\delta} \partial F_{\sigma\nu}} (\mathbf{F}^e(\mathbf{x}^{n+1,e}_{k+1l})) \delta F^e_{\sigma\nu k+1l} \right) = \tag{C.6}$$

$$\sum_{i^e \in \Omega^e} m_{i^e j^e} \delta_{\alpha\beta} \delta x^e_{j^e \beta k+1l} + \Delta t^2 \sum_{i^e \in \Omega^e} B^e_{i^e \alpha \gamma \delta} \frac{\partial^2 \Psi}{\partial F_{\gamma\delta} \partial F_{\sigma\nu k+1l}} (\mathbf{F}^e(\mathbf{x}^{n+1,e}_{k+1l})) \delta F^e_{\sigma\nu k+1l} = \tag{C.7}$$

$$\sum_{i^e \in \Omega^e} m_{i^e j^e} \delta x^e_{j^e \alpha k+1l} = \tag{C.8}$$

$$- \sum_{i^e \in \Omega^e} g_{i^e \alpha k+1l}. \tag{C.9}$$

Using Equation (C.3), $M_{i^e \alpha j^e \beta} = \delta_{\alpha\beta} m_{i^e j^e}$ where $m_{i^e j^e}$ is diagonal and $m_e = \sum_{i^e \in \Omega^e} m_{i^e i^e}$

$$\sum_{i^e \in \Omega^e} m_{i^e j^e} \delta x^e_{j^e \alpha k+1l} = \sum_{i^e \in \Omega^e} m_{i^e j^e} D^e_{i^e \alpha \epsilon\tau} \delta F^e_{\epsilon\tau k+1l} + m_e b^e_{\alpha k+1l} = \tag{C.10}$$

$$(m_e X^e_{\tau \mathrm{com}} - m_e X^e_{\tau \mathrm{com}}) \delta F^e_{\epsilon\tau k+1l} + m_e b^e_{\alpha k+1l} = m_e b^e_{\alpha k+1l}. \tag{C.11}$$

Therefore

$$\mathbf{b}^e_{k+1l} = \frac{-1}{m_e} \sum_{i^e \in \Omega^e} \mathbf{g}_{i^e k+1l} \tag{C.12}$$

Furthermore, multiplying Equation (C.5) by the transpose of $\mathbf{D}$ yields

$$D^e_{i^e \alpha \eta\nu} \left(M_{i^e \alpha j^e \beta} \delta x^e_{j^e \beta k+1l} + \Delta t^2 B^e_{i\alpha\gamma\delta} \frac{\partial^2 \Psi}{\partial F_{\gamma\delta} \partial F_{\sigma\nu}} (\mathbf{F}^e(\mathbf{x}^{n+1}_{(k)})) \delta F^e_{\sigma\nu k+1l} \right) = \tag{C.13}$$

$$D^e_{i^e \alpha \eta\nu} m_{i^e j^e} \delta x^e_{j^e \beta k+1l} + \Delta t^2 \frac{\partial^2 \Psi}{\partial F_{\eta\nu} \partial F_{\sigma\nu}} (\mathbf{F}^e(\mathbf{x}^{n+1}_{(k)})) \delta F^e_{\sigma\nu k+1l} \tag{C.14}$$

$$D^e_{i^e \alpha \eta\nu} m_{i^e j^e} (D^e_{i^e \alpha \epsilon\tau} \delta F^e_{\epsilon\tau k+1l} + b^e_{\alpha k+1l}) + \Delta t^2 \frac{\partial^2 \Psi}{\partial F_{\eta\nu} \partial F_{\sigma\nu}} (\mathbf{F}^e(\mathbf{x}^{n+1}_{(k)})) \delta F^e_{\sigma\nu k+1l} \tag{C.15}$$

where we use the fact that $\mathbf{D}^T\mathbf{B} = V^e\mathbf{I}$, which is shown below:

$$(\mathbf{D}^T\mathbf{B})_{\epsilon\tau\beta\nu} = D^e_{i^e\alpha\epsilon\tau}B^e_{i^e\alpha\beta\nu} \tag{C.16}$$

$$= \sum_{i^e,\alpha}\delta_{\alpha\beta}\delta_{\alpha\epsilon}(X^e_{i^e\tau} - X^e_{\tau\text{com}})\frac{\partial N_{i^e}}{\partial X_\nu}(\mathbf{X}^e)V^e \tag{C.17}$$

$$= \delta_{\epsilon\beta}V^e\sum_{i^e}(X^e_{i^e\tau} - X^e_{\tau\text{com}})\frac{\partial N_{i^e}}{\partial X_\nu}(\mathbf{X}^e) \tag{C.18}$$

$$= \delta_{\epsilon\beta}V^e(\sum_{i^e}\frac{\partial N_{i^e}}{\partial X_\nu}(\mathbf{X}^e)X^e_{i^e\tau} - \sum_{i^e}\frac{\partial N_{i^e}}{\partial X_\nu}(\mathbf{X}^e)X^e_{\tau\text{com}}) \tag{C.19}$$

$$= \delta_{\epsilon\beta}V^e(\sum_{i^e}\frac{\partial N_{i^e}}{\partial X_\nu}(\mathbf{X}^e)X^e_{i^e\tau} - (\sum_{i^e}\frac{\partial N_{i^e}}{\partial X_\nu}(\mathbf{X}^e))X^e_{\tau\text{com}}) \tag{C.20}$$

$$= \delta_{\epsilon\beta}V^e(\sum_{i^e}\frac{\partial N_{i^e}}{\partial X_\nu}(\mathbf{X}^e)X^e_{i^e\tau} - \frac{\partial(\sum_{i^e}N_{i^e})}{\partial X_\nu}(\mathbf{X}^e)X^e_{\tau\text{com}}) \tag{C.21}$$

$$= \delta_{\epsilon\beta}V^e(\sum_{i^e}\frac{\partial N_{i^e}}{\partial X_\nu}(\mathbf{X}^e)X^e_{i^e\tau} - \frac{\partial 1}{\partial X_\nu}(\mathbf{X}^e)X^e_{\tau\text{com}}) \tag{C.22}$$

$$= \delta_{\epsilon\beta}V^e\sum_{i^e}\frac{\partial N_{i^e}}{\partial X_\nu}(\mathbf{X}^e)\mathbf{X}^e_{i^e\tau} \tag{C.23}$$

$$= V^e\delta_{\epsilon\beta}\delta_{\tau\nu} \tag{C.24}$$

where we use the fact N_{i^e} is a partition of unity and reproduction of linear function properties.

C.1.1 Mass Term Computation

We compute $\tilde{\mathbf{M}}^e$:

$$\tilde{M}^e_{\eta\nu\epsilon\tau} = \sum_{i^e,j^e\in\Omega^e}D^e_{i^e\alpha\eta\nu}m_{i^ej^e}D^e_{j^e\alpha\epsilon\tau} \tag{C.25}$$

$$= \sum_{\alpha,i^e,j^e,\beta}\delta_{\alpha\eta}(X^e_{i^e\nu} - X^e_{\nu\text{com}}))\delta_{i^ej^e}\delta_{\alpha\beta}\delta_{\alpha\epsilon}(X^e_{j^e\tau} - X^e_{\tau\text{com}})) \tag{C.26}$$

$$= \delta_{\eta\epsilon}\sum_{i^e}m_{i^e}(X^e_{i^e\nu} - X^e_{\nu\text{com}})(X^e_{i^e\tau} - X^e_{\tau\text{com}}) \tag{C.27}$$

Let $\hat{M}^e_{\nu\tau} = \sum_{i^e} m_{i^e}(X^e_{i^e\nu} - X^e_{\nu\text{com}})(X^e_{i^e\tau} - X^e_{\tau\text{com}})$ then $\tilde{\mathbf{M}}^e = \begin{pmatrix} \hat{\mathbf{M}}^e & & \\ & \hat{\mathbf{M}}^e & \\ & & \hat{\mathbf{M}}^e \end{pmatrix}$

C.1.2 Quasi-Newton

In the paper we made an approximation of the Hessian term. The true hessian is:

$$\frac{\partial^2 \Psi}{\partial \mathbf{F}^2}(\mathbf{F}^e) = 2\mu\mathbf{I} + 2\mu\frac{\partial \mathbf{R}(\mathbf{F})}{\partial \mathbf{F}} + \lambda\frac{\partial \det(\mathbf{F}^e)}{\partial \mathbf{F}^e} \otimes \frac{\partial \det(\mathbf{F}^e)}{\partial \mathbf{F}^e} + \lambda \det(\mathbf{F}^e)\frac{\partial^2 \det(\mathbf{F}^e)}{\partial (\mathbf{F}^e)^2} \tag{C.28}$$

$$\approx 2\mu\mathbf{I} + \lambda\frac{\partial \det(\mathbf{F}^e)}{\partial \mathbf{F}^e} \otimes \frac{\partial \det(\mathbf{F}^e)}{\partial \mathbf{F}^e} \tag{C.29}$$

The term we omit is $2\mu\frac{\partial \mathbf{R}(\mathbf{F})}{\partial \mathbf{F}} + \lambda \det(\mathbf{F}^e)\frac{\partial^2 \det(\mathbf{F}^e)}{\partial (\mathbf{F}^e)^2}$. This approximation makes it easier to compute the inverse of the matrix. Also it ensures that the matrix is positive definite, so that the whole Newton solve becomes more stable.

C.1.3 Corotated Fiber Term

Here is a more detailed description of the temrs for corotated fiber model:

$$f_p = \begin{cases} 0.0076\lambda_{ofl} e^{6.6(\frac{l^e}{\lambda_{ofl}} - 1)} - 0.05(l^e - \lambda_{ofl}) & l^e > \lambda_{ofl} \\ 0 & \text{otherwise} \end{cases} \tag{C.30}$$

$$f_a = \begin{cases} 0 & l^e \leq 0.4\lambda_{ofl} \\ 3\lambda_{ofl}(\frac{l^e}{\lambda_{ofl}} - 0.4)^3 & 0.6\lambda_{ofl} > l^e > 0.4\lambda_{ofl} \\ -0.6612\lambda_{ofl} + l^e - 1.33\lambda_{ofl}(\frac{l^e}{\lambda_{ofl}} - 1)^3 & 1.4\lambda_{ofl} \geq l^e \geq 0.6\lambda_{ofl} \\ 0.6774\lambda_{ofl} + 3\lambda_{ofl}(\frac{l^e}{\lambda_{ofl}} - 1.6)^3 & 1.6\lambda_{ofl} \geq l^e \geq 1.4\lambda_{ofl} \\ 0.6774\lambda_{ofl} & l^e > 1.6\lambda_{ofl} \end{cases} \tag{C.31}$$

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

C.2 Parallel Gauss-Seidel

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. 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 C.1.

We also offer a coloring scheme for the grid-based variation. 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 C.2. Since grid nodes incident to an element change every timestep, the elements have to be recolored every timestep. We can speed up the coloring process by using the coloring results from previous timestep as an initial guess.

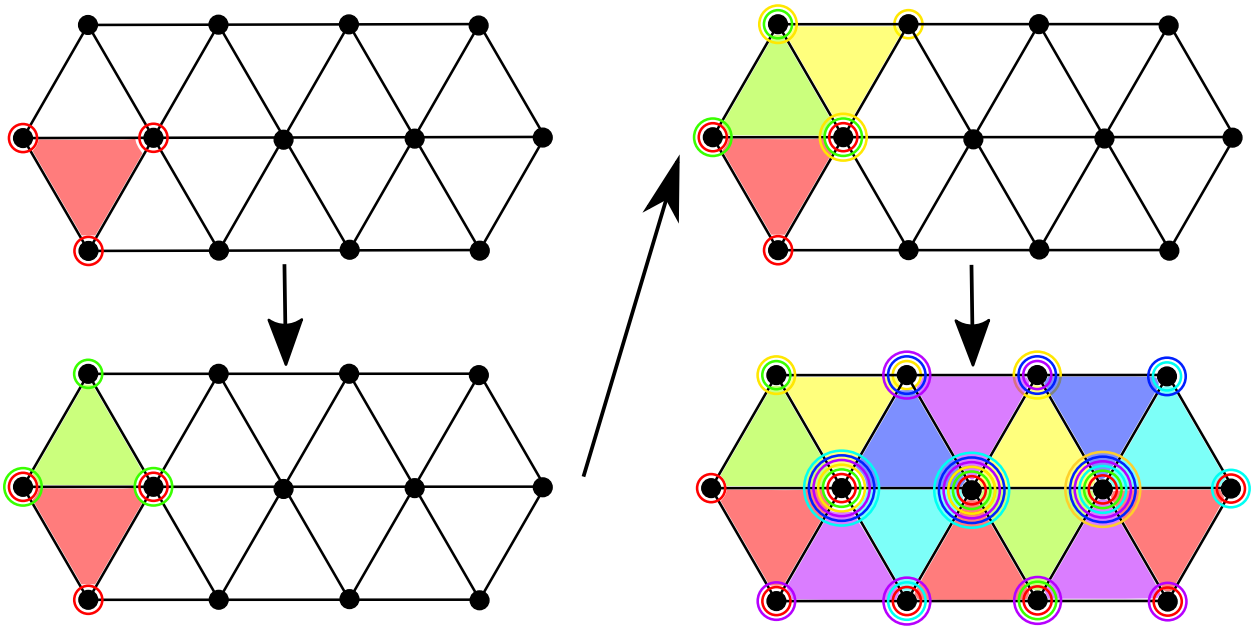


Figure C.1: **Tetrahedron Mesh Coloring**. A step-by-step tetrahedron mesh coloring scheme is shown. After the first element is colored red, all its incident points will register red as used color. Other elements incident to the first element have to choose other colors.

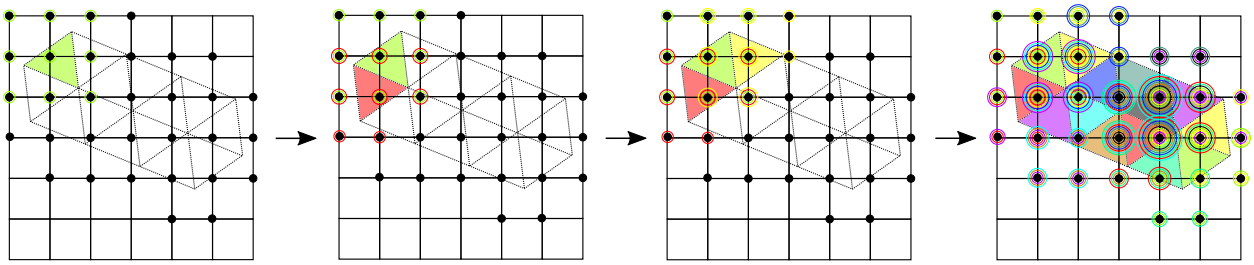


Figure C.2: **Grid-Based Mesh Coloring**. A step-by-step grid-based tetrahedron mesh coloring scheme for 2D is shown. The grid uses linear interpolation here, so each particle on the mesh is interpolated by the 4 grid nodes on the cell containing it. An element can have at maximum 12 incident grid nodes. After the first element is colored green, 9 grid nodes that are incident will register green as a used color, so that other elements incident to those nodes won't use it.

APPENDIX D

Supplementary Material for Nonlinear Gauss-Seidel

D.1 Linear Elasticity

D.1.1 Potential

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

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

D.1.2 First-Piola-Kirchhoff Stress

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

D.1.3 Hessian

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

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

D.1.4 General Isotropic Elasticity Modified Hessian

We use the modified Hessian

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

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.

D.2 Neo-Hookean

D.2.1 Neo-Hookean Potential

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

D.2.2 First-Piola-Kirchhoff Stress

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

D.2.3 Hessian

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

D.2.3.1 Determinant Hessian

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

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

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

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

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

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

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	

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

[AAB15] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems.", 2015. Software available from tensorflow.org.

[ADP20] J. Ackmann, P. D. Düben, T. N. Palmer, and P. K. Smolarkiewicz. "Machine-learned preconditioners for linear solvers in geophysical fluid flows." *arXiv preprint arXiv:2010.02866*, 2020.

[BLM13] T. Belytschko, W. Liu, B. Moran, and K. Elkhodary. *Nonlinear finite elements for continua and structures.* John Wiley and sons, 2013.

[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.

[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.

[BR86] J. Brackbill and H. Ruppel. "FLIP: A method for adaptively zoned, Particle-In-Cell calculations of fluid flows in two dimensions." *J Comp Phys*, **65**:314–343, 1986.

[Bri08] R. Bridson. *Fluid simulation for computer graphics.* Taylor & Francis, 2008.

[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.

[CCS12] M. Corsini, P. Cignoni, and R. Scopigno. "Efficient and Flexible Sampling with Blue Noise Properties of Triangular Meshes." *IEEE Trans Vis Comp Graph*, **18**(6):914–924, 2012.

[CHC23a] Yizhou Chen, Yushan Han, Jingyu Chen, Shiqian Ma, Ronald Fedkiw, and Joseph Teran. "Primal Extended Position Based Dynamics for Hyperelasticity." In *Proceedings of the 16th ACM SIGGRAPH Conference on Motion, Interaction and*

Games, MIG '23, New York, NY, USA, 2023. Association for Computing Machinery.

[CHC23b] Yizhou Chen, Yushan Han, Jingyu Chen, and Joseph Teran. "Position-Based Nonlinear Gauss-Seidel for Quasistatic Hyperelasticity.", 2023.

[Che95] E. Chernyaev. "Marching cubes 33: Construction of topologically correct isosurfaces." Technical report, 1995.

[Cho67] A. Chorin. "A numerical method for solving incompressible viscous flow problems." *J Comp Phys*, **2**(1):12–26, 1967.

[CKM21] J. Chen, V. Kala, A. Marquez-Razon, E. Gueidon, D. A. B. Hyde, and J. Teran. "A Momentum-Conserving Implicit Material Point Method for Surface Tension with Contact Angles and Spatial Gradients." *ACM Trans. Graph.*, **40**(4), jul 2021.

[CKS17] C. R. A. Chaitanya, A. S. Kaplanyan, C. Schied, M. Salvi, A. Lefohn, D. Nowrouzezahrai, and T. Aila. "Interactive reconstruction of Monte Carlo image sequences using a recurrent denoising autoencoder." *ACM Trans Graph*, **36**(4):1–12, 2017.

[CMM90] Alexandre Joel Chorin, Jerrold E Marsden, and Jerrold E Marsden. *A mathematical introduction to fluid mechanics*, volume 3. Springer, 1990.

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

[CWS13] P. Clausen, M. Wicke, J. R. Shewchuk, and J. F. O'Brien. "Simulating liquids and solid-liquid interactions with Lagrangian meshes." *ACM Transactions on Graphics (TOG)*, **32**(2):17, 2013.

[DBW15] F. Da, C. Batty, C. Wojtan, and E. Grinspun. "Double bubbles sans toil and trouble: discrete circulation-preserving vortex sheets for soap films and foams." *ACM Trans Graph (SIGGRAPH 2015)*, 2015.

[DHB16] F. Da, D. Hahn, C. Batty, C. Wojtan, and E. Grinspun. "Surface-only liquids." *ACM Trans Graph (TOG)*, **35**(4):1–12, 2016.

[DHW19] M. Ding, X. Han, S. Wang, T. Gast, and J. Teran. "A thermomechanical material point method for baking and cooking." *ACM Trans Graph*, **38**(6):192, 2019.

[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.

[FGG17] C. Fu, Q. Guo, T. Gast, C. Jiang, and J. Teran. "A Polynomial Particle-in-cell Method." *ACM Trans Graph*, **36**(6):222:1–222:12, November 2017.

[FLP14] Y. Fan, J. Litven, and D. Pai. "Active Volumetric Musculoskeletal Systems." *ACM Trans Graph*, **33**(4):152:1–152:9, 2014.

[flu22] fluidnetsc22. "fluidnetsc22/fluidnet_sc22: v0.0.1.", April 2022. doi: 10.5281/zenodo.6424901, URL: `https://doi.org/10.5281/zenodo.6424901`.

[FPF04] R. Farahi, A. Passian, T. Ferrell, and T. Thundat. "Microfluidic manipulation via Marangoni forces." *Applied Phys Let*, **85**(18):4237–4239, 2004.

[FPS19] Joel H Ferziger, Milovan Perić, and Robert L Street. *Computational methods for fluid dynamics*. springer, 2019.

[FSJ01] R. Fedkiw, J. Stam, and H. Jensen. "Visual simulation of smoke." In *SIGGRAPH*, pp. 15–22. ACM, 2001.

[FSK06] M. M. Francois, J. M. Sicilian, and D. B. Kothe. "Modeling of thermocapillary forces within a volume tracking algorithm." In *Modeling of Casting, Welding and Advanced Solidification Processes–XI*, pp. 935–942, 2006.

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

[GA18] M. Götz and H. Anzt. "Machine learning-aided numerical linear algebra: Convolutional neural networks for the efficient preconditioner generation." In *2018 IEEE/ACM 9th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (scalA)*, pp. 49–56. IEEE, 2018.

[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.

[GGB19] D. Greenfeld, M. Galun, R. Basri, I. Yavneh, and R. Kimmel. "Learning to optimize multigrid PDE solvers." In *Int Conf Mach Learn*, pp. 2415–2423. PMLR, 2019.

[GHB20] C. Gissler, A. Henne, S. Band, A. Peer, and M. Teschner. "An implicit compressible SPH solver for snow simulation." *ACM Trans Graph (TOG)*, **39**(4):36–1, 2020.

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

[GHM20] S. Gagniere, D. Hyde, A. Marquez-Razon, C. Jiang, Z. Ge, X. Han, Q. Guo, and J. Teran. "A Hybrid Lagrangian/Eulerian Collocated Velocity Advection and Projection Method for Fluid Simulation." *Computer Graphics Forum*, **39**(8):1–14, 2020.

[GIF18] I. Georgiev, T. Ize, M. Farnsworth, R. Montoya-Vozmediano, A. King, B. Van Lommel, A. Jimenez, O. Anson, S. Ogaki, E. Johnston, A. Herubel, D. Russell, F. Servant, and M. Fajardo. "Arnold: A brute-force production path tracer." *ACM Transactions on Graphics (TOG)*, **37**(3):1–12, 2018.

[GL12] G. Golub and C. Van Loan. *Matrix computations*, volume 3. JHU Press, 2012.

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

[GSK16] A. Grebhahn, N. Siegmund, H. Köstler, and S. Apel. "Performance prediction of multigrid-solver configurations." In *Software for Exascale Computing-SPPEXA 2013-2015*, pp. 69–88. Springer, 2016.

[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.

[Har64] F. Harlow. "The particle-in-cell method for numerical solution of problems in fluid dynamics." *Meth Comp Phys*, **3**:319–343, 1964.

[HGM20] D.A.B. Hyde, S.W. Gagniere, A. Marquez-Razon, and J. Teran. "An Implicit Updated Lagrangian Formulation for Liquids with Large Surface Energy." *ACM Trans Graph*, **39**(6), November 2020.

[HIK20] W. Huang, J. Iseringhausen, T. Kneiphof, Z. Qu, C. Jiang, and M.B. Hullin. "Chemomechanical Simulation of Soap Film Flow on Spherical Bubbles." *ACM Trans Graph*, **39**(4), July 2020.

[HS52] M. R. Hestenes and E. Stiefel. "Methods of Conjugate Gradients for Solving Linear Systems." *Journal of research of the National Bureau of Standards*, **49**(6):409, 1952.

[HSN18] M. Hopp-Hirschler, M. S. Shadloo, and U. Nieken. "A Smoothed Particle Hydrodynamics approach for thermo-capillary flows." *Comp Fluids*, **176**:1 – 19, 2018.

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

[HW65] F. Harlow and E. Welch. "Numerical Calculation of Time Dependent Viscous Flow of Fluid with a Free Surface." *Phys Fluid*, **8**(12):2182–2189, 1965.

[HZR16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep Residual Learning for Image Recognition." In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[IFH20] T. Ichimura, K. Fujita, M. Hori, L. Maddegedara, N. Ueda, and Y. Kikuchi. "A Fast Scalable Iterative Implicit Solver with Green's function-based Neural Networks." In *2020 IEEE/ACM 11th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA)*, pp. 61–68, 2020.

[ITF04] G. Irving, J. Teran, and R. Fedkiw. "Invertible Finite Elements for Robust Simulation of Large Deformation." In *Proc ACM SIGGRAPH/Eurograph Symp Comp Anim*, pp. 131–140, 2004.

[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.

[JST17] C. Jiang, C. Schroeder, and J. Teran. "An angular momentum conserving affine-particle-in-cell method." *J Comp Phys*, **338**:137 – 164, 2017.

[KAC23] A. Kaneda, O. Akar, J. Chen, V.A.T. Kala, D. Hyde, and J. Teran. "A Deep Conjugate Direction Method for Iteratively Solving Linear Systems." In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 15720–15736. PMLR, 23–29 Jul 2023.

[KB15] D. P. Kingma and J. Ba. "Adam: A Method for Stochastic Optimization." *CoRR*, **abs/1412.6980**, 2015.

[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.

[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.

[Lan50] C. Lanczos. "An iteration method for the solution of the eigenvalue problem of linear differential and integral operators." 1950.

[Lan02] D. Langbein. *Capillary surfaces: shape – stability – dynamics, in particular under weightlessness*, volume 178. Springer Science & Business Media, 2002.

[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.

[LGF04] F. Losasso, F. Gibou, and R. Fedkiw. "Simulating water and smoke with an octree data structure." *ACM Trans. Graph.*, **23**(3):457–462, 2004.

[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.

[LGM20] I. Luz, M. Galun, H. Maron, R. Basri, and I. Yavneh. "Learning algebraic multigrid using graph neural networks." In *Int Conf Mach Learn*, pp. 6489–6499. PMLR, 2020.

[LKB21] K. Luna, K. Klymko, and J. P. Blaschke. "Accelerating GMRES with Deep Learning in Real-Time.", 2021.

[LLD20] W. Li, D. Liu, M. Desbrun, J. Huang, and X. Liu. "Kinetic-based Multiphase Flow Simulation." *IEEE Trans Vis Comp Graph*, 2020.

[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.

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

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

[MH94] Jerrold E Marsden and Thomas JR Hughes. *Mathematical foundations of elasticity*. Courier Corporation, 1994.

[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.

[MM21] M. Macklin and M. Muller. "A Constraint-based Formulation of Stable Neo-Hookean Materials." In *Motion, Interaction and Games*, p. 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.

[Mon92] J. Monaghan. "Smoothed particle hydrodynamics." *Ann Rev Astron Astroph*, **30**(1):543–574, 1992.

[MST10] A. McAdams, E. Sifakis, and J. Teran. "A Parallel Multigrid Poisson Solver for Fluids Simulation on Large Grids." In *Proc 2010 ACM SIGGRAPH/Eurograph Symp Comp Anim*, pp. 65–74. Eurographics Association, 2010.

[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.

[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.

[OFC02] R. Osada, T. Funkhouser, B. Chazelle, and D. Dobkin. "Shape Distributions." *ACM Trans. Graph.*, **21**(4):807–832, October 2002.

[Pai71] C. C. Paige. *The computation of eigenvalues and eigenvectors of very large sparse matrices*. PhD thesis, University of London, 1971.

[PGG23] J. Panuelos, R. Goldade, E. Grinspun, D.I.W. Levin, and C. Batty. "PolyStokes: A Polynomial Model Reduction Method for Viscous Fluid Simulation." *ACM Trans Graph (TOG)*, **42**(4), 2023.

[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.

[RGJ15] D. Ram, T. Gast, C. Jiang, C. Schroeder, A. Stomakhin, J. Teran, and P. Kavehpour. "A material point method for viscoelastic fluids, foams and sponges." In *Proc ACM SIGGRAPH/Eurograph Symp Comp Anim*, pp. 157–163, 2015.

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

[RTM01] R. Rioboo, C. Tropea, and M. Marengo. "Outcomes from a Drop Impact on Solid Surfaces." *Atomization and Sprays*, **11**(2), 2001.

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

[Saa03] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, USA, 2nd edition, 2003.

[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.

[SCS94] D. Sulsky, Z. Chen, and H. Schreyer. "A particle method for history-dependent materials." *Comp Meth App Mech Eng*, **118**(1):179–196, 1994.

[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.

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

[SKB08] M. Steffen, R. Kirby, and M. Berzins. "Analysis and reduction of quadrature errors in the material point method (MPM)." *Int J Numer Meth Eng*, **76**(6):922–948, 2008.

[SS60] L. Scriven and C. Sternling. "The marangoni effects." *Nature*, **187**(4733):186–188, 1960.

[SSC13] A. Stomakhin, C. Schroeder, L. Chai, J. Teran, and A. Selle. "A Material Point Method for snow simulation." *ACM Trans Graph*, **32**(4):102:1–102:10, 2013.

[SSH19] J. Sappl, L. Seiler, M. Harders, and W. Rauch. "Deep Learning of Preconditioners for Conjugate Gradient Solvers in Urban Water Related Problems.", 2019.

[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.

[Sta99] J. Stam. "Stable Fluids." In *Siggraph*, volume 99, pp. 121–128, 1999.

[Sta20] R. Stanaityte. *ILU and Machine Learning Based Preconditioning For The Discretized Incompressible Navier-Stokes Equations*. PhD thesis, University of Houston, 2020.

[Sti52] E. Stiefel. "Über einige methoden der relaxationsrechnung." *Zeitschrift für angewandte Mathematik und Physik ZAMP*, **3**(1):1–33, 1952.

[SYE00] Y. Saad, M. Yeung, J. Erhel, and F. Guyomarc'h. "A Deflated Version of the Conjugate Gradient Algorithm." *SIAM Journal on Scientific Computing*, **21**:1909–1926, 2000.

[Tho55] J. Thomson. "XLII. On certain curious motions observable at the surfaces of wine and other alcoholic liquors." *London, Edinburgh, and Dublin Phil Mag J Sci*, **10**(67):330–333, 1855.

[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.

[TSS17] J. Tompson, K. Schlachter, P. Sprechmann, and K. Perlin. "Accelerating Eulerian fluid simulation with convolutional networks." In D. Precup and Y. Teh, editors, *Proc 34th Int Conf Mach Learn*, volume 70 of *Proc Mach Learn Res*, pp. 3424–3433. PMLR, 06–11 Aug 2017.

[TWG10] N. Thürey, C. Wojtan, M. Gross, and G. Turk. "A multiscale approach to mesh-based surface tension flows." *ACM Trans Graph (TOG)*, **29**(4):1–10, 2010.

[UBF20] K. Um, R. Brand, Y. Fei, P. Holl, and N. Thuerey. "Solver-in-the-Loop: Learning from Differentiable Physics to Interact with Iterative PDE-Solvers." In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pp. 6111–6122. Curran Associates, Inc., 2020.

[VGO20] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas,

Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python." *Nature Methods*, **17**:261–272, 2020.

[VS15] D. C. Venerus and D. N. Simavilla. "Tears of wine: New insights on an old phenomenon." *Scientific reports*, **5**:16162, 2015.

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

[WTG10] C. Wojtan, N. Thürey, M. Gross, and G. Turk. "Physics-inspired topology changes for thin fluid features." *ACM Trans Graph*, **29**(4):50:1–50:8, 2010.

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

[YHW16] S. Yang, X. He, H. Wang, S. Li, G. Wang, E. Wu, and K. Zhou. "Enriching SPH simulation by approximate capillary waves." In *Symp Comp Anim*, pp. 29–36, 2016.

[You05] T. Young. "III. An essay on the cohesion of fluids." *Phil Trans Royal Soc London*, **95**:65–87, 1805.

[YYJ23] Y.Chen, Y.Han, J.Chen, S.Ma, R.Fedkiw, and J.Teran. *Supplementary Technical Document*, 2023.

[YYX16] C. Yang, X. Yang, and X. Xiao. "Data-driven projection method in fluid simulation." *Comp Anim Virt Worlds*, **27**(3-4):415–424, 2016.

[ZB05] Y. Zhu and R. Bridson. "Animating sand as a fluid." *ACM Trans Graph*, **24**(3):965–972, 2005.

[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.

[ZQC14] B. Zhu, E. Quigley, M. Cong, J. Solomon, and R. Fedkiw. "Codimensional surface tension flow on simplicial complexes." *ACM Trans Graph (TOG)*, **33**(4):1–11, 2014.

[ZZK15] W. Zheng, B. Zhu, B. Kim, and R. Fedkiw. "A new incompressibility discretization for a hybrid particle MAC grid representation with surface tension." *J Comp Phys*, **280**:96–142, 2015.